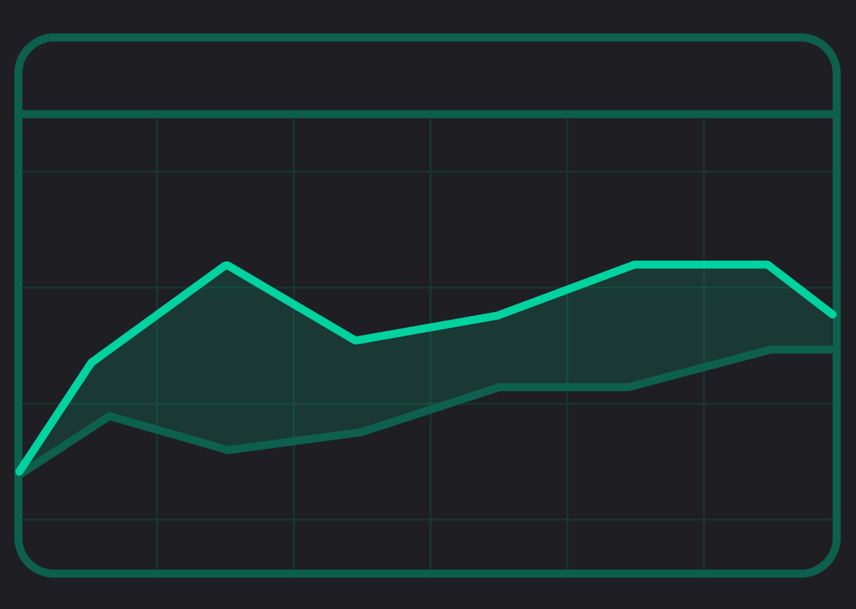
AD.QM



Arenadata QuickMarts

Arenadata QuickMarts (ADQM) is a cluster column-oriented DBMS built on ClickHouse. ADQM generates various online analytical reports based on vast amounts of information stored in flat marts. ADQM is much faster than traditional DBMS.

Version CURRENT Language: EN

Contents

1. Concepts
    - ADQM specific features
    - ADQM architecture
    - Sharding
    - Typical cluster
    - Geo-distributed cluster
    - Proxying

2. Planning guide
    - Hardware requirements
    - Network requirements
    - Software requirements

3. Get started
    - Installation
        - Online installation
            - Install ADCM
            - Prepare hosts
            - Install ADQM cluster
                - Create a cluster
                - Add services
                - Add hosts to a cluster
                - Add components
                - Configure services
                - Configure a cluster
                - Install a cluster
            - Install monitoring
        - Offline installation
            - Install ADCM
            - Prepare hosts
            - Install Enterprise Tools cluster
                - Create a cluster
                - Add services
                - Add hosts to a cluster
                - Add components
                - Install a cluster
            - Install ADQM cluster
                - Create a cluster
                - Add services
                - Add hosts to a cluster
                - Add components
                - Configure services
                - Configure a cluster
                - Import ET settings
                - Install a cluster
            - Install monitoring
    - Connect to ADQM
        - ClickHouse web interface
        - ClickHouse command-line client
        - JDBC driver
        - Third-party client tools
            - DBeaver

4. Access management
    - Authentication
        - Password
        - LDAP
        - SSL certificate
        - Kerberos
            - MIT Kerberos
            - MS Active Directory
                - Encryption type issue on RedOS
            - FreeIPA
    - Authorization
    - Encryption of database connections

5. Database maintenance
    - Cluster management via ADCM
        - Cluster actions
        - Service actions
        - Confgure logical clusters
        - Rolling upgrade
        - Maintenance mode
    - Coordination systems
        - ZooKeeper
        - ClickHouse Keeper
    - clickhouse-copier
    - Configure data storages
    - System tables
    - Settings
        - Configure user settings
        - Named collections
    - Monitoring metrics
    - Logging

6. Data management
    - Overview of database objects
        - Databases
        - Tables
            - Table engines
            - Tables overview
            - Row update and deletion
            - Data compression
            - TTL
        - Indexes
        - Dictionaries
        - Materialized views
        - Projections
        - User-defined functions and Executable tables
    - Data querying
        - Example of working with tables via clickhouse-client
        - Query types
            - JOIN
            - Subqueries
            - Combine queries
            - Common table expressions
        - Use functions
            - Aggregate functions
                - flameGraph
            - Window functions
            - Table functions
        - Work with complex data types
            - JSON
            - Nested data structures
        - Full-text search
        - Query optimization
            - Improve query performance
            - Analyze queries
            - Sampling query profiler
        - Kill a query
    - Encryption codecs
    - Machine learning

7. Integration with external systems
    - JDBC Bridge
    - ADH
        - HDFS
        - Hive
    - ADS
        - Kafka
    - ADPG
    - S3
    - Import and export to files

8. References
    - Configuration parameters

9. Release notes
    - ADQM releases
    - Known issues

ADQM specific features

Elena Dvoryadkina

To Table of Contents

Arenadata QuickMarts (ADQM) is a database management system for online analytical processing (OLAP) of structured big data. It can be used in various fields to address the tasks requiring high-speed processing of non-aggregated data that is also constantly added in real-time, for example:

- web and mobile application analytics;
- analytics of metrics in real time;
- quick data marts;
- monitoring and analysis of structured logs and events;
- real-time data monitoring and analysis.

ADQM is built on the ClickHouse open-source DBMS that allows generating analytical data reports in real time. Specific features of ADQM/ClickHouse:

- **Column-oriented data storage** — each column data is stored independently. This allows reading data quickly (searching for data values in required columns only, not in the entire table) and compressing data effectively (since each column contains data of the same type).
- **Primary indexes** — ADQM/ClickHouse keeps data physically sorted by primary key. This makes it possible to quickly retrieve data based on key values or value ranges.
- **Vector calculations** — data processing by vectors (parts of columns) allows achieving high CPU efficiency.
- **Parallel query processing** — large queries are parallelized across multiple cores and use all the necessary resources available on the current server.
- **Distributed query processing on multiple servers** of a cluster due to the sharding mechanism.
- **Sampling** (executing a query based on a part of data) and **approximate calculations** provide acceptable trade-off between accuracy and performance.
- **Linear scalability** — you can add new nodes to a cluster to extend it so that it can store and process petabytes of data.
- **Working on hard drives** — ADQM/ClickHouse can work on regular hard drives, unlike other column-oriented DBMSs that can work in RAM only. This allows you to optimize data storage cost as hard drives are cheaper than RAM. SSD and additional RAM can also be used if available.
- **Asynchronous multi-master replication** — after data has been written to any available replica, all the remaining replicas get their copy in the background. As the system maintains full data identity on different replicas, it can automatically recover data after most failures.
- **SQL query syntax** — ADQM/ClickHouse supports an SQL language that is similar to ANSI SQL but includes some extensions: arrays and nested data structures, probabilistic structures, the availability to connect an external key/value store, etc.
- Support for multiple **database engines** and **table engines** — the Database Engines and Table Engines articles of the ClickHouse documentation list and describe all database and table engines that ADQM/ClickHouse can use.
- **Real-time data inserts** — the MergeTree table engine allows ADQM/ClickHouse to add data to tables continually in real time (without locks when inserting new data).
- **Integration with external systems** — ADQM/ClickHouse can be integrated with different external data sources such as Kafka, RabbitMQ, Hadoop (HDFS), MySQL, PostgreSQL, MongoDB, etc.

ADQM/ClickHouse also has some limitations:

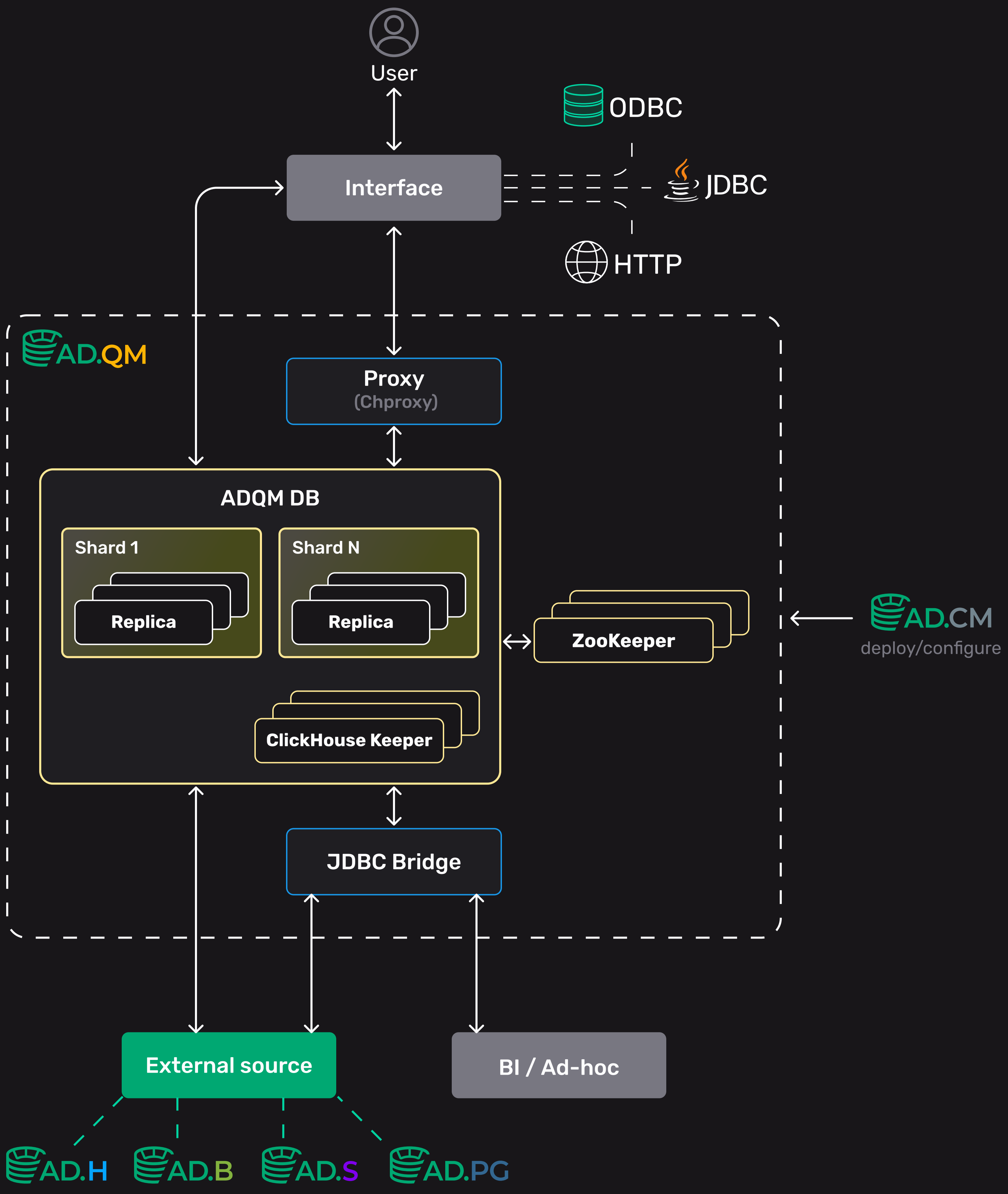
- Transactions are not supported.
- Updating or deleting previously inserted data is not fast enough.
- Low speed of point queries retrieving specific rows by their keys due to a sparse index.

ADQM architecture

Elena Dvoryadkina

To Table of Contents

The high-level architecture view of ADQM is shown below.



ADQM architecture

Main points of the scheme:

- To access ADQM, users can use JDBC and ODBC drivers, HTTP interface, console client, as well as wrappers on Python, PHP, Node.js, Perl, Ruby, Go, etc. Additional information about interfaces for working with ClickHouse is available in the Drivers and Interfaces section of the ClickHouse documentation.
- ADQM cluster includes the following services:
  - **ADQMDB** — ClickHouse column-oriented database management system. Data can be stored across different shards, where each shard is a set of replicas. Replication works at the table level on each shard independently of other shards.
  - **ZooKeeper**, **ClickHouse Keeper** — services that are responsible for coordinating data replication and are not involved in processing and executing queries.
  - **Chproxy** — an HTTP proxy and load balancer for the ADQMDB/ClickHouse database.

  Users can manage an ADQM cluster using Arenadata Cluster Manager (ADCM) — a universal hybrid landscape orchestrator that allows installing, configuring, and upgrading Arenadata data services via the graphical interface.
- ADQM can connect to an external data source (directly or via JDBC Bridge) and access its data. For example, ADQM can be integrated with the following systems: MySQL, MongoDB, Hadoop (HDFS), Apache Kafka, PostgreSQL, etc. The Table Engines for Integrations section lists all supported integrations.

  For connecting to Arenadata products — Arenadata Database (ADB), Arenadata Hadoop (ADH), Arenadata Streaming (ADS), Arenadata Postgres (ADPG) — native integration is available within Arenadata Enterprise Data Platform (EDP).
- For visualizing data processed by ADQM and generating analytical reports, any BI tools providing access via JDBC/ODBC can be used. For example, BusinessObjects, Power BI, Luxms, Visiology, etc.

Sharding

Elena Dvoryadkina

Contents

To Table of Contents

- Distributed tables
  - Create a distributed table
  - Cluster configuration
- Write data to a cluster
  - Write data directly to shards
  - Write data to a distributed table
- Read data from a distributed table
- Resharding

Sharding is a database design principle that suggests locating parts of the same table on different shards. **Shard** is a cluster node that can consist of one or more replicas. **Replicas** are servers that duplicate data within a shard. `SELECT` and `INSERT` queries can be sent to any replica of a shard, there is no a dedicated master.

Sharding can be useful in the following cases:

- The amount of data to be stored and the frequency of queries to a database increase.
- A system requires more resources, but a cluster cannot be scaled using the vertical strategy (which involves updating the hardware of existing servers) due to physical limits — in terms of the number of cores per processor, the number of processors, the amount of memory, etc.

Sharding allows you to:

- **Overcome technical limitations.** If a large data set does not fit on a single server, or the current storage infrastructure is running at hardware performance limits, you can scale a cluster horizontally — add new servers to the existing set and distribute data across multiple shards.
- **Improve fault tolerance.** Sharding helps mitigate the impact of failures. If data is stored on a single server, the failure of that server can lead to loss of access to all data. When data is distributed (for example, across five shards), the failure of one shard leaves 80% of data available.
- **Increase the speed of executing queries.** Queries compete with each other for the computing resources of cluster servers. This may slow processing queries. In a sharded cluster where queries to the same table can be executed in parallel, competition for shared resources is eliminated and query processing time is reduced.

Distributed tables

For sharding, use tables on the Distributed engine. A **distributed table** does not store data. Data is stored in local tables placed on servers of each shard. A distributed table only provides routing of queries to these tables, in other words, it allows processing distributed queries on multiple servers.

Create a distributed table

A distributed table can be created anywhere, even on servers that are not included into shards that store tables with data. However, a standard approach is to place a distributed table on all shards that host tables with data.

An example of a query that creates a distributed table on all servers of the cluster:

```
CREATE TABLE distributed_table ON CLUSTER cluster_name AS table_name
ENGINE = Distributed(cluster_name, database, table_name [, sharding_key][, policy_name]);
```

The Distributed table engine accepts the following parameters:

- cluster name in a server's configuration file;
- name of a database for a table that the distributed table will access (the database name should be the same on all servers of the cluster);
- name of a table that the distributed table will access (the table name should be the same on all servers of the cluster);
- optionally — sharding key used to decide how to distribute data across shards when executing `INSERT` queries;
- optionally — policy name for storing temporary files for asynchronous uploads.

Cluster configuration

The Distributed table engine requires a cluster to be defined in each server's configuration file (*config.xml*). A cluster defined in the configuration file is a logical entity that groups servers. One server can participate in several logical clusters. This provides the flexibility to distribute data between servers.

A cluster can be configured as follows:

```
<remote_servers>
    <cluster_name>
        <shard>
            <weight>2</weight>
            <internal_replication>false</internal_replication>
            <replica>
                <priority>1</priority>
                <host>host_01_1</host>
                <port>9000</port>
            </replica>
            <replica>
                <host>host_01_2</host>
                <port>9000</port>
            </replica>
        </shard>
        <shard>
            <weight>1</weight>
            <internal_replication>false</internal_replication>
            <replica>
                <host>host_02_1</host>
                <port>9000</port>
            </replica>
            <replica>
                <host>host_02_2</host>
                <port>9000</port>
            </replica>
        </shard>
    </cluster_name>
</remote_servers>
```

In this example, a distributed cluster (`cluster_name`) consists of two shards. Each shard contains two replicas. Each replica parameters are `host` (remote server address) and `port` (TCP port for inter-server communication, usually `9000`).

A cluster can include one shard (in this case, query processing is called remote, not distributed) or multiple shards. In each shard, you can specify one or any number of replicas. The number of replicas for each shard within the cluster can be different.

You can specify the following optional parameters for a shard:

- `weight` (default is `1`) — shard weight used when writing data. Data is distributed across shards in the amount proportional to shard weights. For example, there are two shards — the first shard's weight is `2`, the second one's weight is `1`. In this case, two thirds (`2/3`) of data rows to be inserted will be sent to the first shard, and one third (`1/3`) of rows will be sent to the second shard.
- `internal_replication` — defines how to insert data to replicas:
  - `true` — data is written to the first available replica. Use this parameter value when inserting data to a replicated table that manages data replication itself. Data written to a replica will be copied to other replicas of the shard in the asynchronous mode.
  - `false` (default) — data is written to all replicas. In this case, the distributed table replicates data itself. This is less preferred than using replicated tables as the identity of replicas is not controlled (they may contain slightly different data over time).

You can also configure the load balancing algorithm for replicas (specify preferences on which of the replicas to send a request first). To do this, you can specify the `priority` parameter of replicas in cluster configuration (less value has more priority, a default value is `1`), or use the load_balancing setting.

Configure a cluster via ADCM

You can configure an ADQM logical cluster in the ADCM interface using parameters in the ***Cluster configuration*** section on the ADQMDB service's configuration page. To do this, you can:

- Set the ***Replication factor*** parameter value before installing ADQM — all ADQM hosts will be automatically combined into a logical cluster based on the specified replication factor.
- Manually add and configure required clusters using visual elements of the ***Cluster Configuration*** setting.

After installing ADQM or restarting the ADQMDB service, the cluster definition will be automatically added to the configuration file. For more details, see Configure logical clusters in the ADCM interface.

Write data to a cluster

There are two ways to write data to a cluster.

Write data directly to shards

You can manage data distribution across servers yourself (determine shards to insert data manually) and write data directly to each shard. In other words, you can send `INSERT` queries directly to local tables that are underlying for the distributed table. This is the most flexible and optimal solution as you can use a sharding scheme meeting the domain requirements and write data to different shards completely independently.

Write data to a distributed table

You can send `INSERT` queries to a distributed table. In this case, the table will distribute data to be inserted across servers itself. To do this, you need to set a **sharding key** when creating the distributed table (except if there is only one shard).

A sharding key can be specified as any expression that returns an integer. For example, it can be:

- the `UserID` table column — to distribute data by user ID (data of one user will be located on one shard);
- the toDayOfWeek function — to distribute data across shards depending on the day of the week of a certain date (for example, if a table has the `saleDate` field containing sale dates, and a cluster consists of 7 shards);
- the `rand()` expression — to distribute data across shards randomly.

To select a shard to which a data row should be sent, the Distributed engine uses the remainder of dividing the sharding key by the total weight of shards. It sends a data row to the shard if the remainder of dividing is in the range between `prev_weights` and `prev_weights + weight`, where `prev_weights` is the total weight of shards with a lower number, and `weight` is the weight of this shard. For example, there are 2 shards — the first shard's weight is `2`, and the second one's weight is `1`. The total weight of these shards is `2 + 1 = 3`. In this case, possible remainders from dividing the sharding key by the total weight of shards are `0`, `1`, and `2`. A data row will be written to the first shard if the remainder is `0` or `1` (a value from the `[0, 2)` range), and to the second shard if the remainder is `2` (a value from the `[2, 3)` range).

A simple remainder of dividing is not always appropriate for sharding. It is suitable for medium to large data sets (dozens of servers), but not for very large data sets (hundreds of servers or more). In the latter case, it is better to use a sharding scheme meeting the domain requirements rather than using the ability to write to distributed tables.

Read data from a distributed table

Unlike `INSERT` queries, `SELECT` queries are sent to all shards of a cluster regardless of how data is distributed across shards (the optimize_skip_unused_shards setting manages this behavior). For reading data on each shard, one of the available replicas is selected. A read request is processed on remote servers and partially aggregated results are sent to a distributed table. It then combines all received data and send the full result to a user.

Resharding

Auto resharding is not supported, but you can use the clickhouse-copier functionality (provided by the adqm-clickhouse-copier package that comes with the ADQMDB service) to redistribute data. This tool allows you to copy data from tables in one cluster (source cluster) to tables in another cluster (destination cluster). To use it, do the following:

1. Create a configuration file for tasks to copy tables. In this file, define:
   - a source cluster and a destination cluster (in the `remote_servers` section);
   - tasks for sharding (in the `tables` section);
   - a new sharding key (in the `sharding_key` section).
2. Create a ZooKeeper configuration file.
3. Create a task in ZooKeeper.
4. Run clickhouse-copier to copy data.
5. Create a new distributed table, switch from the old cluster to the new one.

For more details, refer to the Usage of clickhouse-copier article.

Typical cluster

Elena Dvoryadkina

Contents

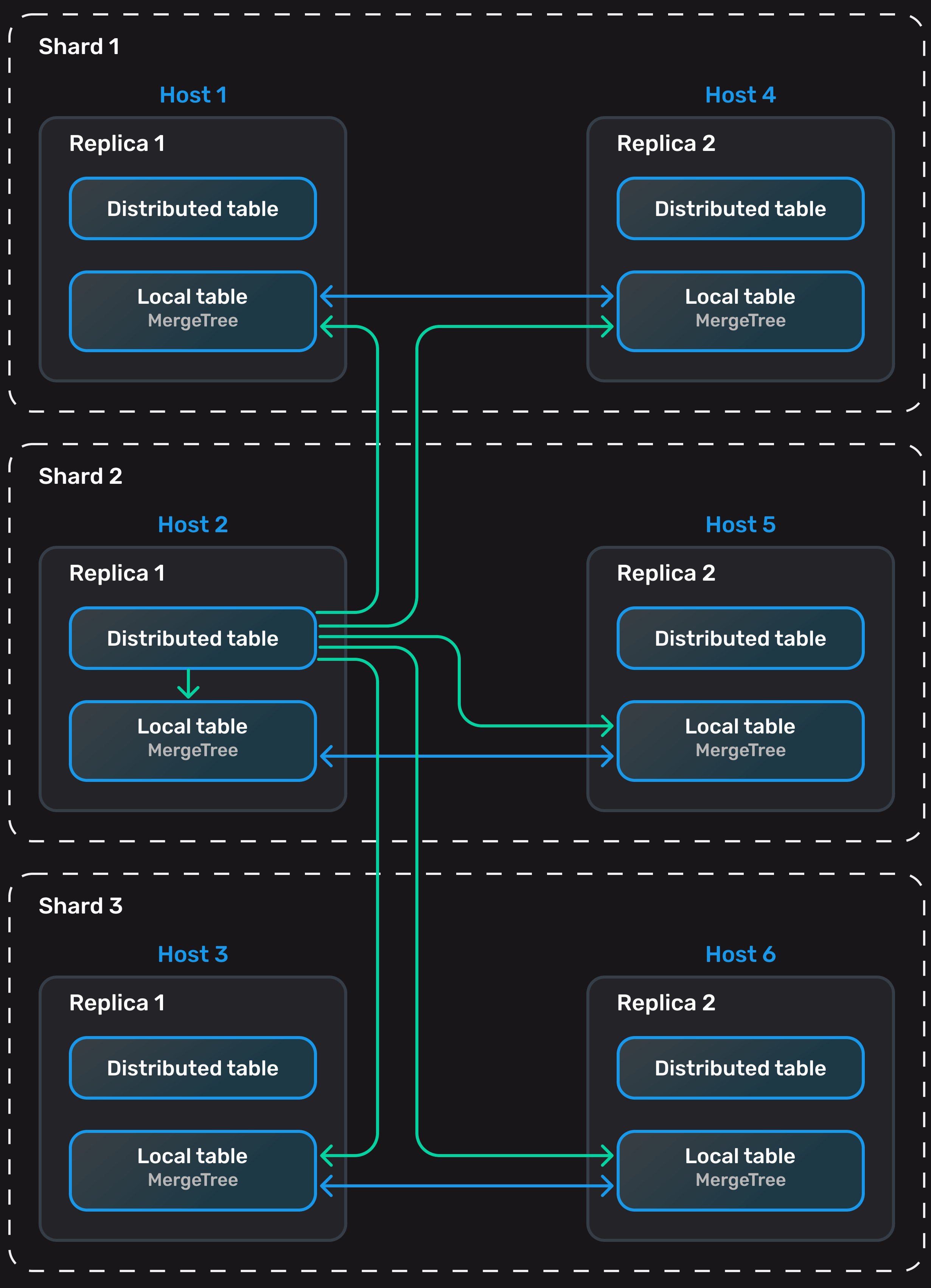
To Table of Contents

Create replicated tables

Create distributed tables

ADQM cluster does not have a master node and a single entry point. Queries can be sent to any host in the cluster.

Consider a typical ADQM cluster — three shards, each one consists of two replicas (in other words, a distributed cluster of 3 nodes with replication factor set to `2`). Shards are servers or groups of servers that store different parts of the same database. **Replicas** are duplicating servers within a shard that store the same data.



Typical ADQM cluster

Local tables are replicated tables responsible for storing data. Distributed tables do not store data, they allow querying multiple local tables distributed across hosts that are grouped in a virtual cluster.

Each server configuration (*config.xml*) should include the following sections:

- `remote_servers` — clusters used for creating distributed tables and executing distributed queries;
- `macros` — macros used to substitute values corresponding to a specific host when creating replicated tables.

The `remote_servers` configuration for the presented cluster:

```
<remote_servers>
  <default_cluster>
        <shard>
            <replica>
                <host>host_1</host>
                <port>9000</port>
            </replica>
            <replica>
                <host>host_4</host>
                <port>9000</port>
            </replica>
        </shard>
        <shard>
            <replica>
                <host>host_2</host>
                <port>9000</port>
            </replica>
            <replica>
                <host>host_5</host>
                <port>9000</port>
            </replica>
        </shard>
        <shard>
            <replica>
                <host>host_3</host>
                <port>9000</port>
            </replica>
            <replica>
                <host>host_6</host>
                <port>9000</port>
            </replica>
        </shard>
  </default_cluster>
</remote_servers>
```

This configuration specifies a distributed cluster (`default_cluster`) that consists of three shards where each shard includes two replicas. Each server parameters are `host` (remote server address) and `port` (TCP port for inter-server communication, usually `9000`).

The `macros` section contains the shard and replica identifiers for each server. For example, macros for `host-1`:

```
<macros>
    <replica>1</replica>
    <shard>1</shard>
</macros>
```

NOTE

To group ADQM hosts into a cluster, use the ADQMDB service's configuration parameters in ADCM — for details, see Configure logical clusters in the ADCM interface.

Create replicated tables

An example of query that creates replicated tables:

```
CREATE TABLE test_local ON CLUSTER default_cluster
(
    id Int32,
    value_string String
)
ENGINE = ReplicatedMergeTree('/clickhouse/tables/{shard}/test_local', '{replica}')
ORDER BY id;
```

The ReplicatedMergeTree table engine is used for creating replicated table. Meta information related to the replication process is stored in ZooKeeper or ClickHouse Keeper, so the ReplicatedMergeTree engine requires the following parameters to build a unique path to each replicated table in ZooKeeper/ClickHouse Keeper:

- path to a table in ZooKeeper/ClickHouse Keeper;
- replica name in ZooKeeper/ClickHouse Keeper (it identifies different replicas of the same table).

In this example, replicated tables are created via a distributed DDL query (`ON CLUSTER`) — a query is executed once on a single server, and the necessary tables will be created on all hosts in the cluster. The shard identifier and replica name specified as the `{shard}` and `{replica}` macros within parameters of ReplicatedMergeTree are replaced with specific values from the `macros` section of each server's configuration.

If you do not use the `ON CLUSTER` clause, perform the `CREATE TABLE` query on each server separately. In this case, you can define the parameters explicitly instead of using macros. However, it is recommended to use macros to reduce the error probability when working with large clusters.

Create distributed tables

The Distributed table engine is used for creating distributed tables, for example:

```
CREATE TABLE test_distr ON CLUSTER default_cluster AS default.test_local
ENGINE = Distributed(default_cluster, default, test_local, rand());
```

In this example, the Distributed table engine accepts the following parameters:

- cluster on which the query will be run (cluster name in the `remote_servers` section of the server's config file);
- database for the replicated table that the distributed table will access;
- replicated table that the distributed table will access;
- sharding key for inserting data into the distributed table (the `rand()` expression is used to randomly distribute data across shards).

Geo-distributed cluster

Elena Dvoryadkina

Contents

To Table of Contents

- Stretched cluster
  - Configure a logical cluster
  - Create distributed tables
- Stretched cluster with local reads
  - Configure logical clusters
  - Create distributed tables
- Stretched cluster with local writes
  - Configure logical clusters
  - Create distributed tables
- Asymmetric cluster
  - Configure logical clusters
  - Create distributed tables
- Cluster with local reads, without native geo replication
  - Configure logical clusters
  - Create distributed tables

A geo-distributed cluster is a cluster whose nodes are located in several geographically distant data centers. This solution ensures high reliability of the system, which will remain operational even if one of the data centers fails.

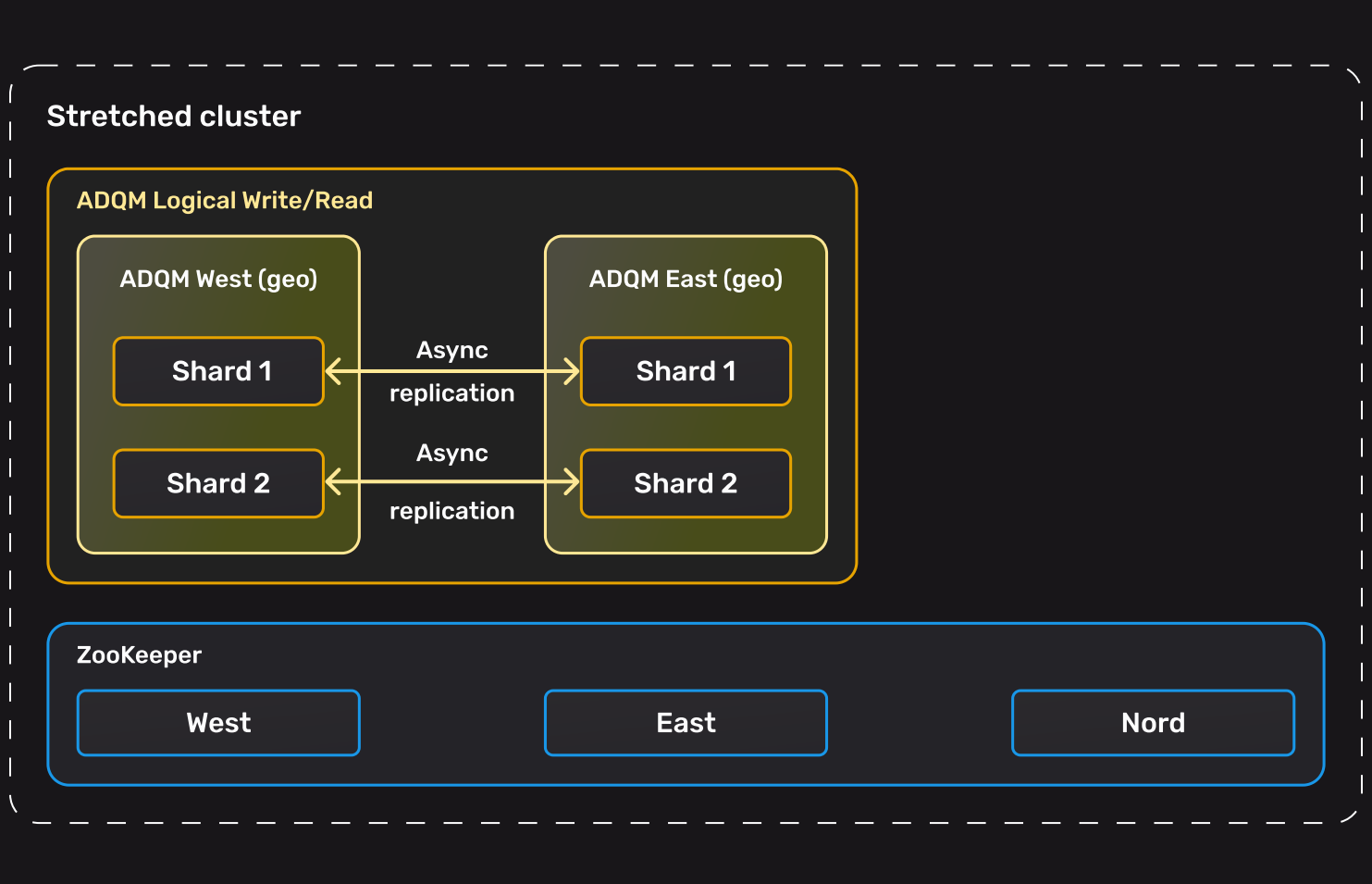
This article contains examples of geo-distributed ADQM cluster topologies. The most suitable topology depends on the required balance between reliability, low latency, disk storage, and maintainability. For example, the tretched clust , tretched cluster with local read and tretched cluster with local write topologies are often used to ensure the cluster performance in the high-latency environment.

In each of the examples below, cluster servers are located in two data centers — West and East.

Stretched cluster

Topology features:

- `SELECT` and `INSERT` queries can be routed to East or West, but it is not guaranteed that all reads are local (it is possible that a query will be executed by replicas in different data centers).
- ZooKeeper nodes are placed in the East and West data centers, as well as in the third Nord data center, as ZooKeeper requires an odd number of nodes.



Stretched cluster

Configure a logical cluster

In each server configuration (*config.xml*), define the `stretched_cluster` cluster that contains two shards with two replicas (located in different data centers) in each one. For shards, set the `internal_replication` parameter to `true` to replicate data through a plicated tabl (data is written to any available replica, and another replica receives its copy of data automatically).

config.xml

```
<remote_servers>
  <stretched_cluster>
    <shard>
      <internal_replication>true</internal_replication>
      <replica>
        <host>shard1.west</host>
        <port>9000</port>
      </replica>
      <replica>
        <host>shard1.east</host>
        <port>9000</port>
      </replica>
    </shard>
    <shard>
      <internal_replication>true</internal_replication>
      <replica>
        <host>shard2.west</host>
        <port>9000</port>
      </replica>
      <replica>
        <host>shard2.east</host>
        <port>9000</port>
      </replica>
    </shard>
  </stretched_cluster>
</remote_servers>
```

Create distributed tables

On all nodes of the cluster, create a replicated table (`replicated_table`) based on the ReplicatedMergeTree engine.

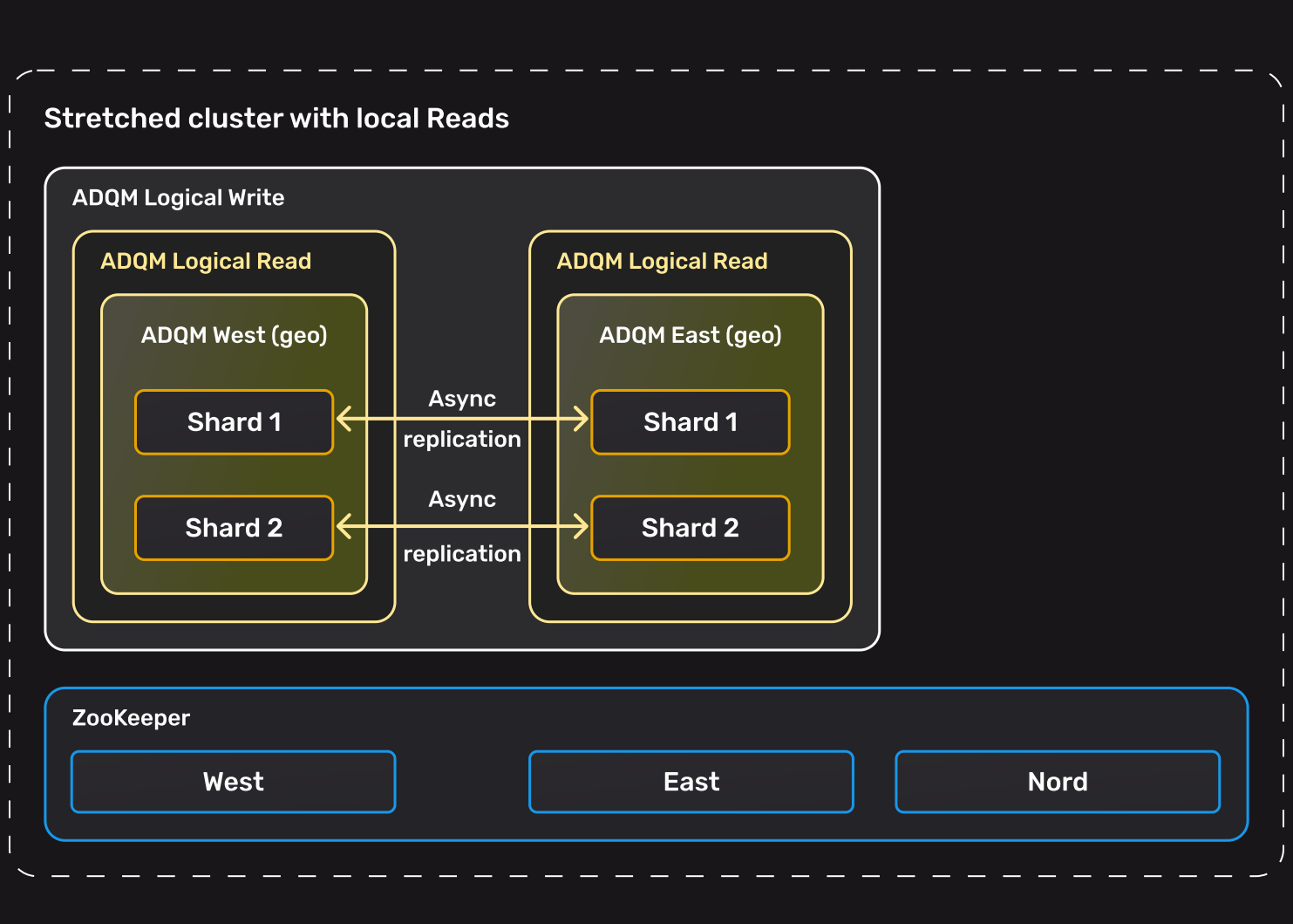
Use the Distributed engine to create distributed tables both for reads and writes on all nodes of the cluster. To do this, run the following query on any node:

```
CREATE TABLE table_distributed ON CLUSTER stretched_cluster AS replicated_table
ENGINE = Distributed(stretched_cluster, default, replicated_table);
```

Stretched cluster with local reads

Topology features:

- `SELECT` and `INSERT` queries can be routed to East or West, and it is guaranteed that all reads are local.
- ZooKeeper nodes are placed in the East and West data centers, as well as in the third Nord data center, as ZooKeeper requires an odd number of nodes.



Stretched cluster with local reads

Configure logical clusters

In each server configuration (*config.xml*), define the following clusters:

- `stretched_cluster_writes` — hosts of both West and East data centers for writes (two shards with two replicas in each one);
- `stretched_cluster_west_reads` — hosts of the West data center for reads (two shards with one replica in each one);
- `stretched_cluster_east_reads` — hosts of the East data center for reads (two shards with one replica in each one).

For shards, set the `internal_replication` parameter to `true` to replicate data through a replicated table (data is written to any available replica, and another replica receives its copy of data automatically).

config.xml

```
<remote_servers>
  <stretched_cluster_writes>
    <shard>
      <internal_replication>true</internal_replication>
      <replica>
        <host>shard1.west</host>
        <port>9000</port>
      </replica>
      <replica>
        <host>shard1.east</host>
        <port>9000</port>
      </replica>
    </shard>
    <shard>
      <internal_replication>true</internal_replication>
      <replica>
        <host>shard2.west</host>
        <port>9000</port>
      </replica>
      <replica>
        <host>shard2.east</host>
        <port>9000</port>
      </replica>
    </shard>
  </stretched_cluster_writes>
  <stretched_cluster_west_reads>
    <shard>
      <internal_replication>true</internal_replication>
      <replica>
        <host>shard1.west</host>
        <port>9000</port>
      </replica>
    </shard>
    <shard>
      <internal_replication>true</internal_replication>
      <replica>
        <host>shard2.west</host>
        <port>9000</port>
      </replica>
    </shard>
  </stretched_cluster_west_reads>
  <stretched_cluster_east_reads>
    <shard>
      <internal_replication>true</internal_replication>
      <replica>
        <host>shard1.east</host>
        <port>9000</port>
      </replica>
    </shard>
    <shard>
      <internal_replication>true</internal_replication>
      <replica>
        <host>shard2.east</host>
        <port>9000</port>
      </replica>
    </shard>
  </stretched_cluster_east_reads>
</remote_servers>
```

Create distributed tables

On all nodes of the cluster, create a replicated table (`replicated_table`) based on the ReplicatedMergeTree engine.

Create distributed tables for reads separately in each data center:

- West:

```
CREATE TABLE table_distributed_west_reads ON CLUSTER stretched_cluster_west_reads AS replicated_table
ENGINE = Distributed(stretched_cluster_west_reads, default, replicated_table);
```

- East:

```
CREATE TABLE table_distributed_east_reads ON CLUSTER stretched_cluster_east_reads AS replicated_table
ENGINE = Distributed(stretched_cluster_east_reads, default, replicated_table);
```

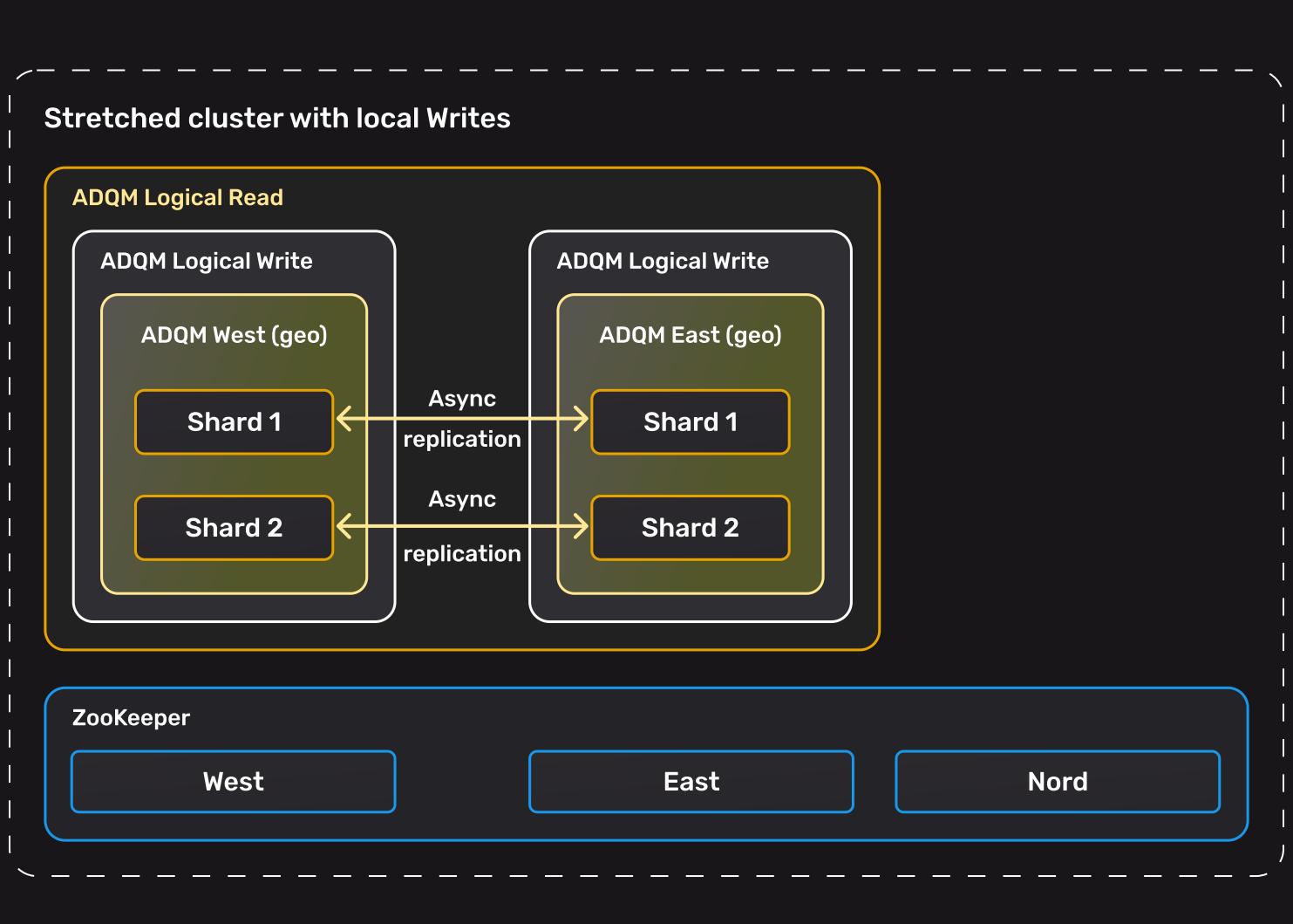
Create identical distributed tables for writes in a logical cluster that combines both data centers:

```
CREATE TABLE table_distributed_writes ON CLUSTER stretched_cluster_writes AS replicated_table
ENGINE = Distributed(stretched_cluster_writes, default, replicated_table);
```

Stretched cluster with local writes

Topology features:

- `SELECT` and `INSERT` queries can be routed to East or West, and it is guaranteed that all writes are local.
- ZooKeeper nodes are placed in the East and West data centers, as well as in the third Nord data center, as ZooKeeper requires an odd number of nodes.



Stretched cluster with local writes

Configure logical clusters

In each server configuration (*config.xml*), define the following clusters:

- `stretched_cluster_reads` — hosts of both West and East data centers for reads (two shards with two replicas in each one);
- `stretched_cluster_west_writes` — hosts of the West data center for writes (two shards with one replica in each one);
- `stretched_cluster_east_writes` — hosts of the East data center for writes (two shards with one replica in each one).

For shards, set the `internal_replication` parameter to `true` to replicate data through a replicated table (data is written to any available replica, and another replica receives its copy of data automatically).

config.xml

```
<remote_servers>
  <stretched_cluster_reads>
    <shard>
      <internal_replication>true</internal_replication>
      <replica>
        <host>shard1.west</host>
        <port>9000</port>
      </replica>
      <replica>
        <host>shard1.east</host>
        <port>9000</port>
      </replica>
    </shard>
    <shard>
      <internal_replication>true</internal_replication>
      <replica>
        <host>shard2.west</host>
        <port>9000</port>
      </replica>
      <replica>
        <host>shard2.east</host>
        <port>9000</port>
      </replica>
    </shard>
  </stretched_cluster_reads>
  <stretched_cluster_west_writes>
    <shard>
      <internal_replication>true</internal_replication>
      <replica>
        <host>shard1.west</host>
        <port>9000</port>
      </replica>
    </shard>
    <shard>
      <internal_replication>true</internal_replication>
      <replica>
        <host>shard2.west</host>
        <port>9000</port>
      </replica>
    </shard>
  </stretched_cluster_west_writes>
  <stretched_cluster_east_writes>
    <shard>
      <internal_replication>true</internal_replication>
      <replica>
        <host>shard1.east</host>
        <port>9000</port>
      </replica>
    </shard>
    <shard>
      <internal_replication>true</internal_replication>
      <replica>
        <host>shard2.east</host>
        <port>9000</port>
      </replica>
    </shard>
  </stretched_cluster_east_writes>
</remote_servers>
```

Create distributed tables

On all nodes of the cluster, create a replicated table (`replicated_table`) based on the ReplicatedMergeTree engine.

Create distributed tables for writes separately in each data center:

- West:

```
CREATE TABLE table_distributed_west_writes ON CLUSTER stretched_cluster_west_writes AS replicated_table
ENGINE = Distributed(stretched_cluster_west_writes, default, replicated_table);
```

- East:

```
CREATE TABLE table_distributed_east_writes ON CLUSTER stretched_cluster_east_writes AS replicated_table
ENGINE = Distributed(stretched_cluster_east_writes, default, replicated_table);
```

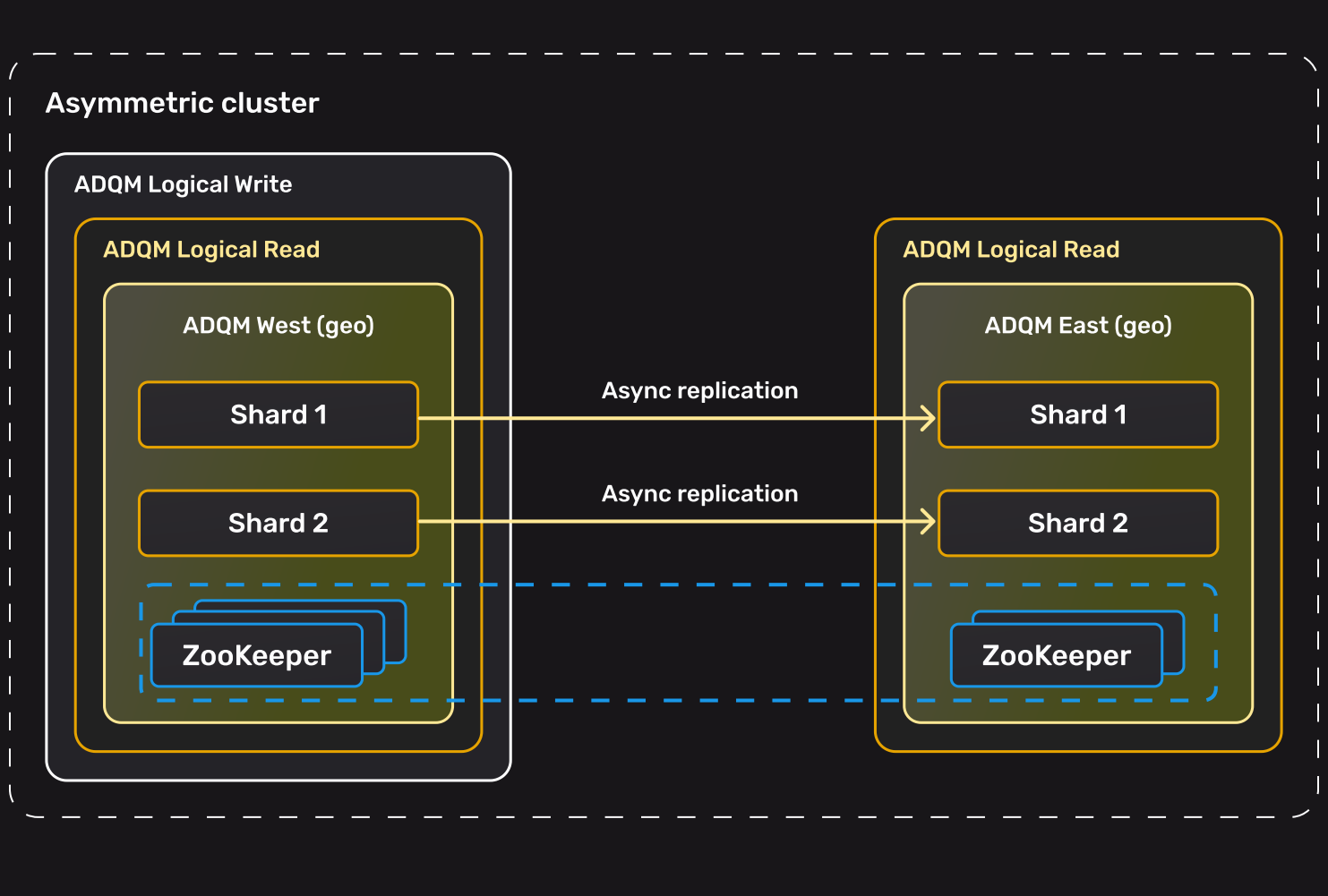
Create identical distributed tables for reads in a logical cluster that combines both data centers:

```
CREATE TABLE table_distributed_reads ON CLUSTER stretched_cluster_reads AS replicated_table
ENGINE = Distributed(stretched_cluster_reads, default, replicated_table);
```

Asymmetric cluster

Topology features:

- Reads are possible both in the West and in East data centers, while writes — in West only.
- Servers of a single ZooKeeper ensemble are located in two data centers (the third data center is not needed), while only servers in West participate in the quorum. Replicas in the East data center cannot be leaders (the `replicated_can_become_leader=0` setting in the `merge_tree` section of *config.xml* controls this behavior).
- The insert_quorum parameter is set to `1` as the number of servers in the quorum should be equal or less than the number of replicas in the main data center (in this example, it is the West data center to which data can be inserted).



Asymmetric cluster

Configure logical clusters

In each server configuration (*config.xml*), define the following clusters:

- `asymmetric_cluster_west` — hosts of the West data center (two shards with one replica in each one);
- `asymmetric_cluster_east` — hosts of the East data center (two shards with one replica in each one).

For shards, set the `internal_replication` parameter to `true` to replicate data through a replicated table (data is written to any available replica, and another replica receives its copy of data automatically).

config.xml

```
<remote_servers>
  <asymmetric_cluster_west>
    <shard>
      <internal_replication>true</internal_replication>
      <replica>
        <host>shard1.west</host>
        <port>9000</port>
      </replica>
    </shard>
    <shard>
      <internal_replication>true</internal_replication>
      <replica>
        <host>shard2.west</host>
        <port>9000</port>
      </replica>
    </shard>
  </asymmetric_cluster_west>
  <asymmetric_cluster_east>
    <shard>
      <internal_replication>true</internal_replication>
      <replica>
        <host>shard1.east</host>
        <port>9000</port>
      </replica>
    </shard>
    <shard>
      <internal_replication>true</internal_replication>
      <replica>
        <host>shard2.east</host>
        <port>9000</port>
      </replica>
    </shard>
  </asymmetric_cluster_east>
</remote_servers>
```

Create distributed tables

On all nodes of the cluster, create a replicated table (`replicated_table`) based on the ReplicatedMergeTree engine.

Create distributed tables for reads separately in each data center:

- West:

```
CREATE TABLE table_distributed_reads ON CLUSTER asymmetric_cluster_west AS replicated_table
ENGINE = Distributed(asymmetric_cluster_west, default, replicated_table);
```

- East:

```
CREATE TABLE table_distributed_reads ON CLUSTER asymmetric_cluster_east AS replicated_table
ENGINE = Distributed(asymmetric_cluster_east, default, replicated_table);
```

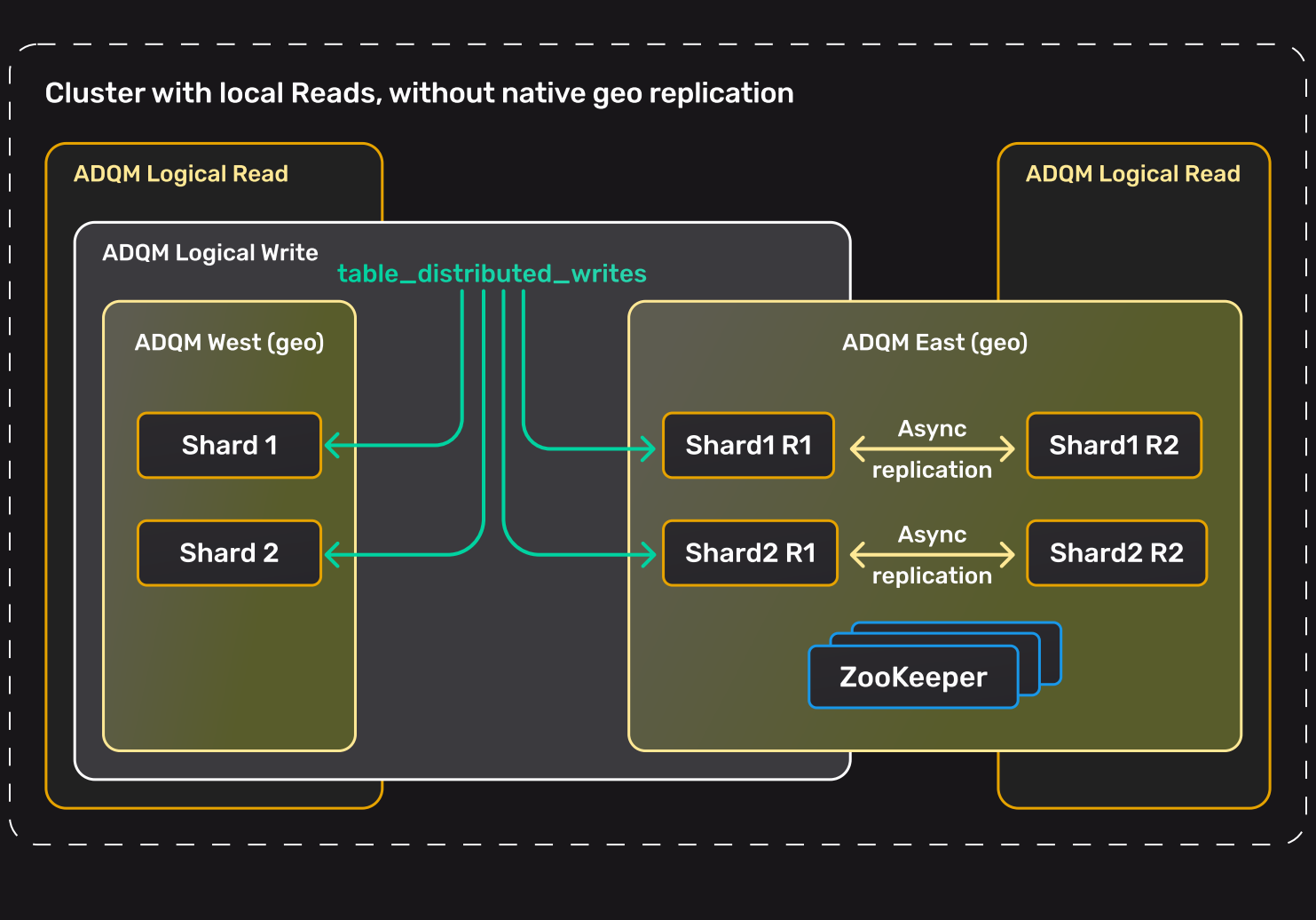
Create a distributed table for writes in the West data center only:

```
CREATE TABLE table_distributed_writes ON CLUSTER asymmetric_cluster_west AS replicated_table
ENGINE = Distributed(asymmetric_cluster_west, default, replicated_table);
```

Cluster with local reads, without native geo replication

Topology features:

- Native ClickHouse replication between data centers is not supported. Replication between data centers is provided by distributed tables.
- Set the insert_quorum parameter to `2` — when executing an `INSERT` query, a distributed table will write data to two replicas located in different data centers.
- `SELECT` and `INSERT` queries can be routed to East or West, and it is guaranteed that all reads are local.
- High latency between West and East is possible.
- ZooKeeper is available in one data center only.



Cluster with local reads, without native geo replication

Configure logical clusters

In each server configuration (*config.xml*), define the following clusters:

- `cluster_writes` — includes hosts of the West and East data centers for writes (two shards with two replicas in each one);
- `cluster_west_reads` — includes hosts of the West data center for reads (two shards with one replica in each one);
- `cluster_east_reads` — includes hosts of the East data center for reads (two shards with one replica in each one).

The `internal_replication` parameter for shards of the `cluster_writes` cluster to which data is written is set to `false` so that the distributed table writes data to both replicas located in different data centers.

config.xml

```
<remote_servers>
  <cluster_writes>
    <shard>
      <internal_replication>false</internal_replication>
      <replica>
        <host>shard1.west</host>
        <port>9000</port>
      </replica>
      <replica>
        <host>sh1r1.east</host>
        <port>9000</port>
      </replica>
    </shard>
    <shard>
      <internal_replication>false</internal_replication>
      <replica>
        <host>shard2.west</host>
        <port>9000</port>
      </replica>
      <replica>
        <host>sh2r1.east</host>
        <port>9000</port>
      </replica>
    </shard>
  </cluster_writes>
  <cluster_west_reads>
    <shard>
      <replica>
        <host>shard1.west</host>
        <port>9000</port>
      </replica>
    </shard>
    <shard>
      <replica>
        <host>shard2.west</host>
        <port>9000</port>
      </replica>
    </shard>
  </cluster_west_reads>
  <cluster_east_reads>
    <shard>
      <internal_replication>true</internal_replication>
      <replica>
        <host>shard1r2.east</host>
        <port>9000</port>
      </replica>
    </shard>
    <shard>
      <internal_replication>true</internal_replication>
      <replica>
        <host>shard2r2.east</host>
        <port>9000</port>
      </replica>
    </shard>
  </cluster_east_reads>
</remote_servers>
```

Create distributed tables

On all nodes of the cluster, create a replicated table (`replicated_table`) based on the ReplicatedMergeTree engine.

Create distributed tables for reads in all nodes of the West data center individually (ZooKeeper is required to make `ON CLUSTER` working):

```
CREATE TABLE table_distributed_reads AS replicated_table
ENGINE = Distributed(cluster_west_reads, default, replicated_table);
```

To create a distributed table for reads in East, it is enough to run the following query on any node:

```
CREATE TABLE table_distributed_reads ON CLUSTER cluster_east_reads AS replicated_table
ENGINE = Distributed(cluster_east_reads, default, replicated_table);
```

A distributed table for writes should be created on the *shard1.west*, *shard2.west*, *shard1r1.east*, *shard2r1.east* hosts:

```
CREATE TABLE table_distributed_writes AS replicated_table
ENGINE = Distributed(cluster_writes, default, replicated_table);
```

Proxying

Elena Dvoryadkina

Contents

To Table of Contents

Install Chproxy

Configure Chproxy via ADCM

For balancing the load and distributing access to the ADQM database, you can use Chproxy. It is an HTTP proxy that can:

- evenly distribute queries between all nodes of the ADQM cluster (all nodes are considered equal);
- monitor health of cluster nodes and prevent from sending requests to unavailable nodes;
- limit HTTP access by a list of allowed IP addresses/IP subnet masks;
- authorize users safely — Chproxy can map users sending requests to users of the ADQM cluster without exposing real usernames and passwords used in ADQM.

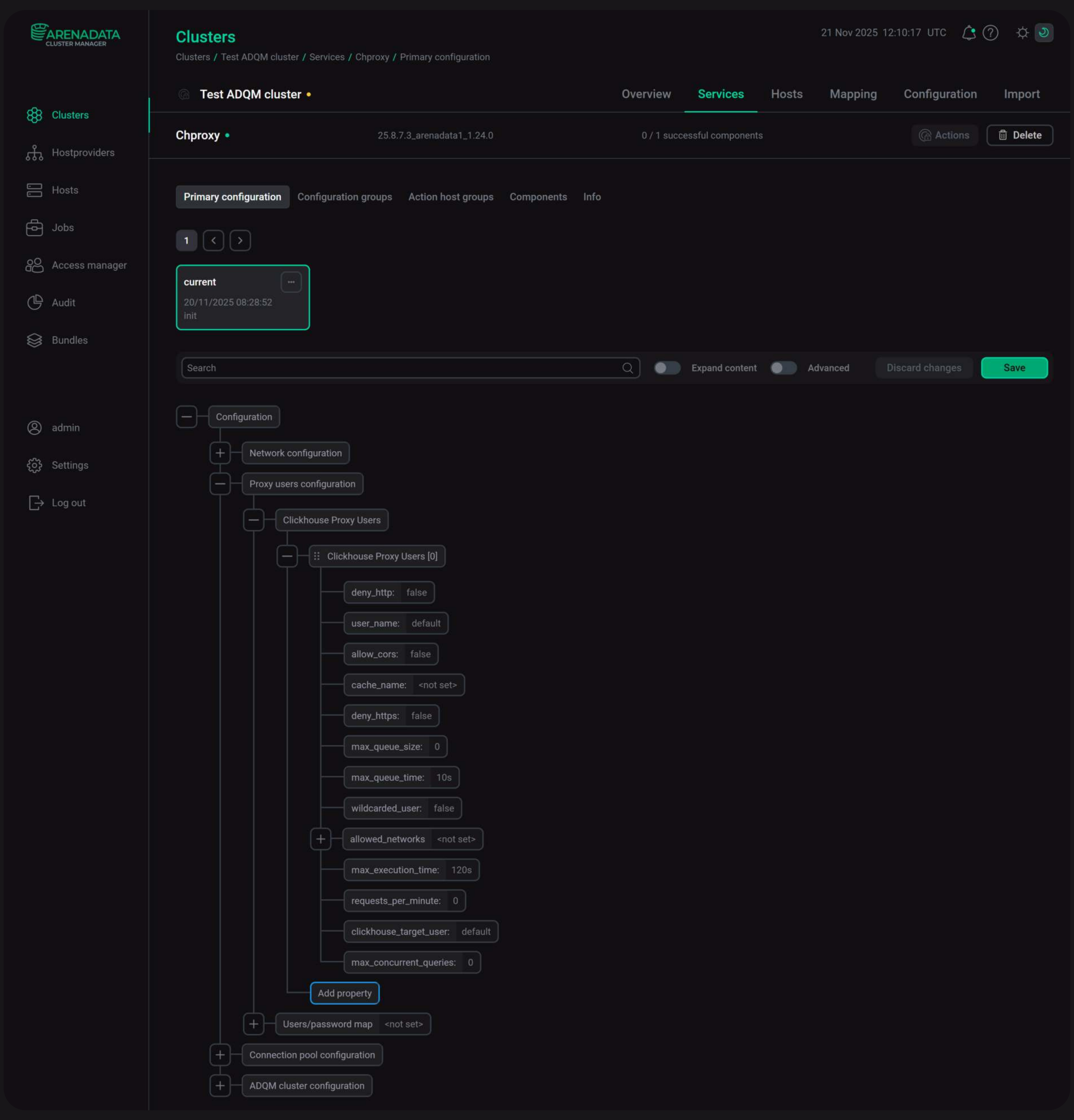
Install Chproxy

The Enterprise edition of ADQM includes the Chproxy service (starting with version 22.3.7.28.1.b1) that you can install to an ADQM cluster as any other service. To do this, follow the steps below:

1. Add the Chproxy service to a cluster.
2. Add the service component to one or more hosts of the cluster.
3. Configure the service.
4. Install the service.

Configure Chproxy via ADCM

You can configure parameters of the Chproxy service in the corresponding window of the ADCM web interface. The Configure services article includes detailed instructions on how to access this window.



ARENADATA
CLUSTER MANAGER
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Clusters
Clusters / Test ADQM cluster / Services / Chproxy / Primary configuration
21 Nov 2025 12:10:17 UTC
Test ADQM cluster
Overview
Services
Hosts
Mapping
Configuration
Import
Chproxy
25.8.7.3_arenadata1_1.24.0
0 / 1 successful components
Actions
Delete
Primary configuration
Configuration groups
Action host groups
Components
Info
1
current
20/11/2025 08:28:52
init
Search
Expand content
Advanced
Discard changes
Save
Configuration
Network configuration
Proxy users configuration
Clickhouse Proxy Users
Clickhouse Proxy Users [0]
Add property
Users/password map <not set>
Connection pool configuration
ADQM cluster configuration

deny_http: false
user_name: default
allow_cors: false
cache_name: <not set>
deny_https: false
max_queue_size: 0
max_queue_time: 10s
wildcarded_user: false
allowed_networks <not set>
max_execution_time: 120s
requests_per_minute: 0
clickhouse_target_user: default
max_concurrent_queries: 0

Chproxy service configuration

The following parameters are available for the Chproxy service:

- ***Network configuration***:
  - ***Enable HTTP*** — listening for HTTP requests (enabled by default);
  - ***HTTP listen port*** — port to listen for HTTP requests (the default is `9090`);
  - ***HTTP allowed networks*** — list of networks from which HTTP access is allowed (you can enter an IP address or IP subnet mask to define a network, the `127.0.0.1` value is added to the list by default);
  - ***Enable HTTPS*** — listening for HTTPS requests (disabled by default);
  - ***HTTPS listen port*** — port to listen for HTTPS requests (the default is `8543`);
  - ***HTTPS allowed networks*** — list of networks from which HTTPS access is allowed (you can enter an IP address or IP subnet mask to define a network, the `127.0.0.1` value is added to the list by default);
  - ***Certificate file*** — path to the server SSL certificate file;
  - ***Private key file*** — path to the file with the private key of the server SSL certificate.

- ***Proxy users configuration***:
  - ***Clickhouse Proxy Users*** — users whose requests will be proxied to ADQM. Using the settings in this section, you can set up the configuration for each user separately: configure access restrictions based on a list of IP addresses or subnet masks, prohibit or allow HTTP and HTTPS connections, enable caching responses, set limits on the duration and frequency of requests, as well as limit the number of concurrent requests.

    To map multiple distinct input users to a single ADQM user, you can create a wildcarded user configuration. To do this, enable the ***wildcarded_user*** option, enter a Chproxy user name pattern (like `<prefix>*` or `*<suffix>`) in the ***user_name*** field, and specify an ADQM user in the ***clickhouse_target_user*** field. For example, if `user_name = dba_*` and `clickhouse_target_user = db_admin` (where `db_admin` is an existing ADQM user), then any user with a name matching this pattern (for example, `dba_john` and `dba_mary`) will be authorized in ADQM as `db_admin`.
  - ***Users/password map*** — passwords of users specified in ***Clickhouse Proxy Users***.

- ***Connection pool configuration***:
  - ***Max idle connections*** — maximum number of idle connections in the pool;
  - ***Max idle connections per host*** — maximum number of idle connections per host.

- ***ADQM cluster configuration***:
  - ***Interval*** — interval for checking all cluster nodes for availability (the default is `5s`);
  - ***Timeout*** — waiting time of response from cluster nodes (the default is `5s`).

  These parameters are used for checking health of cluster nodes. Chproxy service components send a health check request to all nodes of the ADQM cluster at a frequency specified by the ***Interval*** parameter. If this request does not receive a response from some node of the cluster within the time specified by the ***Timeout*** parameter, Chproxy will stop sending queries to this node.

Hardware requirements

Elena Dvoryadkina

Contents

To Table of Contents

- ClickHouse server
- Zookeeper server
- Arenadata Enterprise Tools
- Arenadata Monitoring
- ADQM Control

IMPORTANT

The following system requirements are **minimal**. The target sizing should be calculated based on the customer requirements.

ClickHouse server

The minimum hardware requirements for ClickHouse servers to install an ADQM cluster are:

- Physical server.
- Storage: *ext4*-formatted storage device mounted on file system.
- CPU: 8 cores or more.
- RAM: 64 GB or more.
- Network: servers should be available on ports 9000, 8123 and 9009 within the cluster.

For quick marts, an array of SSD drives is suitable. For storing big tables with facts, RAID 10 of HDDs is preferable. It is recommended to use a software RAID on Linux (mdadm), and not recommended to use LVM. When creating a RAID 10, select the *far* layout.

On each ClickHouse server in a cluster, the device schema, disks, and mount point should be the same.

Examples of requirements for ClickHouse servers depending on the load pattern for the ADQM cluster are listed below.

Proof of concept

Number of nodes	Storage	CPU	RAM	Network
1-4	1 x 500 GB HDD for OS 4 x 1.2 ТБ SAS HDD RAID 10 for Data	8 cores	96 GB	1 x 10 Gbps

Quick marts

Number of nodes	Storage	CPU	RAM	Network
4	2 x 500 GB SSD RAID 1 for OS 12 x 1 TB SAS12G SSD MixedUse RAID 10 (PCIe Gen3 x8) for Data	24 cores (2 x 12)	256 GB (8 x 32)	4 x 25 Gbps (2 adapters PCIe Gen4 x4)

Big tables with facts

Number of nodes	Storage	CPU	RAM	Network
4	2 x 500 GB SSD RAID 1 for OS 12 x 1.8 TB SAS10K HDD RAID 10 for Data	24 cores	256 GB	2 x 25 Gbps (2 adapters PCIe Gen3 x4)

Zookeeper server

The minimum hardware requirements for Zookeeper servers in an ADQM cluster are:

- Physical or virtual server. For pilot clusters, you can use the same servers as for ClickHouse. For large clusters, at least three physical servers are required.
- Storage: 100 GB or more; for large clusters, a fast SSD or RAID 1 is preferred.
- CPU: 4 cores or more.
- RAM: 4 GB or more.

The following table shows an example of the Zookeeper server configuration.

Number of nodes	Storage	CPU	RAM	Network
3 or more	2 x 500 GB SSD RAID 1 for OS 2 x 2.4 TB SSD RAID 1 for Data	4 cores (P-cores)	8 GB	2 x 10 Gbps

IMPORTANT

For guaranteed HA (High Availability), it is necessary to have an odd number of Zookeeper server instances.

Arenadata Enterprise Tools

If you use ET along with your product, take into account the following requirements:

- Virtual server.
- CPU: 4 cores or more.
- RAM: 8 GB or more.
- Disk: 50 GB or more.

Arenadata Monitoring

If you use ADM along with your product, take into account the following requirements:

- Virtual server.
- CPU: 8 cores or more.
- RAM: 16 GB or more.
- Disk: 10 GB per monitored host or more.

ADQM Control

If you use ADQM Control along with your product, take into account the following requirements for hosts to install ADQM Control's components:

- Backend:
  - CPU: 4 cores or more.
  - RAM: 8 GB or more.
  - Disk: SSD 50 GB or more.
- Database (PostgreSQL):
  - CPU: 4 cores or more.
  - RAM: 8 GB or more.
  - Disk: SSD 50 GB or more depending of ADQMDB load.

NOTE

ADQM Control is supported for ADQM clusters of versions not older than one year.

Network requirements

Elena Dvoryadkina

To Table of Contents

- One or more dedicated and separate peer-to-peer Interconnect networks are required for internal cluster communication.
- All cluster servers should be connected to Interconnect networks.
- All servers should have fully qualified domain names (FQDN) and the ability to communicate with each other at required speeds.
- IP address of each server should be known.
- Ping access to any server (ICMP) should be provided.
- For online installation of ADQM, access to Arenadata Cluster Manager deployed on a dedicated server is required. See the Install ADCM article for details on how to install Arenadata Cluster Manager.
- For offline installation of ADQM, access to Arenadata Enterprise Tools deployed on a dedicated server is required. The Install Enterprise Tools cluster section contains instructions on how to install Arenadata Enterprise Tools.
- Ports below should be open on servers where you install the cluster.

NOTE

Each port refers to the server where a specific service is installed.

ADQM

Service	Port	Purpose
Common	22	Port for connection to SSH provider
	2015	TCP port to send metrics to the ADM monitoring cluster (deprecated)
	2016	UDP port to send metrics to the ADM monitoring cluster (deprecated)
ClickHouse	8123	HTTP port for client or proxy connections
	8443	Port for secure client connections via the HTTPS interface (see Encryption of database connections for details)
	9000	Port for client connections via the Native (TCP) protocol. This port is also used for interserver interaction when performing distributed queries
	9440	TCP port for secure communication with clients and interserver communication for distributed queries
	9009	Port for communication between replicas
	9100	Port for client connections via gRPC
	9019	JDBC Bridge port
	9363	Port to listen for ClickHouse metrics in the Prometheus format
ZooKeeper	2181	Service access port
	2888 3888	Server-to-server communication ports for quorum components
	2281	Port for secure client connections
	9020	Port to listen for ZooKeeper metrics in the Prometheus format
ClickHouse Keeper	2129	TCP port for client connections
	9444	Recommended port for RAFT connections (also used for secure communications if `<secure>1</secure>` is enabled)
	9281	Recommended port for secure SSL-encrypted interaction
	9010	Port to listen for ClickHouse Keeper metrics in the Prometheus format
Monitoring	11210	Port to access the Grafana web interface
	11203	Port to access ADQM host system metrics in the Prometheus format
Chproxy	9090	HTTP port for client connections
	8543	HTTPS port for client connections

Monitoring

Port	Purpose
3000	HTTP port to access the Grafana web interface. You can change the default value when installing the monitoring cluster on the configuration page of the Grafana service in the ***ip_and_ports*** section
80	TCP port to access the Graphite web interface. You can change the default value when installing the monitoring cluster on the configuration page of the Graphite service in the ***ip_and_ports*** section

Enterprise Tools

Port	Purpose
443	TCP port used by Arenadata Enterprise Tools (Docker Registry service)
81	TCP port used by Arenadata Enterprise Tools (access to the official repositories for offline installation)

ADCM

Port	Purpose
8000	TCP port to send statuses of cluster components to ADCM

Software requirements

Elena Dvoryadkina

Contents

To Table of Contents

Operating system
Required information
Specifics of ADQM installation

Operating system

1. ADQM supports only x86_64 architecture and the following operating systems:
    - Centos 7
    - RHEL 7
    - Alt Linux 8.4 SP (starting with the ADQM version 22.3.2.2.1.b1), 10.2 (starting with the ADQM version 25.3.2.39.1.b1)
    - Astra Linux SE 1.7 Orel with Axiom JDK (for the Enterprise edition of ADQM, starting with the version 22.8.12.45.1.b1), 1.7.6 Voronezh (starting with the ADQM version 25.3.2.39.1.b1)
    - RED OS 7.3 Certified edition (for the Enterprise edition of ADQM, starting with the version 23.3.2.37.1.b1)
    - Ubuntu 22.04 (starting with the ADQM version 24.3.2.23.2.b1)
2. You need to prepare all dependent repositories together with the chosen operating system.
3. All machines in the cluster should have access to the official repositories (or local mirrors should be created):
    - CentOS Extras
    - CentOS Updates
    - CentOS Base
4. For online installation of ADQM, the machines should have access to Arenadata repositories specified on the Configuration tab in the ADQM cluster menu available in ADCM UI. For offline installation, provide access to the host where the Enterprise Tools cluster is installed.

Required information

The following information and conditions are required to install an ADQM cluster:

- IP addresses and hostnames of all servers in the cluster, mapped to their roles.
- Access to the `root` user or any user with `sudo` rights on each server in the cluster.
- Correctly configured DNS. All hostnames and fully qualified domain names (FQDN) should resolve equally on all servers in the cluster.

Specifics of ADQM installation

1. The installation process modifies the following system files that should not be controlled by the configuration management system (if available):
    - */etc/hosts*;
    - */etc/selinux/config*;
    - */etc/sysctl.conf*.
2. The installation process creates files in */usr/lib/systemd/system/*. These files should not be controlled by the configuration management system (if available).
3. Services installed on a cluster should not be controlled by the configuration management system (if available).

Installation

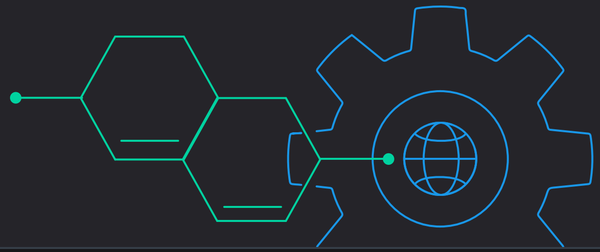
To Table of Contents

This tutorial guides you through the process of installing an Arenadata QuickMarts (ADQM) cluster. The easiest way to install it is to use Arenadata Cluster Manager (ADCM) — a separate Arenadata product designed for simple, convenient, and fast software deployment and exploitation.

You can install ADQM via ADCM in two ways.

Online installation

on hosts with access to Internet



Offline installation

on hosts without access to Internet

Online installation

To Table of Contents

This topic describes **online** installation of ADQM. It can be applied to the hosts with access to Internet.

The process of installation via ADCM includes the following steps.

Step 1.

Install ADCM

Step 2.

Prepare hosts

ARENADATA
QUICKMARTS

Step 3.

Install ADQM cluster

Step 4.

Install monitoring

ARENADATA
ADQM CONTROL

Step 5.

Install ADQM Control

Install ADCM

Daria Barysheva

Contents

To Table of Contents

- Step 1. Check prerequisites
  - Software and hardware requirements
  - Install Docker
  - Disable SELinux
  - Stop firewalld
  - Allow access to the Docker CLI without root privileges
- Step 2. Install ADCM
- Step 3. Start ADCM
- Step 4. Check installation
  - Check the Docker container status
  - Check the 8000 port availability
  - Check the ADCM web-interface
  - Set the ADCM URL
- Step 5. Update ADCM

ADCM requires a separate host. Before installing ADCM, we recommend to read its documentation.

The installation steps are listed below.

Step 1. Check prerequisites

Software and hardware requirements

ADCM is a software that is distributed in the form of a Docker image. To install it, you need:

- One of the following operating systems:
  - Ubuntu 22.04
  - CentOS 7
  - RHEL 7
  - Alt Linux SP 8
  - Alt Linux SP 10.2
  - Astra Linux SE 1.7 Orel
  - Astra Linux 1.7.6 Voronezh
  - RED OS 7.3 Certified edition
- A user account with `sudo` privileges.
- An access to the official repository of CentOS Extras and CentOS Base. This repository is enabled by default, but if you have disabled it — you need to re-enable it (e.g. via editing the repository configuration file in the */etc/yum.repos.d/* directory).
- The software package installer YUM/APT.
- The hardware that meets the following conditions:

IMPORTANT

The following system requirements are **minimal**. The target sizing should be calculated based on the customer requirements.

  - **CPU**: 4 CPU cores;
  - **RAM**: 16 GB;
  - **HDD**: 50 GB.

IMPORTANT

Starting with ADCM 2.7.0, any Container Registry Client with downloaded images of ADPG, ADCM, and ADCM Installer will be required to install ADPG without the Internet connection and use it as the ADCM database. The installation is done via the ADCM Installer by specifying the URL of the local storage of the ADCM and ADPG images.

To grant the user the necessary rights to install ADCM, do the following:

1. Grant the `root` privileges for the user:

CentOS 7/RHEL 7/RED OS 7.3/Alt Linux SP 8

```
$ sudo usermod -aG wheel <user>
```

where `<user>` is a username.

Ubuntu 22.04/Astra Linux SE 1.7 Orel

```
$ sudo usermod -aG sudo <user>
```

where `<user>` is a username.

2. Make sure that the */etc/sudoers* file contains the record specified below depending on the operating system with the following command:

```
$ sudo vi /etc/sudoers
```

To return to the command line, press `Esc`, type `:q!`, and press `Enter`.

CentOS 7/RHEL 7/RED OS 7.3

```
## Allows people in group wheel to run all commands
%wheel  ALL=(ALL)       ALL
```

Ubuntu 22.04/Astra Linux SE 1.7 Orel

```
# Allow members of group sudo to execute any command
%sudo   ALL=(ALL:ALL) ALL
```

Alt Linux SP 8

```
## Uncomment to allow members of group wheel to execute any command
WHEEL_USERS ALL=(ALL:ALL) ALL
```

Install Docker

CAUTION

Do not install Docker from the official Docker site. Instead, use the official repositories of your Linux distribution. The repositories provided by operating system developers tend to be more stable and secure.

Below are the step-by-step procedures of Docker installation on the examples of several operating systems supported by ADCM. These steps may differ for operating system versions other than those specified on tabs.

CentOS 7/RHEL 7

1. Using `root` privileges, update all packages that are currently installed in your system:

```
$ sudo yum update -y
```

2. Install the packages required for Docker installation and usage:

```
$ sudo yum install -y yum-utils docker device-mapper-persistent-data lvm2
```

where:

- `yum-utils` is the package that provides the `yum-config-manager` utility;
- `device-mapper-persistent-data` and `lvm2` are packages required by the devicemapper storage driver.

3. Start Docker:

```
$ sudo systemctl start docker
```

4. Enable Docker as a system service:

```
$ sudo systemctl enable docker
```

Ubuntu 22.04

1. Using `root` privileges, update all packages that are currently installed in your system:

```
$ sudo apt-get update
```

2. Install the packages required for Docker installation and usage:

```
$ sudo apt-get install ca-certificates curl
```

where:

- `ca-certificates` is the package of Root CA certificates;
- `curl` is the package for working with web resources.

Confirm the use of additional disk space during the execution of the command.

3. Add the official GPG key for the Docker repository:

```
$ sudo install -m 0755 -d /etc/apt/keyrings
$ sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.asc
$ sudo chmod a+r /etc/apt/keyrings/docker.asc
```

4. Add the Docker repository:

```
$ echo \
"deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc] https://download.docker.com/linux/ubuntu \
$(. /etc/os-release && echo "jammy") stable" | \
sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

5. Update all packages, including packages for the Docker repository:

```
$ sudo apt-get update
```

6. Install Docker Engine:

```
$ sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin
```

Confirm the use of additional disk space during the execution of the command.

7. Start Docker:

```
$ sudo systemctl start docker
```

8. Enable Docker as a system service:

```
$ sudo systemctl enable docker
```

Alt Linux SP 8

1. Using `root` privileges, update all packages that are currently installed in your system:

```
$ sudo apt-get update
```

2. Install Docker Engine:

```
$ sudo apt-get install docker-engine
```

Confirm the use of additional disk space during the execution of the command.

3. Start and enable Docker as a system service:

```
$ sudo systemctl enable --now docker
```

Astra Linux SE 1.7 Orel

1. Using `root` privileges, update all packages that are currently installed in your system:

```
$ sudo apt-get update
```

2. Install the packages required for Docker installation and usage:

```
$ sudo apt-get install ca-certificates curl net-tools
```

where:

- `ca-certificates` is the package of Root CA certificates;
- `curl` is the package for working with web resources;
- `net-tools` is the package for controlling the network subsystem of the Linux kernel, in particular for executing the `netstat` command.

Confirm the use of additional disk space during the execution of the command.

3. Add the official GPG key for the Docker repository:

```
$ curl -fsSL https://download.docker.com/linux/debian/gpg | sudo apt-key add -
```

4. Add the Docker repository:

```
$ echo "deb [arch=amd64] https://download.docker.com/linux/debian stretch stable" | sudo tee -a /etc/apt/sources.list
```

5. Update all packages, including packages for the Docker repository:

```
$ sudo apt-get update
```

6. Install Docker Engine:

```
$ sudo apt install docker-ce docker-ce-cli containerd.io
```

Confirm the use of additional disk space during the execution of the command.

7. Start Docker:

```
$ sudo systemctl start docker
```

8. Enable Docker as a system service:

```
$ sudo systemctl enable docker
```

RED OS 7.3

1. Install Docker Engine:

```
$ sudo dnf install docker-ce docker-ce-cli
```

Confirm the use of additional disk space during the execution of the command.

2. Start and enable Docker as a system service:

```
$ sudo systemctl enable docker --now
```

Disable SELinux

NOTE

It is recommended to disable SELinux only if CentOS 7 or RHEL 7 is used.

To disable SELinux permanently, set `SELINUX=disabled` in the */etc/selinux/config* file, and reboot your system.

You can edit the file via the `vi` or `vim` command:

```
$ sudo vi /etc/selinux/config
```

The content of the changed file should look like this:

```
# This file controls the state of SELinux on the system.
# SELINUX= can take one of these three values:
#       enforcing - SELinux security policy is enforced.
#       permissive - SELinux prints warnings instead of enforcing.
#       disabled - SELinux is fully disabled.
SELINUX=disabled
# SELINUXTYPE= type of policy in use. Possible values are:
#       targeted - Only targeted network daemons are protected.
#       strict - Full SELinux protection.
SELINUXTYPE=targeted
```

You can also disable SELinux temporarily via the following command:

```
$ sudo setenforce 0
```

Still, it is strongly recommended to disable SELinux permanently, so that it does not restart on each system reboot.

CAUTION

Do not forget to reboot your host after SELinux is disabled.

Stop firewalld

If you use the firewalld service, stop it before installing ADCM.

Before stopping firewalld, make sure that it is installed, started, and enabled:

```
$ sudo systemctl status firewalld
```

The example of the command execution result:

```
● firewalld.service - firewalld - dynamic firewall daemon
   Loaded: loaded (/usr/lib/systemd/system/firewalld.service; enabled; vendor preset: enabled)
   Active: active (running) since Mon 2024-12-02 10:02:57 UTC; 3min 25s ago
     Docs: man:firewalld(1)
 Main PID: 3023 (firewalld)
    Tasks: 2
   Memory: 22.8M
   CGroup: /system.slice/firewalld.service
           └─3023 /usr/bin/python2 -Es /usr/sbin/firewalld --nofork --nopid
```

Stop firewalld:

```
$ sudo systemctl stop firewalld
```

As an alternative, you can disable the firewalld service, so that it will not apply rules to network packets:

```
$ sudo systemctl disable firewalld
```

Allow access to the Docker CLI without root privileges

If in the future you want to access the Docker CLI without using `root` privileges, you should create a `docker` user group (if it does not exist) and add the current user to this group with the certain rights:

```
$ sudo groupadd docker
$ sudo usermod -a -G docker $USER
```

After running these commands, you should re-login under the current user.

Step 2. Install ADCM

NOTE

- It is recommended to configure PostgreSQL as an external database or to deploy an embedded database using ADCM Installer. For more information on configuring PostgreSQL, see the Work with an external database article.
- Starting with ADCM 2.5.0, the minimum required version of PostgreSQL is 14.
- To see the full list of images, refer to the ADCM repository on Docker Hub.
- ADCM has some mandatory persistent information stored in the */adcm/data* directory inside a container. This means that you have to map a volume to that directory and provide backups for that volume.

To install ADCM, perform the following steps:

1. Pull a Docker image from the Arenadata Docker Registry:

```
$ sudo docker pull hub.arenadata.io/adcm/adcm:<version>
```

where `<version>` is a desired ADCM image version in one of the following formats:

- `<major>.<minor>.<patch>` — if you need a specific ADCM patch. For example, `2.0.0`.
- `<major>.<minor>` — if you need the last patch within the selected ADCM version. For example, `2.0`.

IMPORTANT

- `latest` was used for earlier versions of ADCM and is no longer supported starting from version `2.0.0`.
- Before ADCM 2.0.0, the following format was used for versioning: `YYYY.MM.DD.HH`.

2. Add a container based on that image:

NOTE

The command below creates a container with ADCM on the `8000` port and establishes a connection to an external database. All its data will be stored on the host machine in the */opt/adcm/* directory. The environment variables description is given in the table below.

```
$ sudo docker create --name adcm -p 8000:8000 -v /opt/adcm:/adcm/data -e DB_HOST="<DATABASE_HOSTNAME_OR_IP_ADDRESS>" -e DB_PORT="
<DATABASE_TCP_PORT>" -e DB_USER="<DATABASE_USERNAME>" -e DB_NAME="<DATABASE_NAME>" -e DB_PASS="<DATABASE_USER_PASSWORD>" -e
DB_OPTIONS="<DATABASE_OPTIONS>" hub.arenadata.io/adcm/adcm:<version>
```

If you want to run ADCM **with SELinux enabled**, a Docker container should be created with the `:Z` option in the data catalog parameter:

```
$ sudo docker create --name adcm -p 8000:8000 -v /opt/adcm:/adcm/data:Z hub.arenadata.io/adcm/adcm:<version>
```

Optionally, when you create a container, you can specify the environment variables listed below. Each variable is specified in the following format: `-e <name>="<value>"`. For example:

```
$ sudo docker create --name adcm -p 8000:8000 -v /opt/adcm:/adcm/data -e LOG_LEVEL="INFO" hub.arenadata.io/adcm/adcm:<version>
```

NOTE

Special environment variables, such as `STATUS_LOG_LEVEL`, `ADCM_LOG_LEVEL`, `AUDIT_LOG_LEVEL`, `LDAP_LOG_LEVEL`, `BACKGROUND_TASKS_LOG_LEVEL`, and `TASK_RUNNER_LOG_LEVEL` have higher priority than the `LOG_LEVEL` variable.

Environment variables

Name	Description
DB_HOST	Name or IP address of the host where the database is deployed
DB_PORT	TCP port number that is used to connect to the database
DB_USER	Username that is used to connect to the database
DB_NAME	Database name
DB_PASS	Password that is used to connect to the database

DB_OPTIONS	Options for connecting to the database:
	- `sslmode` — flag that is used for establishing an SSL connection. Possible values:
	  - `disable` — only try a non-SSL connection.
	  - `allow` — first try a non-SSL connection. If that fails, try an SSL connection.
	  - `prefer` (default) — first try an SSL connection. If that fails, try a non-SSL connection.
	  - `require` — only try an SSL connection. If a root certificate authority (CA) file is present, verify the certificate in the same way as if `verify-ca` was specified.
	  - `verify-ca` — only try an SSL connection, and verify that the server certificate is issued by a trusted CA.
	  - `verify-full` — only try an SSL connection, and verify that the server certificate is issued by a trusted CA and that the requested server host name matches that in the certificate.
	- `sslcert` — specifies the file name of the client SSL certificate. This parameter is ignored if the connection is made without SSL.
	- `sslkey` — specifies the location for the secret key used for the client certificate. This parameter is ignored if the connection is made without SSL.
	- `sslrootcert` — specifies the name of a file containing SSL certificate authority certificate(s). If the file exists, the server certificate will be verified to be signed by one of these authorities. A special system value may be specified instead, in which case the system trusted CA roots will be loaded. The exact locations of these root certificates differ by SSL implementation and platform.

DEFAULT_ADCM_URL	URL of the host where the ADCM container is deployed. Use the following format: `http://<IP>:8000/`. If `DEFAULT_ADCM_URL` is not specified, you should set the URL manually in the ***ADCM's URL*** field on the ***Settings*** page
LOG_LEVEL	Log level for all log types. Possible values: - `DEBUG` - `INFO` - `WARNING` - `ERROR` (default value) - `CRITICAL`
STATUS_LOG_LEVEL	Log level for *status.log*. Possible values: - `DEBUG` - `INFO` - `WARNING` - `ERROR` - `CRITICAL`
ADCM_LOG_LEVEL	Log level for *adcm.log* and *adcm_debug.log*. Possible values: - `DEBUG` - `INFO` - `WARNING` - `ERROR` - `CRITICAL`
AUDIT_LOG_LEVEL	Log level for *audit.log*. Possible values: - `DEBUG` - `INFO` (default value) - `WARNING` - `ERROR` - `CRITICAL`
LDAP_LOG_LEVEL	Log level for *ldap.log*. Possible values: - `DEBUG` - `INFO` - `WARNING` - `ERROR` - `CRITICAL`
BACKGROUND_TASKS_LOG_LEVEL	Log level for *cron_task.log*. Possible values: - `DEBUG` - `INFO` - `WARNING` - `ERROR` - `CRITICAL`
TASK_RUNNER_LOG_LEVEL	Log level for *task_runner.err*. Possible values: - `DEBUG` - `INFO` - `WARNING` - `ERROR` - `CRITICAL`
MIGRATION_MODE	Flag that indicates whether migration mode (ability to migrate ADCM database from SQLite to PostgreSQL) is turned on. For more information, see Migration. Possible values: - `0` — migration mode is off; - `1` — migration mode is on.
CONSUL_URL	HashiCorp Consul server URL. Use the following format: `http://<IP>:8500/`
CONSUL_DATACENTER	HashiCorp Consul Datacenter. If the variable value is not specified, the value specified in HashiCorp Consul by default will be used
CONSUL_CACERT_FILE	Path to the certificate authority (CA) certificate file if encryption is enabled in HashiCorp Consul
CONSUL_CLIENT_CERT_FILE	Path to the client certificate file if encryption is enabled in HashiCorp Consul
CONSUL_CLIENT_KEY_FILE	Path to the client private key file if encryption is enabled in HashiCorp Consul

Step 3. Start ADCM

ADCM is just a container operated by Docker. So you are able to use simple Docker commands to start and stop ADCM:

- To **start ADCM**, use the following command.

```
$ sudo docker start adcm
```

- To **stop ADCM**, use the following command.

```
$ sudo docker stop adcm
```

- In order to provide the **automatic start** of the Docker container after unexpected errors, use the following command.

```
$ sudo docker update --restart=on-failure adcm
```

NOTE

- For more information about Docker restart policies, refer to the Docker documentation.
- If you allowed your user to access the Docker CLI without `root` privileges, then you can omit the `sudo` keyword from the commands listed above.

Step 4. Check installation

Check the Docker container status

To make sure that the Docker container `adcm` is successfully deployed, run the following command:

```
$ sudo docker container ls
```

The example of the command execution result is listed below. Make sure that the `STATUS` value is equal to `Up`.

```
CONTAINER ID        IMAGE                                COMMAND             CREATED             STATUS              PORTS
NAMES
74b5628146e5        hub.arenadata.io/adcm/adcm:2.0   "/etc/startup.sh"   4 days ago          Up 21 minutes       0.0.0.0:8000-
>8000/tcp   adcm
```

Check the 8000 port availability

After ADCM is installed and started, its web interface should become available on the `8000` port of the deployed container.

For Ubuntu 22.04, first install the packages required for executing the `netstat` command:

```
$ sudo apt-get install net-tools
```

You can check the 8000 port availability using these commands:

- The `netstat` command — to display information about port listening.

```
$ sudo netstat -ntpl | grep 8000
```

The example of the command execution result:

```
tcp6       0      0 :::8000                 :::*                    LISTEN
1514/docker-proxy-c
```

- The `curl` command — to check the URL connection.

```
$ curl http://localhost:8000
```

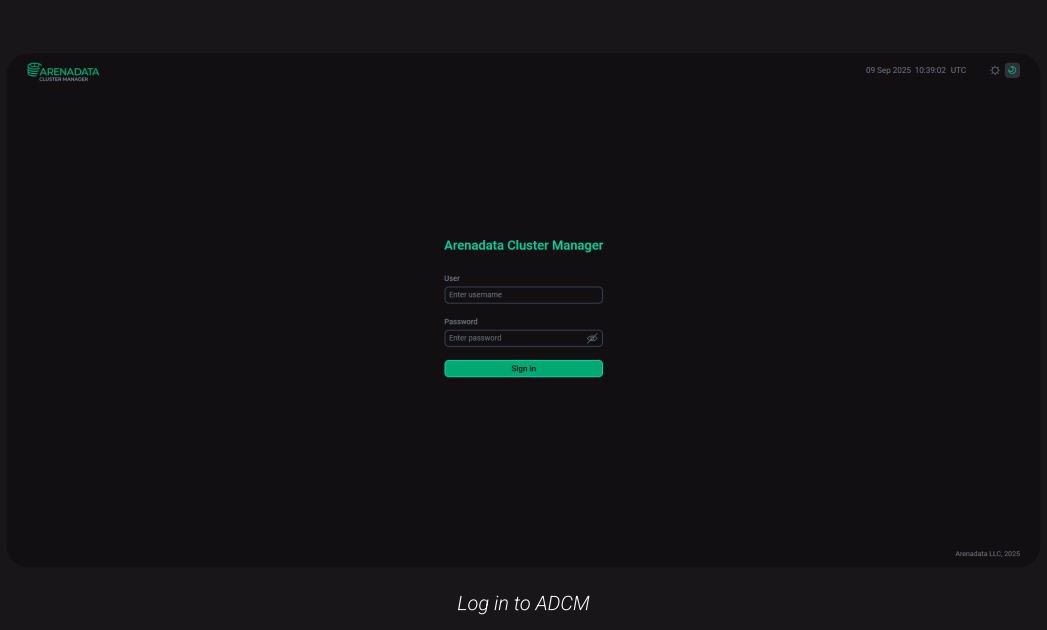
The example of the command execution result:

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Arenadata Cluster Manager</title>
  <base href="/">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="assets/favicon.ico">
<link rel="stylesheet" href="styles.10db6328264e0907c52f.css"></head>
<body>
  <app-root></app-root>
<script src="runtime-es2015.7eb406ed18bf0258cd35.js" type="module"></script><script src="runtime-es5.7eb406ed18bf0258cd35.js"
nomodule defer></script><script src="polyfills-es5.2e224d70daec4412d3c2.js" nomodule defer></script><script src="polyfills-es2015.
37e0553ac06970d6a5b5.js" type="module"></script><script src="main-es2015.39851da0ebf9ed6fec45.js" type="module"></script><script
src="main-es5.39851da0ebf9ed6fec45.js" nomodule defer></script></body>
</html>
```

Check the ADCM web-interface

To log in to ADCM, use the user with administrator rights that is created by default:

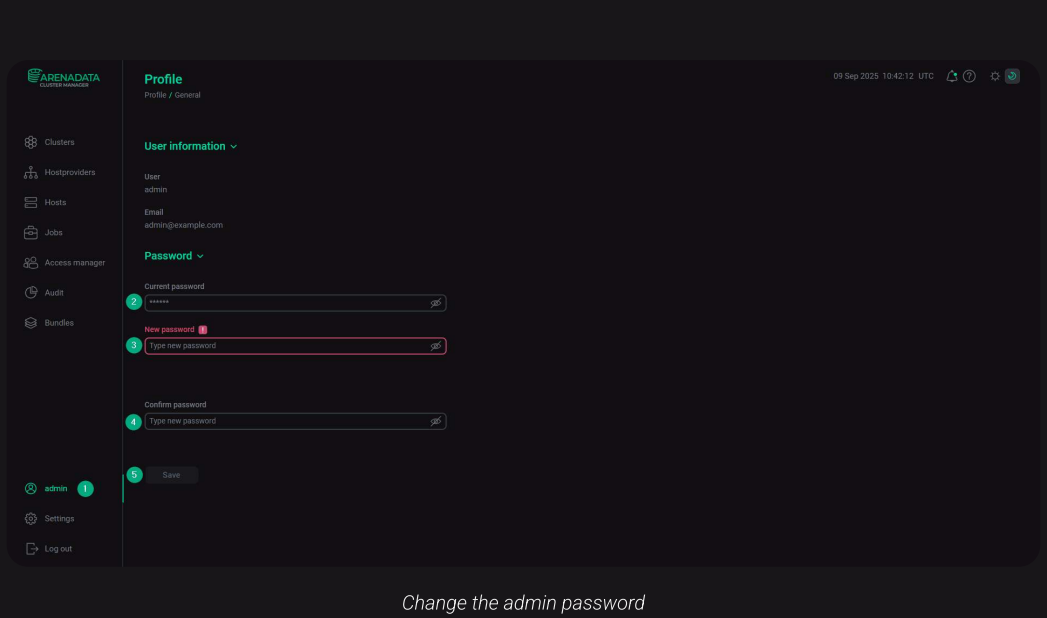
- ***User***: admin
- ***Password***: admin



Log in to ADCM

Later you can change the password if necessary. To do this:

1. Open the user profile in the left navigation menu.
2. Enter the current password in the ***Current password*** field.
3. Fill in the ***New password*** field.
4. Confirm a new password in the ***Confirm password*** field.
5. Click ***Save***.

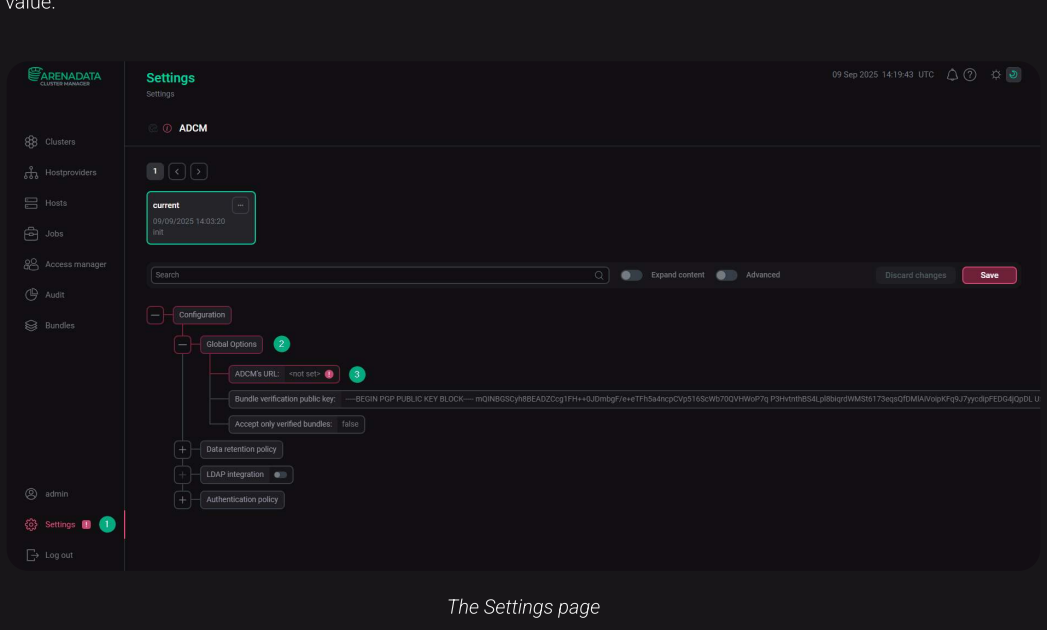


Change the admin password

Set the ADCM URL

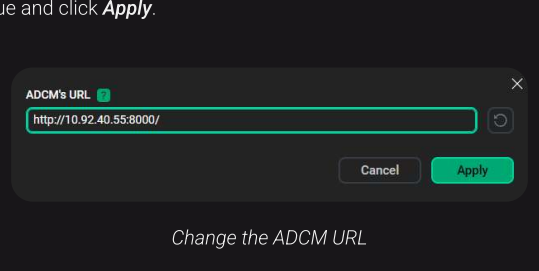
For successful communication between ADCM and product clusters deployed via ADCM, it is important to correctly define the external ADCM address. This address is used by cluster components for sending information about their state to ADCM. To specify the ADCM URL, use the environment variable `DEFAULT_ADCM_URL` when creating the ADCM container. If you did not specify that variable, you should manually install the URL in the ADCM web interface. To do this, follow the steps:

1. Select the ***Settings*** item in the left navigation menu.
2. Expand the ***Global Options*** node in the tree of configuration settings.
3. Click the ***ADCM's URL*** field value.



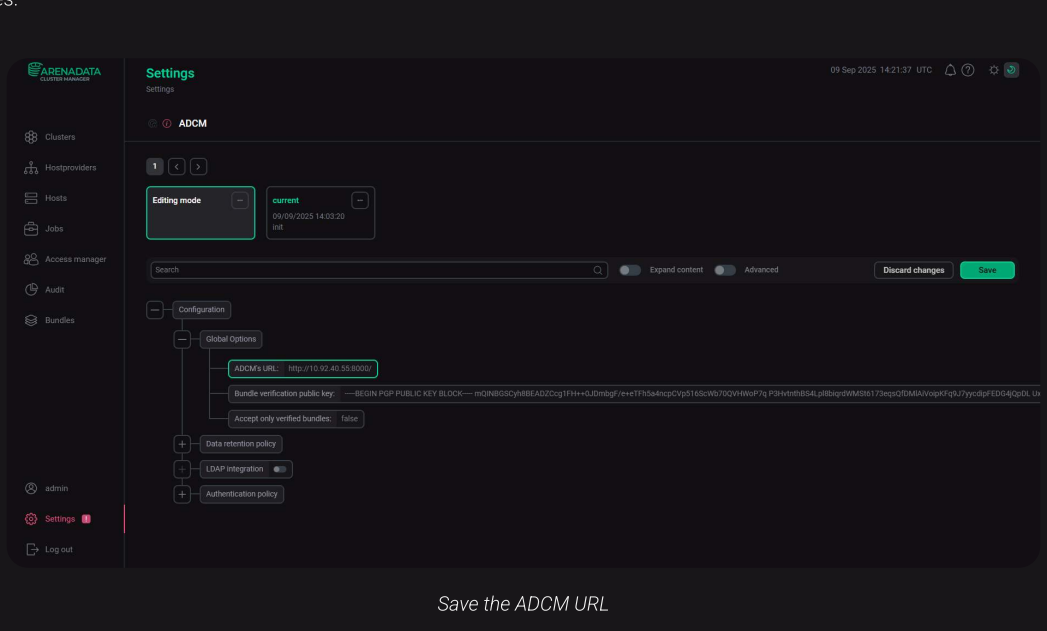
The Settings page

4. In the window that opens, edit the ***ADCM's URL*** field value and click ***Apply***.



Change the ADCM URL

5. Click ***Save*** to apply changes.



Save the ADCM URL

Step 5. Update ADCM

CAUTION

- Before upgrading, it is strongly recommended to back up ADCM data. For more information, see Backup and restore.
- Downgrade of the ADCM version is not supported.
- It is not recommended to update ADCM more than 5 minor versions ahead (see `<minor>` in the versioning format below).
- Before upgrading ADCM to version 2.7.0, migrate data from SQLite to PostgreSQL. Migration can be performed in any ADCM version (up to 2.6.0).

If you need to upgrade the previously installed version of ADCM, do the following:

1. Stop the `adcm` container:

```
$ sudo docker stop adcm
```

2. Remove the `adcm` container:

```
$ sudo docker container rm adcm
```

3. Download the required Docker image from the Arenadata Docker Registry:

```
$ sudo docker pull hub.arenadata.io/adcm/adcm:<version>
```

where `<version>` is a desired ADCM image version in one of the following formats:

- `<major>.<minor>.<patch>` — if you need a specific ADCM patch. For example, `2.0.0`.
- `<major>.<minor>` — if you need the last patch within the selected ADCM version. For example, `2.0`.

IMPORTANT

- `latest` was used for earlier versions of ADCM and is no longer supported starting from version `2.0.0`.
- Before ADCM 2.0.0, the following format was used for versioning: `YYYY.MM.DD.HH`.

4. Create a new container based on the downloaded image:

```
$ sudo docker create --name adcm -p 8000:8000 -v /opt/adcm:/adcm/data -e DB_HOST="<DATABASE_HOSTNAME_OR_IP_ADDRESS>" -e DB_PORT="
<DATABASE_TCP_PORT>" -e DB_USER="<DATABASE_USERNAME>" -e DB_NAME="<DATABASE_NAME>" -e DB_PASS="<DATABASE_USER_PASSWORD>" -e
DB_OPTIONS="<DATABASE_OPTIONS>" hub.arenadata.io/adcm/adcm:<version>
```

NOTE

For more information on the environment variables (`DB_HOST`, `DB_PORT`, `DB_USER`, `DB_NAME`, `DB_PASS`, and `DB_OPTIONS`), see the Installation article.

5. Start the new `adcm` container:

```
$ sudo docker start adcm
```

Prepare hosts

To Table of Contents

To manage a cluster and its nodes via ADCM, it is necessary to prepare hosts for a cluster and link these hosts to ADCM. This is the function of ADCM **hostproviders** — special plugins for ADCM, which encapsulate the complex mechanism of interaction between ADCM and the hosts controlled by it.

ADCM supports several types of hostproviders. For more information on how to configure hostproviders and use them to create hosts, refer to the following documentation:

- SSH
- Yandex Cloud
- VCloud (VMware)
- Cloud.ru Advanced
- CROC Cloud
- ADQM Cloud Bundle

CAUTION

If you are going to use Kerberos authentication in ADQM, you should specify FQDNs when creating ADQM hosts.

Create a cluster

Daria Barysheva, Elena Dvoryadkina

Contents

To Table of Contents

Step 1. Download a cluster bundle
Step 2. Upload a cluster bundle to ADCM
Step 3. Create a cluster based on the uploaded bundle

In ADCM a **cluster** means a set of services running on a set of hosts.

Clusters distributions for ADCM are made in the form of so-called bundles. Generally a bundle is a regular archive that includes a description and software logic of cluster deployment and exploitation.

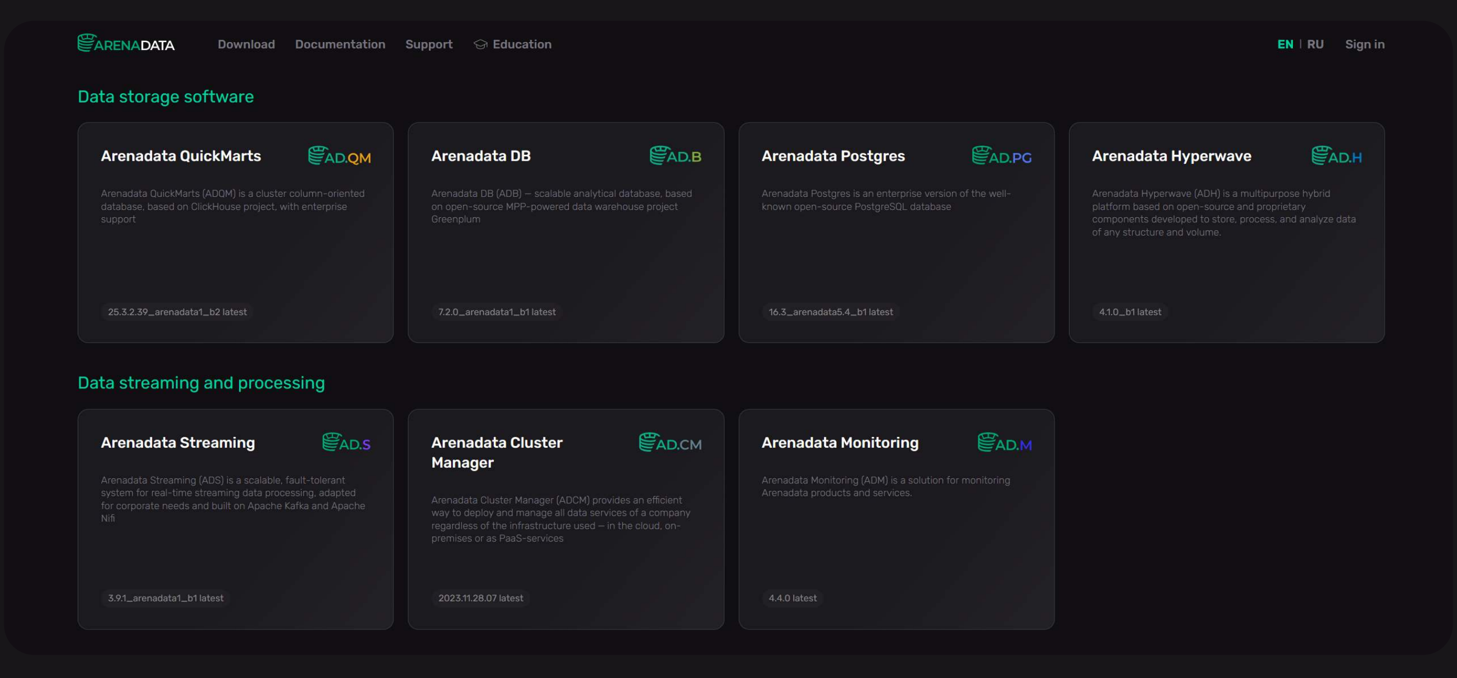
The steps for adding a cluster to ADCM are listed below.

NOTE

If you use the Enterprise Edition, skip the step 1, get a bundle from the Arenadata support team, and move to the step 2.

Step 1. Download a cluster bundle

1. Go to https://network.arenadata.io/ and choose ***Arenadata QuickMarts***.


ARENADATA
Download
Documentation
Support
Education
EN | RU
Sign in

Data storage software

Arenadata QuickMarts — Arenadata QuickMarts (ADQM) is a cluster column-oriented database, based on ClickHouse project, with enterprise support — 25.3.2.39_arenadata1_b2 latest

Arenadata DB — Arenadata DB (ADB) — scalable analytical database, based on open-source MPP-powered data warehouse project Greenplum — 7.2.0_arenadata1_b1 latest

Arenadata Postgres — Arenadata Postgres is an enterprise version of the well-known open-source PostgreSQL database — 16.3_arenadata5.4_b1 latest

Arenadata Hyperwave — Arenadata Hyperwave (ADH) is a multipurpose hybrid platform based on open-source and proprietary components developed to store, process, and analyze data of any structure and volume. — 4.1.0_b1 latest

Data streaming and processing

Arenadata Streaming — Arenadata Streaming (ADS) is a scalable, fault-tolerant system for real-time streaming data processing, adapted for corporate needs and built on Apache Kafka and Apache Nifi — 3.9.1_arenadata1_b1 latest

Arenadata Cluster Manager — Arenadata Cluster Manager (ADCM) provides an efficient way to deploy and manage all data services of a company regardless of the infrastructure used – in the cloud, on-premises or as PaaS-services — 2023.11.28.07 latest

Arenadata Monitoring — Arenadata Monitoring (ADM) is a solution for monitoring Arenadata products and services. — 4.4.0 latest

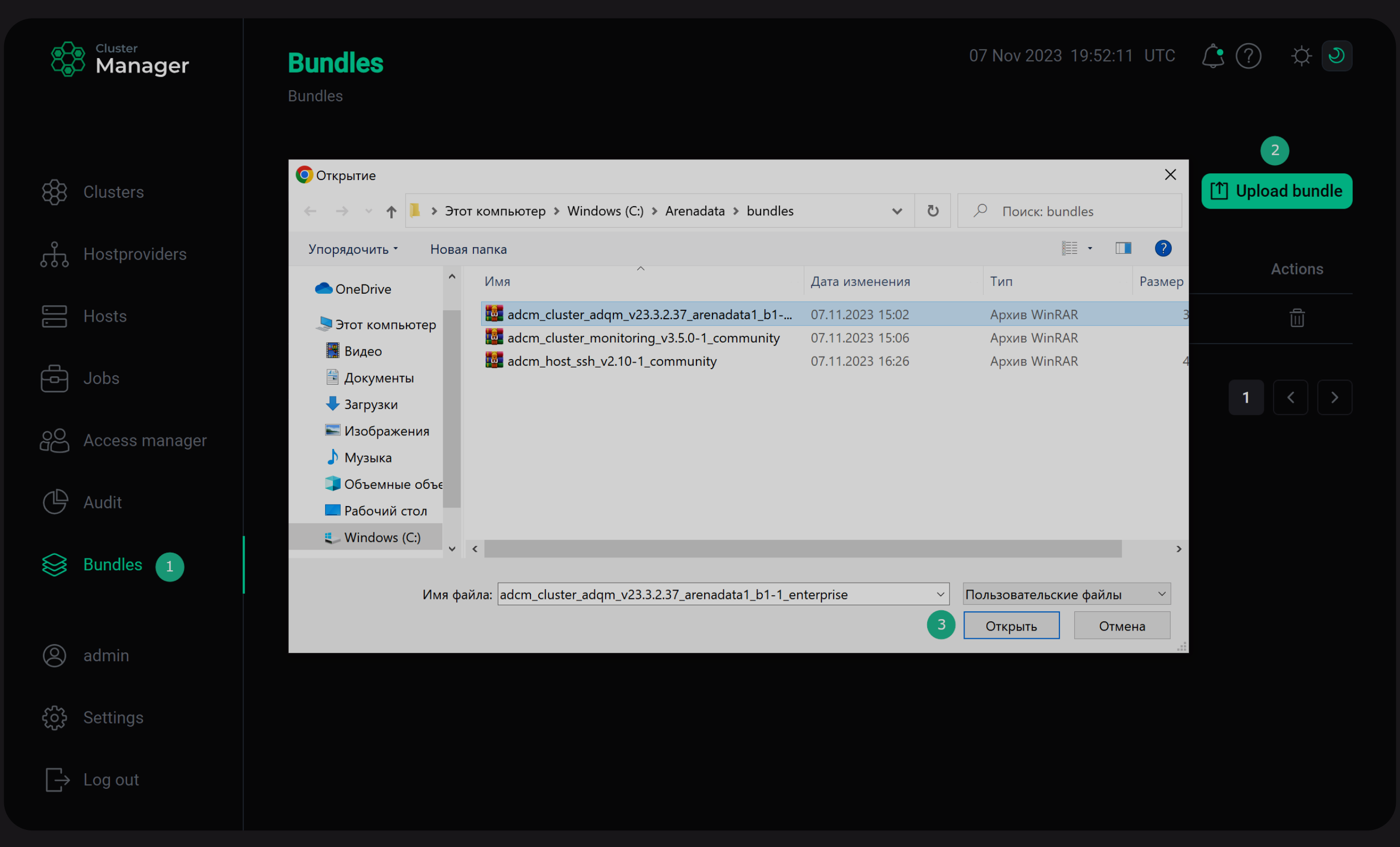
Switch to the bundle download page

2. On the next page, download a bundle for the required version of ADQM.

Step 2. Upload a cluster bundle to ADCM

To upload a bundle to ADCM, follow the steps:

1. Select the ***Bundles*** item in the left navigation menu and click ***Upload bundle***.
2. Select a bundle in the Open File Dialog.


Cluster Manager
Bundles
Bundles
07 Nov 2023 19:52:11 UTC
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
1
admin
Settings
Log out
2
Upload bundle
Actions
Открытие
Этот компьютер > Windows (C:) > Arenadata > bundles
Поиск: bundles
Упорядочить
Новая папка
OneDrive
Этот компьютер
Видео
Документы
Загрузки
Изображения
Музыка
Объемные объе
Рабочий стол
Windows (C:)
3
Открыть
Отмена

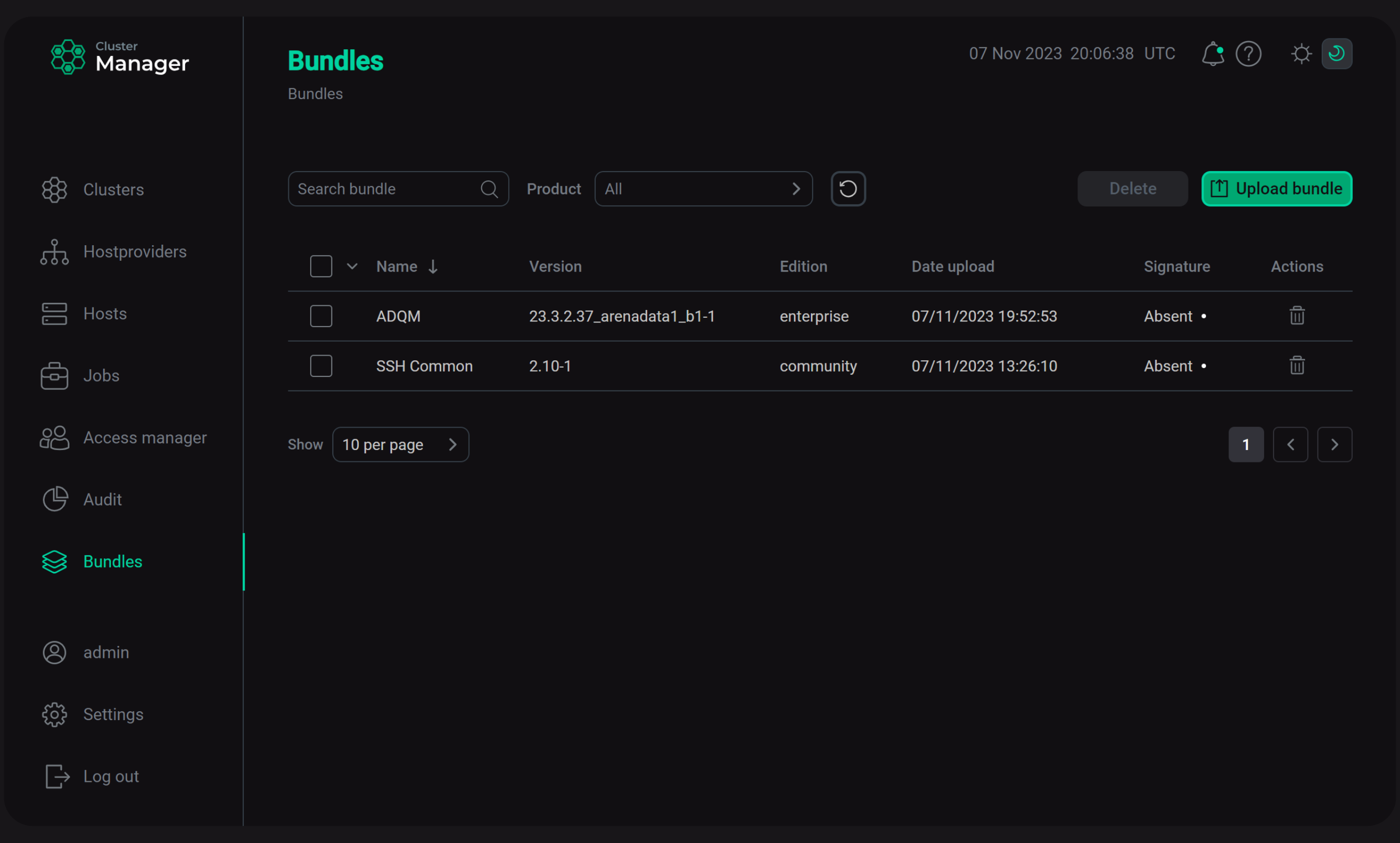
Имя	Дата изменения	Тип
adcm_cluster_adqm_v23.3.2.37_arenadata1_b1-...	07.11.2023 15:02	Архив WinRAR
adcm_cluster_monitoring_v3.5.0-1_community	07.11.2023 15:06	Архив WinRAR
adcm_host_ssh_v2.10-1_community	07.11.2023 16:26	Архив WinRAR

Имя файла: adcm_cluster_adqm_v23.3.2.37_arenadata1_b1-1_enterprise

Пользовательские файлы

Upload a bundle

3. As a result of the performed actions, a bundle is displayed on the ***Bundles*** page.


Cluster Manager
Bundles
Bundles
07 Nov 2023 20:06:38 UTC
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Search bundle
Delete
Upload bundle
Show
10 per page
1

Product: All

Name	Version	Edition	Date upload	Signature	Actions
ADQM	23.3.2.37_arenadata1_b1-1	enterprise	07/11/2023 19:52:53	Absent •	
SSH Common	2.10-1	community	07/11/2023 13:26:10	Absent •	

The result of successful uploading a bundle

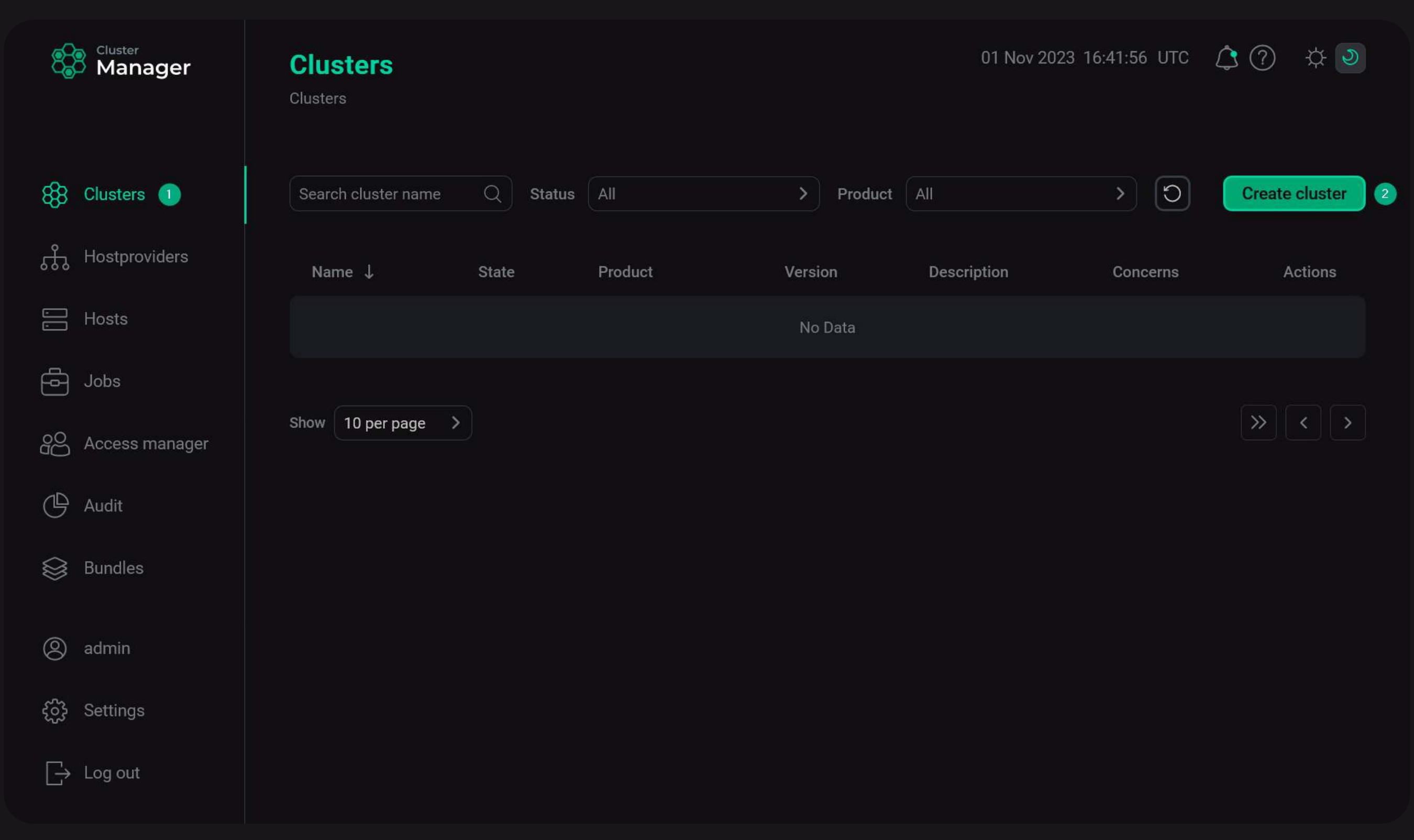
Step 3. Create a cluster based on the uploaded bundle

IMPORTANT

When creating a cluster via the ADCM web interface, you generate a new ADCM cluster instance. It means only adding the cluster to ADCM — at this stage no services are deployed.

The steps for cluster creation are given below:

1. Select the ***Clusters*** item in the left navigation menu and click ***Create cluster***.


Cluster Manager
Clusters
Clusters
01 Nov 2023 16:41:56 UTC
Clusters
1
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Search cluster name
Create cluster
2
Show
10 per page

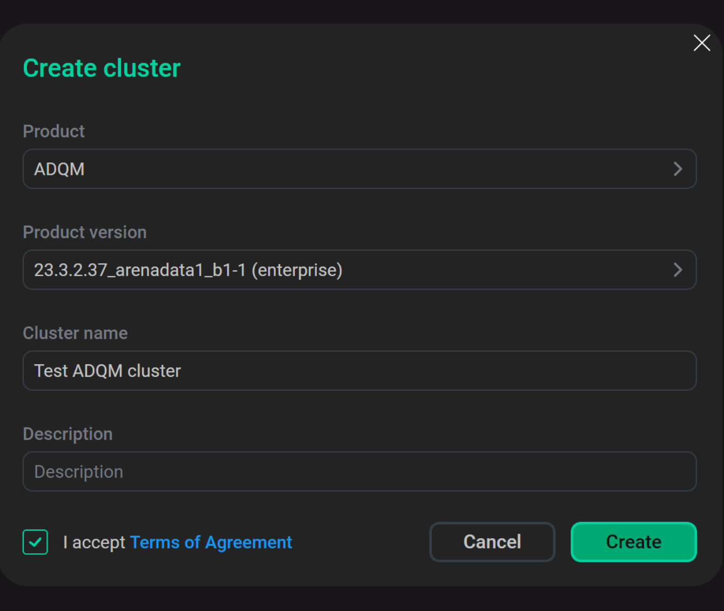
Status: All

Product: All

Name	State	Product	Version	Description	Concerns	Actions
No Data						

Switch to cluster creation

2. In the opened dialog, describe a new cluster instance:
   - Select an uploaded cluster bundle in the ***Product*** field.
   - Select a bundle version in the ***Product version*** field. Several versions become available in the case of different versions of the same bundle being uploaded.
   - Enter a cluster name in the ***Cluster name*** field.
   - Enter a cluster description in the ***Description*** field if necessary.
   - Set the ***I accept Terms of Agreement*** flag to sign the license agreement. This flag is displayed only for Enterprise bundles. To read the agreement text, refer to the ***Terms of Agreement*** link. You can also sign the license agreement on the ***Bundles*** page by clicking the desired product bundle.
   - Click ***Create***.


Create cluster
Description
Cancel
Create

Product: ADQM

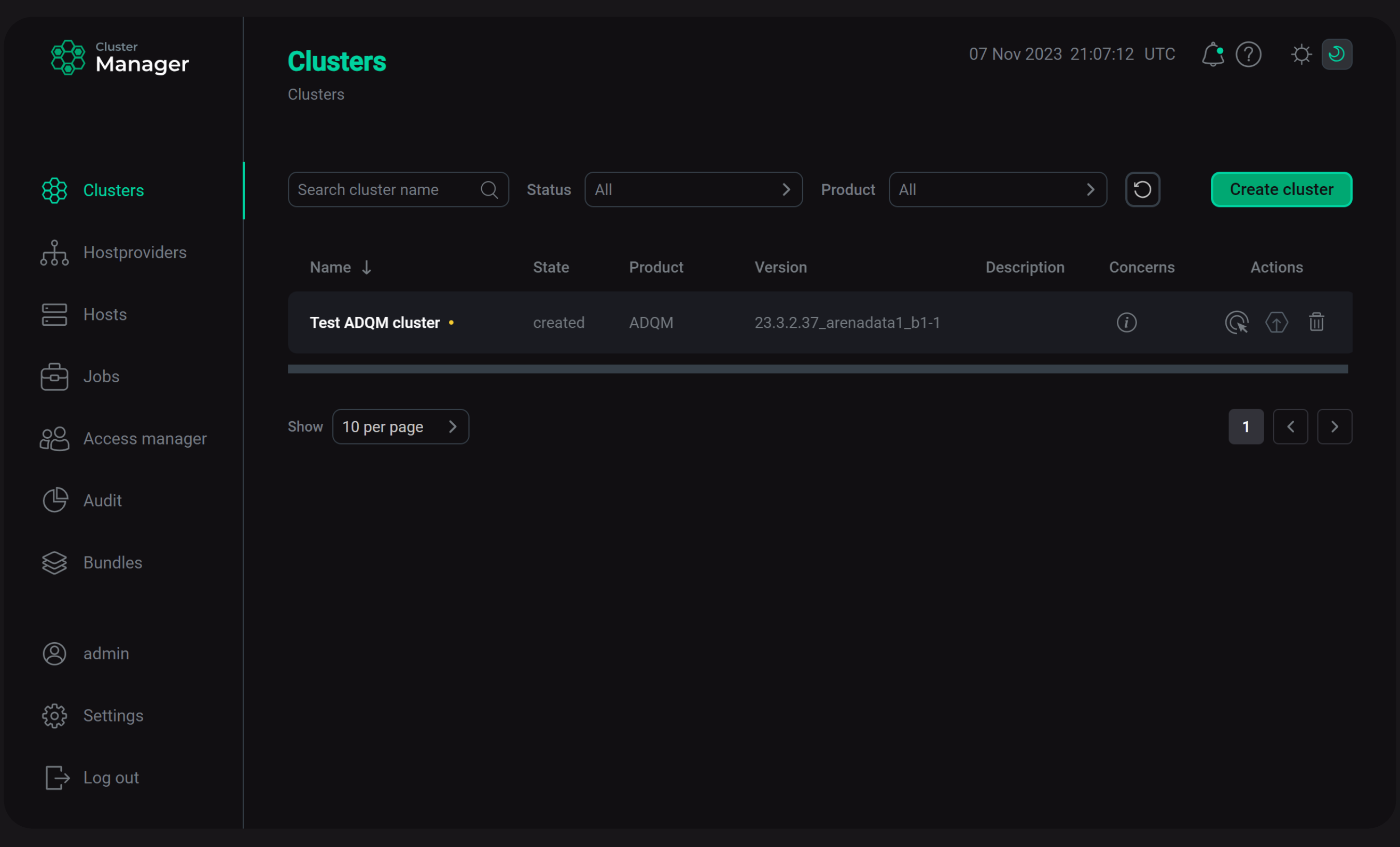
Product version: 23.3.2.37_arenadata1_b1-1 (enterprise)

Cluster name: Test ADQM cluster

☑ I accept Terms of Agreement

Create a cluster

3. As a result, the created cluster instance is displayed on the ***Clusters*** page.


Cluster Manager
Clusters
Clusters
07 Nov 2023 21:07:12 UTC
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Search cluster name
Create cluster
Show
10 per page
1

Status: All

Product: All

Name	State	Product	Version	Description	Concerns	Actions
Test ADQM cluster •	created	ADQM	23.3.2.37_arenadata1_b1-1			

The result of successful cluster creation

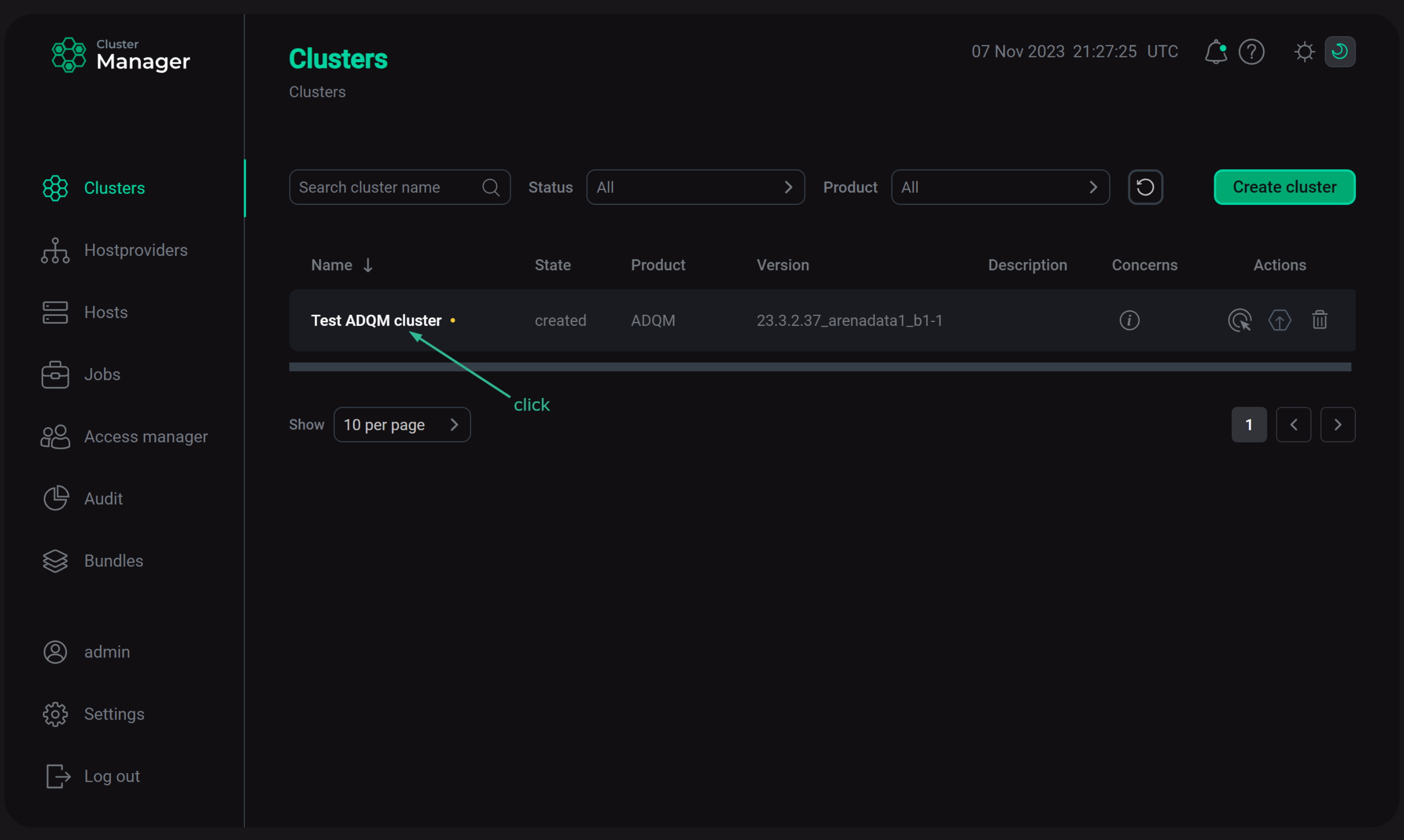
Add services

Daria Barysheva, Elena Dvoryadkina

To Table of Contents

In ADCM a **service** means a software that performs some function. Examples of services for ADQM clusters: ADQMDB, Monitoring Clients, Zookeeper, etc. The steps for adding services to a cluster are listed below:

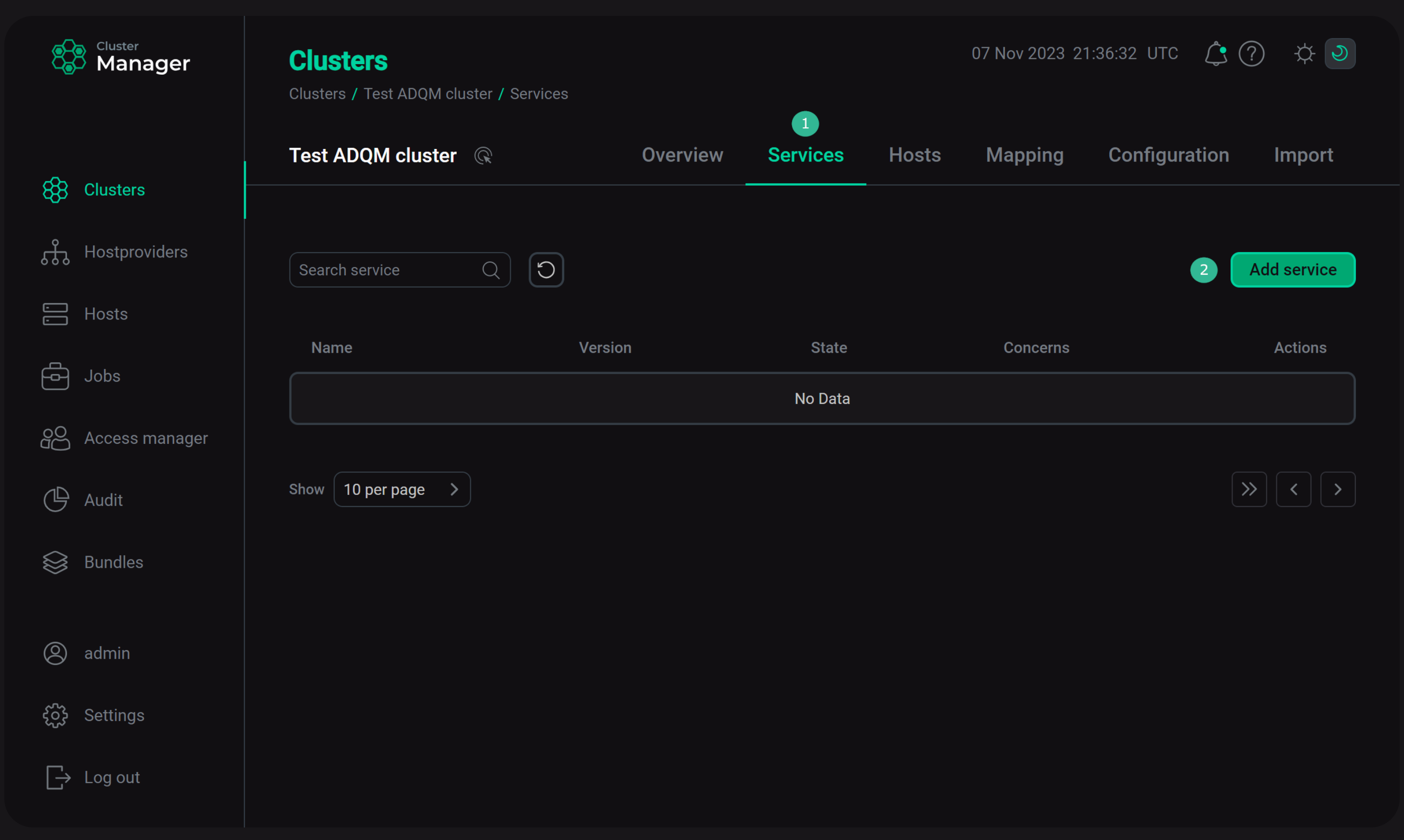
1. Select a cluster on the ***Clusters*** page. To do this, click a cluster name in the ***Name*** column.


Cluster
Manager
Clusters
Clusters
07 Nov 2023 21:27:25 UTC
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Search cluster name
Status
All
Product
All
Create cluster
click
Show
10 per page
1

Name	State	Product	Version	Description	Concerns	Actions
Test ADQM cluster	created	ADQM	23.3.2.37_arenadata1_b1-1			

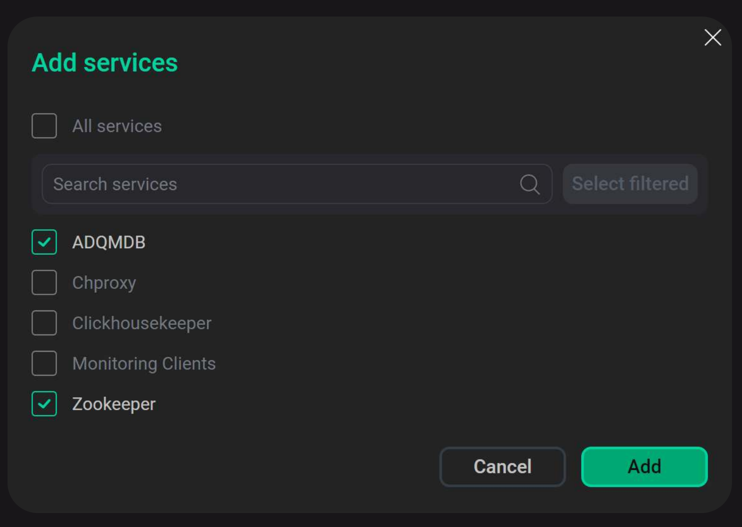
Select a cluster

2. Open the ***Services*** tab on the cluster page and click ***Add service***.


Cluster
Manager
Clusters
Clusters / Test ADQM cluster / Services
07 Nov 2023 21:36:32 UTC
Test ADQM cluster
Overview
1
Services
Hosts
Mapping
Configuration
Import
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Search service
2
Add service
Name
Version
State
Concerns
Actions
No Data
Show
10 per page

Switch to adding services

3. In the opened dialog, select services that should be added to the cluster and click ***Add***.


Add services
Search services
Select filtered
Cancel
Add

☐ All services

☑ ADQMDB
☐ Chproxy
☐ Clickhousekeeper
☐ Monitoring Clients
☑ Zookeeper

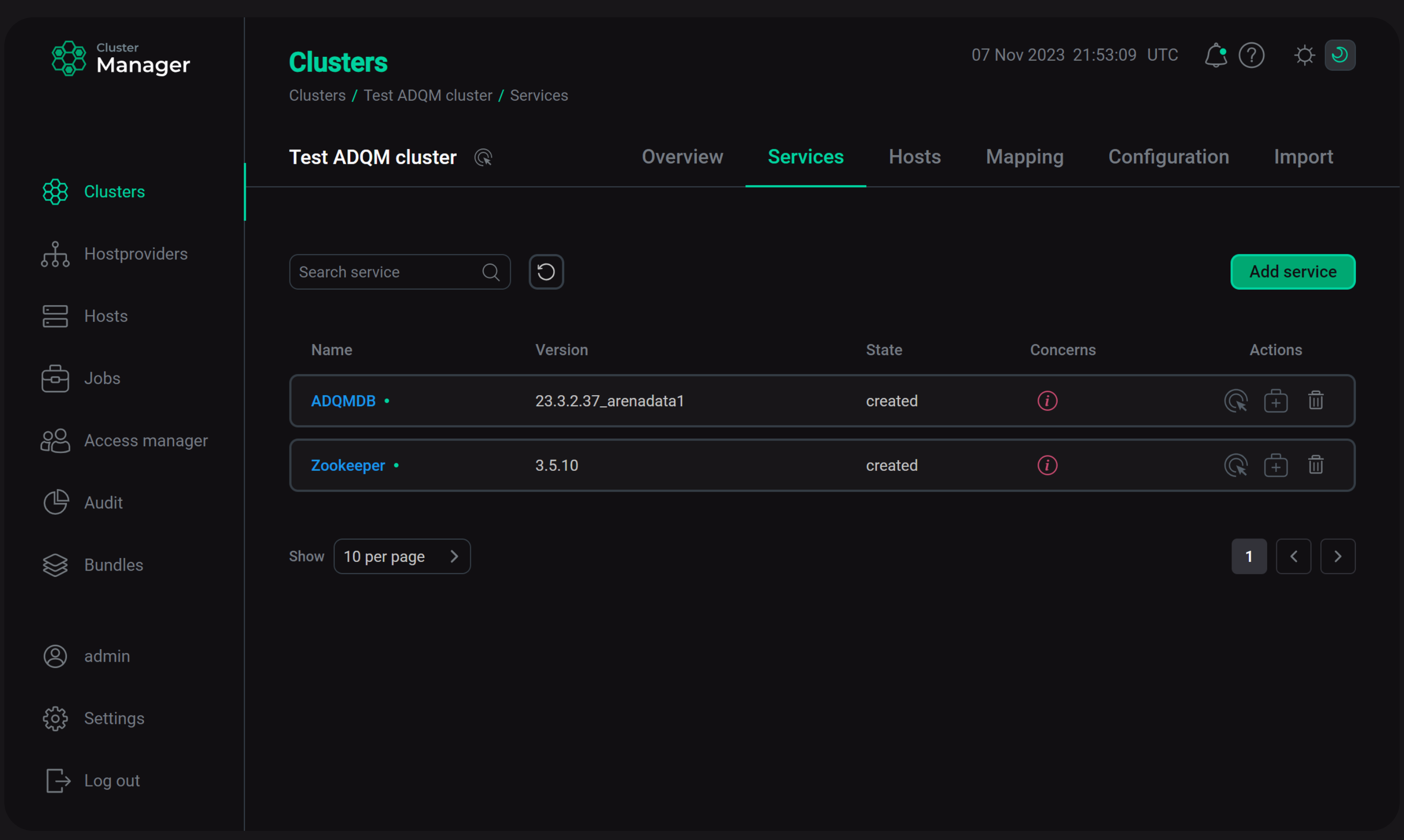
Select services

The brief description of available services is listed below.

Services that can be added to the ADQM cluster

Service	Mandatory	Purpose	Availability
ADQMDB	Yes	ClickHouse is a column-oriented database management system (DBMS) for online analytical processing of queries (OLAP)	Community/Enterprise
Zookeeper	No	Centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services	Community/Enterprise
Clickhousekeeper	No	Coordination system for data replication and distributed DDL queries execution (an alternative to ZooKeeper)	Community/Enterprise
Monitoring Clients	No	Agents that send information on hosts to the monitoring system, that is based on such services as Diamond, Graphite, and Grafana. The service should be added after the monitoring cluster is installed and configured, if monitoring of the current cluster is planned	Community/Enterprise
Monitoring	No	Service for collecting and storing ADQM cluster monitoring metrics based on the Prometheus monitoring system and Grafana service	Community/Enterprise
Chproxy	No	HTTP proxy and load balancer for ClickHouse	Enterprise

4. As a result, the added services are displayed on the ***Services*** tab.


Cluster
Manager
Clusters
Clusters / Test ADQM cluster / Services
07 Nov 2023 21:53:09 UTC
Test ADQM cluster
Overview
Services
Hosts
Mapping
Configuration
Import
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Search service
Add service
Show
10 per page
1

Name	Version	State	Concerns	Actions
ADQMDB	23.3.2.37_arenadata1	created		
Zookeeper	3.5.10	created		

The result of successful adding services to a cluster

NOTE

You can also add services later. The process of adding new services to already running cluster does not differ from installing a service from scratch.

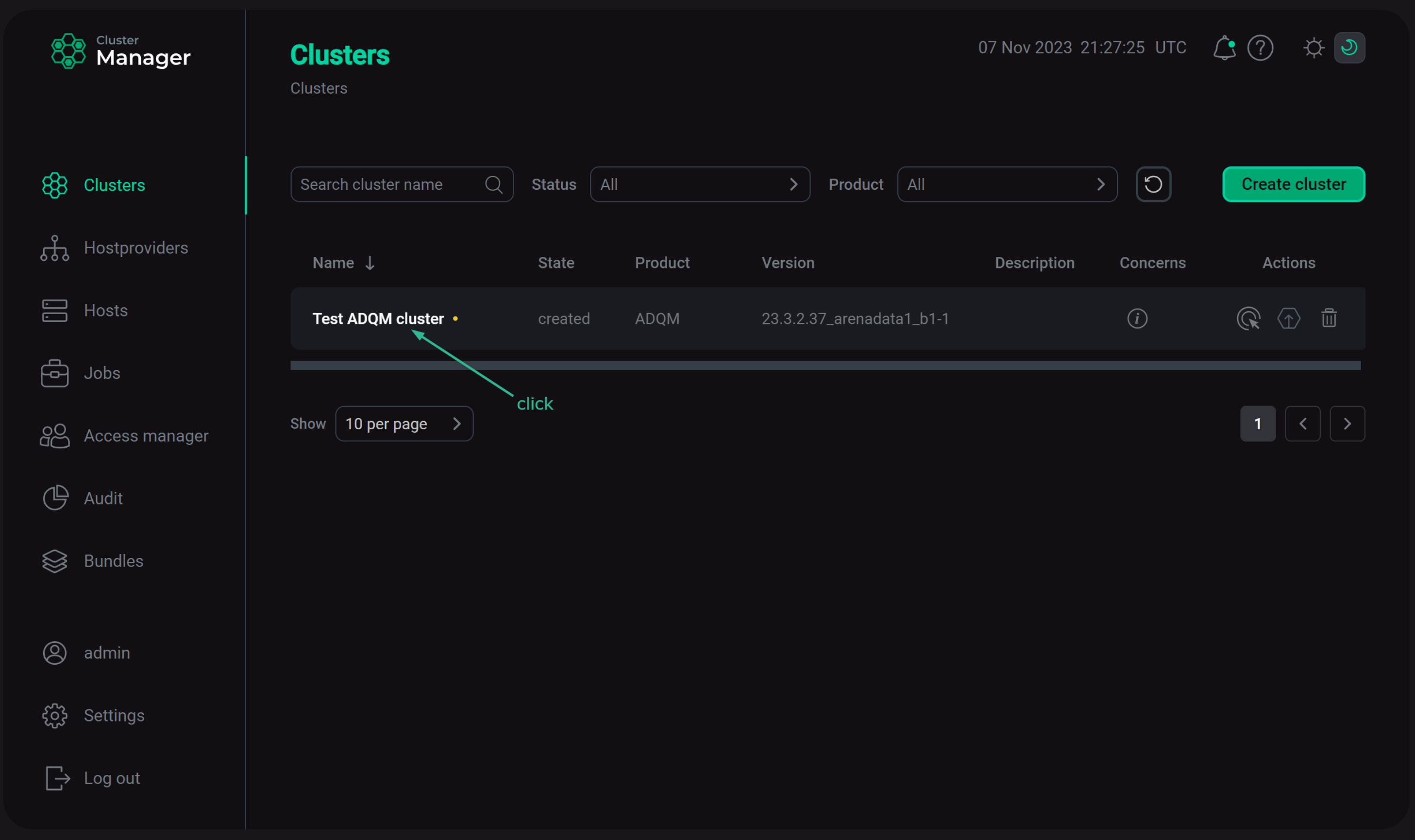
Add hosts to a cluster

Daria Barysheva, Elena Dvoryadkina

To Table of Contents

The steps for adding hosts to a cluster are listed below:

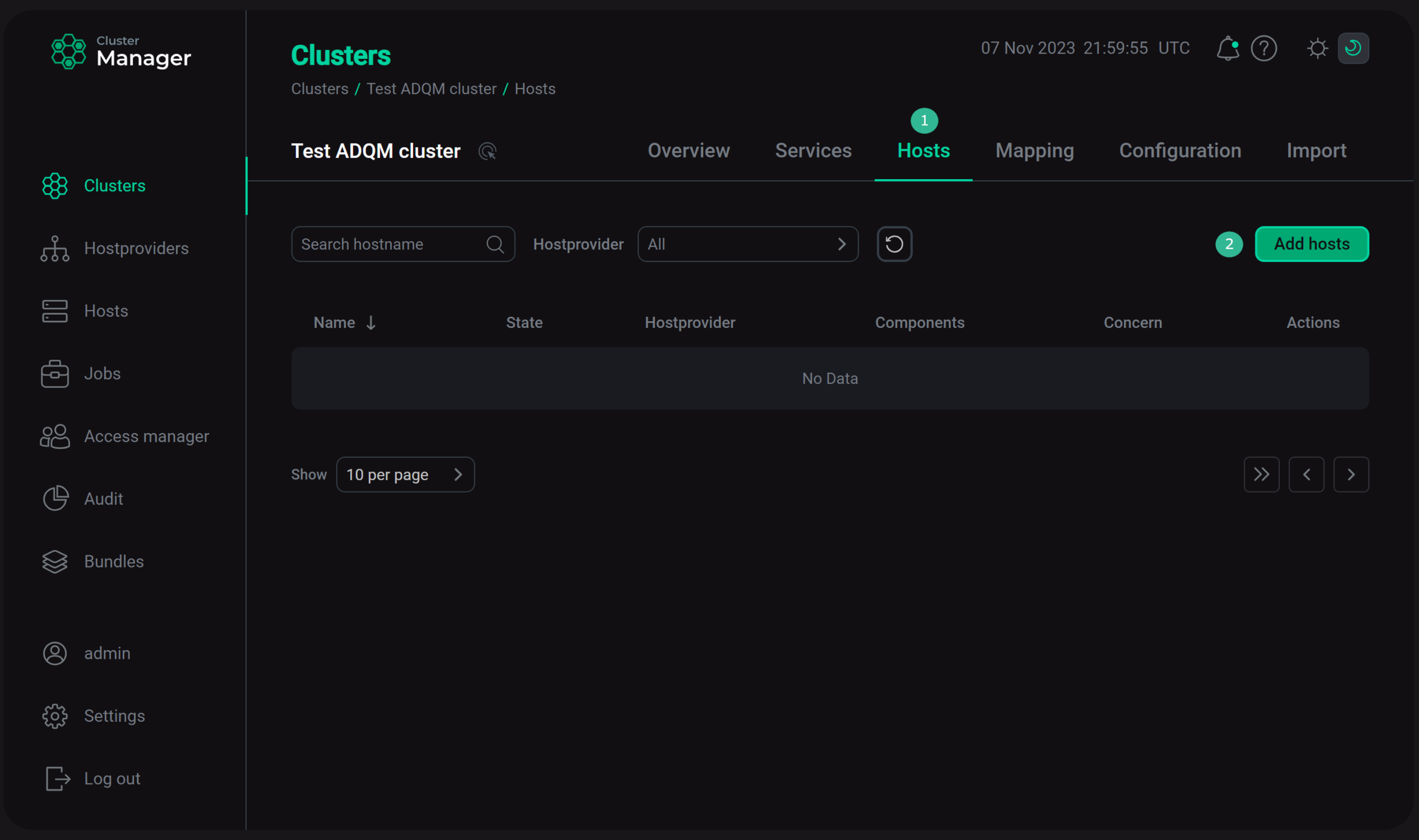
1. Select a cluster on the ***Clusters*** page. To do this, click a cluster name in the ***Name*** column.


Cluster Manager
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Clusters
Clusters
07 Nov 2023 21:27:25 UTC
Search cluster name
Status
All
Product
All
Create cluster
click
Show
10 per page
1

Name	State	Product	Version	Description	Concerns	Actions
Test ADQM cluster	created	ADQM	23.3.2.37_arenadata1_b1-1			

Select a cluster

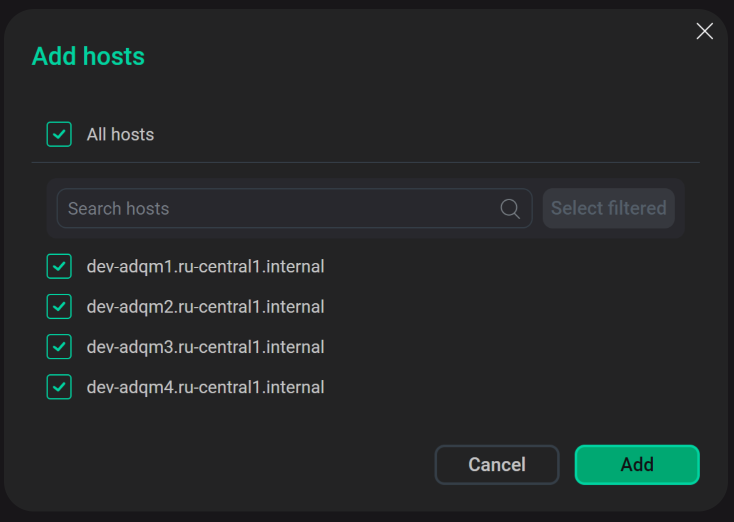
2. Open the ***Hosts*** tab on the cluster page and click ***Add hosts***.


Cluster Manager
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Clusters
Clusters / Test ADQM cluster / Hosts
07 Nov 2023 21:59:55 UTC
Test ADQM cluster
Overview
Services
1
Hosts
Mapping
Configuration
Import
Search hostname
Hostprovider
All
2
Add hosts
Show
10 per page

Name	State	Hostprovider	Components	Concern	Actions
No Data					

Switch to adding hosts

3. In the opened dialog, select hosts that should be added to the cluster and click ***Add***.

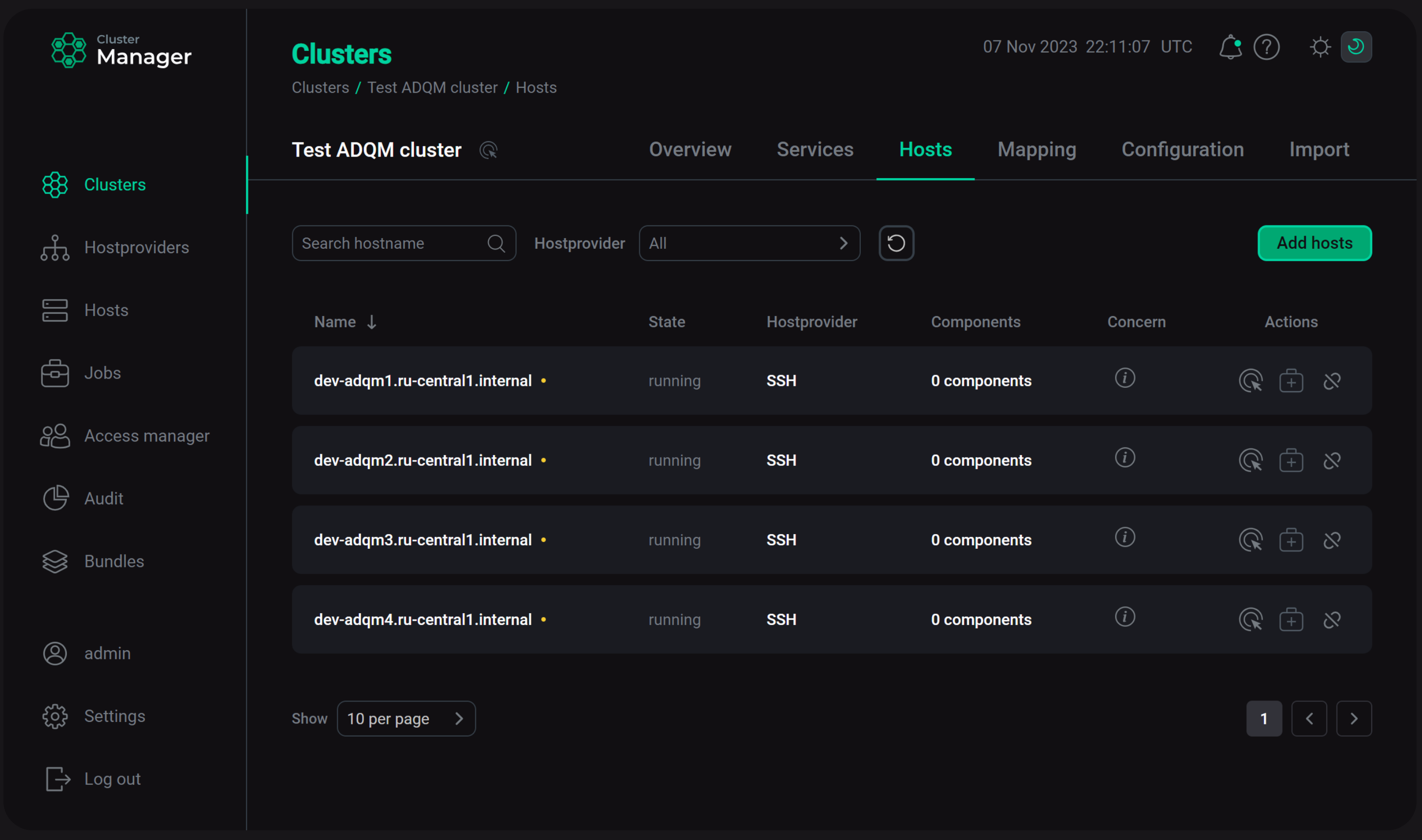

Add hosts
Search hosts
Select filtered
Cancel
Add

☑ All hosts

☑ dev-adqm1.ru-central1.internal
☑ dev-adqm2.ru-central1.internal
☑ dev-adqm3.ru-central1.internal
☑ dev-adqm4.ru-central1.internal

Select hosts

4. As a result, the added hosts are displayed on the ***Hosts*** tab.


Cluster Manager
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Clusters
Clusters / Test ADQM cluster / Hosts
07 Nov 2023 22:11:07 UTC
Test ADQM cluster
Overview
Services
Hosts
Mapping
Configuration
Import
Search hostname
Hostprovider
All
Add hosts
Show
10 per page
1

Name	State	Hostprovider	Components	Concern	Actions
dev-adqm1.ru-central1.internal	running	SSH	0 components		
dev-adqm2.ru-central1.internal	running	SSH	0 components		
dev-adqm3.ru-central1.internal	running	SSH	0 components		
dev-adqm4.ru-central1.internal	running	SSH	0 components		

The result of successful adding hosts to a cluster

Add components

Daria Barysheva, Elena Dvoryadkina

To Table of Contents

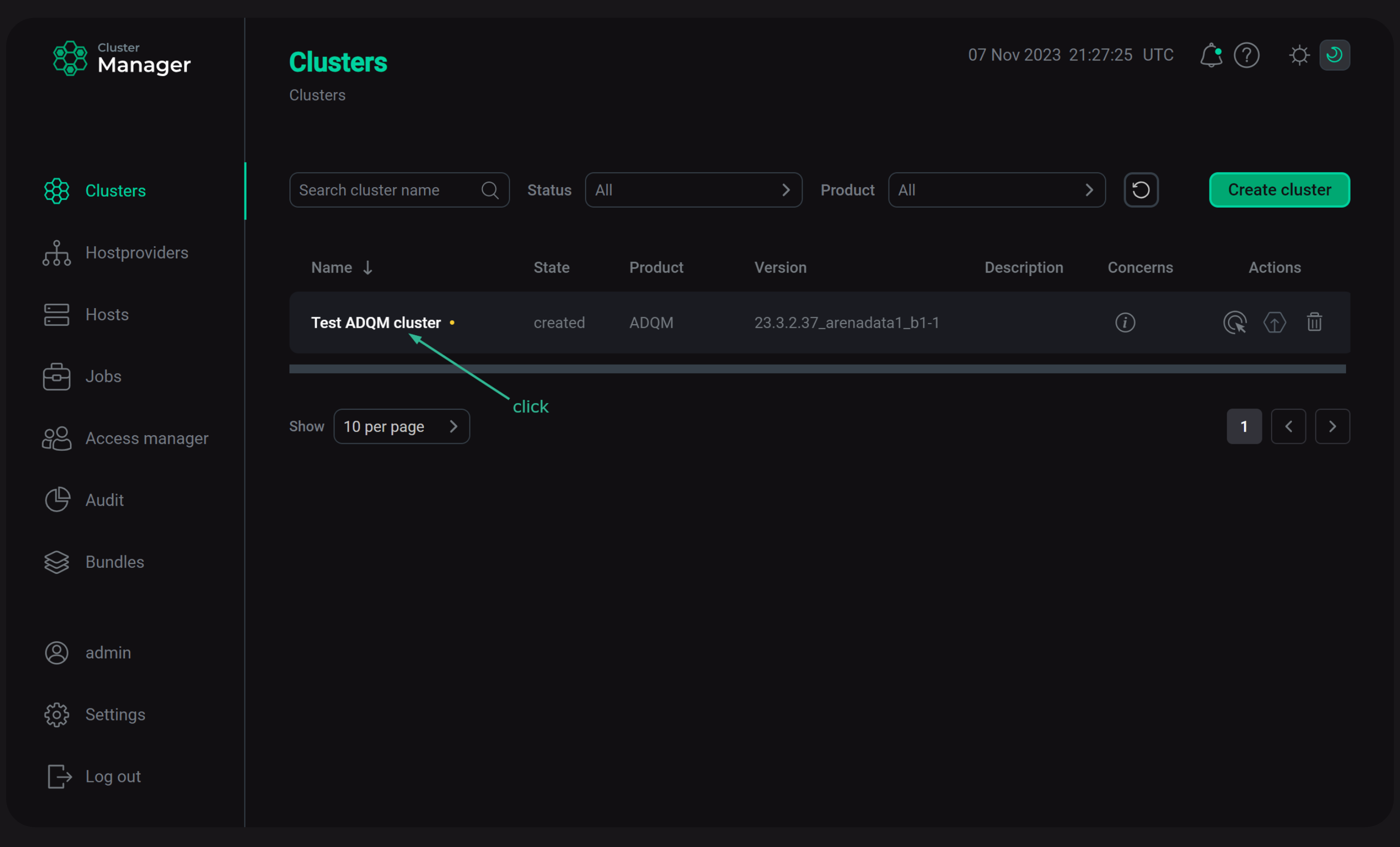
In ADCM a **component** means a part of a service that should be deployed on one or several cluster hosts.

IMPORTANT

Initially, there are no components on any of cluster hosts. You should map components to hosts manually.

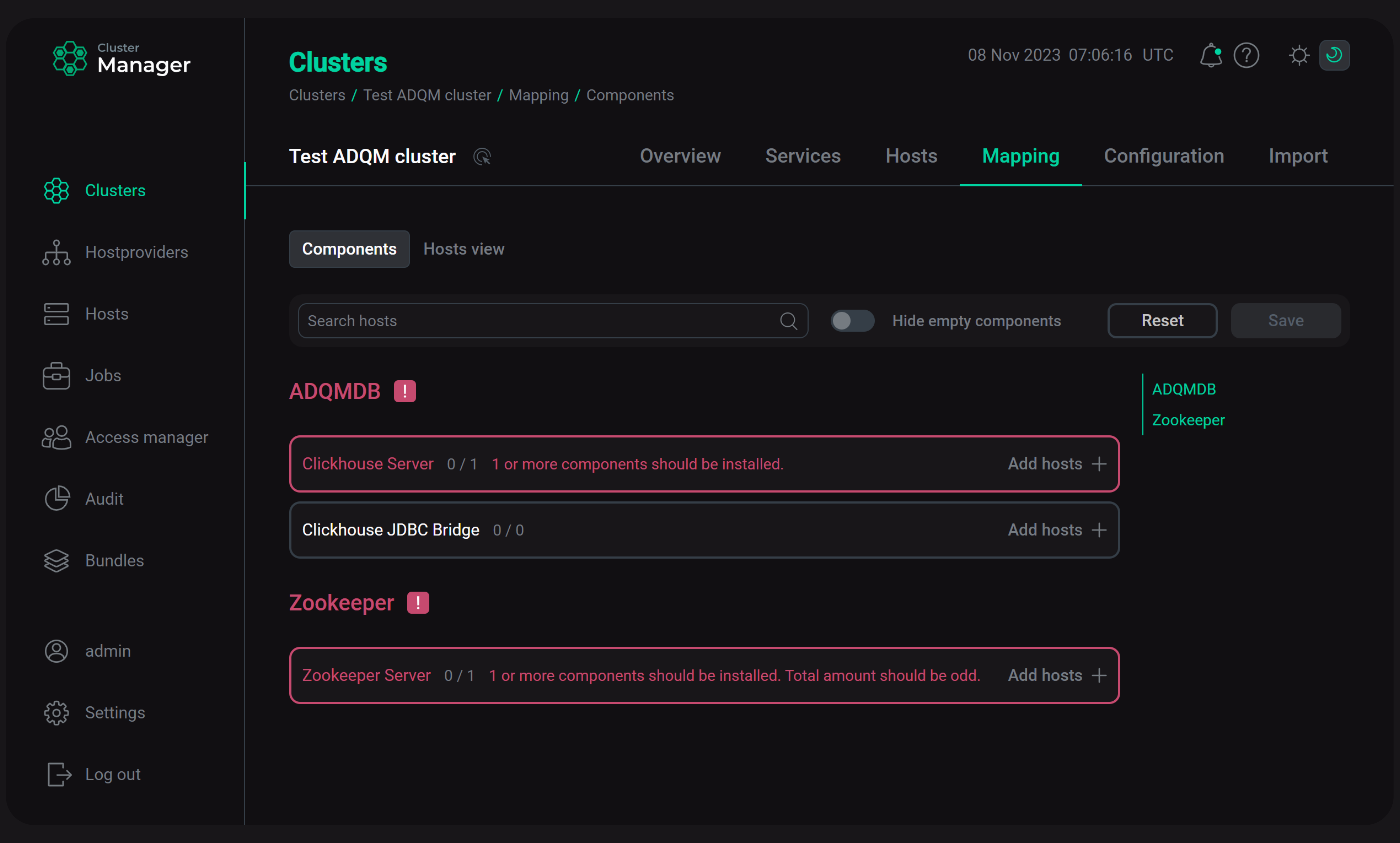
The steps for adding components to hosts are listed below:

1. Select a cluster on the ***Clusters*** page. To do this, click a cluster name in the ***Name*** column.



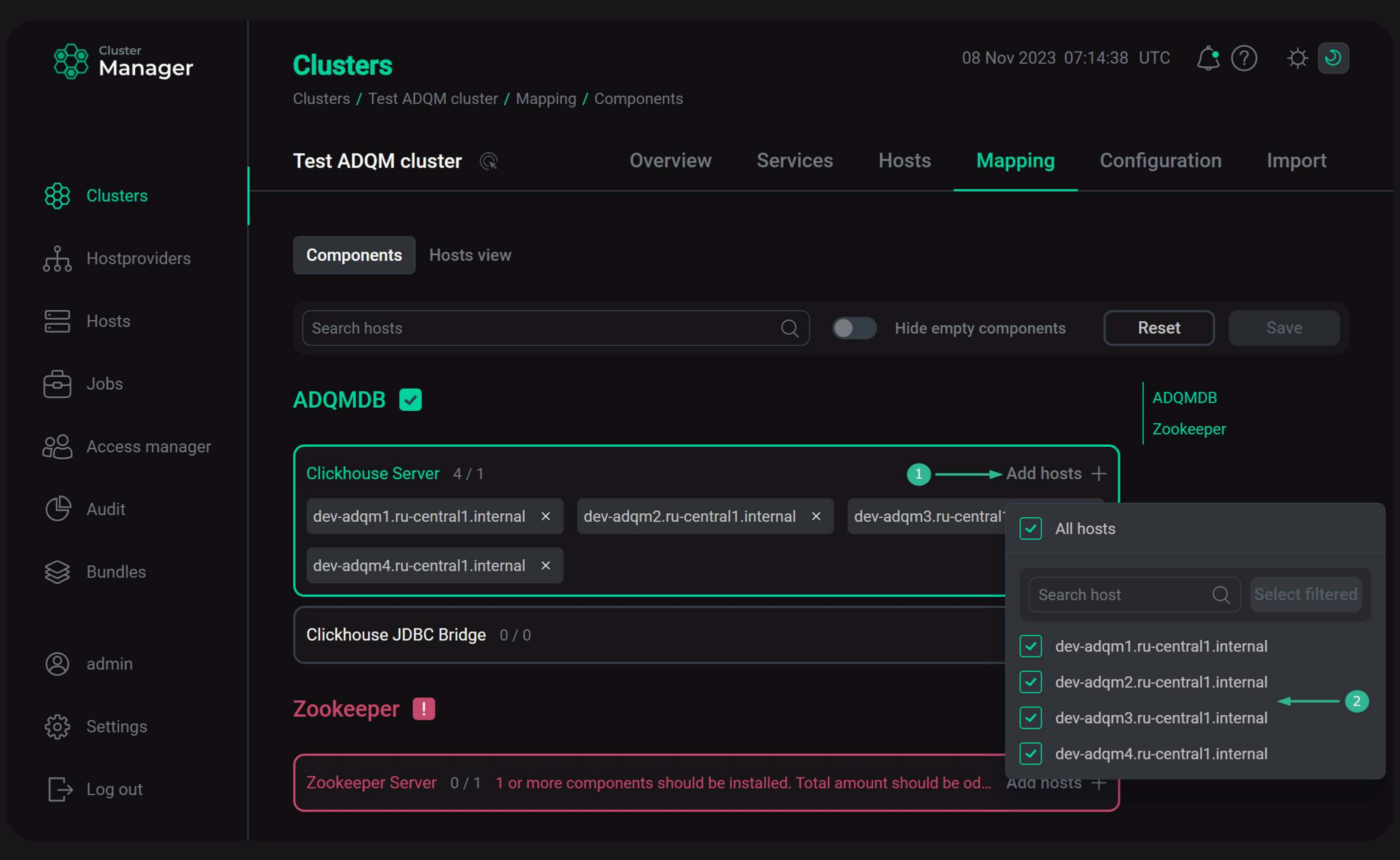
Select a cluster

2. Open the ***Mapping*** tab on the cluster page.



Switch to mapping service components

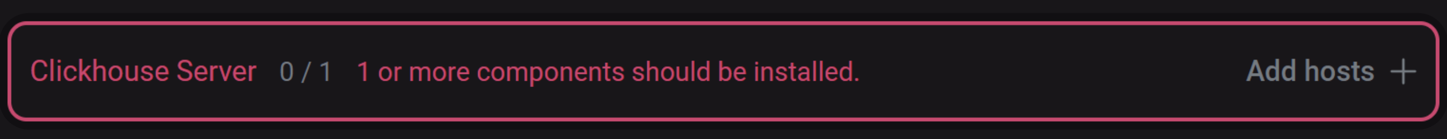
3. In the window that opens, map components to hosts. To assign hosts for each component, click ***Add hosts*** and select hosts in the pop-up window.



Select hosts for a component

Mandatory components

Each service can have mandatory and optional components. Mandatory components are highlighted in red. The ADCM UI also displays how many mandatory components should be installed.


Clickhouse Server 0 / 1 1 or more components should be installed.
Add hosts +

Mandatory components

CAUTION

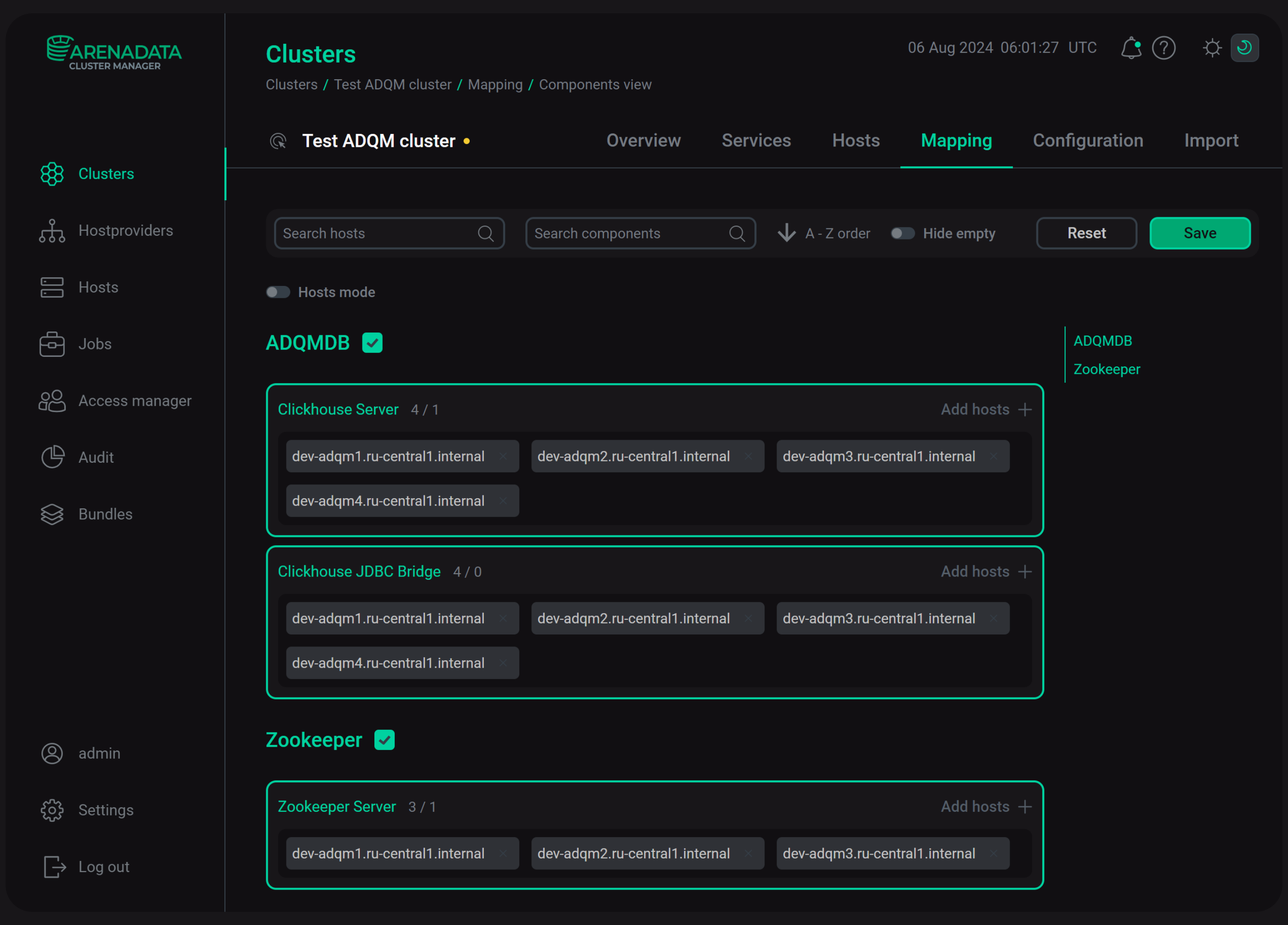
Without assigning a necessary number of hosts to mandatory components, the component mapping cannot be saved.

The typical scheme of components distribution for ADQM clusters is described in the table below.

ADQM components distribution

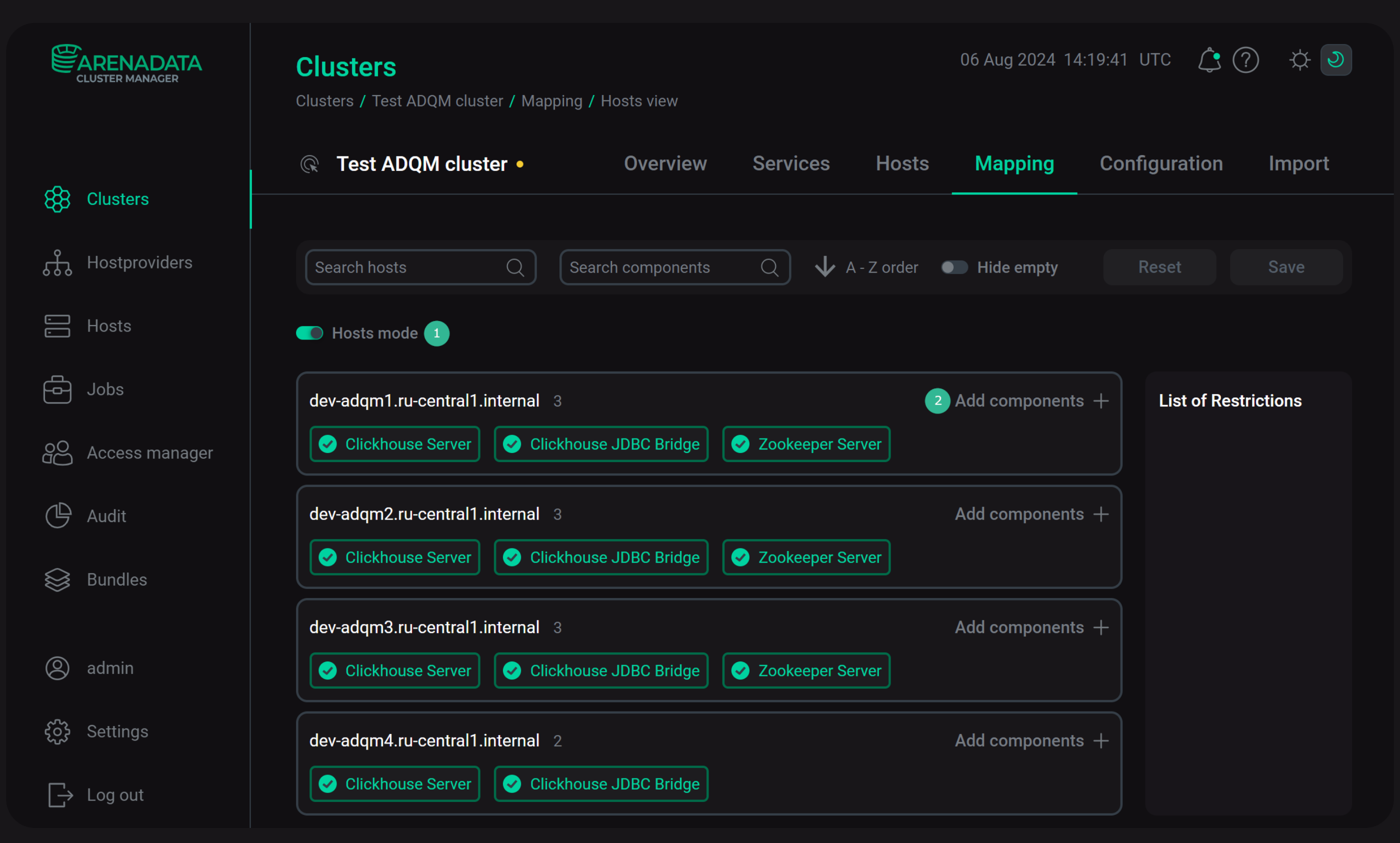
Host	Service	Component	Notes
Any host	ADQMDB	Clickhouse Server	One or more components should be installed
All hosts with Clickhouse Server	ADQMDB	Clickhouse JDBC Bridge	If you install this component (optionally), add it to all hosts in the cluster where the Clickhouse Server component is installed
Any host	Zookeeper	Zookeeper Server	One or more components should be installed. Total amount should be odd
Any host	Clickhousekeeper	Clickhouse Keeper Server	One or more components should be installed. Total amount should be odd
Any host	Chproxy	Chproxy Proxy Server	One or more components should be installed

4. After component distribution is completed, click ***Save***.



Save mapping of components

Starting with ADCM 2.2.0, you can use another mapping mode — by hosts. To do this, switch on the ***Hosts mode*** toggle and map components to hosts by clicking ***Add components*** for each host sequentially.



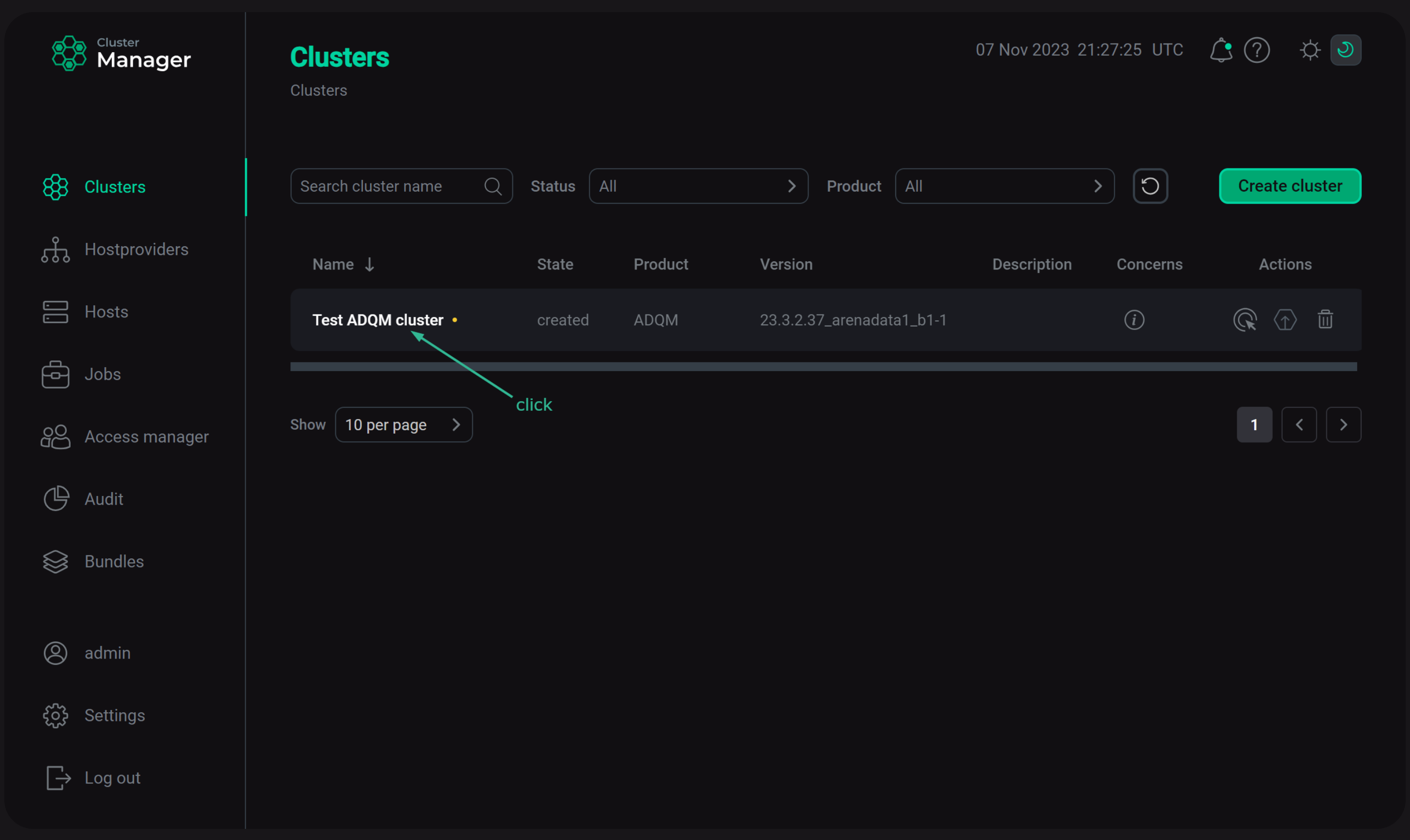
Use mapping by hosts

Configure services

Daria Barysheva, Elena Dvoryadkina

To Table of Contents

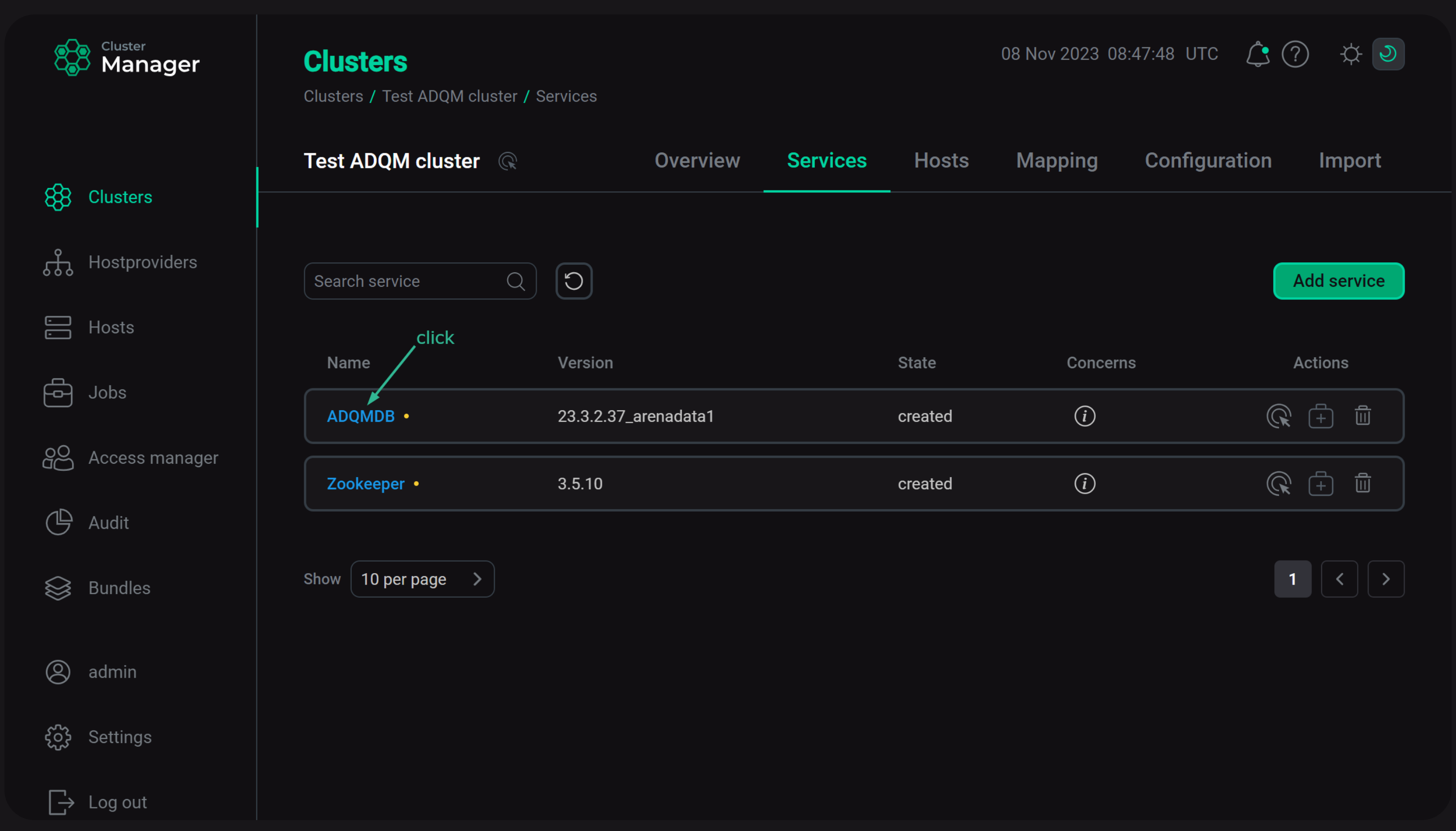
1. Select a cluster on the ***Clusters*** page. To do this, click a cluster name in the ***Name*** column.


Cluster
Manager
Clusters
Clusters
07 Nov 2023 21:27:25 UTC
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Search cluster name
Status
All
Product
All
Create cluster
click
Show
10 per page
1

Name	State	Product	Version	Description	Concerns	Actions
Test ADQM cluster •	created	ADQM	23.3.2.37_arenadata1_b1-1			

Select a cluster

2. Open the ***Services*** tab on the cluster page, find the row that contains the desired service, and click its name in the ***Name*** column.

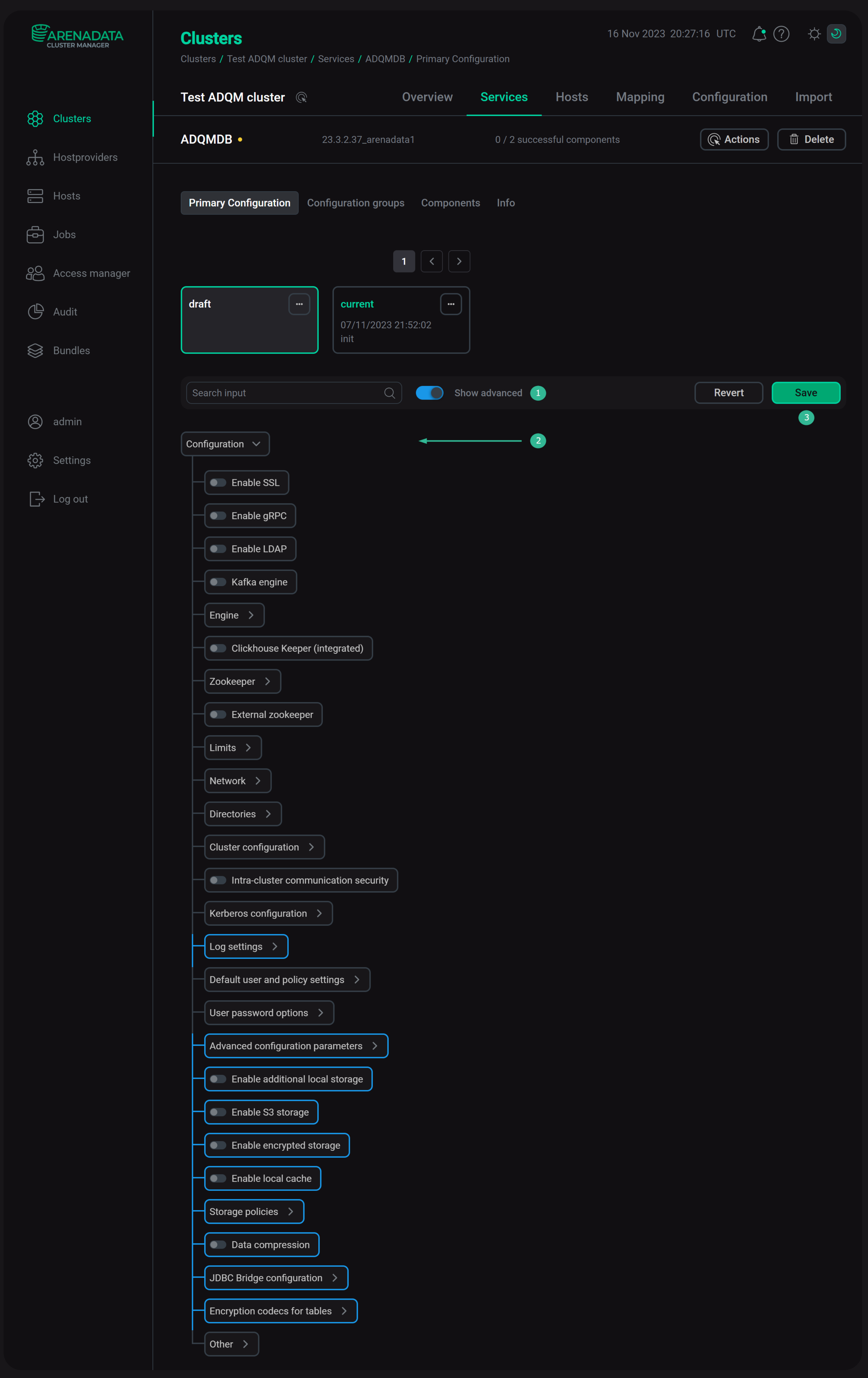

Cluster
Manager
Clusters
Clusters / Test ADQM cluster / Services
08 Nov 2023 08:47:48 UTC
Test ADQM cluster
Overview
Services
Hosts
Mapping
Configuration
Import
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Search service
Add service
click
Show
10 per page
1

Name	Version	State	Concerns	Actions
ADQMDB •	23.3.2.37_arenadata1	created		
Zookeeper •	3.5.10	created		

Switch to configuring a service

3. In the window that opens:

   - Open the ***Primary Configuration*** tab.
   - Switch on the ***Show advanced*** toggle.
   - Fill in configuration parameters of the selected service. The fields highlighted in red are mandatory.
   - Click ***Save***.


ARENADATA
CLUSTER MANAGER
Clusters
Clusters / Test ADQM cluster / Services / ADQMDB / Primary Configuration
16 Nov 2023 20:27:16 UTC
Test ADQM cluster
Overview
Services
Hosts
Mapping
Configuration
Import
ADQMDB •
23.3.2.37_arenadata1
0 / 2 successful components
Actions
Delete
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Primary Configuration
Configuration groups
Components
Info
1
draft
current
07/11/2023 21:52:02
init
Search input
Show advanced
1
Revert
Save
3
Configuration
2
Enable SSL
Enable gRPC
Enable LDAP
Kafka engine
Engine
Clickhouse Keeper (integrated)
Zookeeper
External zookeeper
Limits
Network
Directories
Cluster configuration
Intra-cluster communication security
Kerberos configuration
Log settings
Default user and policy settings
User password options
Advanced configuration parameters
Enable additional local storage
Enable S3 storage
Enable encrypted storage
Enable local cache
Storage policies
Data compression
JDBC Bridge configuration
Encryption codecs for tables
Other

Configure a service

NOTE

For more information about services configuration parameters, see Configuration parameters in the References.

Configure a cluster

Daria Barysheva, Elena Dvoryadkina

Contents

To Table of Contents

Primary configuration

Configuration groups

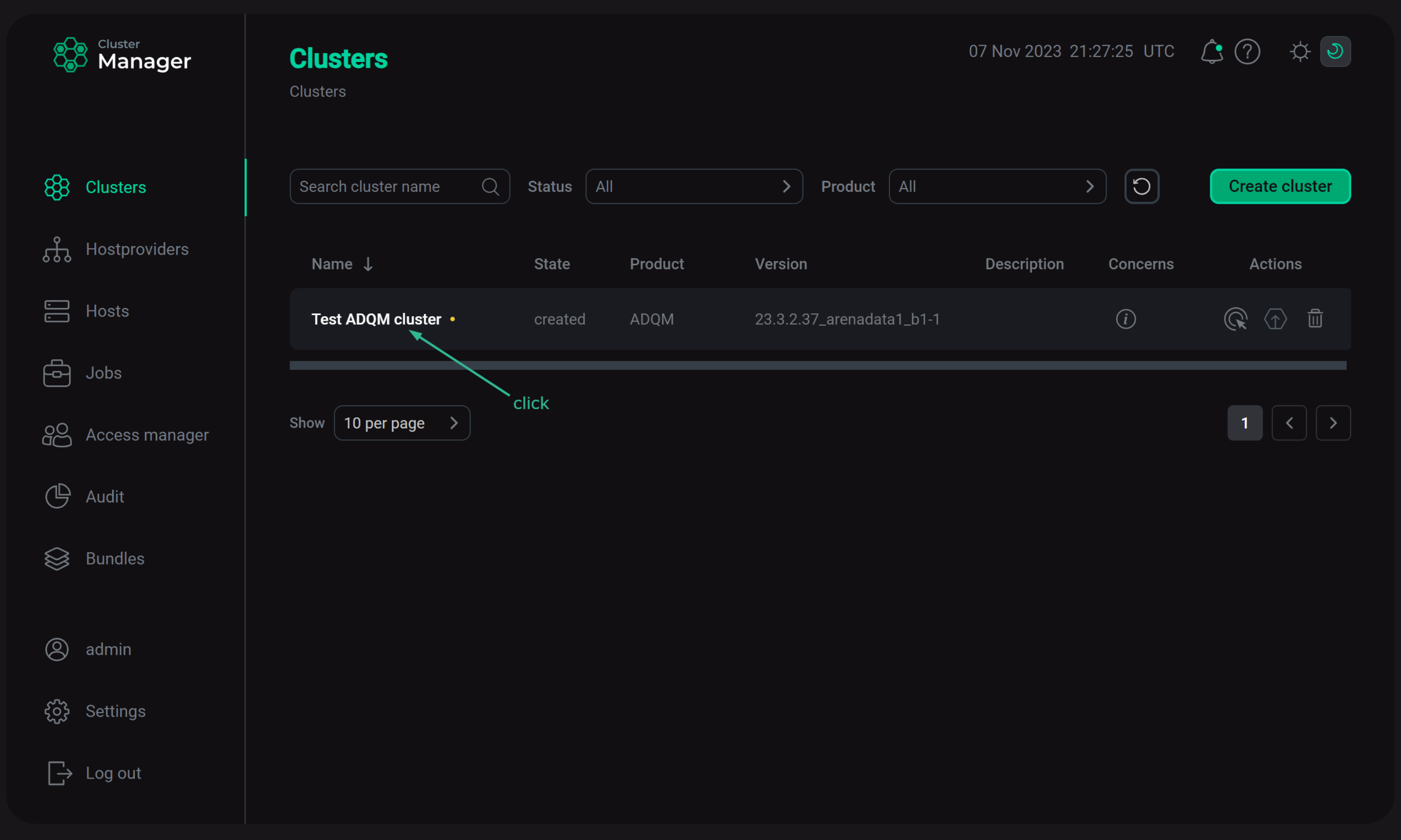
Ansible settings

IMPORTANT

There is usually no need to change cluster configuration parameters. You can leave all parameters at the default values.

After creation a new cluster, you can configure it by performing the following steps:

1. Select a cluster on the ***Clusters*** page. To do this, click a cluster name in the ***Name*** column.



Cluster
Manager
Clusters
Clusters
07 Nov 2023 21:27:25 UTC
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Search cluster name
Status
All
Product
All
Create cluster
Name
State
Product
Version
Description
Concerns
Actions
Test ADQM cluster
created
ADQM
23.3.2.37_arenadata1_b1-1
click
Show
10 per page
1

Select a cluster

2. Open the ***Configuration*** tab on the cluster page. The ***Configuration*** tab includes the following sections: ***Primary configuration***, ***Configuration groups***, ***Ansible settings***.

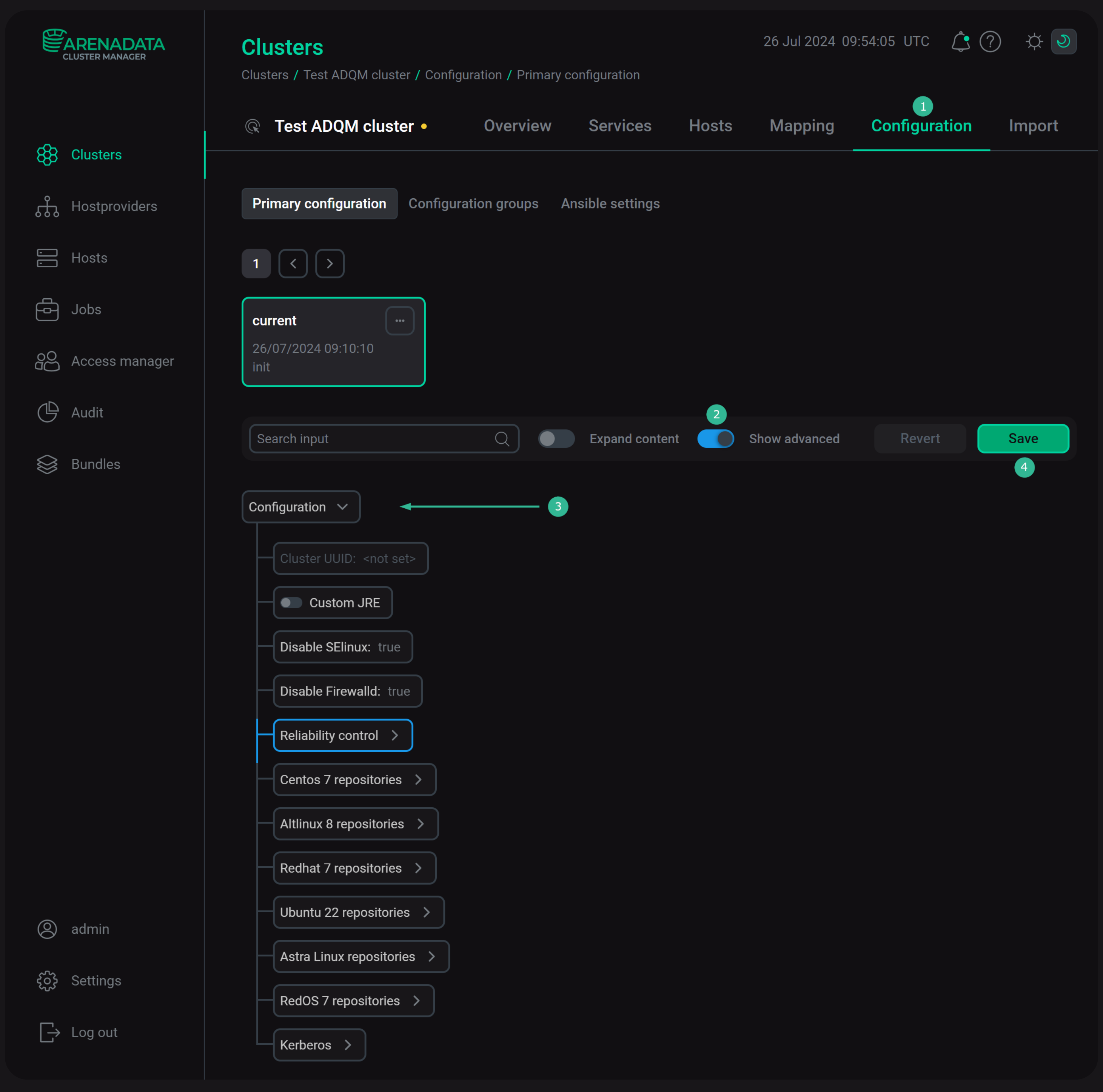
3. Fill in all necessary parameters on the selected tab and click ***Save***.

Primary configuration

The ***Configuration*** → ***Primary configuration*** tab allows you to configure the general cluster settings.

ADQM cluster configuration parameters are listed below:

- ***Custom JRE*** — the switcher that indicates whether to use a custom path to Java for ADQM services. When the switcher is activated, the ***JAVA_HOME*** field becomes available — enter a work Java path in this field. No system JRE packages will be installed.
- ***Disable SElinux*** — disables SElinux on hosts of the cluster (available starting with ADQM 22.8.4.7.1.b1).
- ***Disable Firewalld*** — disables firewalld on hosts of the cluster (available starting with ADQM 22.8.4.7.1.b1).
- ***Reliability control*** — the section with the following settings (visible if the ***Show advanced*** option is enabled):
  - ***retries*** — number of retries for cluster operations (`3` by default);
  - ***delay*** — delay between retries in seconds (`10` by default);
  - ***timeout*** — timeout for checks of network ports in seconds (`60` by default).
- Paths to the repositories that are used during the installation process — depending on the cluster operating system (Centos, Alt Linux, Red Hat, Ubuntu, Astra Linux, or RED OS):
  - ***monitoring*** — repository required to install monitoring;
  - ***zookeeper_repo*** — repository required to install Arenadata Zookeeper;
  - ***ADQM*** — repository required to install ADQM;
  - ***prometheus_repo*** — repository required to install Prometheus;
  - ***ADQM_arenadata_enterprise*** — repository required to install the Enterprise edition of ADQM.
- ***Kerberos*** — configuration for Kerberos authentication (some parameters are visible only with the ***Show advanced*** option enabled).



ARENADATA
CLUSTER MANAGER
Clusters
Clusters / Test ADQM cluster / Configuration / Primary configuration
26 Jul 2024 09:54:05 UTC
Test ADQM cluster
Overview
Services
Hosts
Mapping
1
Configuration
Import
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Primary configuration
Configuration groups
Ansible settings
1
current
26/07/2024 09:10:10
init
Search input
Expand content
2
Show advanced
Revert
Save
4
Configuration
3
Cluster UUID: <not set>
Custom JRE
Disable SElinux: true
Disable Firewalld: true
Reliability control
Centos 7 repositories
Altlinux 8 repositories
Redhat 7 repositories
Ubuntu 22 repositories
Astra Linux repositories
RedOS 7 repositories
Kerberos

Cluster configuration window

Configuration groups

The ***Configuration*** → ***Configuration groups*** tab is designed to set cluster configuration groups.

Ansible settings

The ***Configuration*** → ***Ansible settings*** tab allows you to set Ansible configuration options at the cluster level. The tab is available starting with ADCM 2.2.0.

Ansible settings

Name	Description	Default
forks	The number of parallel processes to spawn when communicating with remote hosts	5

Install a cluster

Daria Barysheva, Elena Dvoryadkina

Contents

To Table of Contents

- Step 1. Run installation
  - Whole cluster
  - Single services
- Step 2. View the results of installation
- Step 3. Check the cluster state after installation

Step 1. Run installation

There are two ways to install cluster services:

1. **Whole cluster**. In this method all services are installed automatically one by one after applying the ***Install*** action to a cluster.
2. **Single services**. In this method each service is installed manually after applying one or more actions to it. This way is also suitable for installing new services in the already deployed cluster.

In both cases each service starts automatically after installation.

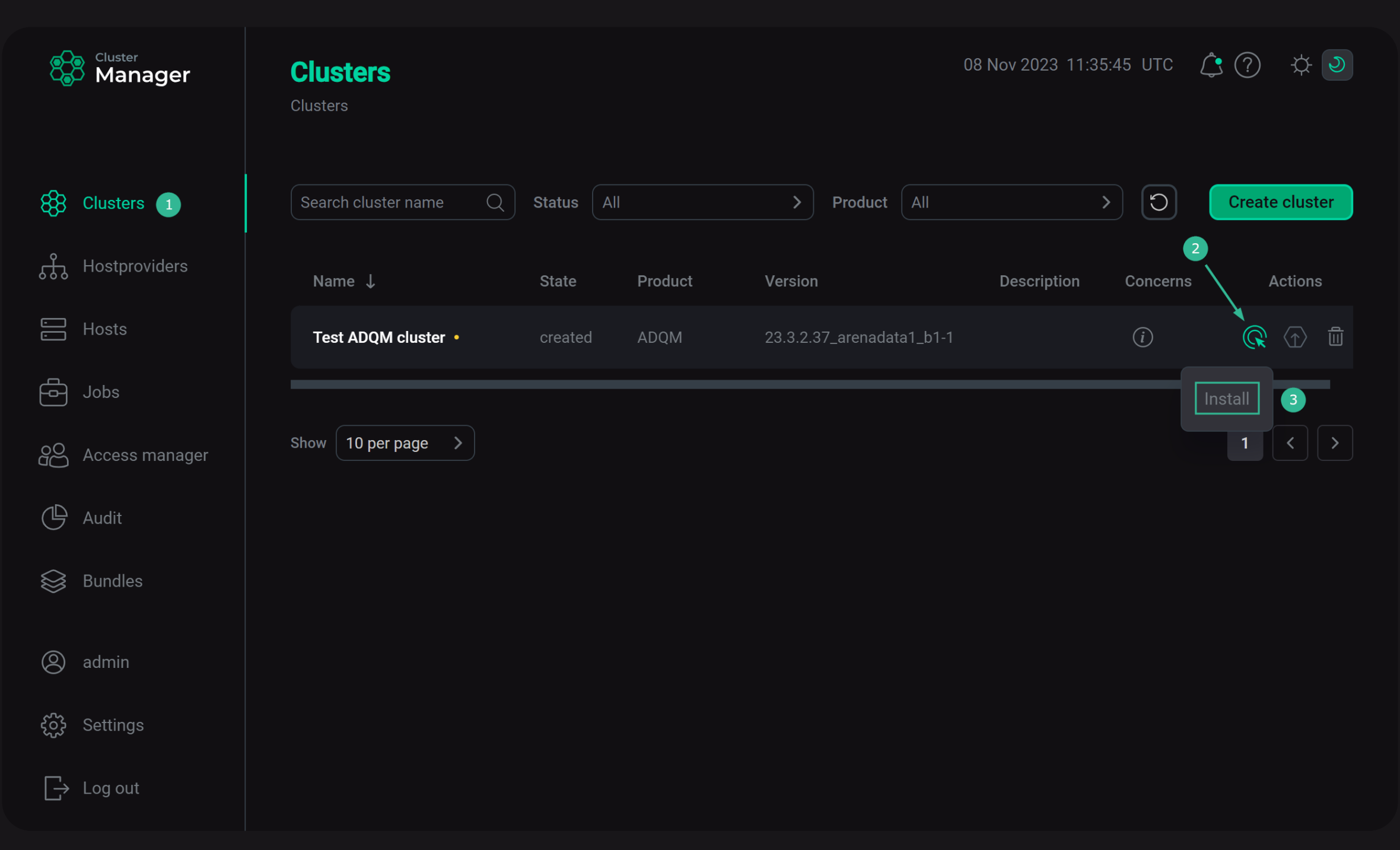
NOTE

In case of errors, you can find logs on the ***Jobs*** page.

Whole cluster

To install all cluster services within a single action, follow the steps:

1. Apply the ***Install*** action to the cluster selected on the ***Clusters*** page by clicking the icon in the ***Actions*** column.

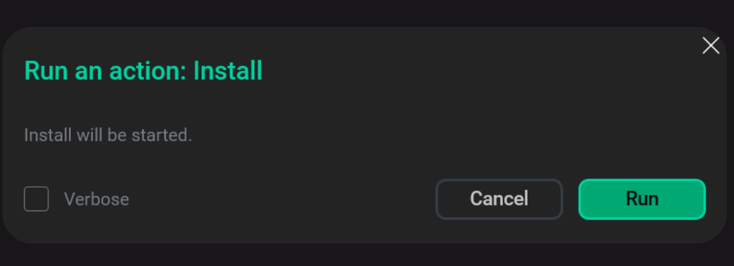


Cluster Manager
Clusters
08 Nov 2023 11:35:45 UTC
Clusters 1
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Search cluster name
Status All
Product All
Create cluster
2
Install
3
Show 10 per page
1

Name	State	Product	Version	Description	Concerns	Actions
Test ADQM cluster	created	ADQM	23.3.2.37_arenadata1_b1-1			

Switch to the whole cluster installation

2. Verify the requested action in the opened window.


Run an action: Install

Install will be started.

☐ Verbose

Verify the action

3. Wait until the job is completed.

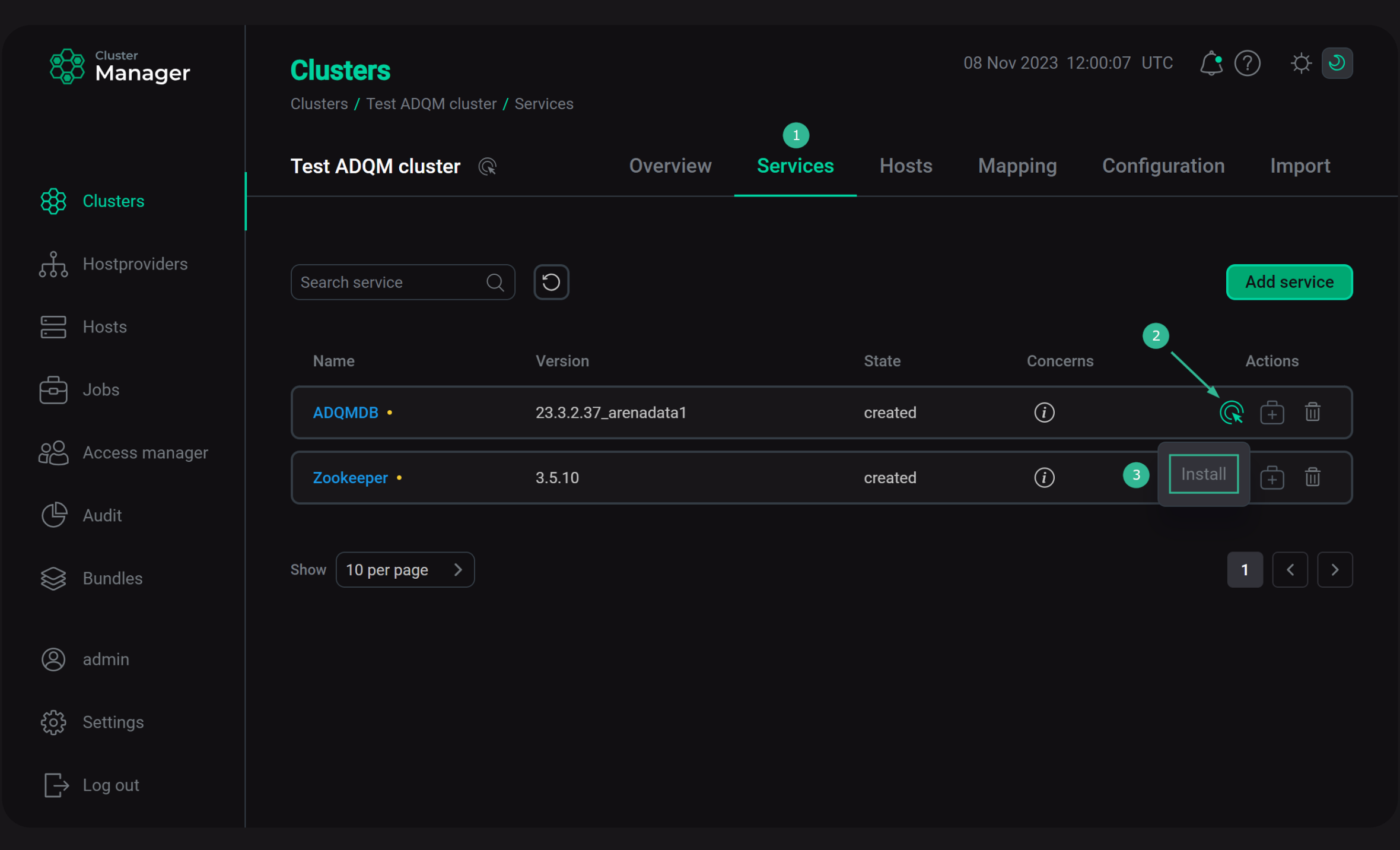
Single services

To install one or several cluster services manually, add these services to the cluster and then perform the actions described below.

CAUTION

Notice that services are installed one by one. Do not install another service until the installation of the previous one is completed.

1. Open the ***Services*** tab on the cluster page. For each service that should be installed, click the icon in the ***Actions*** column and select the ***Install*** action.

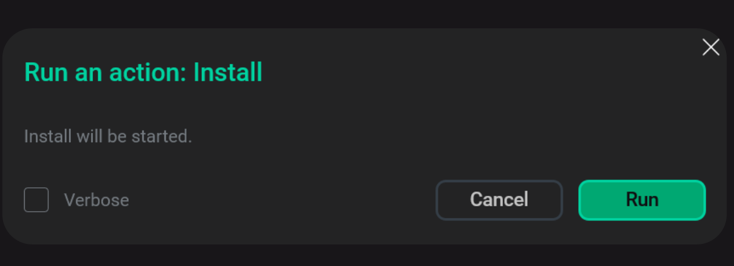


Cluster Manager
Clusters
Clusters / Test ADQM cluster / Services
08 Nov 2023 12:00:07 UTC
Test ADQM cluster
Overview
1
Services
Hosts
Mapping
Configuration
Import
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Search service
Add service
2
3
Install
Show 10 per page
1

Name	Version	State	Concerns	Actions
ADQMDB	23.3.2.37_arenadata1	created		
Zookeeper	3.5.10	created		

Switch to installation of single services

2. Verify the action in the opened window.


Run an action: Install

Install will be started.

☐ Verbose

Verify the action

3. Wait until the job is completed. Then check that the service state has changed from `created` to `installed`.
4. Repeat the previous actions for other services that should be installed.

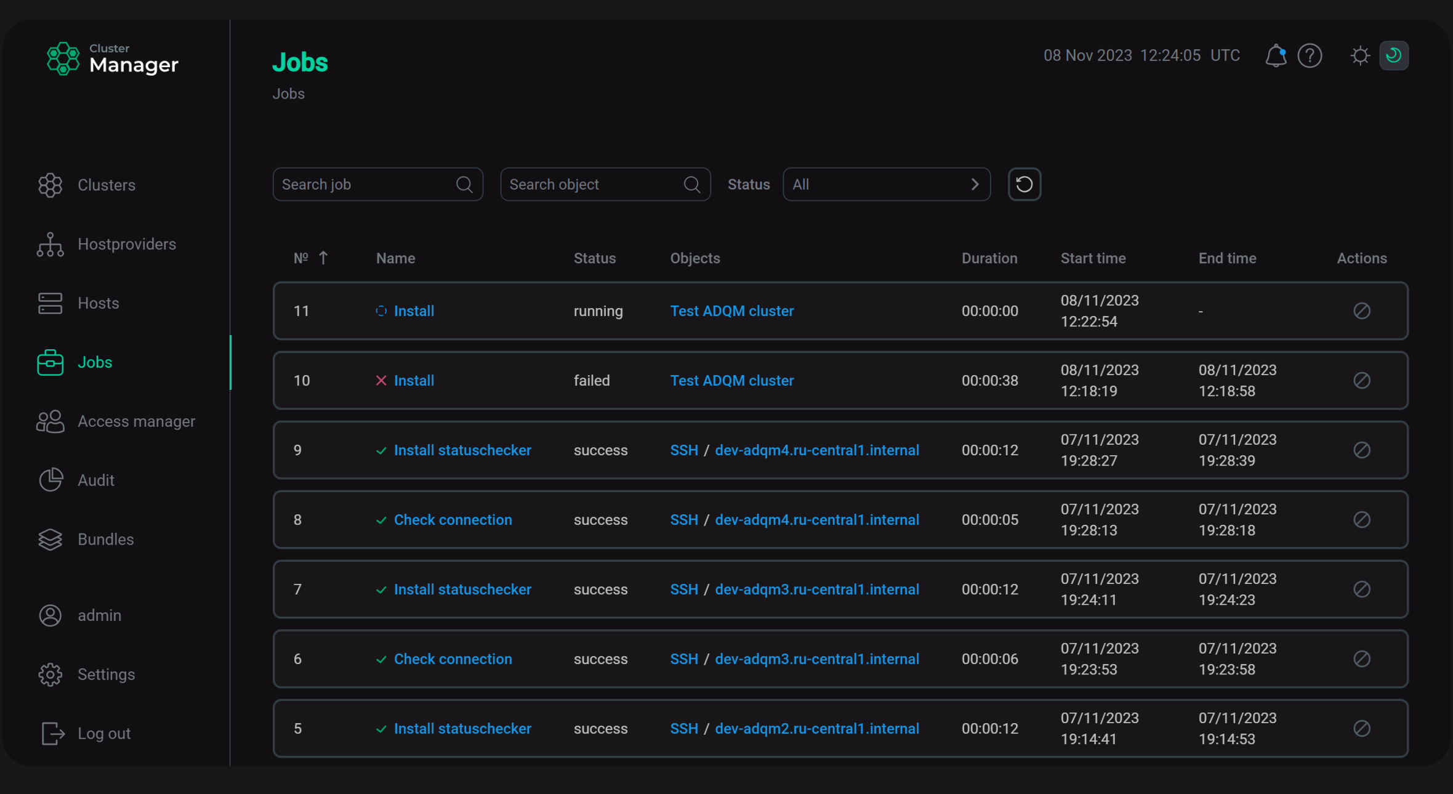
IMPORTANT

When you install services manually, one by one, use the following order:

1. Zookeeper / Clickhousekeeper
2. ADQMDB
3. Chproxy
4. Monitoring Clients / Monitoring

Step 2. View the results of installation

ADCM starts a single job for installation process, as well as for any other task. You can find out about the status of jobs more specifically on the ***Jobs*** page.

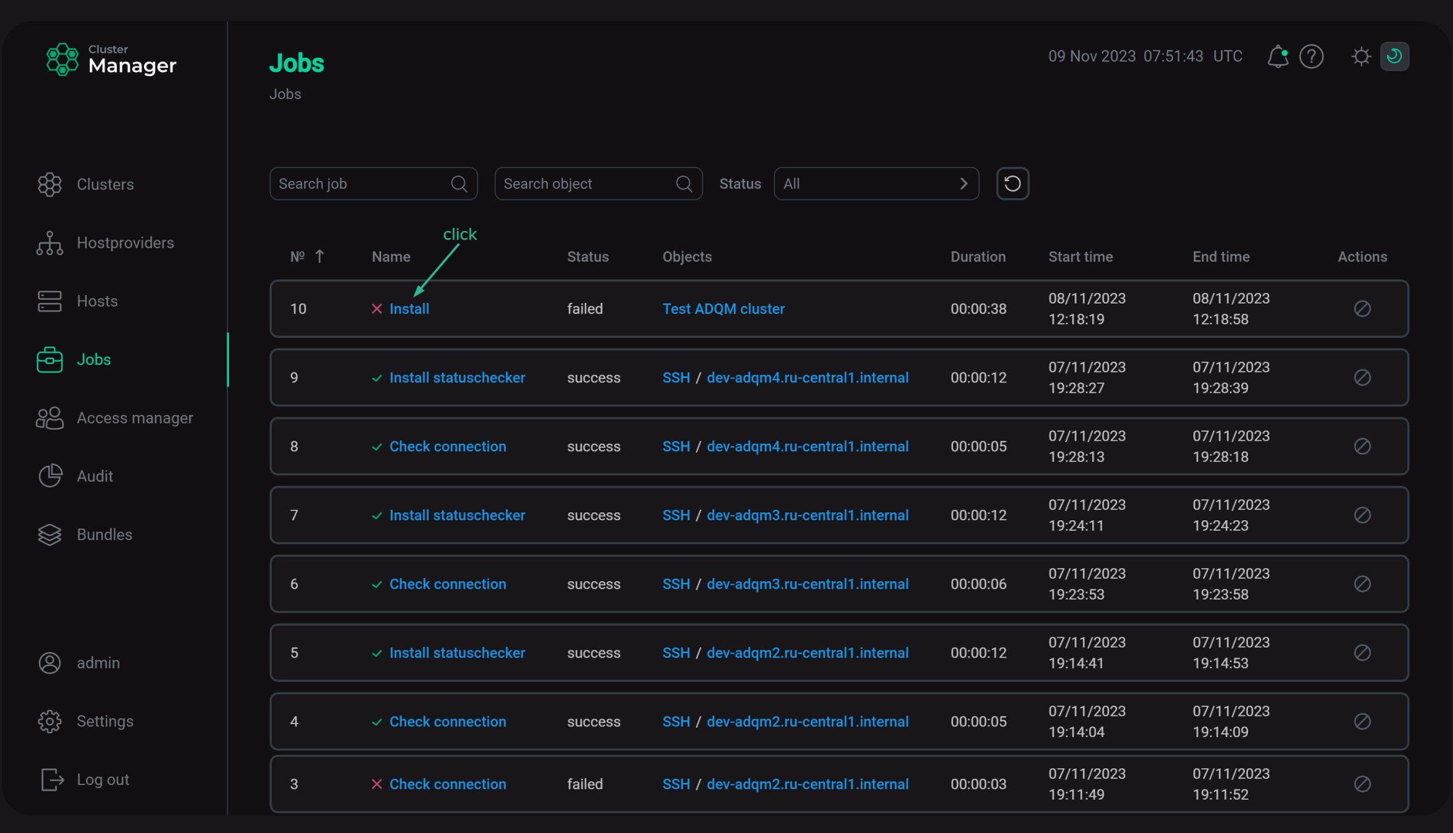


Cluster Manager
Jobs
Jobs
08 Nov 2023 12:24:05 UTC
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Search job
Search object
Status All

№	Name	Status	Objects	Duration	Start time	End time	Actions
11	Install	running	Test ADQM cluster	00:00:00	08/11/2023 12:22:54	-	
10	Install	failed	Test ADQM cluster	00:00:38	08/11/2023 12:18:19	08/11/2023 12:18:58	
9	Install statuschecker	success	SSH / dev-adqm4.ru-central1.internal	00:00:12	07/11/2023 19:28:27	07/11/2023 19:28:39	
8	Check connection	success	SSH / dev-adqm4.ru-central1.internal	00:00:05	07/11/2023 19:28:13	07/11/2023 19:28:18	
7	Install statuschecker	success	SSH / dev-adqm3.ru-central1.internal	00:00:12	07/11/2023 19:24:11	07/11/2023 19:24:23	
6	Check connection	success	SSH / dev-adqm3.ru-central1.internal	00:00:06	07/11/2023 19:23:53	07/11/2023 19:23:58	
5	Install statuschecker	success	SSH / dev-adqm2.ru-central1.internal	00:00:12	07/11/2023 19:14:41	07/11/2023 19:14:53	

The Jobs page

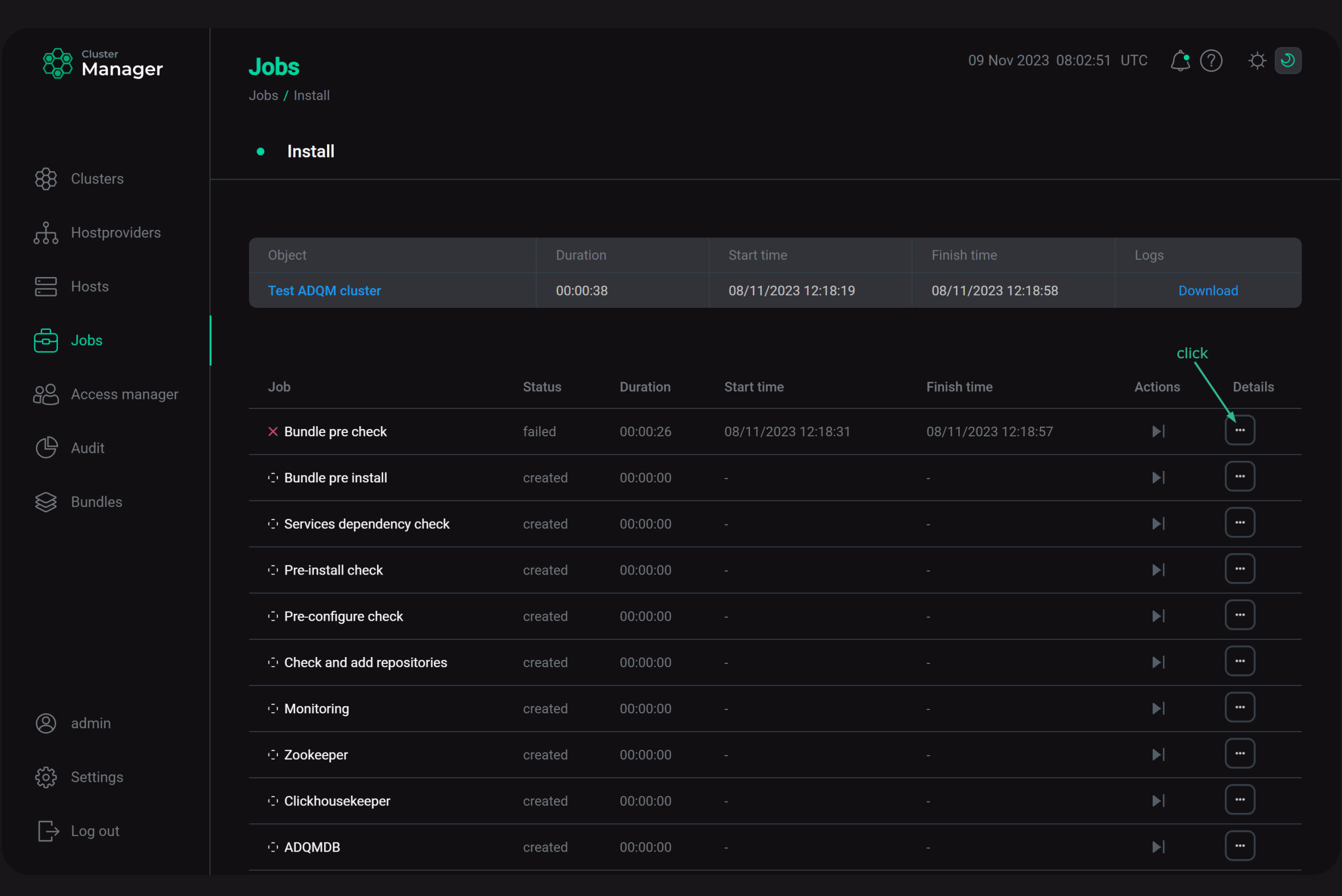
The successful completion of the service installation is determined by the transition of the job from the `running` status to the `success` status on the ***Jobs*** page. If the job fails, it switches to the `failed` status. In this case it is possible to see more detailed information about the errors that occurred by clicking the failed job on the ***Jobs*** page.



Cluster Manager
Jobs
Jobs
09 Nov 2023 07:51:43 UTC
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Search job
Search object
Status All
click

№	Name	Status	Objects	Duration	Start time	End time	Actions
10	Install	failed	Test ADQM cluster	00:00:38	08/11/2023 12:18:19	08/11/2023 12:18:58	
9	Install statuschecker	success	SSH / dev-adqm4.ru-central1.internal	00:00:12	07/11/2023 19:28:27	07/11/2023 19:28:39	
8	Check connection	success	SSH / dev-adqm4.ru-central1.internal	00:00:05	07/11/2023 19:28:13	07/11/2023 19:28:18	
7	Install statuschecker	success	SSH / dev-adqm3.ru-central1.internal	00:00:12	07/11/2023 19:24:11	07/11/2023 19:24:23	
6	Check connection	success	SSH / dev-adqm3.ru-central1.internal	00:00:06	07/11/2023 19:23:53	07/11/2023 19:23:58	
5	Install statuschecker	success	SSH / dev-adqm2.ru-central1.internal	00:00:12	07/11/2023 19:14:41	07/11/2023 19:14:53	
4	Check connection	success	SSH / dev-adqm2.ru-central1.internal	00:00:05	07/11/2023 19:14:04	07/11/2023 19:14:09	
3	Check connection	failed	SSH / dev-adqm2.ru-central1.internal	00:00:03	07/11/2023 19:11:49	07/11/2023 19:11:52	

Switch to a job page



Cluster Manager
Jobs
Jobs / Install
09 Nov 2023 08:02:51 UTC
Install
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
click

Object	Duration	Start time	Finish time	Logs
Test ADQM cluster	00:00:38	08/11/2023 12:18:19	08/11/2023 12:18:58	Download

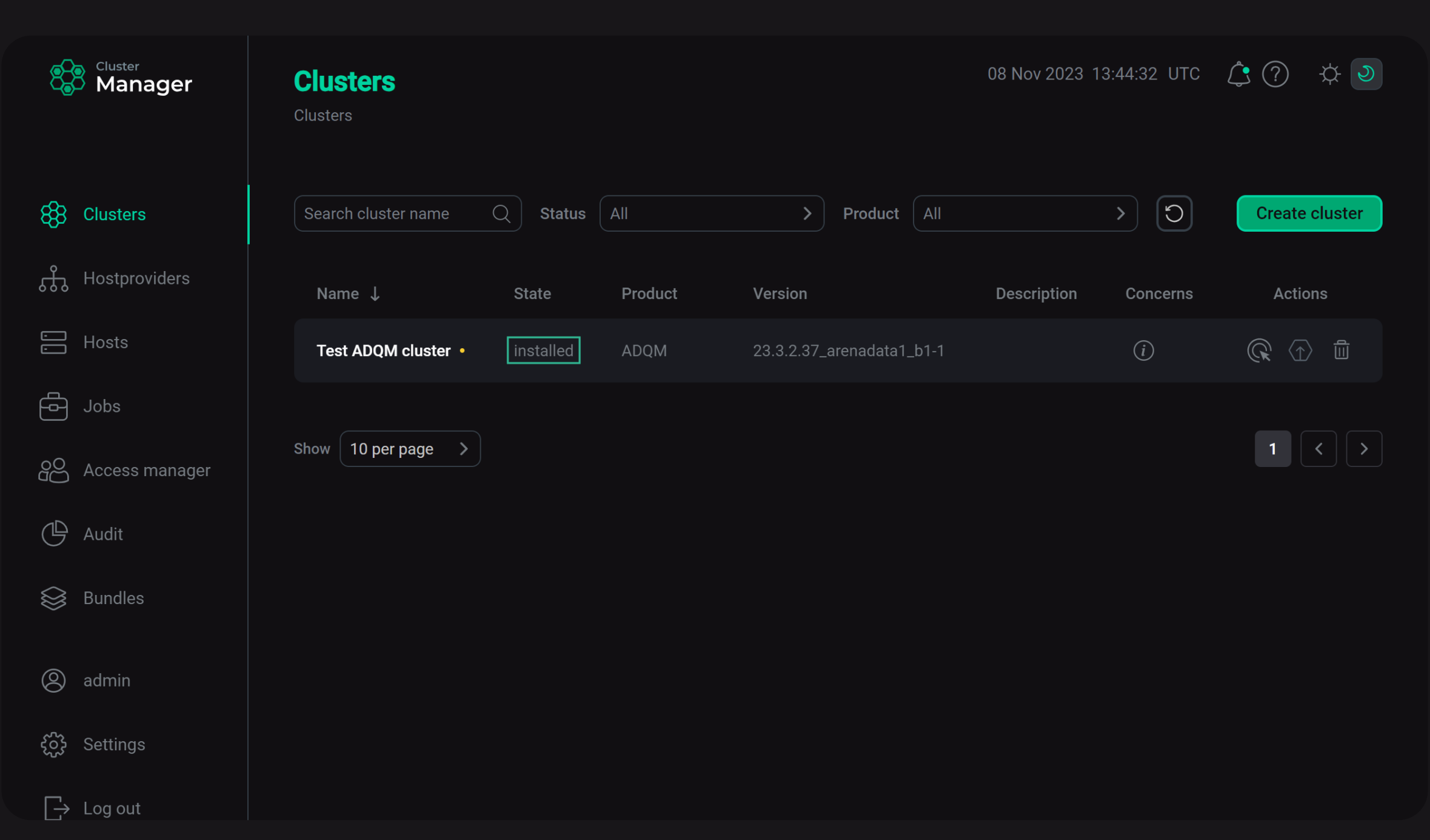
Job	Status	Duration	Start time	Finish time	Actions	Details
Bundle pre check	failed	00:00:26	08/11/2023 12:18:31	08/11/2023 12:18:57		
Bundle pre install	created	00:00:00	-	-		
Services dependency check	created	00:00:00	-	-		
Pre-install check	created	00:00:00	-	-		
Pre-configure check	created	00:00:00	-	-		
Check and add repositories	created	00:00:00	-	-		
Monitoring	created	00:00:00	-	-		
Zookeeper	created	00:00:00	-	-		
Clickhousekeeper	created	00:00:00	-	-		
ADQMDB	created	00:00:00	-	-		

Install cluster job page

Step 3. Check the cluster state after installation

As a result of successful installation, the cluster and its services change their state according to the following rules:

- The cluster state, displayed in the ***State*** field on the ***Clusters*** page, is being changed from `created` to `installed`.

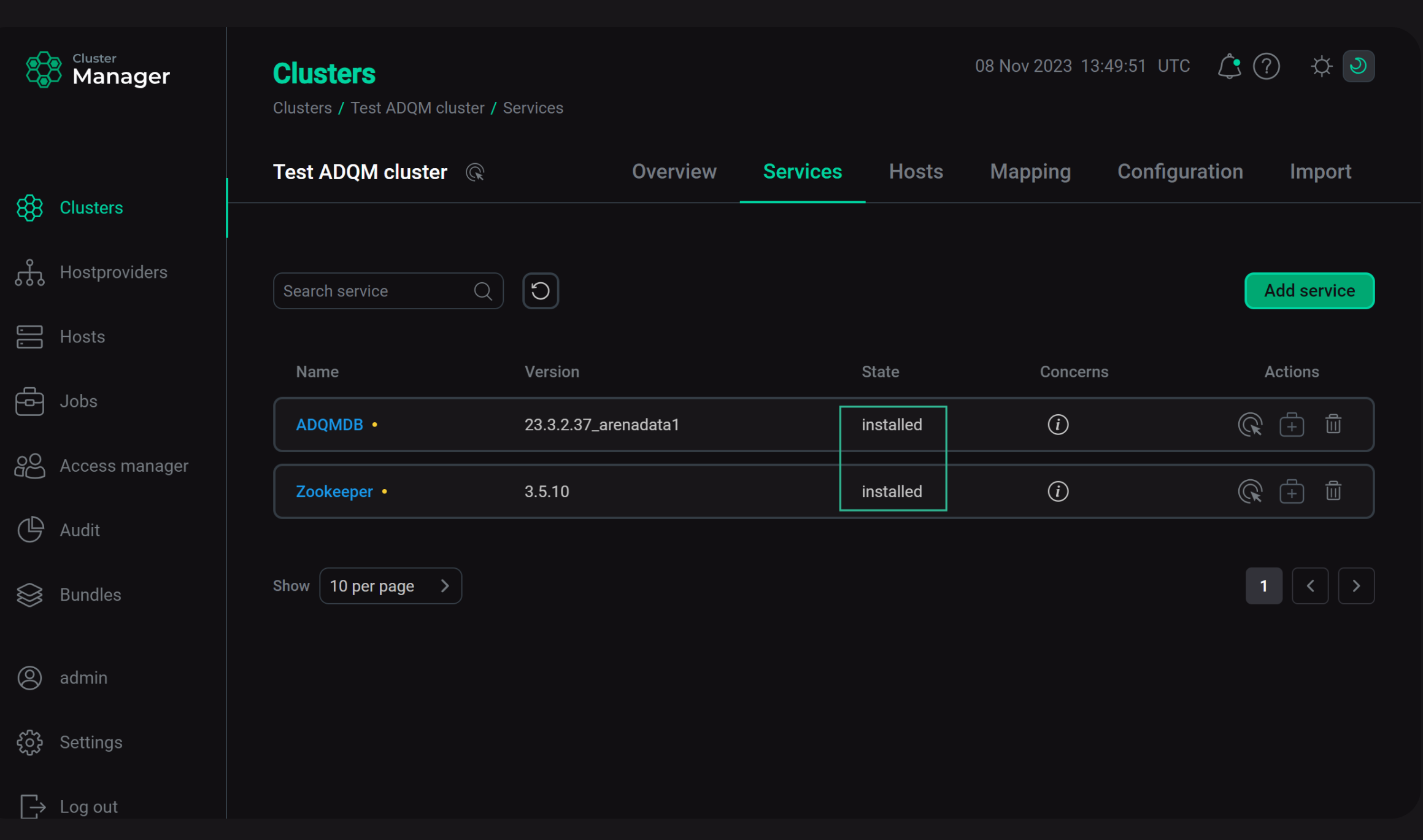


Cluster Manager
Clusters
Clusters
08 Nov 2023 13:44:32 UTC
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Search cluster name
Status All
Product All
Create cluster
Show 10 per page
1

Name	State	Product	Version	Description	Concerns	Actions
Test ADQM cluster	installed	ADQM	23.3.2.37_arenadata1_b1-1			

The cluster state after successful installation

- The state of cluster services, displayed on the ***Services*** tab of the cluster page, is being changed from `created` to `installed`.



Cluster Manager
Clusters
Clusters / Test ADQM cluster / Services
08 Nov 2023 13:49:51 UTC
Test ADQM cluster
Overview
Services
Hosts
Mapping
Configuration
Import
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Search service
Add service
Show 10 per page
1

Name	Version	State	Concerns	Actions
ADQMDB	23.3.2.37_arenadata1	installed		
Zookeeper	3.5.10	installed		

The state of cluster services after successful installation

Monitoring service

Elena Dvoryadkina

Contents

To Table of Contents

Overview

Step 1. Add the service

Step 2. Add components

Step 3. Configure the service

Step 4. Install the service

Step 5. View results

- Metrics in the Prometheus format
- Prometheus web interface
- Grafana web interface

The Monitoring service (available starting with ADQM 23.8.2.7) deploys the Prometheus server inside ADQM to collect and store ADQM cluster monitoring metrics, and also supports the ability to use the Grafana web application for visualization and analysis of information. This article describes steps required to add this service to an ADQM cluster.

Overview

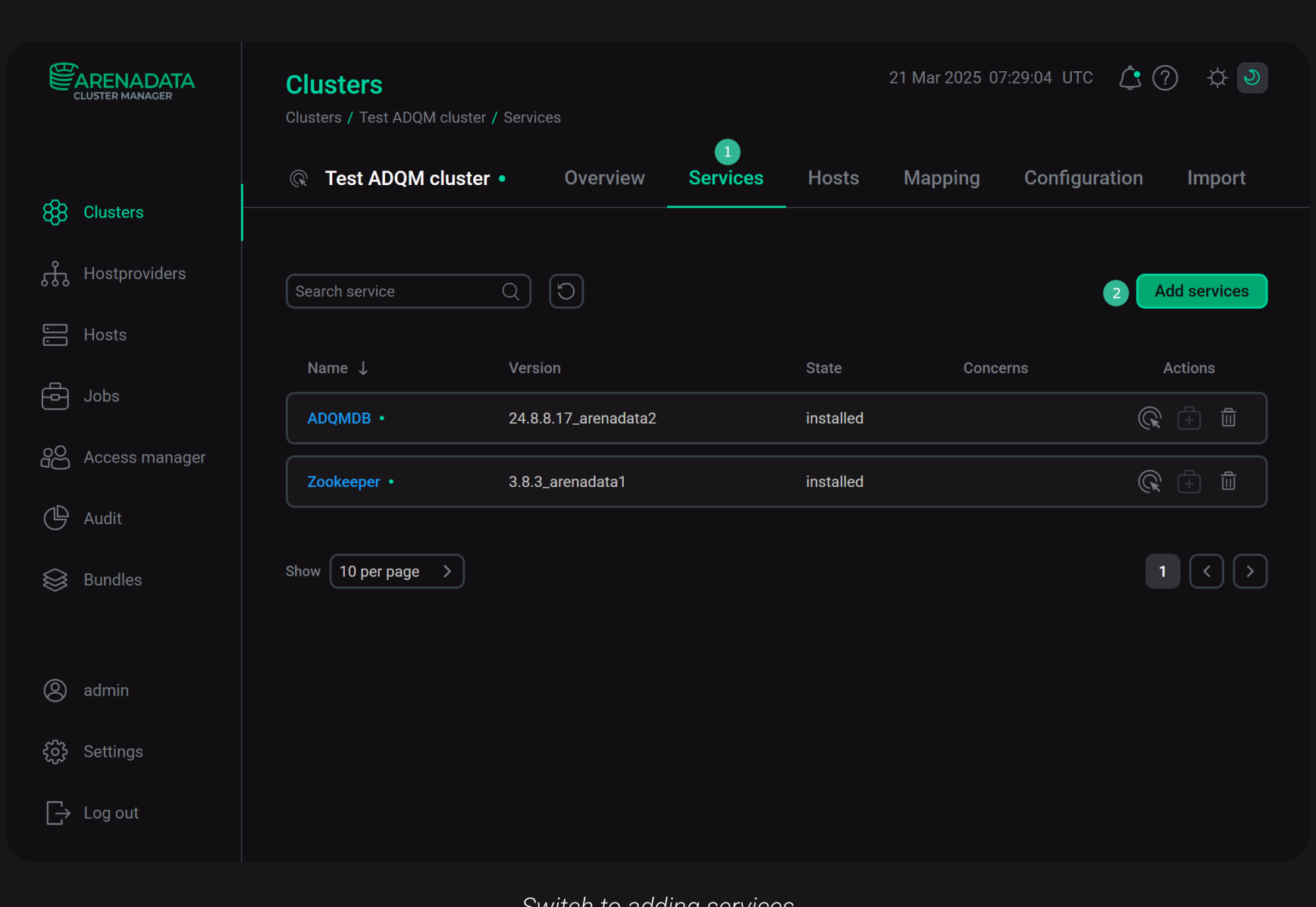
When adding the Monitoring service to ADQM, you can install the node exporter on all or selected hosts of your cluster. It is a monitoring agent that reads host system metrics that Prometheus collects. Prometheus can also collect metrics from ADQM services (ClickHouse, ZooKeeper, ClickHouse Keeper, Chproxy). These metrics will be available in the Prometheus format on the ports and endpoints specified in the ***Metrics*** section of the service configurations. You can also use the Prometheus and Grafana web interfaces to view and analyze data that the monitoring service collects.

NOTE

- If you already have a Prometheus-compatible monitoring system set up (for example, your own Prometheus server or VictoriaMetrics), you can use it to collect ADQM metrics. To do this, configure access to ADQM metrics in your monitoring system using ADQM's Prometheus parameters specified on the Monitoring service page.
- You can also use the Federation mechanism to migrate all metrics from the Prometheus server deployed in ADQM to your Prometheus.

Step 1. Add the service

1. In the ADCM interface, open the ***Clusters*** page and click your ADQM cluster name. On the cluster page that opens, switch to the ***Services*** tab and click ***Add services***.

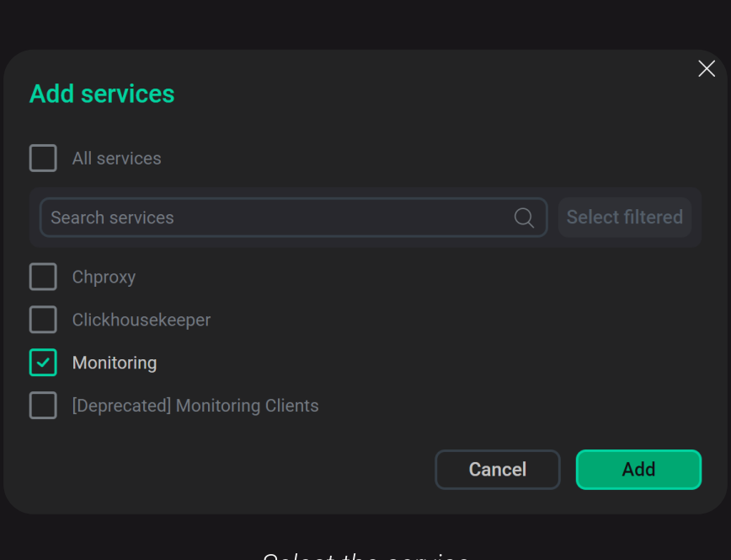


ARENADATA CLUSTER MANAGER
Clusters
Clusters / Test ADQM cluster / Services
21 Mar 2025 07:29:04 UTC
Test ADQM cluster
Overview
1
Services
Hosts
Mapping
Configuration
Import
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Search service
2
Add services
Show
10 per page

Name	Version	State	Concerns	Actions
ADQMDB	24.8.8.17_arenadata2	installed		
Zookeeper	3.8.3_arenadata1	installed		

Switch to adding services

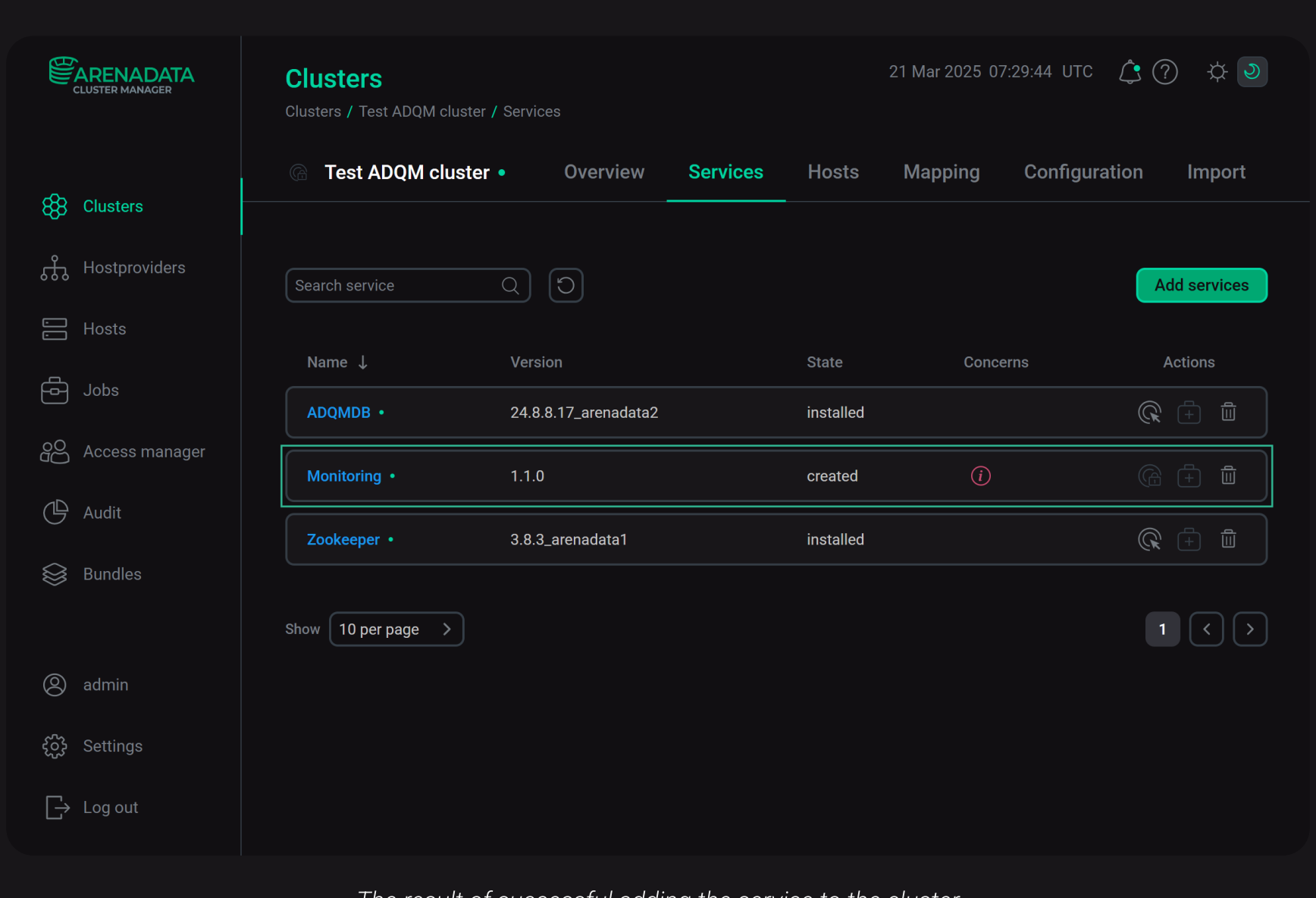
2. In the opened dialog, select the ***Monitoring*** service and click ***Add***.



Add services
☐ All services
Search services
Select filtered
☐ Chproxy
☐ Clickhousekeeper
☑ Monitoring
☐ [Deprecated] Monitoring Clients
Cancel
Add

Select the service

As a result, the added service is displayed on the ***Services*** tab.



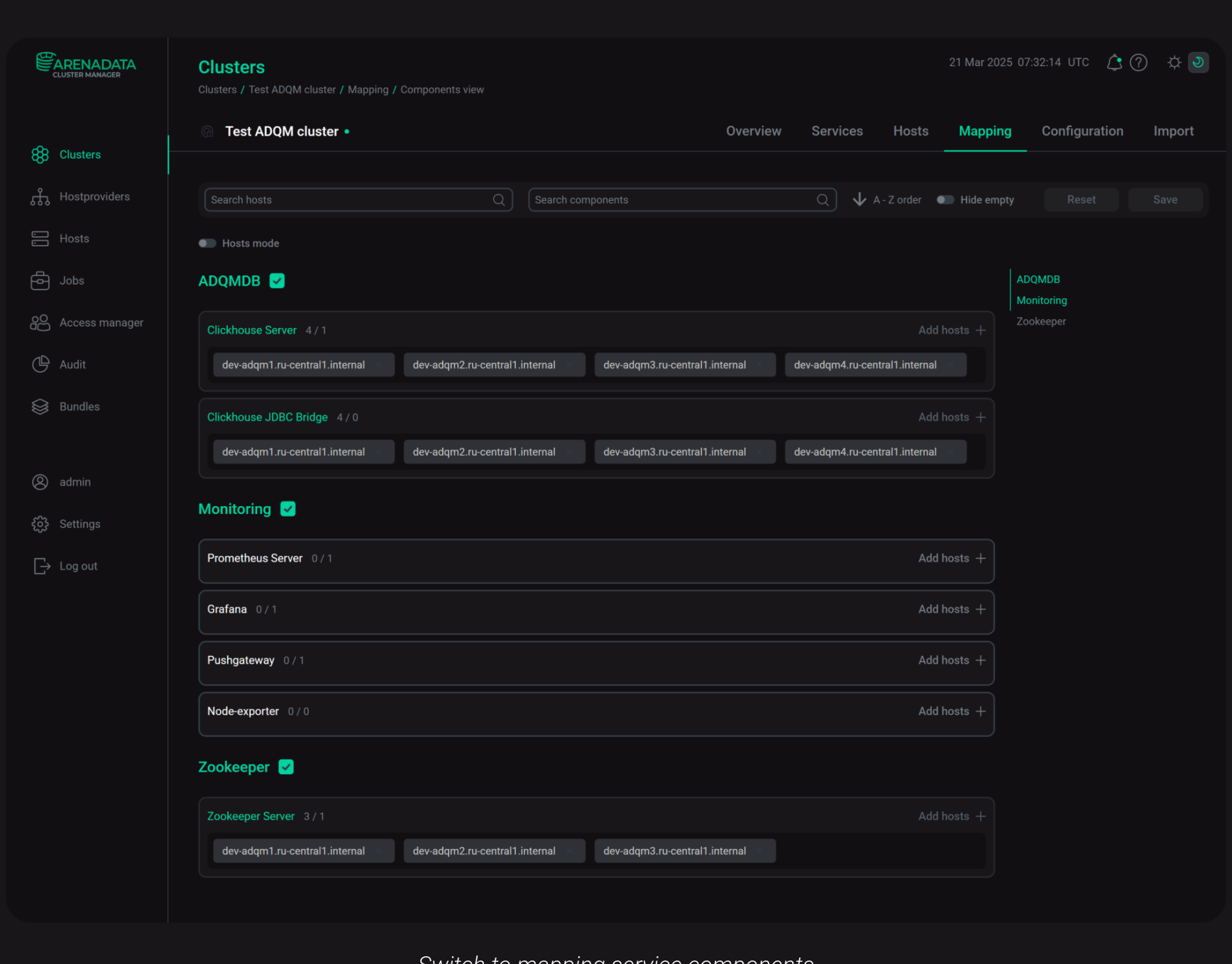
Clusters
Clusters / Test ADQM cluster / Services
21 Mar 2025 07:29:44 UTC
Test ADQM cluster
Overview
Services
Hosts
Mapping
Configuration
Import
Search service
Add services
Show
10 per page

Name	Version	State	Concerns	Actions
ADQMDB	24.8.8.17_arenadata2	installed		
Monitoring	1.1.0	created	ⓘ	
Zookeeper	3.8.3_arenadata1	installed		

The result of successful adding the service to the cluster

Step 2. Add components

1. On the cluster page, open the ***Mapping*** tab to proceed to mapping service components to cluster hosts.



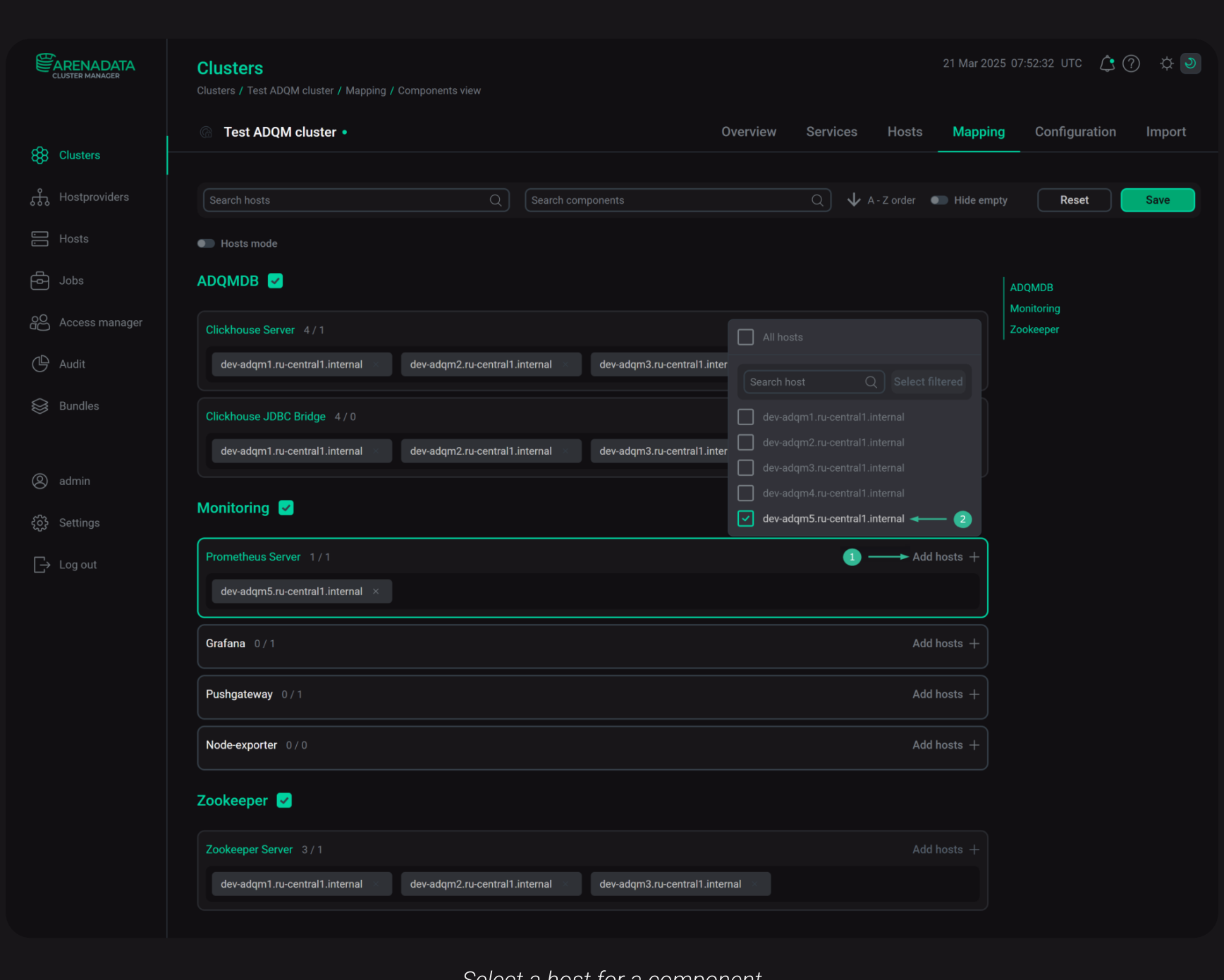
Clusters
Clusters / Test ADQM cluster / Mapping / Components view
Test ADQM cluster
Overview
Services
Hosts
Mapping
Configuration
Import
Search hosts
Search components
A - Z order
Hide empty
Reset
Save
Hosts mode
ADQMDB
Clickhouse Server 4 / 1
Clickhouse JDBC Bridge 4 / 0
Monitoring
Prometheus Server 0 / 1
Grafana 0 / 1
Pushgateway 0 / 1
Node-exporter 0 / 0
Zookeeper
Zookeeper Server 3 / 1
Add hosts

Switch to mapping service components

Monitoring service components

Component	Description
Prometheus Server	Stores and allows requesting all metrics for an ADQM cluster
Grafana	Visualizes ADQM metrics as graphs and charts organized into dashboards
Pushgateway	Receives static metrics and sends them to Prometheus. In ADQM, it is used to pass a cluster structure to Prometheus
Node Exporter	Collects system metrics and sends them to Prometheus

2. Assign hosts to components of the Monitoring service — click ***Add hosts*** and select the desired hosts in the pop-up window.



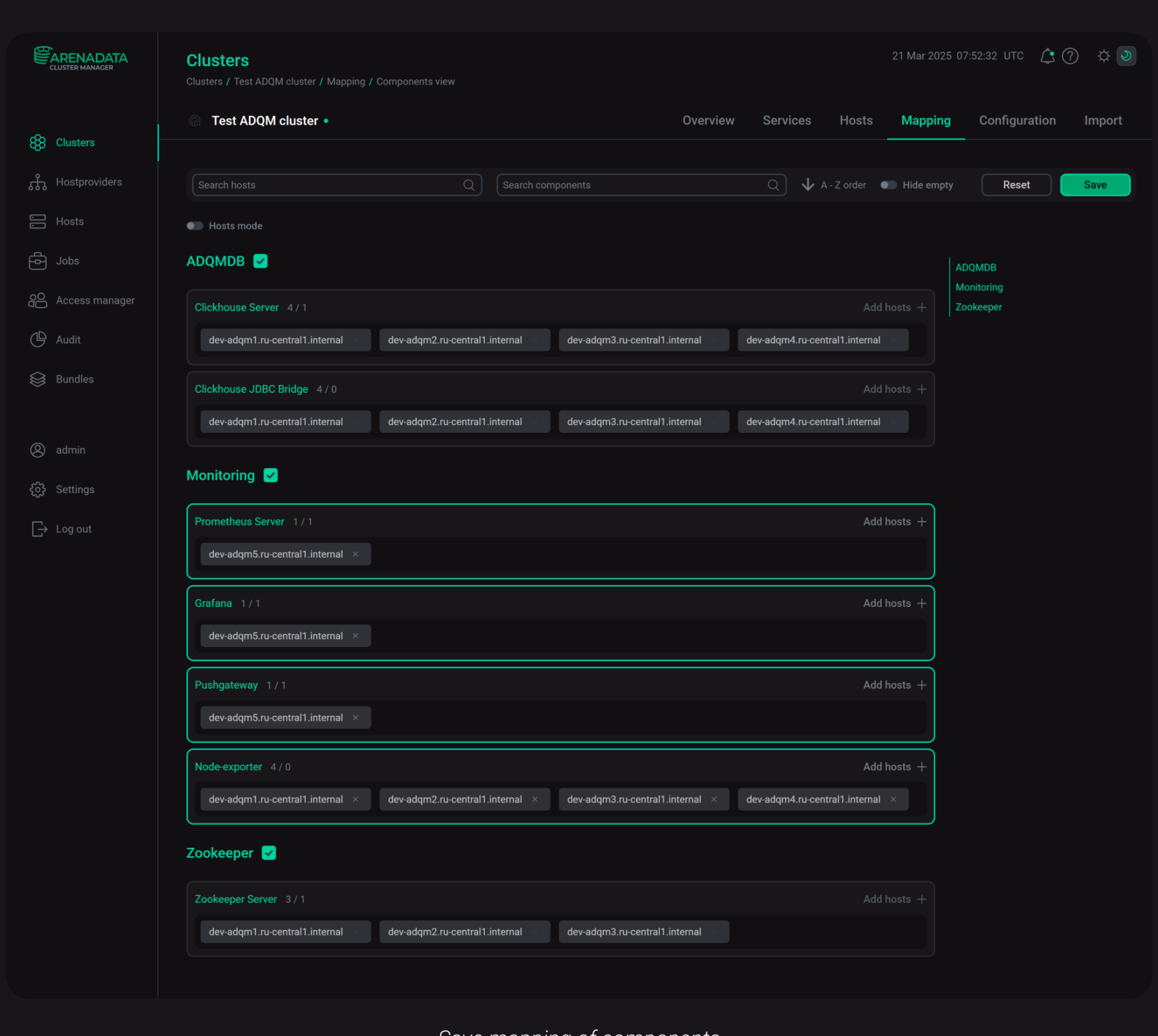
Clusters
Test ADQM cluster
Mapping
Save
ADQMDB
Monitoring
Prometheus Server 1 / 1
dev-adqm5.ru-central1.internal
1
Add hosts
2
All hosts
Search host
Select filtered
Grafana 0 / 1
Pushgateway 0 / 1
Node-exporter 0 / 0
Zookeeper

Select a host for a component

CAUTION

It is not recommended to install the Prometheus Server and Pushgateway components on hosts with ADQM — use separate hosts for them. Otherwise, if an ADQM host fails and/or the load on it is critically high, information about the corresponding problems will not be saved.

3. After the distribution of components is completed, click ***Save***.

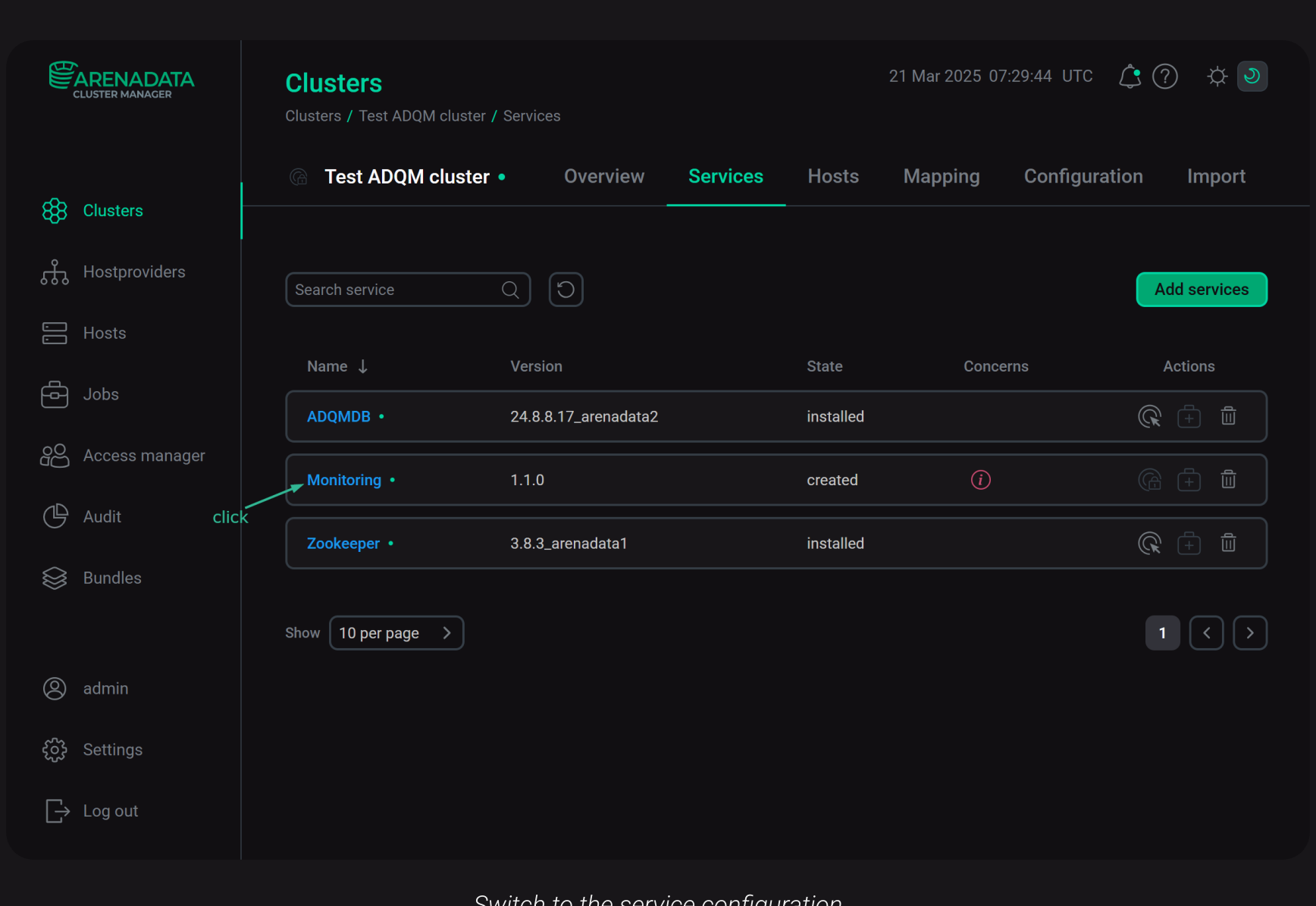


Clusters
Test ADQM cluster
Mapping
Reset
Save
ADQMDB
Clickhouse Server 4 / 1
Clickhouse JDBC Bridge 4 / 0
Monitoring
Prometheus Server 1 / 1
Grafana 1 / 1
Pushgateway 1 / 1
Node-exporter 4 / 0
Zookeeper
Zookeeper Server 3 / 1

Save mapping of components

Step 3. Configure the service

1. Open the ***Services*** tab on the cluster page and click the ***Monitoring*** service name in the ***Name*** column.

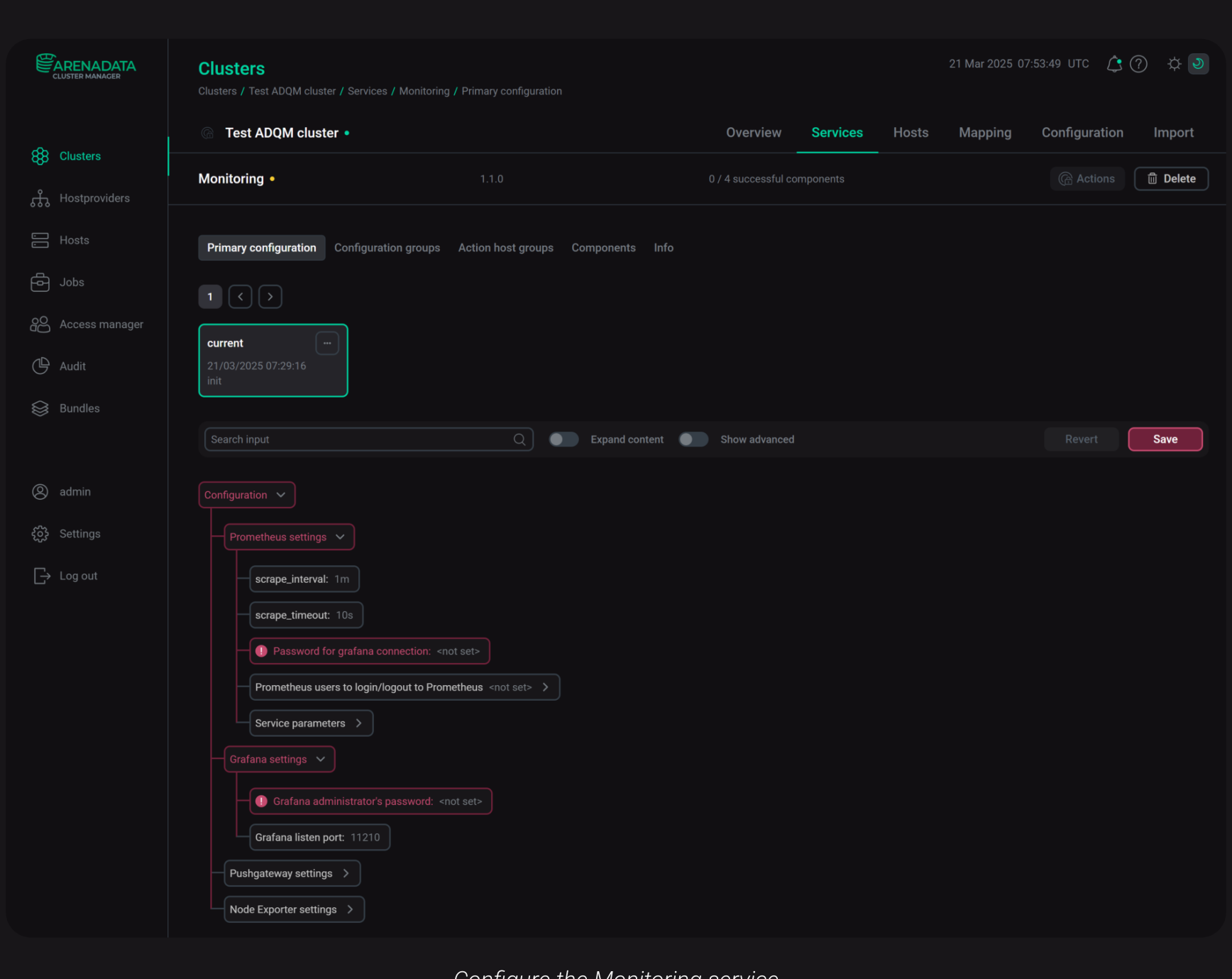


Clusters
Clusters / Test ADQM cluster / Services
Test ADQM cluster
Services
click
Add services

Name	Version	State	Concerns	Actions
ADQMDB	24.8.8.17_arenadata2	installed		
Monitoring	1.1.0	created	ⓘ	
Zookeeper	3.8.3_arenadata1	installed		

Switch to the service configuration

2. On the page that opens, fill in the service's configuration parameters — see the ***Monitoring*** section in the Configuration parameters article for parameter descriptions. Fields highlighted in red are required.



Clusters
Clusters / Test ADQM cluster / Services / Monitoring / Primary configuration
Monitoring
1.1.0
0 / 4 successful components
Actions
Delete
Primary configuration
Configuration groups
Action host groups
Components
Info
current
Search input
Expand content
Show advanced
Revert
Save
Configuration
Prometheus settings
scrape_interval 1m
scrape_timeout 10s
Password for grafana connection <not set>
Prometheus users to login/logout to Prometheus <not set>
Service parameters
Grafana settings
Grafana administrator's password <not set>
Grafana listen port 11210
Pushgateway settings
Node Exporter settings

Configure the Monitoring service

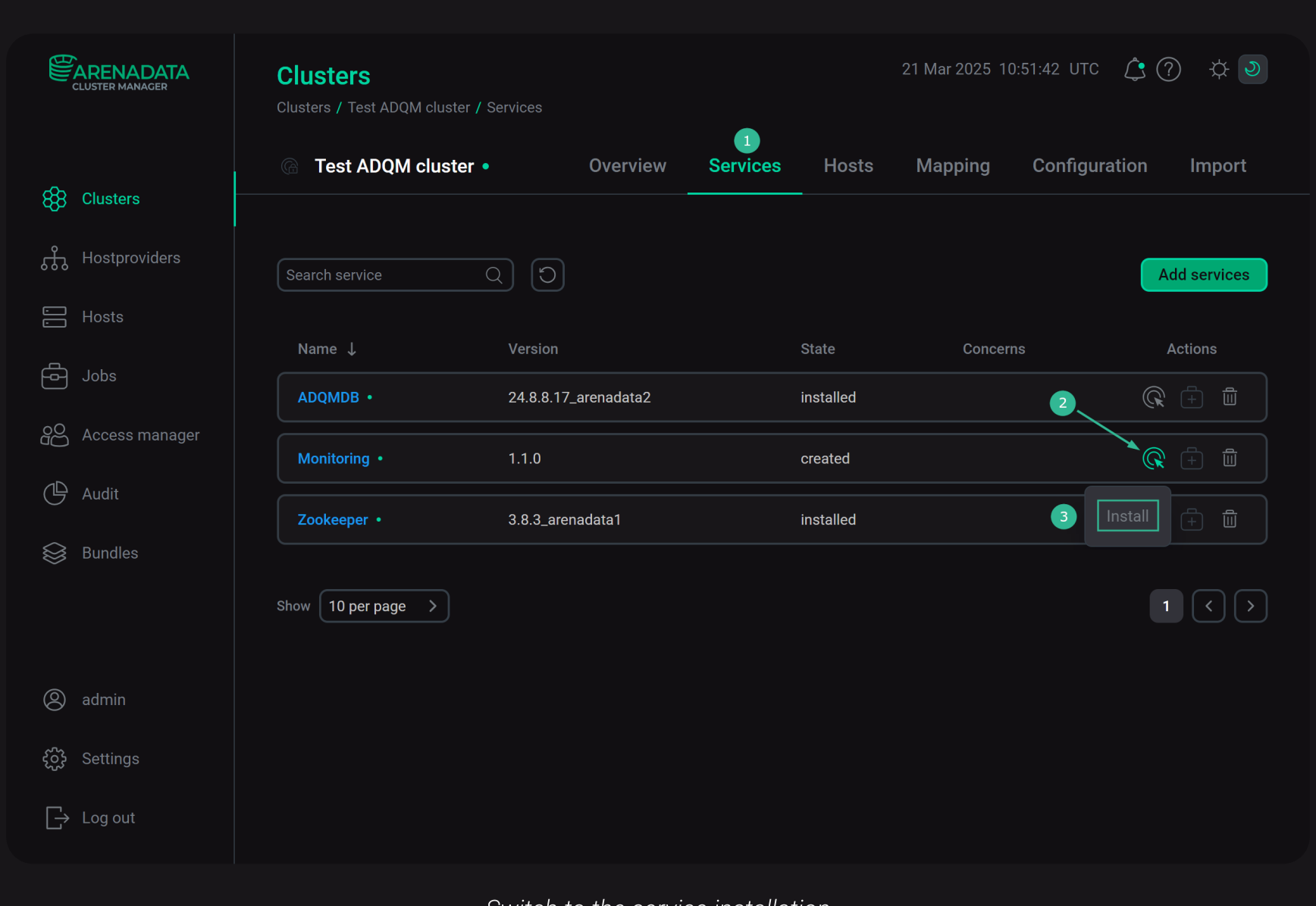
After specifying all necessary parameters, click ***Save***.

NOTE

To configure settings for access to metrics of the ADQMDB, Zookeeper, Clickhousekeeper services, use parameters in the ***Metrics*** activatable section of each service configuration. If you do not specify these parameters manually, the Monitoring service will automatically enable the ***Metrics*** section for ADQM services during installation and will use the default values.

Step 4. Install the service

1. On the ***Services*** tab, click the icon for the Monitoring service in the ***Actions*** column and run the ***Install*** action.

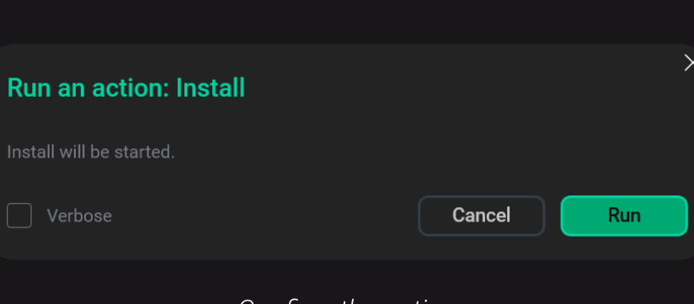


Clusters
Clusters / Test ADQM cluster / Services
21 Mar 2025 10:51:42 UTC
Test ADQM cluster
1
Services
Add services
2
3
Install

Name	Version	State	Concerns	Actions
ADQMDB	24.8.8.17_arenadata2	installed		
Monitoring	1.1.0	created		
Zookeeper	3.8.3_arenadata1	installed		

Switch to the service installation

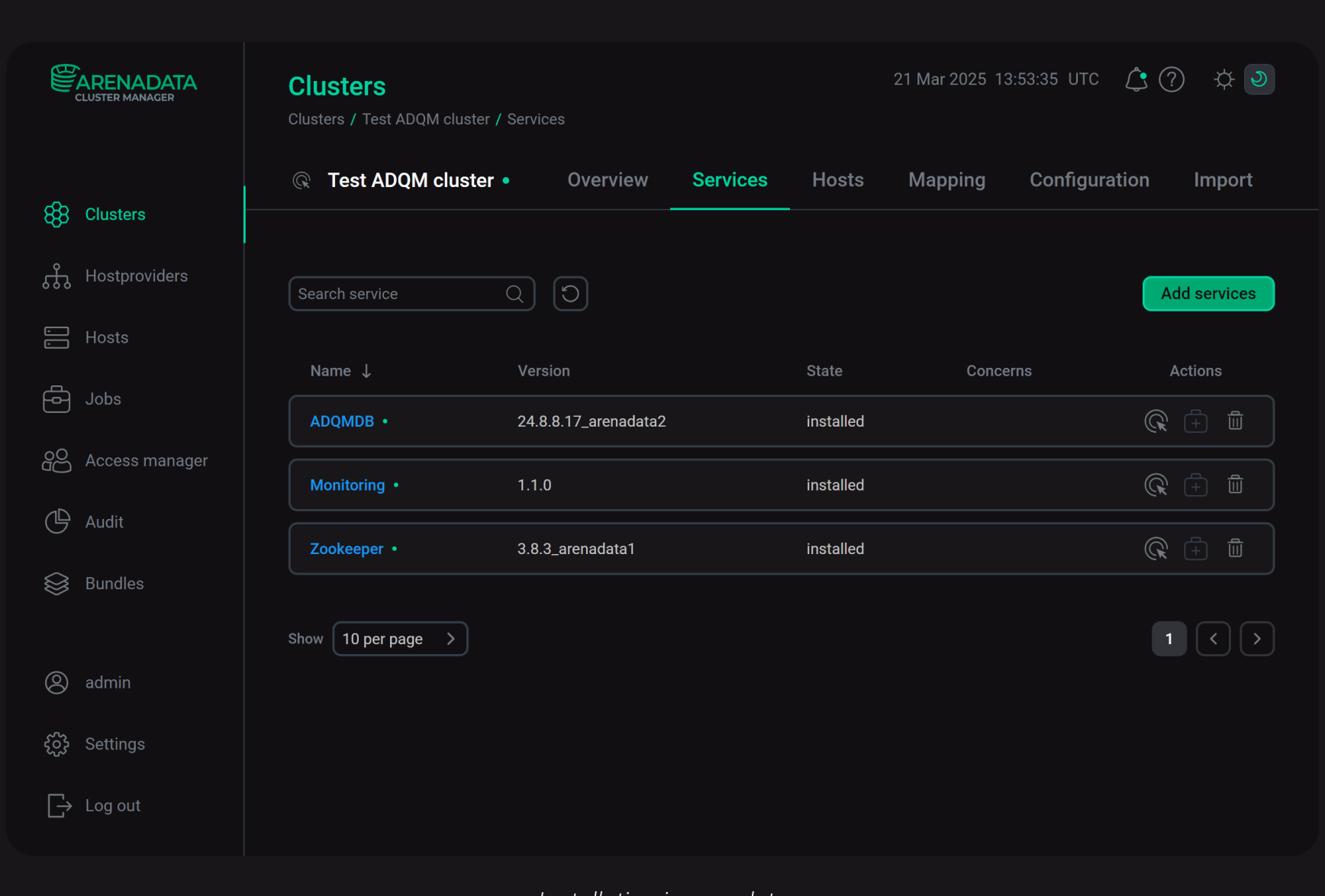
2. Confirm the action in the opened window.



Run an action: Install
Install will be started.
☐ Verbose
Cancel
Run

Confirm the action

3. Wait until the installation is completed. Then check that the service state has changed from `created` to `installed`.

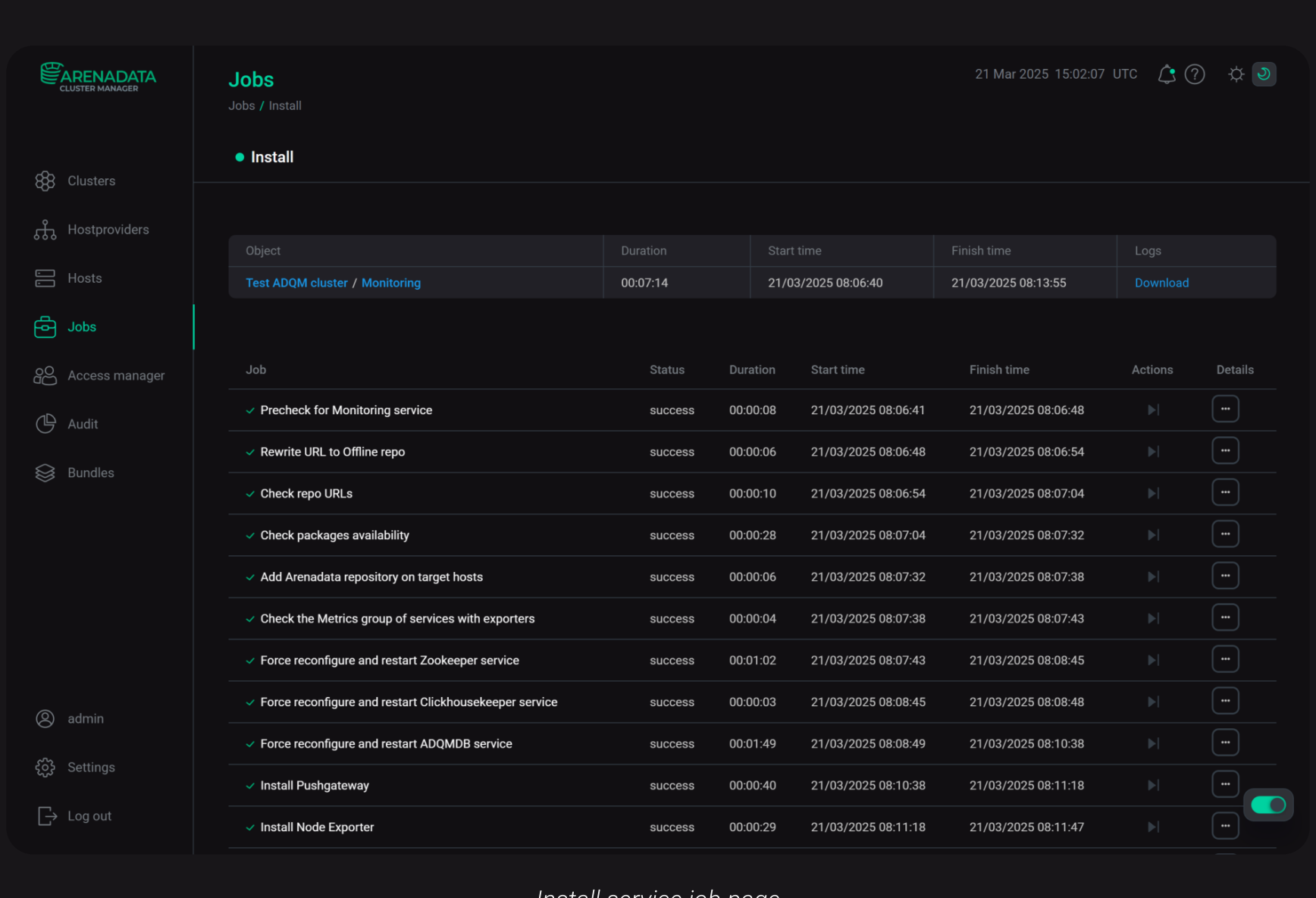


Clusters
Clusters / Test ADQM cluster / Services
21 Mar 2025 13:53:35 UTC
Test ADQM cluster
Services
Add services

Name	Version	State	Concerns	Actions
ADQMDB	24.8.8.17_arenadata2	installed		
Monitoring	1.1.0	installed		
Zookeeper	3.8.3_arenadata1	installed		

Installation is complete

To view the service installation process and analyze errors if they occur, select ***Jobs*** in the left navigation menu and click the ***Install*** job name in the ***Jobs*** list.



Jobs
Jobs / Install
21 Mar 2025 15:02:07 UTC
Install

Object	Duration	Start time	Finish time	Logs
Test ADQM cluster / Monitoring	00:07:14	21/03/2025 08:06:40	21/03/2025 08:13:55	Download

Job	Status	Duration	Start time	Finish time	Actions	Details
Precheck for Monitoring service	success	00:00:08	21/03/2025 08:06:41	21/03/2025 08:06:48		
Rewrite URL to Offline repo	success	00:00:06	21/03/2025 08:06:48	21/03/2025 08:06:54		
Check repo URLs	success	00:00:10	21/03/2025 08:06:54	21/03/2025 08:07:04		
Check packages availability	success	00:00:28	21/03/2025 08:07:04	21/03/2025 08:07:32		
Add Arenadata repository on target hosts	success	00:00:06	21/03/2025 08:07:32	21/03/2025 08:07:38		
Check the Metrics group of services with exporters	success	00:00:04	21/03/2025 08:07:38	21/03/2025 08:07:43		
Force reconfigure and restart Zookeeper service	success	00:01:02	21/03/2025 08:07:43	21/03/2025 08:08:45		
Force reconfigure and restart Clickhousekeeper service	success	00:00:03	21/03/2025 08:08:45	21/03/2025 08:08:48		
Force reconfigure and restart ADQMDB service	success	00:01:49	21/03/2025 08:08:49	21/03/2025 08:10:38		
Install Pushgateway	success	00:00:40	21/03/2025 08:10:38	21/03/2025 08:11:18		
Install Node Exporter	success	00:00:29	21/03/2025 08:11:18	21/03/2025 08:11:47		

Install service job page

Step 5. View results

The Monitoring service starts automatically after installation. To ensure that monitoring works correctly, check that both system and ADQM service metrics are collected from the cluster hosts — see ADQM monitoring metrics. To do this, view metrics in the Prometheus format in the browser or use the Prometheus or Grafana web interface.

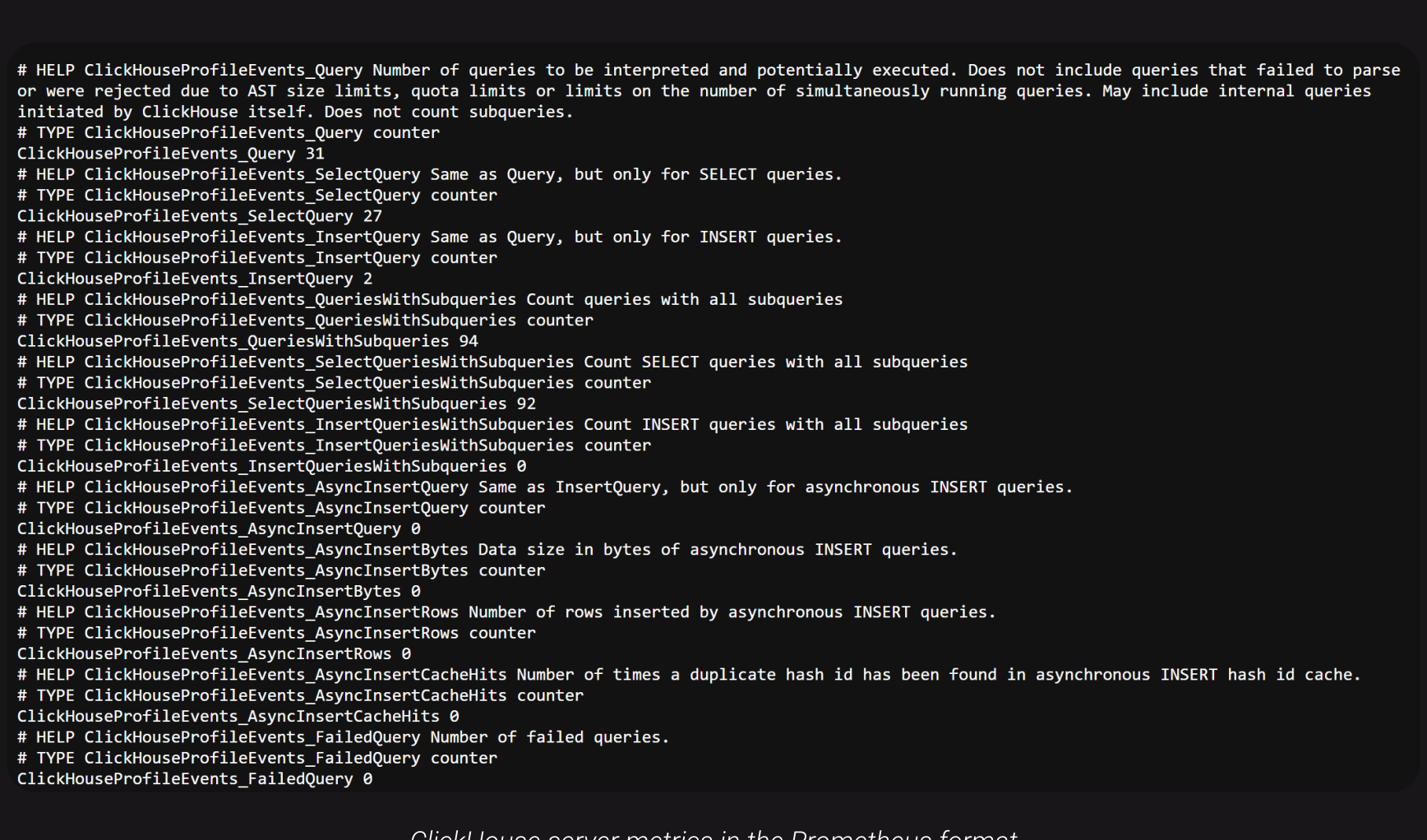
Metrics in the Prometheus format

1. In the address bar of your browser, enter an address of an ADQM host with the specified port and endpoint to listen for service or system metrics. Port numbers and endpoints are defined by the parameters located in:

   - the ***Metrics*** section on the configuration pages of the ADQMDB, Zookeeper, and Clickhousekeeper services — settings for access to monitoring metrics of these services;
   - the ***Node Exporter settings*** section on the configuration page of the Monitoring service — settings for access to system metrics of the ADQM cluster hosts.

   For example, *http://10.92.40.25:9363/metrics* is an address to view metrics of the ClickHouse server on a host with IP `10.92.40.25`.

2. The page that opens will display monitoring metrics from the specified host of the ADQM cluster in the Prometheus format.



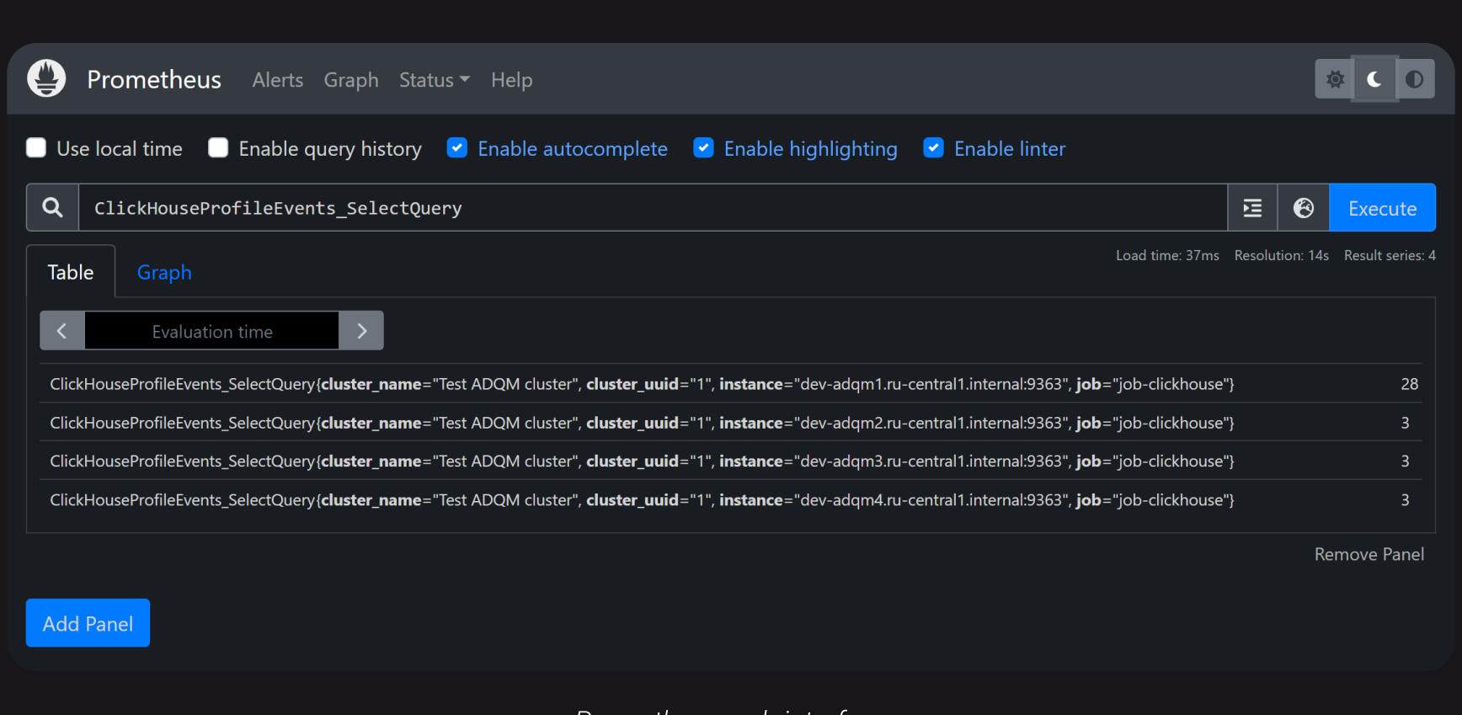
```
# HELP ClickHouseProfileEvents_Query Number of queries to be interpreted and potentially executed. Does not include queries that failed to parse
or were rejected due to AST size limits, quota limits or limits on the number of simultaneously running queries. May include internal queries
initiated by ClickHouse itself. Does not count subqueries.
# TYPE ClickHouseProfileEvents_Query counter
ClickHouseProfileEvents_Query 31
# HELP ClickHouseProfileEvents_SelectQuery Same as Query, but only for SELECT queries.
# TYPE ClickHouseProfileEvents_SelectQuery counter
ClickHouseProfileEvents_SelectQuery 27
# HELP ClickHouseProfileEvents_InsertQuery Same as Query, but only for INSERT queries.
# TYPE ClickHouseProfileEvents_InsertQuery counter
ClickHouseProfileEvents_InsertQuery 2
# HELP ClickHouseProfileEvents_QueriesWithSubqueries Count queries with all subqueries
# TYPE ClickHouseProfileEvents_QueriesWithSubqueries counter
ClickHouseProfileEvents_QueriesWithSubqueries 94
# HELP ClickHouseProfileEvents_SelectQueriesWithSubqueries Count SELECT queries with all subqueries
# TYPE ClickHouseProfileEvents_SelectQueriesWithSubqueries counter
ClickHouseProfileEvents_SelectQueriesWithSubqueries 92
# HELP ClickHouseProfileEvents_InsertQueriesWithSubqueries Count INSERT queries with all subqueries
# TYPE ClickHouseProfileEvents_InsertQueriesWithSubqueries counter
ClickHouseProfileEvents_InsertQueriesWithSubqueries 0
# HELP ClickHouseProfileEvents_AsyncInsertQuery Same as InsertQuery, but only for asynchronous INSERT queries.
# TYPE ClickHouseProfileEvents_AsyncInsertQuery counter
ClickHouseProfileEvents_AsyncInsertQuery 0
# HELP ClickHouseProfileEvents_AsyncInsertBytes Data size in bytes of asynchronous INSERT queries.
# TYPE ClickHouseProfileEvents_AsyncInsertBytes counter
ClickHouseProfileEvents_AsyncInsertBytes 0
# HELP ClickHouseProfileEvents_AsyncInsertRows Number of rows inserted by asynchronous INSERT queries.
# TYPE ClickHouseProfileEvents_AsyncInsertRows counter
ClickHouseProfileEvents_AsyncInsertRows 0
# HELP ClickHouseProfileEvents_AsyncInsertCacheHits Number of times a duplicate hash id has been found in asynchronous INSERT hash id cache.
# TYPE ClickHouseProfileEvents_AsyncInsertCacheHits counter
ClickHouseProfileEvents_AsyncInsertCacheHits 0
# HELP ClickHouseProfileEvents_FailedQuery Number of failed queries.
# TYPE ClickHouseProfileEvents_FailedQuery counter
ClickHouseProfileEvents_FailedQuery 0
```

ClickHouse server metrics in the Prometheus format

Prometheus web interface

1. In the address bar of your browser, enter an IP address of a host where the Prometheus Server component is installed. Specify also a port number from the ***web.listen-address*** parameter in the ***Service parameters*** group within the ***Prometheus settings*** section of the Monitoring service configuration (the default port is `11200`). For example: *http://10.92.40.107:11200*. To log into the Prometheus interface, use a username and password that are also specified in the ***Prometheus settings*** section — the ***Prometheus users to login/logout to Prometheus*** setting.

2. In the ***Expression*** field, you can enter a metric and click ***Execute*** — values of this metric on all hosts of the ADQM cluster will be shown in the interface.



Prometheus
Alerts
Graph
Status
Help
☐ Use local time
☐ Enable query history
☑ Enable autocomplete
☑ Enable highlighting
☑ Enable linter
Execute
Load time: 37ms
Resolution: 14s
Result series: 4
Table
Graph
Evaluation time
Remove Panel
Add Panel

```
ClickHouseProfileEvents_SelectQuery
```

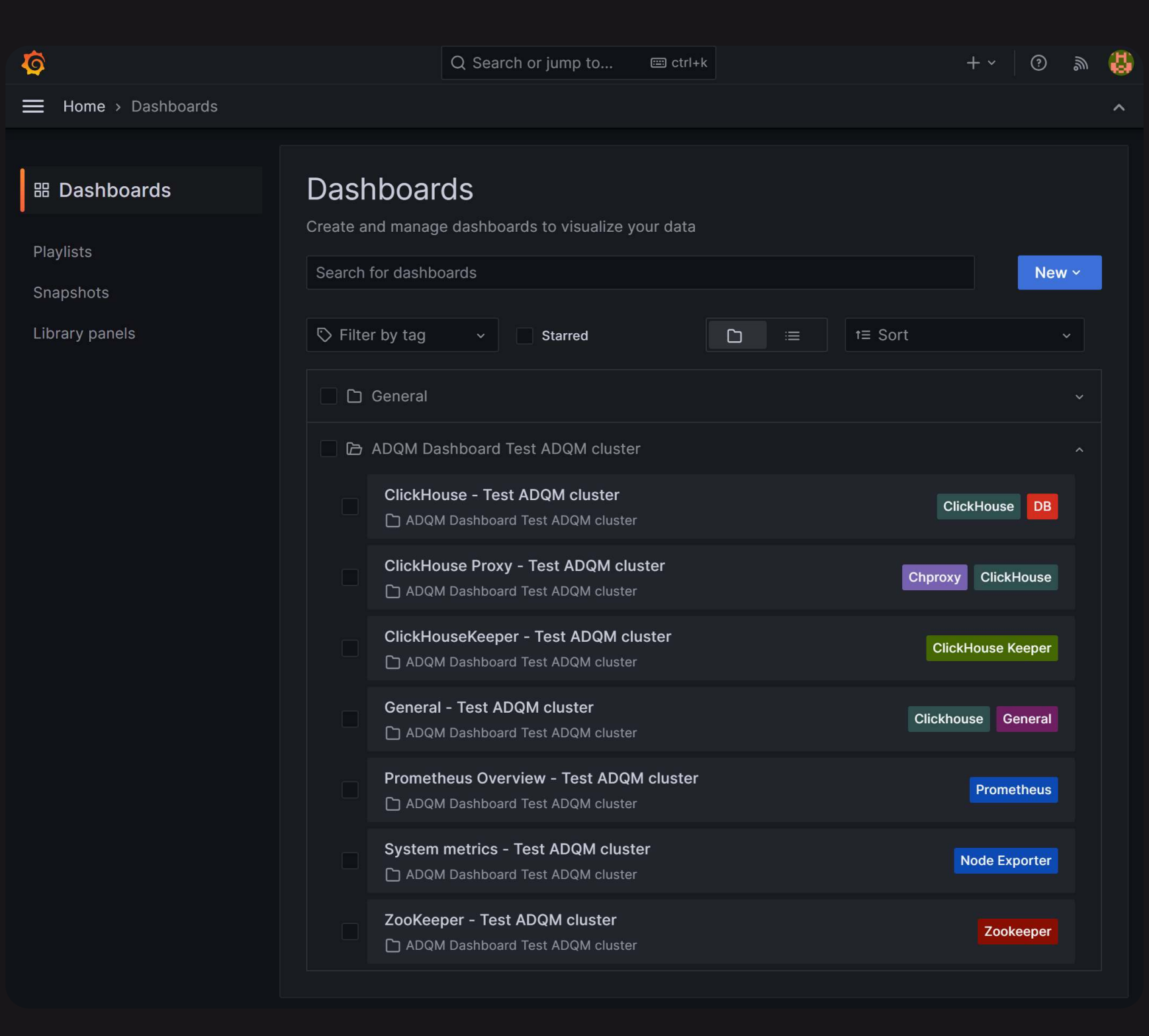
ClickHouseProfileEvents_SelectQuery{cluster_name="Test ADQM cluster", cluster_uuid="1", instance="dev-adqm1.ru-central1.internal:9363", job="job-clickhouse"}	28
ClickHouseProfileEvents_SelectQuery{cluster_name="Test ADQM cluster", cluster_uuid="1", instance="dev-adqm2.ru-central1.internal:9363", job="job-clickhouse"}	3
ClickHouseProfileEvents_SelectQuery{cluster_name="Test ADQM cluster", cluster_uuid="1", instance="dev-adqm3.ru-central1.internal:9363", job="job-clickhouse"}	3
ClickHouseProfileEvents_SelectQuery{cluster_name="Test ADQM cluster", cluster_uuid="1", instance="dev-adqm4.ru-central1.internal:9363", job="job-clickhouse"}	3

Prometheus web interface

Grafana web interface

1. In the address bar of your browser, enter an address of a host on which Grafana is deployed and add a port number — a value of the ***Grafana listen port*** parameter located in the ***Grafana settings*** section of the Monitoring service configuration (the default value is `11210`). For example, *http://10.92.40.107:11210*. To log in, use `admin` as a username, and the ***Grafana administrator's password*** parameter value (also found in the ***Grafana settings*** section of service configuration parameters) as a password.

2. In the window that opens, navigate to ***Home*** → ***Dashboards*** and expand the ***ADQM Dashboard <ADQM_cluster_name>*** section. In this section, you can select one of available dashboards to view service or system metrics coming from your ADQM cluster.



Search or jump to...
ctrl+k
Home
Dashboards
Dashboards
Playlists
Snapshots
Library panels
Dashboards
Create and manage dashboards to visualize your data
Search for dashboards
New
Filter by tag
Starred
Sort
General
ADQM Dashboard Test ADQM cluster
ClickHouse - Test ADQM cluster
ClickHouse
DB
ClickHouse Proxy - Test ADQM cluster
Chproxy
ClickHouse
ClickHouseKeeper - Test ADQM cluster
ClickHouse Keeper
General - Test ADQM cluster
Clickhouse
General
Prometheus Overview - Test ADQM cluster
Prometheus
System metrics - Test ADQM cluster
Node Exporter
ZooKeeper - Test ADQM cluster
Zookeeper
ADQM Dashboard Test ADQM cluster

View ADQM metrics in Grafana

Offline installation

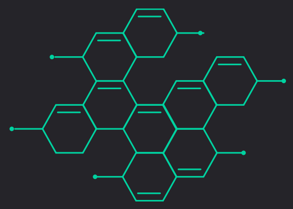
To Table of Contents

This topic describes **offline** installation of ADQM. It can be applied to the hosts without access to Internet.

For offline installation, you need to install and configure Arenadata Enterprise Tools (ET). ET is a solution that provides the necessary infrastructure for Arenadata product deployment in environments with limited internet access. The solution is a part of the Arenadata data platform. It provides the repositories necessary for installing and updating the platform components.

The process of offline installation via ADCM includes the following steps.

> **NOTE**
>
> This function is available only in the Enterprise Edition of ADQM.



Step 1.

Install ADCM

Step 2.

Prepare hosts

Step 3.

Install ET cluster

ARENADATA
QUICKMARTS

Step 4.

Install ADQM cluster

Step 5.

Install monitoring

ARENADATA
ADQM CONTROL

Step 6.

Install ADQM Control

Install ADCM

Daria Barysheva

Contents

To Table of Contents

- Step 1. Check prerequisites
  - Software and hardware requirements
  - Install Docker
  - Disable SELinux
  - Stop firewalld
  - Allow access to the Docker CLI without root privileges
- Step 2. Install ADCM
- Step 3. Start ADCM
- Step 4. Check installation
  - Check the Docker container status
  - Check the 8000 port availability
  - Check the ADCM web-interface
  - Set the ADCM URL
- Step 5. Update ADCM

ADCM requires a separate host. Before installing ADCM, we recommend to read its documentation.

The installation steps are listed below.

Step 1. Check prerequisites

Software and hardware requirements

ADCM is a software that is distributed in the form of a Docker image. To install it, you need:

- One of the following operating systems:
  - Ubuntu 22.04
  - CentOS 7
  - RHEL 7
  - Alt Linux SP 8
  - Alt Linux SP 10.2
  - Astra Linux SE 1.7 Orel
  - Astra Linux 1.7.6 Voronezh
  - RED OS 7.3 Certified edition
- A user account with `sudo` privileges.
- An access to the official repository of CentOS Extras and CentOS Base. This repository is enabled by default, but if you have disabled it — you need to re-enable it (e.g. via editing the repository configuration file in the */etc/yum.repos.d/* directory).
- The software package installer YUM/APT.
- The hardware that meets the following conditions:

> IMPORTANT
>
> The following system requirements are **minimal**. The target sizing should be calculated based on the customer requirements.

  - **CPU**: 4 CPU cores;
  - **RAM**: 16 GB;
  - **HDD**: 50 GB.

> IMPORTANT
>
> Starting with ADCM 2.7.0, any Container Registry Client with downloaded images of ADPG, ADCM, and ADCM Installer will be required to install ADPG without the Internet connection and use it as the ADCM database. The installation is done via the ADCM Installer by specifying the URL of the local storage of the ADCM and ADPG images.

To grant the user the necessary rights to install ADCM, do the following:

1. Grant the `root` privileges for the user:

CentOS 7/RHEL 7/RED OS 7.3/Alt Linux SP 8

```
$ sudo usermod -aG wheel <user>
```

where `<user>` is a username.

Ubuntu 22.04/Astra Linux SE 1.7 Orel

```
$ sudo usermod -aG sudo <user>
```

where `<user>` is a username.

2. Make sure that the */etc/sudoers* file contains the record specified below depending on the operating system with the following command:

```
$ sudo vi /etc/sudoers
```

To return to the command line, press `Esc`, type `:q!`, and press `Enter`.

CentOS 7/RHEL 7/RED OS 7.3

```
## Allows people in group wheel to run all commands
%wheel  ALL=(ALL)       ALL
```

Ubuntu 22.04/Astra Linux SE 1.7 Orel

```
# Allow members of group sudo to execute any command
%sudo   ALL=(ALL:ALL) ALL
```

Alt Linux SP 8

```
## Uncomment to allow members of group wheel to execute any command
WHEEL_USERS ALL=(ALL:ALL) ALL
```

Install Docker

> CAUTION
>
> Do not install Docker from the official Docker site. Instead, use the official repositories of your Linux distribution. The repositories provided by operating system developers tend to be more stable and secure.

Below are the step-by-step procedures of Docker installation on the examples of several operating systems supported by ADCM. These steps may differ for operating system versions other than those specified on tabs.

CentOS 7/RHEL 7

1. Using `root` privileges, update all packages that are currently installed in your system:

```
$ sudo yum update -y
```

2. Install the packages required for Docker installation and usage:

```
$ sudo yum install -y yum-utils docker device-mapper-persistent-data lvm2
```

where:

- `yum-utils` is the package that provides the `yum-config-manager` utility;
- `device-mapper-persistent-data` and `lvm2` are packages required by the devicemapper storage driver.

3. Start Docker:

```
$ sudo systemctl start docker
```

4. Enable Docker as a system service:

```
$ sudo systemctl enable docker
```

Ubuntu 22.04

1. Using `root` privileges, update all packages that are currently installed in your system:

```
$ sudo apt-get update
```

2. Install the packages required for Docker installation and usage:

```
$ sudo apt-get install ca-certificates curl
```

where:

- `ca-certificates` is the package of Root CA certificates;
- `curl` is the package for working with web resources.

Confirm the use of additional disk space during the execution of the command.

3. Add the official GPG key for the Docker repository:

```
$ sudo install -m 0755 -d /etc/apt/keyrings
$ sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.asc
$ sudo chmod a+r /etc/apt/keyrings/docker.asc
```

4. Add the Docker repository:

```
$ echo \
"deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc] https://download.docker.com/linux/ubuntu \
$(. /etc/os-release && echo "jammy") stable" | \
sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

5. Update all packages, including packages for the Docker repository:

```
$ sudo apt-get update
```

6. Install Docker Engine:

```
$ sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin
```

Confirm the use of additional disk space during the execution of the command.

7. Start Docker:

```
$ sudo systemctl start docker
```

8. Enable Docker as a system service:

```
$ sudo systemctl enable docker
```

Alt Linux SP 8

1. Using `root` privileges, update all packages that are currently installed in your system:

```
$ sudo apt-get update
```

2. Install Docker Engine:

```
$ sudo apt-get install docker-engine
```

Confirm the use of additional disk space during the execution of the command.

3. Start and enable Docker as a system service:

```
$ sudo systemctl enable --now docker
```

Astra Linux SE 1.7 Orel

1. Using `root` privileges, update all packages that are currently installed in your system:

```
$ sudo apt-get update
```

2. Install the packages required for Docker installation and usage:

```
$ sudo apt-get install ca-certificates curl net-tools
```

where:

- `ca-certificates` is the package of Root CA certificates;
- `curl` is the package for working with web resources;
- `net-tools` is the package for controlling the network subsystem of the Linux kernel, in particular for executing the `netstat` command.

Confirm the use of additional disk space during the execution of the command.

3. Add the official GPG key for the Docker repository:

```
$ curl -fsSL https://download.docker.com/linux/debian/gpg | sudo apt-key add -
```

4. Add the Docker repository:

```
$ echo "deb [arch=amd64] https://download.docker.com/linux/debian stretch stable" | sudo tee -a /etc/apt/sources.list
```

5. Update all packages, including packages for the Docker repository:

```
$ sudo apt-get update
```

6. Install Docker Engine:

```
$ sudo apt install docker-ce docker-ce-cli containerd.io
```

Confirm the use of additional disk space during the execution of the command.

7. Start Docker:

```
$ sudo systemctl start docker
```

8. Enable Docker as a system service:

```
$ sudo systemctl enable docker
```

RED OS 7.3

1. Install Docker Engine:

```
$ sudo dnf install docker-ce docker-ce-cli
```

Confirm the use of additional disk space during the execution of the command.

2. Start and enable Docker as a system service:

```
$ sudo systemctl enable docker --now
```

Disable SELinux

> NOTE
>
> It is recommended to disable SELinux only if CentOS 7 or RHEL 7 is used.

To disable SELinux permanently, set `SELINUX=disabled` in the */etc/selinux/config* file, and reboot your system.

You can edit the file via the `vi` or `vim` command:

```
$ sudo vi /etc/selinux/config
```

The content of the changed file should look like this:

```
# This file controls the state of SELinux on the system.
# SELINUX= can take one of these three values:
#       enforcing - SELinux security policy is enforced.
#       permissive - SELinux prints warnings instead of enforcing.
#       disabled - SELinux is fully disabled.
SELINUX=disabled
# SELINUXTYPE= type of policy in use. Possible values are:
#       targeted - Only targeted network daemons are protected.
#       strict - Full SELinux protection.
SELINUXTYPE=targeted
```

You can also disable SELinux temporarily via the following command:

```
$ sudo setenforce 0
```

Still, it is strongly recommended to disable SELinux permanently, so that it does not restart on each system reboot.

> CAUTION
>
> Do not forget to reboot your host after SELinux is disabled.

Stop firewalld

If you use the firewalld service, stop it before installing ADCM.

Before stopping firewalld, make sure that it is installed, started, and enabled:

```
$ sudo systemctl status firewalld
```

The example of the command execution result:

```
● firewalld.service - firewalld - dynamic firewall daemon
   Loaded: loaded (/usr/lib/systemd/system/firewalld.service; enabled; vendor preset: enabled)
   Active: active (running) since Mon 2024-12-02 10:02:57 UTC; 3min 25s ago
     Docs: man:firewalld(1)
 Main PID: 3023 (firewalld)
    Tasks: 2
   Memory: 22.8M
   CGroup: /system.slice/firewalld.service
           └─3023 /usr/bin/python2 -Es /usr/sbin/firewalld --nofork --nopid
```

Stop firewalld:

```
$ sudo systemctl stop firewalld
```

As an alternative, you can disable the firewalld service, so that it will not apply rules to network packets:

```
$ sudo systemctl disable firewalld
```

Allow access to the Docker CLI without root privileges

If in the future you want to access the Docker CLI without using `root` privileges, you should create a `docker` user group (if it does not exist) and add the current user to this group with the certain rights:

```
$ sudo groupadd docker
$ sudo usermod -a -G docker $USER
```

After running these commands, you should re-login under the current user.

Step 2. Install ADCM

Offline ADCM installation includes the following steps:

1. Get an archive with the *sh.xz* extension that contains a special installation shell script from the Arenadata support team.
2. Make a directory on the host where ADCM should be deployed, and put the received archive into it.
3. Unpack the archive into the created directory using any of the available methods.

> NOTE
>
> Running the command can take some time. Please wait until it is completed.

4. Load the Docker image to the local storage using the received script:

```
$ sudo bash <script_name> unpack_adcm
```

where `<script_name>` is the full script name with the extension (for example, *adcm_2.5.0.sh*).

The example of a successful command execution result:

```
Loaded image: hub.arenadata.io/adcm/adcm:2.5.0
```

5. Create a new container based on the downloaded image:

```
$ sudo docker create --name adcm -p 8000:8000 -v /opt/adcm:/adcm/data -e DB_HOST="<DATABASE_HOSTNAME_OR_IP_ADDRESS>" -e DB_PORT="<DATABASE_TCP_PORT>" -e DB_USER="<DATABASE_USERNAME>" -e DB_NAME="<DATABASE_NAME>" -e DB_PASS="<DATABASE_USER_PASSWORD>" -e DB_OPTIONS="<DATABASE_OPTIONS>" hub.arenadata.io/adcm/adcm:<version>
```

Step 3. Start ADCM

ADCM is just a container operated by Docker. So you are able to use simple Docker commands to start and stop ADCM:

- To **start ADCM**, use the following command.

```
$ sudo docker start adcm
```

- To **stop ADCM**, use the following command.

```
$ sudo docker stop adcm
```

- In order to provide the **automatic start** of the Docker container after unexpected errors, use the following command.

```
$ sudo docker update --restart=on-failure adcm
```

> NOTE
>
> - For more information about Docker restart policies, refer to the Docker documentation.
> - If you allowed your user to access the Docker CLI without `root` privileges, then you can omit the `sudo` keyword from the commands listed above.

Step 4. Check installation

Check the Docker container status

To make sure that the Docker container `adcm` is successfully deployed, run the following command:

```
$ sudo docker container ls
```

The example of the command execution result is listed below. Make sure that the `STATUS` value is equal to `Up`.

```
CONTAINER ID        IMAGE                               COMMAND            CREATED            STATUS             PORTS
NAMES
74b5628146e5        hub.arenadata.io/adcm/adcm:2.0   "/etc/startup.sh"   4 days ago          Up 21 minutes       0.0.0.0:8000-
>8000/tcp   adcm
```

Check the 8000 port availability

After ADCM is installed and started, its web interface should become available on the `8000` port of the deployed container.

For Ubuntu 22.04, first install the packages required for executing the `netstat` command:

```
$ sudo apt-get install net-tools
```

You can check the 8000 port availability using these commands:

- The `netstat` command — to display information about port listening.

```
$ sudo netstat -ntpl | grep 8000
```

The example of the command execution result:

```
tcp6       0      0 :::8000                 :::*                    LISTEN
1514/docker-proxy-c
```

- The `curl` command — to check the URL connection.

```
$ curl http://localhost:8000
```

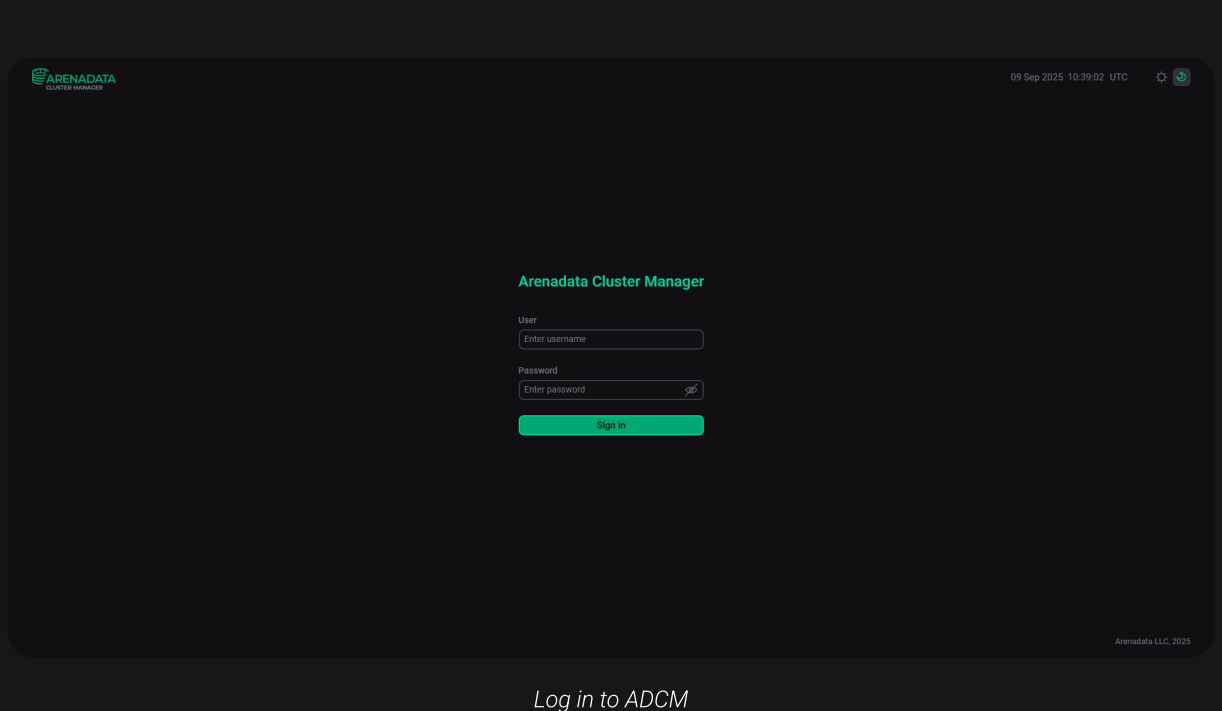
The example of the command execution result:

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Arenadata Cluster Manager</title>
  <base href="/">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="assets/favicon.ico">
<link rel="stylesheet" href="styles.10db6328264e0907c52f.css"></head>
<body>
  <app-root></app-root>
<script src="runtime-es2015.7eb406ed18bf0258cd35.js" type="module"></script><script src="runtime-es5.7eb406ed18bf0258cd35.js"
nomodule defer></script><script src="polyfills-es5.2e224d70daec4412d3c2.js" nomodule defer></script><script src="polyfills-es2015.
37e0553ac06970d6a5b5.js" type="module"></script><script src="main-es2015.39851da0ebf9ed6fec45.js" type="module"></script><script
src="main-es5.39851da0ebf9ed6fec45.js" nomodule defer></script></body>
</html>
```

Check the ADCM web-interface

To log in to ADCM, use the user with administrator rights that is created by default:

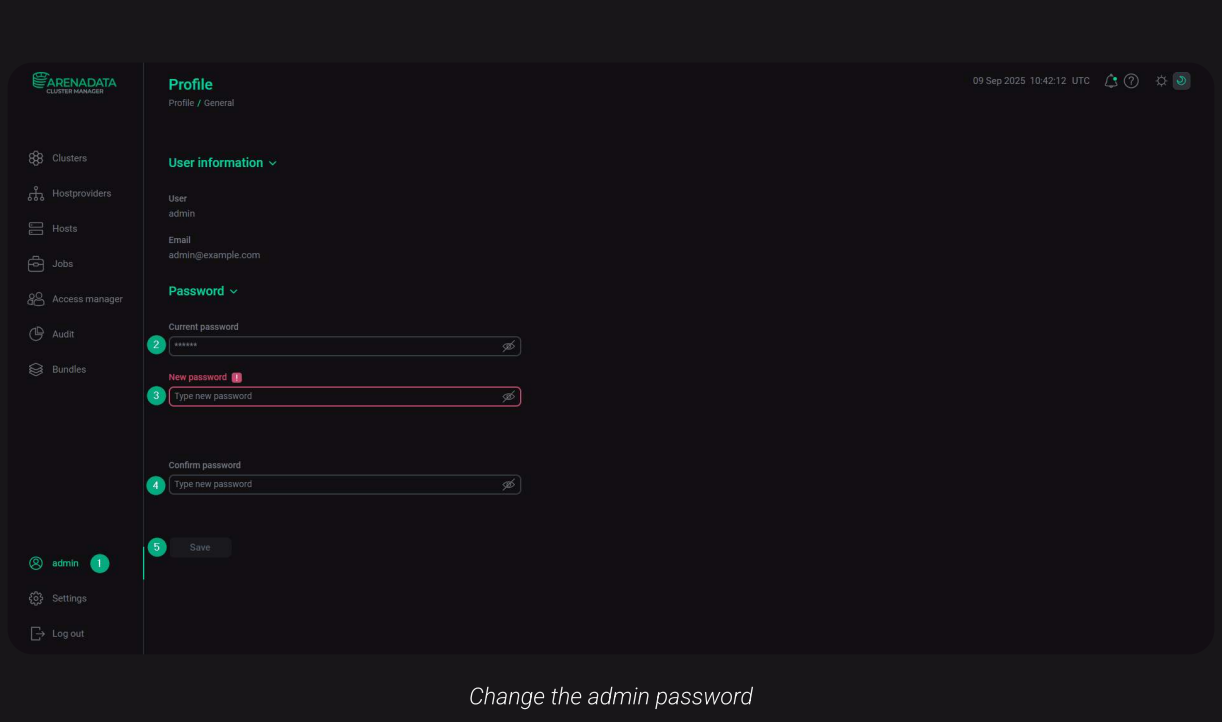
- ***User***: admin
- ***Password***: admin



Log in to ADCM

Later you can change the password if necessary. To do this:

1. Open the user profile in the left navigation menu.
2. Enter the current password in the ***Current password*** field.
3. Fill in the ***New password*** field.
4. Confirm a new password in the ***Confirm password*** field.
5. Click ***Save***.

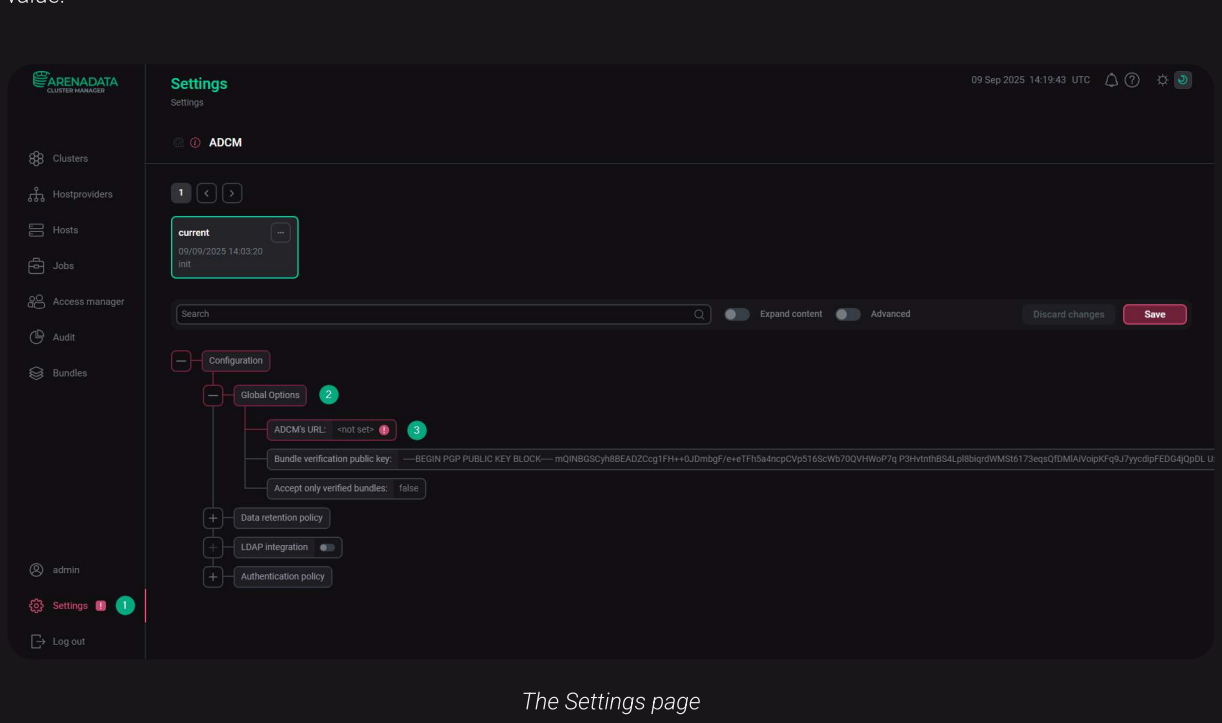


Change the admin password

Set the ADCM URL

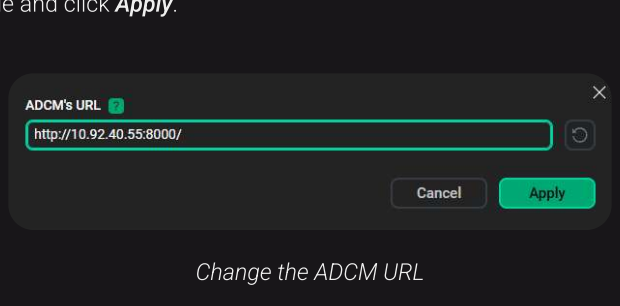
For successful communication between ADCM and product clusters deployed via ADCM, it is important to correctly define the external ADCM address. This address is used by cluster components for sending information about their state to ADCM. To specify the ADCM URL, use the environment variable `DEFAULT_ADCM_URL` when creating the ADCM container. If you did not specify that variable, you should manually install the URL in the ADCM web interface. To do this, follow the steps:

1. Select the ***Settings*** item in the left navigation menu.
2. Expand the ***Global Options*** node in the tree of configuration settings.
3. Click the ***ADCM's URL*** field value.



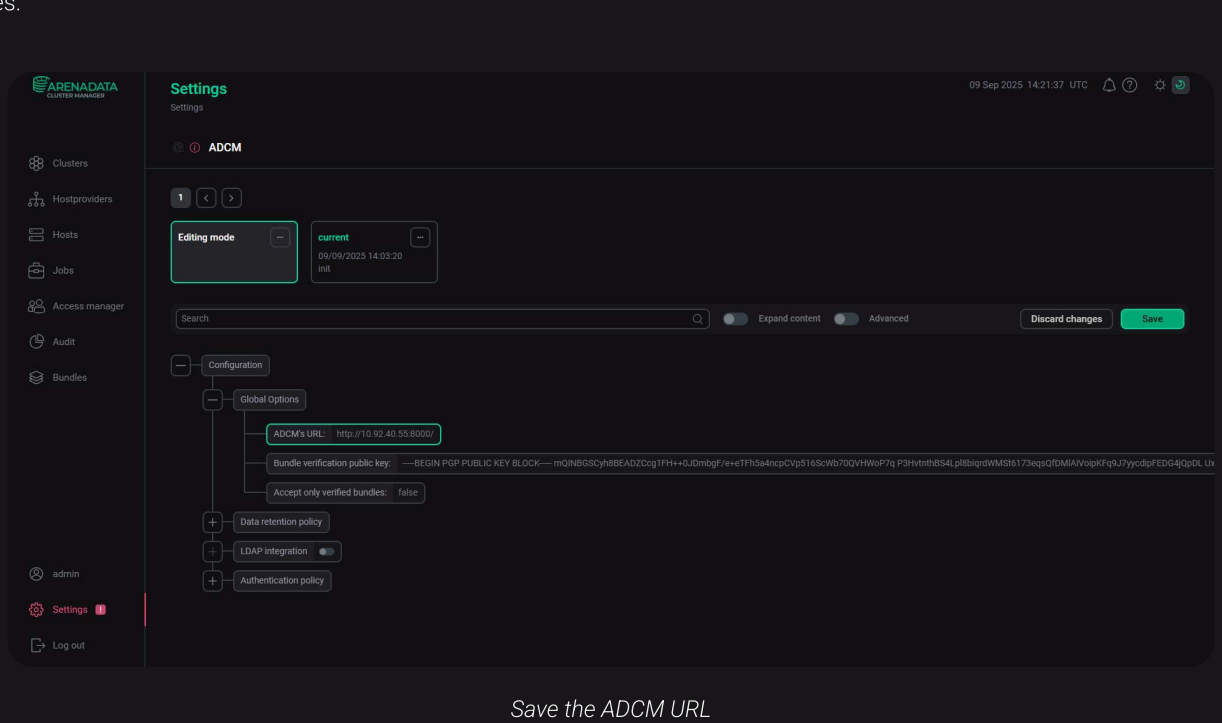
The Settings page

4. In the window that opens, edit the ***ADCM's URL*** field value and click ***Apply***.


ADCM's URL: http://10.92.40.55:8000/

Change the ADCM URL

5. Click ***Save*** to apply changes.



Save the ADCM URL

Step 5. Update ADCM

> CAUTION
>
> - Before upgrading, it is strongly recommended to back up ADCM data. For more information, see Backup and restore.
> - Downgrade of the ADCM version is not supported.
> - It is not recommended to update ADCM more than 5 minor versions ahead (see `<minor>` in the versioning format below).
> - Before upgrading ADCM to version 2.7.0, migrate data from SQLite to PostgreSQL. Migration can be performed in any ADCM version (up to 2.6.0).

To upgrade the previously installed ADCM version offline, follow these steps:

1. Stop the `adcm` container:

```
$ sudo docker stop adcm
```

2. Remove the `adcm` container:

```
$ sudo docker container rm adcm
```

> NOTE
>
> For customers who use Enterprise editions of products, it is possible to contact the Arenadata support team to get an archive with the *sh.xz* extension that contains a special installation shell script. After stopping the `adcm` container, the ADCM update process includes the installation according to the instruction.

3. Download the required Docker image from the Arenadata Docker Registry on a host that has Internet access:

```
$ sudo docker pull hub.arenadata.io/adcm/adcm:<version>
```

where `<version>` is a desired ADCM image version in one of the following formats:

- `<major>.<minor>.<patch>` — if you need a specific ADCM patch. For example, `2.0.0`.
- `<major>.<minor>` — if you need the last patch within the selected ADCM version. For example, `2.0`.

> IMPORTANT
>
> - `latest` was used for earlier versions of ADCM and is no longer supported starting from version `2.0.0`.
> - Before ADCM 2.0.0, the following format was used for versioning: `YYYY.MM.DD.HH`.

4. Save the Docker image to a *.tar* file:

```
$ sudo docker save -o <file_name> hub.arenadata.io/adcm/adcm:<version>
```

where `<file_name>` is the full name of the created file with the extension (for example, *adcm_2.5.0_arenadata.tar*).

> NOTE
>
> Running the command can take some time. Please wait until it is completed.

5. Copy the saved file from the current host to a host without Internet connection using any available methods.

> IMPORTANT
>
> Make sure you have configured access to hosts and necessary file permissions.

6. Load the Docker image from the *.tar* file:

```
$ sudo docker load -i <file_path>
```

where `<file_path>` is the absolute path to the *.tar* file with the Docker image.

The example of a successful command execution result:

```
Loaded image: hub.arenadata.io/adcm/adcm:2.5.0
```

7. Create a new container based on the downloaded image:

```
$ sudo docker create --name adcm -p 8000:8000 -v /opt/adcm:/adcm/data -e DB_HOST="<DATABASE_HOSTNAME_OR_IP_ADDRESS>" -e DB_PORT="<DATABASE_TCP_PORT>" -e DB_USER="<DATABASE_USERNAME>" -e DB_NAME="<DATABASE_NAME>" -e DB_PASS="<DATABASE_USER_PASSWORD>" -e DB_OPTIONS="<DATABASE_OPTIONS>" hub.arenadata.io/adcm/adcm:<version>
```

> NOTE
>
> For more information on the environment variables (`DB_HOST`, `DB_PORT`, `DB_USER`, `DB_NAME`, `DB_PASS`, and `DB_OPTIONS`), see the Installation article.

8. Start the new `adcm` container:

```
$ sudo docker start adcm
```

Prepare hosts

To manage a cluster and its nodes via ADCM, it is necessary to prepare hosts for a cluster and link these hosts to ADCM. This is the function of ADCM **hostproviders** — special plugins for ADCM, which encapsulate the complex mechanism of interaction between ADCM and the hosts controlled by it.

ADCM supports several types of hostproviders. For more information on how to configure hostproviders and use them to create hosts, refer to the following documentation:

- SSH
- Yandex Cloud
- VCloud (VMware)
- Cloud.ru Advanced
- CROC Cloud
- ADQM Cloud Bundle

For offline installation, the SSH hostprovider is recommended.

CAUTION

If you are going to use Kerberos authentication in ADQM, you should specify FQDNs when creating ADQM hosts.

Create a cluster

Daria Barysheva

Contents

To Table of Contents

Step 1. Get the Enterprise version of a cluster bundle

Step 2. Upload a cluster bundle to ADCM

Step 3. Create a cluster based on the uploaded bundle

In ADCM a **cluster** means a set of services running on a set of hosts.

Clusters distributions for ADCM are made in the form of so-called bundles. Generally a bundle is a regular archive that includes a description and software logic of cluster deployment and exploitation.

The steps for adding a cluster to ADCM are listed below.

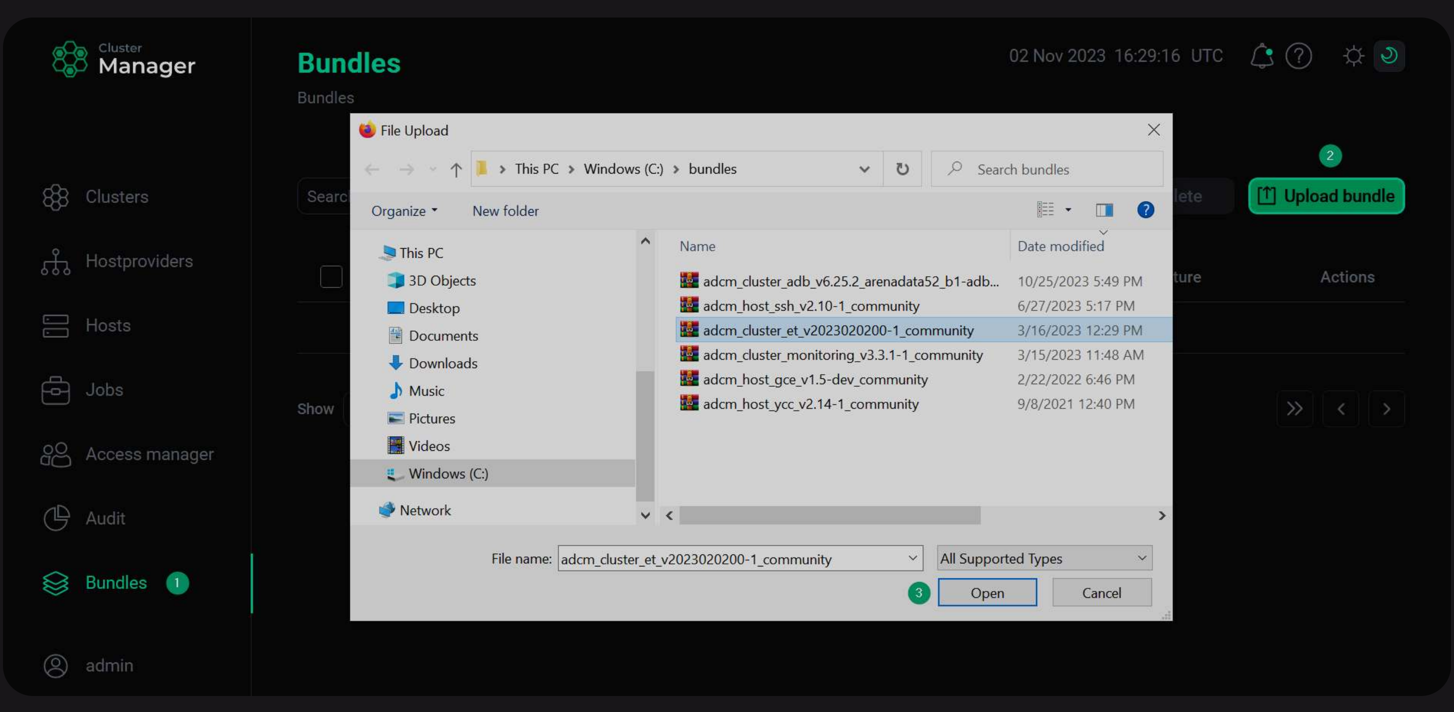
Step 1. Get the Enterprise version of a cluster bundle

Please get a cluster bundle from the Arenadata support team.

Step 2. Upload a cluster bundle to ADCM

To upload a bundle to ADCM, follow the steps:

1. Select the ***Bundles*** item in the left navigation menu and click ***Upload bundle***.
2. Select a bundle in the Open File Dialog.

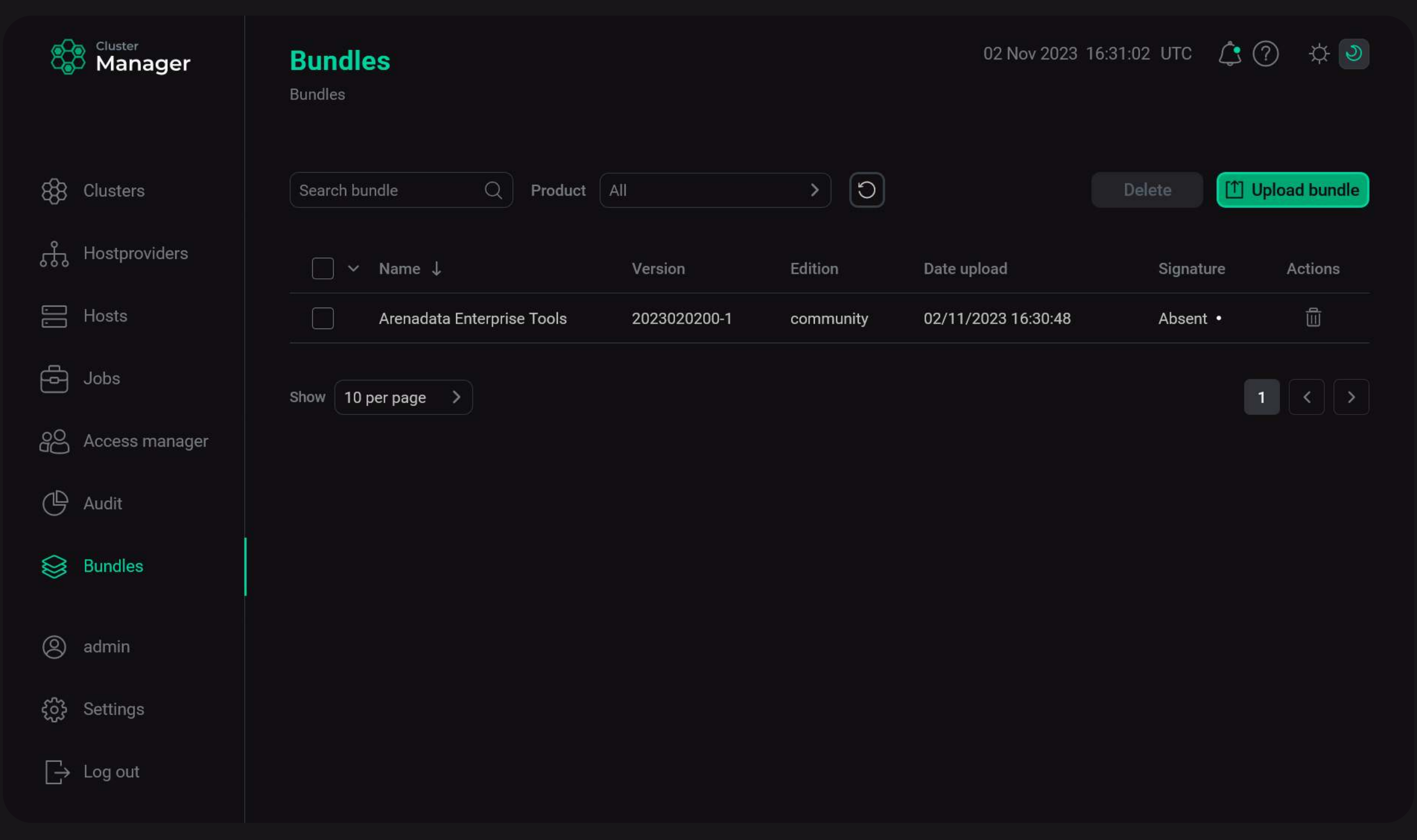


Manager
Bundles
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
File Upload
Upload bundle
Open
Cancel

File name: adcm_cluster_et_v2023020200-1_community

Upload a bundle

3. As a result of the performed actions, a bundle is displayed on the ***Bundles*** page.



Manager
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Bundles
02 Nov 2023 16:31:02 UTC
Search bundle
Product
All
Delete
Upload bundle
Show
10 per page

Name	Version	Edition	Date upload	Signature	Actions
Arenadata Enterprise Tools	2023020200-1	community	02/11/2023 16:30:48	Absent •	

The result of successful uploading a bundle

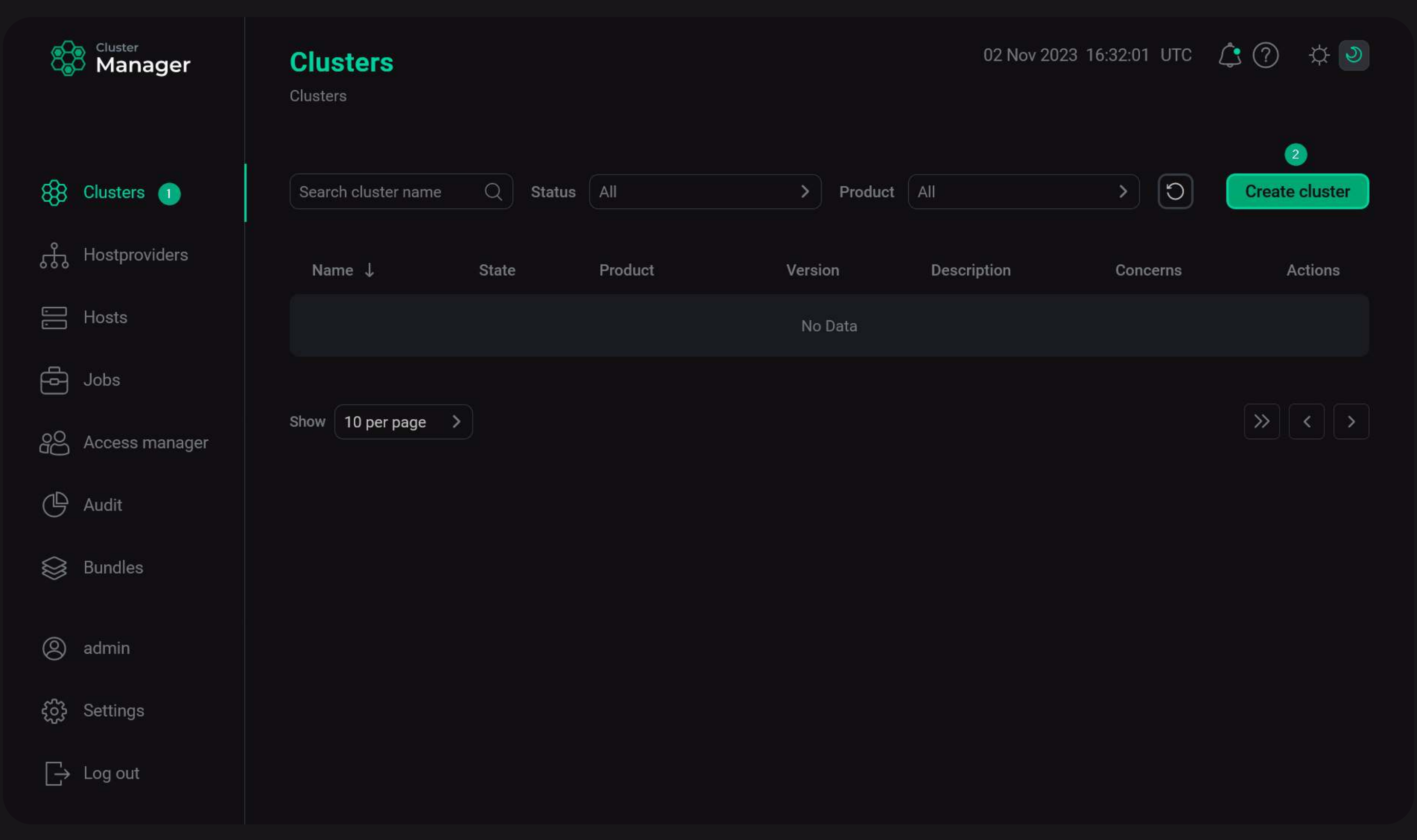
Step 3. Create a cluster based on the uploaded bundle

IMPORTANT

When creating a cluster via the ADCM web interface, you generate a new ADCM cluster instance. It means only adding the cluster to ADCM — at this stage no services are deployed.

The steps for cluster creation are given below:

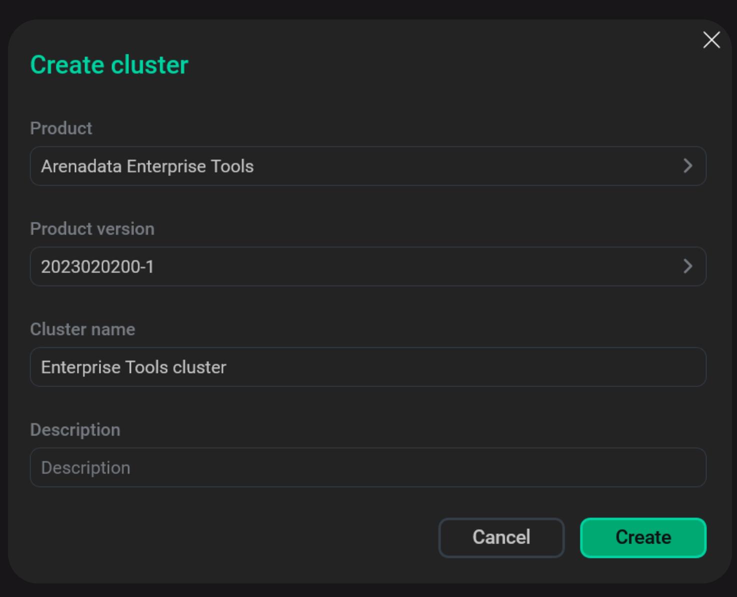
1. Select the ***Clusters*** item in the left navigation menu and click ***Create cluster***.



Manager
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Clusters
02 Nov 2023 16:32:01 UTC
Search cluster name
Status
All
Product
All
Create cluster
Name
State
Product
Version
Description
Concerns
Actions
No Data
Show
10 per page

Switch to cluster creation

2. In the opened dialog, describe a new cluster instance:
   - Select an uploaded cluster bundle in the ***Product*** field.
   - Select a bundle version in the ***Product version*** field. Several versions become available in the case of different versions of the same bundle being uploaded.
   - Enter a cluster name in the ***Cluster name*** field.
   - Enter a cluster description in the ***Description*** field if necessary.
   - Set the ***I accept Terms of Agreement*** flag to sign the license agreement. This flag is displayed only for Enterprise bundles. To read the agreement text, refer to the ***Terms of Agreement*** link. You can also sign the license agreement on the ***Bundles*** page by clicking the desired product bundle.
   - Click ***Create***.



Create cluster
Cancel
Create

Product: Arenadata Enterprise Tools

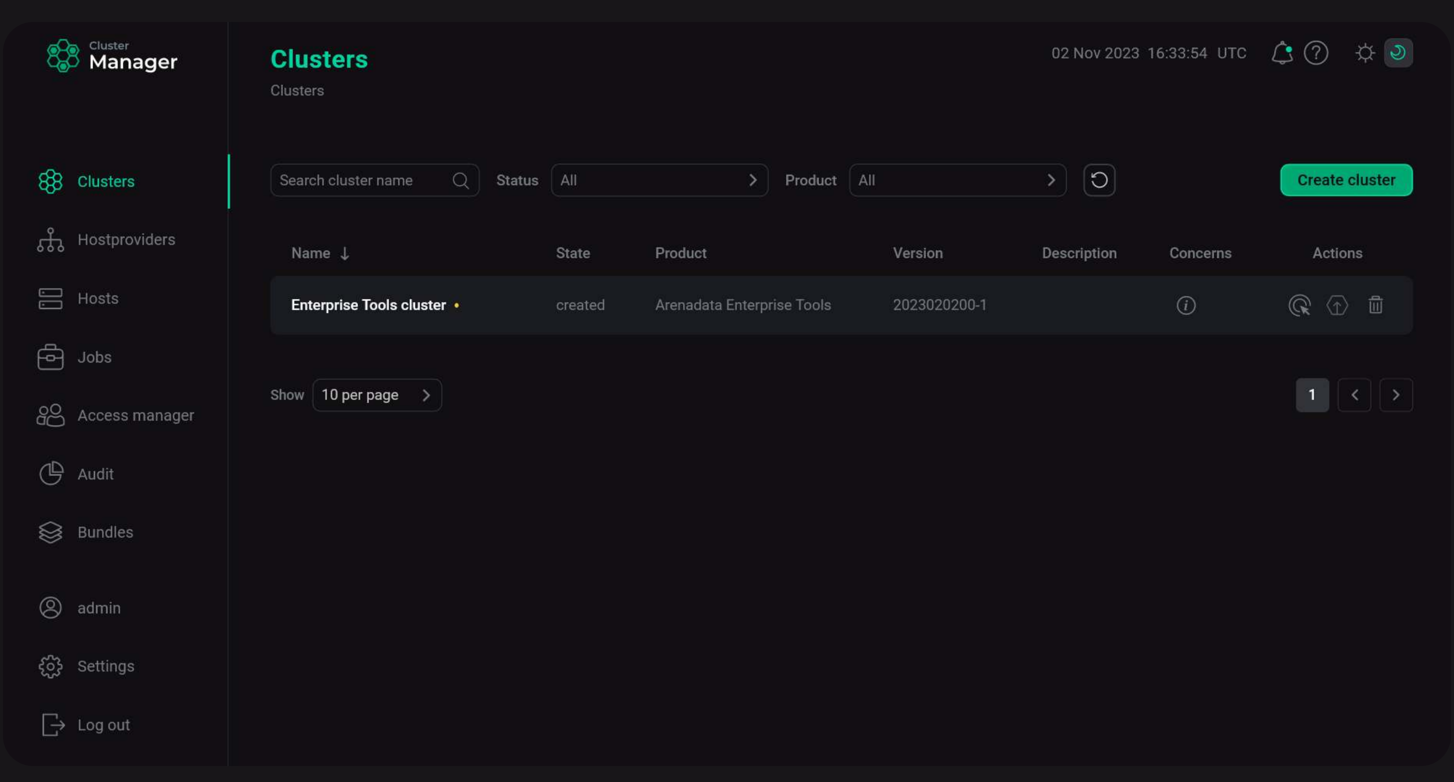
Product version: 2023020200-1

Cluster name: Enterprise Tools cluster

Description: Description

Create a cluster

3. As a result, the created cluster instance is displayed on the ***Clusters*** page.



Manager
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Clusters
02 Nov 2023 16:33:54 UTC
Search cluster name
Status
All
Product
All
Create cluster
Show
10 per page

Name	State	Product	Version	Description	Concerns	Actions
Enterprise Tools cluster •	created	Arenadata Enterprise Tools	2023020200-1			

The result of successful cluster creation

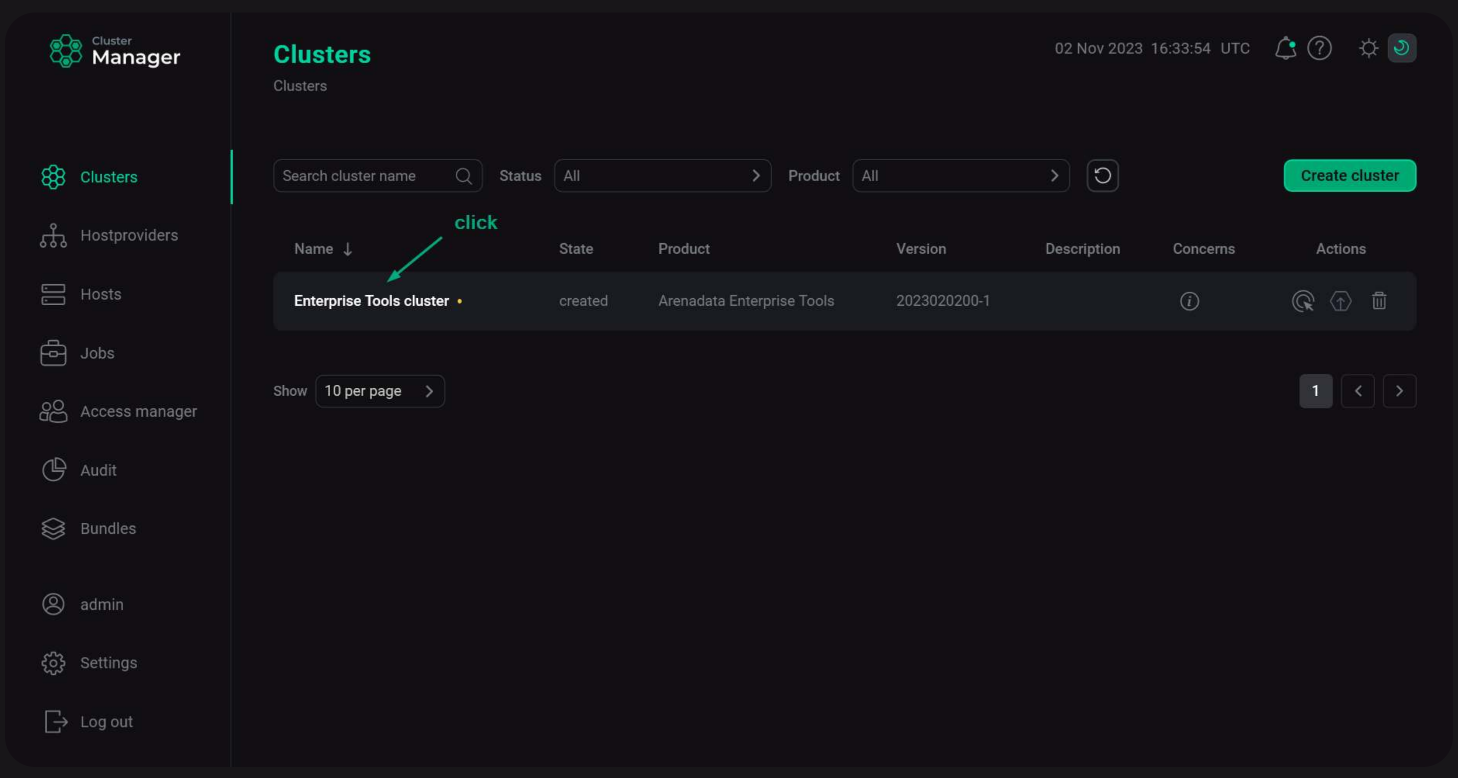
Add services

Daria Barysheva

To Table of Contents

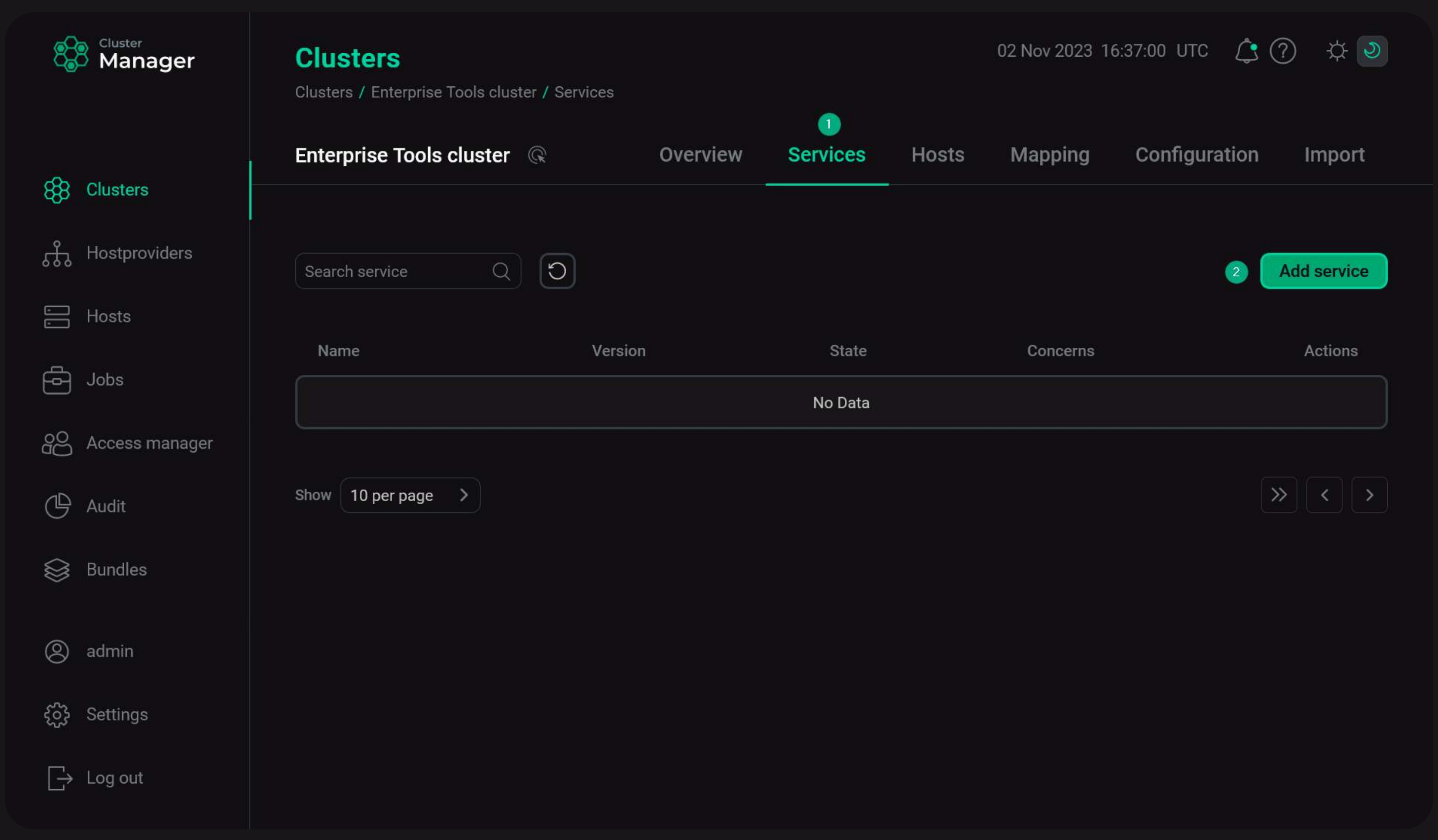
In ADCM a **service** means a software that performs some function. The steps for adding services to a cluster are listed below:

1. Select a cluster on the ***Clusters*** page. To do this, click a cluster name in the ***Name*** column.


Cluster Manager
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Clusters
02 Nov 2023 16:33:54 UTC
Search cluster name
Status All
Product All
Create cluster
click
Name
State
Product
Version
Description
Concerns
Actions
Enterprise Tools cluster
created
Arenadata Enterprise Tools
2023020200-1
Show 10 per page
1

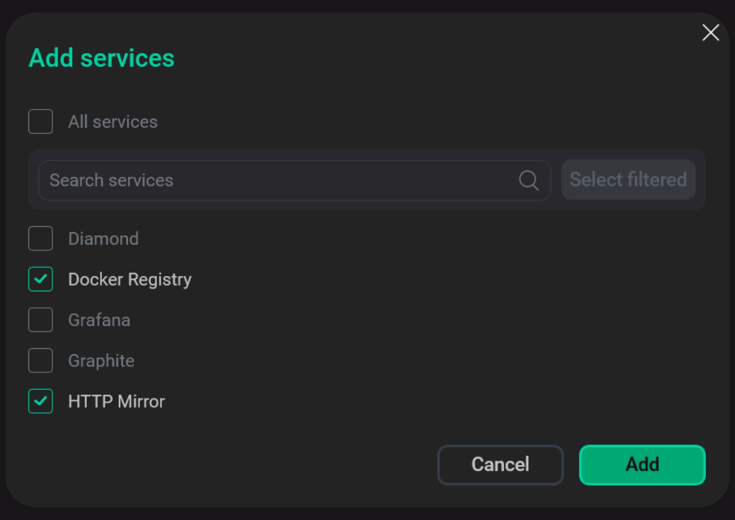
Select a cluster

2. Open the ***Services*** tab on the cluster page and click ***Add service***.


Cluster Manager
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Clusters
Clusters / Enterprise Tools cluster / Services
02 Nov 2023 16:37:00 UTC
Enterprise Tools cluster
Overview
1
Services
Hosts
Mapping
Configuration
Import
Search service
2
Add service
Name
Version
State
Concerns
Actions
No Data
Show 10 per page

Switch to adding services

3. In the opened dialog, select services that should be added to the cluster and click ***Add***.


Add services
☐ All services
Search services
Select filtered
☐ Diamond
☑ Docker Registry
☐ Grafana
☐ Graphite
☑ HTTP Mirror
Cancel
Add

Select services

The brief description of available services is listed below.

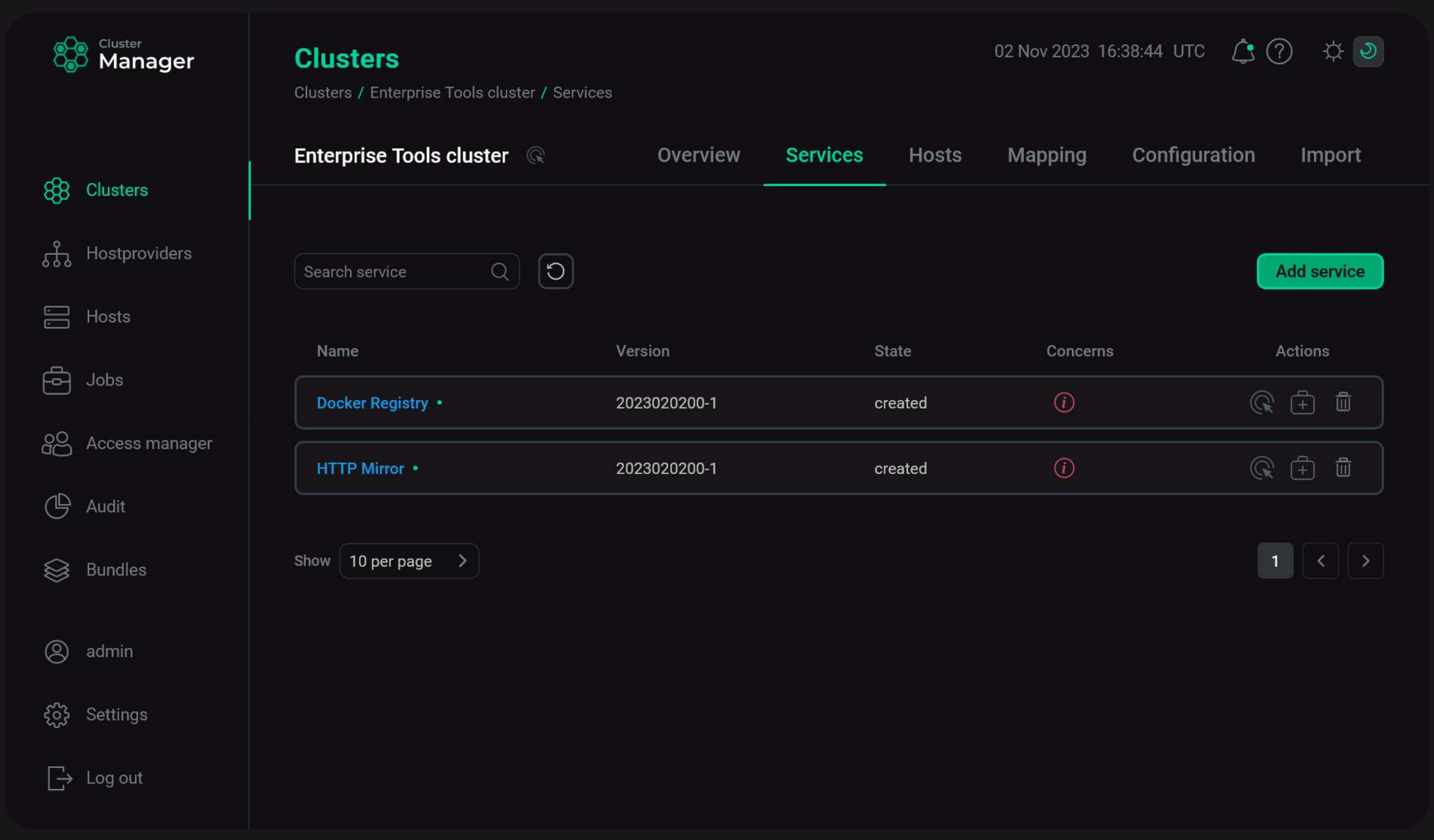
Services that should be added to the Enterprise Tools cluster

Service	Purpose
HTTP Mirror	A service used for hosting packages of the repositories that are needed for offline installation of products
Docker Registry	A service used to store Docker Images that are needed for offline installation of products

NOTE

The CoreDNS service is deprecated — do not select it. The Diamond, Graphite, and Grafana services are used for building monitoring — they will be described separately.

4. As a result, the added services are displayed on the ***Services*** tab.


Cluster Manager
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Clusters
Clusters / Enterprise Tools cluster / Services
02 Nov 2023 16:38:44 UTC
Enterprise Tools cluster
Overview
Services
Hosts
Mapping
Configuration
Import
Search service
Add service
Name
Version
State
Concerns
Actions
Docker Registry
2023020200-1
created
HTTP Mirror
2023020200-1
created
Show 10 per page
1

The result of successful adding services to a cluster

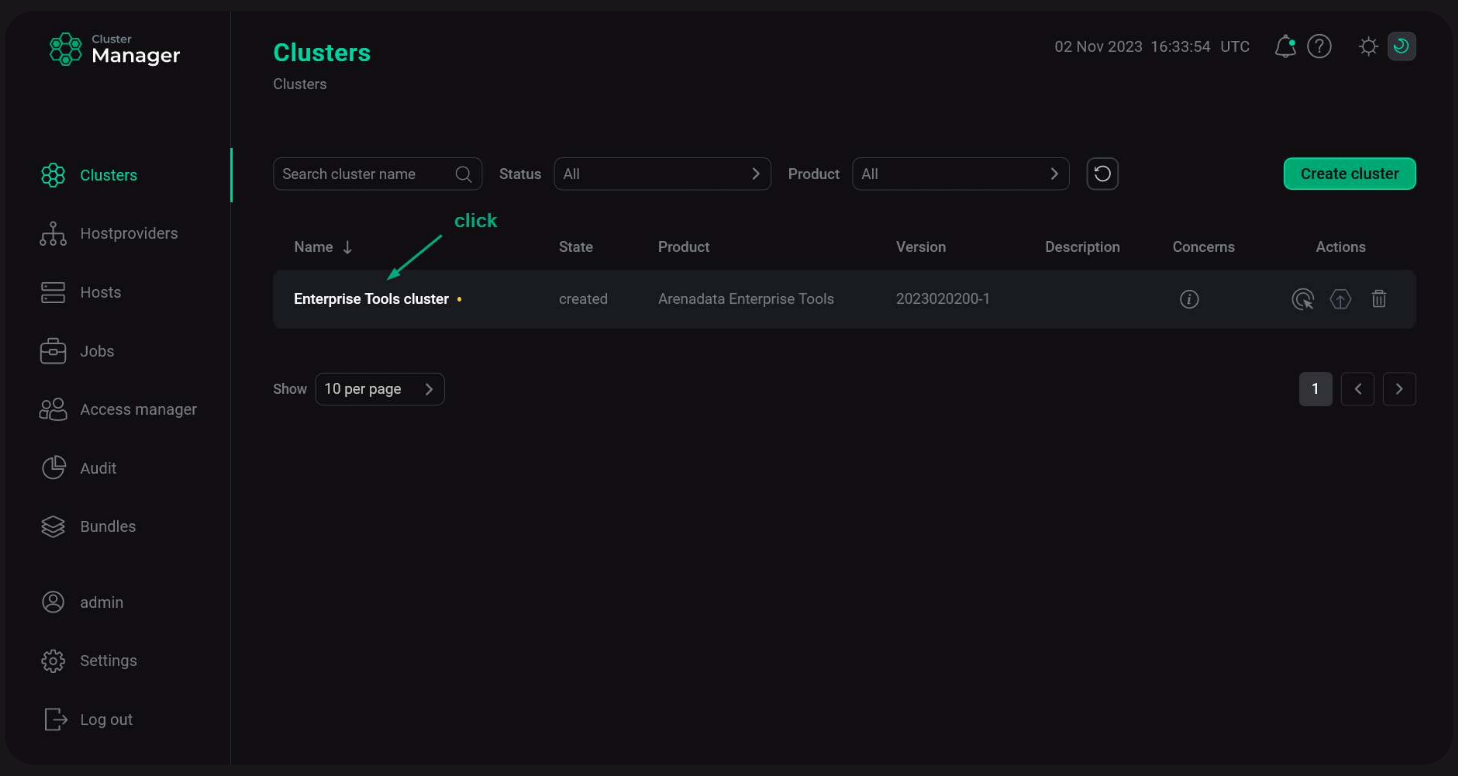
Add hosts to a cluster

Daria Barysheva

To Table of Contents

The steps for adding hosts to a cluster are listed below:

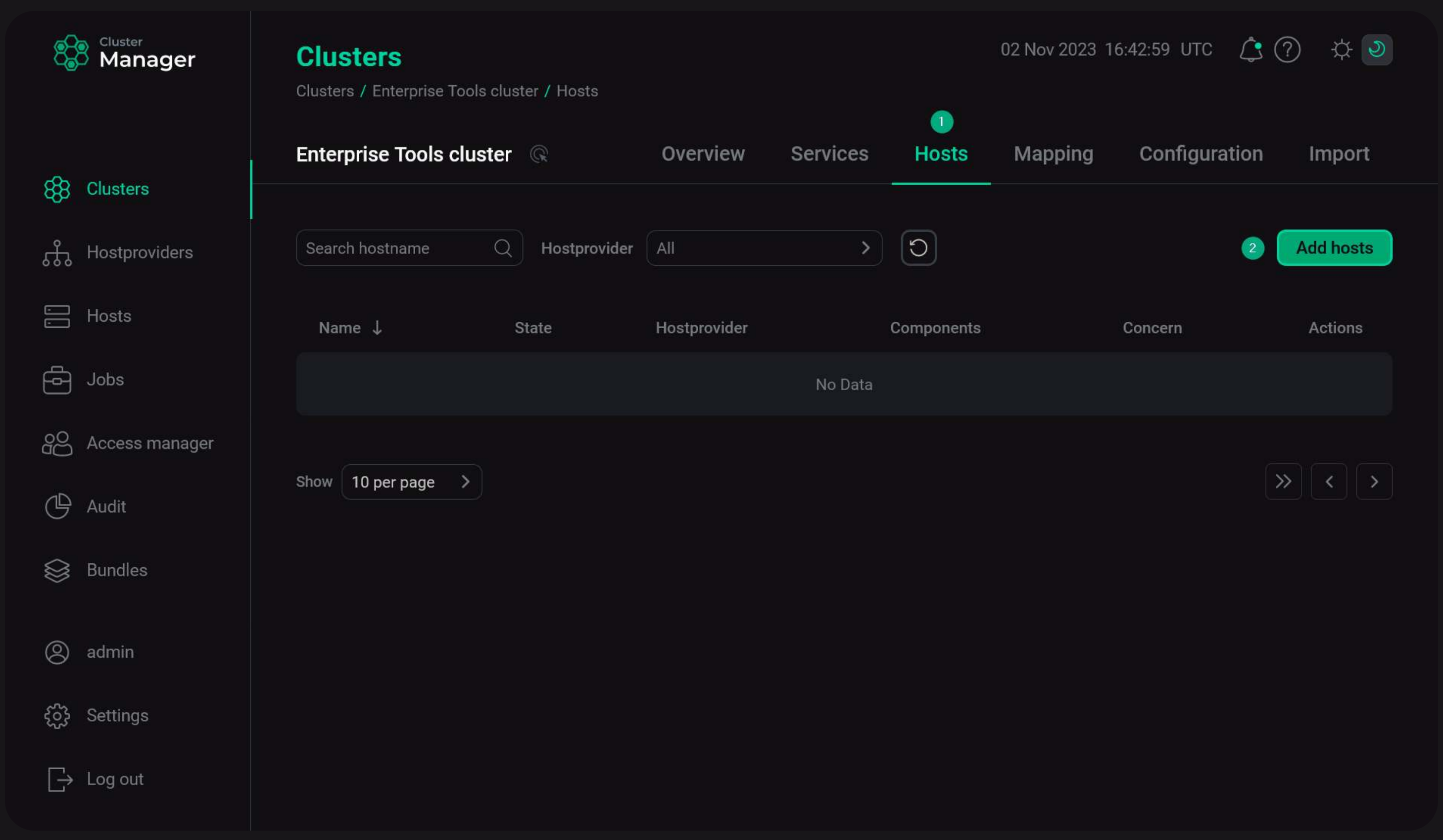
1. Select a cluster on the ***Clusters*** page. To do this, click a cluster name in the ***Name*** column.


Cluster Manager
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Clusters
Clusters
02 Nov 2023 16:33:54 UTC
Search cluster name
Status
All
Product
All
Create cluster
click
Show
10 per page
1

Name	State	Product	Version	Description	Concerns	Actions
Enterprise Tools cluster	created	Arenadata Enterprise Tools	2023020200-1			

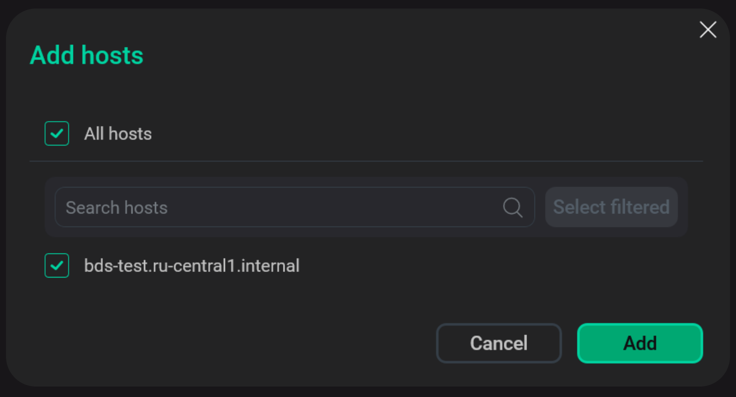
Select a cluster

2. Open the ***Hosts*** tab on the cluster page and click ***Add hosts***.


Cluster Manager
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Clusters
Clusters / Enterprise Tools cluster / Hosts
02 Nov 2023 16:42:59 UTC
Enterprise Tools cluster
Overview
Services
1
Hosts
Mapping
Configuration
Import
Search hostname
Hostprovider
All
2
Add hosts
Name
State
Hostprovider
Components
Concern
Actions
No Data
Show
10 per page

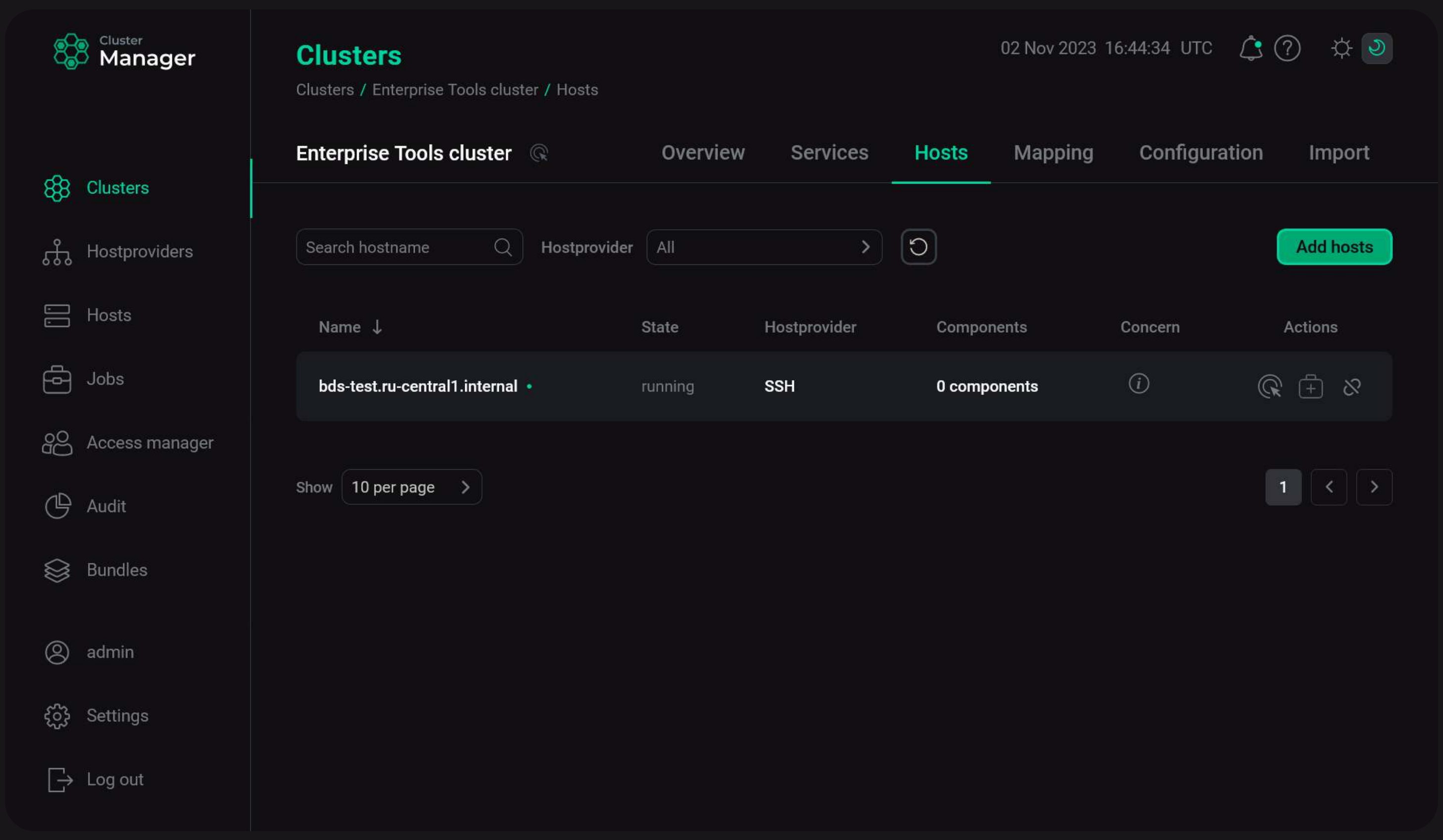
Switch to adding hosts

3. In the opened dialog, select hosts that should be added to the cluster and click ***Add***.


Add hosts
☑ All hosts
Search hosts
Select filtered
☑ bds-test.ru-central1.internal
Cancel
Add

Select hosts

4. As a result, the added hosts are displayed on the ***Hosts*** tab.


Cluster Manager
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Clusters
Clusters / Enterprise Tools cluster / Hosts
02 Nov 2023 16:44:34 UTC
Enterprise Tools cluster
Overview
Services
Hosts
Mapping
Configuration
Import
Search hostname
Hostprovider
All
Add hosts
Show
10 per page
1

Name	State	Hostprovider	Components	Concern	Actions
bds-test.ru-central1.internal	running	SSH	0 components		

The result of successful adding hosts to a cluster

Add components

Daria Barysheva

To Table of Contents

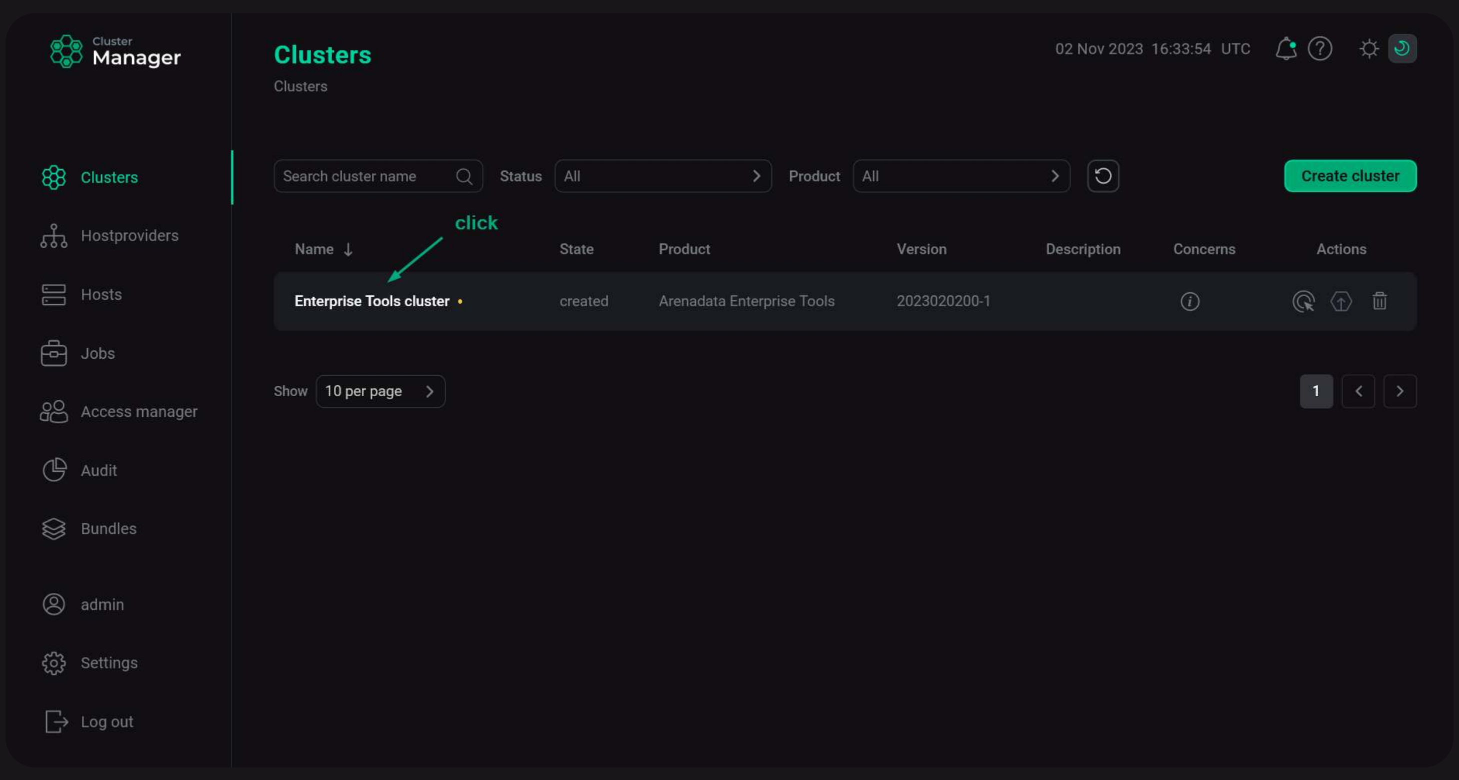
In ADCM a **component** means a part of a service that should be deployed on one or several cluster hosts.

IMPORTANT

Initially, there are no components on any of cluster hosts. You should map components to hosts manually.

The steps for adding components to hosts are listed below:

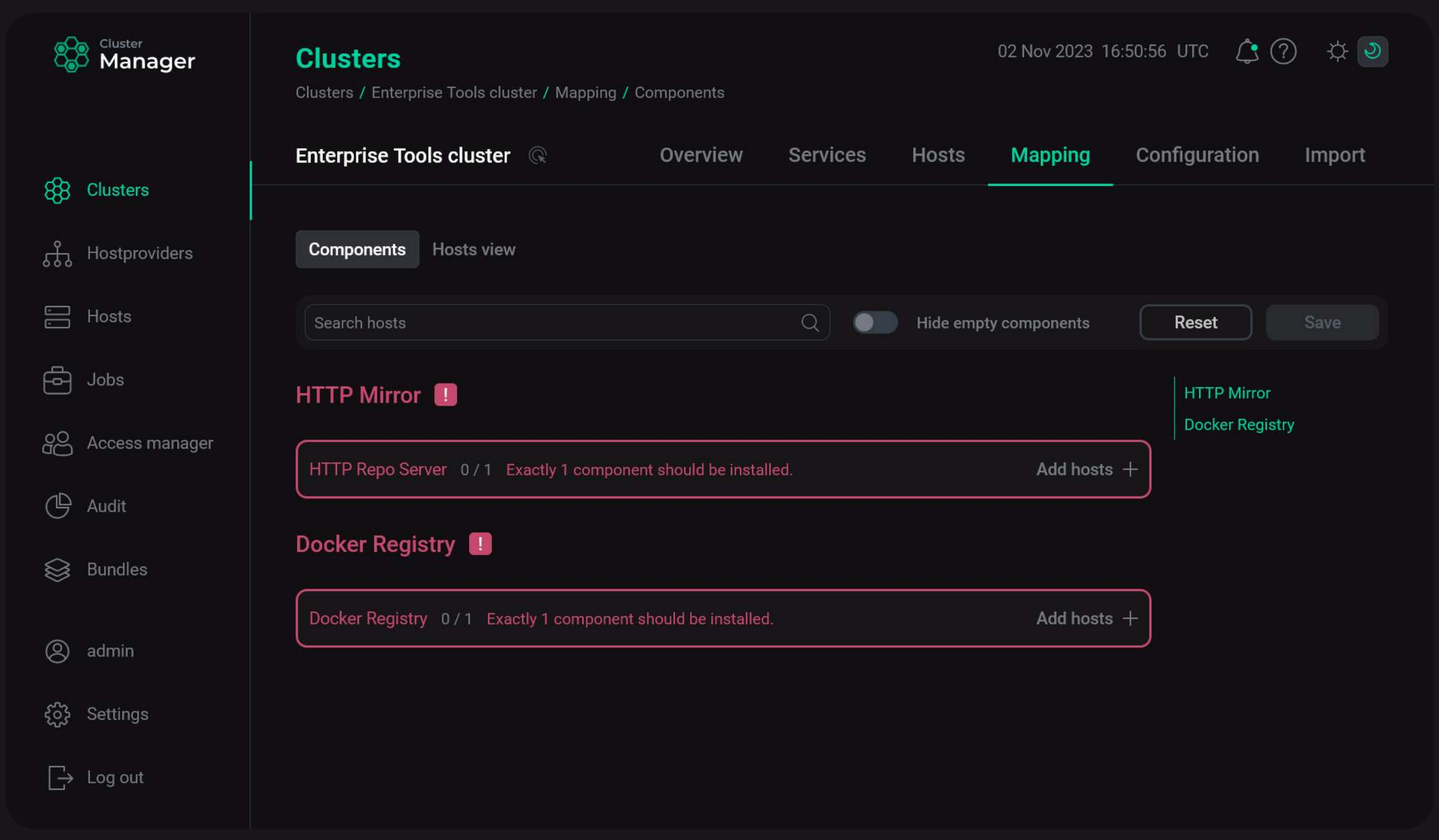
1. Select a cluster on the ***Clusters*** page. To do this, click a cluster name in the ***Name*** column.


Cluster Manager
Clusters
Clusters
02 Nov 2023 16:33:54 UTC
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Search cluster name
Status
All
Product
All
Create cluster
click
Show
10 per page
1

Name	State	Product	Version	Description	Concerns	Actions
Enterprise Tools cluster	created	Arenadata Enterprise Tools	2023020200-1			

Select a cluster

2. Open the ***Mapping*** tab on the cluster page.

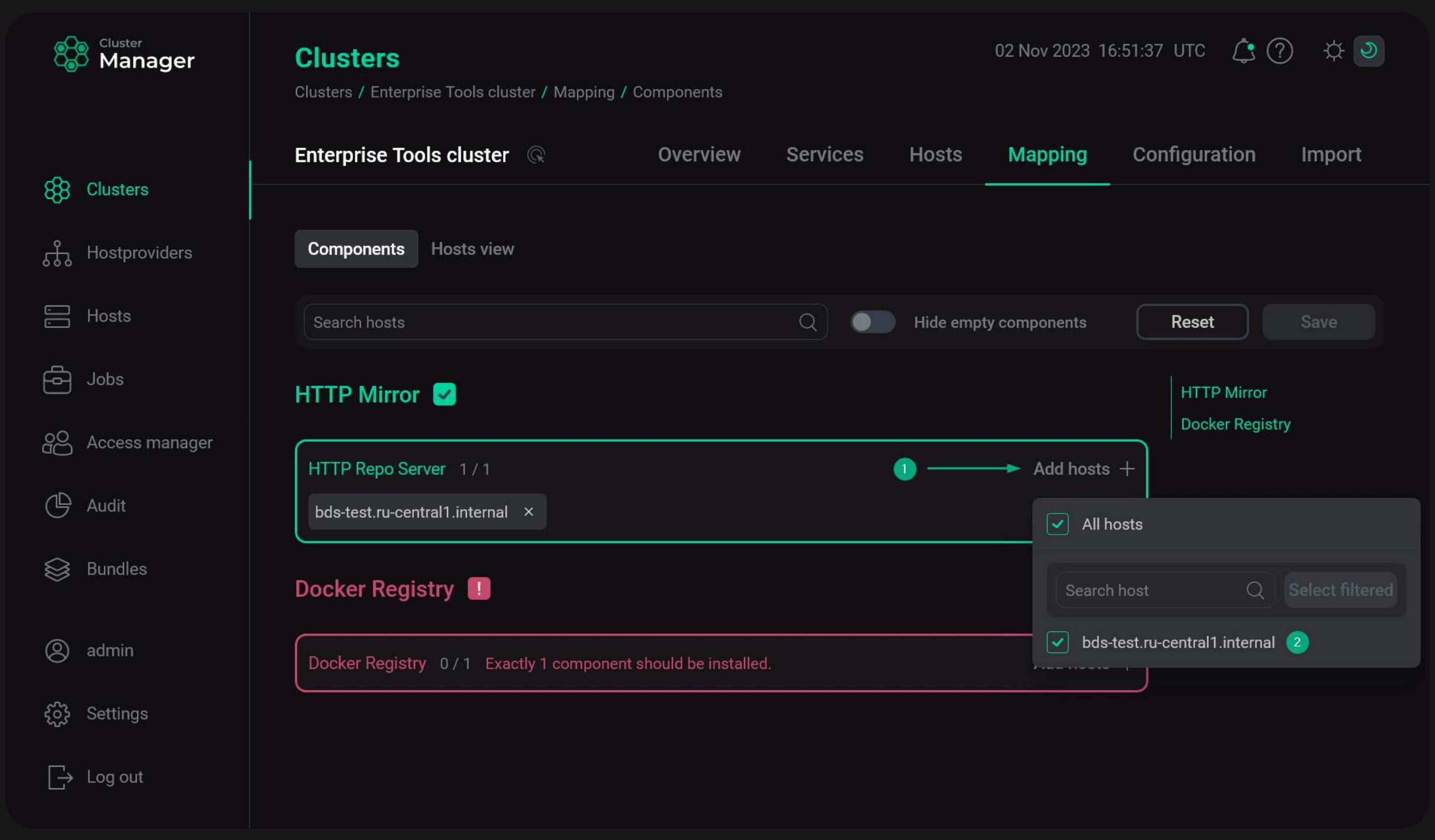

Cluster Manager
Clusters
Clusters / Enterprise Tools cluster / Mapping / Components
02 Nov 2023 16:50:56 UTC
Enterprise Tools cluster
Overview
Services
Hosts
Mapping
Configuration
Import
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Components
Hosts view
Search hosts
Hide empty components
Reset
Save
HTTP Mirror
Docker Registry
HTTP Mirror
Docker Registry
Add hosts +
Add hosts +

HTTP Repo Server 0 / 1 Exactly 1 component should be installed.

Docker Registry 0 / 1 Exactly 1 component should be installed.

Switch to mapping service components

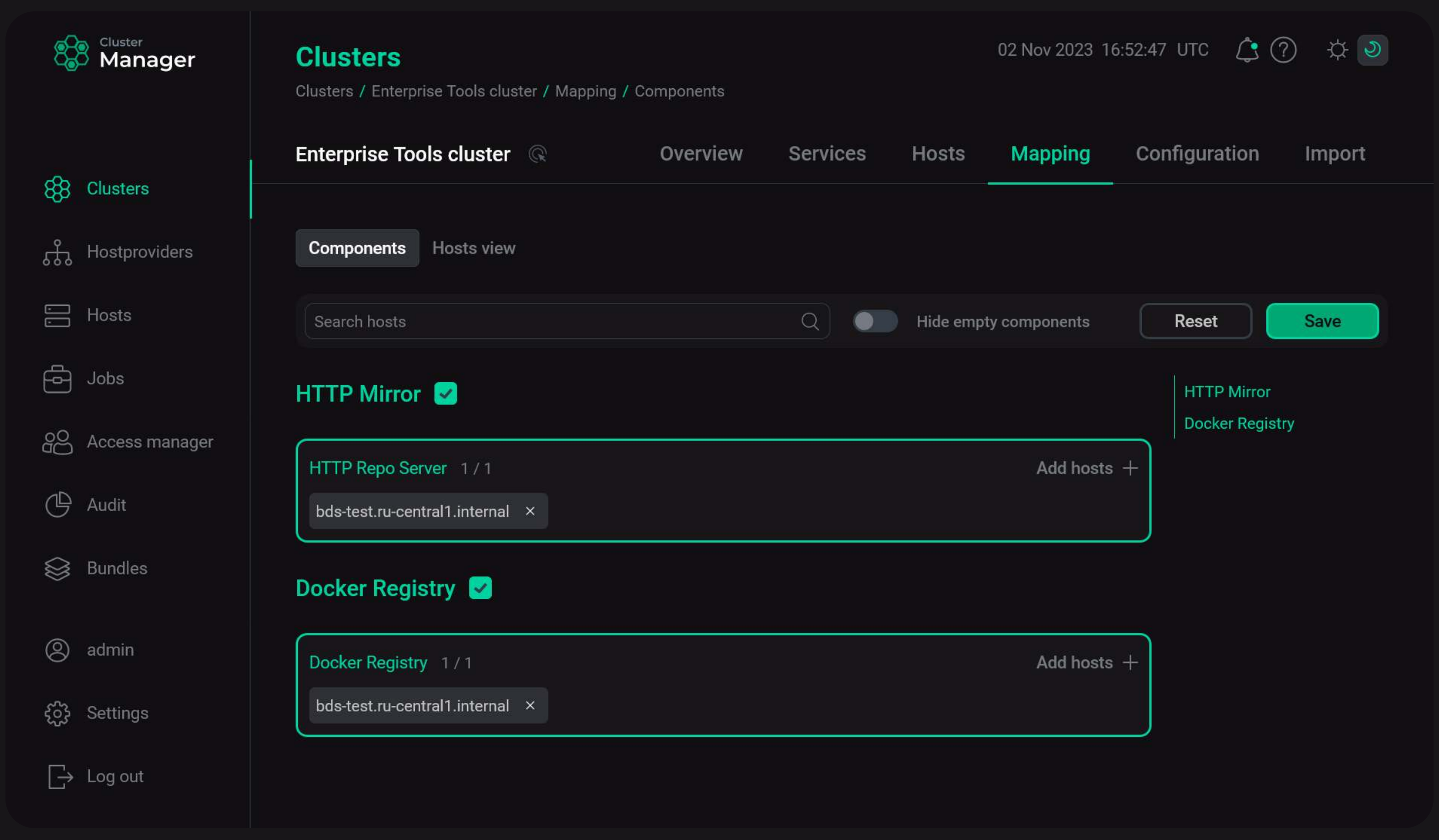
3. In the window that opens, map components to hosts. To assign hosts for each component, click ***Add hosts*** and select hosts in the pop-up window.


Cluster Manager
Clusters
Clusters / Enterprise Tools cluster / Mapping / Components
02 Nov 2023 16:51:37 UTC
Enterprise Tools cluster
Overview
Services
Hosts
Mapping
Configuration
Import
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Components
Hosts view
Search hosts
Hide empty components
Reset
Save
HTTP Mirror
Docker Registry
HTTP Mirror
Docker Registry
HTTP Repo Server 1 / 1
bds-test.ru-central1.internal ×
1
Add hosts +
☑ All hosts
Search host
Select filtered
☑ bds-test.ru-central1.internal
2

Docker Registry 0 / 1 Exactly 1 component should be installed.

Select hosts for a component

4. After component distribution is completed, click ***Save***.


Cluster Manager
Clusters
Clusters / Enterprise Tools cluster / Mapping / Components
02 Nov 2023 16:52:47 UTC
Enterprise Tools cluster
Overview
Services
Hosts
Mapping
Configuration
Import
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Components
Hosts view
Search hosts
Hide empty components
Reset
Save
HTTP Mirror ☑
Docker Registry ☑
HTTP Mirror
Docker Registry
HTTP Repo Server 1 / 1
bds-test.ru-central1.internal ×
Add hosts +
Docker Registry 1 / 1
bds-test.ru-central1.internal ×
Add hosts +

Save mapping of components

Install a cluster

Daria Barysheva

Contents

To Table of Contents

Step 1. Run installation

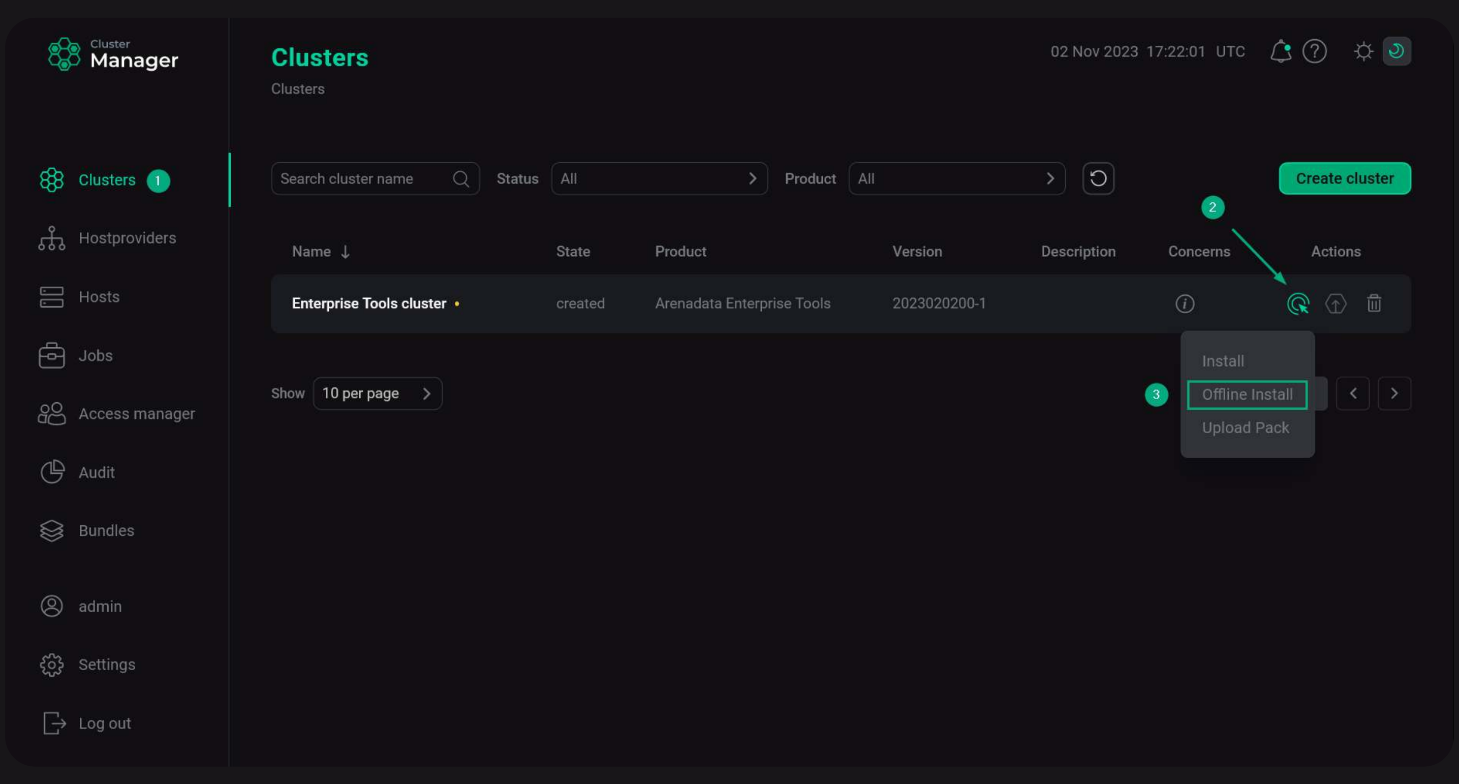
Step 2. Upload packages for offline installation

Step 3. Verify results

Step 1. Run installation

The last step of adding a new Enterprise Tools cluster is to install its services. To do this:

1. Apply the ***Offline Install*** action to the cluster selected on the ***Clusters*** page by clicking the icon in the ***Actions*** column.

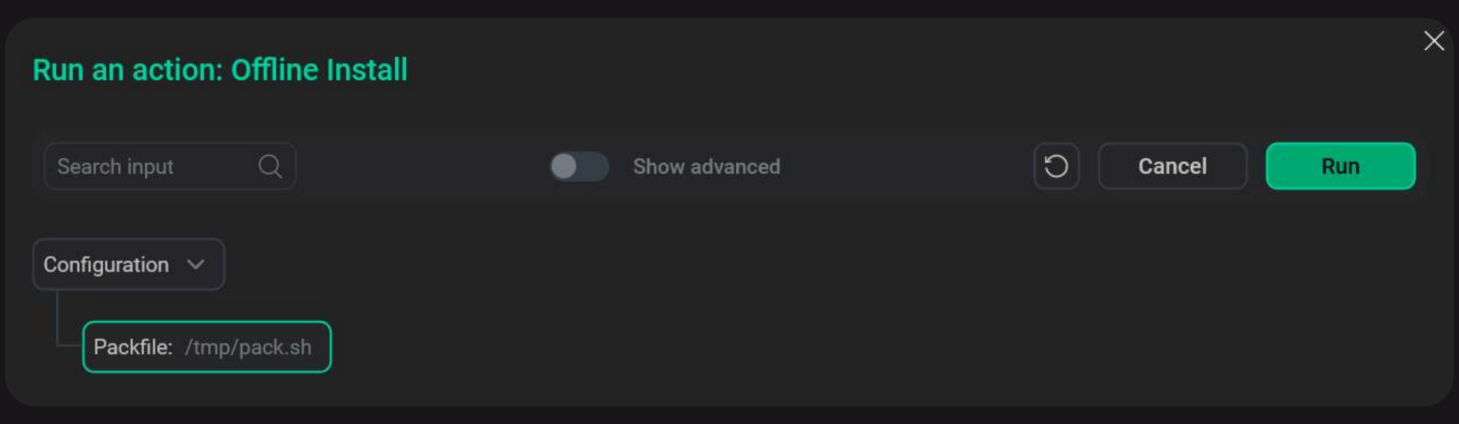


Cluster Manager
Clusters
Clusters
02 Nov 2023 17:22:01 UTC
Clusters 1
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Search cluster name
Status All
Product All
Create cluster
2
3
Install
Offline Install
Upload Pack
Show 10 per page

Name	State	Product	Version	Description	Concerns	Actions
Enterprise Tools cluster	created	Arenadata Enterprise Tools	2023020200-1			

Switch to cluster installation

2. In the opened window, enter a full path to the shell script that is received from the Arenadata support team and click ***Run***.



Run an action: Offline Install
Search input
Show advanced
Cancel
Run
Configuration

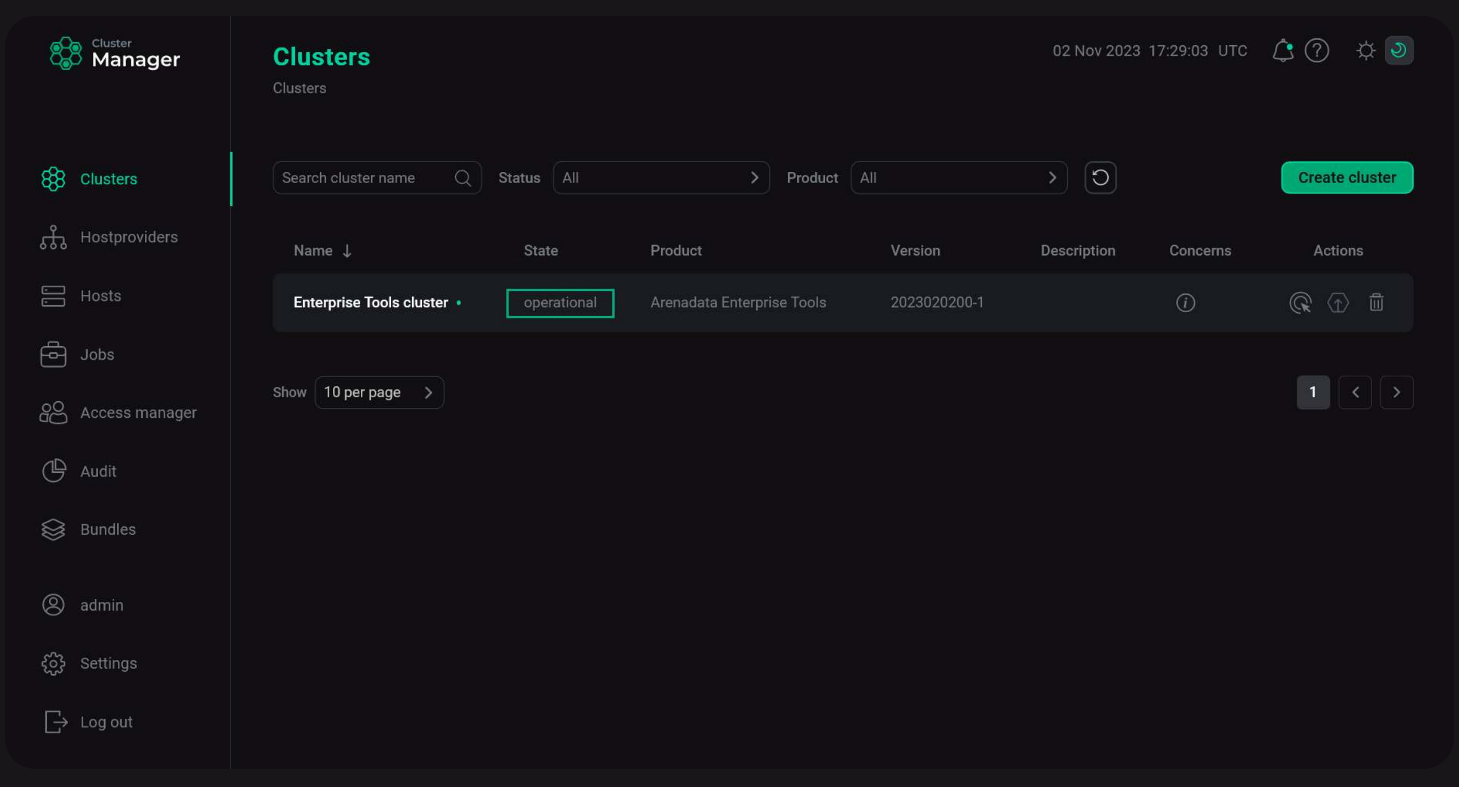
Packfile: /tmp/pack.sh

Verify installation

3. Wait until the installation process is completed.

NOTE

In case of errors, you can find logs on the ***Jobs*** page.



Cluster Manager
Clusters
Clusters
02 Nov 2023 17:29:03 UTC
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Search cluster name
Status All
Product All
Create cluster
Show 10 per page
1

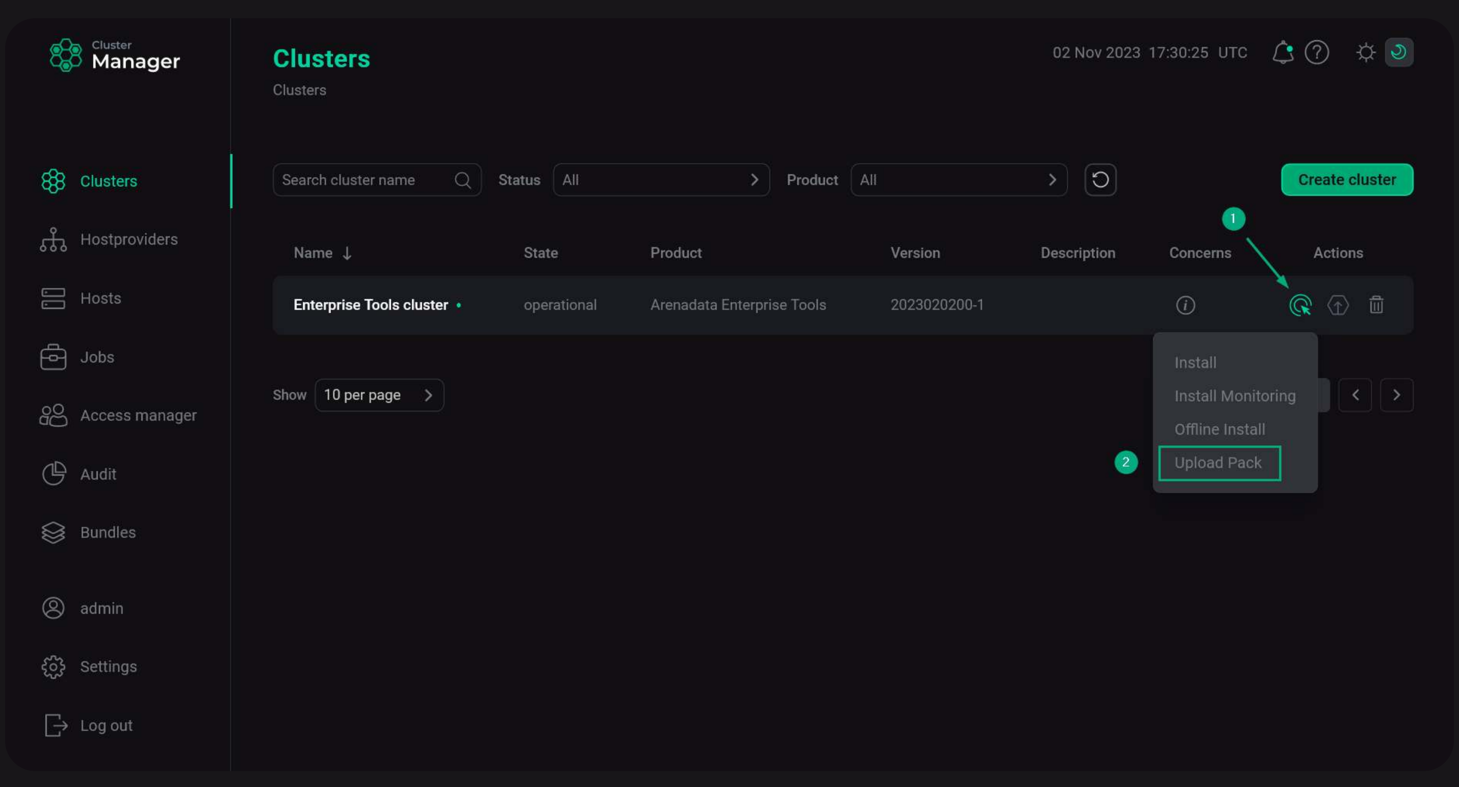
Name	State	Product	Version	Description	Concerns	Actions
Enterprise Tools cluster	operational	Arenadata Enterprise Tools	2023020200-1			

Cluster installation is completed

Step 2. Upload packages for offline installation

In addition to installing the cluster, you should process the shell script that is received from the Arenadata support team. It is necessary to enable offline installation of products in the future. To do this:

1. Apply the ***Upload Pack*** action to the cluster selected on the ***Clusters*** page by clicking the icon in the ***Actions*** column.

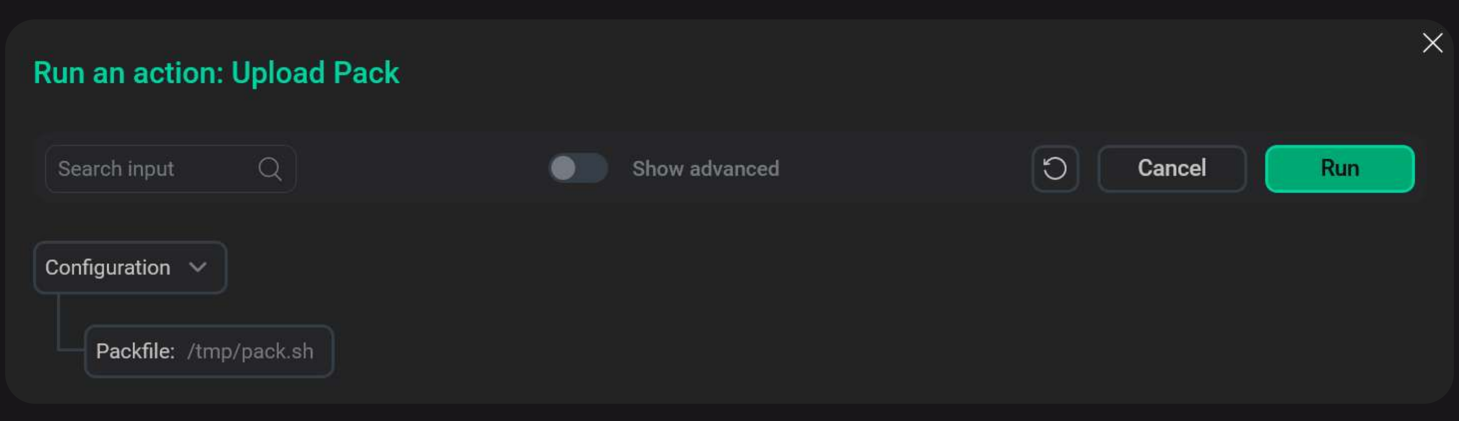


Cluster Manager
Clusters
Clusters
02 Nov 2023 17:30:25 UTC
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Search cluster name
Status All
Product All
Create cluster
1
2
Install
Install Monitoring
Offline Install
Upload Pack
Show 10 per page

Name	State	Product	Version	Description	Concerns	Actions
Enterprise Tools cluster	operational	Arenadata Enterprise Tools	2023020200-1			

Switch to uploading packages

2. In the opened window, enter a full path to the shell script received from the Arenadata support team and click ***Run***.



Run an action: Upload Pack
Search input
Show advanced
Cancel
Run
Configuration

Packfile: /tmp/pack.sh

Verify upload

3. Wait until the upload process is completed.

NOTE

In case of errors, you can find logs on the ***Jobs*** page.

Step 3. Verify results

The steps for checking the installation results are listed below:

1. Check that ports `443` and `81` are listened at the host where cluster is installed. The first port is used by the Docker Registry service, the second one — by the HTTP Mirror service.

```
$ sudo netstat -ntlp|grep L|grep -E "81|443"
```

   The result should look like this:

```
tcp6       0      0 :::443                  :::*                    LISTEN      1694/docker-proxy-c
tcp6       0      0 :::81                   :::*                    LISTEN      1708/docker-proxy-c
```

2. Check that the */opt/arenadata/etools/httprepo/arenadata-repo* folder exists at the host where the cluster is installed. This folder should contain the data necessary for offline installation of the products that you have chosen before getting a shell script from the Arenadata support team.

```
$ ls -la /opt/arenadata/etools/httprepo/arenadata-repo
```

   The following example displays the data that can be placed in this folder for ADQM installation.

```
total 0
drwxr-xr-x. 5 root root 46 june 15 09:00 .
drwxr-xr-x. 3 root root 28 june 15 08:59 ..
drwxr-xr-x. 3 root root 17 june 15 09:00 ADM
drwxr-xr-x. 3 root root 33 june 15 08:59 ADQM
drwxr-xr-x. 3 root root 19 june 15 09:00 zookeeper
```

Create a cluster

Daria Barysheva, Elena Dvoryadkina

Contents

To Table of Contents

Step 1. Get the Enterprise version of a cluster bundle
Step 2. Upload a cluster bundle to ADCM
Step 3. Create a cluster based on the uploaded bundle

In ADCM a **cluster** means a set of services running on a set of hosts.

Clusters distributions for ADCM are made in the form of so-called bundles. Generally a bundle is a regular archive that includes a description and software logic of cluster deployment and exploitation.

The steps for adding a cluster to ADCM are listed below.

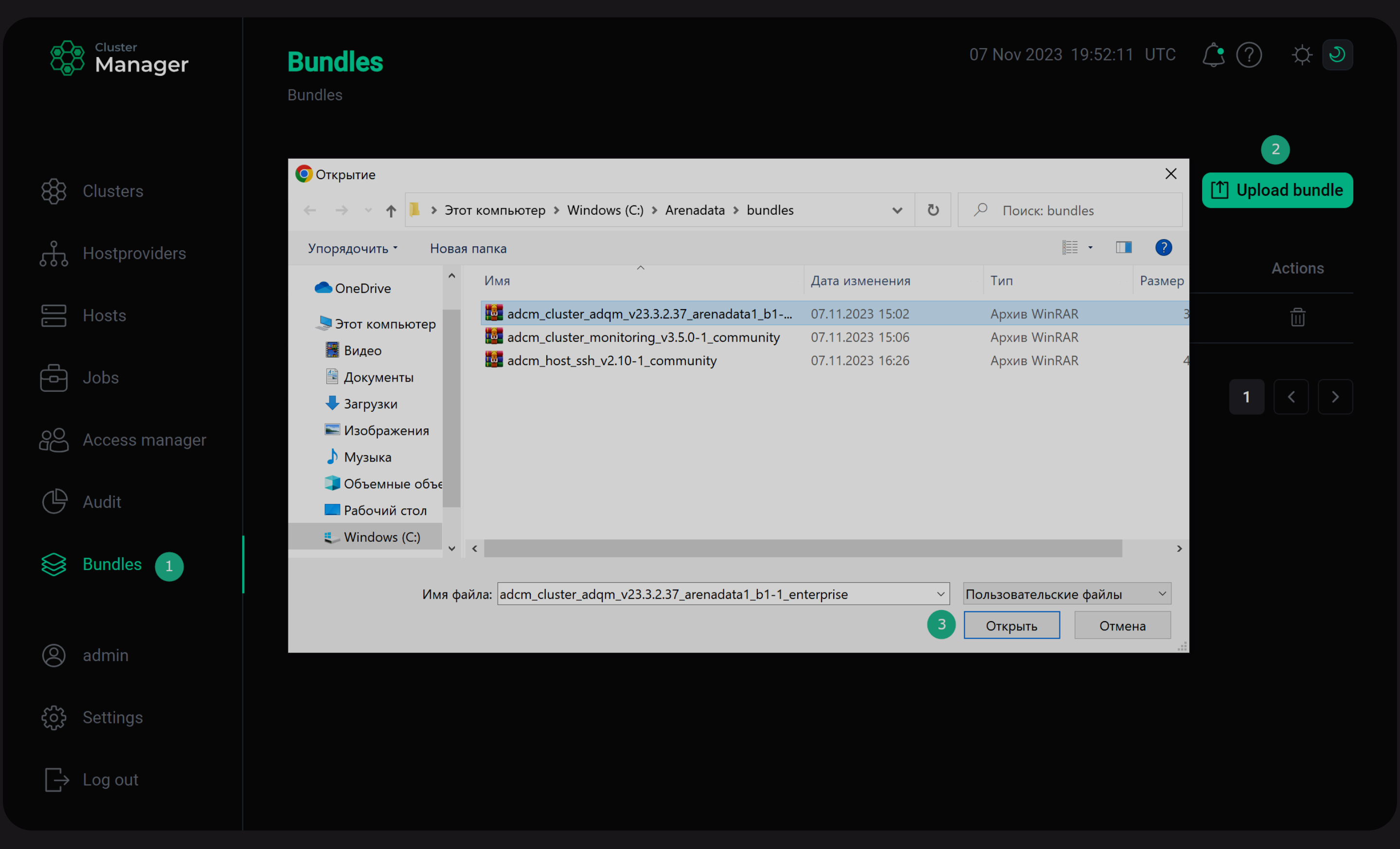
Step 1. Get the Enterprise version of a cluster bundle

Please get a cluster bundle from the Arenadata support team.

Step 2. Upload a cluster bundle to ADCM

To upload a bundle to ADCM, follow the steps:

1. Select the ***Bundles*** item in the left navigation menu and click ***Upload bundle***.
2. Select a bundle in the Open File Dialog.



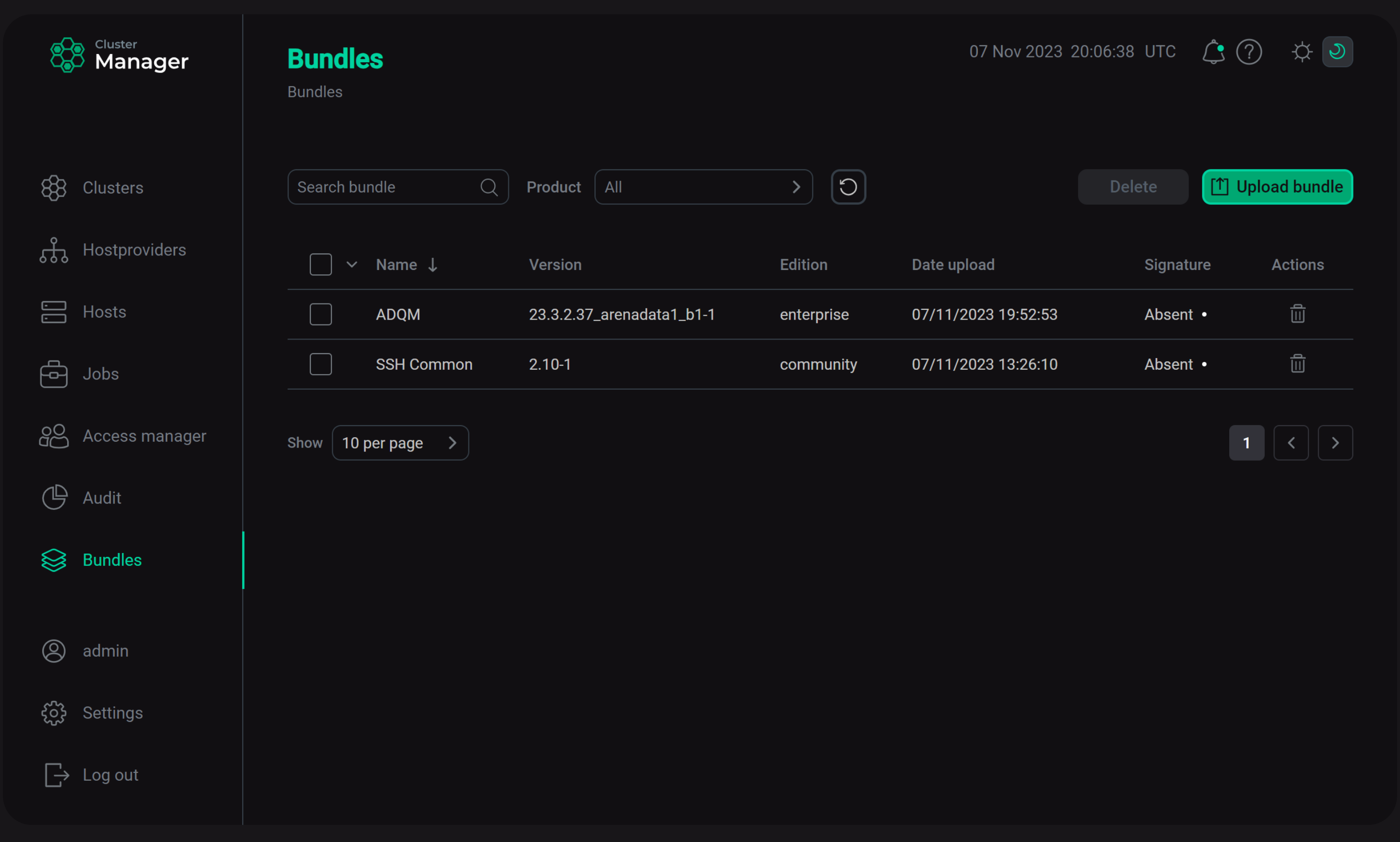
Cluster Manager
Bundles
Bundles
07 Nov 2023 19:52:11 UTC
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
1
admin
Settings
Log out
2
Upload bundle
Actions
Открытие
Этот компьютер > Windows (C:) > Arenadata > bundles
Поиск: bundles
Упорядочить
Новая папка
OneDrive
Этот компьютер
Видео
Документы
Загрузки
Изображения
Музыка
Объемные объе
Рабочий стол
Windows (C:)
Пользовательские файлы
3
Открыть
Отмена

Имя	Дата изменения	Тип
adcm_cluster_adqm_v23.3.2.37_arenadata1_b1-...	07.11.2023 15:02	Архив WinRAR
adcm_cluster_monitoring_v3.5.0-1_community	07.11.2023 15:06	Архив WinRAR
adcm_host_ssh_v2.10-1_community	07.11.2023 16:26	Архив WinRAR

Имя файла: adcm_cluster_adqm_v23.3.2.37_arenadata1_b1-1_enterprise

Upload a bundle

3. As a result of the performed actions, a bundle is displayed on the ***Bundles*** page.



Cluster Manager
Bundles
Bundles
07 Nov 2023 20:06:38 UTC
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Search bundle
Product
All
Delete
Upload bundle
Show
10 per page
1

Name	Version	Edition	Date upload	Signature	Actions
ADQM	23.3.2.37_arenadata1_b1-1	enterprise	07/11/2023 19:52:53	Absent •	
SSH Common	2.10-1	community	07/11/2023 13:26:10	Absent •	

The result of successful uploading a bundle

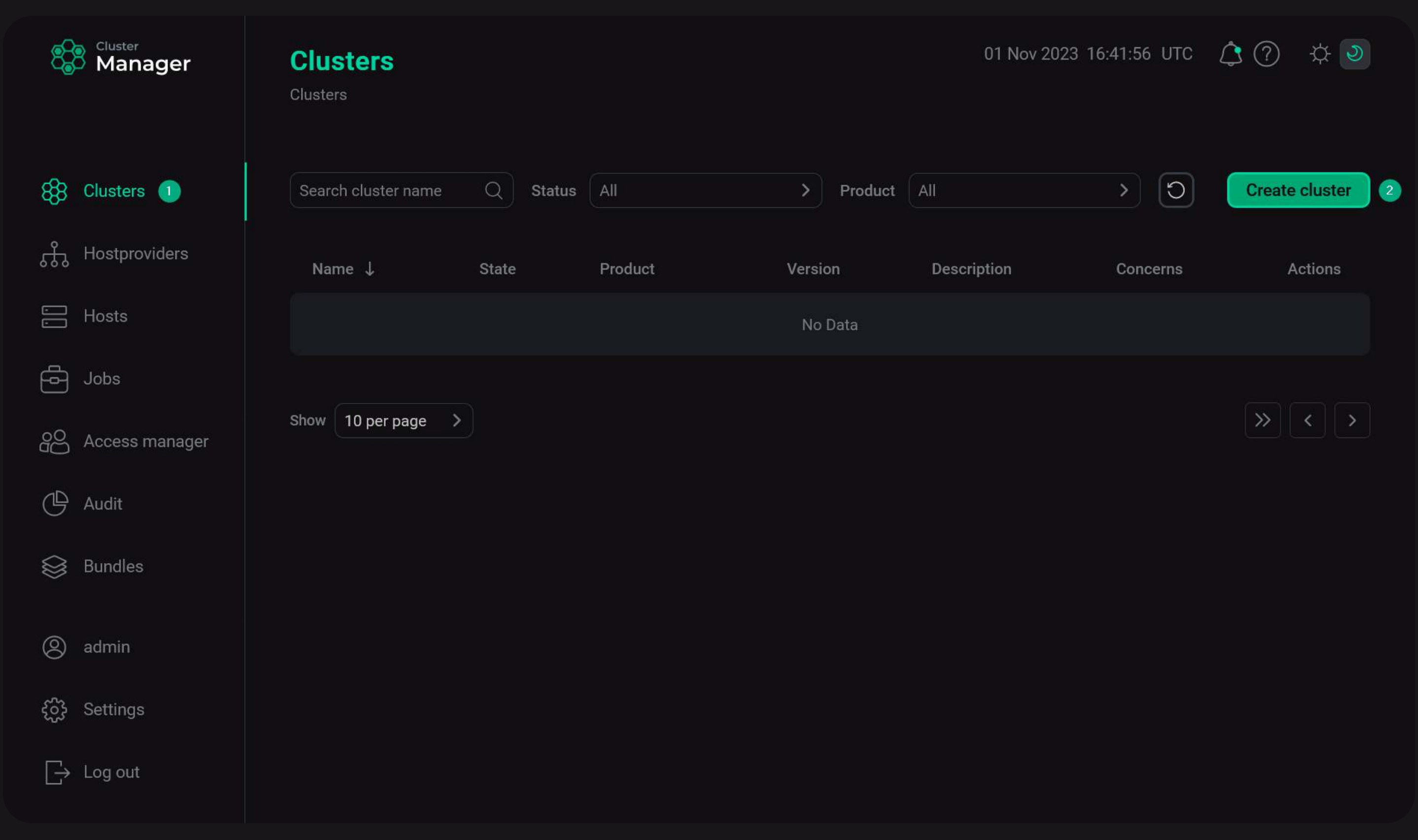
Step 3. Create a cluster based on the uploaded bundle

IMPORTANT

When creating a cluster via the ADCM web interface, you generate a new ADCM cluster instance. It means only adding the cluster to ADCM — at this stage no services are deployed.

The steps for cluster creation are given below:

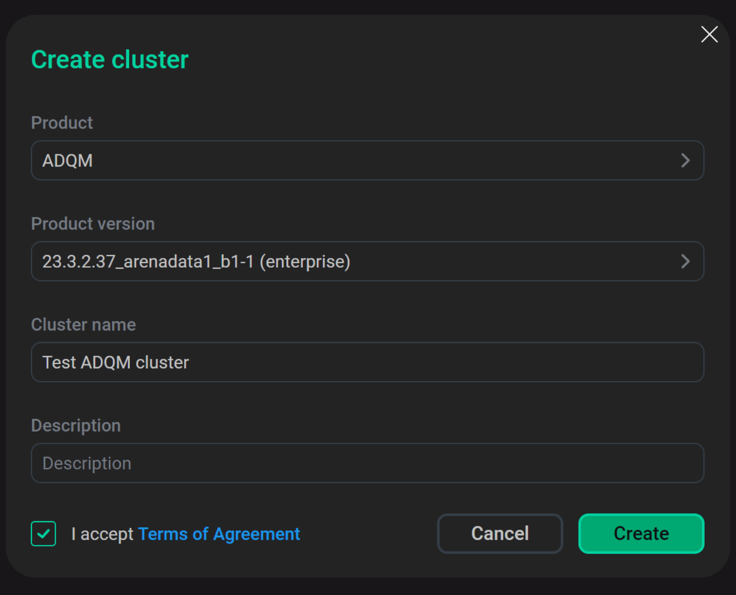
1. Select the ***Clusters*** item in the left navigation menu and click ***Create cluster***.



Cluster Manager
Clusters
Clusters
01 Nov 2023 16:41:56 UTC
Clusters
1
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Search cluster name
Status
All
Product
All
Create cluster
2
Name
State
Product
Version
Description
Concerns
Actions
No Data
Show
10 per page

Switch to cluster creation

2. In the opened dialog, describe a new cluster instance:
   - Select an uploaded cluster bundle in the ***Product*** field.
   - Select a bundle version in the ***Product version*** field. Several versions become available in the case of different versions of the same bundle being uploaded.
   - Enter a cluster name in the ***Cluster name*** field.
   - Enter a cluster description in the ***Description*** field if necessary.
   - Set the ***I accept Terms of Agreement*** flag to sign the license agreement. This flag is displayed only for Enterprise bundles. To read the agreement text, refer to the ***Terms of Agreement*** link. You can also sign the license agreement on the ***Bundles*** page by clicking the desired product bundle.
   - Click ***Create***.



Create cluster
Cancel
Create

Product: ADQM

Product version: 23.3.2.37_arenadata1_b1-1 (enterprise)

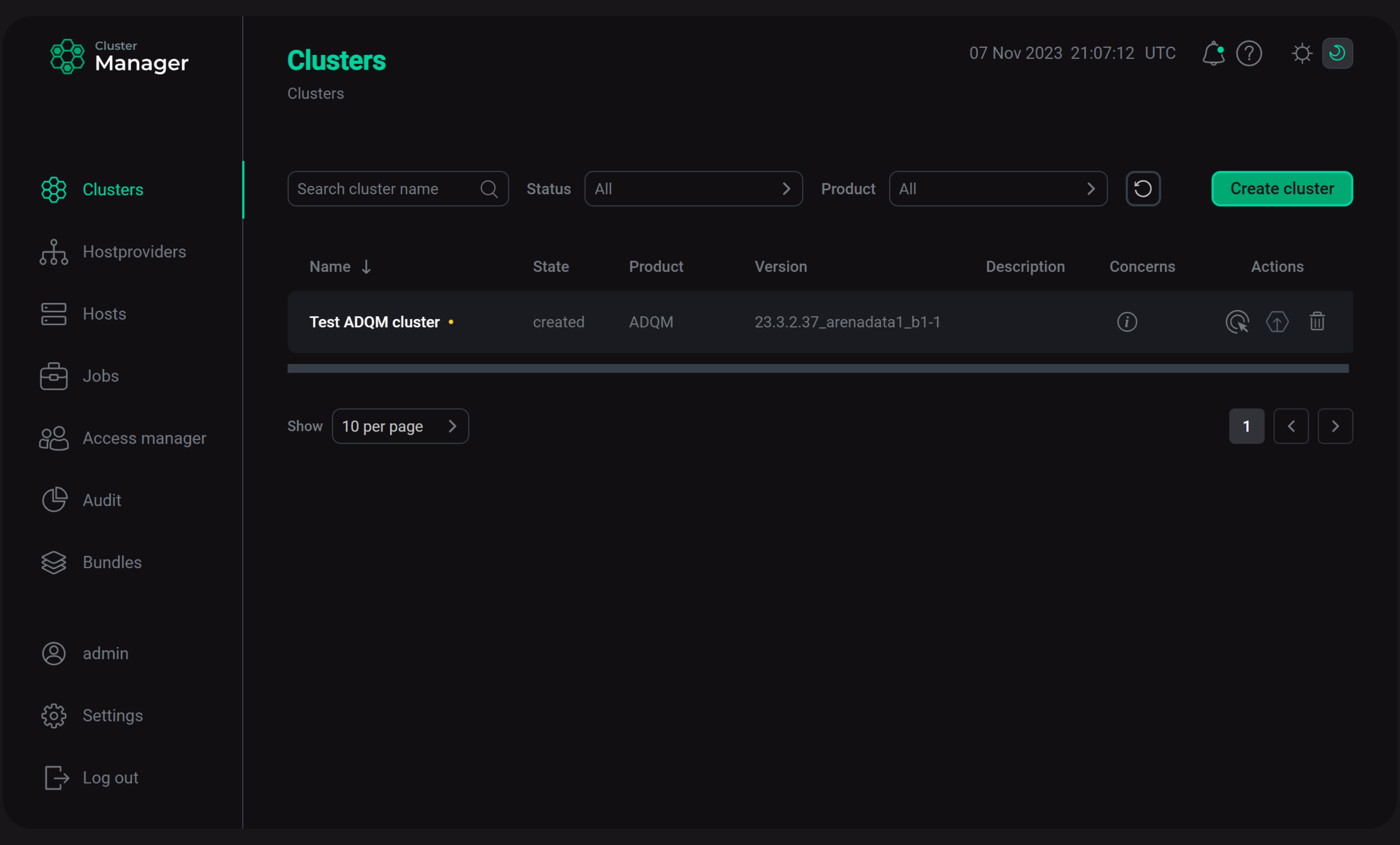
Cluster name: Test ADQM cluster

Description: Description

☑ I accept Terms of Agreement

Create a cluster

3. As a result, the created cluster instance is displayed on the ***Clusters*** page.



Cluster Manager
Clusters
Clusters
07 Nov 2023 21:07:12 UTC
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Search cluster name
Status
All
Product
All
Create cluster
Show
10 per page
1

Name	State	Product	Version	Description	Concerns	Actions
Test ADQM cluster •	created	ADQM	23.3.2.37_arenadata1_b1-1			

The result of successful cluster creation

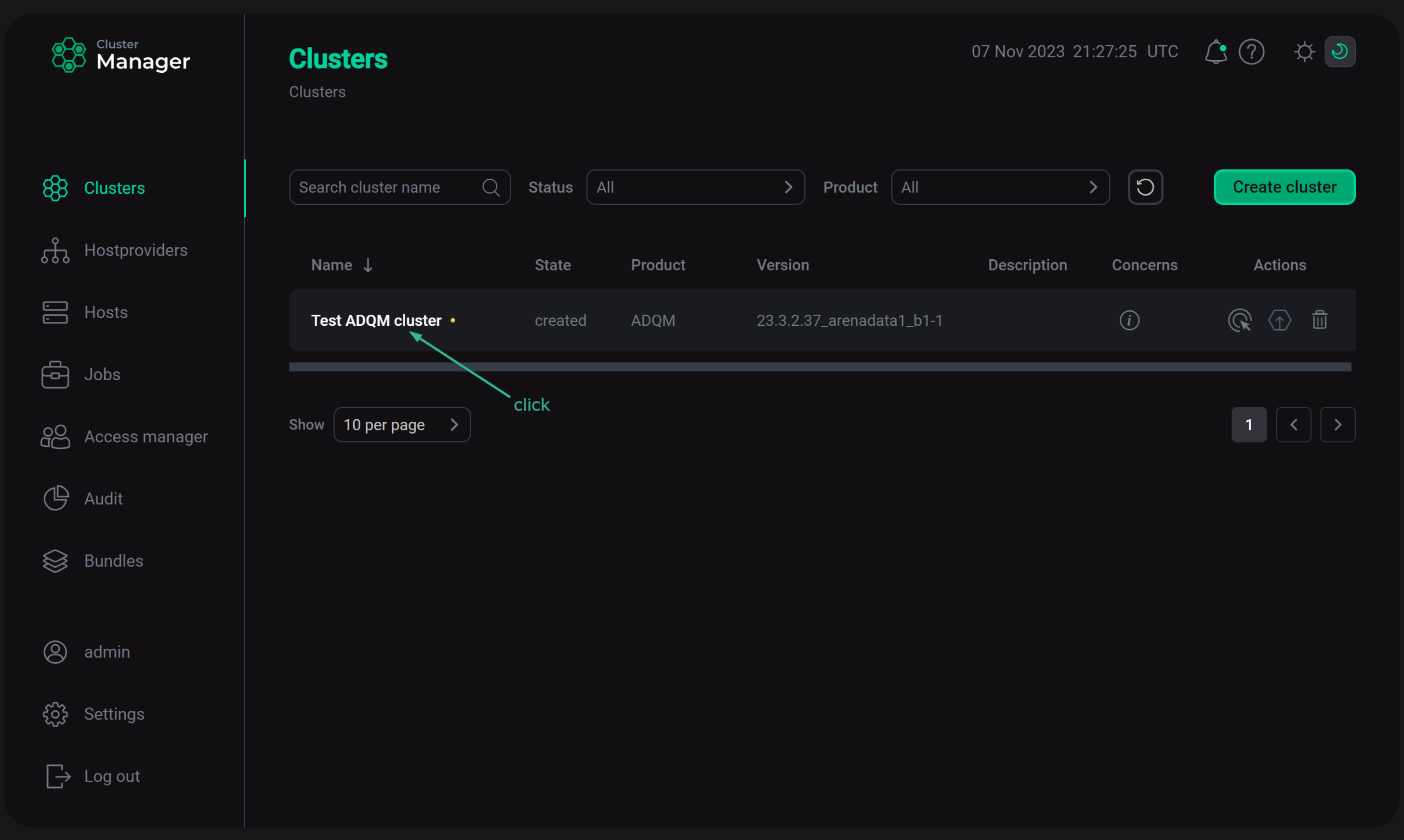
Add services

Daria Barysheva, Elena Dvoryadkina

To Table of Contents

In ADCM a **service** means a software that performs some function. Examples of services for ADQM clusters: ADQMDB, Monitoring Clients, Zookeeper, etc. The steps for adding services to a cluster are listed below:

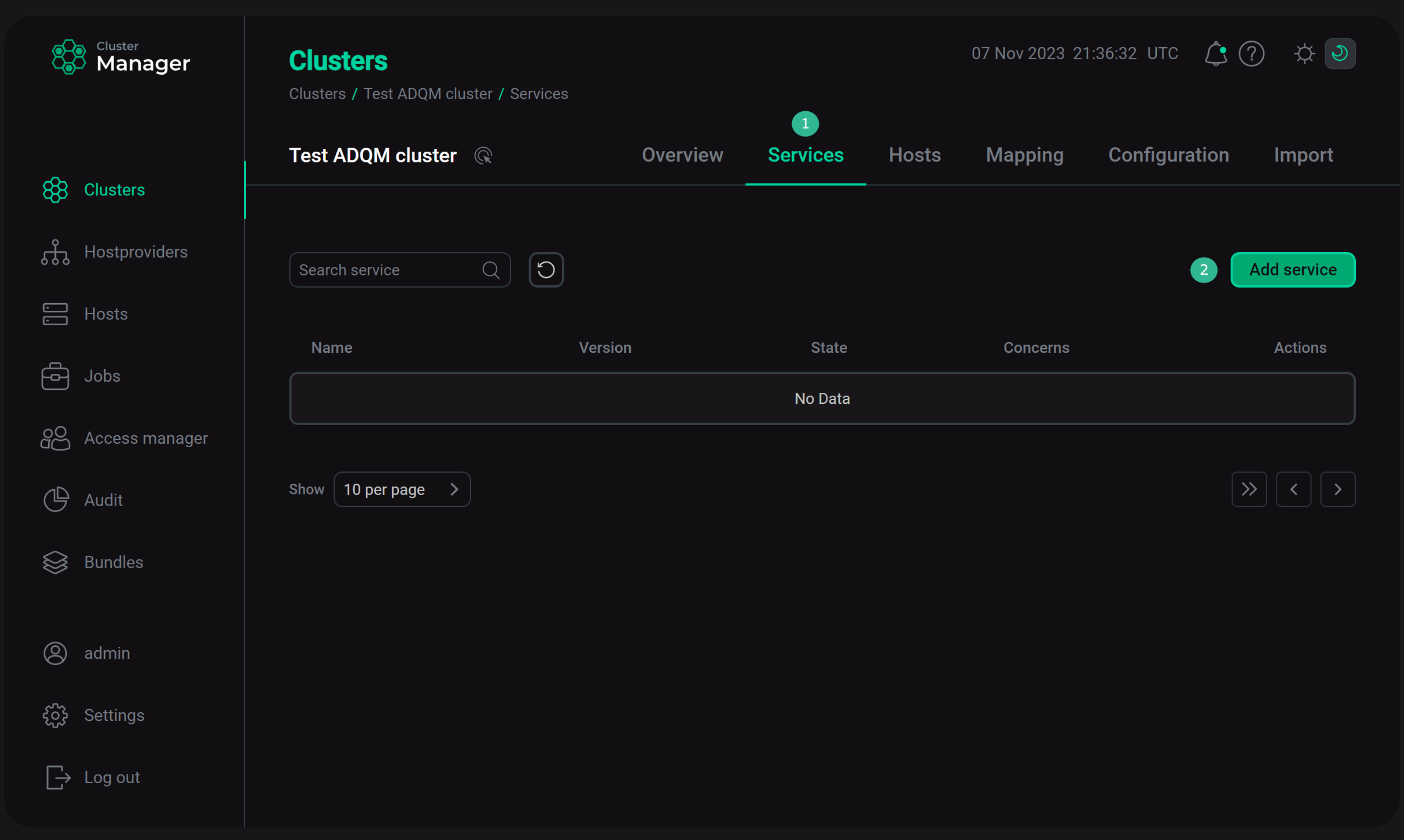
1. Select a cluster on the ***Clusters*** page. To do this, click a cluster name in the ***Name*** column.


Cluster
Manager
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Clusters
Clusters
07 Nov 2023 21:27:25 UTC
Search cluster name
Status
All
Product
All
Create cluster
click
Show
10 per page
1

Name	State	Product	Version	Description	Concerns	Actions
Test ADQM cluster	created	ADQM	23.3.2.37_arenadata1_b1-1			

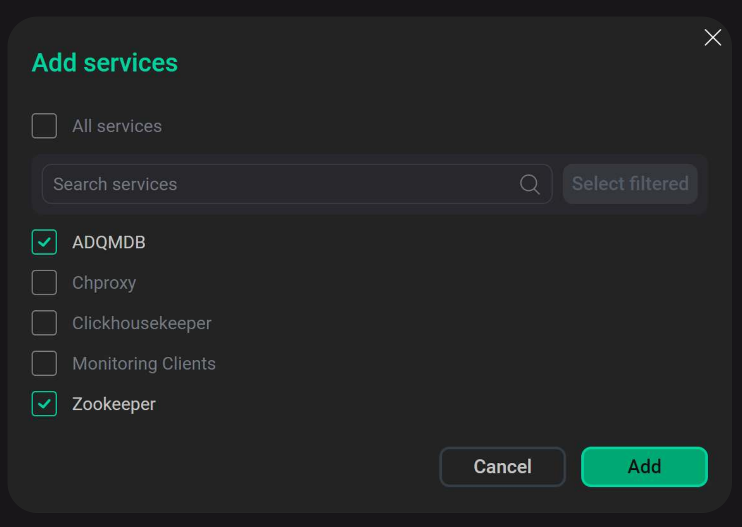
Select a cluster

2. Open the ***Services*** tab on the cluster page and click ***Add service***.


Cluster
Manager
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Clusters
Clusters / Test ADQM cluster / Services
07 Nov 2023 21:36:32 UTC
Test ADQM cluster
Overview
1
Services
Hosts
Mapping
Configuration
Import
Search service
2
Add service
Name
Version
State
Concerns
Actions
No Data
Show
10 per page

Switch to adding services

3. In the opened dialog, select services that should be added to the cluster and click ***Add***.


Add services
Search services
Select filtered
Cancel
Add

☐ All services

☑ ADQMDB
☐ Chproxy
☐ Clickhousekeeper
☐ Monitoring Clients
☑ Zookeeper

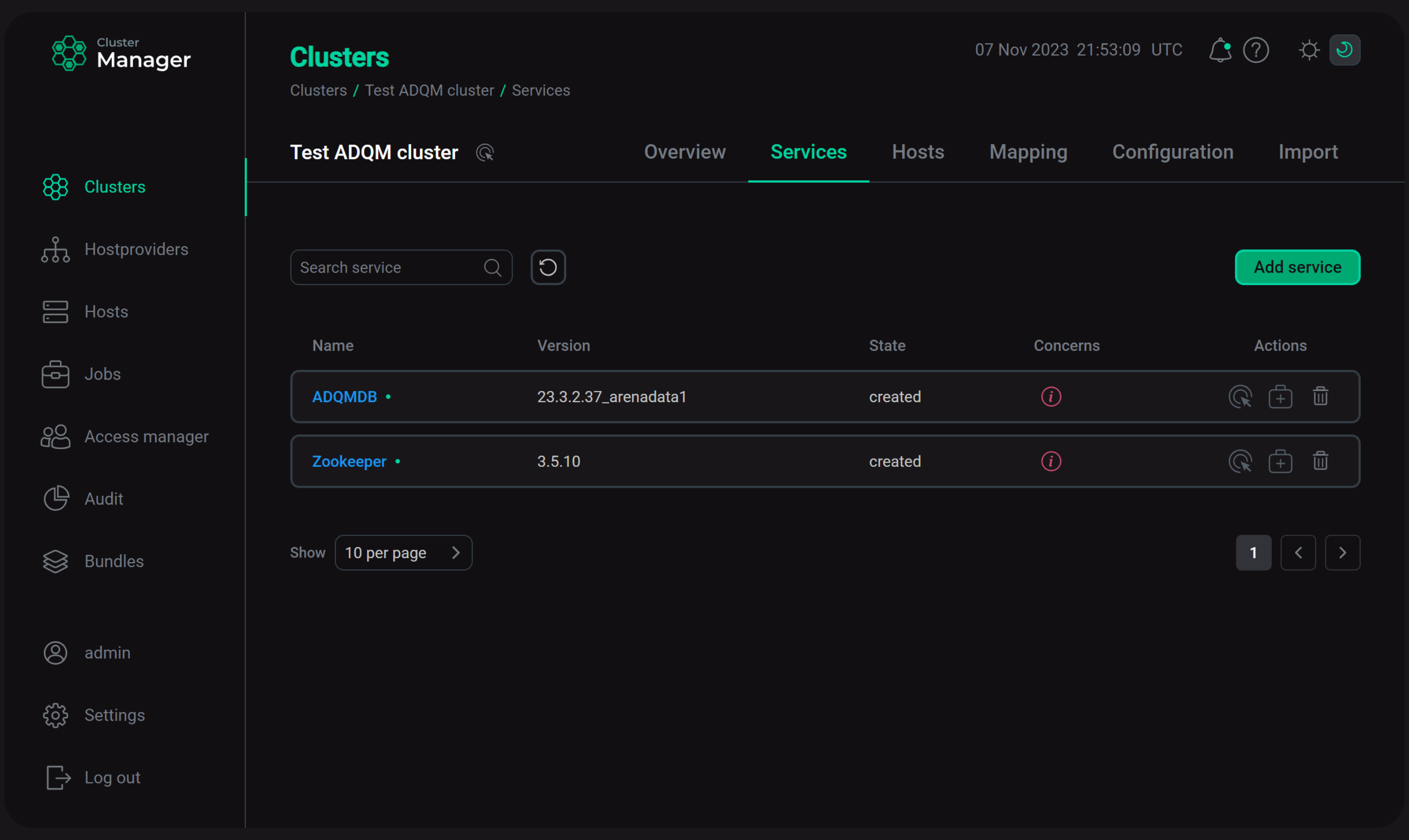
Select services

The brief description of available services is listed below.

Services that can be added to the ADQM cluster

Service	Mandatory	Purpose	Availability
ADQMDB	Yes	ClickHouse is a column-oriented database management system (DBMS) for online analytical processing of queries (OLAP)	Community/Enterprise
Zookeeper	No	Centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services	Community/Enterprise
Clickhousekeeper	No	Coordination system for data replication and distributed DDL queries execution (an alternative to ZooKeeper)	Community/Enterprise
Monitoring Clients	No	Agents that send information on hosts to the monitoring system, that is based on such services as Diamond, Graphite, and Grafana. The service should be added after the monitoring cluster is installed and configured, if monitoring of the current cluster is planned	Community/Enterprise
Monitoring	No	Service for collecting and storing ADQM cluster monitoring metrics based on the Prometheus monitoring system and Grafana service	Community/Enterprise
Chproxy	No	HTTP proxy and load balancer for ClickHouse	Enterprise

4. As a result, the added services are displayed on the ***Services*** tab.


Cluster
Manager
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Clusters
Clusters / Test ADQM cluster / Services
07 Nov 2023 21:53:09 UTC
Test ADQM cluster
Overview
Services
Hosts
Mapping
Configuration
Import
Search service
Add service
Show
10 per page
1

Name	Version	State	Concerns	Actions
ADQMDB	23.3.2.37_arenadata1	created		
Zookeeper	3.5.10	created		

The result of successful adding services to a cluster

NOTE

You can also add services later. The process of adding new services to already running cluster does not differ from installing a service from scratch.

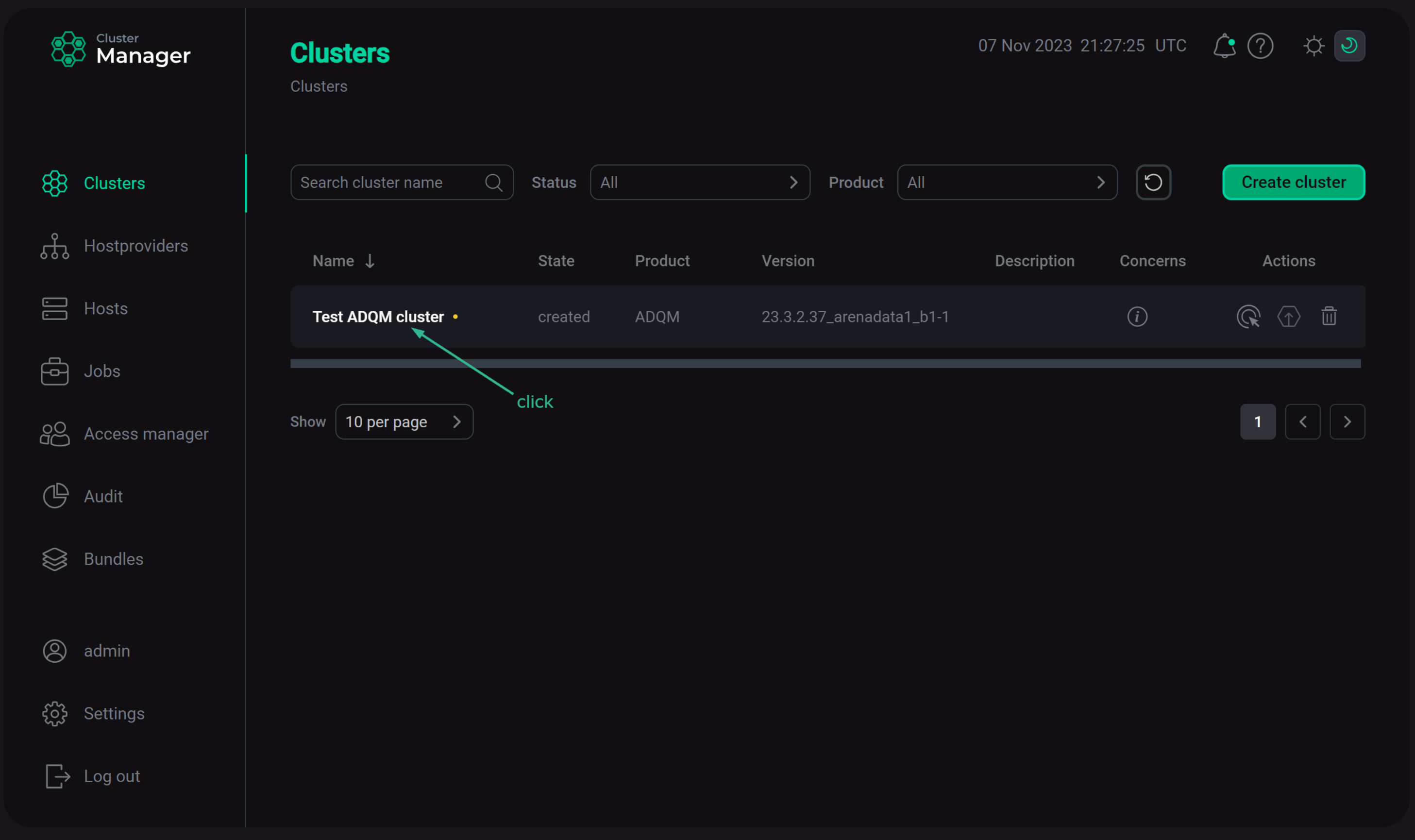
Add hosts to a cluster

Daria Barysheva, Elena Dvoryadkina

To Table of Contents

The steps for adding hosts to a cluster are listed below:

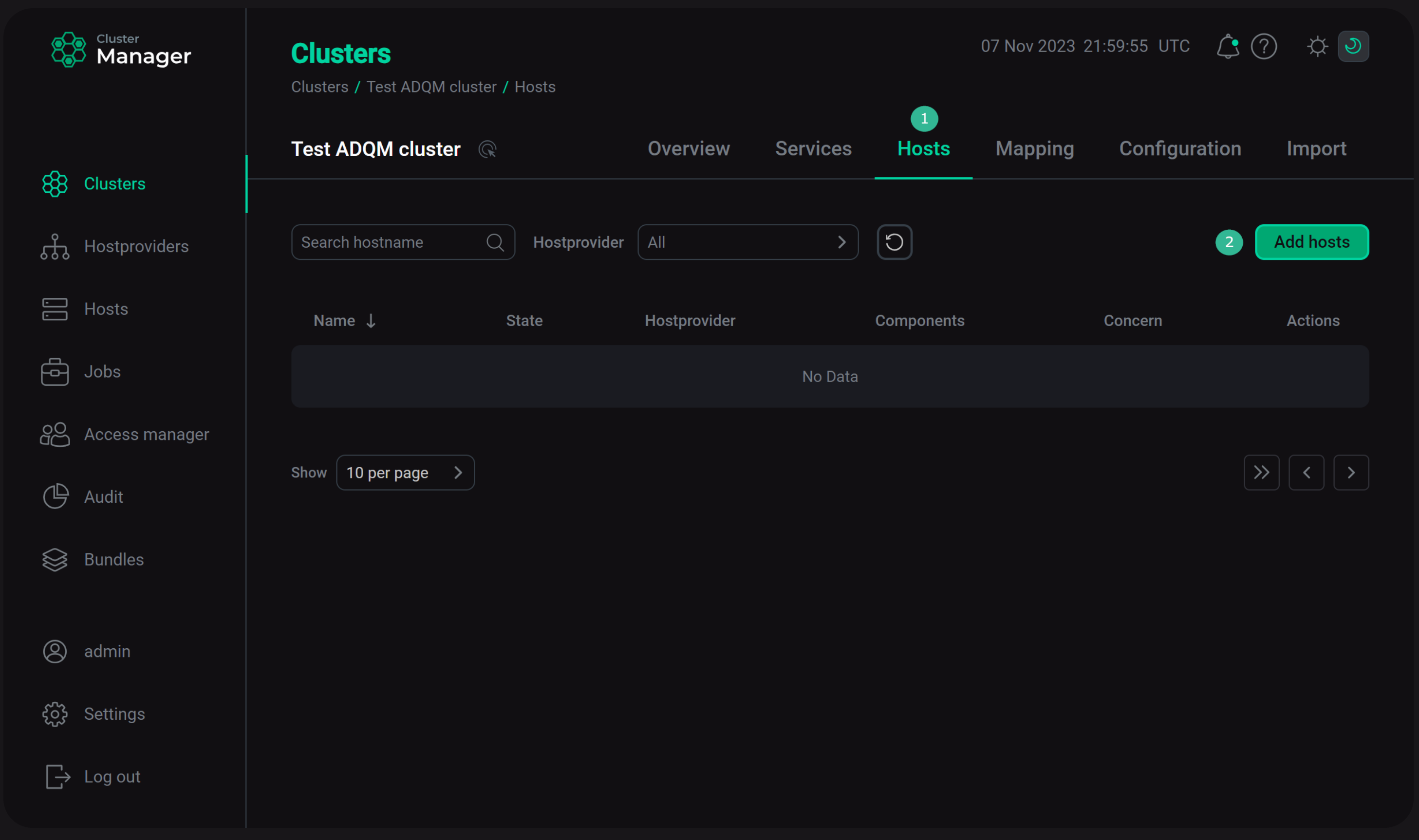
1. Select a cluster on the ***Clusters*** page. To do this, click a cluster name in the ***Name*** column.


Cluster
Manager
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Clusters
Clusters
07 Nov 2023 21:27:25 UTC
Search cluster name
Status
All
Product
All
Create cluster
click
Show
10 per page
1

Name ↓	State	Product	Version	Description	Concerns	Actions
Test ADQM cluster	created	ADQM	23.3.2.37_arenadata1_b1-1			

Select a cluster

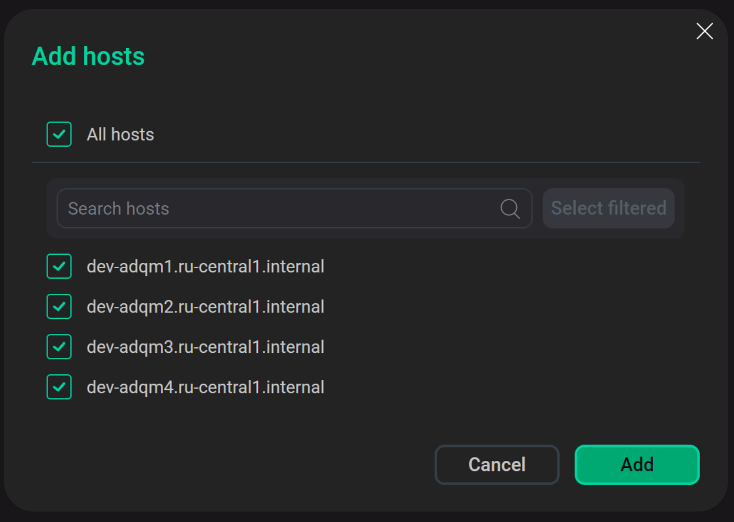
2. Open the ***Hosts*** tab on the cluster page and click ***Add hosts***.


Cluster
Manager
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Clusters
Clusters / Test ADQM cluster / Hosts
07 Nov 2023 21:59:55 UTC
Test ADQM cluster
Overview
Services
1
Hosts
Mapping
Configuration
Import
Search hostname
Hostprovider
All
2
Add hosts
Show
10 per page

Name ↓	State	Hostprovider	Components	Concern	Actions
No Data					

Switch to adding hosts

3. In the opened dialog, select hosts that should be added to the cluster and click ***Add***.

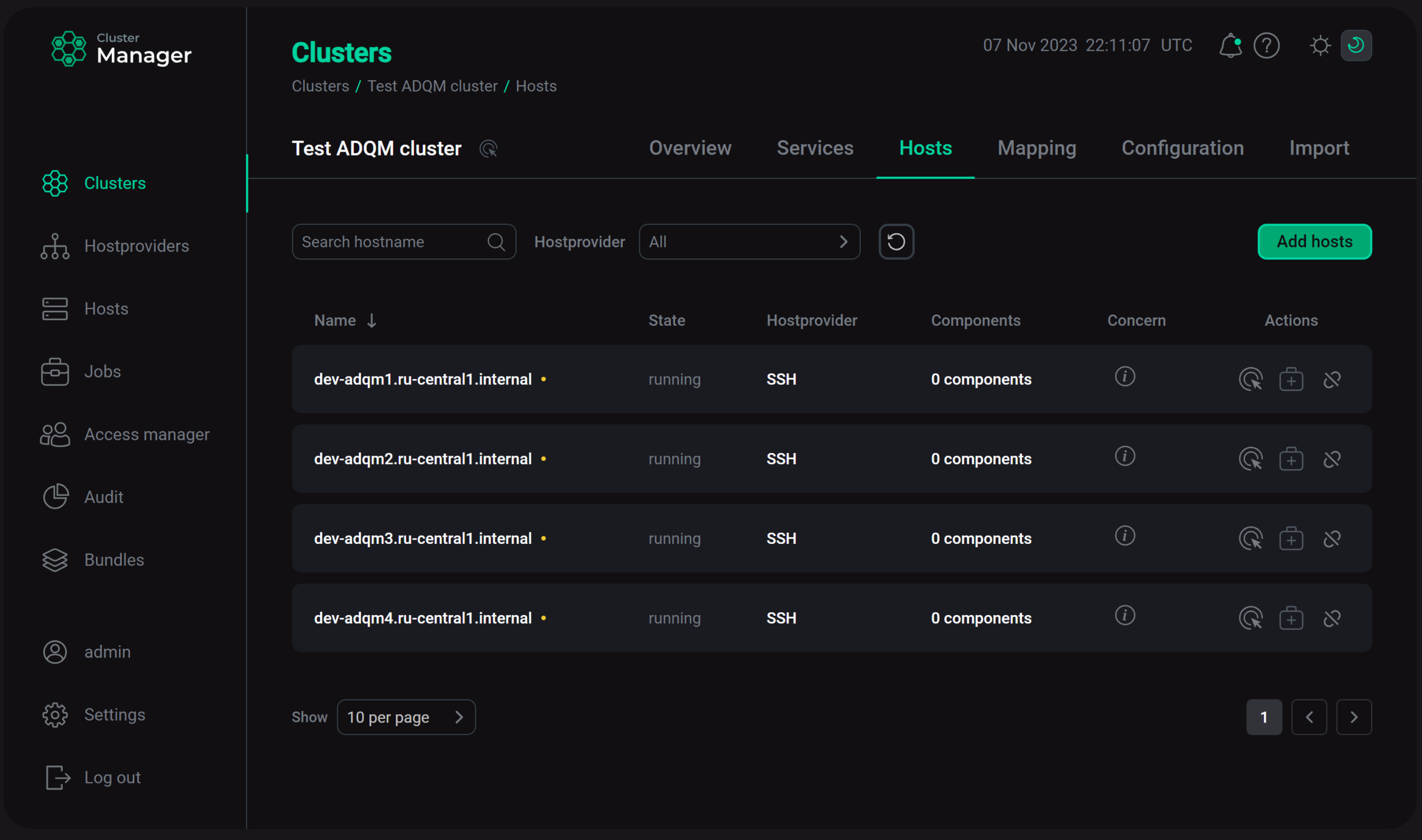

Add hosts
Search hosts
Select filtered
Cancel
Add

☑ All hosts

☑ dev-adqm1.ru-central1.internal
☑ dev-adqm2.ru-central1.internal
☑ dev-adqm3.ru-central1.internal
☑ dev-adqm4.ru-central1.internal

Select hosts

4. As a result, the added hosts are displayed on the ***Hosts*** tab.


Cluster
Manager
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Clusters
Clusters / Test ADQM cluster / Hosts
07 Nov 2023 22:11:07 UTC
Test ADQM cluster
Overview
Services
Hosts
Mapping
Configuration
Import
Search hostname
Hostprovider
All
Add hosts
Show
10 per page
1

Name ↓	State	Hostprovider	Components	Concern	Actions
dev-adqm1.ru-central1.internal	running	SSH	0 components		
dev-adqm2.ru-central1.internal	running	SSH	0 components		
dev-adqm3.ru-central1.internal	running	SSH	0 components		
dev-adqm4.ru-central1.internal	running	SSH	0 components		

The result of successful adding hosts to a cluster

Add components

Daria Barysheva, Elena Dvoryadkina

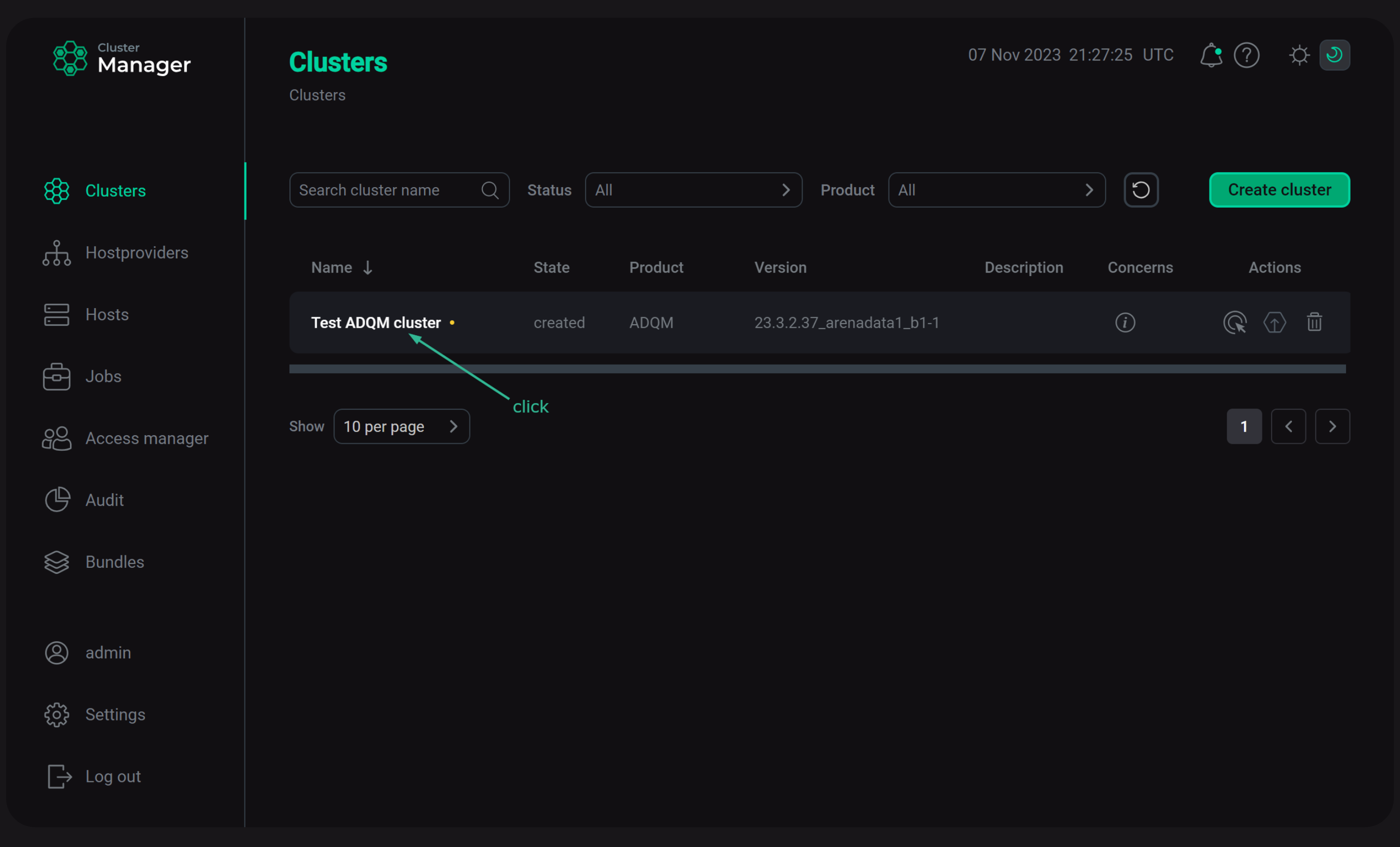
To Table of Contents

In ADCM a **component** means a part of a service that should be deployed on one or several cluster hosts.

> **IMPORTANT**
>
> Initially, there are no components on any of cluster hosts. You should map components to hosts manually.

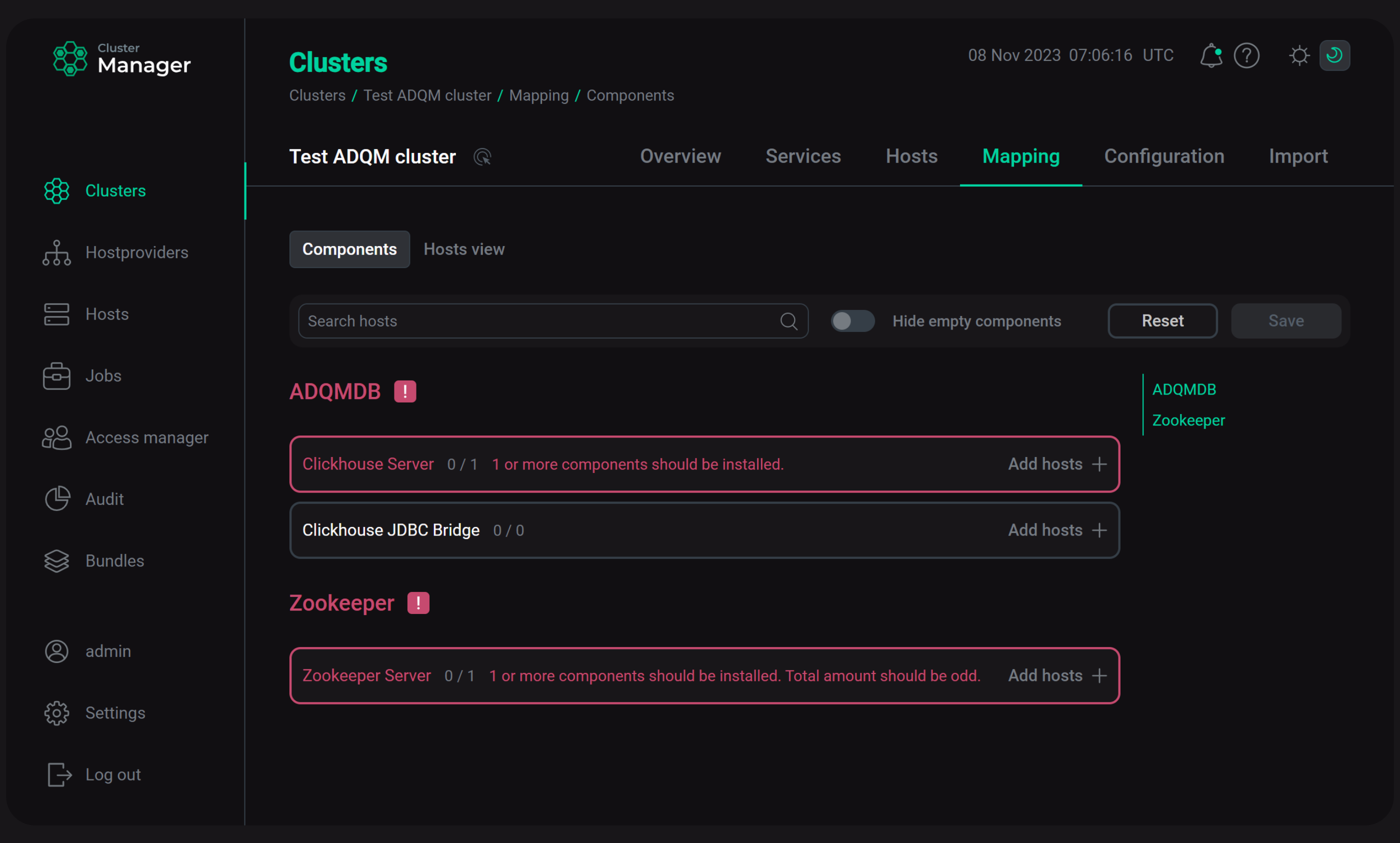
The steps for adding components to hosts are listed below:

1. Select a cluster on the ***Clusters*** page. To do this, click a cluster name in the ***Name*** column.


Cluster Manager
Clusters
07 Nov 2023 21:27:25 UTC
Create cluster
Name
State
Product
Version
Description
Concerns
Actions
Test ADQM cluster
created
ADQM
23.3.2.37_arenadata1_b1-1
Click

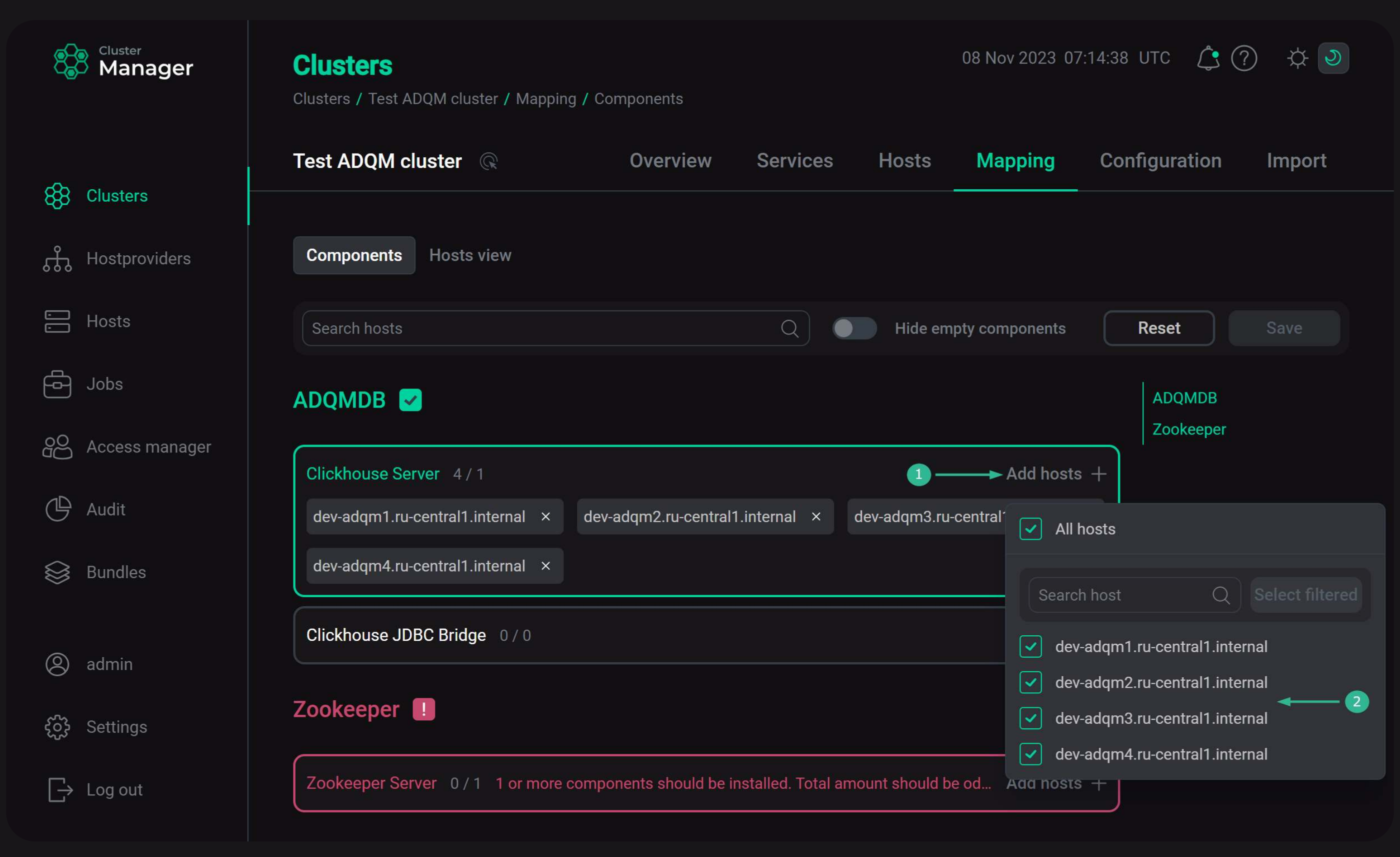
Select a cluster

2. Open the ***Mapping*** tab on the cluster page.


Clusters
Test ADQM cluster
Overview
Services
Hosts
Mapping
Configuration
Import
Components
Hosts view
ADQMDB
Clickhouse Server 0 / 1 1 or more components should be installed
Clickhouse JDBC Bridge 0 / 0
Zookeeper
Zookeeper Server 0 / 1 1 or more components should be installed. Total amount should be odd.
Add hosts

Switch to mapping service components

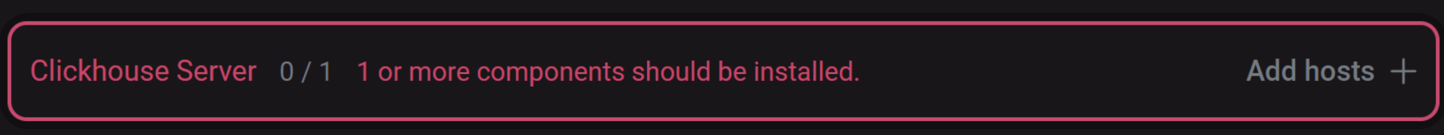
3. In the window that opens, map components to hosts. To assign hosts for each component, click ***Add hosts*** and select hosts in the pop-up window.


Clickhouse Server 4 / 1
Add hosts
1
All hosts
Search host
Select filtered
dev-adqm1.ru-central1.internal
dev-adqm2.ru-central1.internal
dev-adqm3.ru-central1.internal
dev-adqm4.ru-central1.internal
2

Select hosts for a component

Mandatory components

Each service can have mandatory and optional components. Mandatory components are highlighted in red. The ADCM UI also displays how many mandatory components should be installed.


Clickhouse Server 0 / 1 1 or more components should be installed.
Add hosts

Mandatory components

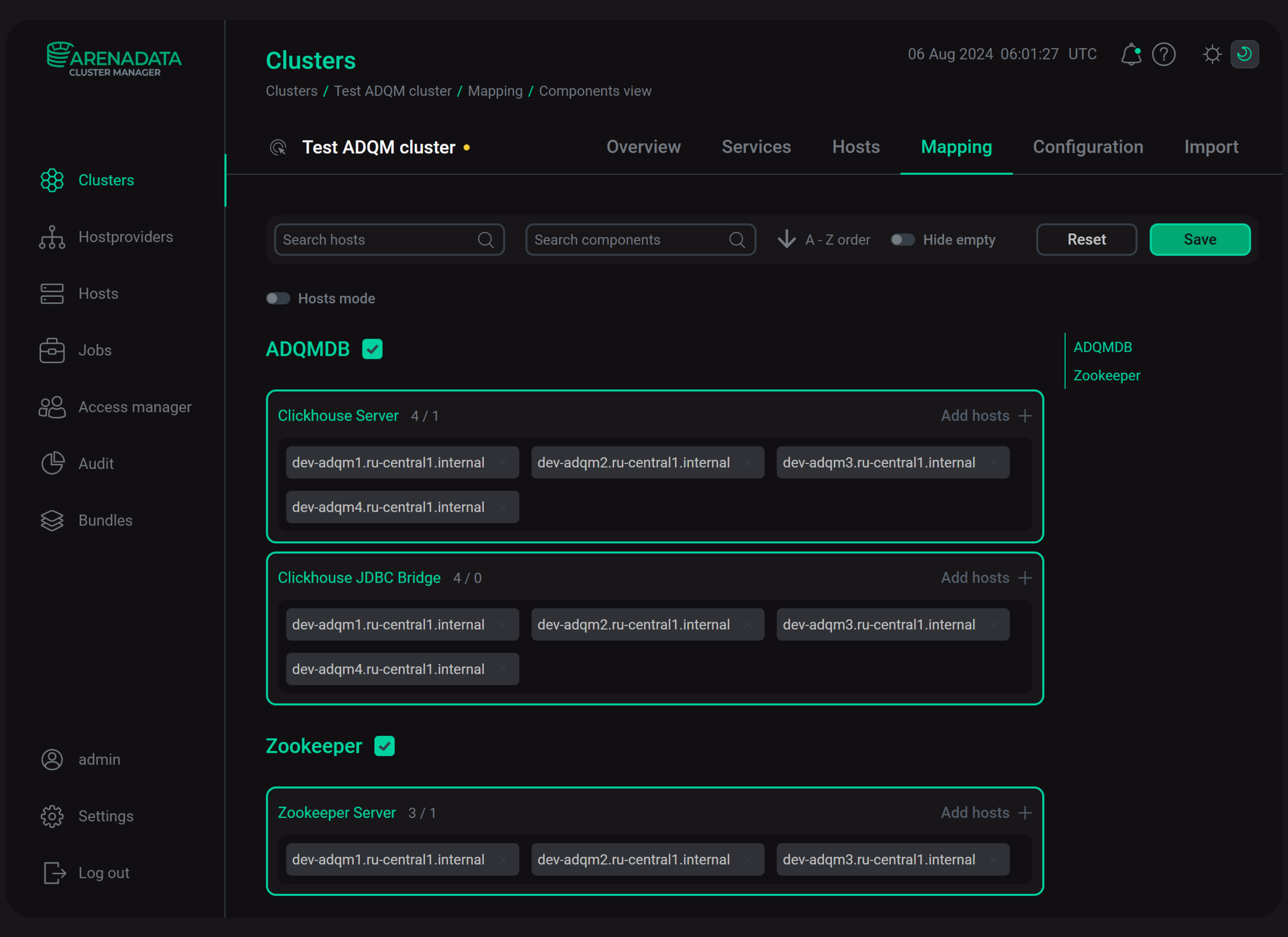
> **CAUTION**
>
> Without assigning a necessary number of hosts to mandatory components, the component mapping cannot be saved.

The typical scheme of components distribution for ADQM clusters is described in the table below.

ADQM components distribution

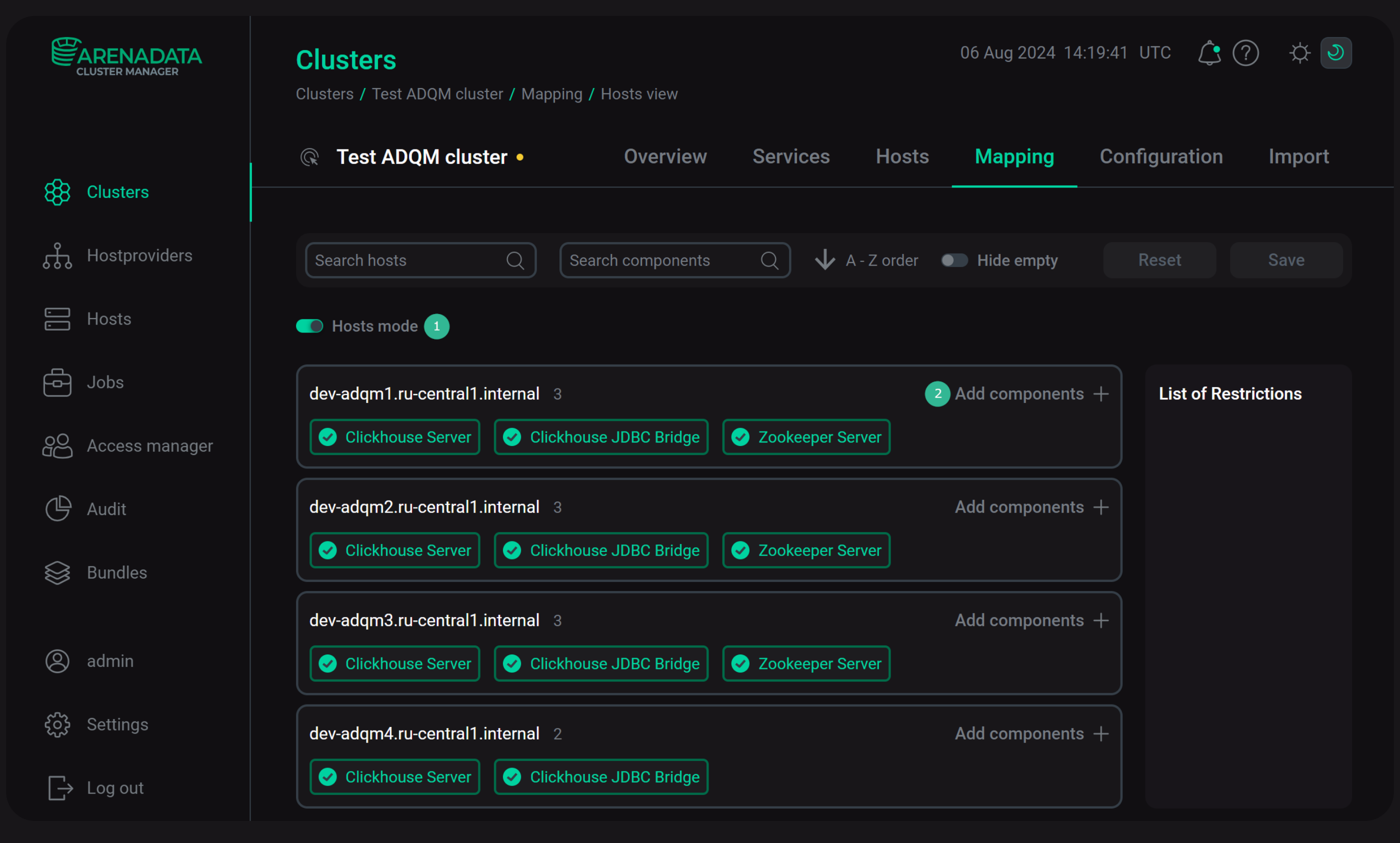
Host	Service	Component	Notes
Any host	ADQMDB	Clickhouse Server	One or more components should be installed
All hosts with Clickhouse Server	ADQMDB	Clickhouse JDBC Bridge	If you install this component (optionally), add it to all hosts in the cluster where the Clickhouse Server component is installed
Any host	Zookeeper	Zookeeper Server	One or more components should be installed. Total amount should be odd
Any host	Clickhousekeeper	Clickhouse Keeper Server	One or more components should be installed. Total amount should be odd
Any host	Chproxy	Chproxy Proxy Server	One or more components should be installed

4. After component distribution is completed, click ***Save***.


ARENADATA CLUSTER MANAGER
Clusters
06 Aug 2024 06:01:27 UTC
Test ADQM cluster
Mapping
Search hosts
Search components
A - Z order
Hide empty
Reset
Save
Hosts mode
ADQMDB
Clickhouse Server 4 / 1
Clickhouse JDBC Bridge 4 / 0
Zookeeper
Zookeeper Server 3 / 1
dev-adqm1.ru-central1.internal
dev-adqm2.ru-central1.internal
dev-adqm3.ru-central1.internal
dev-adqm4.ru-central1.internal

Save mapping of components

Starting with ADCM 2.2.0, you can use another mapping mode — by hosts. To do this, switch on the ***Hosts mode*** toggle and map components to hosts by clicking ***Add components*** for each host sequentially.


Clusters / Test ADQM cluster / Mapping / Hosts view
06 Aug 2024 14:19:41 UTC
Hosts mode
1
2
Add components
List of Restrictions
dev-adqm1.ru-central1.internal 3
dev-adqm2.ru-central1.internal 3
dev-adqm3.ru-central1.internal 3
dev-adqm4.ru-central1.internal 2
Clickhouse Server
Clickhouse JDBC Bridge
Zookeeper Server

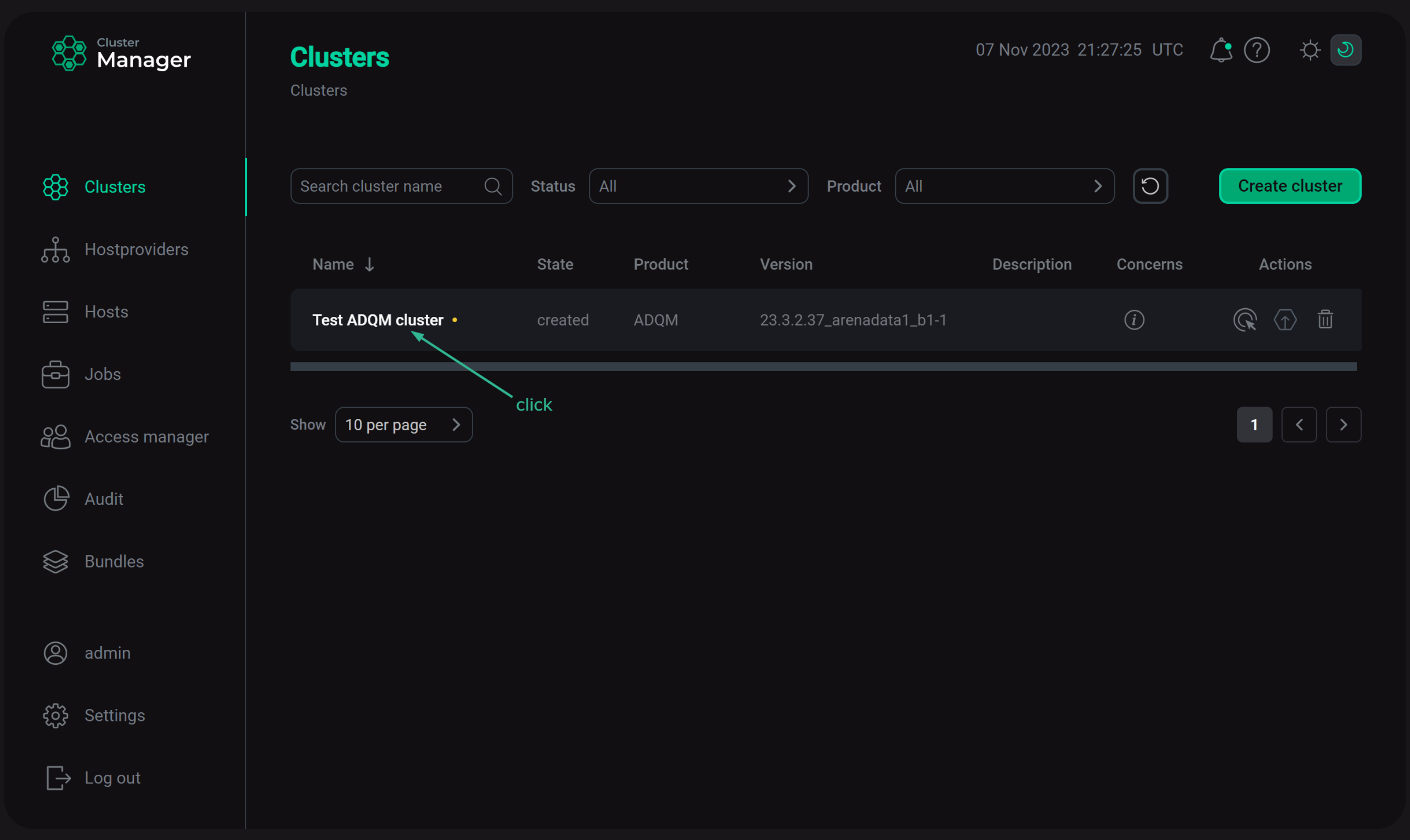
Use mapping by hosts

Configure services

Daria Barysheva, Elena Dvoryadkina

To Table of Contents

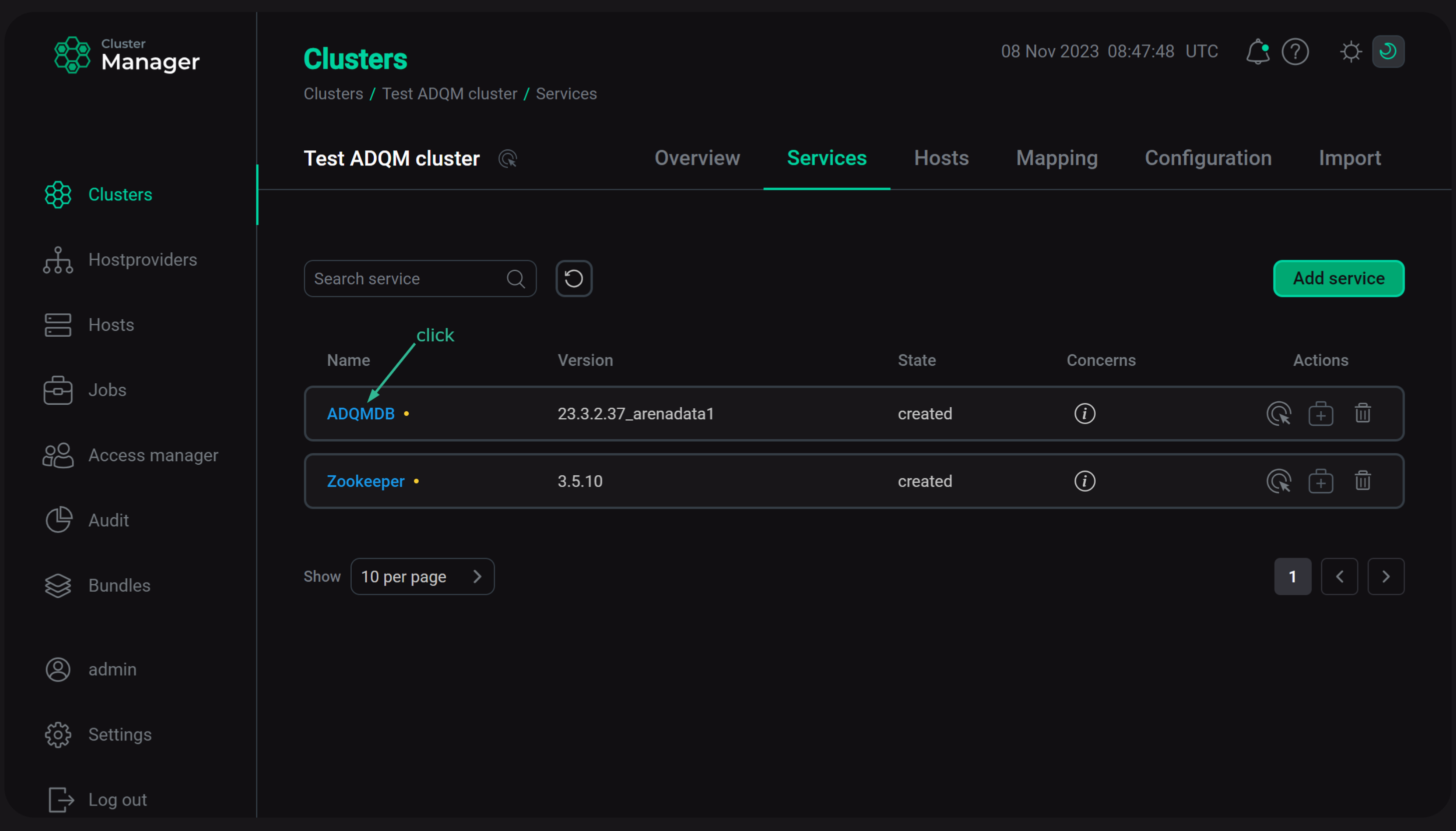
1. Select a cluster on the ***Clusters*** page. To do this, click a cluster name in the ***Name*** column.


Cluster Manager
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Clusters
Clusters
07 Nov 2023 21:27:25 UTC
Search cluster name
Status
All
Product
All
Create cluster
click
Show
10 per page
1

Name	State	Product	Version	Description	Concerns	Actions
Test ADQM cluster	created	ADQM	23.3.2.37_arenadata1_b1-1			

Select a cluster

2. Open the ***Services*** tab on the cluster page, find the row that contains the desired service, and click its name in the ***Name*** column.

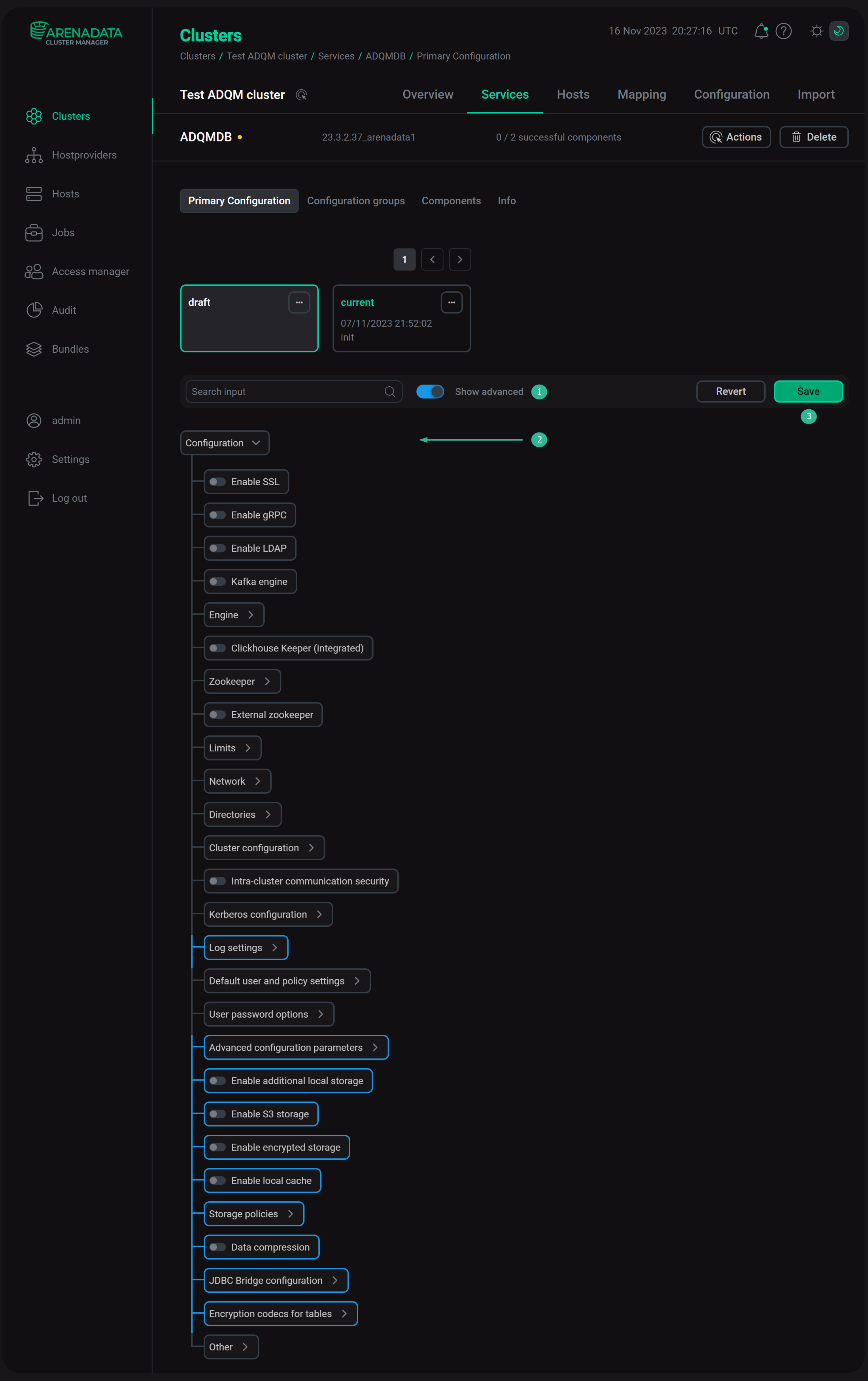

Cluster Manager
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Clusters
Clusters / Test ADQM cluster / Services
08 Nov 2023 08:47:48 UTC
Test ADQM cluster
Overview
Services
Hosts
Mapping
Configuration
Import
Search service
Add service
click
Show
10 per page
1

Name	Version	State	Concerns	Actions
ADQMDB	23.3.2.37_arenadata1	created		
Zookeeper	3.5.10	created		

Switch to configuring a service

3. In the window that opens:

- Open the ***Primary Configuration*** tab.
- Switch on the ***Show advanced*** toggle.
- Fill in configuration parameters of the selected service. The fields highlighted in red are mandatory.
- Click ***Save***.


ARENADATA CLUSTER MANAGER
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Clusters
Clusters / Test ADQM cluster / Services / ADQMDB / Primary Configuration
16 Nov 2023 20:27:16 UTC
Test ADQM cluster
Overview
Services
Hosts
Mapping
Configuration
Import
ADQMDB
23.3.2.37_arenadata1
0 / 2 successful components
Actions
Delete
Primary Configuration
Configuration groups
Components
Info
1
draft
current
07/11/2023 21:52:02
init
Search input
Show advanced
1
Revert
Save
3
2
Configuration
Enable SSL
Enable gRPC
Enable LDAP
Kafka engine
Engine
Clickhouse Keeper (integrated)
Zookeeper
External zookeeper
Limits
Network
Directories
Cluster configuration
Intra-cluster communication security
Kerberos configuration
Log settings
Default user and policy settings
User password options
Advanced configuration parameters
Enable additional local storage
Enable S3 storage
Enable encrypted storage
Enable local cache
Storage policies
Data compression
JDBC Bridge configuration
Encryption codecs for tables
Other

Configure a service

NOTE

For more information about services configuration parameters, see Configuration parameters in the References.

Configure a cluster

Daria Barysheva, Elena Dvoryadkina

Contents

To Table of Contents

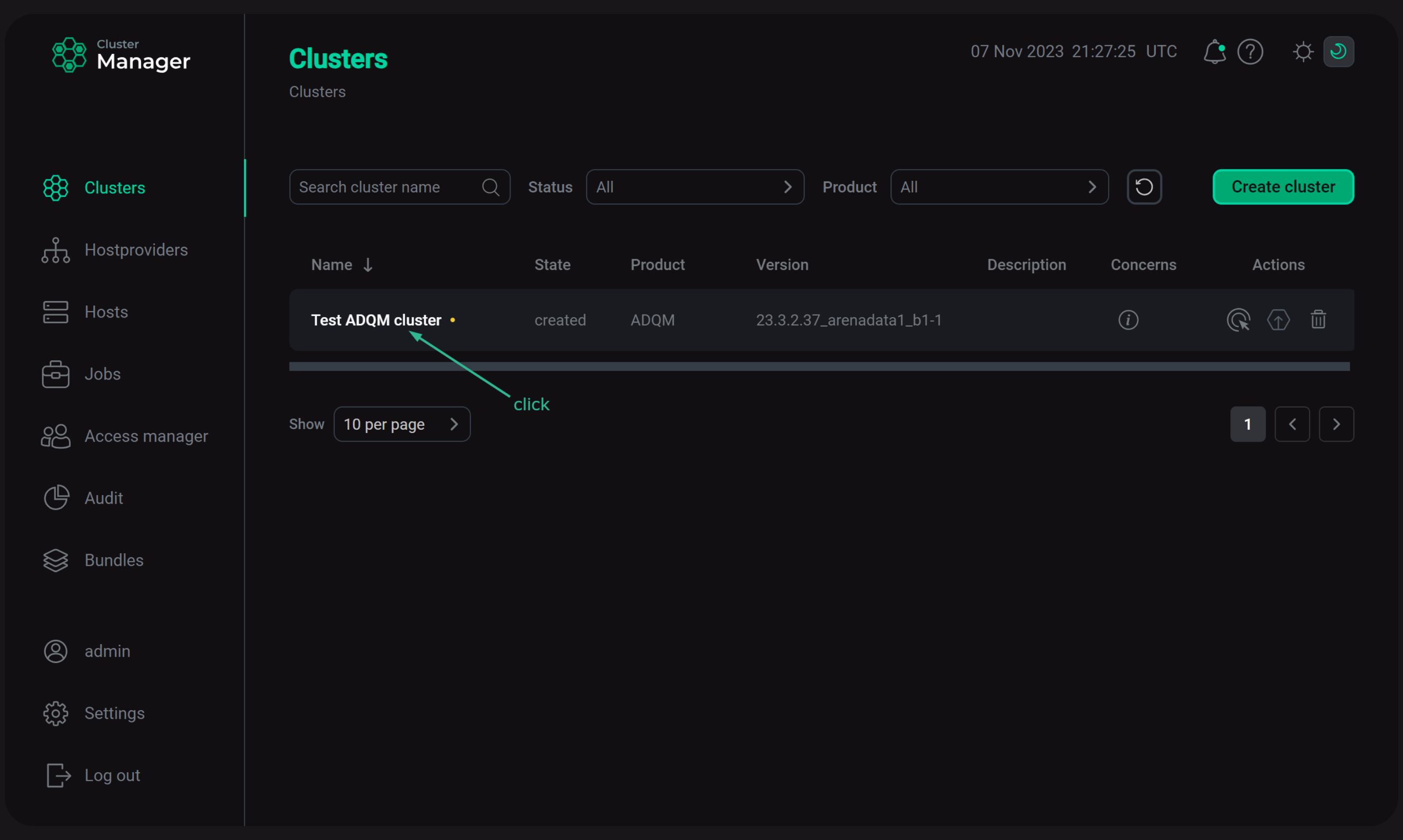
Primary configuration
Configuration groups
Ansible settings

IMPORTANT

There is usually no need to change cluster configuration parameters. You can leave all parameters at the default values.

After creation a new cluster, you can configure it by performing the following steps:

1. Select a cluster on the ***Clusters*** page. To do this, click a cluster name in the ***Name*** column.


Cluster
Manager
Clusters
Clusters
07 Nov 2023 21:27:25 UTC
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Search cluster name
Status
All
Product
All
Create cluster
click
Show
10 per page
1

Name ↓	State	Product	Version	Description	Concerns	Actions
Test ADQM cluster	created	ADQM	23.3.2.37_arenadata1_b1-1			

Select a cluster

2. Open the ***Configuration*** tab on the cluster page. The ***Configuration*** tab includes the following sections: ***Primary configuration***, ***Configuration groups***, ***Ansible settings***.

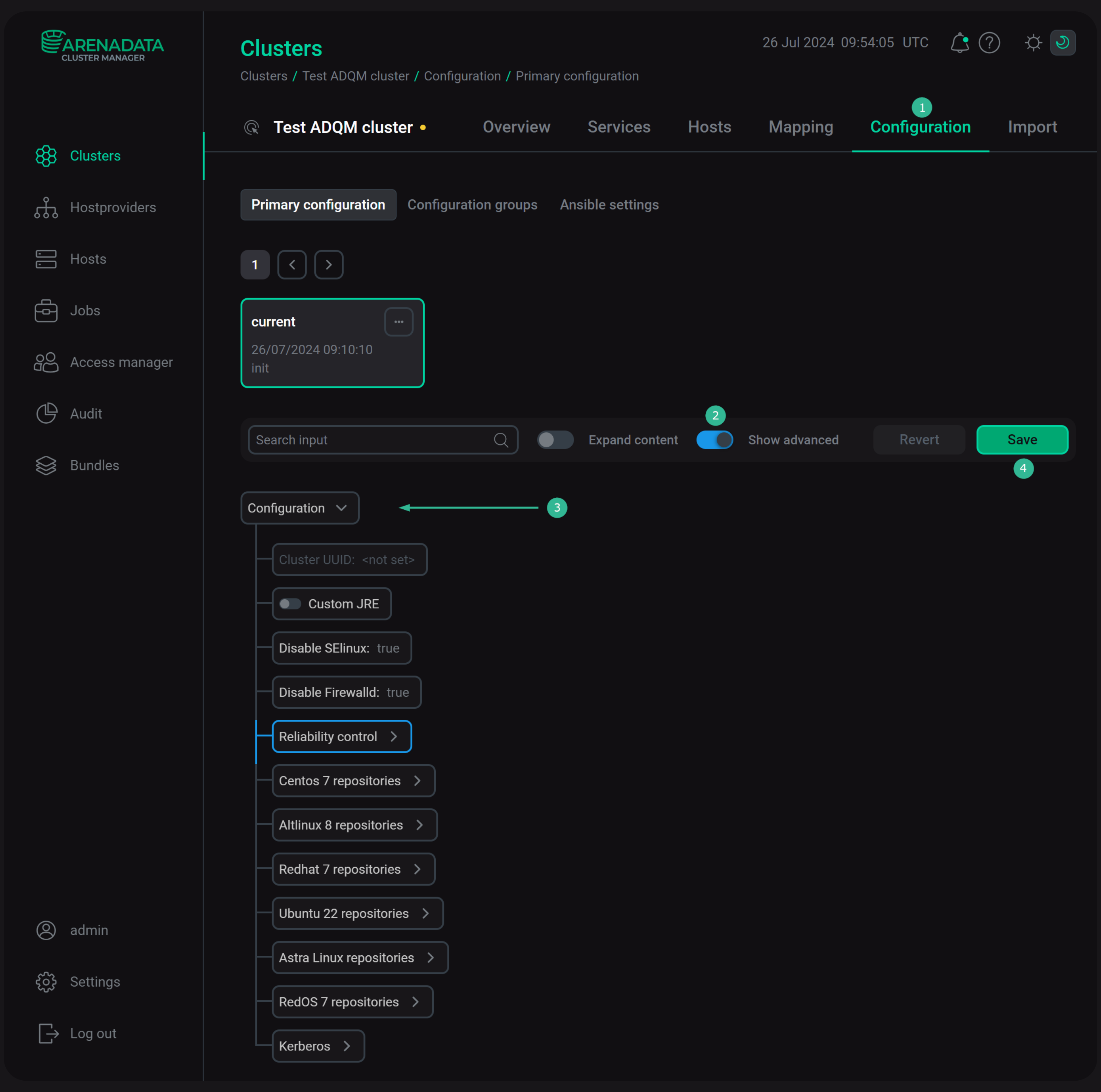
3. Fill in all necessary parameters on the selected tab and click ***Save***.

Primary configuration

The ***Configuration*** → ***Primary configuration*** tab allows you to configure the general cluster settings.

ADQM cluster configuration parameters are listed below:

- ***Custom JRE*** — the switcher that indicates whether to use a custom path to Java for ADQM services. When the switcher is activated, the ***JAVA_HOME*** field becomes available — enter a work Java path in this field. No system JRE packages will be installed.
- ***Disable SELinux*** — disables SElinux on hosts of the cluster (available starting with ADQM 22.8.4.7.1.b1).
- ***Disable Firewalld*** — disables firewalld on hosts of the cluster (available starting with ADQM 22.8.4.7.1.b1).
- ***Reliability control*** — the section with the following settings (visible if the ***Show advanced*** option is enabled):
  - ***retries*** — number of retries for cluster operations (`3` by default);
  - ***delay*** — delay between retries in seconds (`10` by default);
  - ***timeout*** — timeout for checks of network ports in seconds (`60` by default).
- Paths to the repositories that are used during the installation process — depending on the cluster operating system (Centos, Alt Linux, Red Hat, Ubuntu, Astra Linux, or RED OS):
  - ***monitoring*** — repository required to install monitoring;
  - ***zookeeper_repo*** — repository required to install Arenadata Zookeeper;
  - ***ADQM*** — repository required to install ADQM;
  - ***prometheus_repo*** — repository required to install Prometheus;
  - ***ADQM_arenadata_enterprise*** — repository required to install the Enterprise edition of ADQM.
- ***Kerberos*** — configuration for Kerberos authentication (some parameters are visible only with the ***Show advanced*** option enabled).


ARENADATA
CLUSTER MANAGER
Clusters
Clusters / Test ADQM cluster / Configuration / Primary configuration
26 Jul 2024 09:54:05 UTC
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Test ADQM cluster
Overview
Services
Hosts
Mapping
1
Configuration
Import
Primary configuration
Configuration groups
Ansible settings
1
current
26/07/2024 09:10:10
init
Search input
Expand content
2
Show advanced
Revert
Save
4
Configuration
3
Custom JRE
Reliability control
Centos 7 repositories
Altlinux 8 repositories
Redhat 7 repositories
Ubuntu 22 repositories
Astra Linux repositories
RedOS 7 repositories
Kerberos

Cluster UUID: <not set>
Disable SElinux: true
Disable Firewalld: true

Cluster configuration window

Configuration groups

The ***Configuration*** → ***Configuration groups*** tab is designed to set cluster configuration groups.

Ansible settings

The ***Configuration*** → ***Ansible settings*** tab allows you to set Ansible configuration options at the cluster level. The tab is available starting with ADCM 2.2.0.

Ansible settings

Name	Description	Default
forks	The number of parallel processes to spawn when communicating with remote hosts	5

NOTE

During offline installation these parameters are ignored. The data stored locally in the Enterprise Tools cluster is used instead.

Import ET settings

Daria Barysheva, Elena Dvoryadkina

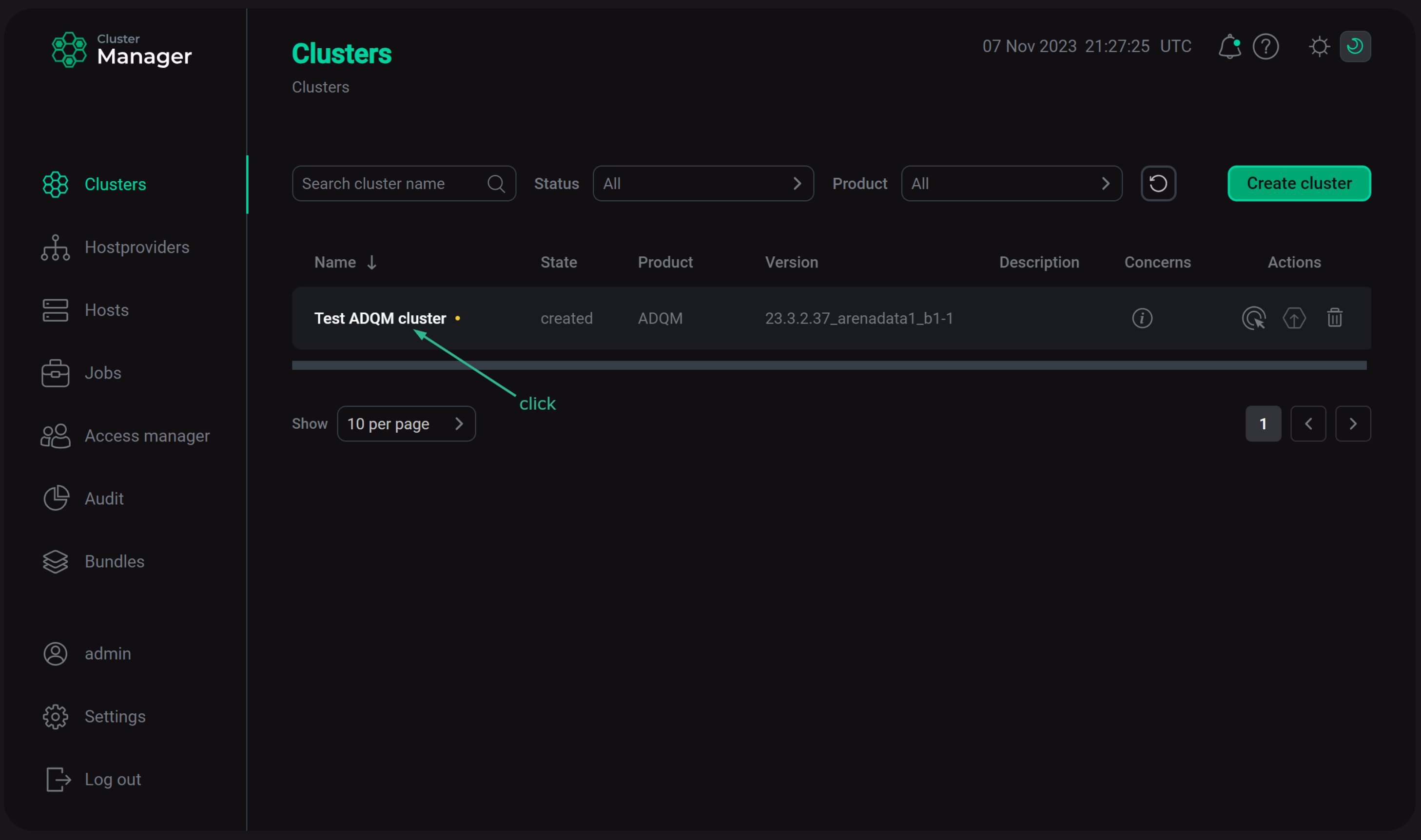
To Table of Contents

IMPORTANT

This step should be performed only in the case of offline installation.

In order to determine that a new cluster should be installed offline, it is necessary to integrate it with the previously created Enterprise Tools (ET) cluster. Follow the steps:

1. Select a cluster on the ***Clusters*** page. To do this, click a cluster name in the ***Name*** column.


Cluster Manager
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Clusters
Clusters
07 Nov 2023 21:27:25 UTC
Search cluster name
Status
All
Product
All
Create cluster
click
Show
10 per page
1

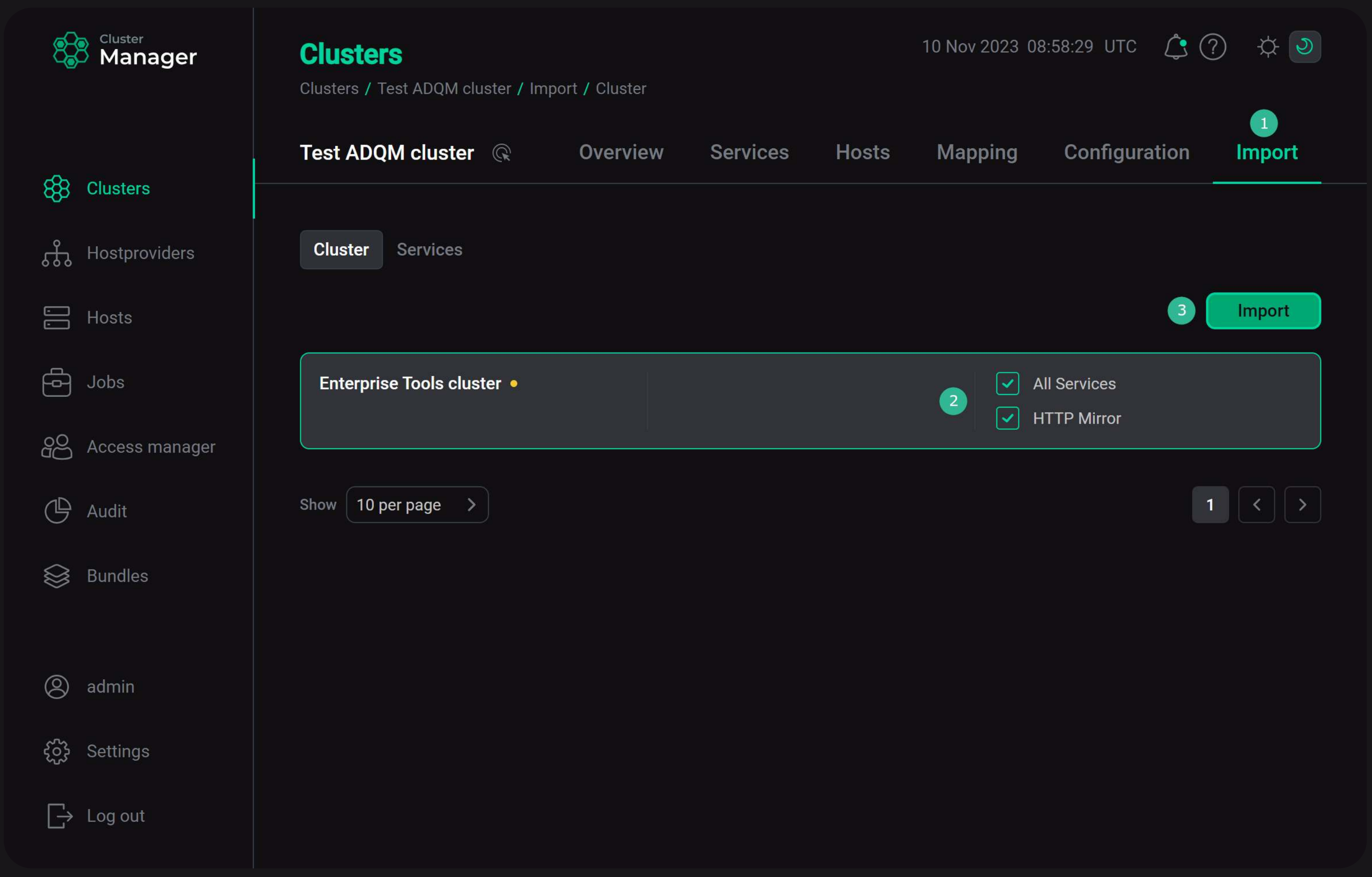
Name ↓	State	Product	Version	Description	Concerns	Actions
Test ADQM cluster	created	ADQM	23.3.2.37_arenadata1_b1-1			

Select a cluster

2. Open the ***Import*** tab on the cluster page.

3. Select the ***HTTP Mirror*** service of the previously created ET cluster.

4. Click ***Import***.


Cluster Manager
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Clusters
10 Nov 2023 08:58:29 UTC
Clusters / Test ADQM cluster / Import / Cluster
Test ADQM cluster
Overview
Services
Hosts
Mapping
Configuration
1
Import
Cluster
Services
3
Import
Enterprise Tools cluster
2
☑ All Services
☑ HTTP Mirror
Show
10 per page
1

Integrate with the ET cluster

NOTE

After successful import, the data stored locally in the Enterprise Tools cluster can be used at the cluster installation stage. So there is no need to extract this data via Internet.

Install a cluster

Daria Barysheva, Elena Dvoryadkina

Contents

To Table of Contents

- Step 1. Run installation
  - Whole cluster
  - Single services
- Step 2. View the results of installation
- Step 3. Check the cluster state after installation

Step 1. Run installation

There are two ways to install cluster services:

1. **Whole cluster**. In this method all services are installed automatically one by one after applying the ***Install*** action to a cluster.
2. **Single services**. In this method each service is installed manually after applying one or more actions to it. This way is also suitable for installing new services in the already deployed cluster.

In both cases each service starts automatically after installation.

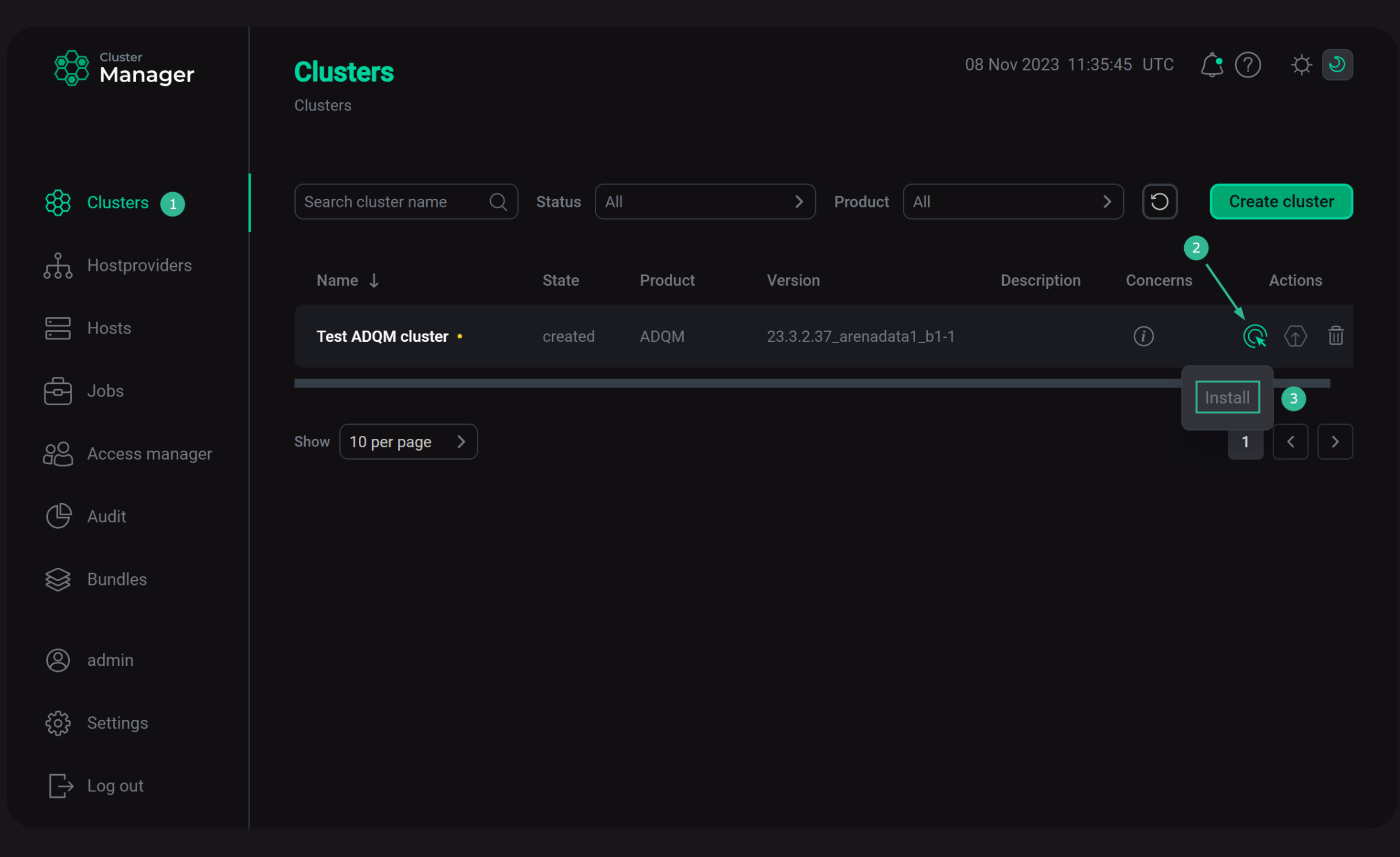
NOTE

In case of errors, you can find logs on the ***Jobs*** page.

Whole cluster

To install all cluster services within a single action, follow the steps:

1. Apply the ***Install*** action to the cluster selected on the ***Clusters*** page by clicking the icon in the ***Actions*** column.

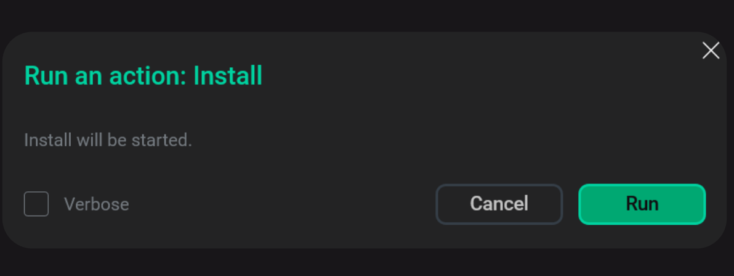


Cluster Manager
Clusters
Clusters
08 Nov 2023 11:35:45 UTC
Clusters
1
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Search cluster name
Status
All
Product
All
Create cluster
2
Install
3
Show
10 per page
1

Name	State	Product	Version	Description	Concerns	Actions
Test ADQM cluster	created	ADQM	23.3.2.37_arenadata1_b1-1			

Switch to the whole cluster installation

2. Verify the requested action in the opened window.



Run an action: Install

Install will be started.

☐ Verbose

Verify the action

3. Wait until the job is completed.

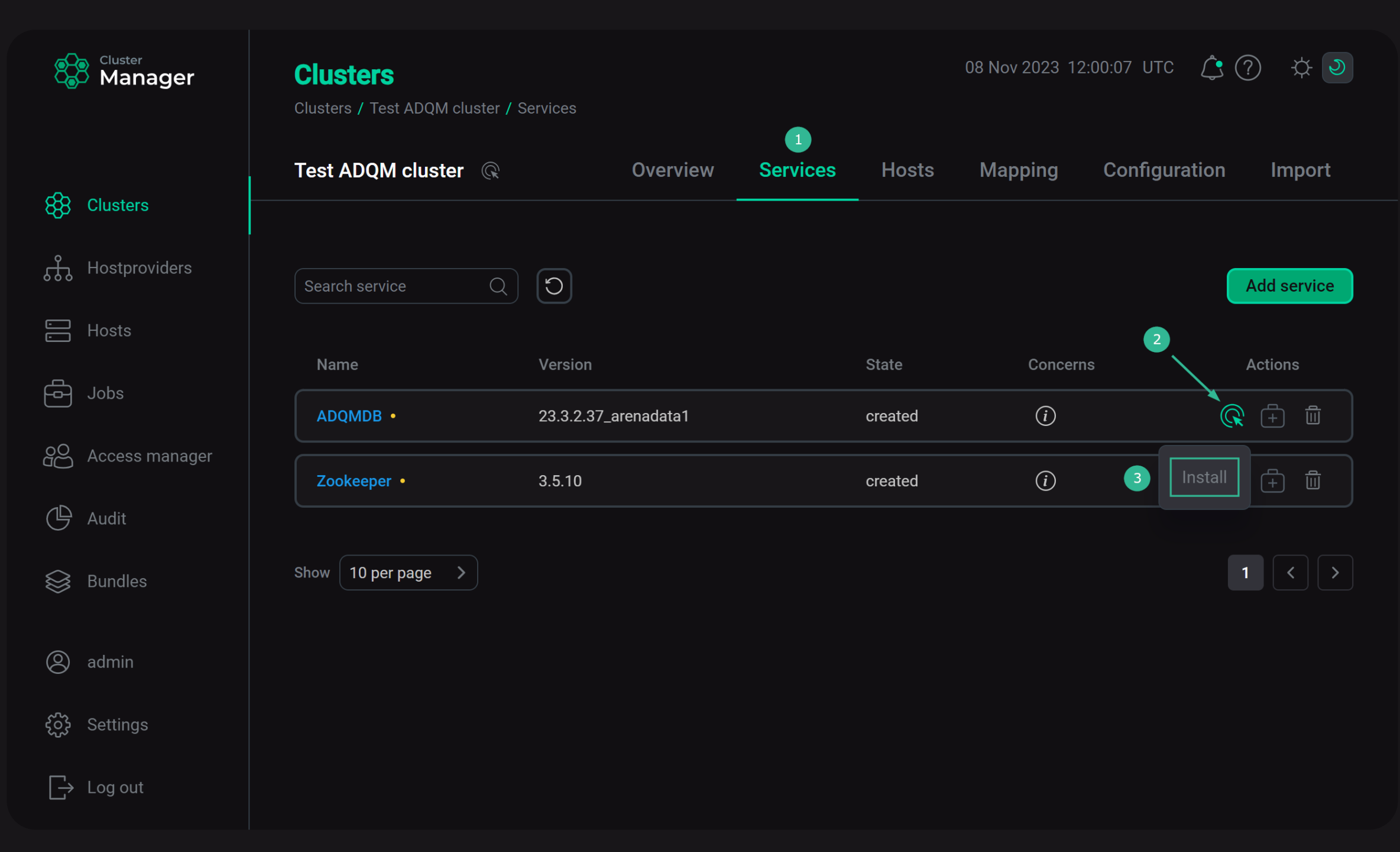
Single services

To install one or several cluster services manually, add these services to the cluster and then perform the actions described below.

CAUTION

Notice that services are installed one by one. Do not install another service until the installation of the previous one is completed.

1. Open the ***Services*** tab on the cluster page. For each service that should be installed, click the icon in the ***Actions*** column and select the ***Install*** action.

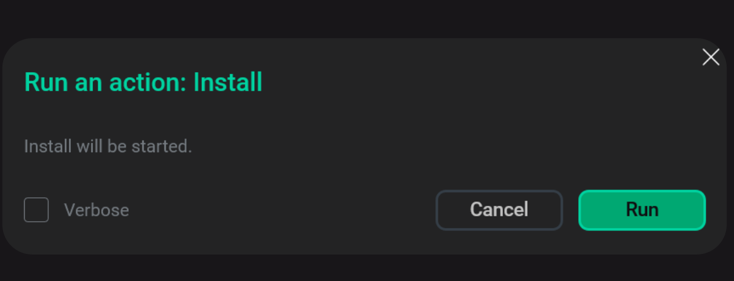


Cluster Manager
Clusters
Clusters / Test ADQM cluster / Services
08 Nov 2023 12:00:07 UTC
Test ADQM cluster
Overview
1
Services
Hosts
Mapping
Configuration
Import
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Search service
Add service
2
3
Install
Show
10 per page
1

Name	Version	State	Concerns	Actions
ADQMDB	23.3.2.37_arenadata1	created		
Zookeeper	3.5.10	created		

Switch to installation of single services

2. Verify the action in the opened window.



Run an action: Install

Install will be started.

☐ Verbose

Verify the action

3. Wait until the job is completed. Then check that the service state has changed from `created` to `installed`.
4. Repeat the previous actions for other services that should be installed.

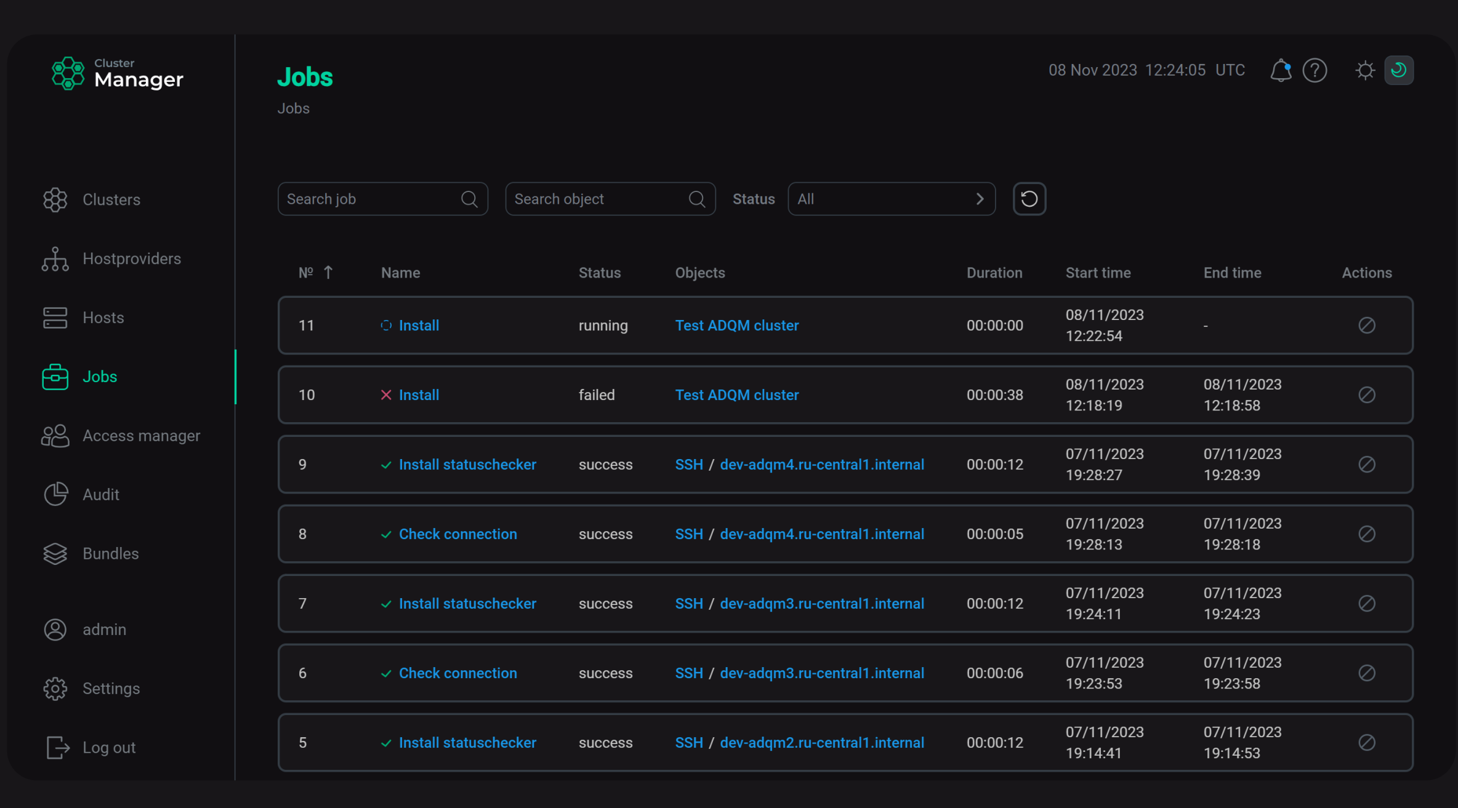
IMPORTANT

When you install services manually, one by one, use the following order:

1. Zookeeper / Clickhousekeeper
2. ADQMDB
3. Chproxy
4. Monitoring Clients / Monitoring

Step 2. View the results of installation

ADCM starts a single job for installation process, as well as for any other task. You can find out about the status of jobs more specifically on the ***Jobs*** page.

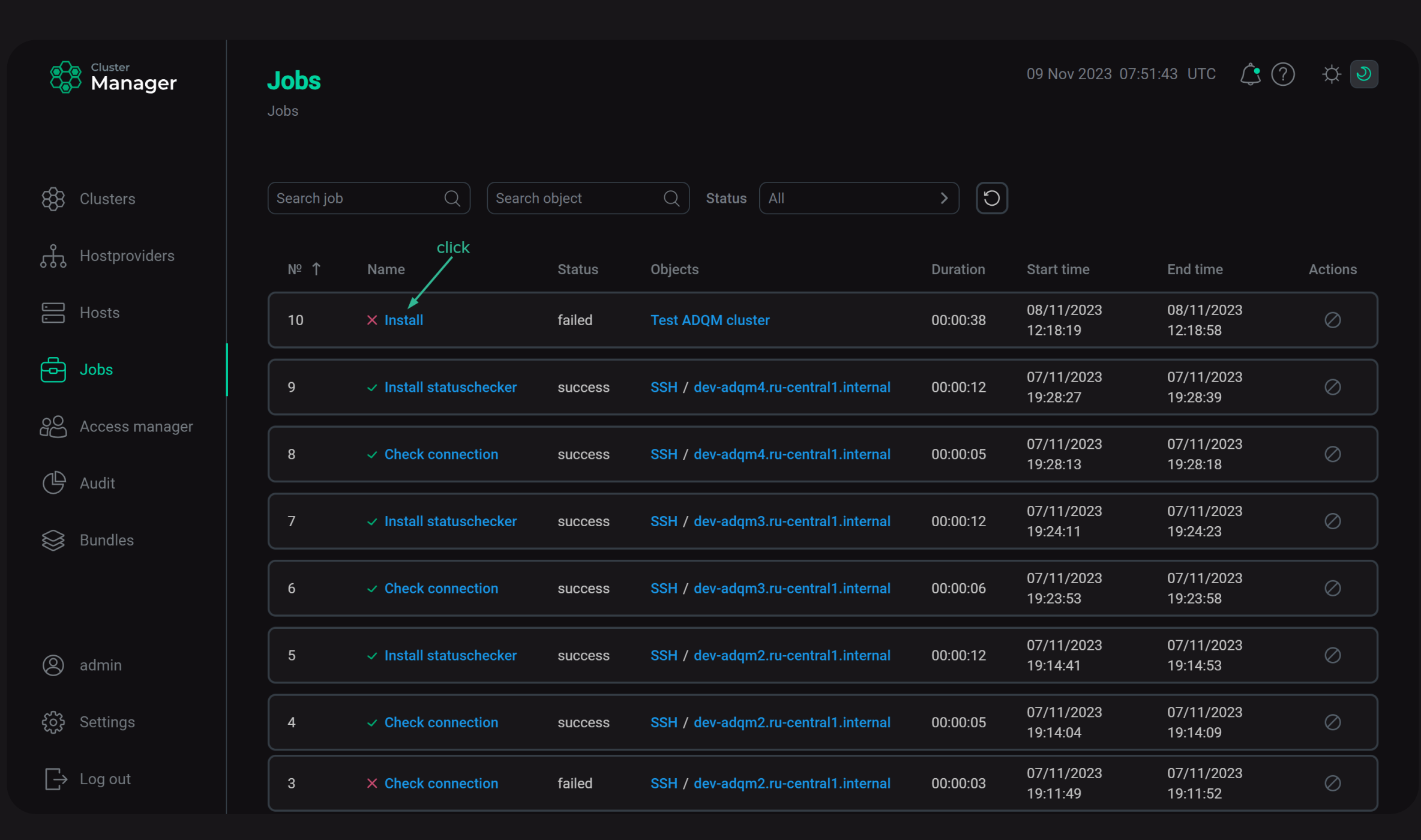


Cluster Manager
Jobs
Jobs
08 Nov 2023 12:24:05 UTC
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Search job
Search object
Status
All

№	Name	Status	Objects	Duration	Start time	End time	Actions
11	Install	running	Test ADQM cluster	00:00:00	08/11/2023 12:22:54	-	
10	Install	failed	Test ADQM cluster	00:00:38	08/11/2023 12:18:19	08/11/2023 12:18:58	
9	Install statuschecker	success	SSH / dev-adqm4.ru-central1.internal	00:00:12	07/11/2023 19:28:27	07/11/2023 19:28:39	
8	Check connection	success	SSH / dev-adqm4.ru-central1.internal	00:00:05	07/11/2023 19:28:13	07/11/2023 19:28:18	
7	Install statuschecker	success	SSH / dev-adqm3.ru-central1.internal	00:00:12	07/11/2023 19:24:11	07/11/2023 19:24:23	
6	Check connection	success	SSH / dev-adqm3.ru-central1.internal	00:00:06	07/11/2023 19:23:53	07/11/2023 19:23:58	
5	Install statuschecker	success	SSH / dev-adqm2.ru-central1.internal	00:00:12	07/11/2023 19:14:41	07/11/2023 19:14:53	

The Jobs page

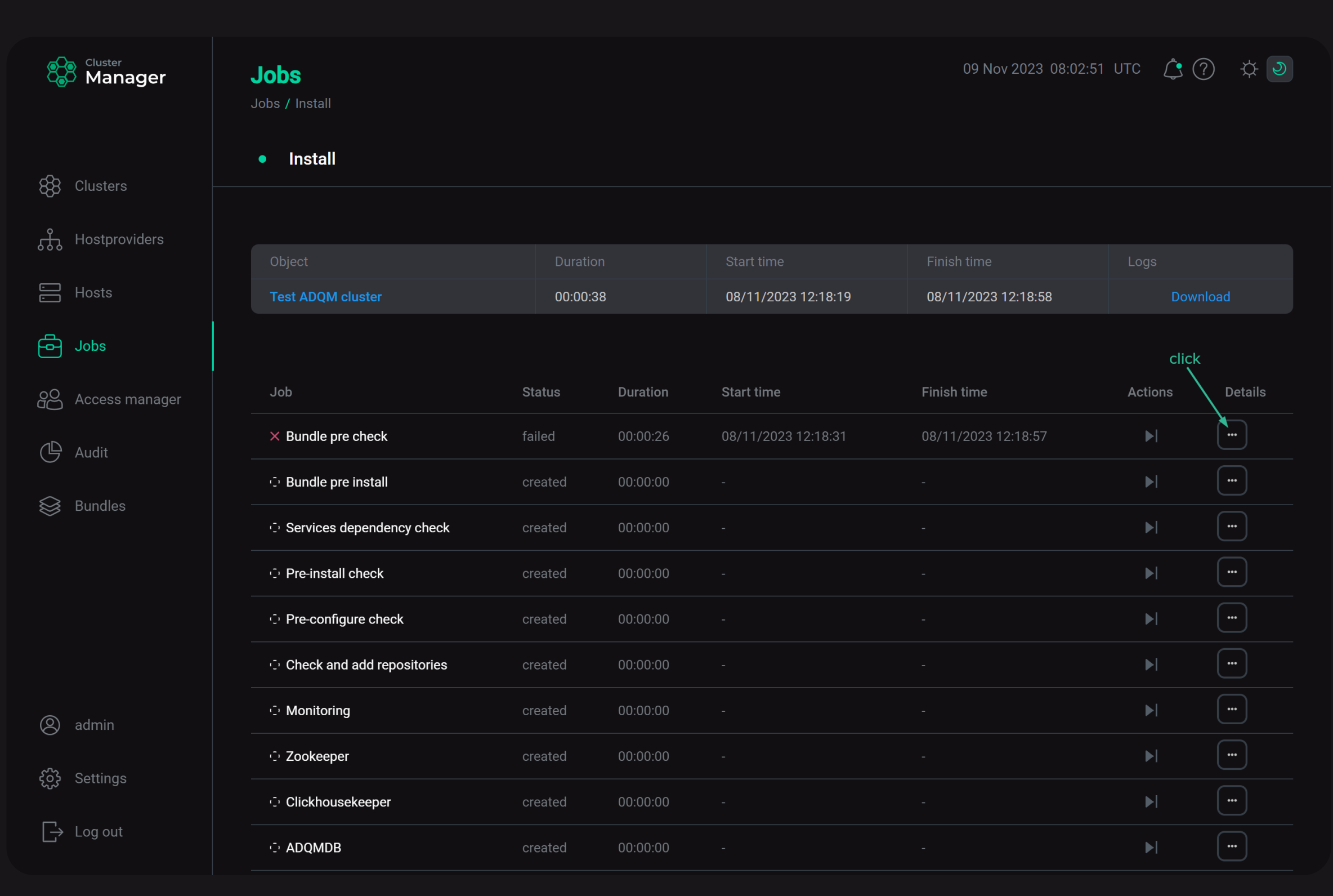
The successful completion of the service installation is determined by the transition of the job from the `running` status to the `success` status on the ***Jobs*** page. If the job fails, it switches to the `failed` status. In this case it is possible to see more detailed information about the errors that occurred by clicking the failed job on the ***Jobs*** page.



Cluster Manager
Jobs
Jobs
09 Nov 2023 07:51:43 UTC
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Search job
Search object
Status
All
click

№	Name	Status	Objects	Duration	Start time	End time	Actions
10	Install	failed	Test ADQM cluster	00:00:38	08/11/2023 12:18:19	08/11/2023 12:18:58	
9	Install statuschecker	success	SSH / dev-adqm4.ru-central1.internal	00:00:12	07/11/2023 19:28:27	07/11/2023 19:28:39	
8	Check connection	success	SSH / dev-adqm4.ru-central1.internal	00:00:05	07/11/2023 19:28:13	07/11/2023 19:28:18	
7	Install statuschecker	success	SSH / dev-adqm3.ru-central1.internal	00:00:12	07/11/2023 19:24:11	07/11/2023 19:24:23	
6	Check connection	success	SSH / dev-adqm3.ru-central1.internal	00:00:06	07/11/2023 19:23:53	07/11/2023 19:23:58	
5	Install statuschecker	success	SSH / dev-adqm2.ru-central1.internal	00:00:12	07/11/2023 19:14:41	07/11/2023 19:14:53	
4	Check connection	success	SSH / dev-adqm2.ru-central1.internal	00:00:05	07/11/2023 19:14:04	07/11/2023 19:14:09	
3	Check connection	failed	SSH / dev-adqm2.ru-central1.internal	00:00:03	07/11/2023 19:11:49	07/11/2023 19:11:52	

Switch to a job page



Cluster Manager
Jobs
Jobs / Install
09 Nov 2023 08:02:51 UTC
Install
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
click

Object	Duration	Start time	Finish time	Logs
Test ADQM cluster	00:00:38	08/11/2023 12:18:19	08/11/2023 12:18:58	Download

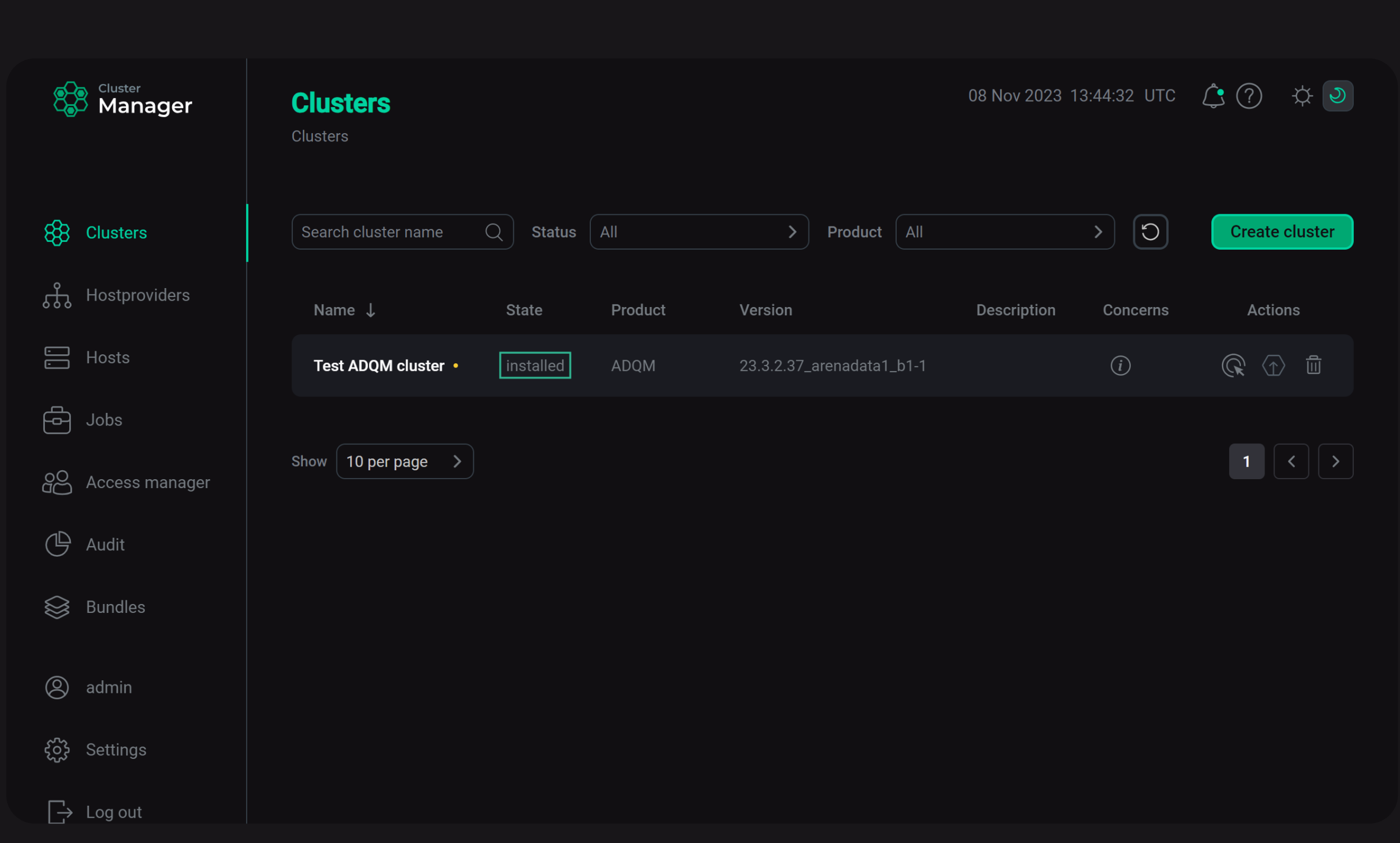
Job	Status	Duration	Start time	Finish time	Actions	Details
Bundle pre check	failed	00:00:26	08/11/2023 12:18:31	08/11/2023 12:18:57		
Bundle pre install	created	00:00:00	-	-		
Services dependency check	created	00:00:00	-	-		
Pre-install check	created	00:00:00	-	-		
Pre-configure check	created	00:00:00	-	-		
Check and add repositories	created	00:00:00	-	-		
Monitoring	created	00:00:00	-	-		
Zookeeper	created	00:00:00	-	-		
Clickhousekeeper	created	00:00:00	-	-		
ADQMDB	created	00:00:00	-	-		

Install cluster job page

Step 3. Check the cluster state after installation

As a result of successful installation, the cluster and its services change their state according to the following rules:

- The cluster state, displayed in the ***State*** field on the ***Clusters*** page, is being changed from `created` to `installed`.

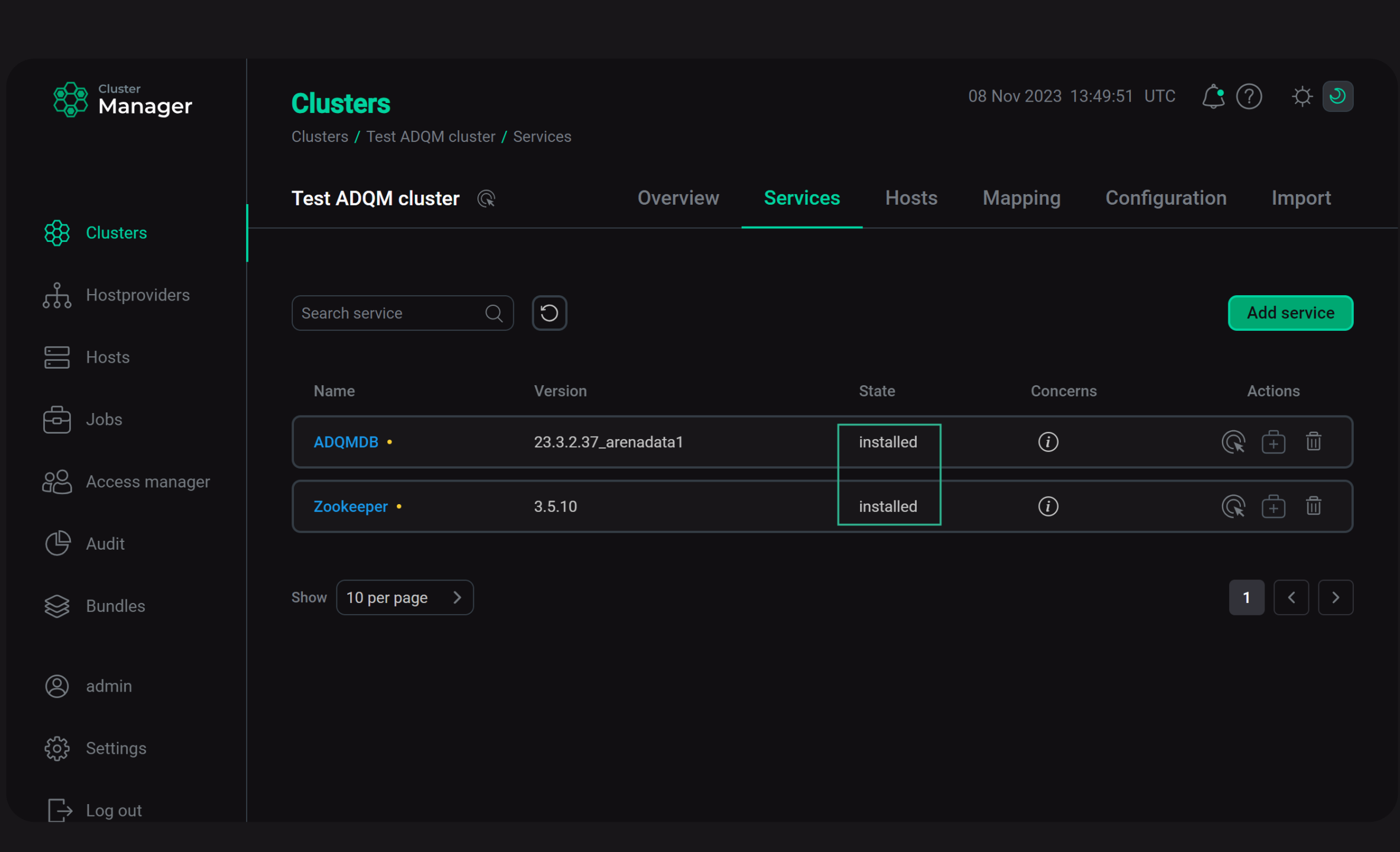


Cluster Manager
Clusters
Clusters
08 Nov 2023 13:44:32 UTC
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Search cluster name
Status
All
Product
All
Create cluster
Show
10 per page
1

Name	State	Product	Version	Description	Concerns	Actions
Test ADQM cluster	installed	ADQM	23.3.2.37_arenadata1_b1-1			

The cluster state after successful installation

- The state of cluster services, displayed on the ***Services*** tab of the cluster page, is being changed from `created` to `installed`.



Cluster Manager
Clusters
Clusters / Test ADQM cluster / Services
08 Nov 2023 13:49:51 UTC
Test ADQM cluster
Overview
Services
Hosts
Mapping
Configuration
Import
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Search service
Add service
Show
10 per page
1

Name	Version	State	Concerns	Actions
ADQMDB	23.3.2.37_arenadata1	installed		
Zookeeper	3.5.10	installed		

The state of cluster services after successful installation

Monitoring service

Elena Dvoryadkina

Contents

To Table of Contents

Overview
Step 1. Add the service
Step 2. Add components
Step 3. Configure the service
Step 4. Install the service
Step 5. View results
- Metrics in the Prometheus format
- Prometheus web interface
- Grafana web interface

The Monitoring service (available starting with ADQM 23.8.2.7) deploys the Prometheus server inside ADQM to collect and store ADQM cluster monitoring metrics, and also supports the ability to use the Grafana web application for visualization and analysis of information. This article describes steps required to add this service to an ADQM cluster.

Overview

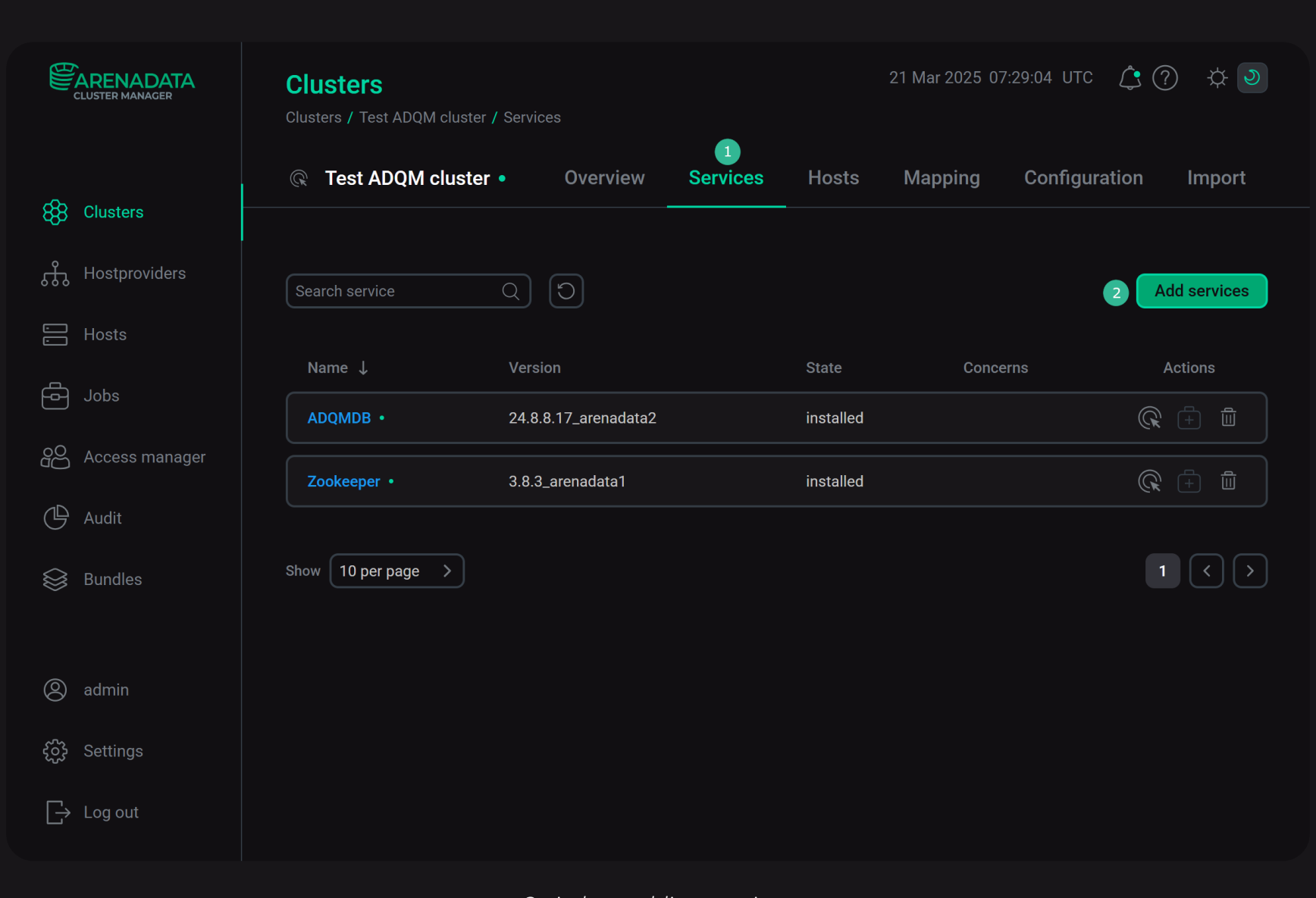
When adding the Monitoring service to ADQM, you can install the node exporter on all or selected hosts of your cluster. It is a monitoring agent that reads host system metrics that Prometheus collects. Prometheus can also collect metrics from ADQM services (ClickHouse, ZooKeeper, ClickHouse Keeper, Chproxy). These metrics will be available in the Prometheus format on the ports and endpoints specified in the ***Metrics*** section of the service configurations. You can also use the Prometheus and Grafana web interfaces to view and analyze data that the monitoring service collects.

NOTE

- If you already have a Prometheus-compatible monitoring system set up (for example, your own Prometheus server or VictoriaMetrics), you can use it to collect ADQM metrics. To do this, configure access to ADQM metrics in your monitoring system using ADQM's Prometheus parameters specified on the Monitoring service page.
- You can also use the Federation mechanism to migrate all metrics from the Prometheus server deployed in ADQM to your Prometheus.

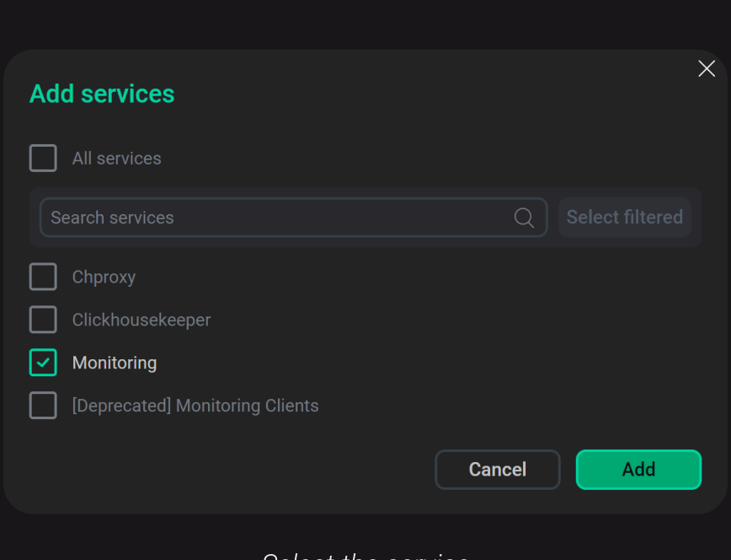
Step 1. Add the service

1. In the ADCM interface, open the ***Clusters*** page and click your ADQM cluster name. On the cluster page that opens, switch to the ***Services*** tab and click ***Add services***.


Clusters
Clusters / Test ADQM cluster / Services
21 Mar 2025 07:29:04 UTC
Test ADQM cluster
Overview
1
Services
Hosts
Mapping
Configuration
Import
2
Add services
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out

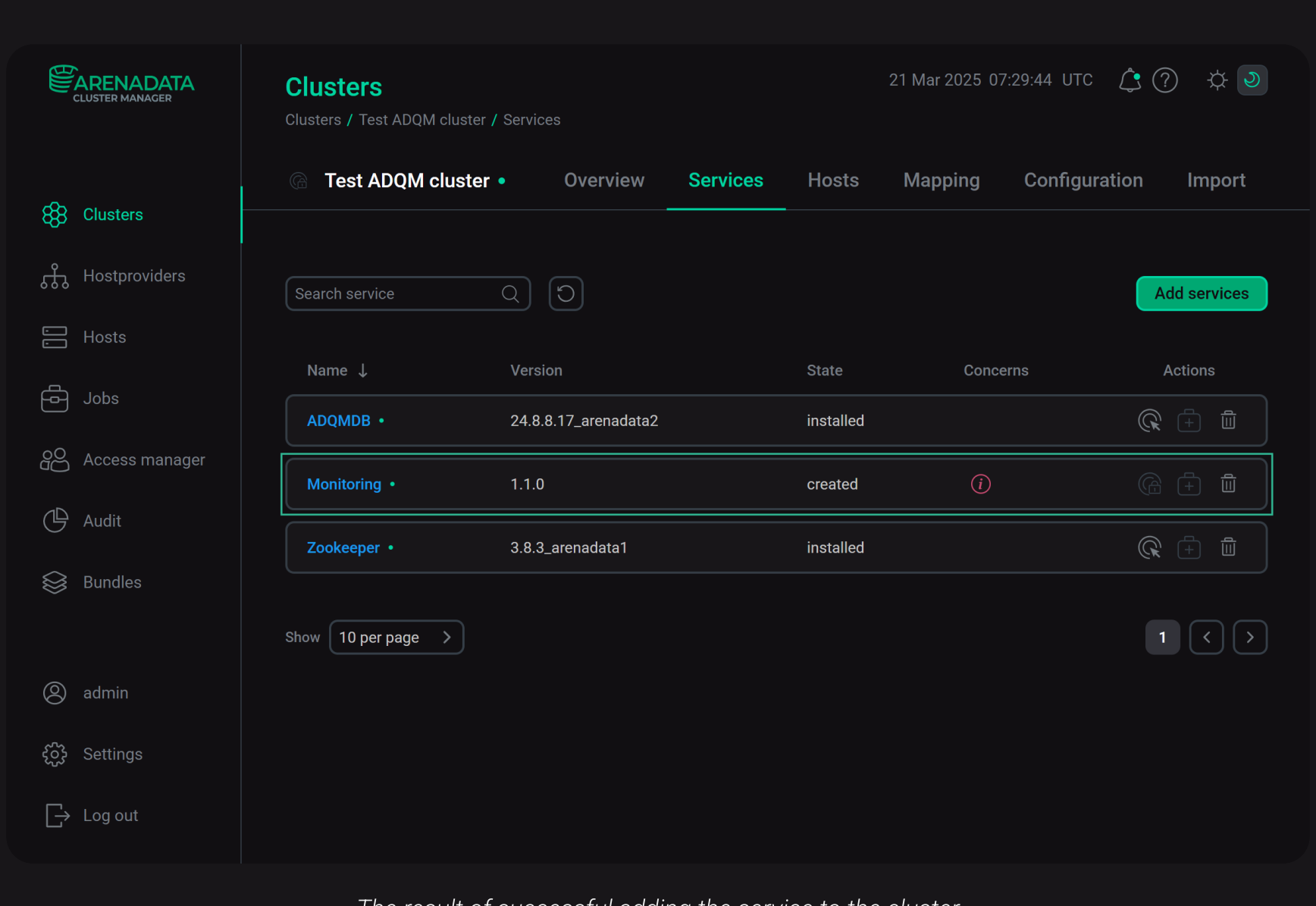
Switch to adding services

2. In the opened dialog, select the ***Monitoring*** service and click ***Add***.


Add services
All services
Search services
Select filtered
Chproxy
Clickhousekeeper
Monitoring
[Deprecated] Monitoring Clients
Cancel
Add

Select the service

As a result, the added service is displayed on the ***Services*** tab.

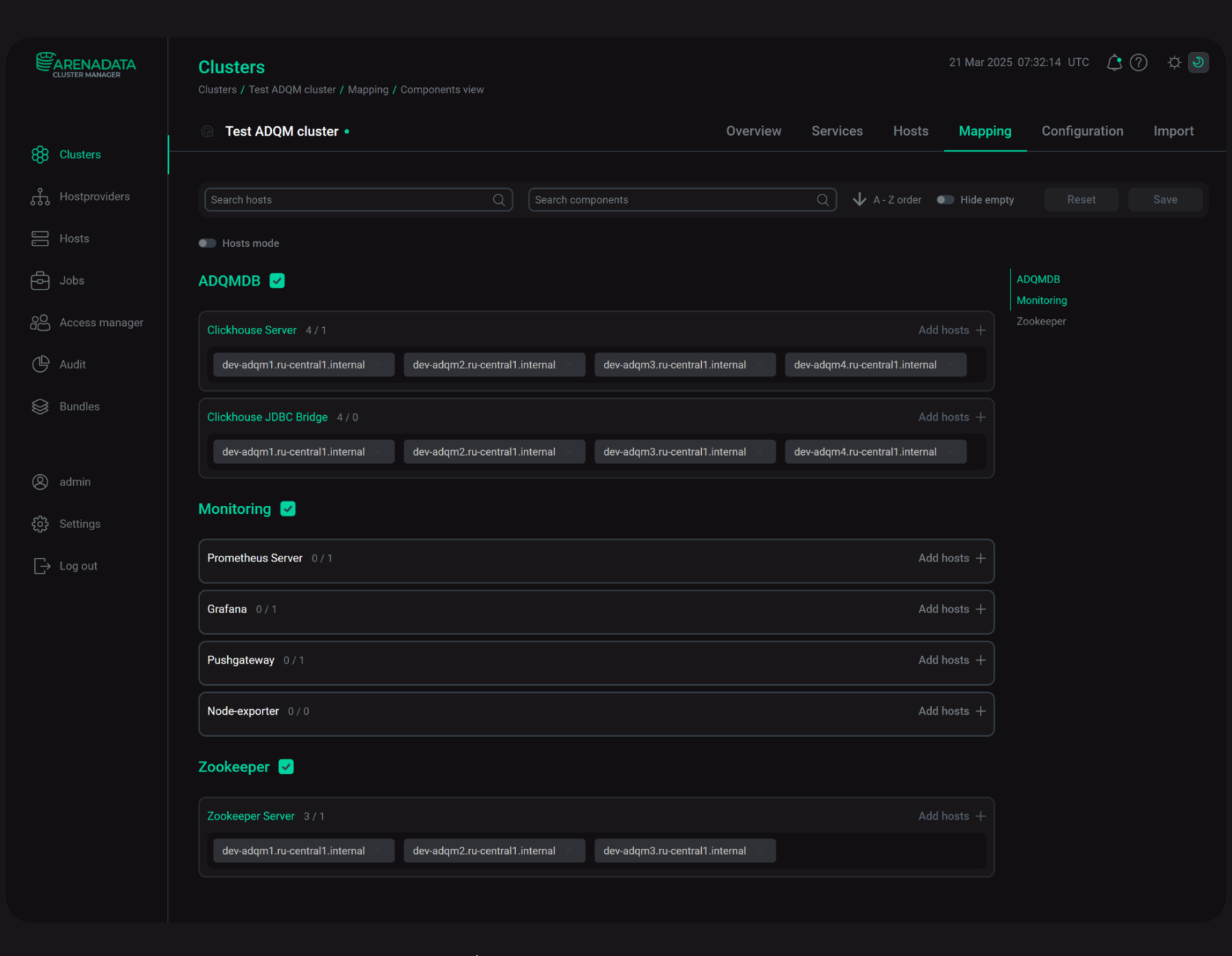

Clusters
Clusters / Test ADQM cluster / Services
Test ADQM cluster
Services
Add services

Name	Version	State
ADQMDB	24.8.8.17_arenadata2	installed
Monitoring	1.1.0	created
Zookeeper	3.8.3_arenadata1	installed

The result of successful adding the service to the cluster

Step 2. Add components

1. On the cluster page, open the ***Mapping*** tab to proceed to mapping service components to cluster hosts.

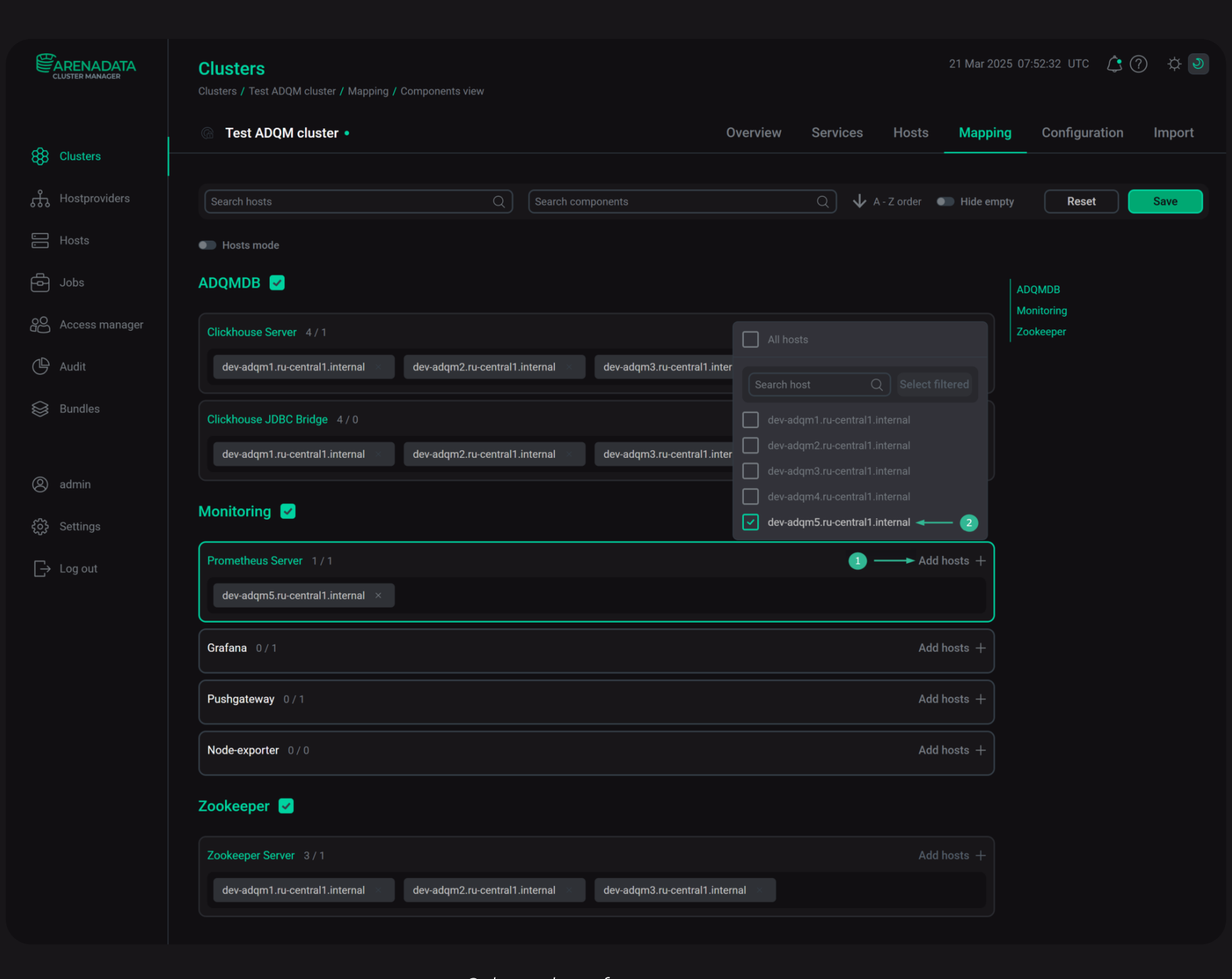

Clusters
Test ADQM cluster
Mapping
ADQMDB
Clickhouse Server 4 / 1
Clickhouse JDBC Bridge 4 / 0
Monitoring
Prometheus Server 0 / 1
Grafana 0 / 1
Pushgateway 0 / 1
Node-exporter 0 / 0
Zookeeper
Zookeeper Server 3 / 1
Add hosts

Switch to mapping service components

Monitoring service components

Component	Description
Prometheus Server	Stores and allows requesting all metrics for an ADQM cluster
Grafana	Visualizes ADQM metrics as graphs and charts organized into dashboards
Pushgateway	Receives static metrics and sends them to Prometheus. In ADQM, it is used to pass a cluster structure to Prometheus
Node Exporter	Collects system metrics and sends them to Prometheus

2. Assign hosts to components of the Monitoring service — click ***Add hosts*** and select the desired hosts in the pop-up window.

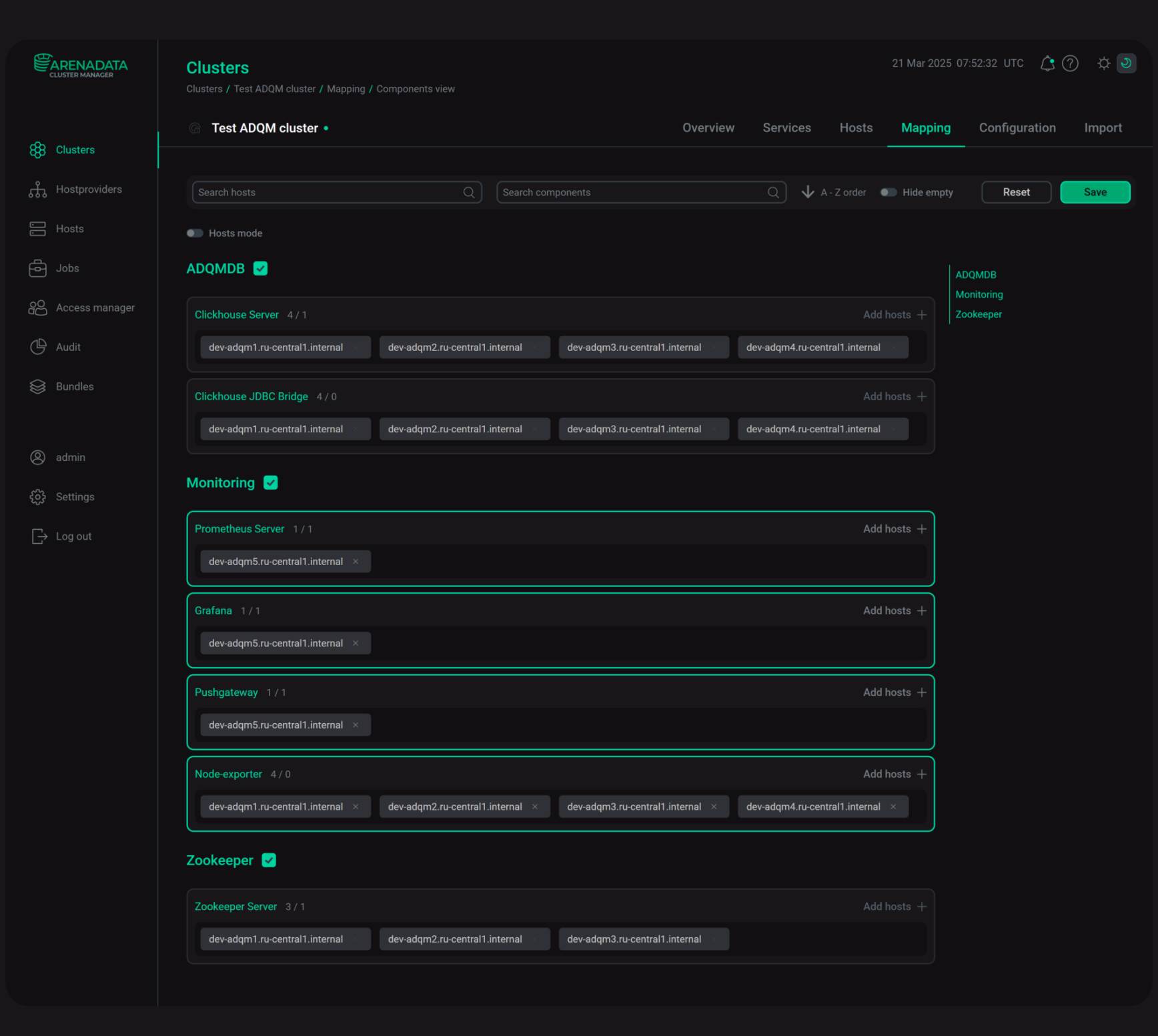

All hosts
Search host
Select filtered
dev-adqm1.ru-central1.internal
dev-adqm2.ru-central1.internal
dev-adqm3.ru-central1.internal
dev-adqm4.ru-central1.internal
dev-adqm5.ru-central1.internal
1
2
Prometheus Server 1 / 1
Add hosts

Select a host for a component

CAUTION

It is not recommended to install the Prometheus Server and Pushgateway components on hosts with ADQM — use separate hosts for them. Otherwise, if an ADQM host fails and/or the load on it is critically high, information about the corresponding problems will not be saved.

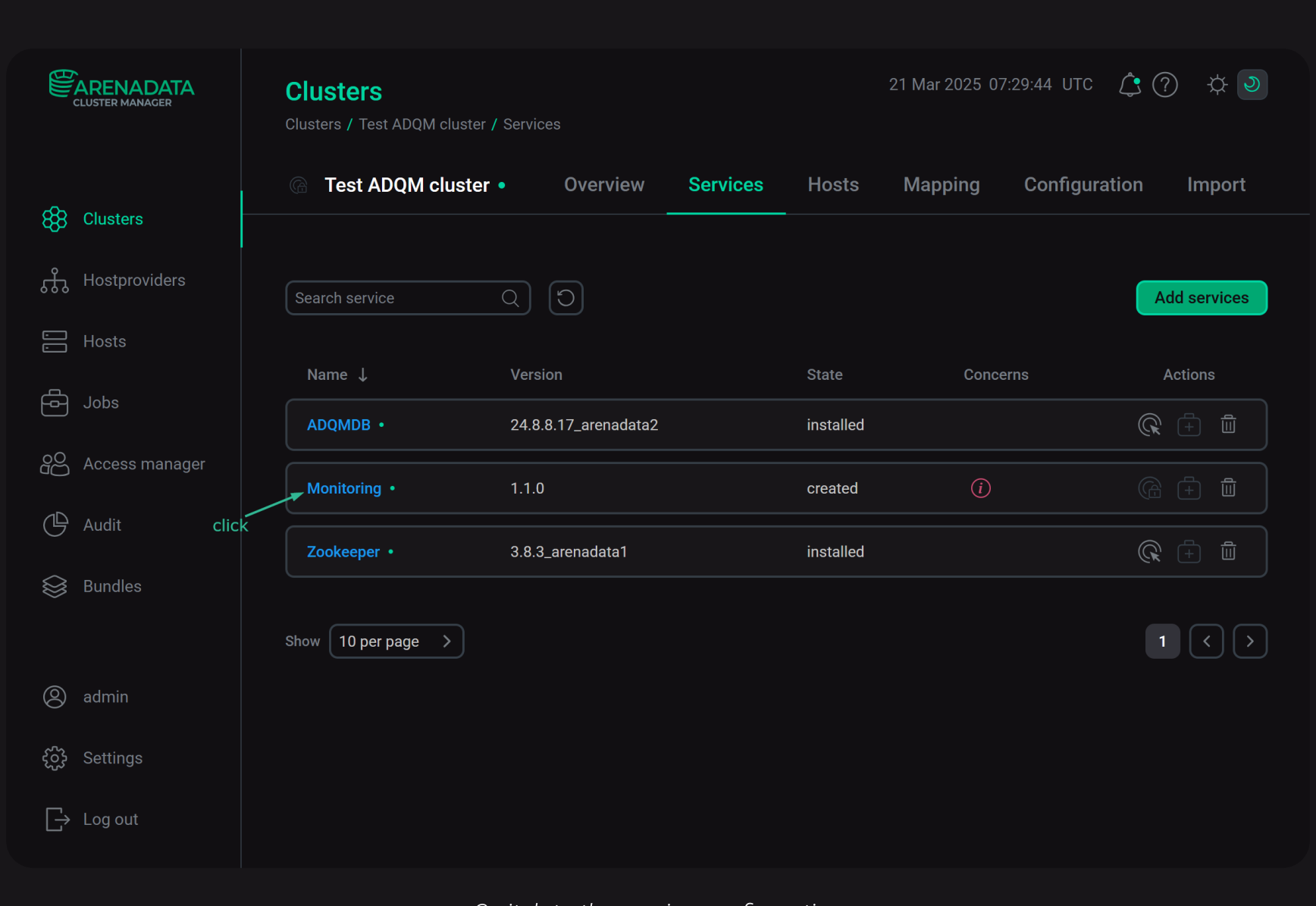
3. After the distribution of components is completed, click ***Save***.


Reset
Save
Prometheus Server 1 / 1
Grafana 1 / 1
Pushgateway 1 / 1
Node-exporter 4 / 0

Save mapping of components

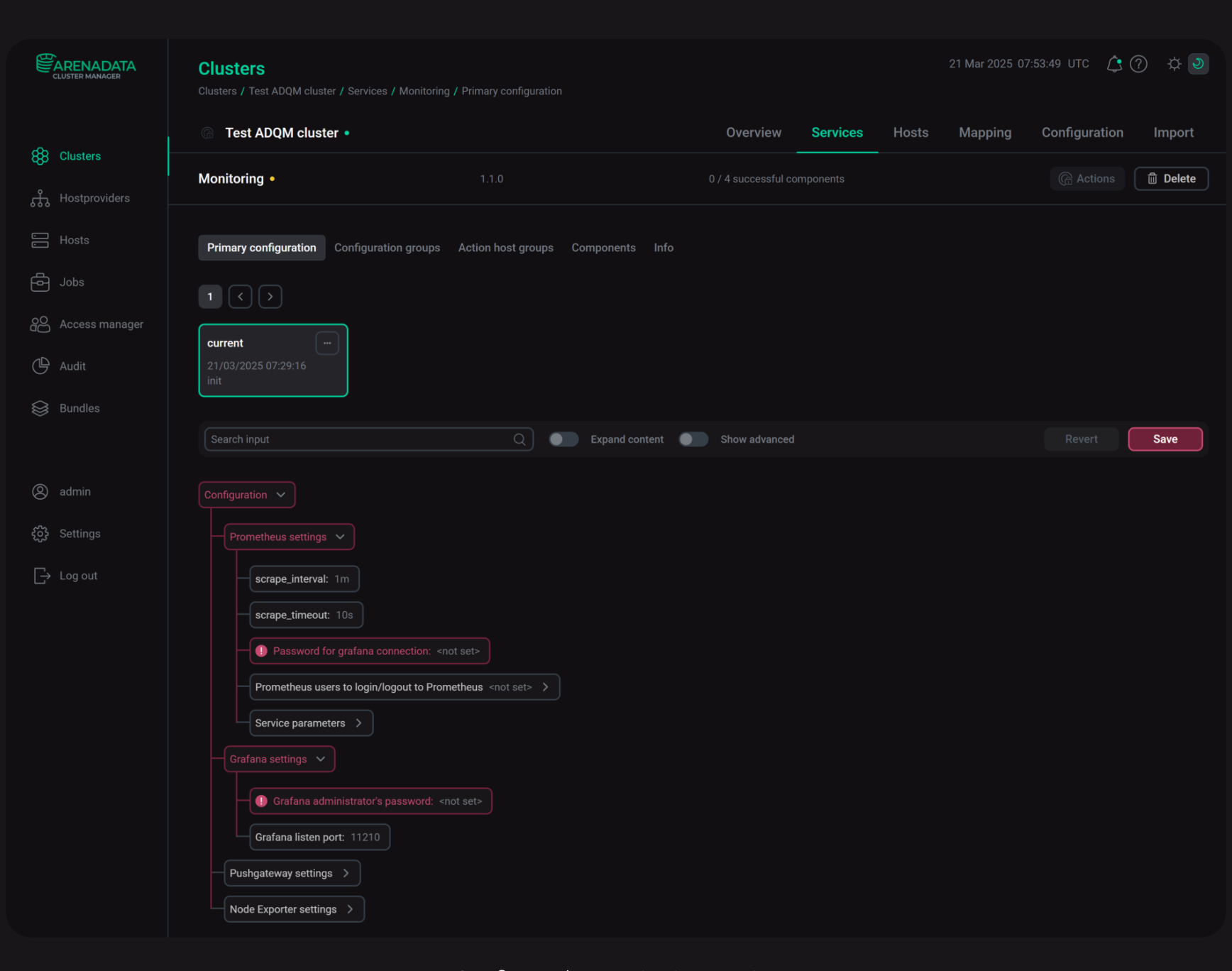
Step 3. Configure the service

1. Open the ***Services*** tab on the cluster page and click the ***Monitoring*** service name in the ***Name*** column.


click
Monitoring

Switch to the service configuration

2. On the page that opens, fill in the service's configuration parameters — see the ***Monitoring*** section in the Configuration parameters article for parameter descriptions. Fields highlighted in red are required.


Primary configuration
Configuration groups
Action host groups
Components
Info
current
Save
Configuration
Prometheus settings
scrape_interval: 1m
scrape_timeout: 10s
Password for grafana connection: <not set>
Prometheus users to login/logout to Prometheus <not set>
Service parameters
Grafana settings
Grafana administrator's password: <not set>
Grafana listen port: 11210
Pushgateway settings
Node Exporter settings

Configure the Monitoring service

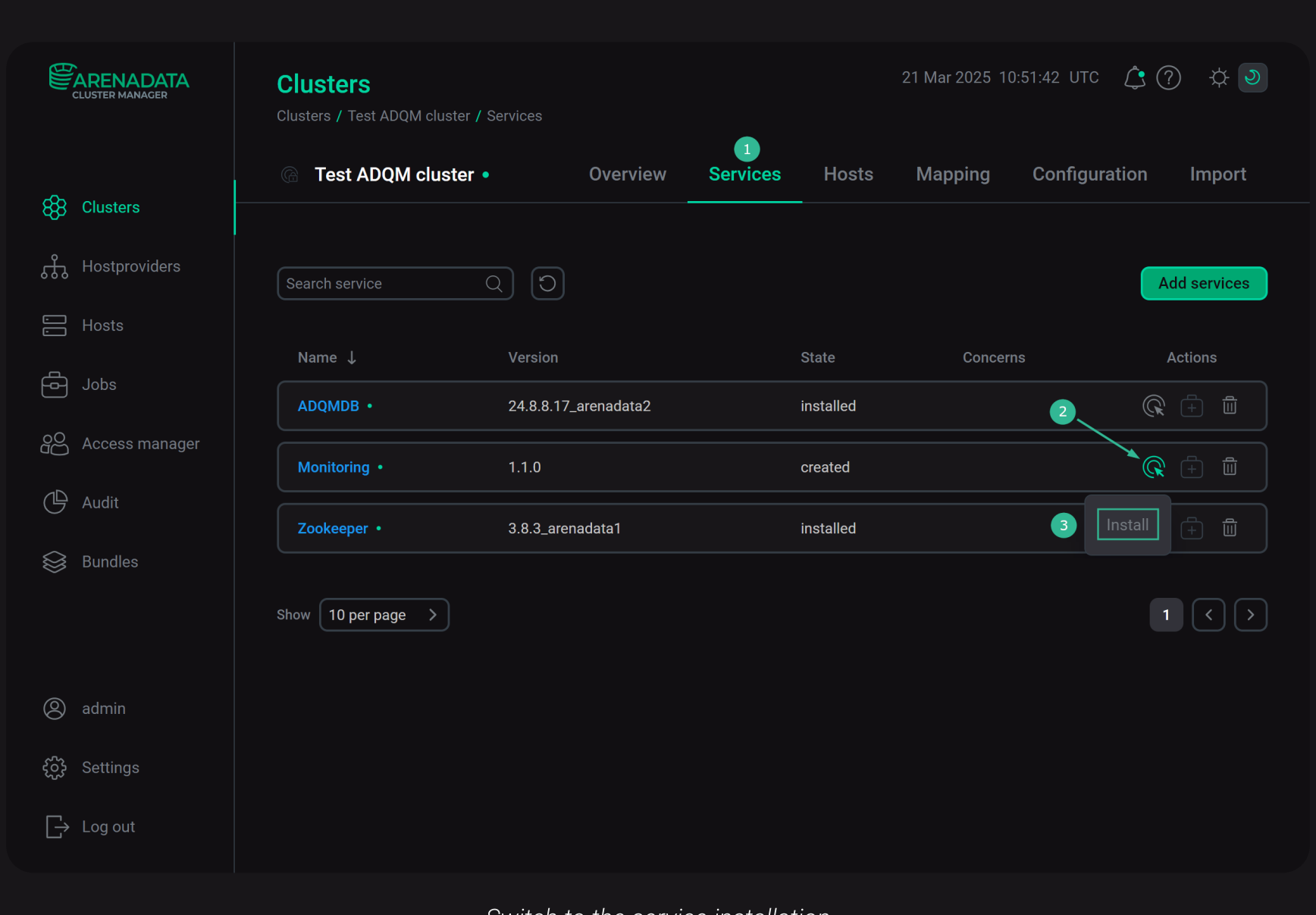
After specifying all necessary parameters, click ***Save***.

NOTE

To configure settings for access to metrics of the ADQMDB, Zookeeper, Clickhousekeeper services, use parameters in the ***Metrics*** activatable section of each service configuration. If you do not specify these parameters manually, the Monitoring service will automatically enable the ***Metrics*** section for ADQM services during installation and will use the default values.

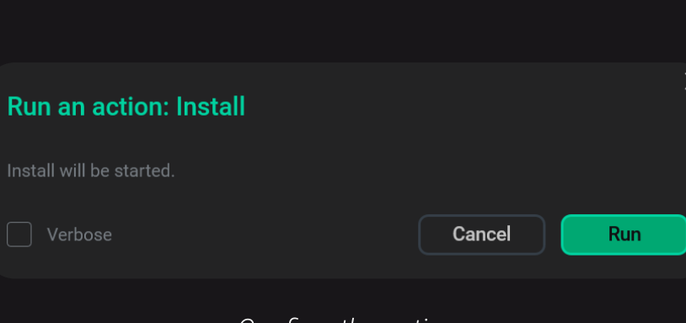
Step 4. Install the service

1. On the ***Services*** tab, click the icon for the Monitoring service in the ***Actions*** column and run the ***Install*** action.


1
Services
2
3
Install

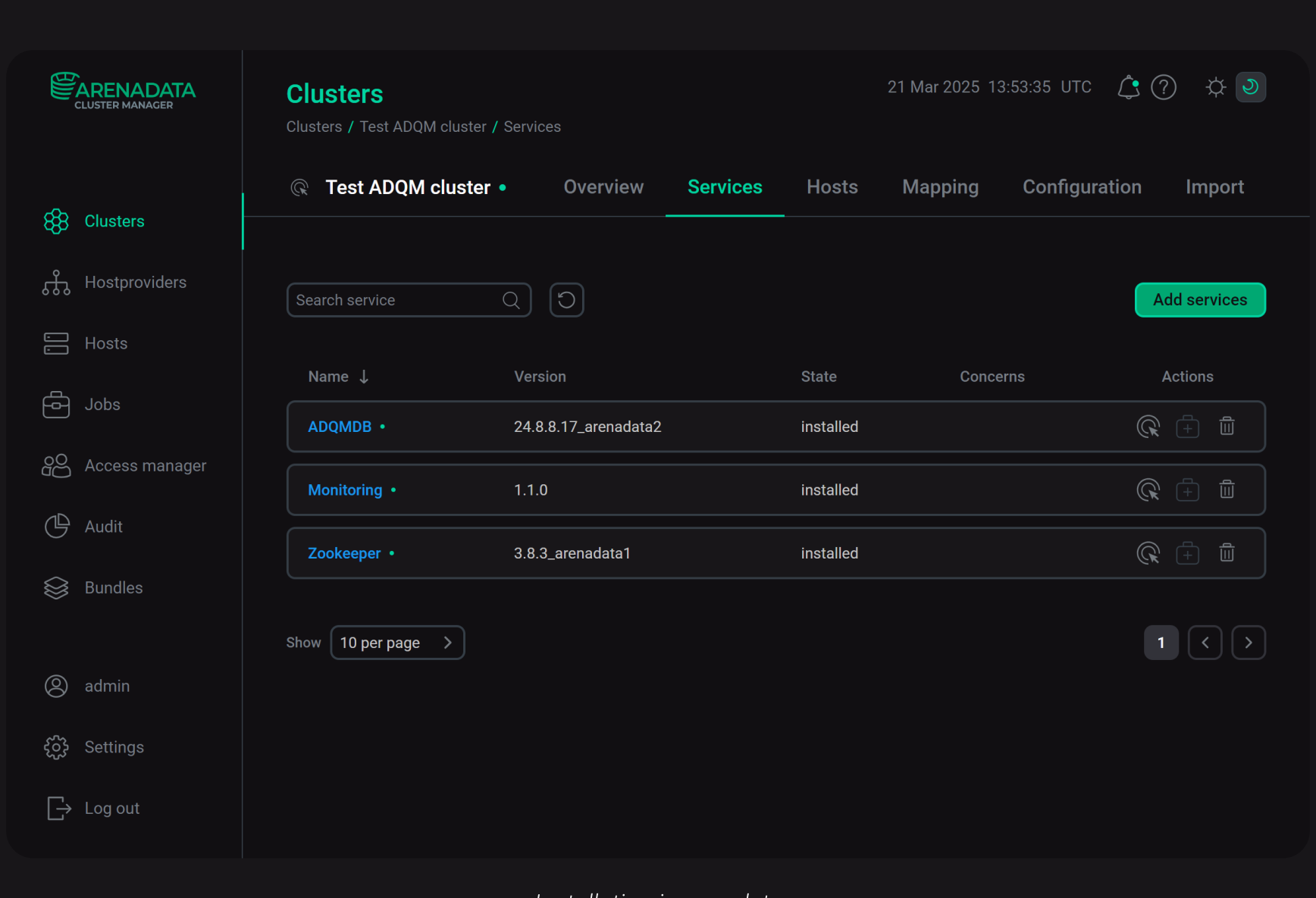
Switch to the service installation

2. Confirm the action in the opened window.


Run an action: Install
Install will be started.
Verbose
Cancel
Run

Confirm the action

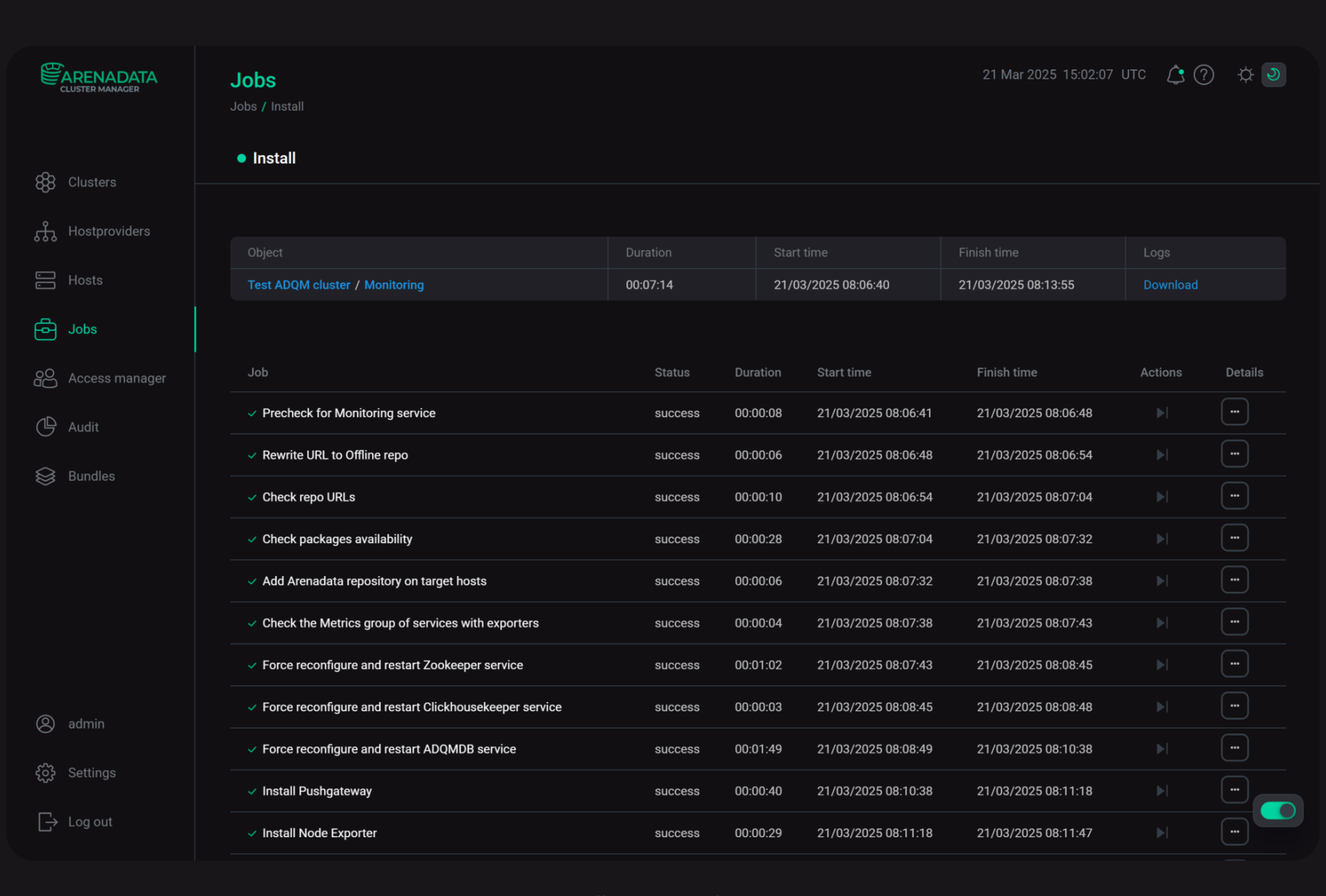
3. Wait until the installation is completed. Then check that the service state has changed from `created` to `installed`.



Name	Version	State
ADQMDB	24.8.8.17_arenadata2	installed
Monitoring	1.1.0	installed
Zookeeper	3.8.3_arenadata1	installed

Installation is complete

To view the service installation process and analyze errors if they occur, select ***Jobs*** in the left navigation menu and click the ***Install*** job name in the ***Jobs*** list.


Jobs
Jobs / Install
Install

Object	Duration	Start time	Finish time	Logs
Test ADQM cluster / Monitoring	00:07:14	21/03/2025 08:06:40	21/03/2025 08:13:55	Download

Job	Status	Duration	Start time	Finish time
Precheck for Monitoring service	success	00:00:08	21/03/2025 08:06:41	21/03/2025 08:06:48
Rewrite URL to Offline repo	success	00:00:06	21/03/2025 08:06:48	21/03/2025 08:06:54
Check repo URLs	success	00:00:10	21/03/2025 08:06:54	21/03/2025 08:07:04
Check packages availability	success	00:00:28	21/03/2025 08:07:04	21/03/2025 08:07:32
Add Arenadata repository on target hosts	success	00:00:06	21/03/2025 08:07:32	21/03/2025 08:07:38
Check the Metrics group of services with exporters	success	00:00:04	21/03/2025 08:07:38	21/03/2025 08:07:43
Force reconfigure and restart Zookeeper service	success	00:01:02	21/03/2025 08:07:43	21/03/2025 08:08:45
Force reconfigure and restart Clickhousekeeper service	success	00:00:03	21/03/2025 08:08:45	21/03/2025 08:08:48
Force reconfigure and restart ADQMDB service	success	00:01:49	21/03/2025 08:08:49	21/03/2025 08:10:38
Install Pushgateway	success	00:00:40	21/03/2025 08:10:38	21/03/2025 08:11:18
Install Node Exporter	success	00:00:29	21/03/2025 08:11:18	21/03/2025 08:11:47

Install service job page

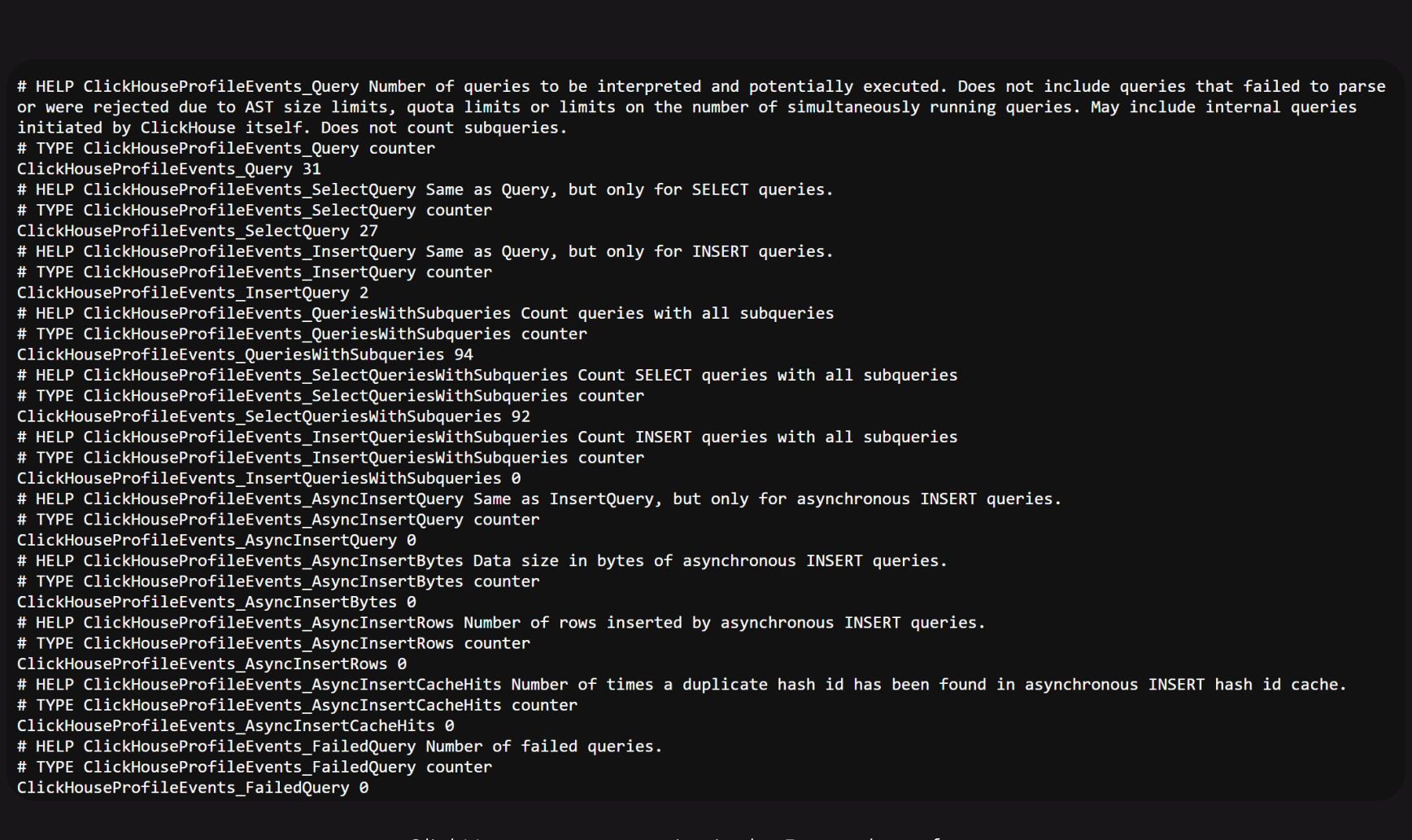
Step 5. View results

The Monitoring service starts automatically after installation. To ensure that monitoring works correctly, check that both system and ADQM service metrics are collected from the cluster hosts — see ADQM monitoring metrics. To do this, view metrics in the Prometheus format in the browser or use the Prometheus or Grafana web interface.

Metrics in the Prometheus format

1. In the address bar of your browser, enter an address of an ADQM host with the specified port and endpoint to listen for service or system metrics. Port numbers and endpoints are defined by the parameters located in:
   - the ***Metrics*** section on the configuration pages of the ADQMDB, Zookeeper, and Clickhousekeeper services — settings for access to monitoring metrics of these services;
   - the ***Node Exporter settings*** section on the configuration page of the Monitoring service — settings for access to system metrics of the ADQM cluster hosts.

   For example, *http://10.92.40.25:9363/metrics* is an address to view metrics of the ClickHouse server on a host with IP `10.92.40.25`.
2. The page that opens will display monitoring metrics from the specified host of the ADQM cluster in the Prometheus format.

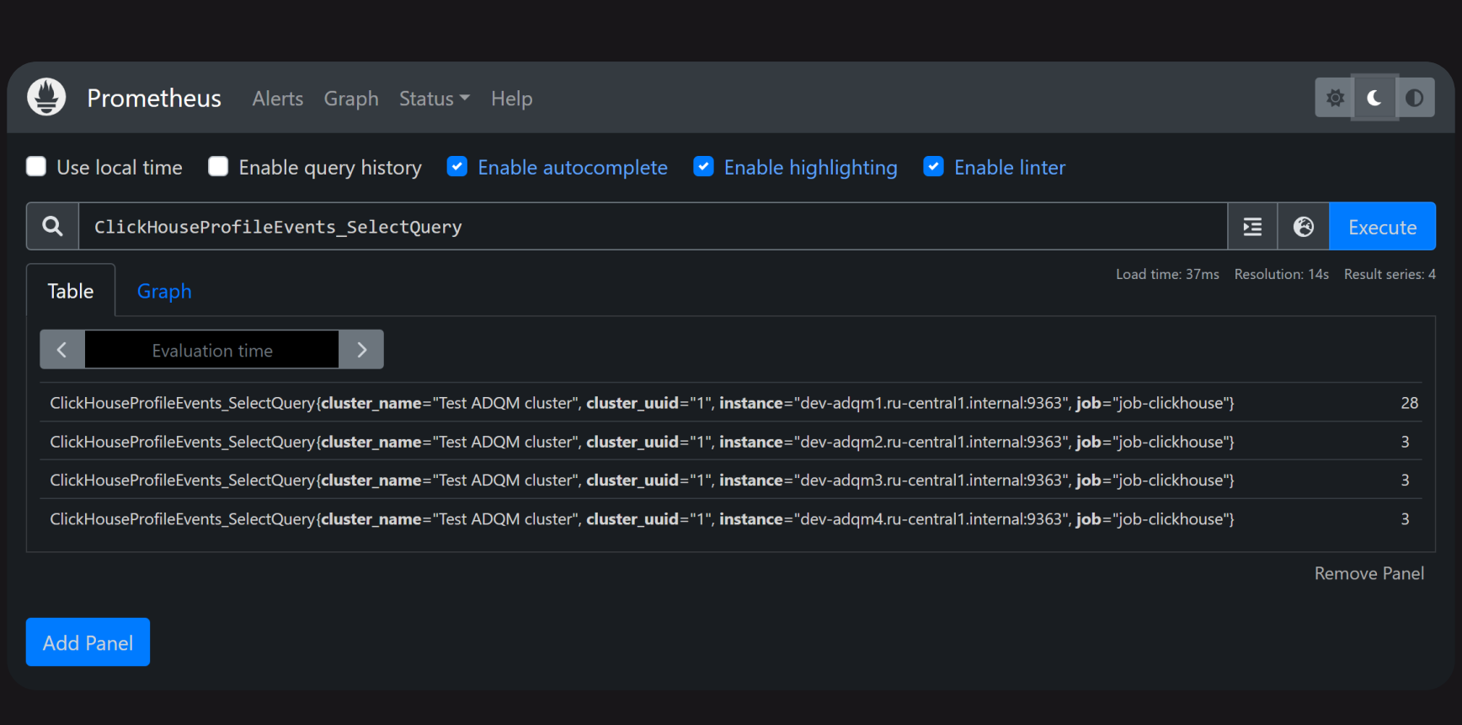


```
# HELP ClickHouseProfileEvents_Query Number of queries to be interpreted and potentially executed. Does not include queries that failed to parse
or were rejected due to AST size limits, quota limits or limits on the number of simultaneously running queries. May include internal queries
initiated by ClickHouse itself. Does not count subqueries.
# TYPE ClickHouseProfileEvents_Query counter
ClickHouseProfileEvents_Query 31
# HELP ClickHouseProfileEvents_SelectQuery Same as Query, but only for SELECT queries.
# TYPE ClickHouseProfileEvents_SelectQuery counter
ClickHouseProfileEvents_SelectQuery 27
# HELP ClickHouseProfileEvents_InsertQuery Same as Query, but only for INSERT queries.
# TYPE ClickHouseProfileEvents_InsertQuery counter
ClickHouseProfileEvents_InsertQuery 2
# HELP ClickHouseProfileEvents_QueriesWithSubqueries Count queries with all subqueries
# TYPE ClickHouseProfileEvents_QueriesWithSubqueries counter
ClickHouseProfileEvents_QueriesWithSubqueries 94
# HELP ClickHouseProfileEvents_SelectQueriesWithSubqueries Count SELECT queries with all subqueries
# TYPE ClickHouseProfileEvents_SelectQueriesWithSubqueries counter
ClickHouseProfileEvents_SelectQueriesWithSubqueries 92
# HELP ClickHouseProfileEvents_InsertQueriesWithSubqueries Count INSERT queries with all subqueries
# TYPE ClickHouseProfileEvents_InsertQueriesWithSubqueries counter
ClickHouseProfileEvents_InsertQueriesWithSubqueries 0
# HELP ClickHouseProfileEvents_AsyncInsertQuery Same as InsertQuery, but only for asynchronous INSERT queries.
# TYPE ClickHouseProfileEvents_AsyncInsertQuery counter
ClickHouseProfileEvents_AsyncInsertQuery 0
# HELP ClickHouseProfileEvents_AsyncInsertBytes Data size in bytes of asynchronous INSERT queries.
# TYPE ClickHouseProfileEvents_AsyncInsertBytes counter
ClickHouseProfileEvents_AsyncInsertBytes 0
# HELP ClickHouseProfileEvents_AsyncInsertRows Number of rows inserted by asynchronous INSERT queries.
# TYPE ClickHouseProfileEvents_AsyncInsertRows counter
ClickHouseProfileEvents_AsyncInsertRows 0
# HELP ClickHouseProfileEvents_AsyncInsertCacheHits Number of times a duplicate hash id has been found in asynchronous INSERT hash id cache.
# TYPE ClickHouseProfileEvents_AsyncInsertCacheHits counter
ClickHouseProfileEvents_AsyncInsertCacheHits 0
# HELP ClickHouseProfileEvents_FailedQuery Number of failed queries.
# TYPE ClickHouseProfileEvents_FailedQuery counter
ClickHouseProfileEvents_FailedQuery 0
```

ClickHouse server metrics in the Prometheus format

Prometheus web interface

1. In the address bar of your browser, enter an IP address of a host where the Prometheus Server component is installed. Specify also a port number from the ***web.listen-address*** parameter in the ***Service parameters*** group within the ***Prometheus settings*** section of the Monitoring service configuration (the default port is `11200`). For example: *http://10.92.40.107:11200*. To log into the Prometheus interface, use a username and password that are also specified in the ***Prometheus settings*** section — the ***Prometheus users to login/logout to Prometheus*** setting.
2. In the ***Expression*** field, you can enter a metric and click ***Execute*** — values of this metric on all hosts of the ADQM cluster will be shown in the interface.


Prometheus
Alerts
Graph
Status
Help
Use local time
Enable query history
Enable autocomplete
Enable highlighting
Enable linter
Execute
Load time: 37ms Resolution: 14s Result series: 4
Table
Graph
Evaluation time
Remove Panel
Add Panel

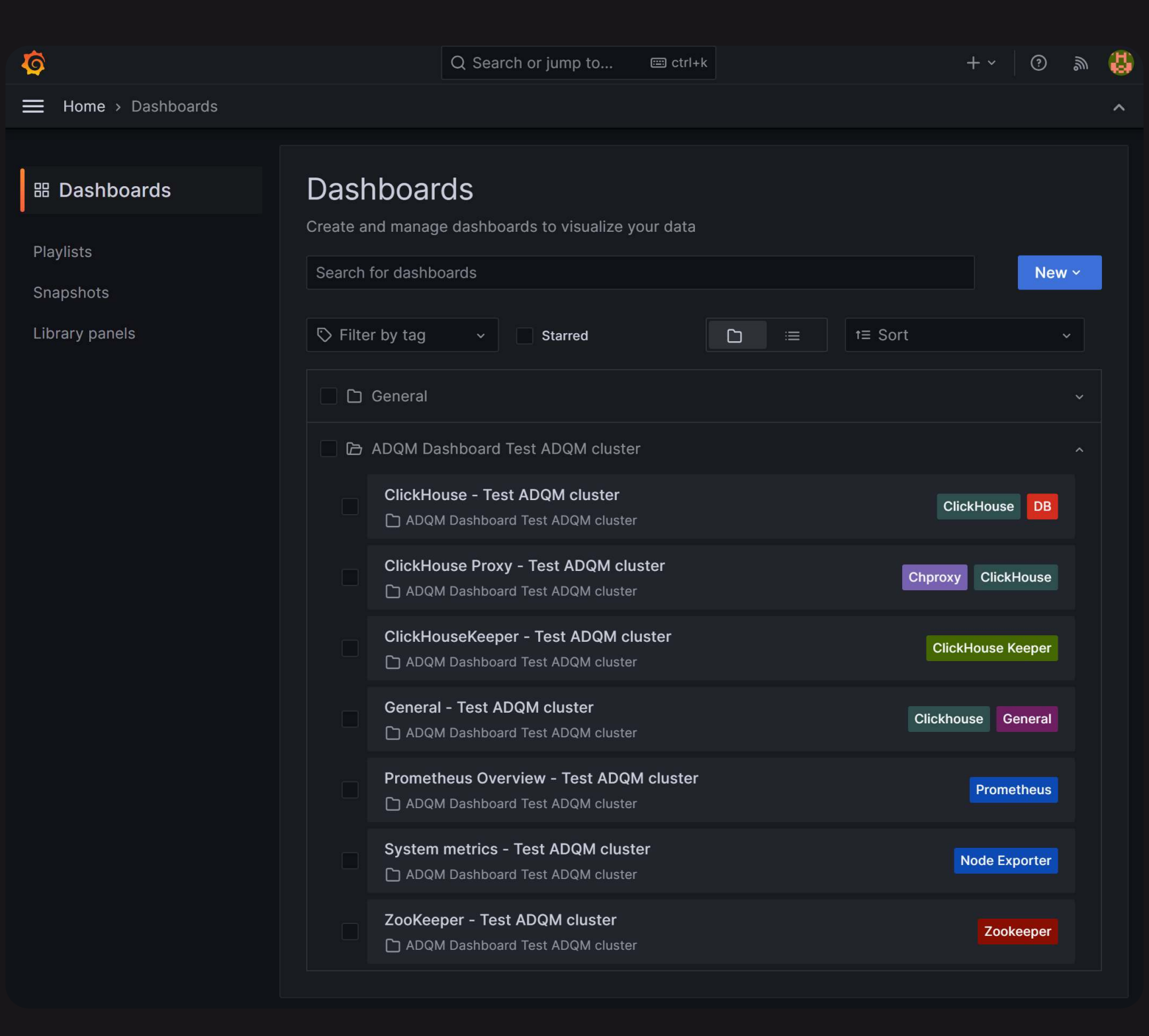
```
ClickHouseProfileEvents_SelectQuery
```

```
ClickHouseProfileEvents_SelectQuery{cluster_name="Test ADQM cluster", cluster_uuid="1", instance="dev-adqm1.ru-central1.internal:9363", job="job-clickhouse"}  28
ClickHouseProfileEvents_SelectQuery{cluster_name="Test ADQM cluster", cluster_uuid="1", instance="dev-adqm2.ru-central1.internal:9363", job="job-clickhouse"}  3
ClickHouseProfileEvents_SelectQuery{cluster_name="Test ADQM cluster", cluster_uuid="1", instance="dev-adqm3.ru-central1.internal:9363", job="job-clickhouse"}  3
ClickHouseProfileEvents_SelectQuery{cluster_name="Test ADQM cluster", cluster_uuid="1", instance="dev-adqm4.ru-central1.internal:9363", job="job-clickhouse"}  3
```

Prometheus web interface

Grafana web interface

1. In the address bar of your browser, enter an address of a host on which Grafana is deployed and add a port number — a value of the ***Grafana listen port*** parameter located in the ***Grafana settings*** section of the Monitoring service configuration (the default value is `11210`). For example, *http://10.92.40.107:11210*. To log in, use `admin` as a username, and the ***Grafana administrator's password*** parameter value (also found in the ***Grafana settings*** section of service configuration parameters) as a password.
2. In the window that opens, navigate to ***Home*** → ***Dashboards*** and expand the ***ADQM Dashboard <ADQM_cluster_name>*** section. In this section, you can select one of available dashboards to view service or system metrics coming from your ADQM cluster.


Search or jump to...
ctrl+k
Home › Dashboards
Dashboards
Playlists
Snapshots
Library panels
Dashboards
Create and manage dashboards to visualize your data
Search for dashboards
New
Filter by tag
Starred
Sort
General
ADQM Dashboard Test ADQM cluster
ClickHouse - Test ADQM cluster
ADQM Dashboard Test ADQM cluster
ClickHouse
DB
ClickHouse Proxy - Test ADQM cluster
ADQM Dashboard Test ADQM cluster
Chproxy
ClickHouse
ClickHouseKeeper - Test ADQM cluster
ADQM Dashboard Test ADQM cluster
ClickHouse Keeper
General - Test ADQM cluster
ADQM Dashboard Test ADQM cluster
Clickhouse
General
Prometheus Overview - Test ADQM cluster
ADQM Dashboard Test ADQM cluster
Prometheus
System metrics - Test ADQM cluster
ADQM Dashboard Test ADQM cluster
Node Exporter
ZooKeeper - Test ADQM cluster
ADQM Dashboard Test ADQM cluster
Zookeeper

View ADQM metrics in Grafana

Connect to ADQM

Elena Dvoryadkina

Contents

To Table of Contents

Overview

Configure user access to ADQM servers

Overview

You can use different tools (depending on your tasks) to connect to ADQM, including:

- Built-in ClickHouse web UI
- Native ClickHouse command-line client — clickhouse-client
- JDBC driver
- Third-party client tools (for example, DBeaver)

NOTE

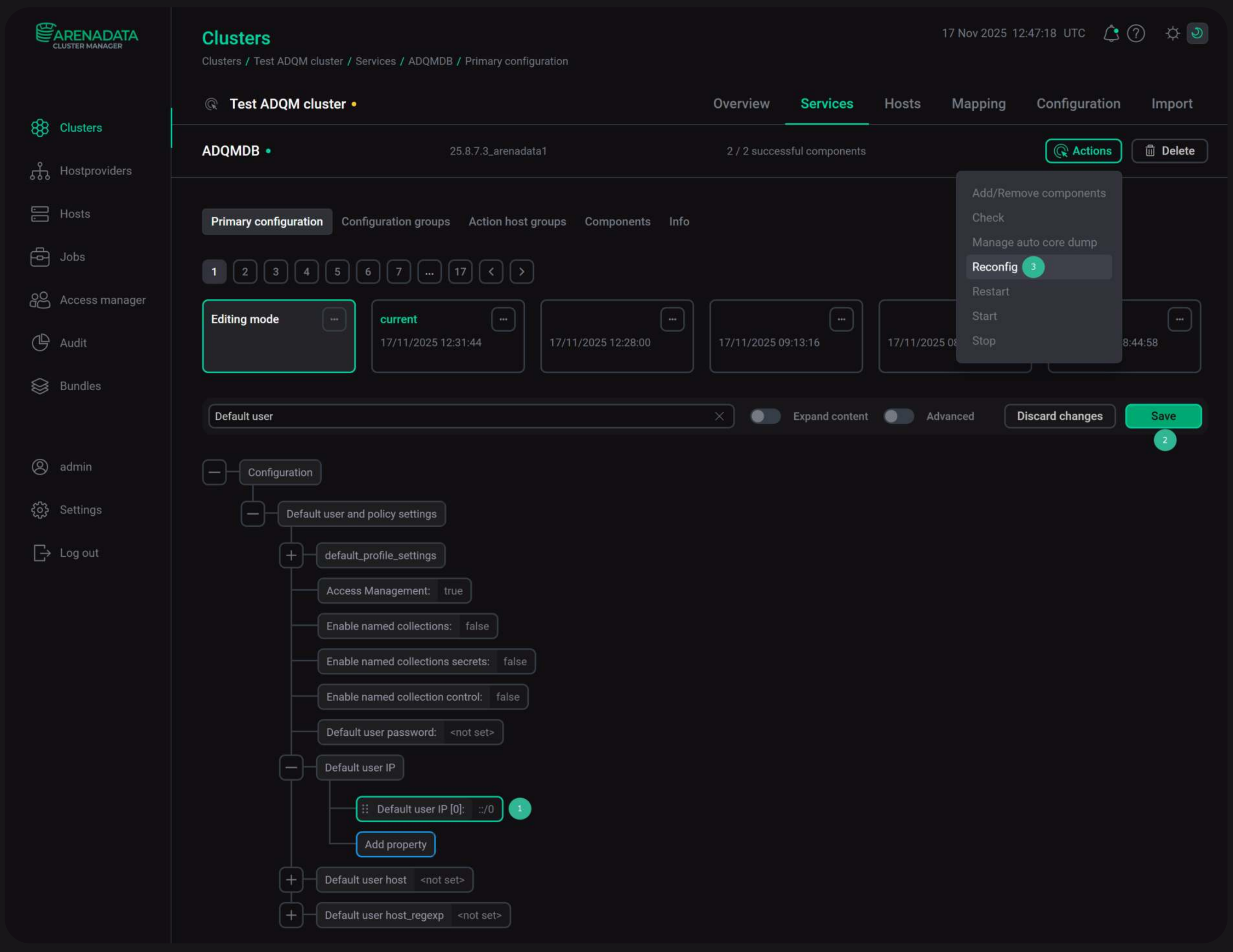
After ADQM installation, you can use ClickHouse web interface and `clickhouse-client` out of the box. If you need to utilize third-party tools, install and configure them according to your requirements.

Configure user access to ADQM servers

To allow access to ADQM via ClickHouse web interface, JDBC driver or any third-party client tool (including, DBeaver), specify networks from which connections to ADQM are allowed in the ADQM user configuration.

For example, to allow the `default` user to access an ADQM server from any IP address, add the `::/0` item (or `0.0.0.0/0` if IPv6 is not supported) to the list of networks from which the user can connect to ADQM. You can do this in one of the following ways:

- Add the `::/0` value for the ***Default user IP*** parameter in the ADCM web interface (***Clusters*** → ***ADQM cluster*** → ***Services*** → ***ADQMDB*** → ***Primary Configuration***) and save changes.


ARENADATA
CLUSTER MANAGER
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Clusters
Clusters / Test ADQM cluster / Services / ADQMDB / Primary configuration
17 Nov 2025 12:47:18 UTC
Test ADQM cluster
Overview
Services
Hosts
Mapping
Configuration
Import
ADQMDB
25.8.7.3_arenadata1
2 / 2 successful components
Actions
Delete
Add/Remove components
Check
Manage auto core dump
Reconfig
3
Restart
Start
Stop
Primary configuration
Configuration groups
Action host groups
Components
Info
1 2 3 4 5 6 7 ... 17
Editing mode
current
17/11/2025 12:31:44
17/11/2025 12:28:00
17/11/2025 09:13:16
17/11/2025 0
8:44:58
Default user
Expand content
Advanced
Discard changes
Save
2
Configuration
Default user and policy settings
default_profile_settings
Default user IP
1
Add property

Access Management: true
Enable named collections: false
Enable named collections secrets: false
Enable named collection control: false
Default user password: <not set>
Default user IP [0]: ::/0
Default user host <not set>
Default user host_regexp <not set>

Specify IP addresses with access to ADQM servers

- Add the `<ip>::/0</ip>` item to the `networks` section of a user configuration in the *users.xml* file:

```
<users>
    <default>
        <password></password>
        <access_management>1</access_management>
        <networks incl="networks" replace="replace">
            <ip>::/0</ip>
            ...
        </networks>
        <profile>default</profile>
        <quota>default</quota>
    </default>
</users>
```

IMPORTANT

Regardless of the chosen method — run ***Reconfig*** for the ADQMDB service in the ADCM web interface to apply changes.

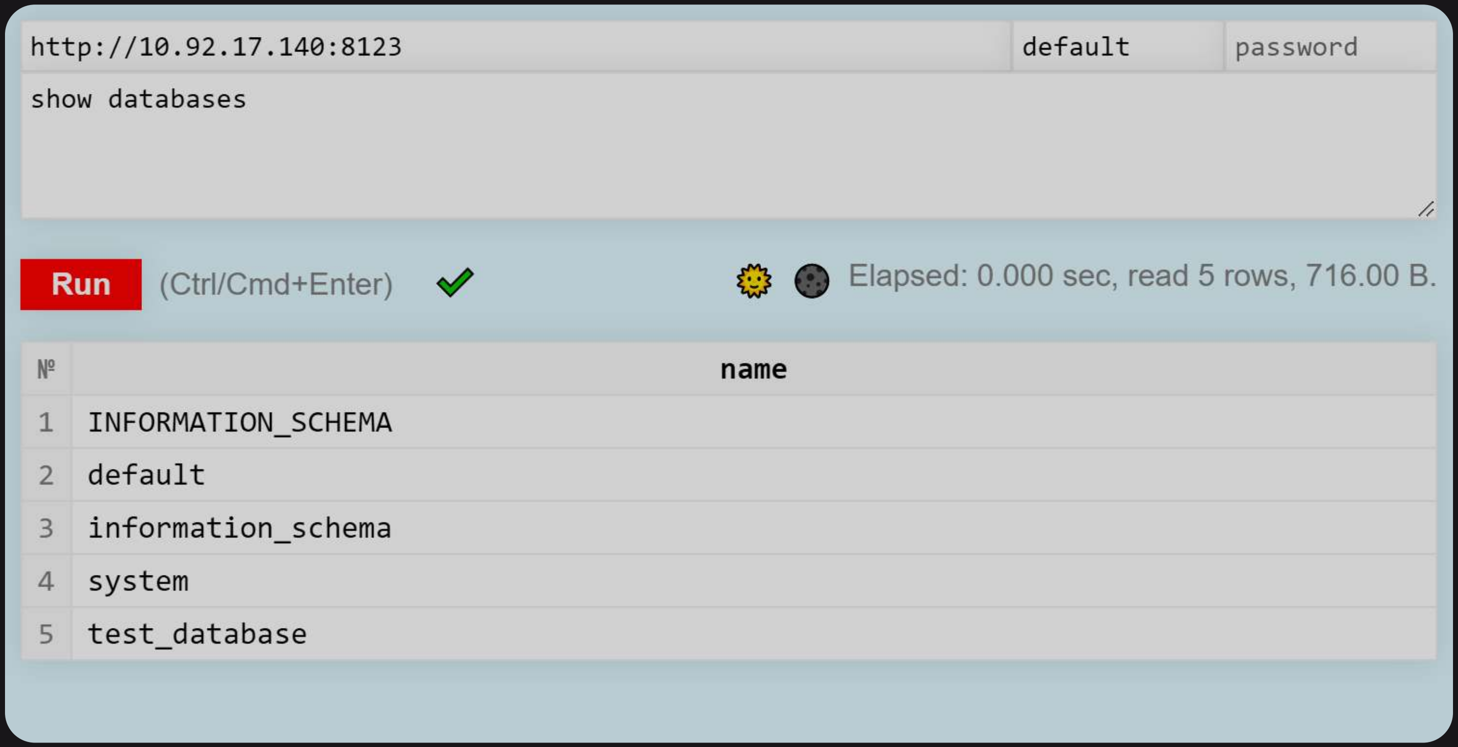
Use built-in ClickHouse web UI

Elena Dvoryadkina

To Table of Contents

To work with ADQM databases, you can use the built-in ClickHouse web interface available at *http://<IP>:8123/play*, where `<IP>` is an IP address of an ADQM server. In the ***user*** and ***password*** input fields located at the top right of the web interface, enter a username and password for connecting to the ADQM server (a user should be defined in the `users` section of the server's *users.xml* configuration file). By default, the `default` username and empty password are used.

Enter your queries in the main text box of the interface and click ***Run*** to execute them.



```
http://10.92.17.140:8123        default        password
show databases
```

Elapsed: 0.000 sec, read 5 rows, 716.00 B.

№	name
1	INFORMATION_SCHEMA
2	default
3	information_schema
4	system
5	test_database

Built-in ClickHouse web interface

NOTE

For ADQM users, it should be specified from which networks connections are allowed. The Connect to ADQM article contains an example on how to configure the `default` user so that you can access an ADQM server as this user via the ClickHouse web interface from any IP address.

Use the native ClickHouse command-line client — clickhouse-client

Elena Dvoryadkina

Contents

To Table of Contents

Interactive mode

Batch mode

Query output format

To connect to ADQM, you can use `clickhouse-client` — a standard ClickHouse command-line client that allows you to run SQL queries and view their results in your terminal application. Once ADQM is installed, `clickhouse-client` is available on each server of the ADQM cluster.

Run the following command to start working with `clickhouse-client`:

```
$ clickhouse-client
```

This command connects to the local ADQM server (`localhost:9000`) as the `default` user without a password.

```
$ clickhouse-client
ClickHouse client version 22.8.4.7.
Connecting to localhost:9000 as user default.
Connected to ClickHouse server version 22.8.4 revision 54460.

:)
```

To connect to a remote ADQM server, or to specify a username, password, and database other than the defaults, you can use the following parameters of the `clickhouse-client` command:

- `--host, -h` — name of a host where an ADQM server is located;
- `--port` — port to connect to;
- `--user, -u` — username;
- `--password` — password;
- `--ask-password` — prompt for a password;
- `--database, -d` — current database. Default value is the default database from ADQM configuration parameters (***ADQMDB*** *→* ***Other*** *→* ***Default database***).

You can use `clickhouse-client` in interactive and batch (non-interactive) mode.

Create tables to run the examples below

To reproduce the examples below, first prepare the following tables:

- `test_table1`

```
CREATE TABLE test_table1 (a Int32, b String, c Int32) ENGINE = MergeTree ORDER BY a;
```

```
INSERT INTO test_table1 VALUES (1, 'aaa', 100), (2, 'bbb', 200), (3, 'ccc', 300), (4, 'ddd', 400), (5, 'eee', 500);
```

- `test_table2`

```
CREATE TABLE test_table2 (a Int32, b String, c Int32) ENGINE = MergeTree ORDER BY a;
```

```
INSERT INTO test_table2 VALUES (1, 'a', 0), (2, 'b', 0), (3, 'c', 0);
```

Interactive mode

After the `clickhouse-client` command is executed, the terminal connects to the ADQM server and switches to the interactive mode — you can enter a query after the `:)` prompt. To run a query, press `Enter` (a semicolon is not necessary at the end of a query).

The `clickhouse-client --multiline` command activates the multiline query input mode. In this case, to run a query, end it with a semicolon and press `Enter`. If there is no semicolon at the end of the entered string, pressing `Enter` prompts you to enter the next line of the query.

Below is an example of entering and executing a query in the interactive mode of the command-line client.

Query:

```
SELECT * FROM test_table1;
```

The output should be similar to:

```
SELECT *
FROM test_table1

Query id: 917d68af-6989-409e-a30f-89a657487e22

--a---b-------c--
│ 1 │ aaa │ 100 │
│ 2 │ bbb │ 200 │
│ 3 │ ccc │ 300 │
│ 4 │ ddd │ 400 │
│ 5 │ eee │ 500 │
-----------------

5 rows in set. Elapsed: 0.001 sec.
```

When processing a query, `clickhouse-client` displays:

- Progress (it might not have time to be displayed for quick queries).
- Formatted query after parsing.
- Query ID.
- Result in the specified format. In the interactive mode, the client uses the `PrettyCompact` format by default.
- Number of lines in the result and execution time.

To exit the interactive mode, press `Ctrl+D` or type `exit`, `quit`, `q`, or `logout` instead of a query.

Batch mode

You can also use `clickhouse-client` in the batch (non-interactive) mode as follows:

- Pass a query to the `--query` parameter of the `clickhouse-client` command:

```
$ clickhouse-client --query "SELECT * FROM test_table2"
```

The output can look like:

```
1       a       0
2       b       0
3       c       0
```

- Send a query to stdin:

```
$ echo "SELECT * FROM test_table2" | clickhouse-client
```

The output can look like:

```
1       a       0
2       b       0
3       c       0
```

- Use both the `--query` parameter and stdin. For example, for an `INSERT` query, you can pass the beginning of the query to the `query` parameter and send data to be inserted to stdin:

```
$ echo "(4,'d',4)" | clickhouse-client --query "INSERT INTO test_table2 VALUES"
$ clickhouse-client --query "SELECT * FROM test_table2"
```

The output can look like:

```
1       a       0
2       b       0
3       c       0
4       d       4
```

By default, a single query only can be processed in the batch mode. To allow executing multiple queries, use the `--multiquery` parameter (it works for all queries except `INSERT`). In this case, the `--query` parameter can accept a string that includes multiple queries separated by semicolons. For example:

```
$ clickhouse-client --multiquery --query "select * from test_table1;select * from test_table2;"
```

In the batch mode, the default format for displaying query results is TabSeparated. When running multiple queries with the `--multiquery` option enabled, query results are output one after another without additional separators. The following section describes how to change the output format.

Query output format

You can specify the format to be applied to query results in the following ways:

- Specify the desired format using the `FORMAT` clause in a query (the specified format will be applied to this query results only) or the `--format` parameter in the command line (the format will be default for all queries).
- Enter `\G` at the end of a query, or use the `--vertical` parameter in the command line to use the Vertical format for a single query or all queries by default, respectively.
- Specify the output format in the configuration file of `clickhouse-client` (*/etc/clickhouse-client/config.xml*).

Below is an example of changing the data output format for the batch mode:

```
$ clickhouse-client --multiquery --query "select * from test_table1;select * from test_table2;" --format PrettyCompact
```

The output should be similar to:

```
--a---b-------c--
│ 1 │ aaa │ 100 │
│ 2 │ bbb │ 200 │
│ 3 │ ccc │ 300 │
│ 4 │ ddd │ 400 │
│ 5 │ eee │ 500 │
-----------------

--a---b---c--
│ 1 │ a │ 0 │
│ 2 │ b │ 0 │
│ 3 │ c │ 0 │
│ 4 │ d │ 4 │
-------------
```

For more details on using `clickhouse-client` read the clickhouse-client article of the ClickHouse documentation.

Use JDBC driver to access ADQM from JVM-based applications

Elena Dvoryadkina

To Table of Contents

To connect a JVM-based application to an ADQM database, you can use ClickHouse JDBC driver.

Below is an example of a Java application that is built with Maven and interacts with an ADQM server via JDBC driver. The source code of the application is stored in the *<project_path>/src/main/java/io/arenadata/adqm/test/TestApp.java* file, and the *pom.xml* file that contains information about the project structure and configuration is located in the project's root folder — *<project_path>/pom.xml*.

1. In the *pom.xml* file, add the following dependency to download a JDBC driver from the Maven central repository:

```
<dependencies>
    <dependency>
        <groupId>com.clickhouse</groupId>
        <artifactId>clickhouse-jdbc</artifactId>
        <version>0.3.2</version>
        <classifier>all</classifier>
        <exclusions>
            <exclusion>
                <groupId>*</groupId>
                <artifactId>*</artifactId>
            </exclusion>
        </exclusions>
    </dependency>
</dependencies>
```

2. Create a Java class with the following code:

```
package io.arenadata.adqm.test;

import com.clickhouse.jdbc.*;
import java.sql.*;
import java.util.*;

public class TestApp {
    private static Connection getConnection(String url) throws SQLException {
        return DriverManager.getConnection(url);
    }

    public static void main(String[] args) {
        String url = "jdbc:clickhouse://<IP>:8123/default";
        try (
            Connection connection = getConnection(url);
            Statement stmt = connection.createStatement();
            ResultSet rs = stmt.executeQuery("SELECT * FROM <table_name> LIMIT 5")) {
                ResultSetMetaData rsMetaData = rs.getMetaData();
                int columns = rsMetaData.getColumnCount();
                while (rs.next()) {
                    for (int c = 1; c <= columns; c++) {
                        System.out.print(rsMetaData.getColumnName(c) + ":" + rs.getString(c) + (c < columns ? " | " : "\n"));
                    }
                }
        }
        catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

   This code connects to the ADQM server using DriverManager (as the `default` user without a password) and selects the first five rows from the specified table of the `default` database. Replace `<IP>` with the IP-address of your ADQM server and `<table_name>` in the query string with a name of a table existing in the `default` database of your ADQM server.

3. Specify networks from which connections to ADQM are allowed for the `default` user as described in the Connect to ADQM article.

4. Compile and run the project. The result should be similar to:

```
a:1 | b:text1 | c:100
a:2 | b:text2 | c:200
a:3 | b:text3 | c:300
```

Use DBeaver to connect to ADQM

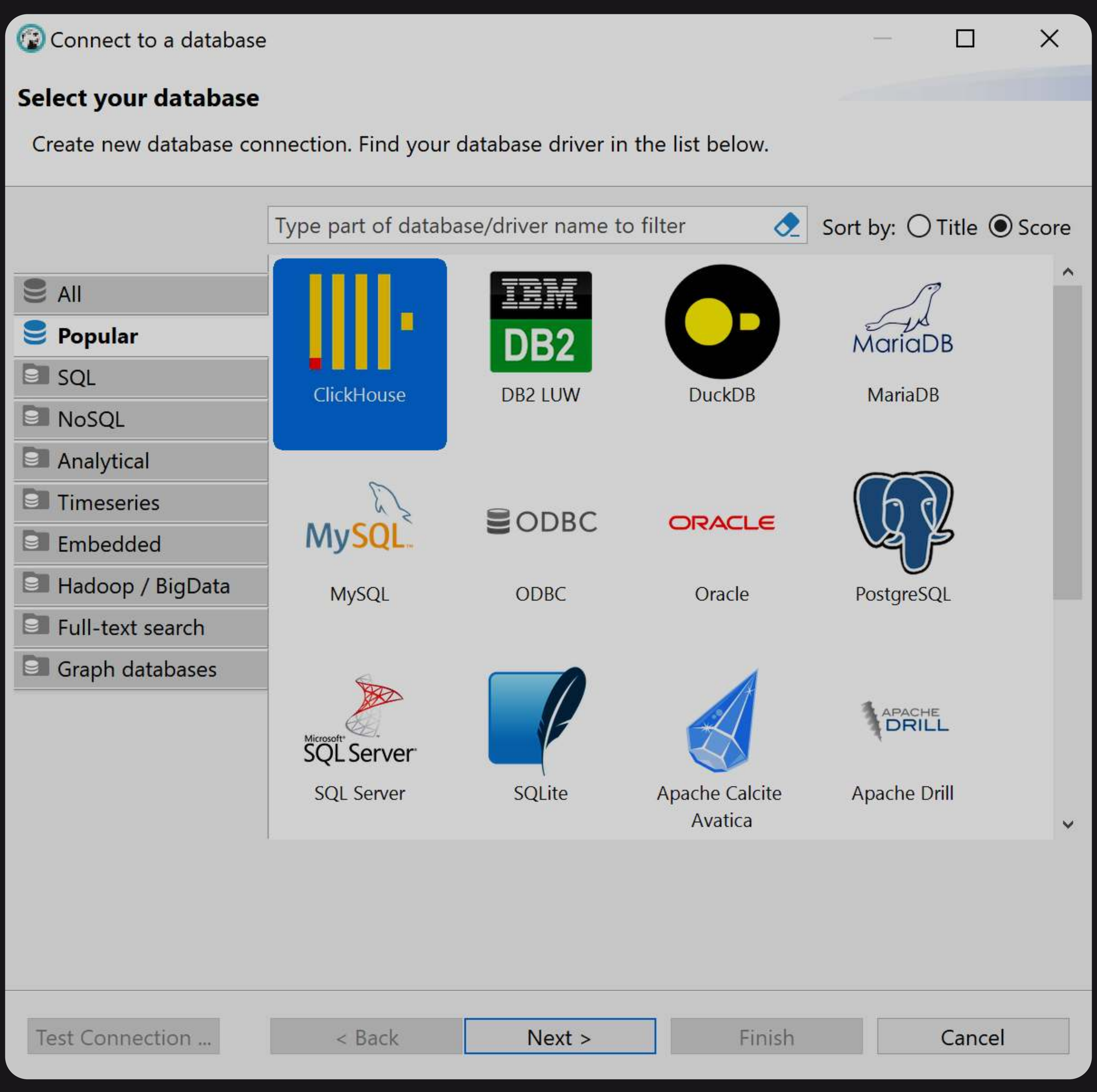
Elena Dvoryadkina

To Table of Contents

To connect to ADQM, you can use DBeaver — a third-party client tool that supports different databases. For information on how to install DBeaver, refer to Installation.

To connect to an ADQM server, launch DBeaver and follow the steps below.

1. Select ***ClickHouse*** in the ***Connect to a database*** window (***Database*** → ***New Database Connection***).



Connect to a database

Select your database

Create new database connection. Find your database driver in the list below.

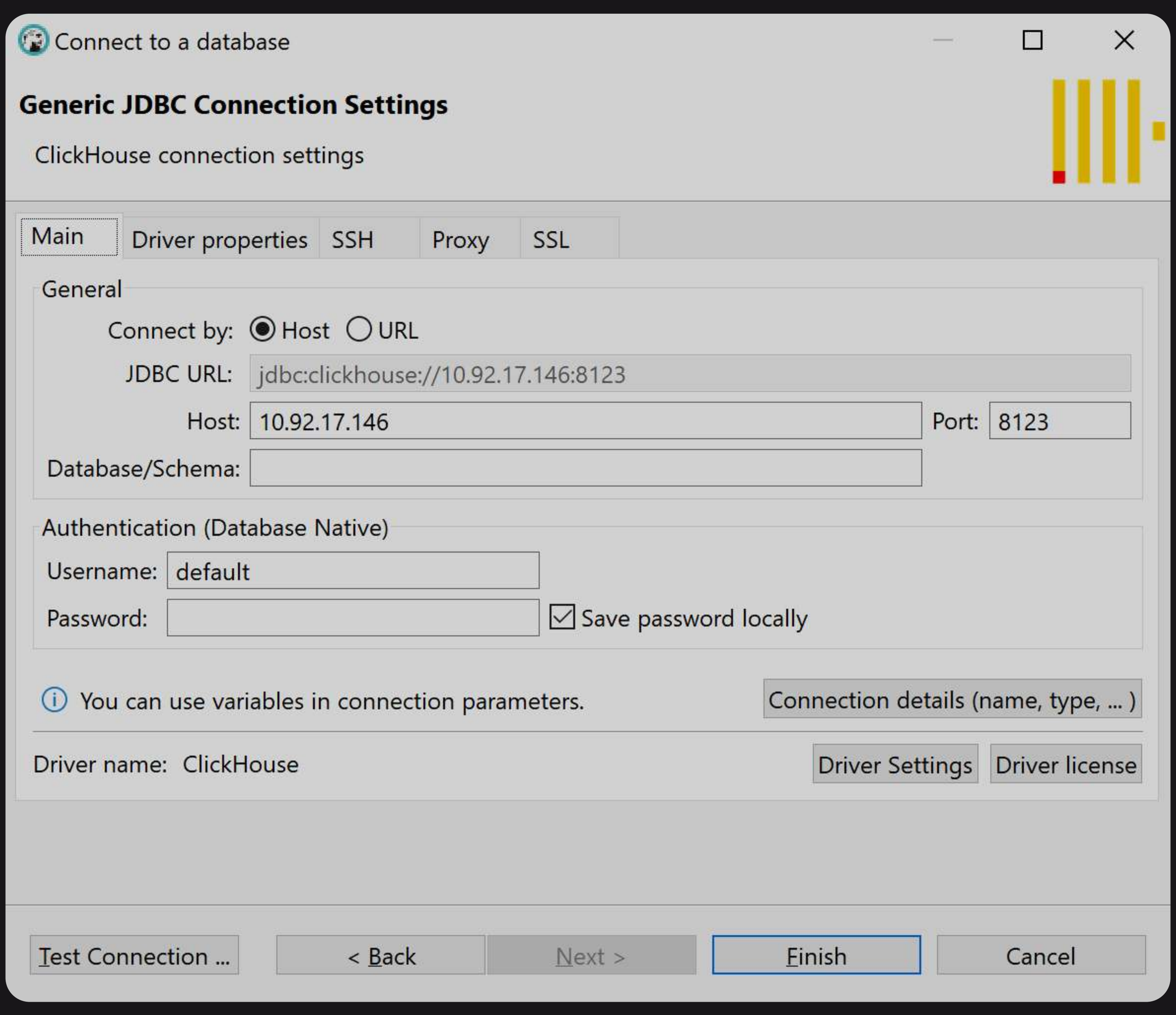
Sort by: ○ Title ● Score

Type part of database/driver name to filter
All
Popular
SQL
NoSQL
Analytical
Timeseries
Embedded
Hadoop / BigData
Full-text search
Graph databases
ClickHouse
DB2 LUW
DuckDB
MariaDB
MySQL
ODBC
Oracle
PostgreSQL
SQL Server
SQLite
Apache Calcite Avatica
Apache Drill
Test Connection ...
< Back
Next >
Finish
Cancel

Select a database type

2. In the connection settings, specify the ADQM host, username, and password (a user should be defined in the `users` section in the *users.xml* configuration file of the ADQM server). The `default` username and empty password are used by default. Ensure that user configuration defines networks from which connections to ADQM are allowed. For an example on how to allow the `default` user to connect to ADQM via DBeaver from any IP address, refer to the Connect to ADQM article.

   In the ***Database/Schema*** field, enter the database name (if you leave this field empty, DBeaver will connect to the `default` database).



Connect to a database

Generic JDBC Connection Settings

ClickHouse connection settings

General

Connect by: ● Host ○ URL

JDBC URL: jdbc:clickhouse://10.92.17.146:8123

Host: 10.92.17.146 Port: 8123

Database/Schema:

Authentication (Database Native)

Username: default

Password:

☑ Save password locally

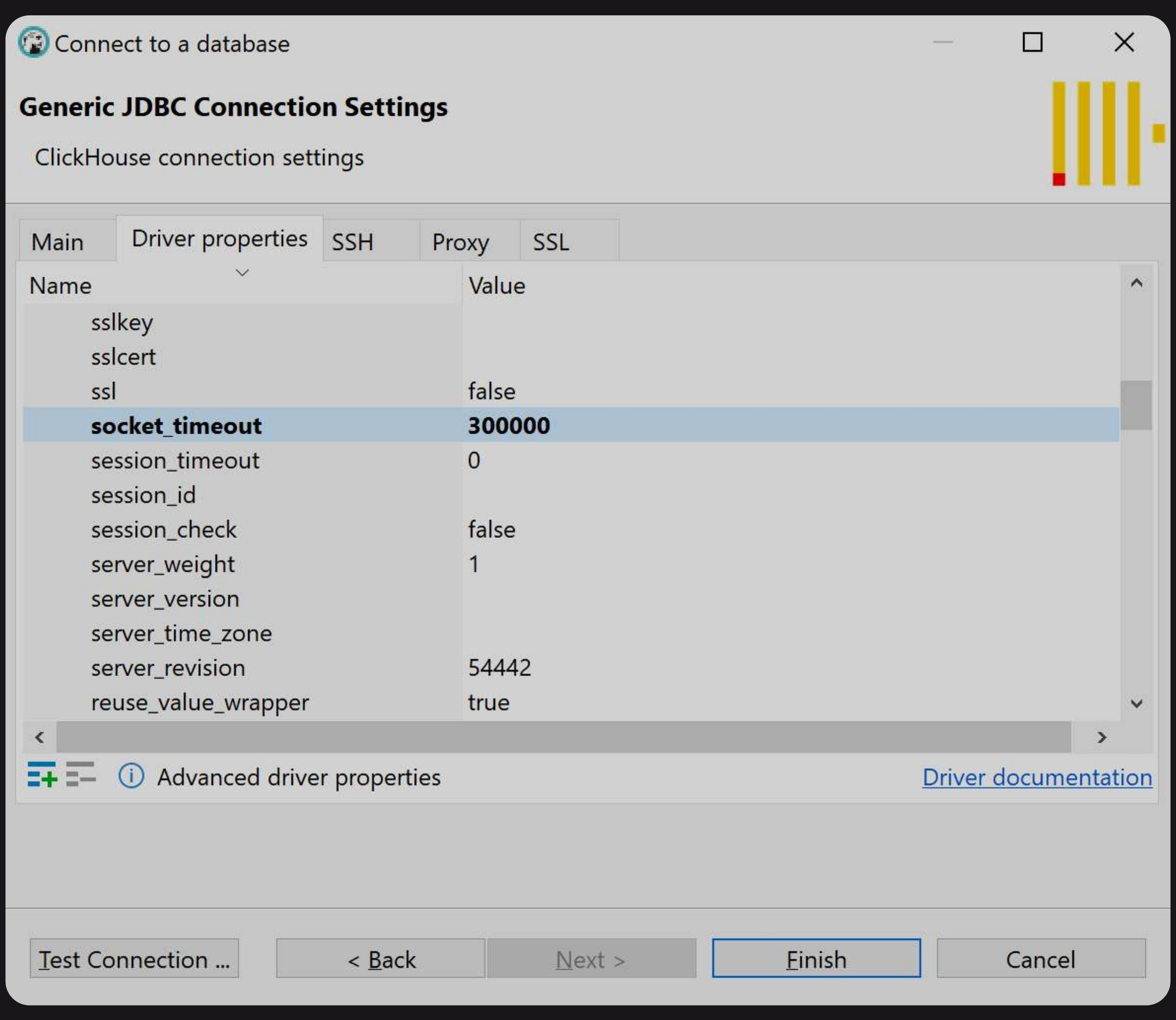
You can use variables in connection parameters.

Driver name: ClickHouse

Main
Driver properties
SSH
Proxy
SSL
Connection details (name, type, ...)
Driver Settings
Driver license
Test Connection ...
< Back
Next >
Finish
Cancel

Connection settings

3. If you need to run analytical queries that take longer than 30 seconds, it is recommended to increase the ***socket_timeout*** property value on the ***Driver properties*** tab (for example, set it to `300000` milliseconds).



Connect to a database

Generic JDBC Connection Settings

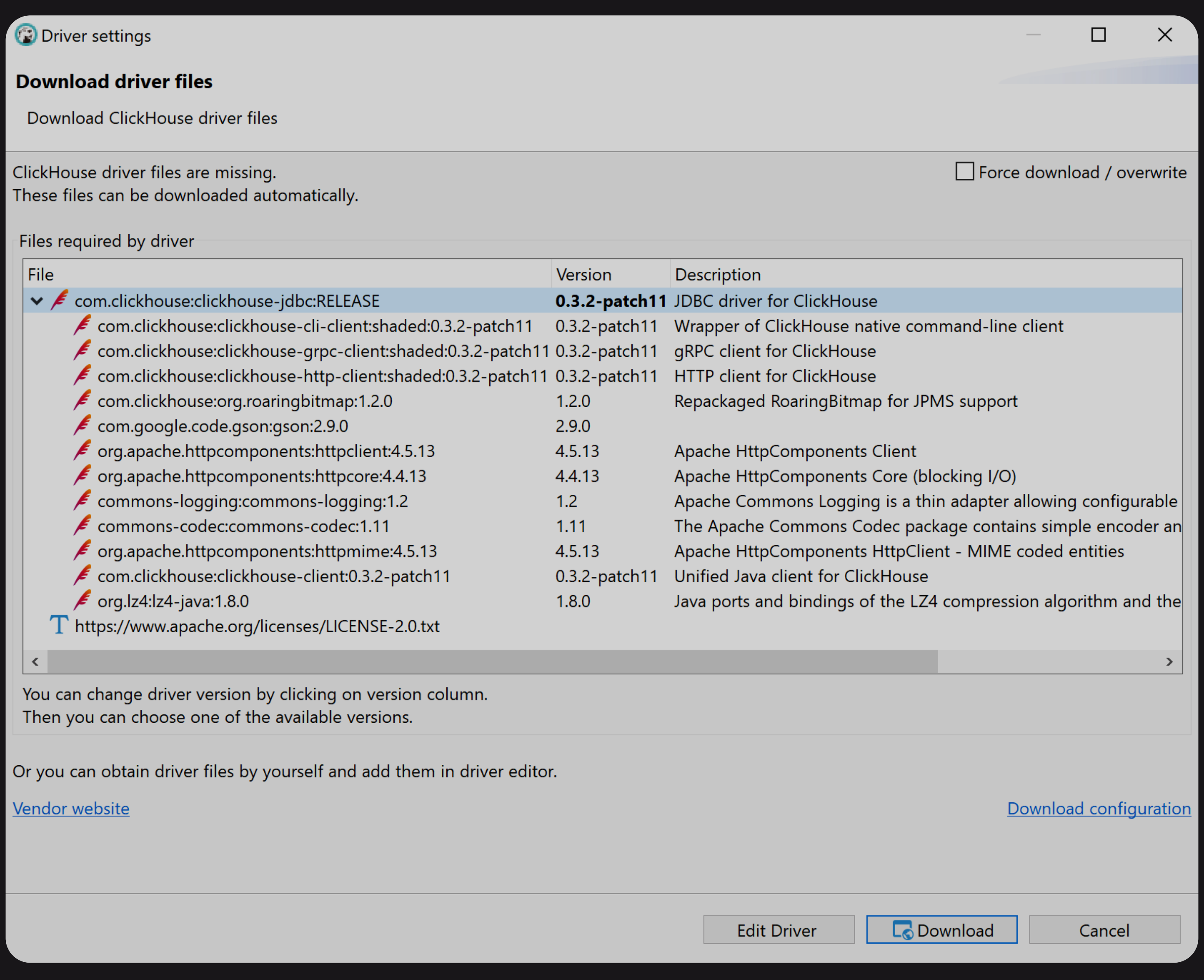
ClickHouse connection settings

Name	Value
sslkey	
sslcert	
ssl	false
socket_timeout	**300000**
session_timeout	0
session_id	
session_check	false
server_weight	1
server_version	
server_time_zone	
server_revision	54442
reuse_value_wrapper	true

Main
Driver properties
SSH
Proxy
SSL
Advanced driver properties
Driver documentation
Test Connection ...
< Back
Next >
Finish
Cancel

Driver properties

4. If you connect to an ADQM server for the first time, download the ClickHouse JDBC driver.



Driver settings

Download driver files

Download ClickHouse driver files

ClickHouse driver files are missing.
These files can be downloaded automatically.

☐ Force download / overwrite

Files required by driver

File	Version	Description
com.clickhouse:clickhouse-jdbc:RELEASE	**0.3.2-patch11**	JDBC driver for ClickHouse
com.clickhouse:clickhouse-cli-client:shaded:0.3.2-patch11	0.3.2-patch11	Wrapper of ClickHouse native command-line client
com.clickhouse:clickhouse-grpc-client:shaded:0.3.2-patch11	0.3.2-patch11	gRPC client for ClickHouse
com.clickhouse:clickhouse-http-client:shaded:0.3.2-patch11	0.3.2-patch11	HTTP client for ClickHouse
com.clickhouse:org.roaringbitmap:1.2.0	1.2.0	Repackaged RoaringBitmap for JPMS support
com.google.code.gson:gson:2.9.0	2.9.0	
org.apache.httpcomponents:httpclient:4.5.13	4.5.13	Apache HttpComponents Client
org.apache.httpcomponents:httpcore:4.4.13	4.4.13	Apache HttpComponents Core (blocking I/O)
commons-logging:commons-logging:1.2	1.2	Apache Commons Logging is a thin adapter allowing configurable
commons-codec:commons-codec:1.11	1.11	The Apache Commons Codec package contains simple encoder an
org.apache.httpcomponents:httpmime:4.5.13	4.5.13	Apache HttpComponents HttpClient - MIME coded entities
com.clickhouse:clickhouse-client:0.3.2-patch11	0.3.2-patch11	Unified Java client for ClickHouse
org.lz4:lz4-java:1.8.0	1.8.0	Java ports and bindings of the LZ4 compression algorithm and the
https://www.apache.org/licenses/LICENSE-2.0.txt		

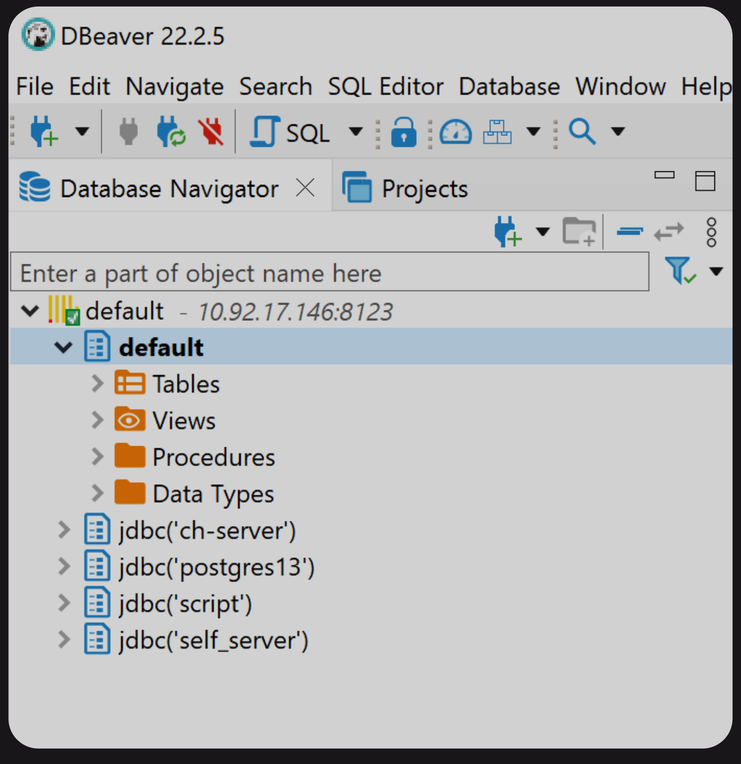
You can change driver version by clicking on version column.
Then you can choose one of the available versions.

Or you can obtain driver files by yourself and add them in driver editor.

Vendor website
Download configuration
Edit Driver
Download
Cancel

Select a ClickHouse driver

After the connection is established, DBeaver will display the database on the ***Database Navigator*** tab.



DBeaver 22.2.5

- default - *10.92.17.146:8123*
  - **default**
    - Tables
    - Views
    - Procedures
    - Data Types
  - jdbc('ch-server')
  - jdbc('postgres13')
  - jdbc('script')
  - jdbc('self_server')

File
Edit
Navigate
Search
SQL Editor
Database
Window
Help
SQL
Database Navigator
Projects
Enter a part of object name here

DBeaver Database Navigator

Authentication

Elena Dvoryadkina

To Table of Contents

When authenticating users, ADQM verifies the identity of users and determines if they are allowed to connect to servers. ADQM uses roles to manage access rights.

The following authentication methods are supported:

- By password — passwords can be hashed using the SHA-256 and double SHA-1 algorithms, or stored as plain text.
- LDAP — an LDAP server can be used as an external authenticator to verify ADQM user passwords or as an external directory to store information about user accounts and their access rights.
- By SSL certificates — users can be authenticated by SSL certificates if SSL channel protection is enabled for ADQM.
- Kerberos — Kerberos-authenticated users can access ADQM through the HTTP interface (a used client should support the SPNEGO mechanism).

Password authentication

Elena Dvoryadkina

Contents

To Table of Contents

- Set user passwords
  - sha256_password
  - double_sha1_password
  - plaintext_password
  - no_password
- Change user passwords
- Set the default user password

ADQM can authenticate users by passwords. It supports the following password authentication methods that differ in how a password is stored on an ADQM server and transmitted over the connection:

- sha256_password — ADQM stores a password hash calculated using the SHA-256 algorithm. For additional password protection, you can use **salt** (SALT) — an arbitrary string that is added to the beginning or end of a password before passing the password to the hash function.
- double_sha1_password — ADQM stores a password hash calculated using the double SHA-1 algorithm (for MySQL compatibility).
- plaintext_password — a password is stored unencrypted. Since this method is insecure, use it only if the connection is protected by SSL encryption.
- no_password — a user can connect to ADQM without password.

You can assign a password to a user in a `CREATE USER` / `ALTER USER` query or in the *users.xml* configuration file.

Set user passwords

sha256_password

To use the SHA-256 algorithm for hashing a user's password, you can specify the password in one of the following ways:

- set the password as a string — ADQM will automatically calculate the SHA-256 hash for the entered string and save it;
- calculate the SHA-256 hash for a desired password string by yourself and then pass the hash to ADQM.

Set password string

Use one of the following queries to set the user password as a string:

```
CREATE USER <user_name> IDENTIFIED WITH sha256_password BY '<user_password>';
```

```
CREATE USER <user_name> IDENTIFIED BY '<user_password>'
```

When executing these queries, ADQM adds a random SALT value to the entered password string, hashes the resulting value, and saves the hash and SALT value.

Example

Query:

```
CREATE USER andrew IDENTIFIED BY 'qwerty';
```

The output:

```
CREATE USER andrew IDENTIFIED WITH sha256_hash BY 'E422256F5E13AB9C590BAFE5A91592BCF2417E8F992D1C74B9CF138C4D3646FF' SALT
'1EE6C0E7CA2F95253F6F0A5C2855AB4D7C00F55AB0BA83E5AB0ED505F168CC06'

Query id: e1de17ec-74fb-4b0a-ad25-e5ef2199d727

Ok.

0 rows in set. Elapsed: 0.001 sec.
```

To connect to ADQM as this user, provide the specified password:

```
$ clickhouse-client --user andrew --password qwerty
```

Set password hash

To assign a password to a user as a hash, use the following query:

```
CREATE USER <user_name> IDENTIFIED WITH sha256_hash BY '<hash>';
```

If you've used a SALT value for the hash calculation, pass the hash and salt to a query as follows:

```
CREATE USER <user_name> IDENTIFIED WITH sha256_hash BY '<hash>' SALT '<salt>';
```

Example

Generate a hash for the `qwerty` string:

```
$ PASSWORD="qwerty"; echo "$PASSWORD"; echo -n "$PASSWORD" | sha256sum | tr -d '-'
```

In the output, the first line is the password and the second line is the password hash:

```
qwerty
65e84be33532fb784c48129675f9eff3a682b27168c0ea744b2cf58ee02337c5
```

When creating a user, pass the generated hash:

```
CREATE USER alex IDENTIFIED WITH sha256_hash BY '65e84be33532fb784c48129675f9eff3a682b27168c0ea744b2cf58ee02337c5';
```

Make sure you can connect to ADQM as the created user with the `qwerty` password:

```
$ clickhouse-client --user alex --password qwerty
```

You can also set the SHA-256 hash in the *users.xml* configuration file — place it in the `password_sha256_hex` element:

```
<users>
    <user_name>
        <password_sha256_hex>65e84be33532fb784c48129675f9eff3a682b27168c0ea744b2cf58ee02337c5</password_sha256_hex>
        <!-- Other settings -->
    </user_name>
</users>
```

double_sha1_password

To use the double SHA-1 algorithm for hashing a user's password, you can:

- set the password as a string — ADQM will automatically calculate the double SHA-1 hash for the entered string and save it;
- calculate the double SHA-1 hash for a desired password string by yourself and then pass the hash to ADQM.

Set password string

Use the following query to set the user password as a string:

```
CREATE USER <user_name> IDENTIFIED WITH double_sha1_password BY '<user_password>';
```

When executing this query, ADQM calculates the hash for the entered string using the double SHA-1 algorithm and stores the hash.

Example

Query:

```
CREATE USER john IDENTIFIED WITH double_sha1_password BY 'qwerty';
```

The output:

```
CREATE USER john IDENTIFIED WITH double_sha1_hash BY 'AA1420F182E88B9E5F874F6FBE7459291E8F4601'

Query id: 7441ff9b-d349-4c29-a687-61612da40323

Ok.

0 rows in set. Elapsed: 0.001 sec.
```

The `john` user can now connect to ADQM using its password:

```
$ clickhouse-client --user john --password qwerty
```

Set password hash

To assign a password as a double sha-1 hash, use the following query:

```
CREATE USER <user_name> IDENTIFIED WITH double_sha1_hash BY '<hash>';
```

Example

Generate a double sha-1 hash for the `qwerty` string:

```
$ PASSWORD="qwerty"; echo "$PASSWORD"; echo -n "$PASSWORD" | sha1sum | tr -d '-' | xxd -r -p | sha1sum | tr -d '-'
```

In the output, the first line is the password and the second line is the password hash:

```
qwerty
AA1420F182E88B9E5F874F6FBE7459291E8F4601
```

When creating a user, pass the generated hash to the `CREATE USER` query as follows:

```
CREATE USER mary IDENTIFIED WITH double_sha1_hash BY 'AA1420F182E88B9E5F874F6FBE7459291E8F4601';
```

Check that you can connect to ADQM as the `mary` user with the `qwerty` password:

```
$ clickhouse-client --user mary --password qwerty
```

You can also set a double SHA-1 hash for a user password in the *users.xml* configuration file. To do this, put the hash in the `password_double_sha1_hex` element:

```
<users>
    <user_name>
        <password_double_sha1_hex>AA1420F182E88B9E5F874F6FBE7459291E8F4601</password_double_sha1_hex>
        <!-- Other settings -->
    </user_name>
</users>
```

plaintext_password

ADQM stores a user password in the clear text format if you set the password in one of the following ways.

- Use the `IDENTIFIED WITH plaintext_password` clause in a `CREATE USER` query:

```
CREATE USER <user_name> IDENTIFIED WITH plaintext_password BY '<user_password>';
```

- Place a password string in the `password` element of the user configuration (the `users` section of the *users.xml* configuration file):

```
<users>
    <user_name>
        <password>user_password</password>
        <!-- Other settings -->
    </user_name>
</users>
```

Empty `password` element allows a user to connect to ADQM without a password:

```
<users>
    <user_name>
        <password></password>
        <!-- Other settings -->
    </user_name>
</users>
```

CAUTION

It is not recommended to store user passwords unencrypted.

no_password

You can also create an ADQM user without a password with one of the following queries:

```
CREATE USER <user_name>;
```

```
CREATE USER <user_name> IDENTIFIED WITH no_password;
```

The `auth_type` field of the system.users table for the user will contain the `no_password` value (unlike a user with an empty `password` element in *users.xml* — in this case, `auth_type` returns `plaintext_password`).

Change user passwords

To change the password for an existing user, execute the `ALTER USER` query:

```
ALTER USER <user_name>
    [NOT IDENTIFIED | IDENTIFIED {[WITH {no_password | plaintext_password | sha256_password | sha256_hash | double_sha1_password |
double_sha1_hash}]
    BY {'<password>' | '<hash>'}}];
```

Example

Run the following query to change the password for the `mary` user:

```
ALTER USER mary IDENTIFIED WITH sha256_password BY 'test';
```

ADQM adds some SALT value to a new password and generates a new SHA-256 hash for resulting string:

```
ALTER USER mary IDENTIFIED WITH sha256_hash BY 'FDBC16CDEBD27426D53E8EF5AC17A02894CB195C3F213009913E5827228A7049' SALT
'48F8DEDC2E1818BE4888553B8A55B2705F1E98494CB0387A36D6F89F06DF745B'

Query id: 6adb558b-3370-4829-9a60-f8264d366a00

Ok.

0 rows in set. Elapsed: 0.001 sec.
```

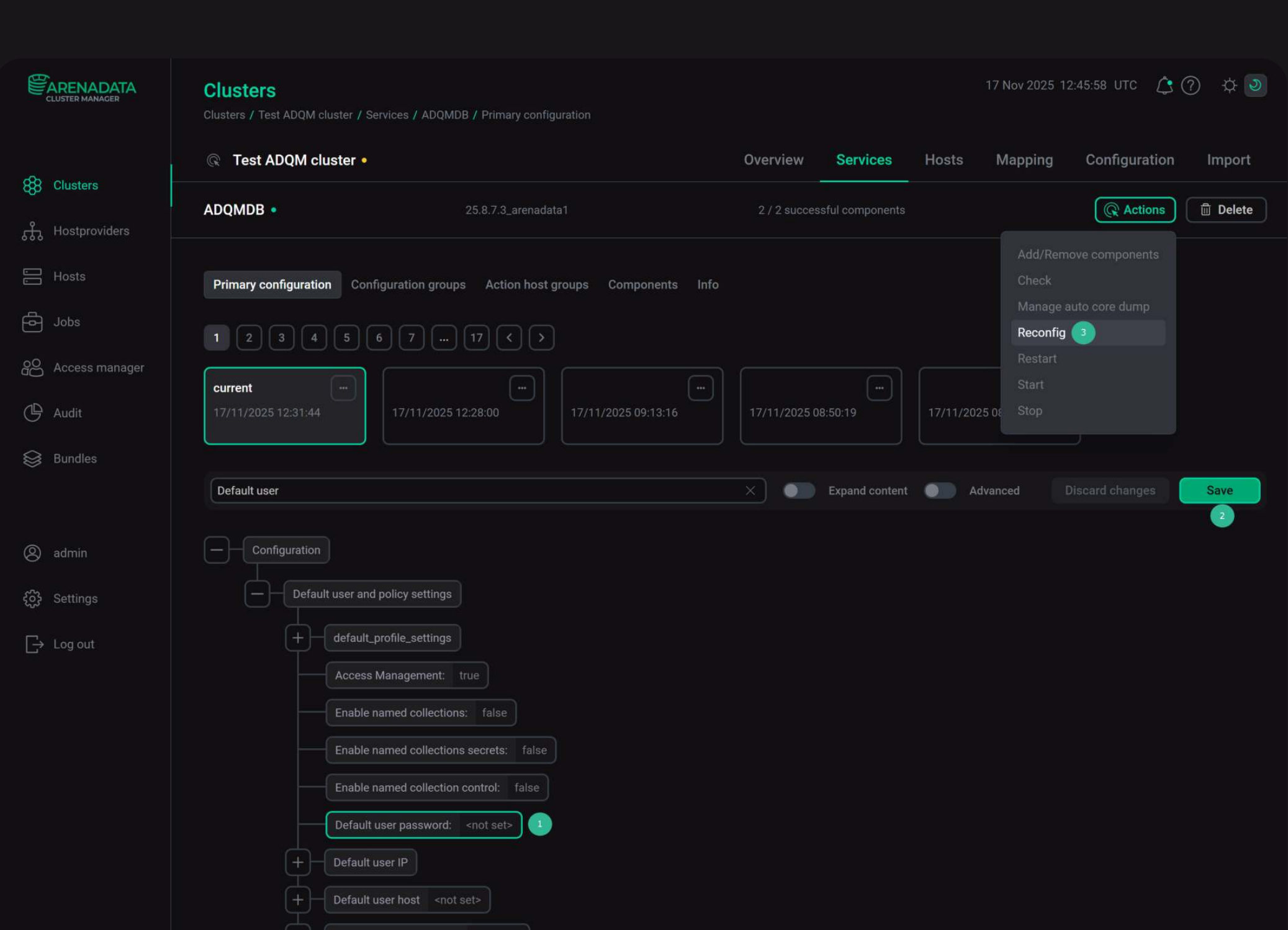
Now, to connect to ADQM as the `mary` user, you should use the `test` password:

```
$ clickhouse-client --user mary --password test
```

To modify a user password, you can also enter a new value in the corresponding element (`password_sha256_hex`, `password_double_sha1_hex` or `password`) for the user in the *users.xml* file.

Set the default user password

By default, ADQM creates a `default` user that can connect without a password. You can set or change this user password through the ADCM interface. To do this, go to the configuration settings of the ADQMDB service (***ADQM cluster*** → ***Services*** → ***ADQMDB*** → ***Primary Configuration***), set and confirm the password in the ***Configuration*** → ***Default user and policy settings*** section, click ***Save***, and execute the ***Reconfig*** action for the ADQMDB service.



Clusters

Clusters / Test ADQM cluster / Services / ADQMDB / Primary configuration

Test ADQM cluster

ADQMDB 25.8.7.3_arenadata1 2 / 2 successful components

Default user

Configuration

Default user and policy settings

default_profile_settings

Access Management: true

Enable named collections: false

Enable named collections secrets: false

Enable named collection control: false

Default user password: <not set>

Default user IP

Default user host <not set>

Default user host_regexp <not set>

Change the default user password

LDAP authentication

Elena Dvoryadkina

Contents

To Table of Contents

Configure connection to the LDAP server

Use LDAP to authenticate ADQM users

Use LDAP to store usernames and permissions

To authenticate users, ADQM can use a remote LDAP server as:

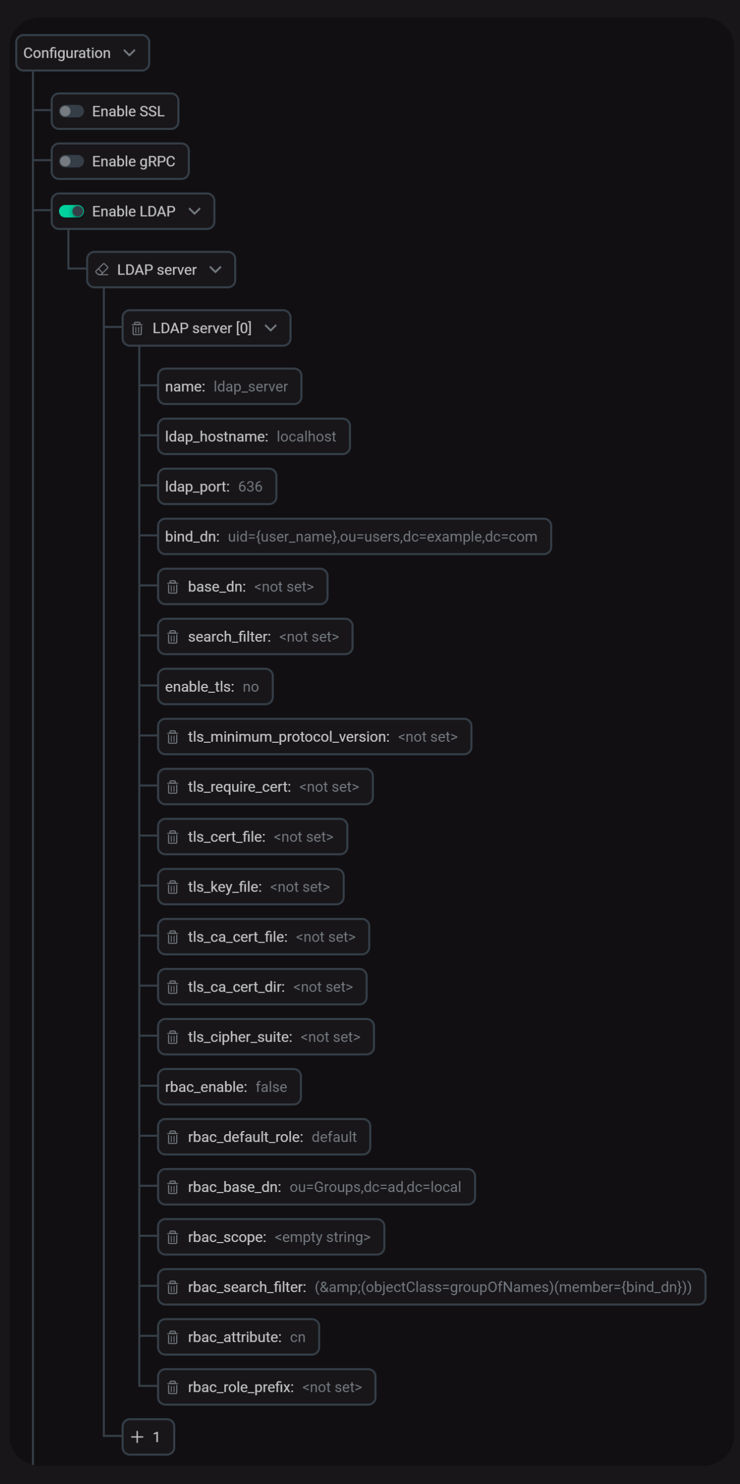
- an external authenticator for existing ADQM users;
- an external user directory that holds information about user accounts and their roles. This approach allows authentication of users that are not created in ADQM directly but defined on the LDAP server.

Both these approaches require you to configure ADQM settings that manage connection to the LDAP server.

Configure connection to the LDAP server

In the ADCM interface, open the configuration page for the ADQMDB service (***ADQM cluster*** → ***Services*** → ***ADQMDB*** → ***Primary Configuration***) and activate the ***Configuration*** → ***Enable LDAP*** option. In the expanded section, configure the following parameters:

- ***name*** — name of the LDAP server connection configuration.
- ***ldap_hostname*** — hostname or IP of the LDAP server.
- ***ldap_port*** — port of the LDAP server (if the ***enable_tls*** setting is set to `yes`, the default port is `636`, otherwise — `389`).
- ***bind_dn*** — template used to construct the DN to bind to. The resulting DN will be constructed by replacing all `{user_name}` substrings of the template with the actual username during each authentication attempt.
- ***base_dn*** and ***search_filter*** — LDAP search parameters for detecting the actual DN of the bound user (it is mainly used in search filters for further role mapping when the server is Active Directory — the resulting user DN will replace `{user_dn}` substrings).
- If secure connection to the LDAP server is required, enable the ***enable_tls*** option and specify paths to the key and SSL/TLS certificates using the corresponding parameters (for more details, see Configuration parameters).



name: ldap_server
ldap_hostname: localhost
ldap_port: 636
bind_dn: uid={user_name},ou=users,dc=example,dc=com
base_dn: <not set>
search_filter: <not set>
enable_tls: no
tls_minimum_protocol_version: <not set>
tls_require_cert: <not set>
tls_cert_file: <not set>
tls_key_file: <not set>
tls_ca_cert_file: <not set>
tls_ca_cert_dir: <not set>
tls_cipher_suite: <not set>
rbac_enable: false
rbac_default_role: default
rbac_base_dn: ou=Groups,dc=ad,dc=local
rbac_scope: <empty string>
rbac_search_filter: (&amp;(objectClass=groupOfNames)(member={bind_dn}))
rbac_attribute: cn
rbac_role_prefix: <not set>

LDAP server connection settings

After you've specified all parameters for connecting to the LDAP server, click ***Save*** and execute the ***Reconfig*** action with the ***Restart*** option for the ADQMDB service to apply the changes. The *config.xml* file will get the `<ldap_servers>` section with the specified LDAP server configuration under the name you have assigned to it (for example, `ldap_server` for the configuration in the image above). The next section explains how to use this name for creating ADQM users to be authenticated via LDAP.

Use LDAP to authenticate ADQM users

A remote LDAP server can be used to store and manage passwords of users defined locally in ADQM (created with a `CREATE USER` query or defined in the *users.xml* file). This allows you not to specify a password when creating a user in ADQM, but use the LDAP server to verify the password entered during authentication. To use this approach, specify the LDAP server configuration when defining an ADQM user in one of the following ways (the examples below use `ldap_server` as the LDAP server configuration name).

- Use the `IDENTIFIED WITH ldap SERVER` clause in a `CREATE USER` query:

```
CREATE USER <ldap_user> IDENTIFIED WITH ldap SERVER 'ldap_server';
```

- Specify the `ldap` element for the user in the *users.xml* configuration file:

```
<users>
    <ldap_user>
        <ldap>
            <server>ldap_server</server>
        </ldap>
        <!- ... -->
    </ldap_user>
</users>
```

where `<ldap_user>` is an ADQM user name (it should match the username defined on the LDAP server — a string between `auth_dn_prefix` and `auth_dn_suffix` in an entry like `dn: auth_dn_prefix{user_name}auth_dn_suffix`).

The `ldap_user` user will then be able to connect to ADQM with the password specified in LDAP.

Use LDAP to store usernames and permissions

An LDAP server can also be used as an external user directory for ADQM. In other words, you can not create users in ADQM locally, but allow user authentication by information from the LDAP directory.

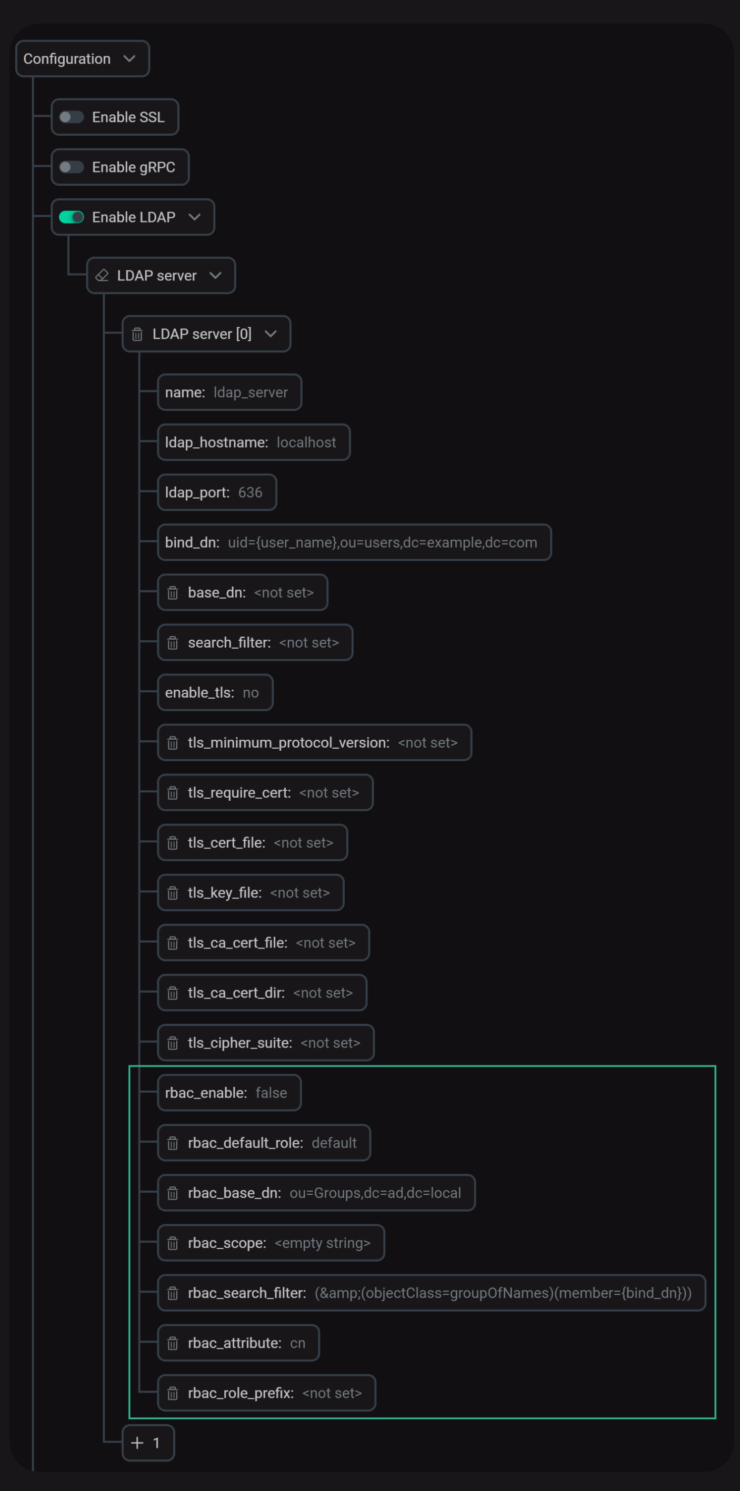
On each user login, ADQM tries to find the local definition of the user and authenticate the user as specified. If the user does not exist in ADQM, it is assumed that the user is defined in an external LDAP directory. In this case, ADQM tries to bind to DN (defined by the LDAP server connection parameters) with provided login credentials. If successful, the user is considered existing and authenticated.

For users authenticated by information from the LDAP directory, it is required to assign roles as described below.

1. Create one or more roles in ADQM that will be assigned to authenticated users. To do this, use a `CREATE ROLE` query:

```
CREATE ROLE <ldap_user_role>;
```

2. On the ADQMDB service configuration page (the ***Configuration*** → ***Enable LDAP*** section), enable the ***rbac_enable*** option (role-based access control) within the ***LDAP server*** settings and set the parameters below depending on how roles should be assigned to users authenticated via the LDAP directory. There are two ways to set user roles.
   - **Set a default role for all authenticated users.** Pass the role created earlier with the `CREATE ROLE` query to the ***rbac_default_role*** parameter.
   - **Map roles from the LDAP directory to ADQM roles.** Specify the following parameters:
     - ***rbac_base_dn*** — base path to start a search for groups with a user.
     - ***rbac_scope*** — scope of the LDAP search (possible values are `base`, `one_level`, `children`, `subtree`).
     - ***rbac_search_filter*** — LDAP search filter to identify groups for mapping users. The filter template can include the `{user_name}`, `{bind_dn}`, `{user_dn}`, and `{base_dn}` substrings that will be replaced with the actual username, bind DN, user DN, and base DN in the resulting filter during each LDAP search.
     - ***rbac_attribute*** — name of the attribute whose values the LDAP search should return (these values will be mapped to ADQM roles).
     - ***rbac_role_prefix*** — prefix of attribute values that the LDAP search should return. The specified prefix will be removed from the found attribute values, and the resulting strings will be treated as names of ADQM local roles — these roles will be assigned to an authenticated user.



name: ldap_server
ldap_hostname: localhost
ldap_port: 636
bind_dn: uid={user_name},ou=users,dc=example,dc=com
base_dn: <not set>
search_filter: <not set>
enable_tls: no
tls_minimum_protocol_version: <not set>
tls_require_cert: <not set>
tls_cert_file: <not set>
tls_key_file: <not set>
tls_ca_cert_file: <not set>
tls_ca_cert_dir: <not set>
tls_cipher_suite: <not set>
rbac_enable: false
rbac_default_role: default
rbac_base_dn: ou=Groups,dc=ad,dc=local
rbac_scope: <empty string>
rbac_search_filter: (&amp;(objectClass=groupOfNames)(member={bind_dn}))
rbac_attribute: cn
rbac_role_prefix: <not set>

RBAC-related settings for LDAP authentication

To complete parameter configuration and apply all changes, click ***Save*** and execute ***Reconfig*** for the ADQMDB service. The specified values will be written to the `roles` (default role) and `role_mapping` (parameters for searching roles in the LDAP directory to map to ADQM roles) sections inside the `user_directories/ldap` section of the *config.xml* configuration file.

You can use one or both of the described methods to assign roles to users. An authenticated user gets a default role (`Default role`). If the role mapping is also configured, the LDAP search results are converted to role names and the corresponding ADQM roles are also assigned to the user. If none of the role assignment methods is configured, users will not be able to perform any actions after authentication.

SSL certificate authentication

Elena Dvoryadkina

Contents

To Table of Contents

Create an SSL user certificate

Configure ADQM for SSL certificate authentication

Create a user

Example

When connecting to ADQM by the SSL protocol, SSL certificates of incoming connections are required to be checked — only connections with trusted certificates can be established. Thus, ADQM can uniquely authenticate an incoming connection, and can identify a user by the corresponding user certificate. You can set multiple certificates for one user.

This article describes the sequence of steps required to enable SSL certificate authentication. To run commands below, you need `openssl` installed — it is a command-line tool for using functions of the OpenSSL cryptographic library.

NOTE

- SSL authentication can only be used if a user connects to ADQM through a client that supports the HTTPS interface.
- In this article, a self-signed CA certificate is used to create a user certificate for demonstration and testing purposes. For production systems, use a trusted certificate authority to sign certificates.

Create an SSL user certificate

1. Generate a key and create a certificate signing request (CSR) file:

```
$ openssl req \
  -newkey rsa:2048 \
  -nodes \
  -subj "/CN=<user_cert_identifier>" \
  -keyout <user_cert_name>.key \
  -out <user_cert_name>.csr
```

   The CN (Common Name) certificate identifier can be any unique value. This value will be used when creating an ADQM user.

2. Create and sign a user certificate that will be used for authentication:

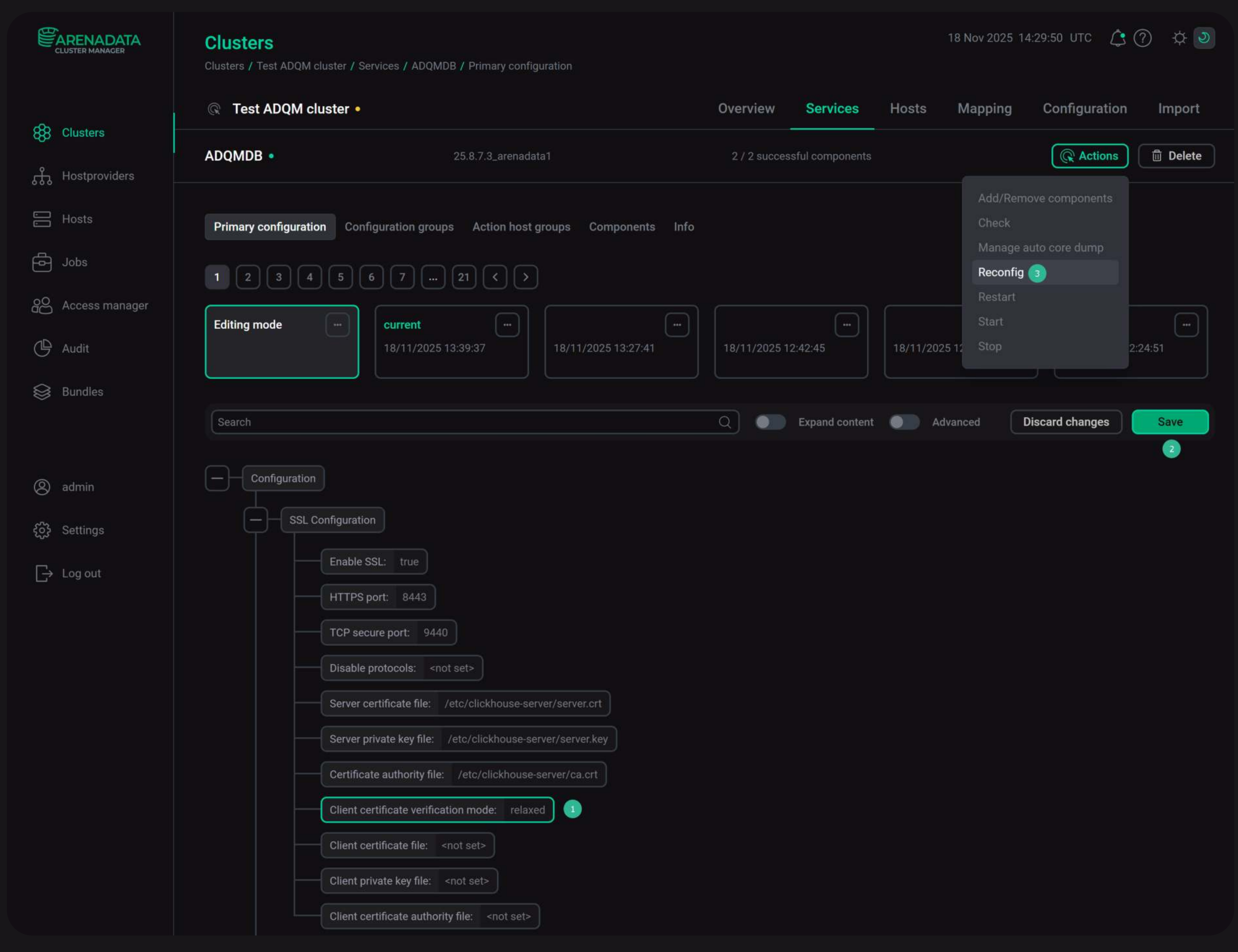
```
$ openssl x509 \
  -req \
  -in <user_cert_name>.csr \
  -out <user_cert_name>.crt \
  -CAcreateserial \
  -CA <ca>.crt \
  -CAkey <ca>.key \
  -days 365
```

   where `<ca>.crt` and `<ca>.key` are files used when configuring ADQM for secure connections over SSL (for details, see Encryption of database connections):

   - `<ca>.crt` — public root certificate of the certificate authority (CA). It should be stored on each ADQM server of the cluster and specified in the ADQMDB service configuration settings (in the ADCM interface: ***ADQM cluster → Services → ADQMDB → Primary configuration → SSL Configuration → Certificate authority file***).
   - `<ca>.key` — private key of the certification authority. It should be kept secret.

Configure ADQM for SSL certificate authentication

In the ADCM interface, go to the ADQMDB service configuration page and set a method of checking user SSL certificates (`relaxed`, `strict`, or `once`) using the ***Client certificate verification mode*** option in the ***SSL Configuration*** section. Click ***Save*** and run the ***Reconfig*** action with the ***Restart*** option for the ADQMDB service to save and apply the parameter value change.



ARENADATA CLUSTER MANAGER
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Clusters
Clusters / Test ADQM cluster / Services / ADQMDB / Primary configuration
18 Nov 2025 14:29:50 UTC
Test ADQM cluster
Overview
Services
Hosts
Mapping
Configuration
Import
ADQMDB
25.8.7.3_arenadata1
2 / 2 successful components
Actions
Delete
Add/Remove components
Check
Manage auto core dump
Reconfig
3
Restart
Start
Stop
Primary configuration
Configuration groups
Action host groups
Components
Info
1 2 3 4 5 6 7 ... 21
Editing mode
current
18/11/2025 13:39:37
18/11/2025 13:27:41
18/11/2025 12:42:45
18/11/2025 12
2:24:51
Search
Expand content
Advanced
Discard changes
Save
2
Configuration
SSL Configuration
1

Enable SSL: true
HTTPS port: 8443
TCP secure port: 9440
Disable protocols: <not set>
Server certificate file: /etc/clickhouse-server/server.crt
Server private key file: /etc/clickhouse-server/server.key
Certificate authority file: /etc/clickhouse-server/ca.crt
Client certificate verification mode: relaxed
Client certificate file: <not set>
Client private key file: <not set>
Client certificate authority file: <not set>

Enable verification of user SSL certificates

Create a user

To create an ADQM user that will authenticate with an SSL certificate, use the `IDENTIFIED WITH ssl_certificate` clause in a `CREATE USER` query and specify the user certificate identifier (CN):

```
CREATE USER <user_name> IDENTIFIED WITH ssl_certificate CN '<user_cert_identifier>';
```

If you define a user in the *users.xml* configuration file, specify the `common_name` element with the user certificate identifier in the `certificates` section for the user (the list of certificates can include more than one element):

```
<users>
    <user_name>
        <certificates>
            <common_name>user_cert_identifier</common_name>
            <!-- More names -->
        </certificates>
        <!-- Other settings -->
    </user_name>
</users>
```

Example

This example shows how to create and use an SSL certificate for authentication of a user that connects to an ADQM cluster for which the SSL encryption of connections is enabled as the Encryption of database connections article describes.

1. Generate a key and create a certificate signing request file:

```
$ openssl req \
  -newkey rsa:2048 \
  -nodes -subj "/CN=arenadata:john" \
  -keyout arenadata_john.key \
  -out arenadata_john.csr
```

2. Create and sign a certificate for user authentication:

```
$ openssl x509 \
  -req \
  -in arenadata_john.csr \
  -out arenadata_john.crt \
  -CAcreateserial \
  -CA arenadata_ca.crt \
  -CAkey arenadata_ca.key \
  -days 365
```

   In this query, *arenadata_ca.crt* is the root CA certificate and *arenadata_ca.key* is the private CA key.

3. In the ADQMDB service settings, turn on checking user certificates using the ***Client certificate verification mode*** option.

4. Create a user (for example, on the *adqm-host-1.arenadata.local* host) that ADQM will authenticate by the provided certificate:

```
CREATE USER john IDENTIFIED WITH ssl_certificate CN 'arenadata:john';
```

5. Grant privileges to the user (for example, privileges to read all tables):

```
GRANT SELECT ON *.* TO john WITH GRANT OPTION;
```

6. Check the connection to ADQM over HTTPS as the created user. For example, run the following query from the directory with certificates:

```
$ echo 'SHOW TABLES' | \
  curl 'https://adqm-host-1.arenadata.local:8443' \
  --cert ./arenadata_john.crt \
  --key arenadata_john.key \
  --cacert arenadata_ca.crt \
  -H "X-ClickHouse-SSL-Certificate-Auth: on" \
  -H "X-ClickHouse-User: john" \
  --data-binary @-
```

   If the user passes authentication, the query will return a list of database tables.

Kerberos authentication

Elena Dvoryadkina

Contents

To Table of Contents

KDC types

Common requirements

ADQM provides the ability to authenticate existing users via the Kerberos protocol that provides a mechanism for mutual authentication of a client and server before establishing a connection between them.

Basic concepts used in Kerberos:

- **Principal** — unique name of a client (user) that is allowed to authenticate with Kerberos.
- **Ticket** — temporary data issued to the client for authentication on the server where the required service is located.
- **Key distribution center** (**KDC**) — service that issues Kerberos tickets and temporary session keys.
- **Realm** — Kerberos network that includes KDC servers and one or multiple clients. The realm name is case-sensitive (it is usually written in uppercase) and is the same as the domain name.

KDC types

The following KDC types are available for an ADQM cluster kerberization:

- MIT Kerberos that consists of the principal database and the storage of Kerberos keys.
- MS Active Directory that consists of the principal database and the storage of Windows Server keys.
- FreeIPA that is a free open source identity management system for Linux/UNIX environments.

When using any of these KDC types, the process of identifying and authenticating users is almost the same. The only difference is the KDC type that is being used.

Common requirements

To kerberize an ADQM cluster, you have to assure the following:

- ADQM cluster is installed, configured, and ready for work.
- MIT Kerberos, Active Directory, or FreeIPA user and key storage is set up and ready for work.
- There is a MIT Kerberos, Active Directory, or FreeIPA administrator account that has full access to create, delete, and manage user accounts.

Users configured for Kerberos authentication can access ADQM only via the HTTP interface, and an HTTP client should support the SPNEGO mechanism. For example, you can use the `curl` utility. The following command allows you to verify whether your `curl` version supports Kerberos and SPNEGO:

```
$ curl --version
```

The output should be similar to:

```
curl 7.81.0 (x86_64-pc-linux-gnu) libcurl/7.81.0 OpenSSL/1.0.2k-fips zlib/1.2.7
Release-Date: 2022-01-05
Protocols: dict file ftp ftps gopher gophers http https imap imaps mqtt pop3 pop3s rtsp smb smbs smtp smtps telnet tftp
Features: alt-svc AsynchDNS GSS-API HSTS HTTPS-proxy IPv6 Kerberos Largefile libz NTLM NTLM_WB SPNEGO SSL UnixSockets
```

Use MIT Kerberos server

Elena Dvoryadkina

Contents

To Table of Contents

Setup Kerberos server

Start MIT Kerberos KDC on ADQM cluster

Test user authentication via Kerberos

Setup Kerberos server

NOTE

- Kerberos KDC can be installed on any host that has access to a host with ADCM installed (FQDN connectivity).
- KDC can also be installed on the same host as ADCM.

An example below describes how to install Kerberos KDC to the *dev-adcm-test.ru-central1.internal* host with the CentOS operating system and ADCM.

1. Install Kerberos packages to the host:

```
$ sudo -i
```

```
yum install krb5-libs krb5-server krb5-workstation
```

2. Make changes to the configuration files.

   - In the */var/kerberos/krb5kdc/kdc.conf* file, set realm to `RU-CENTRAL1.INTERNAL`:

```
[kdcdefaults]
 kdc_ports = 88
 kdc_tcp_ports = 88

[realms]
 RU-CENTRAL1.INTERNAL = {
  #master_key_type = aes256-cts
  acl_file = /var/kerberos/krb5kdc/kadm5.acl
  dict_file = /usr/share/dict/words
  admin_keytab = /var/kerberos/krb5kdc/kadm5.keytab
  supported_enctypes = aes256-cts:normal aes128-cts:normal des3-hmac-sha1:normal arcfour-hmac:normal camellia256-cts:normal camellia128-cts:normal des-hmac-sha1:normal des-cbc-md5:normal des-cbc-crc:normal
 }
```

   - In the */etc/krb5.conf* file, set `RU-CENTRAL1.INTERNAL` as the `default_realm` parameter value, and also fill in the `[realms]` and `[domain_realm]` sections:

```
# Configuration snippets may be placed in this directory as well
includedir /etc/krb5.conf.d/

[logging]
 default = FILE:/var/log/krb5libs.log
 kdc = FILE:/var/log/krb5kdc.log
 admin_server = FILE:/var/log/kadmind.log

[libdefaults]
 dns_lookup_realm = false
 ticket_lifetime = 24h
 renew_lifetime = 7d
 forwardable = true
 rdns = false
 pkinit_anchors = FILE:/etc/pki/tls/certs/ca-bundle.crt
 default_realm = RU-CENTRAL1.INTERNAL
 default_ccache_name = KEYRING:persistent:%{uid}

[realms]
 RU-CENTRAL1.INTERNAL = {
  kdc = dev-adcm-test.ru-central1.internal
  admin_server = dev-adcm-test.ru-central1.internal
 }

[domain_realm]
 .ru-central1.internal = RU-CENTRAL1.INTERNAL
 ru-central1.internal = RU-CENTRAL1.INTERNAL
```

   - Modify the */var/kerberos/krb5kdc/kadm5.acl* file as shown below so that any `/admin` principal in the `RU-CENTRAL1.INTERNAL` realm has all administrative privileges except key extraction:

```
*/admin@RU-CENTRAL1.INTERNAL    *
```

   For detailed descriptions of tags in these files, see the kdc.conf, krb5.conf, and kadm5.acl articles of the MIT Kerberos documentation.

3. Create the Kerberos database using the `kdb5_util` utility:

```
kdb5_util create -s -P P@ssw0rd
```

4. Run the following commands in turn to start the Kerberos and KDC services:

```
systemctl start kadmin
systemctl start krb5kdc
```

   You can check the status of running services with the commands:

```
systemctl status kadmin
systemctl status krb5kdc
```

5. Create user principals. To do this, use the Kerberos administrator console — `kadmin.local`.
   A user principal name in a Kerberos database usually follows the `primary/instance@REALM` pattern, where:

   - `primary` is a username (for successful authentication to ADQM, it is required that this part of a principal matches the name of an ADQM user configured to authenticate via Kerberos);
   - `instance` is an optional component that qualifies `primary`. If `primary` is a username, then `instance` is usually not specified or can be used to define an additional principal for the user (for example, `john@RU-CENTRAL1.INTERNAL` and `john/admin@RU-CENTRAL1.INTERNAL` are separate principals with different passwords and permissions).

```
kadmin.local -q "add_principal -pw P@ssw0rd admin/admin"
```

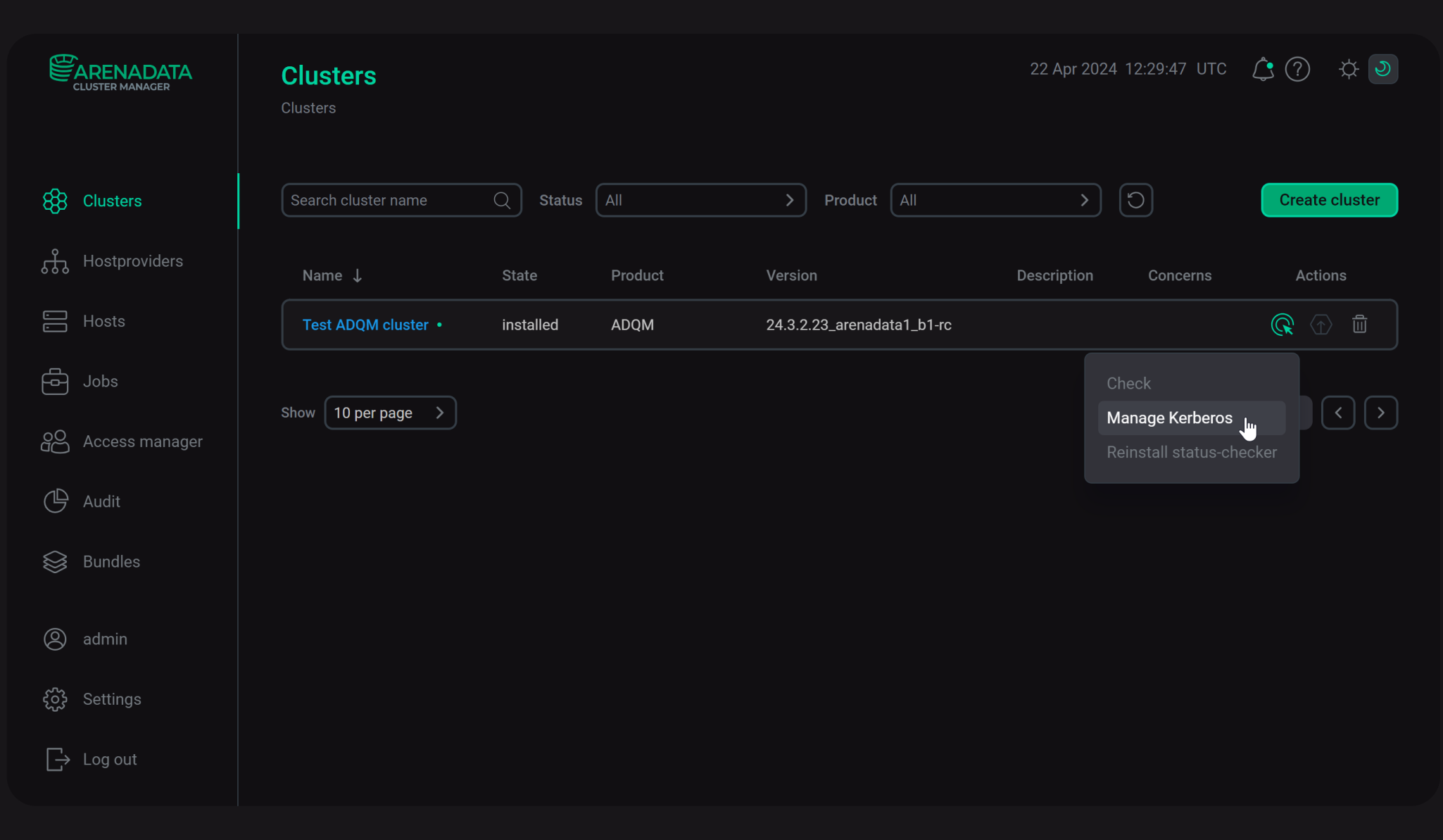
```
kadmin.local -q "add_principal -pw P@ssw0rd adqm_kerb"
```

   For a complete description of the `kadmin.local` command line interface and its options, see the kadmin article of the MIT Kerberos documentation.

Start MIT Kerberos KDC on ADQM cluster

1. On the ***Clusters*** page of the ADCM interface, select the installed and prepared ADQM cluster.

2. Enable Kerberos for the cluster — click the ***Actions*** icon and select the ***Manage Kerberos*** action.

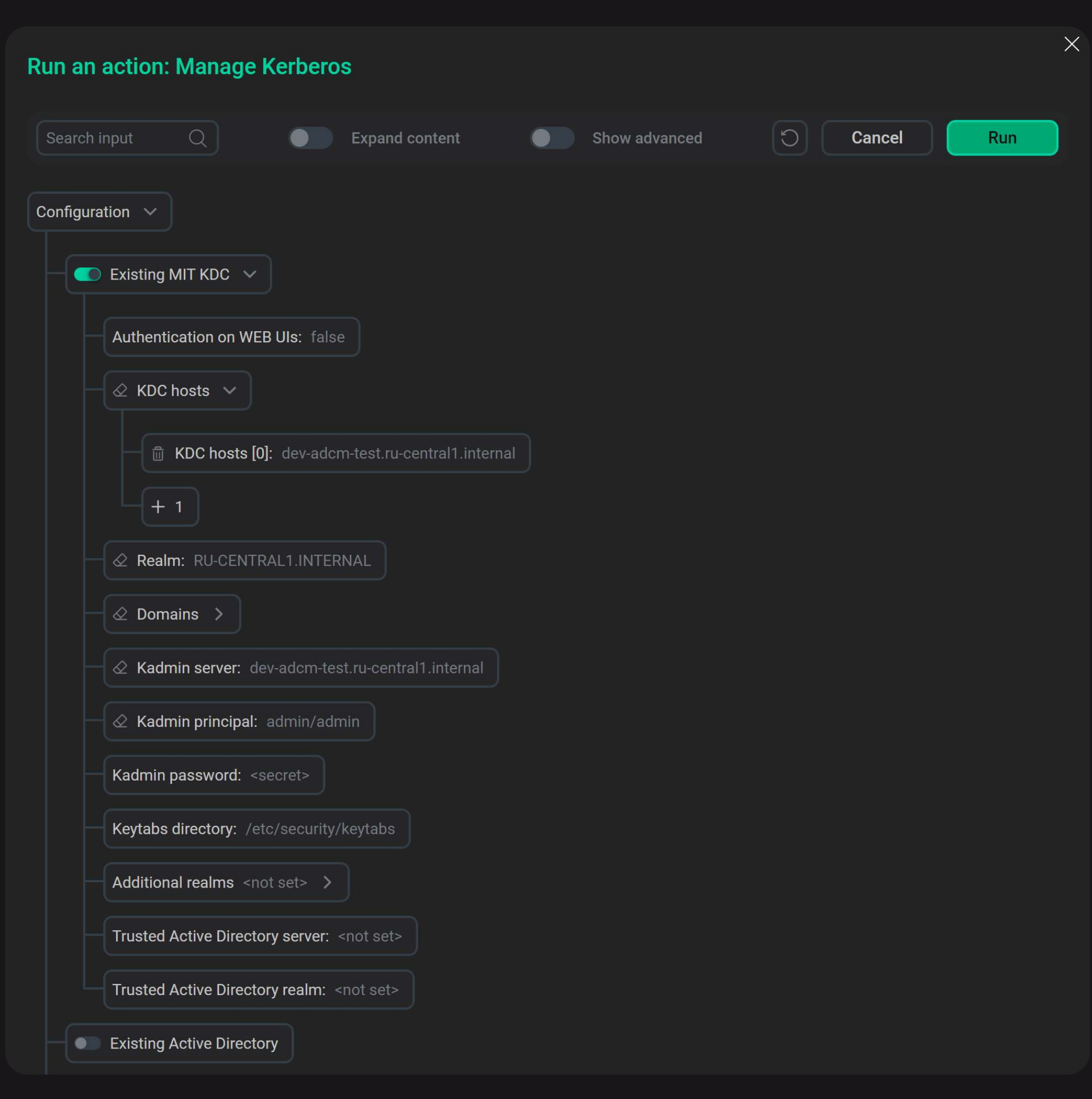


ARENADATA CLUSTER MANAGER
Clusters
Clusters
22 Apr 2024 12:29:47 UTC
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Search cluster name
Status All
Product All
Create cluster
Check
Manage Kerberos
Reinstall status-checker
Show 10 per page

Name	State	Product	Version	Description	Concerns	Actions
Test ADQM cluster	installed	ADQM	24.3.2.23_arenadata1_b1-rc			

Manage Kerberos action

3. Turn on the ***Existing MIT KDC*** option and fill in the configuration parameters in accordance with the previously specified Kerberos server settings.

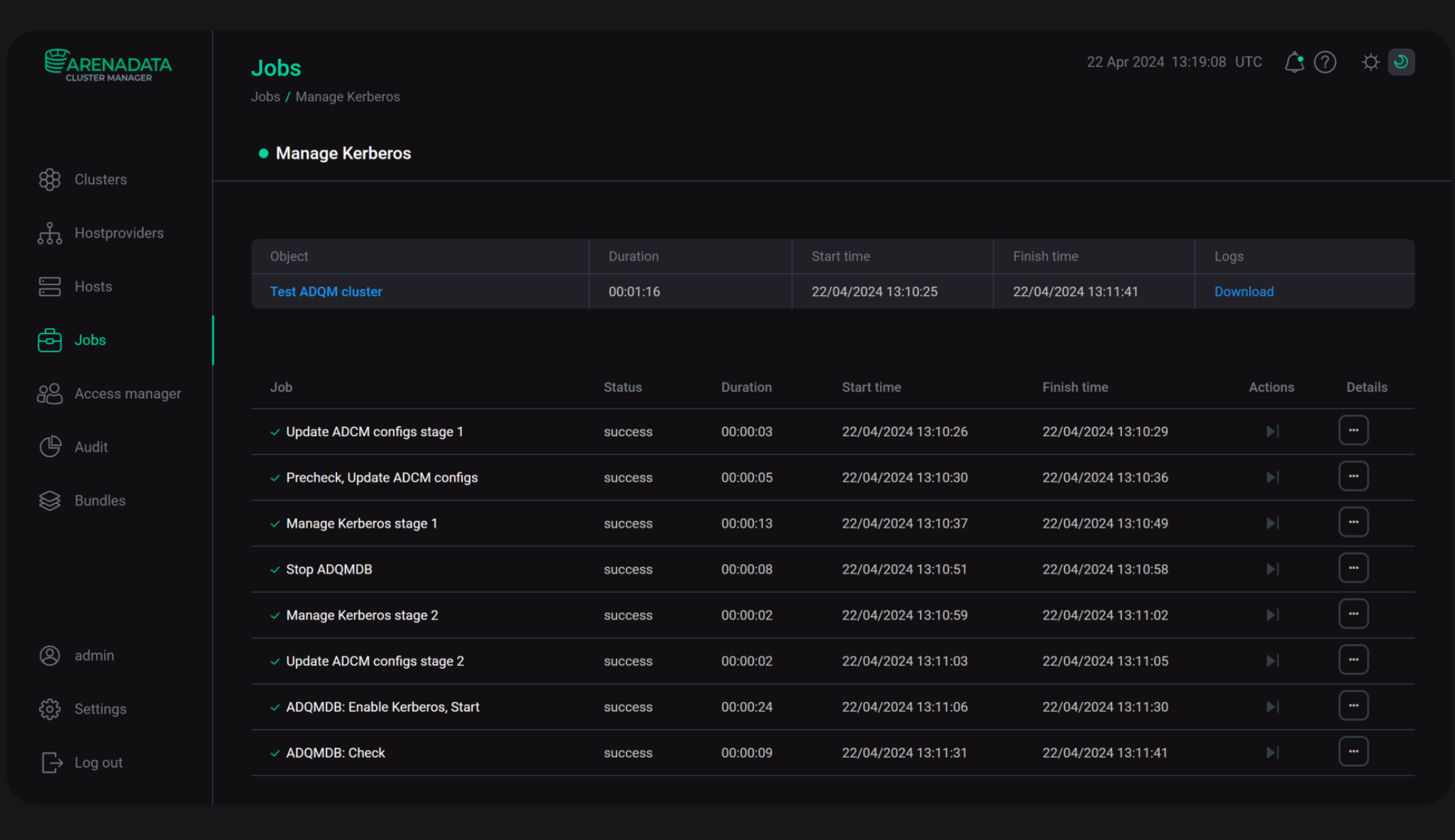


Run an action: Manage Kerberos
Search input
Expand content
Show advanced
Cancel
Run
Configuration
+ 1
Domains
Additional realms <not set>

Existing MIT KDC
Authentication on WEB UIs: false
KDC hosts
KDC hosts [0]: dev-adcm-test.ru-central1.internal
Realm: RU-CENTRAL1.INTERNAL
Kadmin server: dev-adcm-test.ru-central1.internal
Kadmin principal: admin/admin
Kadmin password: <secret>
Keytabs directory: /etc/security/keytabs
Trusted Active Directory server: <not set>
Trusted Active Directory realm: <not set>
Existing Active Directory

MIT KDC parameters

4. Click ***Run*** and wait for the cluster kerberization to complete. On the ***Jobs*** page, you can monitor the progress and result of the ***Manage Kerberos*** task execution.



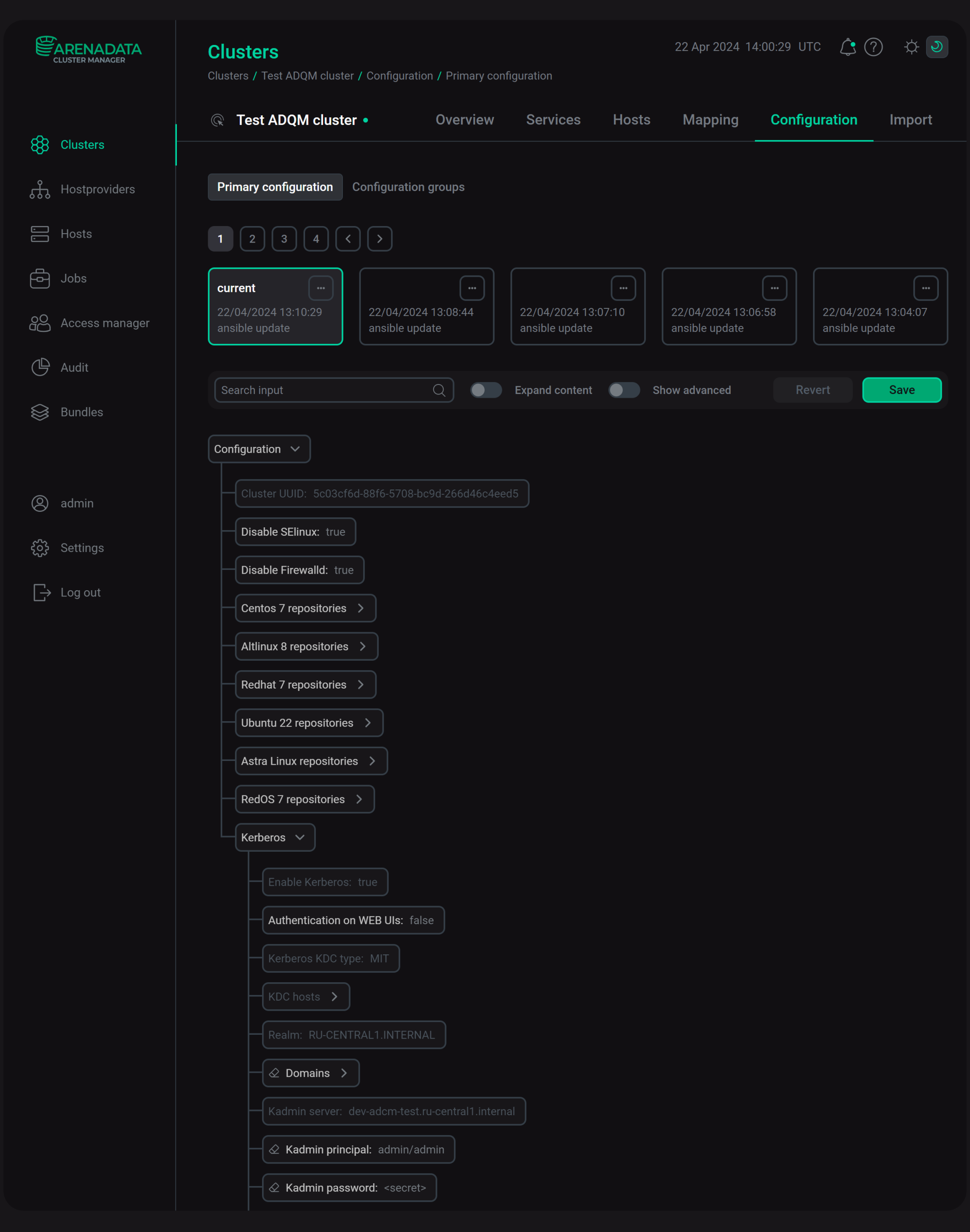
ARENADATA CLUSTER MANAGER
Jobs
Jobs / Manage Kerberos
22 Apr 2024 13:19:08 UTC
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Manage Kerberos

Object	Duration	Start time	Finish time	Logs
Test ADQM cluster	00:01:16	22/04/2024 13:10:25	22/04/2024 13:11:41	Download

Job	Status	Duration	Start time	Finish time	Actions	Details
Update ADCM configs stage 1	success	00:00:03	22/04/2024 13:10:26	22/04/2024 13:10:29		
Precheck, Update ADCM configs	success	00:00:05	22/04/2024 13:10:30	22/04/2024 13:10:36		
Manage Kerberos stage 1	success	00:00:13	22/04/2024 13:10:37	22/04/2024 13:10:49		
Stop ADQMDB	success	00:00:08	22/04/2024 13:10:51	22/04/2024 13:10:58		
Manage Kerberos stage 2	success	00:00:02	22/04/2024 13:10:59	22/04/2024 13:11:02		
Update ADCM configs stage 2	success	00:00:02	22/04/2024 13:11:03	22/04/2024 13:11:05		
ADQMDB: Enable Kerberos, Start	success	00:00:24	22/04/2024 13:11:06	22/04/2024 13:11:30		
ADQMDB: Check	success	00:00:09	22/04/2024 13:11:31	22/04/2024 13:11:41		

Kerberization process

After Kerberos is successfully installed on the ADQM cluster, the corresponding settings appear on the cluster configuration page (in the ***Kerberos*** section) and on the ADQMDB service configuration page (in the ***Kerberos configuration*** section).

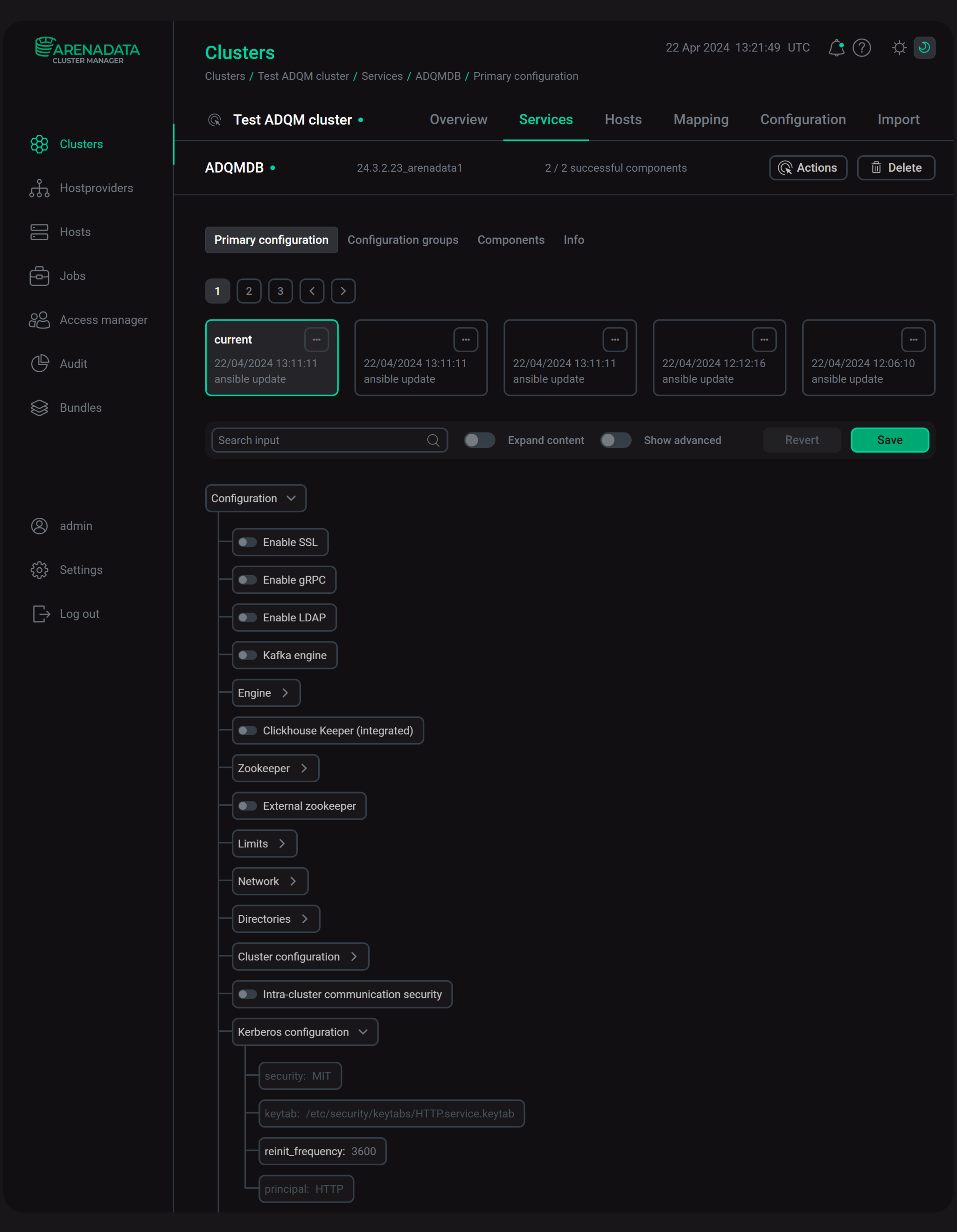


ARENADATA CLUSTER MANAGER
Clusters
Clusters / Test ADQM cluster / Configuration / Primary configuration
22 Apr 2024 14:00:29 UTC
Test ADQM cluster
Overview
Services
Hosts
Mapping
Configuration
Import
Primary configuration
Configuration groups
1
2
3
4
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Search input
Expand content
Show advanced
Revert
Save
Configuration
Centos 7 repositories
Altlinux 8 repositories
Redhat 7 repositories
Ubuntu 22 repositories
Astra Linux repositories
RedOS 7 repositories
Kerberos
KDC hosts
Domains

current 22/04/2024 13:10:29 ansible update
22/04/2024 13:08:44 ansible update
22/04/2024 13:07:10 ansible update
22/04/2024 13:06:58 ansible update
22/04/2024 13:04:07 ansible update

Cluster UUID: 5c03cf6d-88f6-5708-bc9d-266d46c4eed5
Disable SElinux: true
Disable Firewalld: true
Enable Kerberos: true
Authentication on WEB UIs: false
Kerberos KDC type: MIT
Realm: RU-CENTRAL1.INTERNAL
Kadmin server: dev-adcm-test.ru-central1.internal
Kadmin principal: admin/admin
Kadmin password: <secret>

Kerberos settings in the ADQM cluster configuration



ARENADATA CLUSTER MANAGER
Clusters
Clusters / Test ADQM cluster / Services / ADQMDB / Primary configuration
22 Apr 2024 13:21:49 UTC
Test ADQM cluster
Overview
Services
Hosts
Mapping
Configuration
Import
Actions
Delete
Primary configuration
Configuration groups
Components
Info
1
2
3
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Search input
Expand content
Show advanced
Revert
Save
Configuration
Enable SSL
Enable gRPC
Enable LDAP
Kafka engine
Engine
Clickhouse Keeper (integrated)
Zookeeper
External zookeeper
Limits
Network
Directories
Cluster configuration
Intra-cluster communication security
Kerberos configuration

ADQMDB 24.3.2.23_arenadata1 2 / 2 successful components

current 22/04/2024 13:11:11 ansible update
22/04/2024 13:11:11 ansible update
22/04/2024 13:11:11 ansible update
22/04/2024 12:12:16 ansible update
22/04/2024 12:06:10 ansible update

security: MIT
keytab: /etc/security/keytabs/HTTP.service.keytab
reinit_frequency: 3600
principal: HTTP

Kerberos settings in the ADQMDB service configuration

Test user authentication via Kerberos

After a successful kerberization, ADQM commands can only run on cluster nodes after getting a Kerberos ticket. Below is an example that demonstrates how to access the kerberized cluster:

1. Create an ADQM user that has a principal in the Kerberos database. To do this, use the `IDENTIFIED WITH kerberos` clause in the `CREATE USER` query:

```
CREATE USER adqm_kerb IDENTIFIED WITH kerberos;
```

2. Create a Kerberos ticket for the user principal:

```
$ kinit -V adqm_kerb@RU-CENTRAL1.INTERNAL
```

   Enter a password:

```
Password for adqm_kerb@RU-CENTRAL1.INTERNAL:
```

   The kinit article of the MIT Kerberos documentation describes all available options of the `kinit` command.

3. Check the ticket:

```
$ klist
```

```
Ticket cache: FILE:/tmp/krb5cc_1000
Default principal: adqm_kerb@RU-CENTRAL1.INTERNAL

Valid starting       Expires              Service principal
05/04/2023 07:13:58  05/05/2023 07:13:58  krbtgt/RU-CENTRAL1.INTERNAL@RU-CENTRAL1.INTERNAL
```

4. To test authentication via Kerberos, send some command to ADQM using the `curl` utility version that supports the SPNEGO mechanism:

```
$ echo "select currentUser()" | curl --negotiate -u : http://<host_name>:8123/ --data-binary @-
```

   In case of successful authentication, the command should return the kerberized user name:

```
adqm_kerb
```

Use Active Directory for Kerberos

Elena Dvoryadkina

Contents

To Table of Contents

Prepare Active Directory to run on an ADQM cluster
Start Active Directory on ADQM cluster
Active Directory parameters
Test user authentication via Kerberos

Basic concepts used in Kerberos LDAP:

- **Active Directory** — database and set of services that connect users to the necessary network resources.
- **LDAP server** — hierarchical database, a directory service based on Active Directory, used for centralized storage of accounts. The credentials include principals used to validate users in Kerberos.
- **LDAP** — application protocol for accessing the Active Directory directory service, used in cluster kerberization to establish user principals.
- **LDAPS** — LDAP activation using the SSL/TLS protocol to create secure traffic. Creating passwords for ADQM users with accounts in Active Directory during cluster kerberization is only possible using LDAPS.
- **Key distribution center** (**KDC**) — access control system component that provides access tickets and session keys to service user requests for access to resources. It uses Active Directory as the account database.
- **DN (distinguished name)** — account in Active Directory. A DN must be unique within the tree. It describes the content of attributes in the tree (navigation path) for accessing a particular entry.
  A DN consists of a series of **RDN (Relative Distinguished Names)** determined by moving up the tree in the direction of its root entry. RDNs are written from left to right.

> An example of user DN used in this article is *cn=ad_admin,ou=users,dc=example,dc=com*.
>
> The following RDNs are assigned to this entry:
>
> - cn — common name.
> - ou — organizational unit, a container in an Active Directory domain that can contain users, groups, and computers. An organizational unit can have multiple ou within it.
> - dc — parts of the domain name (domain component) representing the top of the LDAP tree, which uses DNS to define its namespace. The notation for an Active Directory domain with the `EXAMPLE.COM` DNS name is `dc=example,dc=com`. The `EXAMPLE.COM` name is also assigned to the realm that is being authenticated.

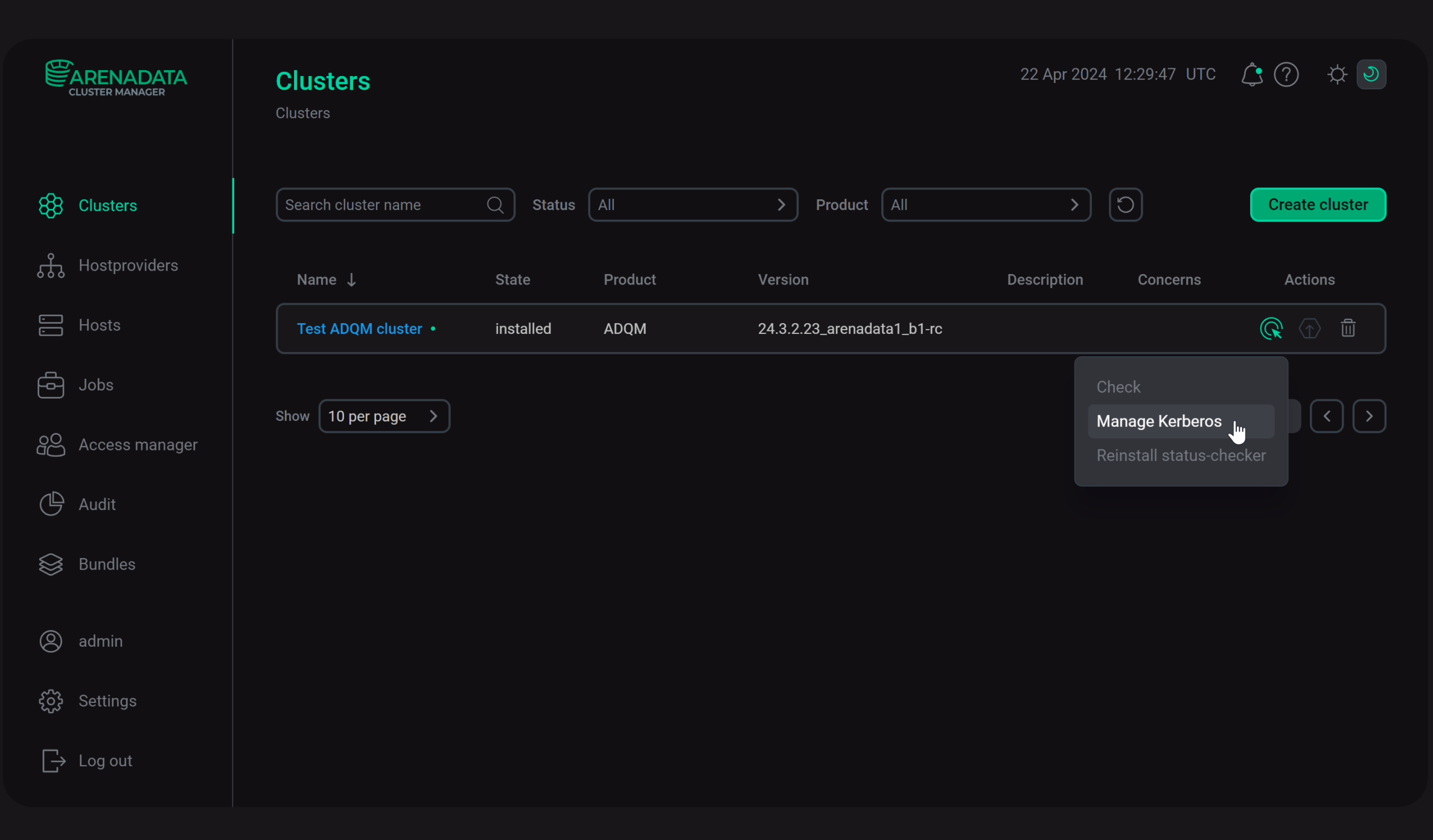
Prepare Active Directory to run on an ADQM cluster

1. Set up an Active Directory server (LDAP server).
2. Create a user account to connect a cluster to the LDAP server using the LDAP protocol.
3. Generate a certificate for the user by enabling LDAP over SSL using a third-party CA.
4. Set a password for the account.
5. Assign a user principal name to the account (for example, `ad_admin@EXAMPLE.COM`) and generate a *keytab* file with these principal and keys.

Start Active Directory on ADQM cluster

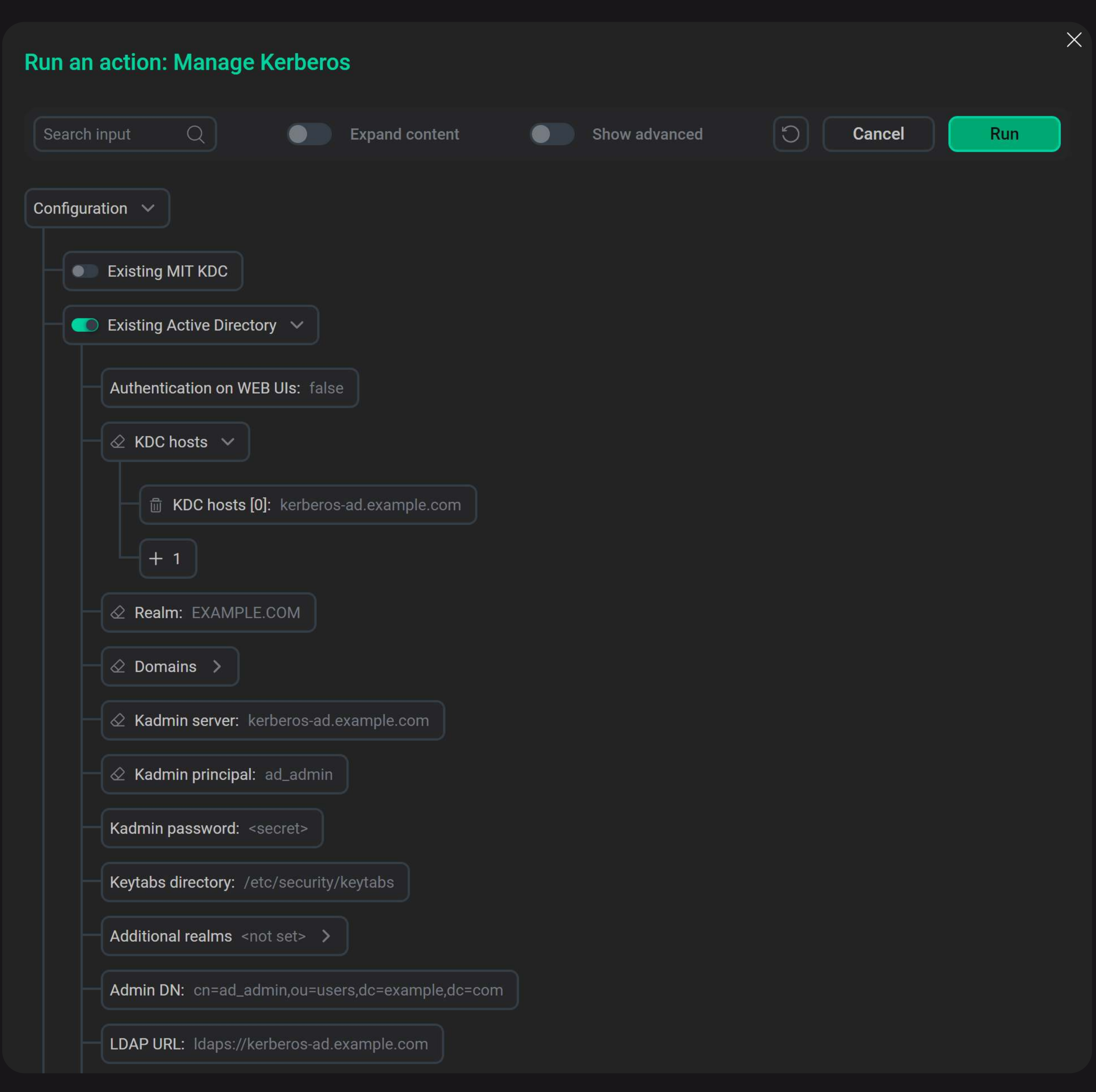
To kerberize a cluster using Active Directory, follow the steps below:

1. On the ***Clusters*** page of the ADCM interface, select the installed and prepared ADQM cluster.
2. Enable Kerberos for the cluster — click the ***Actions*** icon and select the ***Manage Kerberos*** action.


ARENADATA CLUSTER MANAGER
Clusters
22 Apr 2024 12:29:47 UTC
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Search cluster name
Status
All
Product
All
Create cluster
Name
State
Product
Version
Description
Concerns
Actions
Test ADQM cluster
installed
ADQM
24.3.2.23_arenadata1_b1-rc
Show
10 per page
Check
Manage Kerberos
Reinstall status-checker

Manage Kerberos action

3. Turn on the ***Existing Active Directory*** option and set Active Directory parameters according to settings specified when configuring the LDAP server.

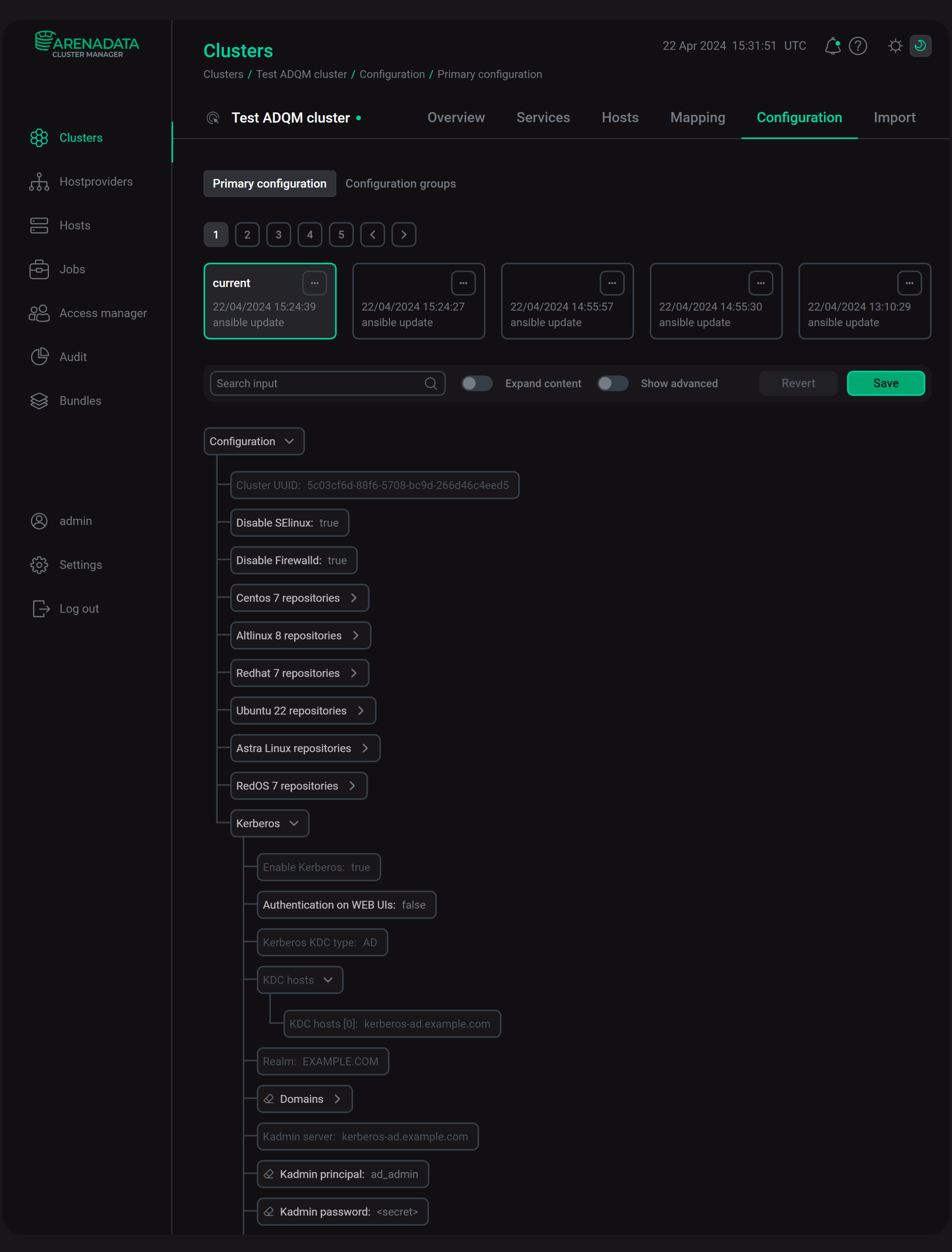

Run an action: Manage Kerberos
Search input
Expand content
Show advanced
Cancel
Run
Configuration
Existing MIT KDC
Existing Active Directory
KDC hosts
+ 1
Domains

Authentication on WEB UIs: false
KDC hosts [0]: kerberos-ad.example.com
Realm: EXAMPLE.COM
Kadmin server: kerberos-ad.example.com
Kadmin principal: ad_admin
Kadmin password: <secret>
Keytabs directory: /etc/security/keytabs
Additional realms <not set>
Admin DN: cn=ad_admin,ou=users,dc=example,dc=com
LDAP URL: ldaps://kerberos-ad.example.com

Active Directory parameters

4. Click ***Run*** and wait for the cluster kerberization to complete. On the ***Jobs*** page, you can monitor the progress and result of the ***Manage Kerberos*** task execution.

After Kerberos is successfully installed on the ADQM cluster, the corresponding settings appear on the cluster configuration page (in the ***Kerberos*** section) and on the ADQMDB service configuration page (in the ***Kerberos configuration*** section).


ARENADATA CLUSTER MANAGER
Clusters
Clusters / Test ADQM cluster / Configuration / Primary configuration
22 Apr 2024 15:31:51 UTC
Test ADQM cluster
Overview
Services
Hosts
Mapping
Configuration
Import
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Primary configuration
Configuration groups
1
2
3
4
5
current
22/04/2024 15:24:39
ansible update
22/04/2024 15:24:27
ansible update
22/04/2024 14:55:57
ansible update
22/04/2024 14:55:30
ansible update
22/04/2024 13:10:29
ansible update
Search input
Expand content
Show advanced
Revert
Save
Configuration
Centos 7 repositories
Altlinux 8 repositories
Redhat 7 repositories
Ubuntu 22 repositories
Astra Linux repositories
RedOS 7 repositories
Kerberos
KDC hosts
Domains

Cluster UUID: 5c03cf6d-88f6-5708-bc9d-266d46c4eed5
Disable SElinux: true
Disable Firewalld: true
Enable Kerberos: true
Authentication on WEB UIs: false
Kerberos KDC type: AD
KDC hosts [0]: kerberos-ad.example.com
Realm: EXAMPLE.COM
Kadmin server: kerberos-ad.example.com
Kadmin principal: ad_admin
Kadmin password: <secret>

Kerberos settings in the ADQM cluster configuration

Active Directory parameters

The following parameters are required to kerberize an ADQM cluster with Active Directory.

Parameter	Description
KDC hosts	Address of the host where KDC is installed (for kerberization with LDAP, this address is the same as the address of the LDAP server)
Realm	Realm for authentication in Kerberos
Kadmin server	Address of the server on which the Kerberos administration system is installed (this address is often the same as the address of the KDC server)
Kadmin principal	Principal name of the user created to connect a cluster to Active Directory with rights to create users and groups in the ***Container DN*** container
Kadmin password	Password for the ***Kadmin principal*** user principal
Admin DN	DN of the user used to connect a cluster to Active Directory
LDAP URL	LDAP server address (`ldaps://<hostname or IP address>:<port>`)
Container DN	DN of a container where service principals are located
TLS CA certificate Path (optional)	Local path where the CA certificate for the ***Admin DN*** user is stored on all cluster hosts (if the certificate is up-to-date, the ***TLS CA certificate*** parameter does not need to be specified)
TLS CA certificate (optional)	Contents of the CA certificate for the ***Admin DN*** user that will be written to a host in the path specified in ***TLS CA certificate Path***

Test user authentication via Kerberos

After a successful kerberization, ADQM commands can only run on cluster nodes after getting a Kerberos ticket. Below is an example that demonstrates how to access the cluster kerberized with Active Directory:

1. Create an ADQM user that has an entry in the Active Directory database and a principal for the specified realm. To do this, use the `IDENTIFIED WITH kerberos` clause in the `CREATE USER` query:

```
CREATE USER adqm_kerb_ad IDENTIFIED WITH kerberos;
```

2. Create a Kerberos ticket for the user principal:

```
$ kinit -V adqm_kerb_ad@EXAMPLE.COM
```

Enter a password:

```
Password for adqm_kerb_ad@EXAMPLE.COM:
```

The kinit article of the MIT Kerberos documentation describes all available options of the `kinit` command.

3. Check the ticket:

```
$ klist
```

```
Ticket cache: FILE:/tmp/krb5cc_1000
Default principal: adqm_kerb_ad@EXAMPLE.COM

Valid starting       Expires              Service principal
05/05/2023 09:05:36  05/06/2023 09:05:36  krbtgt/EXAMPLE.COM@EXAMPLE.COM
```

4. To test authentication via Kerberos, send some command to ADQM using the `curl` utility version that supports the SPNEGO mechanism:

```
$ echo "select currentUser()" | curl --negotiate -u : http://<host_name>:8123/ --data-binary @-
```

In case of successful authentication, the command should return the kerberized user name:

```
adqm_kerb_ad
```

Kerberization issue with unsupported encryption type on RED OS

Elena Dvoryadkina

Contents

To Table of Contents

Cause
Solution — change cryptographic policy
  LEGACY
  CUSTOM

When you install ADQM on a RED OS host and try to kerberize a cluster using Active Directory, kerberization may fail during the `kinit` command execution with the following error:

```
kinit: KDC has no support for encryption type while getting initial credentials
```

Cause

The issue occurs due to different sets of encryption types allowed in Active Directory and RED OS. In RED OS, this set is defined by a cryptographic policy.

By default, the RED OS cryptographic policy is set to `DEFAULT`:

```
$ update-crypto-policies --show
```

```
DEFAULT
```

This policy sets the following encryption types that the Kerberos protocol is allowed to use:

```
$ cat /usr/share/crypto-policies/DEFAULT/krb5.txt
```

```
[libdefaults]
permitted_enctypes = aes256-cts-hmac-sha1-96 aes256-cts-hmac-sha384-192 camellia256-cts-cmac aes128-cts-hmac-sha1-96 aes128-cts-hmac-sha256-128 camellia128-cts-cmac
```

The `arcfour` encryption type (also known as `rc4`) is not in the list, but it is required for requesting Kerberos tickets from an Active Directory server. This encryption type has been excluded from the `DEFAULT` policy as it is considered weak (see Ensuring support for common encryption types in AD and RHEL).

Solution — change cryptographic policy

To enable support for `arcfour` in RED OS and resolve the issue above, it is recommended to switch the default RED OS cryptographic policy to one of the following policies.

LEGACY

The `LEGACY` cryptographic policy is one of RED OS's predefined cryptographic policies that supports a slightly larger list of encryption types than `DEFAULT` and provides the compatibility with older versions and systems.

Encryption types for Kerberos that the `LEGACY` policy supports are:

```
$ cat /usr/share/crypto-policies/LEGACY/krb5.txt
```

```
[libdefaults]
permitted_enctypes = aes256-cts-hmac-sha1-96 aes256-cts-hmac-sha384-192 camellia256-cts-cmac aes128-cts-hmac-sha1-96 aes128-cts-hmac-sha256-128 camellia128-cts-cmac des3-cbc-sha1 arcfour-hmac-md5
```

To switch the cryptographic policy to `LEGACY`, run the following command as the `root` user:

```
$ update-crypto-policies --set LEGACY
```

After switching the cryptographic policy, the system reboot is required:

```
Setting system policy to LEGACY
Note: System-wide crypto policies are applied on application start-up.
It is recommended to restart the system for the change of policies to fully take place.
```

Consider also that the set of permitted encryption types will be changed for other services, such as:

```
$ ls /usr/share/crypto-policies/LEGACY/
```

```
bind.txt gnutls.txt java.txt krb5.txt libreswan.txt nss.txt opensshserver.txt openssh.txt opensslcnf.txt openssl.txt
```

CUSTOM

You can create a new cryptographic policy (for example, `CUSTOM`) based on the `DEFAULT` policy and customize the list of supported encryption types.

1. Copy the `DEFAULT` policy directory to the `CUSTOM` directory:

```
$ cp -r /usr/share/crypto-policies/DEFAULT/ /usr/share/crypto-policies/CUSTOM
```

2. Add the `arcfour-hmac-md5` type to the end of the */usr/share/crypto-policies/CUSTOM/krb5.txt* file.
3. Set a new cryptographic policy:

```
$ update-crypto-policies --set CUSTOM
```

4. After switching the policy to `CUSTOM`, it is also required to restart the system:

```
Setting system policy to CUSTOM
Note: System-wide crypto policies are applied on application start-up.
It is recommended to restart the system for the change of policies to fully take place.
```

Use FreeIPA for Kerberos

Elena Dvoryadkina

Contents

To Table of Contents

Start FreeIPA on ADQM cluster

FreeIPA Kerberos parameters

Check FreeIPA integration

FreeIPA is a free open source identity management system for Linux/UNIX environments. FreeIPA relies on Linux (Fedora), 389 Directory Server, MIT Kerberos, NTP, DNS, Dogtag (certificate system) and its main purpose is to provide centralized account management and authentication.

To ensure security, ADQM uses Kerberos, LDAP, SSL certificates/keys, and other related technologies. For such set of tools, FreeIPA is a good fit as an identity management system.

To enable FreeIPA for an ADQM cluster, use ADCM. During kerberization, ADCM installs IPA clients on each cluster node and connects clients to the FreeIPA server using the provided FreeIPA Kerberos parameters.

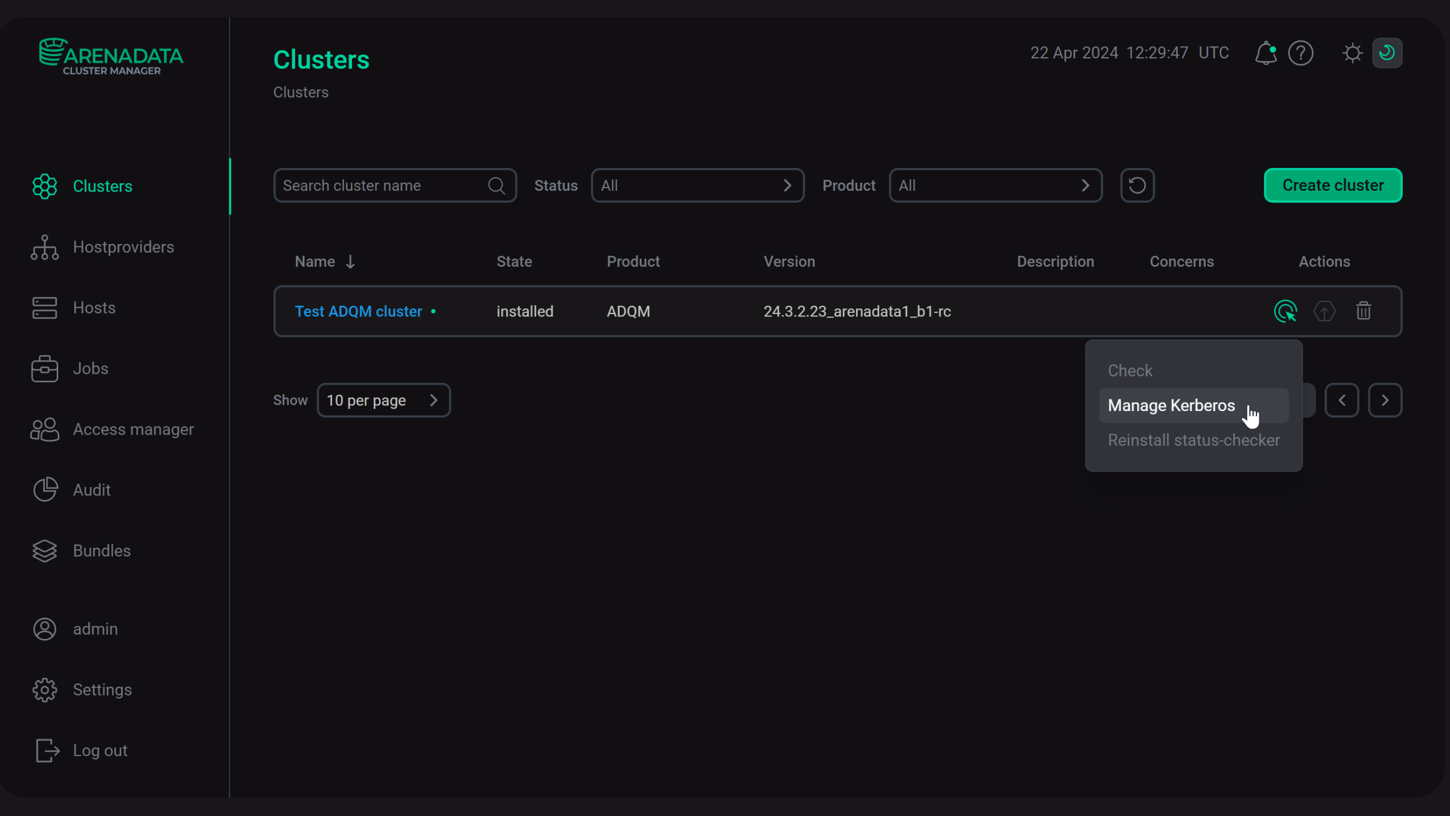
In this tutorial, a sample single-noded FreeIPA server is used to demonstrate an example. It is deployed on the *freeipa-server.ru-central1.internal* host with the CentOS operating system. This is a basic FreeIPA installation (`yum install ipa-server`) with default installation parameters.

Start FreeIPA on ADQM cluster

To kerberize a cluster using FreeIPA, follow the steps below:

1. On the ***Clusters*** page of the ADCM interface, select the installed and prepared ADQM cluster.

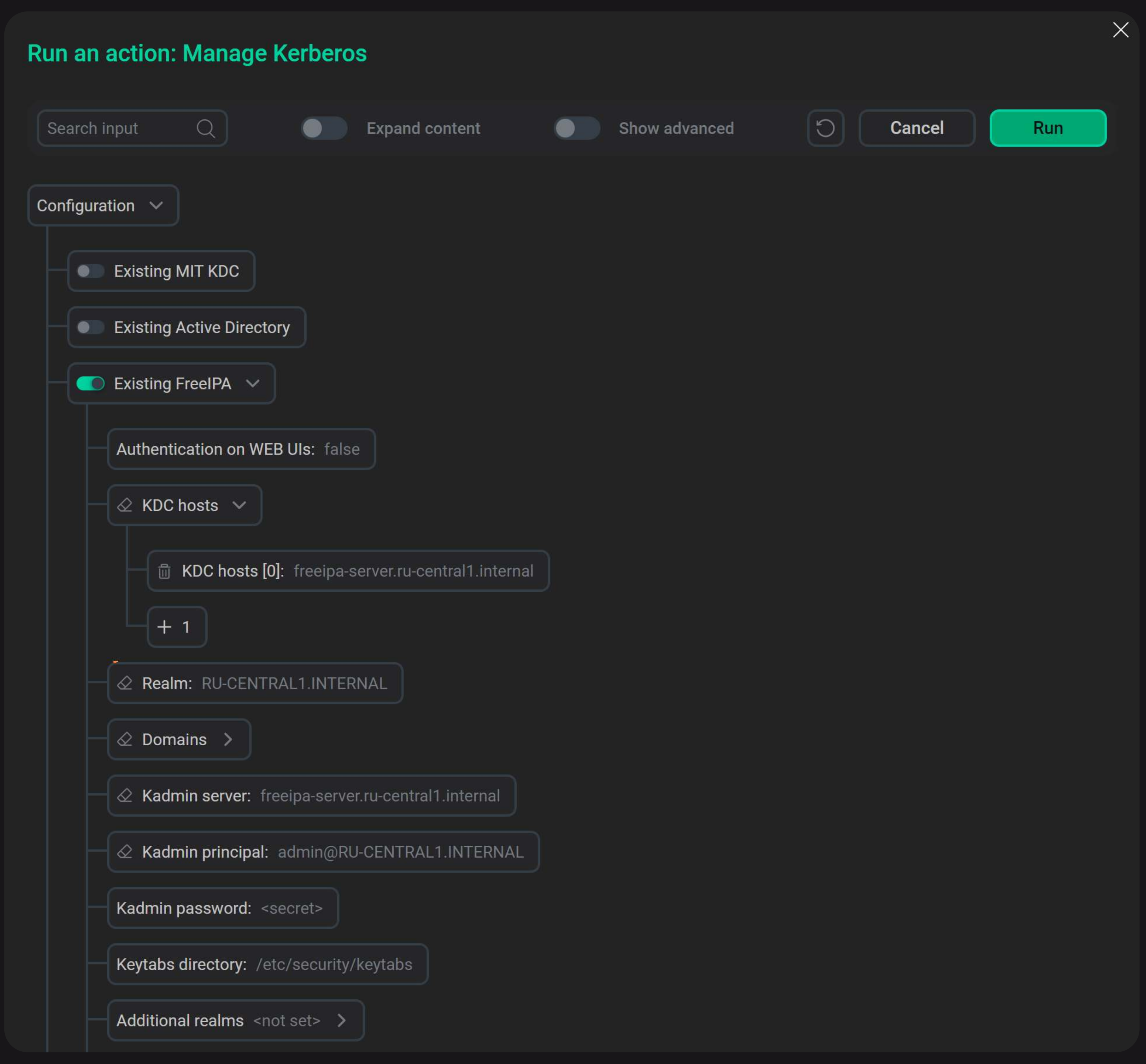
2. Enable Kerberos for the cluster — click the ***Actions*** icon and select the ***Manage Kerberos*** action.



Name	State	Product	Version	Description	Concerns	Actions
Test ADQM cluster	installed	ADQM	24.3.2.23_arenadata1_b1-rc			

Manage Kerberos action

3. Turn on the ***Existing FreeIPA*** option and specify FreeIPA Kerberos parameters.



Authentication on WEB UIs: false

KDC hosts [0]: freeipa-server.ru-central1.internal

Realm: RU-CENTRAL1.INTERNAL

Kadmin server: freeipa-server.ru-central1.internal

Kadmin principal: admin@RU-CENTRAL1.INTERNAL

Kadmin password: <secret>

Keytabs directory: /etc/security/keytabs

Additional realms <not set>

FreeIPA parameters

4. Click ***Run*** and wait for the cluster kerberization to complete. On the ***Jobs*** page, you can monitor the progress and result of the ***Manage Kerberos*** task execution.

After Kerberos is successfully installed on the ADQM cluster, the corresponding settings appear on the cluster configuration page (in the ***Kerberos*** section) and on the ADQMDB service configuration page (in the ***Kerberos configuration*** section).

FreeIPA Kerberos parameters

The following parameters are required to kerberize an ADQM cluster with FreeIPA. To obtain actual values for these parameters, you can run `ipa user-find <ipa_admin>` on your FreeIPA server, where `<ipa_admin>` is the IPA Admin username.

Parameter	Description
KDC hosts	One or more KDC hosts with running FreeIPA server(s)
Realm	Kerberos realm to connect to the FreeIPA server
Domains	One or more domains associated with FreeIPA
Kadmin server	Host where `kadmin` is running
Kadmin principal	Principal name used to connect via `kadmin`, for example `admin@RU-CENTRAL1.INTERNAL`
Kadmin password	IPA Admin password
IpaClient No NTP Autoconf	Disables the NTP configuration during the IPA client installation
IpaClient No DNS Lookup	Disables the DNS lookup for FreeIPA server during the IPA client installation

Check FreeIPA integration

After a successful kerberization, ADQM commands can only run on cluster nodes after getting a Kerberos ticket. Below is an example that demonstrates how to access the cluster kerberized with FreeIPA:

1. Create an ADQM user that matches an account existing in the FreeIPA domain. To do this, use the `IDENTIFIED WITH kerberos` clause in the `CREATE USER` query:

```
CREATE USER adqm_kerb_freeipa IDENTIFIED WITH kerberos;
```

2. Create a Kerberos ticket for the user principal:

```
$ kinit adqm_kerb_freeipa@RU-CENTRAL1.INTERNAL
```

   Enter a password:

```
Password for adqm_kerb_freeipa@RU-CENTRAL1.INTERNAL:
```

   The kinit article of the MIT Kerberos documentation describes all available options of the `kinit` command.

3. Check the ticket:

```
$ klist
```

```
Ticket cache: FILE:/tmp/krb5cc_1000
Default principal: adqm_kerb_freeipa@RU-CENTRAL1.INTERNAL

Valid starting       Expires              Service principal
05/10/2023 10:20:30  05/11/2023 10:20:30  krbtgt/RU-CENTRAL1.INTERNAL@RU-CENTRAL1.INTERNAL
```

4. To test authentication via Kerberos, send some command to ADQM using the `curl` utility version that supports the SPNEGO mechanism:

```
$ echo "select currentUser()" | curl --negotiate -u : http://<host_name>:8123/ --data-binary @-
```

   In case of successful authentication, the command should return the kerberized user name:

```
adqm_kerb_freeipa
```

Authorization

Elena Dvoryadkina

Contents

To Table of Contents

Configure authorization for a new cluster

User account
- Create a user account
- Change an existing user account
- View user account settings
- Manage user privileges
- Delete a user account

Roles
- Manage roles
- Specify role privileges
- Assign a role to a user
- Apply a user role

Row policy
- Create a row policy
- Modify a row policy
- View row policy settings
- Delete row policy

Settings profile
- Create a settings profile
- Modify a settings profile
- View parameters of a settings profile
- Delete a settings profile

Quota
- Create a quota
- Modify a quota
- View information on quotas
- Delete a quota

The ADQM database access control system based on the Role-based access control (RBAC) concept includes the following objects:

- User account
- Role
- Row policy
- Settings profile
- Quota

You can create and configure these objects using SQL queries (recommended) or configuration files (*users.xml* and *config.xml*). This article explains the basics of SQL-driven access control and account management. It describes syntax and provides examples of queries to create ADQM users and roles with specific permissions and restrictions for working with databases, tables, columns, and data rows.

Create a test table and database to run the examples below

To reproduce the examples below, first create a sample table (`test_table`) in the current database and a new database (`test_database`):

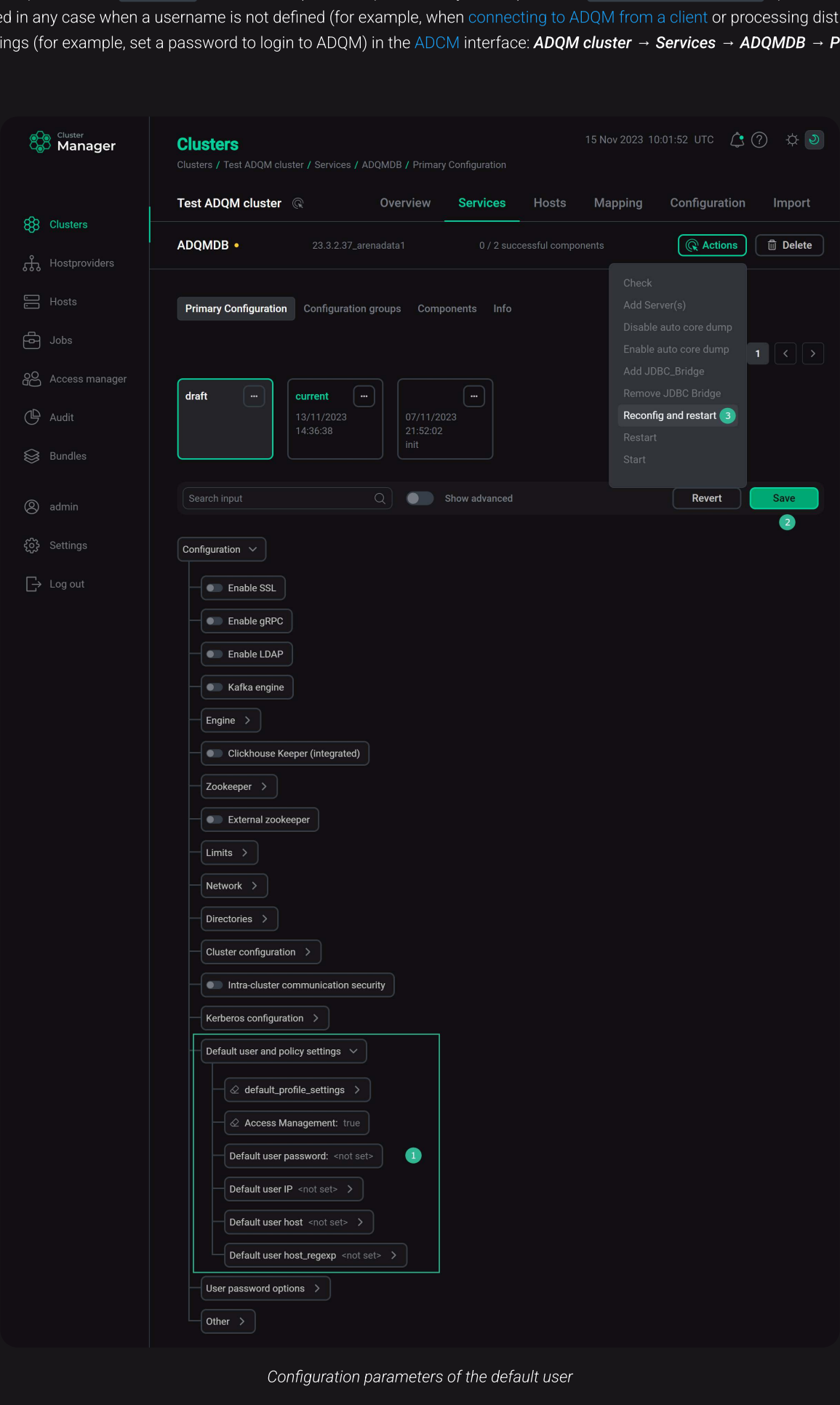
```
CREATE TABLE test_table (x Int32, y String, z Int32) ENGINE = MergeTree ORDER BY x;
```

```
CREATE DATABASE test_database;
```

Configure authorization for a new cluster

The SQL access control mode is enabled if a user's access_management parameter value is set to `1` in the *users.xml* file.

When creating a new cluster, ADQM provides the `default` user account (without a password by default) with the `access_management` option enabled and all access rights and permissions. This account is used in any case when a username is not defined (for example, when connecting to ADQM from a client or processing distributed queries). You can change the `default` user settings (for example, set a password to login to ADQM) in the ADCM interface: ***ADQM cluster → Services → ADQMDB → Primary Configuration → Default user and policy settings***.



Configuration parameters of the default user

Use the `default` user account to set up secure access to the ADQM database right after creating a cluster. To do this, follow these steps:

1. Connect to ADQM as the `default` user (for example, via the clickhouse-client command-line client).
2. Create user accounts required for further work in ADQM, including administrator account.
3. Restrict permissions for the `default` user and disable SQL-based access control and account management for it (the ***Access Management*** option in the ADCM interface).

User account

A **user account** (user) is an access system object used to authorize a user connecting to ADQM. This object contains the following information:

- user identification information;
- privileges that define which queries the user is allowed to execute;
- hosts allowed to connect to ADQM servers;
- roles assigned to the user and default roles;
- settings applied when the user logs in the system.

Create a user account

To create a new user account, use the `CREATE USER` query:

```
CREATE USER [IF NOT EXISTS | OR REPLACE] <user_name1> [ON CLUSTER <cluster_name1>]
    [, <user_name2> [ON CLUSTER <cluster_name2>] ...]
    [NOT IDENTIFIED | IDENTIFIED {[WITH {no_password | plaintext_password | sha256_password | sha256_hash | double_sha1_password |
double_sha1_hash}] BY {'password' | 'hash'}} | {WITH ldap SERVER 'ldap_server'} | {WITH ssl_certificate CN '<certificate_CN>'} |
{WITH kerberos [REALM 'realm']}]
    [HOST {LOCAL | NAME '<fqdn>' | REGEXP '<name_regexp>' | IP '<ip_address>' | LIKE '<pattern>'} [,...] | ANY | NONE]
    [DEFAULT ROLE <role_name> [,...] | ALL | ALL EXCEPT <role_name> [,...] ]
    [DEFAULT DATABASE <database_name> | NONE]
    [GRANTEES {<user_name> | <role_name> | ANY | NONE} [,...] [EXCEPT {<user_name> | <role_name>} [,...]]]
    [SETTINGS <setting_name> [= <value>] [MIN [=] <min_value>] [MAX [=] <max_value>] [READONLY | WRITABLE] | PROFILE
'<profile_name>'] [,...];
```

Basic clauses

ON CLUSTER	By default, the created user can connect only to the server on which the `CREATE USER` query is run. To create a user that can connect to any server of the cluster, use the `ON CLUSTER` clause with the specified cluster name
NOT IDENTIFIED | IDENTIFIED	User authentication method: by a password, via an LDAP server, by an SSL certificate or by the Kerberos protocol
HOST	Host from which the user can connect to an ADQM server
DEFAULT ROLE	Default role that is automatically activated when the user logs in ADQM
DEFAULT DATABASE	Database that will be a current database when the user logs in ADQM
GRANTEES	Users or roles that are allowed to receive privileges from the user being created, if this user is granted the corresponding permission via WITH GRANT OPTION
SETTINGS	Values of settings and their restrictions that are applied when the user logs in. In addition to settings explicitly defined via the `SETTINGS` clause, settings of roles assigned to the user and settings from settings profiles (`CREATE USER <user_name> SETTINGS PROFILE <profile_name>`) are also applied

Example

Create the `john` user that can connect to any server of the `default_cluster` cluster using the `qwerty` password:

```
CREATE USER john ON CLUSTER default_cluster IDENTIFIED WITH sha256_password BY 'qwerty';
```

Change an existing user account

You can change settings of an existing user with the `ALTER USER` query:

```
ALTER USER [IF EXISTS] <user_name1> [ON CLUSTER <cluster_name1>] [RENAME TO <new_user_name1>]
    [, <user_name2> [ON CLUSTER <cluster_name2>] [RENAME TO <new_user_name2>] ...]
    [NOT IDENTIFIED | IDENTIFIED {[WITH {no_password | plaintext_password | sha256_password | sha256_hash | double_sha1_password |
double_sha1_hash}] BY {'password' | 'hash'}} | {WITH ldap SERVER 'ldap_server'} | {WITH ssl_certificate CN '<certificate_CN>'} |
{WITH kerberos [REALM 'realm']}]
    [[ADD | DROP] HOST {LOCAL | NAME '<fqdn>' | REGEXP '<name_regexp>' | IP '<ip_address>' | LIKE '<pattern>'} [,...] | ANY | NONE]
    [DEFAULT ROLE <role_name> [,...] | ALL | ALL EXCEPT <role_name> [,...] ]
    [GRANTEES {<user_name> | <role_name> | ANY | NONE} [,...] [EXCEPT {<user_name> | <role_name>} [,...]]]
    [SETTINGS <setting_name> [= <value>] [MIN [=] <min_value>] [MAX [=] <max_value>] [READONLY | WRITABLE] | PROFILE
'<profile_name>'] [,...]
```

Example

The following query reconfigures the previously created user account so that it can be used for connection from the specified IP address only:

```
ALTER USER john HOST IP '10.92.17.140';
```

View user account settings

To view current settings of a user account (parameters specified when creating the user and all subsequent modifications made via `ALTER USER`) run the `SHOW CREATE USER` query:

```
SHOW CREATE USER [<user_name1> [, <user_name2> ...] | CURRENT_USER];
```

Example

Run the following query to see settings of the `john` user:

```
SHOW CREATE USER john;
```

The output:

```
--CREATE USER john--------------------------------------------------------
│ CREATE USER john IDENTIFIED WITH sha256_password HOST IP '10.92.17.140' │
--------------------------------------------------------------------------
```

Manage user privileges

To define access rights for a user, you need to grant **privileges** to the corresponding user account. A privilege is a permission to execute certain types of queries.

Grant privileges

There are two ways to assign privileges to a user account:

- assign a role (a set of privileges) to the user account;
- run the GRANT query to grant privileges to the user account directly:

```
GRANT [ON CLUSTER <cluster_name>] <privilege>[(<column_name> [,...])] [,...]
    ON {<db_name>.<table_name> | <db_name>.* | *.* | <table_name> | *} TO {<user_name> | CURRENT_USER} [,...] [WITH GRANT OPTION]
[WITH REPLACE OPTION];
```

Query parameters and clauses

<privilege>	Privilege type
<column_name>	Name of a column for which the user is allowed to execute queries defined by the privilege
<db_name>.<table_name>	Database name and table name defining the privilege scope
<user_name>	Name of a user account to which the privilege is granted
WITH GRANT OPTION	Allows granting user privileges of the same scope (or less) to other users and roles specified as `GRANTEES` in the CREATE USER / ALTER USER queries
WITH REPLACE OPTION	All old user privileges are replaced with new privileges (if the clause is not used in the query, new privileges are added to the user account)

Examples:

- Allow the `john` user to execute `SELECT` queries for the `x` and `y` columns of the `test_table` table in the current database:

```
GRANT SELECT(x,y) ON test_table TO john;
```

- Create a new `admin` user account and grant it with full administrator rights:

```
CREATE USER admin IDENTIFIED BY 'admin';
```

```
GRANT ALL ON *.* TO admin WITH GRANT OPTION;
```

View user privileges

A set of user privileges includes privileges granted to the user account directly and privileges of assigned roles. Use the SHOW GRANTS query to see a list of user privileges (if no user is specified, the query returnes the current user's privileges):

```
SHOW GRANTS [FOR <user_name>];
```

A sample query:

```
SHOW GRANTS FOR john;
```

The output:

```
--GRANTS FOR john--------------------------------
│ GRANT SELECT(x,y) ON default.test_table TO john │
-------------------------------------------------
```

Revoke user privileges

To revoke privileges from users, use the `REVOKE` query:

```
REVOKE [ON CLUSTER <cluster_name>] <privilege>[(<column_name> [,...])] [,...]
    ON {<db_name>.<table_name> | <db_name>.* | *.* | <table_name> | *}
    FROM {<user_name> | CURRENT_USER} [,...] | ALL | ALL EXCEPT {<user_name> | CURRENT_USER} [,...];
```

For example, the following query revokes the privilege to execute `SELECT` queries for the `y` column of the `test_table` table from the `john` user:

```
REVOKE SELECT(y) ON test_table FROM john;
```

Delete a user account

To delete a user account, run the `DROP USER` query:

```
DROP USER [IF EXISTS] <user_name> [,...] [ON CLUSTER <cluster_name>];
```

For example, to delete the `john` user account on the `default_cluster` cluster, use the query:

```
DROP USER john ON CLUSTER default_cluster;
```

Roles

A **role** is a set of privileges that a user gets if this role is assigned to the user account.

Manage roles

The table below lists queries that you can use to manage ADQM roles.

Query	Description	Syntax
CREATE ROLE	Creates a new role in ADQM	`CREATE ROLE [IF NOT EXISTS | OR REPLACE] <role_name1> [ON CLUSTER <cluster_name1>] [, <role_name2> [ON CLUSTER <cluster_name2>] ...] [SETTINGS <setting_name> [= <value>] [MIN [=] <min_value>] [MAX [=] <max_value>] [CONST|READONLY|WRITABLE|CHANGEABLE_IN_READONLY] | PROFILE '<profile_name>'] [,...];`
ALTER ROLE	Changes settings of an existing role	`ALTER ROLE [IF EXISTS] <role_name1> [ON CLUSTER <cluster_name1>] [RENAME TO <new_role_name1>] [, <role_name2> [ON CLUSTER <cluster_name2>] [RENAME TO <new_role_name2>] ...] [SETTINGS <setting_name> [= <value>] [MIN [=] <min_value>] [MAX [=] <max_value>] [CONST|READONLY|WRITABLE|CHANGEABLE_IN_READONLY] | PROFILE '<profile_name>'] [,...];`
SHOW CREATE ROLE	Displays parameters used to configure a role	`SHOW CREATE ROLE <role_name> [,...];`
DROP ROLE	Removes a role. A removed role is revoked from all objects of the access system to which it was assigned	`DROP ROLE [IF EXISTS] <role_name> [,...] [ON CLUSTER <cluster_name>];`

Specify role privileges

To specify privileges that a role will grant when assigned to a user, use the `GRANT` query:

```
GRANT [ON CLUSTER <cluster_name>] <privilege>[(<column_name> [,...])] [,...] ON {<db_name>.<table_name> | <db_name>.* | *.* |
<table_name> | *}
    TO <role_name> [,...] [WITH GRANT OPTION] [WITH REPLACE OPTION];
```

Query parameters and clauses

<privilege>	Privilege type
<column_name>	Name of a column for which execution of queries defined by the specified privilege is allowed
<db_name>.<table_name>	Database name and table name defining the privilege scope
<role_name>	Name of a role to which the privilege is granted
WITH GRANT OPTION	Allows granting other users with privileges of the same scope (or less)
WITH REPLACE OPTION	All old role privileges are replaced with new privileges (if the clause is not used in the query, new privileges are added to the role)

Example

Create a role and grant it with permission to read all tables in the `test_database` database:

```
CREATE ROLE reader;
```

```
GRANT SELECT ON test_database.* TO reader;
```

Assign a role to a user

You can assign multiple roles to one user account. A role can also be assigned to other roles. Use the following queries to assign and revoke roles:

Query	Description	Syntax
GRANT	Assigns a role to a user or another role. - `WITH ADMIN OPTION` — allows assigning user roles to another user. - `WITH REPLACE OPTION` — all old roles of the user/role are replaced with new roles (if the option is not specified, new roles are added).	`GRANT [ON CLUSTER <cluster_name>] <role_name> [,...] TO {<user_name> | <another_role_name> | CURRENT_USER} [,...] [WITH ADMIN OPTION] [WITH REPLACE OPTION];`
REVOKE	Revokes a role from a user or another role	`REVOKE [ON CLUSTER <cluster_name>] [ADMIN OPTION FOR] <role_name> [,...] FROM {<user_name> | <another_role_name> | CURRENT_USER} [,...] | ALL | ALL EXCEPT {<user_name> | <another_role_name> | CURRENT_USER} [,...];`

Example

Assign the `reader` role to the `john` user:

```
GRANT reader TO john;
```

Make sure the `reader` role has been added to the previously assigned user privilege:

```
SHOW GRANTS FOR john;
```

Query execution result:

```
--GRANTS FOR john--------------------------------
│ GRANT SELECT(x,y) ON default.test_table TO john │
│ GRANT reader TO john                            │
-------------------------------------------------
```

Apply a user role

Users can apply (activate) their assigned roles in arbitrary combinations using the `SET ROLE` and `SET DEFAULT ROLE` queries. The final scope of user privileges is the combination of privileges assigned to the user account directly and privileges granted by the user's active roles.

Query	Description	Syntax
SET ROLE	Activates roles for the current user	`SET ROLE {DEFAULT | NONE | <role_name> [,...] | ALL | ALL EXCEPT <role_name> [,...]};`
SET DEFAULT ROLE	Sets default roles for a user. Default roles are automatically activated when a user logs in ADQM. Only roles assigned to the user can be set as default roles. Another way to set default roles for a user is to use the `DEFAULT ROLE` clause when executing the CREATE USER or ALTER USER query	`SET DEFAULT ROLE {NONE | <role_name> [,...] | ALL | ALL EXCEPT <role_name> [,...]} TO {<user_name>}`

Example

After the `john` user has been granted with the `reader` role (see the example above), he can apply this role and execute queries this role allows:

```
SET ROLE reader;
```

```
SELECT * FROM test_database.*;
```

Row policy

Row policy is a table-specific filter that defines which rows of the table are available to the specified user or role.

Create a row policy

To create a row policy, use the `CREATE ROW POLICY` query:

```
CREATE [ROW] POLICY [IF NOT EXISTS | OR REPLACE] <policy_name1> [ON CLUSTER <cluster_name1>] ON [<db_name1>.]<table_name1>
        [, <policy_name2> [ON CLUSTER <cluster_name2>] ON [<db_name2>.]<table_name2> ...]
    [AS {PERMISSIVE | RESTRICTIVE}]
    [FOR SELECT] USING <filter_condition>
    [TO {<user_name> | <role_name> [,...] | ALL | ALL EXCEPT <user_name> | <role_name> [,...]}];
```

Basic clauses

AS

Defines how to combine policies if more than one policy is active at the same time for the same table and the same user. Possible values:

- `PERMISSIVE` (default) — policies are combined with the `OR` logical operator;
- `RESTRICTIVE` — policies are combined with the `AND` logical operator.

In other words, a table row will be available to the user if the following condition is met: `any_of(PERMISSIVE_filters) AND all_of(RESTRICTIVE_filters)`.

For example, if the following policies are set, the `john` user can access only rows that have both `x=1` and `y=2`:

```
CREATE ROW POLICY policy_1 ON test_table USING x=1 TO mary, john;
CREATE ROW POLICY policy_2 ON test_table USING y=2 AS RESTRICTIVE TO john, andrew;
```

USING	Condition to filter table rows — a user will see only rows for which the specified `<filter_condition>` expression returns a non-zero value
TO	Users and roles for which the row policy should apply

If no row policy is specified for a table, any user can access all rows of the table. If you create at least one policy for a table, this policy will control access to rows of this table for all users (including users for whom the policy is not specified explicitly).

For example, if you create the policy:

```
CREATE ROW POLICY policy_1 ON test_table USING x=1 TO mary, john;
```

then:

- the `mary` and `john` users will only be able to see rows for which the `x=1` condition is met;
- all other users will not have access to rows of the `test_table` table at all.

To fix this behavior and grant permissions for other users to still have access to all rows, add one more row policy for these users. For example, create the following policy for the `admin` user:

```
CREATE ROW POLICY allow_admin_filter ON test_table USING 1 TO admin;
```

Modify a row policy

The `ALTER ROW POLICY` query allows you to change settings of an existing row policy:

```
ALTER [ROW] POLICY [IF EXISTS] <policy_name1> [ON CLUSTER <cluster_name1>] ON [<db_name1>.]<table_name1> [RENAME TO
<new_policy_name1>]
        [, <policy_name2> [ON CLUSTER <cluster_name2>] ON [<db_name2>.]<table_name2> [RENAME TO <new_policy_name2>] ...]
    [AS {PERMISSIVE | RESTRICTIVE}]
    [FOR SELECT]
    [USING {<filter_condition> | NONE}][,...]
    [TO {<user_name> | <role_name> [,...] | ALL | ALL EXCEPT <user_name> | <role_name> [,...]}];
```

View row policy settings

Use the `SHOW CREATE ROW POLICY` query to view the current settings of a row policy for a particular table:

```
SHOW CREATE [ROW] POLICY <policy_name> ON [<db_name1>.]<table_name1> [, [<db_name2>.]<table_name2> ...];
```

Delete row policy

The `DROP ROW POLICY` query removes a row policy:

```
DROP [ROW] POLICY [IF EXISTS] <policy_name> [,...] ON [<db_name>.]<table_name> [,...] [ON CLUSTER <cluster_name>];
```

Settings profile

Settings profile is a set of settings with the specified values and/or constraints to be assigned to a user account (or role) and applied when the user logs in.

Create a settings profile

Use the `CREATE SETTINGS PROFILE` query to create a settings profile. Query syntax:

```
CREATE SETTINGS PROFILE [IF NOT EXISTS | OR REPLACE] <profile_name1> [ON CLUSTER <cluster_name1>]
    [, <profile_name2> [ON CLUSTER <cluster_name2>] ...]
    [SETTINGS <setting_name> [= <value>] [MIN [=] <min_value>] [MAX [=] <max_value>]
[CONST|READONLY|WRITABLE|CHANGEABLE_IN_READONLY] | INHERIT '<profile_name>'] [,...]
    [TO {<role_name> | <user_name> | CURRENT_USER} [,...] | ALL | ALL EXCEPT {<role_name> | <user_name> | CURRENT_USER} [,...]];
```

Example

Create the `my_profile` settings profile that contains a value of the `max_memory_usage` parameter and prohibits changing this variable, and assign this profile to the `john` user. You can do this in several ways.

- Run the `CREATE SETTINGS PROFILE` query and use the `TO` clause to specify the user:

```
CREATE SETTINGS PROFILE my_profile SETTINGS max_memory_usage = 100000000 READONLY TO john;
```

- Create a settings profile and then assign it to a user with the `SETTINGS PROFILE` clause in the `CREATE USER` or `ALTER USER` query:

```
CREATE SETTINGS PROFILE my_profile SETTINGS max_memory_usage = 100000000 READONLY;
```

```
ALTER USER john SETTINGS PROFILE my_profile;
```

- You can also assign a settings profile to a role, which in turn is assigned to a user:

```
ALTER ROLE reader SETTINGS PROFILE my_profile;
```

Modify a settings profile

If you need to change an existing settings profile, use the `ALTER SETTINGS PROFILE` query:

```
ALTER SETTINGS PROFILE [IF EXISTS] TO <profile_name1> [ON CLUSTER <cluster_name1>] [RENAME TO <new_profile_name1>]
    [, <profile_name2> [ON CLUSTER <cluster_name2>] [RENAME TO <new_profile_name2>] ...]
    [SETTINGS <setting_name> [= <value>] [MIN [=] <min_value>] [MAX [=] <max_value>]
[CONST|READONLY|WRITABLE|CHANGEABLE_IN_READONLY] | INHERIT '<profile_name>'] [,...]
    [TO {<role_name> | <user_name> | CURRENT_USER} [,...] | ALL | ALL EXCEPT {<role_name> | <user_name> | CURRENT_USER} [,...]];
```

View parameters of a settings profile

The `SHOW CREATE SETTINGS PROFILE` query allows you to view parameters used when configuring a settings profile:

```
SHOW CREATE [SETTINGS] PROFILE <profile_name1> [, <profile_name2> ...];
```

Delete a settings profile

The `DROP SETTINGS PROFILE` query removes a settings profile:

```
DROP [SETTINGS] PROFILE [IF EXISTS] <profile_name> [,...] [ON CLUSTER <cluster_name>];
```

A deleted settings profile is revoked from all access system objects to which it is assigned.

Quota

A **quota** is assigned to a user or role and allows you to track or limit the use of resources (for example, the number of processed queries, read rows and bytes) over the specified period of time for that user or role:

- **Track parameter values over a period of time.** During the specified time interval, parameter values are counted after each query execution and stored in the server log. When the interval ends, all collected values are reset and the quota calculation starts again for the next interval.
- **Limit parameter values per a period of time.** If a parameter value exceeds its limit during the specified time interval, an exception is thrown with information about which parameter value has been exceeded, and when a new interval starts to allow sending queries again. For one quota, you can set multiple intervals with different restrictions.

When processing a distributed query, ADQM takes into account resources spent on all remote servers. Accumulated values are stored on the server initiated the query (if the user goes to another server, the quota will "start over" there).

Create a quota

Use the `CREATE QUOTA` query to create a quota and assign it to a user or role:

```
CREATE QUOTA [IF NOT EXISTS | OR REPLACE] <quota_name> [ON CLUSTER <cluster_name>]
    [KEYED BY {user_name | ip_address | client_key | client_key, user_name | client_key, ip_address} | NOT KEYED]
    [FOR [RANDOMIZED] INTERVAL <number> {second | minute | hour | day | week | month | quarter | year}
        {MAX { {queries | query_selects | query_inserts | errors | result_rows | result_bytes | read_rows | read_bytes |
execution_time} = <max_value> } [,...] |
        NO LIMITS | TRACKING ONLY} [,...]]
    [TO {<role_name> | <user_name> | CURRENT_USER} [,...] | ALL | ALL EXCEPT {<role_name> | <user_name> | CURRENT_USER} [,...]];
```

Basic clauses

KEYED BY	Key that specifies how the quota should be shared. Resource accounting for each quota key value is maintained independently. In other words, connections with the same quota and key share the same amount of resources. The `user_name`, `ip_address`, `client_key`, `client_key, user_name`, and `client_key, ip_address` keys correspond to fields of the system.quotas system table
FOR INTERVAL	Time interval per which the use of resources is tracked or limited. You can restrict values of the following parameters that correspond to the system.quotas_usage table fields: `queries`, `query_selects`, `query_inserts`, `errors`, `result_rows`, `result_bytes`, `read_rows`, `read_bytes`, `execution_time`
TO	Users/roles for which the quota is allocated

Example

Create a quota that limits the maximum number of queries the `john` user can execute — no more than 10 queries every 15 minutes:

```
CREATE QUOTA my_quota FOR INTERVAL 15 minute MAX queries = 10 TO john;
```

If the user has executed 10 queries before 15 minutes have elapsed from the start of the quota, the following message will appear on the eleventh query execution:

```
Code: 201. DB::Exception: Received from localhost:9000.
DB::Exception: Quota for user `john` for 900s has been exceeded: queries = 11/10.
Interval will end at 2023-02-08 07:15:00. Name of quota template: `my_quota`. (QUOTA_EXCEEDED)
```

Modify a quota

You can change the settings of an existing quota with the `ALTER QUOTA` query:

```
ALTER QUOTA [IF EXISTS] <quota_name> [ON CLUSTER <cluster_name>]
    [RENAME TO <new_quota_name>]
    [KEYED BY {user_name | ip_address | client_key | client_key, user_name | client_key, ip_address} | NOT KEYED]
    [FOR [RANDOMIZED] INTERVAL <number> {second | minute | hour | day | week | month | quarter | year}
        {MAX { {queries | query_selects | query_inserts | errors | result_rows | result_bytes | read_rows | read_bytes |
execution_time} = <max_value> } [,...] |
        NO LIMITS | TRACKING ONLY} [,...]]
    [TO {<role_name> | <user_name> | CURRENT_USER} [,...] | ALL | ALL EXCEPT {<role_name> | <user_name> | CURRENT_USER} [,...]];
```

View information on quotas

- **List of quota names.** Execute the `SHOW QUOTAS` query to get names of all existing quotas:

```
SHOW QUOTAS;
```

- **Current settings of a quota.** You can obtain quota parameters (with changes, if the `ALTER QUOTA` query was executed for the quota) in one of the following ways:
  - use the `SHOW CREATE QUOTA` query:

```
SHOW CREATE QUOTA [<quota_name> [, ...] | CURRENT];
```

  - read values from the system.quotas system table:

```
SELECT * FROM system.quotas [WHERE name = '<quota_name>'] [FORMAT <format_name>];
```

- **Information about quota usage.** To find out how many resources have been used and how much is left, you can:
  - run the `SHOW QUOTA` query (quota usage by all users or by the current user):

```
SHOW [CURRENT] QUOTA;
```

  - read values from the system.quotas_usage system table (quota usage by all users):

```
SELECT * FROM system.quotas_usage [WHERE quota_name = '<quota_name>'] [FORMAT <format_name>];
```

  - read values from the system.quota_usage system table (quota usage by the current user):

```
SELECT * FROM system.quota_usage [WHERE quota_name = '<quota_name>'] [FORMAT <format_name>];
```

Delete a quota

The `DROP QUOTA` query removes a quota:

```
DROP QUOTA [IF EXISTS] <quota_name> [,...] [ON CLUSTER <cluster_name>];
```

Encryption of database connections

Elena Dvoryadkina

Contents

To Table of Contents

- Create SSL certificates
- Configure ADQM to enable SSL encryption
  - Manage SSL cluster action
  - SSL parameters in service configurations
  - Example
- Test connection
  - Connect via clickhouse-client
  - Check interserver connections
  - Connect via HTTPS
- Disable SSL

You can use the SSL encryption protocol to secure communication between an ADQM server and client, as well as between servers in a cluster. This article provides instructions on how to configure ADQM to enable SSL certificate validation for incoming connections and use secure protocols for interserver communication. The example uses an ADQM cluster that includes the following hosts:

- 10.92.17.146, adqm-host-1.arenadata.local;
- 10.92.17.106, adqm-host-2.arenadata.local;
- 10.92.17.140, adqm-host-3.arenadata.local;
- 10.92.16.245, adqm-host-4.arenadata.local.

> **NOTE**
>
> In this article, a self-signed CA certificate is used to create server SSL certificates for demonstration and testing purposes. For production systems, use a trusted certificate authority to sign certificates.

Create SSL certificates

To run commands below, you need `openssl` installed (minimum version 1.1.1) — it is a command-line tool for using functions of the OpenSSL cryptographic library.

1. Generate a private key:

```
$ openssl genrsa -out arenadata_ca.key 2048
```

   This key will be used to create a CA certificate and to sign other certificates (server certificates and certificates for user authentication), so keep it in a secure place.

2. Create a self-signed CA certificate:

```
$ openssl req \
  -x509 \
  -subj "/CN=arenadata.local CA" \
  -nodes \
  -key arenadata_ca.key \
  -days 1095 \
  -out arenadata_ca.crt
```

3. For each ADQM server, generate a key and create a certificate signing request (CSR):

```
$ openssl req -newkey rsa:2048 -nodes -subj "/CN=adqm-host-1.arenadata.local" -addext "subjectAltName = DNS:adqm-host-1.arenadata.local,IP:10.92.17.146" -keyout adqm-host-1.key -out adqm-host-1.csr
$ openssl req -newkey rsa:2048 -nodes -subj "/CN=adqm-host-2.arenadata.local" -addext "subjectAltName = DNS:adqm-host-2.arenadata.local,IP:10.92.17.106" -keyout adqm-host-2.key -out adqm-host-2.csr
$ openssl req -newkey rsa:2048 -nodes -subj "/CN=adqm-host-3.arenadata.local" -addext "subjectAltName = DNS:adqm-host-3.arenadata.local,IP:10.92.17.140" -keyout adqm-host-3.key -out adqm-host-3.csr
$ openssl req -newkey rsa:2048 -nodes -subj "/CN=adqm-host-4.arenadata.local" -addext "subjectAltName = DNS:adqm-host-4.arenadata.local,IP:10.92.16.245" -keyout adqm-host-4.key -out adqm-host-4.csr
```

   The CN (Common Name) certificate identifier should be the fully qualified domain name (FQDN) of a server for which the certificate is being created. Otherwise, the certificate cannot be verified.

4. For each ADQM server, create and sign a certificate using the private key and the CA certificate:

```
$ openssl x509 -req -in adqm-host-1.csr -out adqm-host-1.crt -CAcreateserial -CA arenadata_ca.crt -CAkey arenadata_ca.key -days 365
$ openssl x509 -req -in adqm-host-2.csr -out adqm-host-2.crt -CAcreateserial -CA arenadata_ca.crt -CAkey arenadata_ca.key -days 365
$ openssl x509 -req -in adqm-host-3.csr -out adqm-host-3.crt -CAcreateserial -CA arenadata_ca.crt -CAkey arenadata_ca.key -days 365
$ openssl x509 -req -in adqm-host-4.csr -out adqm-host-4.crt -CAcreateserial -CA arenadata_ca.crt -CAkey arenadata_ca.key -days 365
```

5. For each ADQM host, copy the CA certificate, as well as the certificate and key of the corresponding server. These files should be stored in the same path and named the same on each host of the cluster — paths and file names of the CA certificate, server certificate and server key are specified in cluster settings (see the SSL parameters in service configurations section below).

   For example, copy files to the *adqm-host-1.arenadata.local* host as follows:

   - *arenadata_ca.crt* to */etc/clickhouse-server/ca.crt*;
   - *adqm-host-1.crt* to */etc/clickhouse-server/server.crt*;
   - *adqm-host-1.key* to */etc/clickhouse-server/server.key*.

> The */etc/clickhouse-server/server.crt* and */etc/clickhouse-server/server.key* files are set by default in the ***Server certificate file*** and ***Server private key file*** parameters of the ADQMDB service, respectively. If the certificate and key files on your ADQM hosts have different names or are located in a different directory, specify the corresponding paths in these parameters.
>
> The self-signed CA certificate should be specified in one of the following ways:
>
> - Pass the path and file name of the CA certificate to the ***Certificate authority file*** parameter of the ADQMDB service configuration (see the example below), which has an empty value by default.
> - Add the CA certificate to trusted certificates. For example, to do this on CentOS, perform the following commands:
>
> ```
> $ sudo cp ./arenadata_ca.crt /etc/pki/ca-trust/source/anchors/
> $ sudo update-ca-trust
> ```

> **NOTE**
>
> Once SSL is enabled for the ADQMDB service (see the next section), `clickhouse` is automatically set as the owner and group for directories storing used SSL certificates and keys and for certificate and key files, and the following access rights are also assigned:
>
> - `0770` — for directories storing server and client certificates and keys;
> - `0771` — for directories storing CA certificates;
> - `0600` — for server and client certificate and key files;
> - `0664` — for CA certificate files.

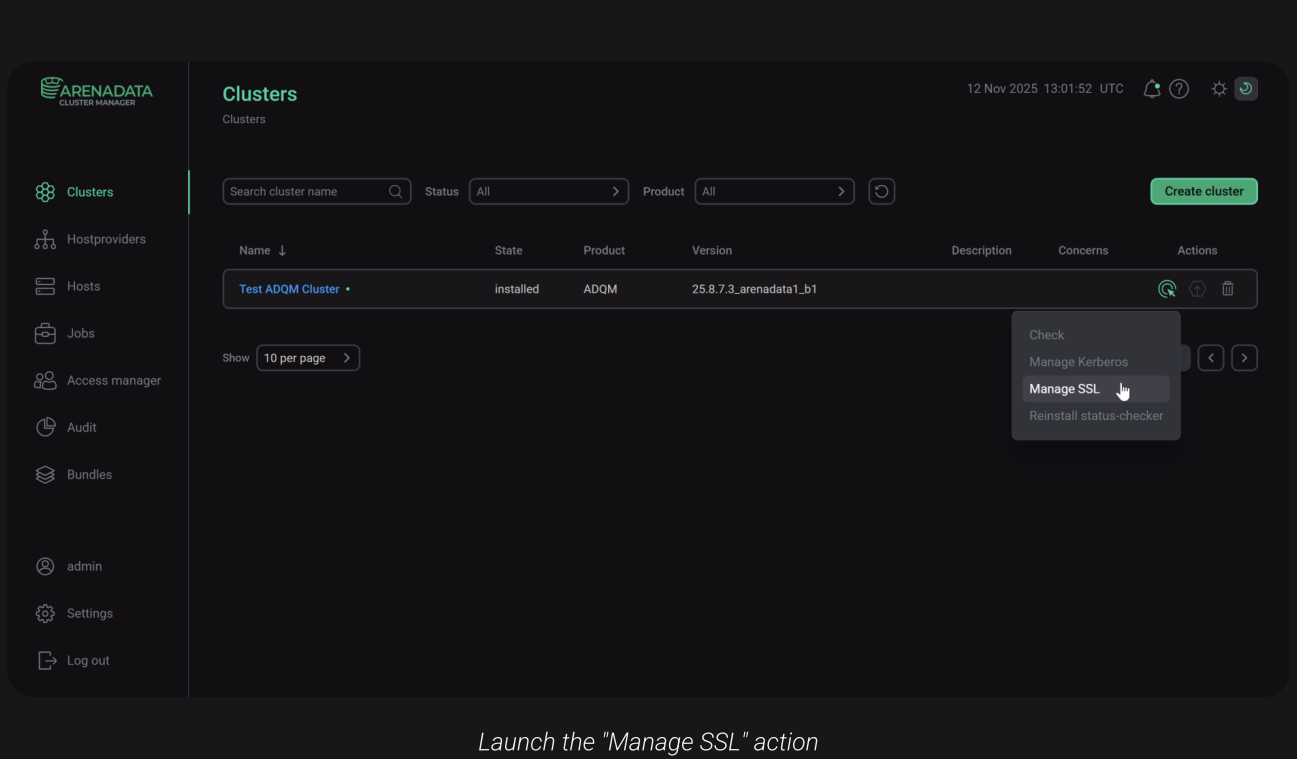
Configure ADQM to enable SSL encryption

You can enable SSL encryption of connections in ADQM using the ADCM interface, which provides the following functionality for this:

- Manage SSL cluster action — allows you to enable SSL for multiple ADQM services at once, as well as disable ports for unsecured connections.
- SSL settings in configurations of ADQM services — allow you to configure SSL for each service separately (for example, you can change the default paths to the certificate and key files, enable client certificate verification, specify the CA certificate, etc.).

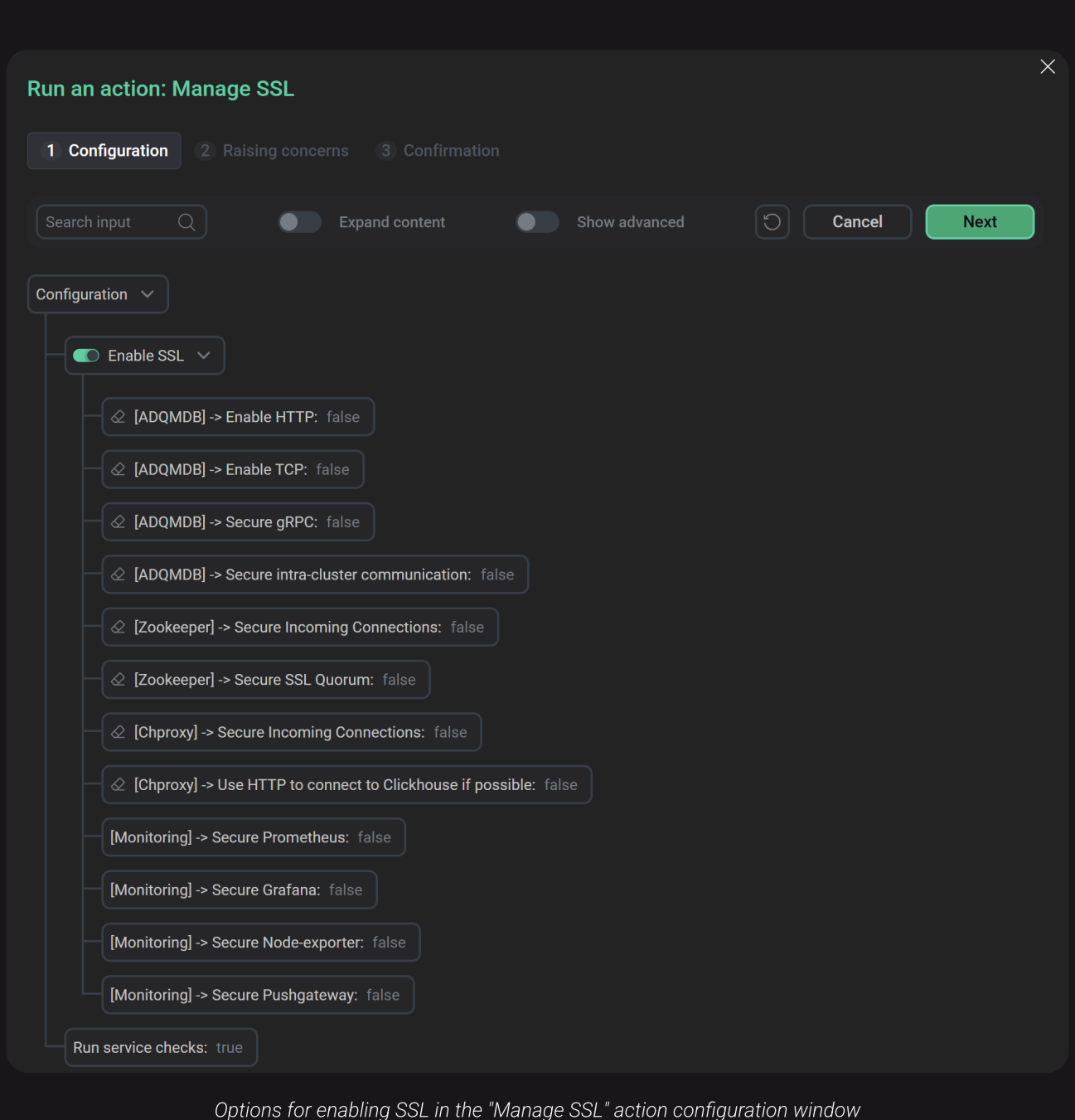
Manage SSL cluster action

1. On the ***Clusters*** page, click the icon in the ADQM cluster row to open the list of cluster actions and select ***Manage SSL***.


ARENADATA CLUSTER MANAGER
Clusters
12 Nov 2025 13:01:52 UTC
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Search cluster name
Status All
Product All
Create cluster
Name
State
Product
Version
Description
Concerns
Actions
Test ADQM Cluster
installed
ADQM
25.8.7.3_arenadata1_b1
Check
Manage Kerberos
Manage SSL
Reinstall status-checker
Show 10 per page

Launch the "Manage SSL" action

   ADCM will display a dialog window. When you turn on the ***Enable SSL*** switch, it provides options allowing you to specify which ADQM services should use secure connections.


Run an action: Manage SSL
1 Configuration
2 Raising concerns
3 Confirmation
Search input
Expand content
Show advanced
Cancel
Next
Configuration
Enable SSL

[ADQMDB] -> Enable HTTP: false
[ADQMDB] -> Enable TCP: false
[ADQMDB] -> Secure gRPC: false
[ADQMDB] -> Secure intra-cluster communication: false
[Zookeeper] -> Secure Incoming Connections: false
[Zookeeper] -> Secure SSL Quorum: false
[Chproxy] -> Secure Incoming Connections: false
[Chproxy] -> Use HTTP to connect to Clickhouse if possible: false
[Monitoring] -> Secure Prometheus: false
[Monitoring] -> Secure Grafana: false
[Monitoring] -> Secure Node-exporter: false
[Monitoring] -> Secure Pushgateway: false
Run service checks: true

Options for enabling SSL in the "Manage SSL" action configuration window

2. Fill in the required parameters in the ***Manage SSL*** action configuration window:

   - Parameters in the ***Enable SSL*** section — to enable SSL encryption of incoming and internal connections for various ADQM services.

Parameters to enable SSL for services

Parameter values specified in the ***Manage SSL*** action window overwrite the values of the corresponding parameters in service configurations after the action execution — see the ***ADQM service parameter*** column in the table below.

Manage SSL parameter	Parameter description	ADQM service parameter
[ADQMDB] → Enable HTTP	Allows you to disable the port for connecting to ADQM over HTTP (to use HTTPS only)	***ADQMDB*** service: ***Network → Enable http*** parameter
[ADQMDB] → Enable TCP	Allows you to disable the non-secure TCP port so that only the secure TCP port (***SSL Configuration → TCP secure port***) is used	***ADQMDB*** service: ***Network → Enable tcp*** parameter
[ADQMDB] → Secure gRPC	Enables SSL for secure communication with clients via gRPC	***ADQMDB*** service: ***Enable gRPC → Enable SSL*** parameter
[ADQMDB] → Secure intra-cluster communication	If this parameter is enabled, only secure protocols (HTTPS, secure TCP) will be used to exchange data between ClickHouse servers	***ADQMDB*** service: ***Intra-cluster communication security → Enable secure intra-cluster communication*** parameter
[Zookeeper] → Secure Incoming Connections	Enables secure incoming connections in ZooKeeper	***Zookeeper*** service: ***SSL configuration → Enable SSL*** parameter
[Zookeeper] → Secure SSL Quorum	Enables encrypted quorum communication in ZooKeeper	***Zookeeper*** service: ***SSL configuration → sslQuorum*** parameter
[Chproxy] → Secure Incoming Connections	Enables secure incoming connections in Chproxy	***Chproxy*** service: ***Network configuration → Enable HTTPS*** parameter
[Chproxy] → Use HTTP to connect to Clickhouse if possible	Specifies which protocol (HTTP or HTTPS) Chproxy will use to connect to ClickHouse	***Chproxy*** service: ***Network configuration → Use HTTP to connect to ADQMDB*** parameter
[Monitoring] → Secure Prometheus	Enables SSL encryption of connections to the Prometheus Server component of the Monitoring service	***Monitoring*** service: ***[Prometheus] → Enable SSL*** parameter
[Monitoring] → Secure Grafana	Enables SSL encryption of connections to the Grafana component of the Monitoring service	***Monitoring*** service: ***[Grafana] → Enable SSL*** parameter
[Monitoring] → Secure Node-exporter	Enables SSL encryption of connections to the Node Exporter component of the Monitoring service	***Monitoring*** service: ***[Node-exporter] → Enable SSL*** parameter
[Monitoring] → Secure Pushgateway	Enables SSL encryption of connections to the Pushgateway component of the Monitoring service	***Monitoring*** service: ***[Pushgateway] → Enable SSL*** parameter

   - The ***Run service checks*** parameter — to specify whether to check the cluster health and availability after applying the settings.

3. To launch the action, click ***Next*** and then ***Run*** in the confirmation window.

During the ***Manage SSL*** action execution, ADCM stops all services one by one, reconfigures them to enable or disable SSL encryption, then starts the services again and checks them.

You can view the result of the action and its execution process on the Jobs page, which displays the execution details for all internal steps of actions, including verbose Ansible stdout/stderr outputs.

SSL parameters in service configurations

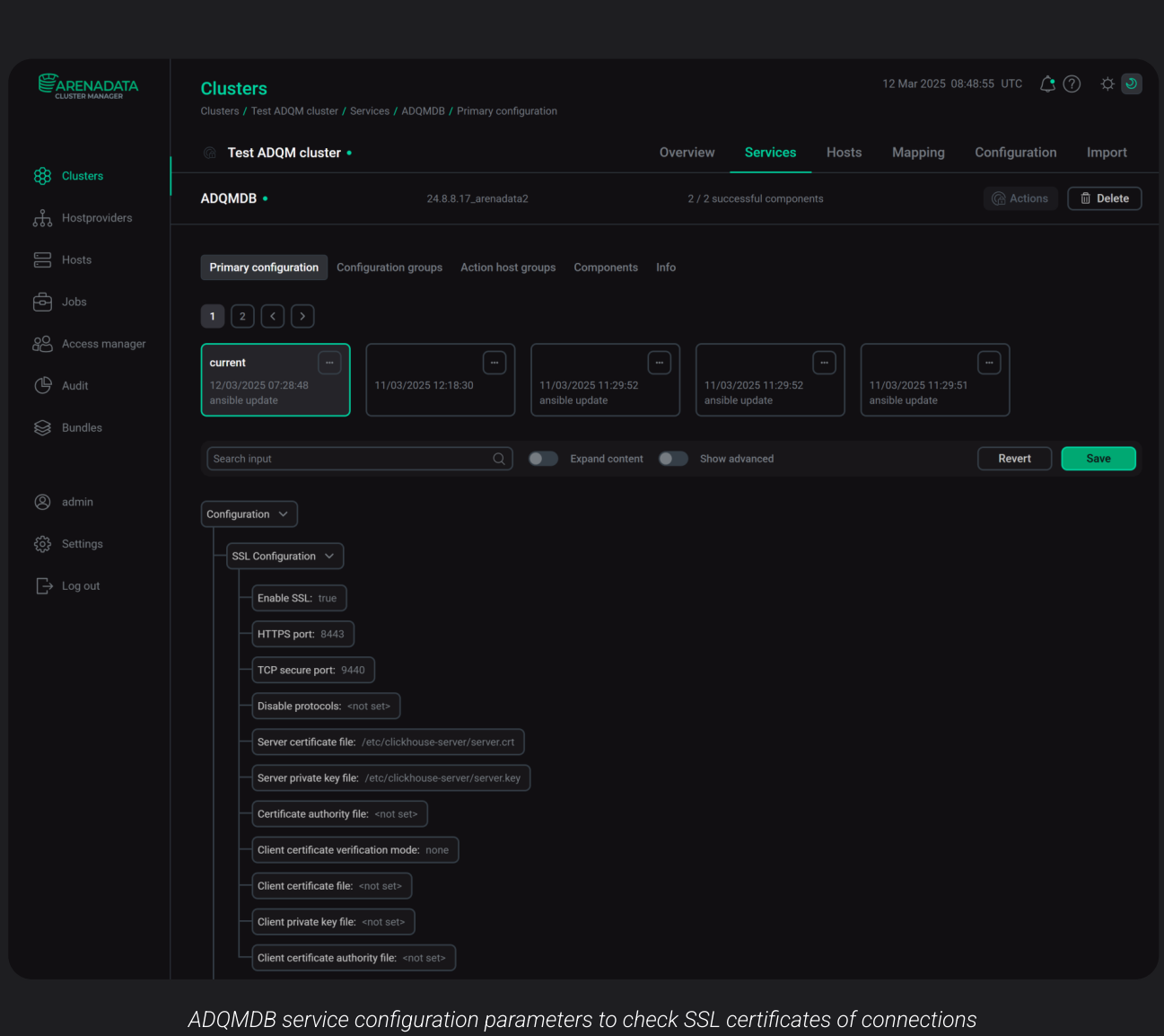
An alternative way to set up secure connections in ADQM is to configure SSL settings for each service separately.

1. Open the ***Services*** tab of your ADQM cluster and select the service for which you want to enable SSL encryption — ADQMDB, Zookeeper, or Chproxy.
2. On the ***Primary configuration*** tab, configure the SSL parameters — the name of the section where these parameters are located depends on the service.

ADQMDB: SSL Configuration

In the ***SSL Configuration*** section of the ADQMDB service configuration, the following parameters are available:

- ***Enable SSL*** — enables HTTPS and TCP secure ports for SSL-protected connections.
- ***HTTPS port*** — port for secure communication via HTTPS (default is `8443`).
- ***TCP secure port*** — TCP port for secure communication with clients and inter-server communication through the binary protocol (default is `9440`).
- ***Disable protocols*** — protocols that are not allowed to be used (for example, `sslv2`, `sslv3`).
- ***Server certificate file*** — path to the server SSL certificate file in the PEM format.
- ***Server private key file*** — path to the file with the private key of the server SSL certificate.
- ***Certificate authority file*** — path to the file with the trusted CA certificate in the PEM format used to verify client certificates. It is required if you need to enable TLS authentication.
- ***Client certificate verification mode*** — method of checking user SSL certificates signed by the CA certificate (the CA certificate file should be specified in the ***Certificate authority file*** field).
- ***Client certificate file***, ***Client private key file***, ***Client certificate authority file*** — parameters to configure a secure connection between ClickHouse and ZooKeeper.


ARENADATA CLUSTER MANAGER
Clusters
Clusters / Test ADQM cluster / Services / ADQMDB / Primary configuration
12 Mar 2025 08:48:55 UTC
Test ADQM cluster
Overview
Services
Hosts
Mapping
Configuration
Import
ADQMDB
24.8.8.17_arenadata2
2 / 2 successful components
Actions
Delete
Primary configuration
Configuration groups
Action host groups
Components
Info
current
Search input
Expand content
Show advanced
Revert
Save
Configuration
SSL Configuration

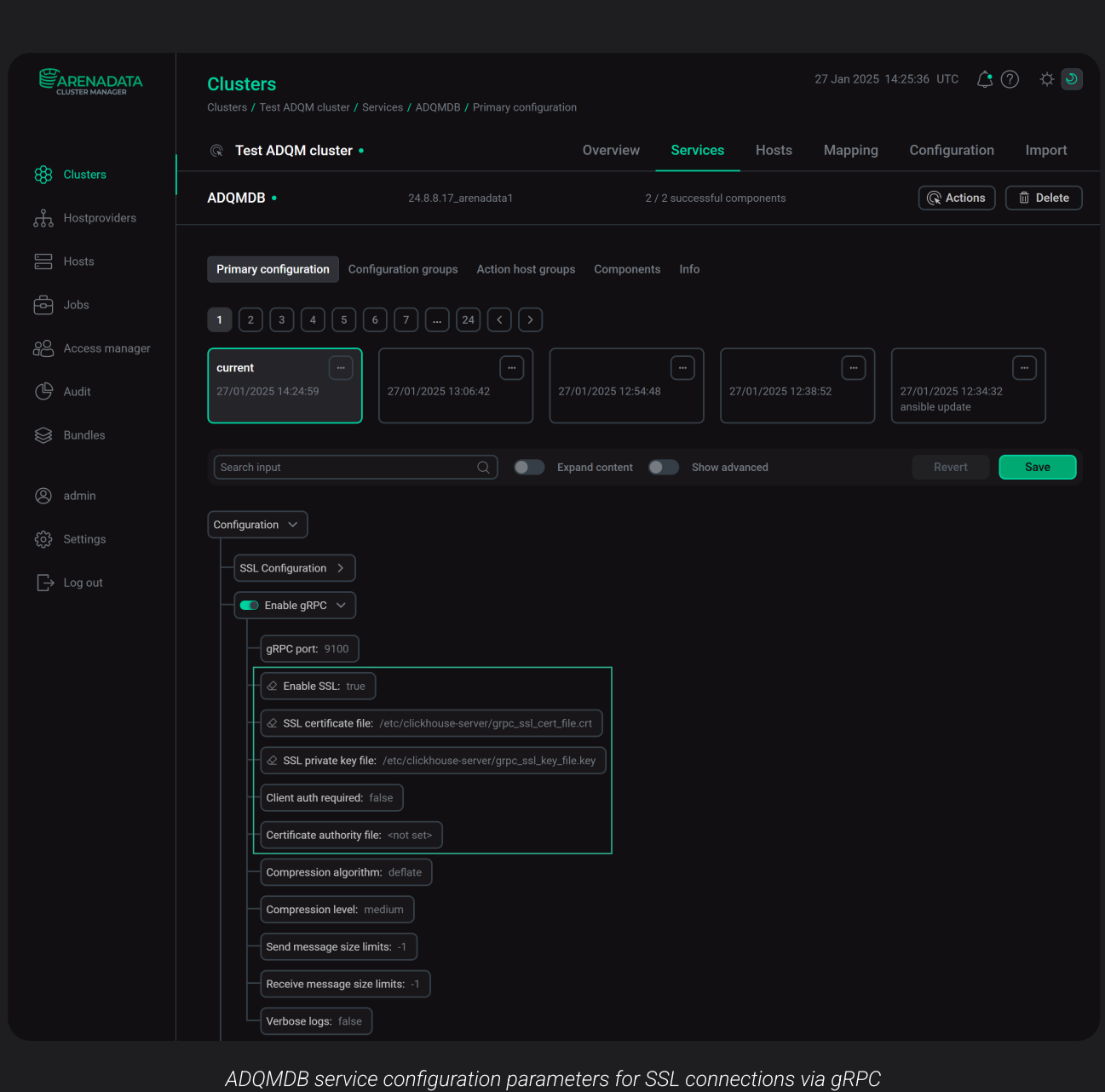
Enable SSL: true
HTTPS port: 8443
TCP secure port: 9440
Disable protocols: <not set>
Server certificate file: /etc/clickhouse-server/server.crt
Server private key file: /etc/clickhouse-server/server.key
Certificate authority file: <not set>
Client certificate verification mode: none
Client certificate file: <not set>
Client private key file: <not set>
Client certificate authority file: <not set>

ADQMDB service configuration parameters to check SSL certificates of connections

ADQMDB: Enable gRPC

If the ***Enable SSL*** (in the ***SSL Configuration*** section) and ***Enable gRPC*** options are enabled for the ADQMDB service, you can also configure the following SSL settings for gRPC connections in the ***Enable gRPC*** section:

- ***Enable SSL*** — enables SSL for secure communication with clients via gRPC.
- ***SSL certificate file*** — path to the file with the SSL certificate in the PEM format for connections via gRPC.
- ***SSL private key file*** — path to the file with the secret key of the SSL certificate for connections via gRPC.
- ***Client auth required*** — indicates whether the server should request a client for a certificate.
- ***Certificate authority file*** — path to the file with the trusted CA certificate in the PEM format that will be used to verify client certificates. It should be specified if ***Client auth required*** is enabled.


ARENADATA CLUSTER MANAGER
Clusters
Clusters / Test ADQM cluster / Services / ADQMDB / Primary configuration
27 Jan 2025 14:25:36 UTC
Test ADQM cluster
Overview
Services
Hosts
Mapping
Configuration
Import
ADQMDB
24.8.8.17_arenadata1
2 / 2 successful components
Actions
Delete
Primary configuration
Configuration groups
Action host groups
Components
Info
current
27/01/2025 14:24:59
27/01/2025 13:06:42
27/01/2025 12:54:48
27/01/2025 12:38:52
27/01/2025 12:34:32
ansible update
Search input
Expand content
Show advanced
Revert
Save
Configuration
SSL Configuration
Enable gRPC

gRPC port: 9100
Enable SSL: true
SSL certificate file: /etc/clickhouse-server/grpc_ssl_cert_file.crt
SSL private key file: /etc/clickhouse-server/grpc_ssl_key_file.key
Client auth required: false
Certificate authority file: <not set>
Compression algorithm: deflate
Compression level: medium
Send message size limits: -1
Receive message size limits: -1
Verbose logs: false

ADQMDB service configuration parameters for SSL connections via gRPC

ADQMDB: Network

In the ***Network*** section of the ADQMDB service configuration, you can find options to disable non-secure ports:

- ***Enable http*** — allows you to disable the port for connections via the HTTP protocol (in favor of connections via the HTTPS port configured by the ***SSL Configuration → HTTPS port*** parameter).
- ***Enable tcp*** — allows you to disable the port for non-secure connections via the ClickHouse TCP protocol (in favor of the ***SSL Configuration → TCP secure port*** port).

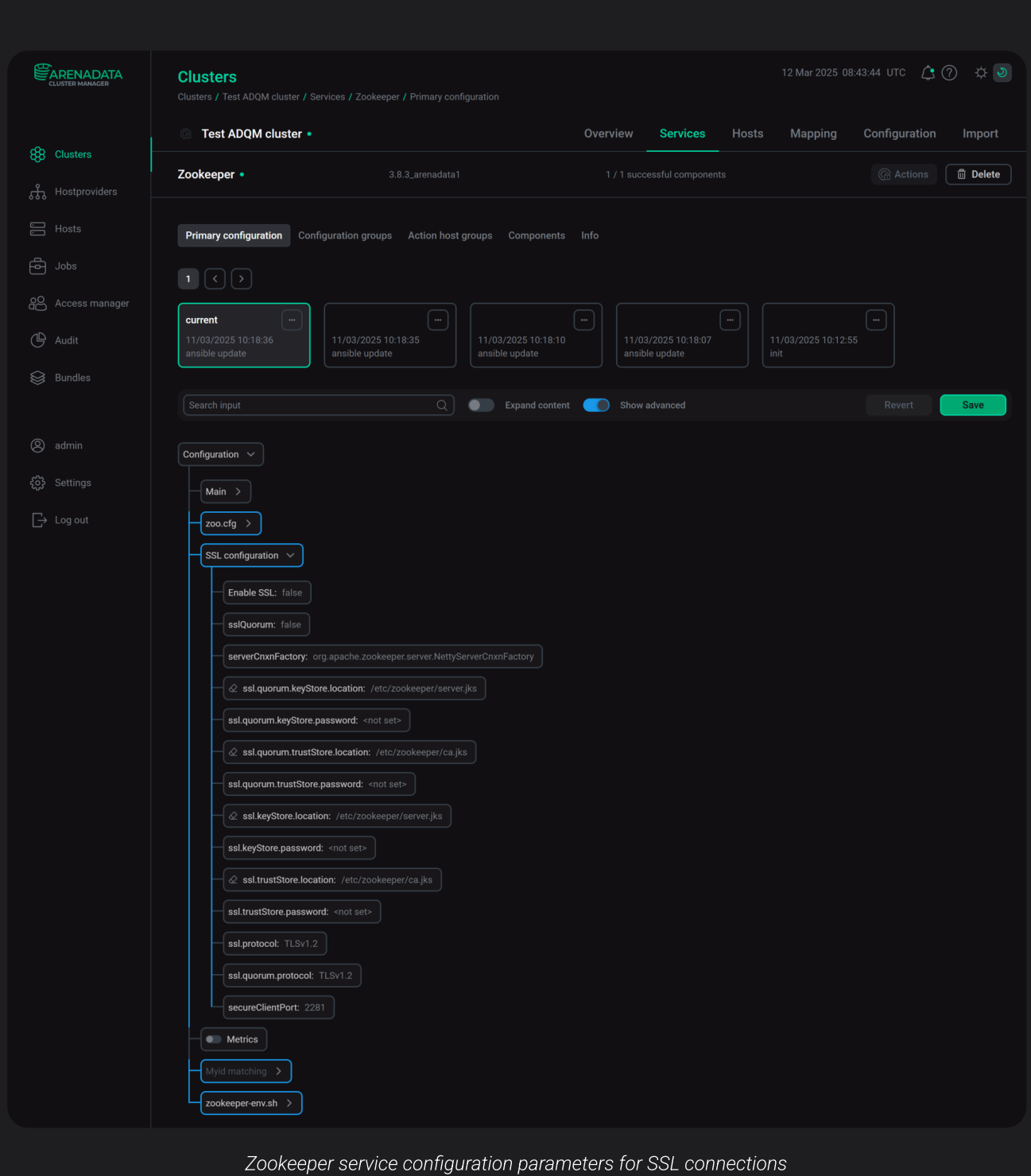
ADQMDB: Secure intra-cluster communication

In the ***Secure intra-cluster communication*** section of the ADQMDB service configuration, there is the ***Enable secure intra-cluster communication*** parameter that enables the use of only secure protocols (HTTPS, secure TCP) for data exchange between ClickHouse servers. To activate the parameter, turn on also ***SSL Configuration → Enable SSL***.

Zookeeper: SSL configuration

The ***SSL configuration*** section of the Zookeeper service configuration (available if the ***Show advanced*** option is enabled) contains parameters to set up SSL connections between ClickHouse and ZooKeeper:

- ***Enable SSL*** — enables secure incoming connections in ZooKeeper;
- ***sslQuorum*** — enables encrypted quorum communication;
- ***serverCnxnFactory*** — specifies `ServerCnxnFactory` implementation (to use TLS-based server communication, set it to `NettyServerCnxnFactory`);
- ***ssl.quorum.keyStore.location*** — path to the server keystore file;
- ***ssl.quorum.keyStore.password*** — password for the keystore specified in ***ssl.quorum.keyStore.location***;
- ***ssl.quorum.trustStore.location*** — path to the server truststore file;
- ***ssl.quorum.trustStore.password*** — password for the truststore specified in ***ssl.quorum.trustStore.location***;
- ***ssl.keyStore.location*** — path to the server keystore file to be used for client TLS connections;
- ***ssl.keyStore.password*** — password for the keystore specified in ***ssl.keyStore.location***;
- ***ssl.trustStore.location*** — path to the server truststore file to be used for client TLS connections;
- ***ssl.trustStore.password*** — password for the truststore specified in ***ssl.trustStore.location***;
- ***ssl.protocol*** — protocol to be used in client TLS negotiation;
- ***ssl.quorum.protocol*** — protocol to be used in quorum TLS negotiation;
- ***secureClientPort*** — port to be used for secure client connections.


ARENADATA CLUSTER MANAGER
Clusters
Clusters / Test ADQM cluster / Services / Zookeeper / Primary configuration
12 Mar 2025 08:43:44 UTC
Test ADQM cluster
Overview
Services
Hosts
Mapping
Configuration
Import
Zookeeper
3.8.3_arenadata1
1 / 1 successful components
Actions
Delete
Primary configuration
Configuration groups
Action host groups
Components
Info
current
Search input
Expand content
Show advanced
Revert
Save
Configuration
Main
zoo.cfg
SSL configuration
Metrics
zookeeper-env.sh

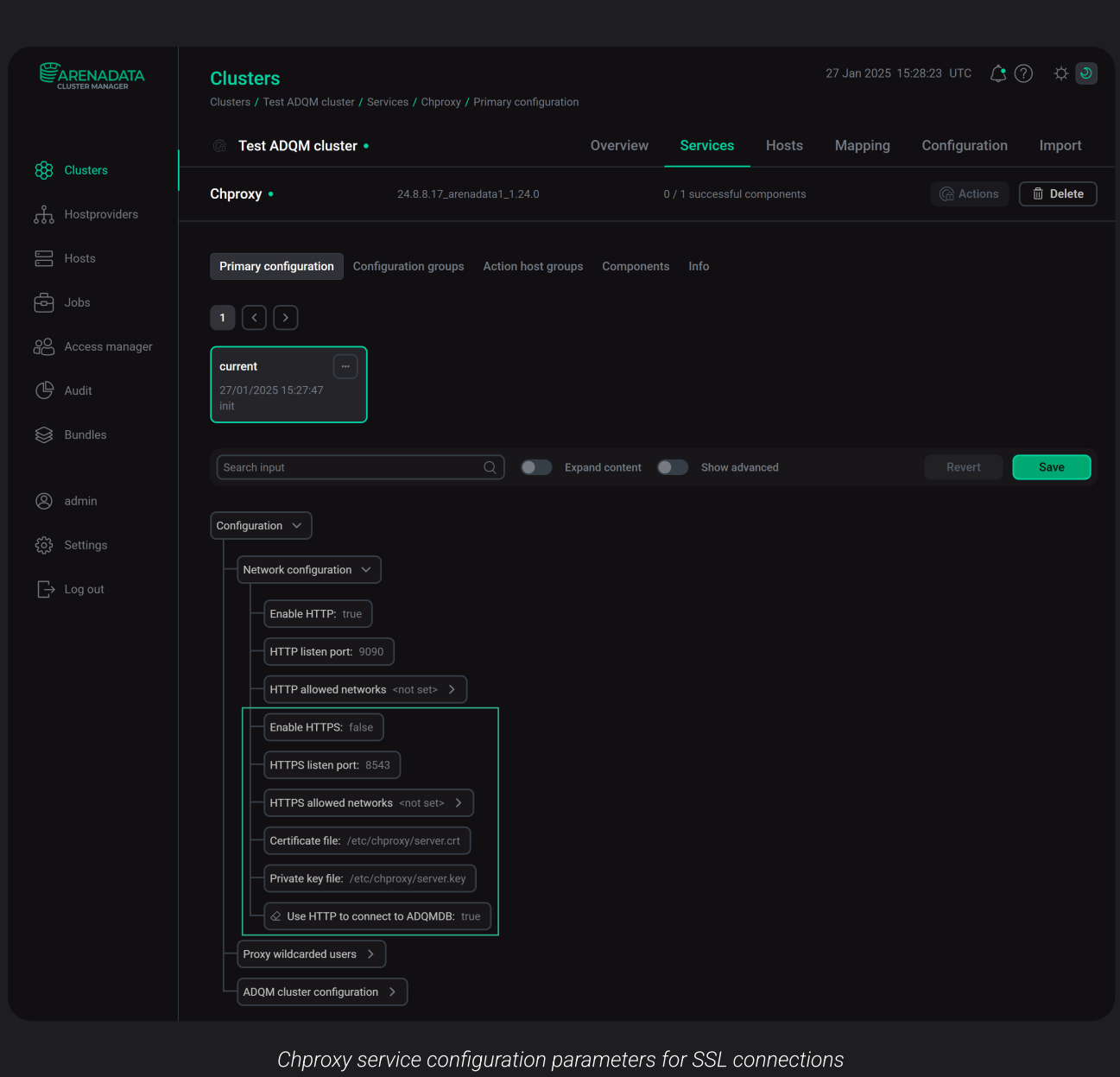
Enable SSL: false
sslQuorum: false
serverCnxnFactory: org.apache.zookeeper.server.NettyServerCnxnFactory
ssl.quorum.keyStore.location: /etc/zookeeper/server.jks
ssl.quorum.keyStore.password: <not set>
ssl.quorum.trustStore.location: /etc/zookeeper/ca.jks
ssl.quorum.trustStore.password: <not set>
ssl.keyStore.location: /etc/zookeeper/server.jks
ssl.keyStore.password: <not set>
ssl.trustStore.location: /etc/zookeeper/ca.jks
ssl.trustStore.password: <not set>
ssl.protocol: TLSv1.2
ssl.quorum.protocol: TLSv1.2
secureClientPort: 2281

Zookeeper service configuration parameters for SSL connections

Chproxy: Network configuration

The ***Network configuration*** section of the Chproxy service configuration includes the following parameters to set up secure connections:

- ***Enable HTTPS*** — enables listening for HTTPS requests;
- ***HTTPS listen port*** — port to listen for HTTPS requests;
- ***HTTPS allowed networks*** — list of networks from which HTTPS access is allowed;
- ***Certificate file*** — path to the server SSL certificate file in the PEM format;
- ***Private key file*** — path to the file with the private key of the server SSL certificate;
- ***Use HTTP to connect to ADQMDB*** — specifies which protocol (HTTP or HTTPS) Chproxy should use to connect to ClickHouse.


ARENADATA CLUSTER MANAGER
Clusters
Clusters / Test ADQM cluster / Services / Chproxy / Primary configuration
27 Jan 2025 15:28:23 UTC
Test ADQM cluster
Overview
Services
Hosts
Mapping
Configuration
Import
Chproxy
24.8.8.17_arenadata1_1.24.0
0 / 1 successful components
Actions
Delete
Primary configuration
Configuration groups
Action host groups
Components
Info
current
27/01/2025 15:27:47
init
Search input
Expand content
Show advanced
Revert
Save
Configuration
Network configuration
Proxy wildcarded users
ADQM cluster configuration

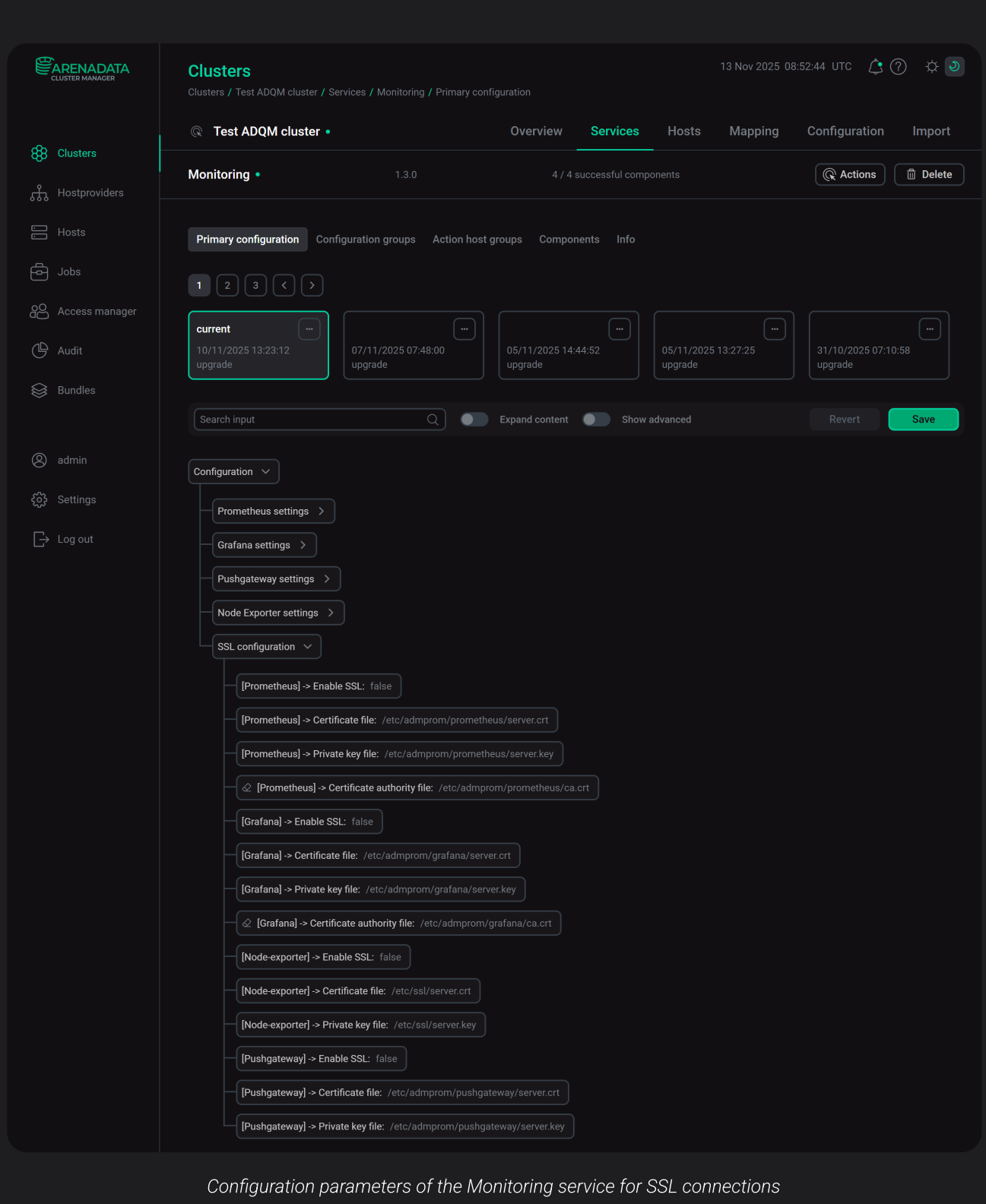
Enable HTTP: true
HTTP listen port: 9090
HTTP allowed networks <not set>
Enable HTTPS: false
HTTPS listen port: 8543
HTTPS allowed networks <not set>
Certificate file: /etc/chproxy/server.crt
Private key file: /etc/chproxy/server.key
Use HTTP to connect to ADQMDB: true

Chproxy service configuration parameters for SSL connections

Monitoring: SSL configuration

The following parameters in the ***SSL configuration*** section of the Monitoring service configuration allow you to enable SSL-encrypted connections to service components (certificate/key files should be stored at the specified paths on all cluster hosts where the corresponding components of monitoring are installed).

- ***[Prometheus] → Enable SSL*** — enables SSL encryption of incoming connections for Prometheus Server;
- ***[Prometheus] → Certificate file*** — path to the server SSL certificate file in the CRT format (**.crt* file) for Prometheus Server;
- ***[Prometheus] → Private key file*** — path to the file with the private key of the server SSL certificate (**.key* file) for Prometheus Server;
- ***[Prometheus] → Certificate authority file*** — path to the file with the CA certificate in the CRT format (**.crt* file) to verify Prometheus targets;
- ***[Grafana] → Enable SSL*** — enables SSL encryption of incoming connections for Grafana;
- ***[Grafana] → Certificate file*** — path to the server SSL certificate file in the CRT format (**.crt* file) for Grafana;
- ***[Grafana] → Private key file*** — path to the file with the private key of the server SSL certificate (**.key* file) for Grafana;
- ***[Grafana] → Certificate authority file*** — path to the file with the CA certificate in the CRT format (**.crt* file) to verify the Prometheus certificate;
- ***[Node-exporter] → Enable SSL*** — enables SSL encryption of incoming connections for node exporters;
- ***[Node-exporter] → Certificate file*** — path to the server SSL certificate file in the CRT format (**.crt* file) for node exporters;
- ***[Node-exporter] → Private key file*** — path to the file with the private key of the server SSL certificate (**.key* file) for node exporters;
- ***[Pushgateway] → Enable SSL*** — enables SSL encryption of incoming connections for Pushgateway;
- ***[Pushgateway] → Certificate file*** — path to the server SSL certificate file in the CRT format (**.crt* file) for Pushgateway;
- ***[Pushgateway] → Private key file*** — path to the file with the private key of the server SSL certificate (**.key* file) for Pushgateway.


ARENADATA CLUSTER MANAGER
Clusters
Clusters / Test ADQM cluster / Services / Monitoring / Primary configuration
13 Nov 2025 08:52:44 UTC
Test ADQM cluster
Overview
Services
Hosts
Mapping
Configuration
Import
Monitoring
1.3.0
4 / 4 successful components
Actions
Delete
Primary configuration
Configuration groups
Action host groups
Components
Info
current
10/11/2025 13:23:12
07/11/2025 07:48:00
05/11/2025 14:44:52
05/11/2025 13:27:25
31/10/2025 07:10:58
upgrade
Search input
Expand content
Show advanced
Revert
Save
Configuration
Prometheus settings
Grafana settings
Pushgateway settings
Node Exporter settings
SSL configuration

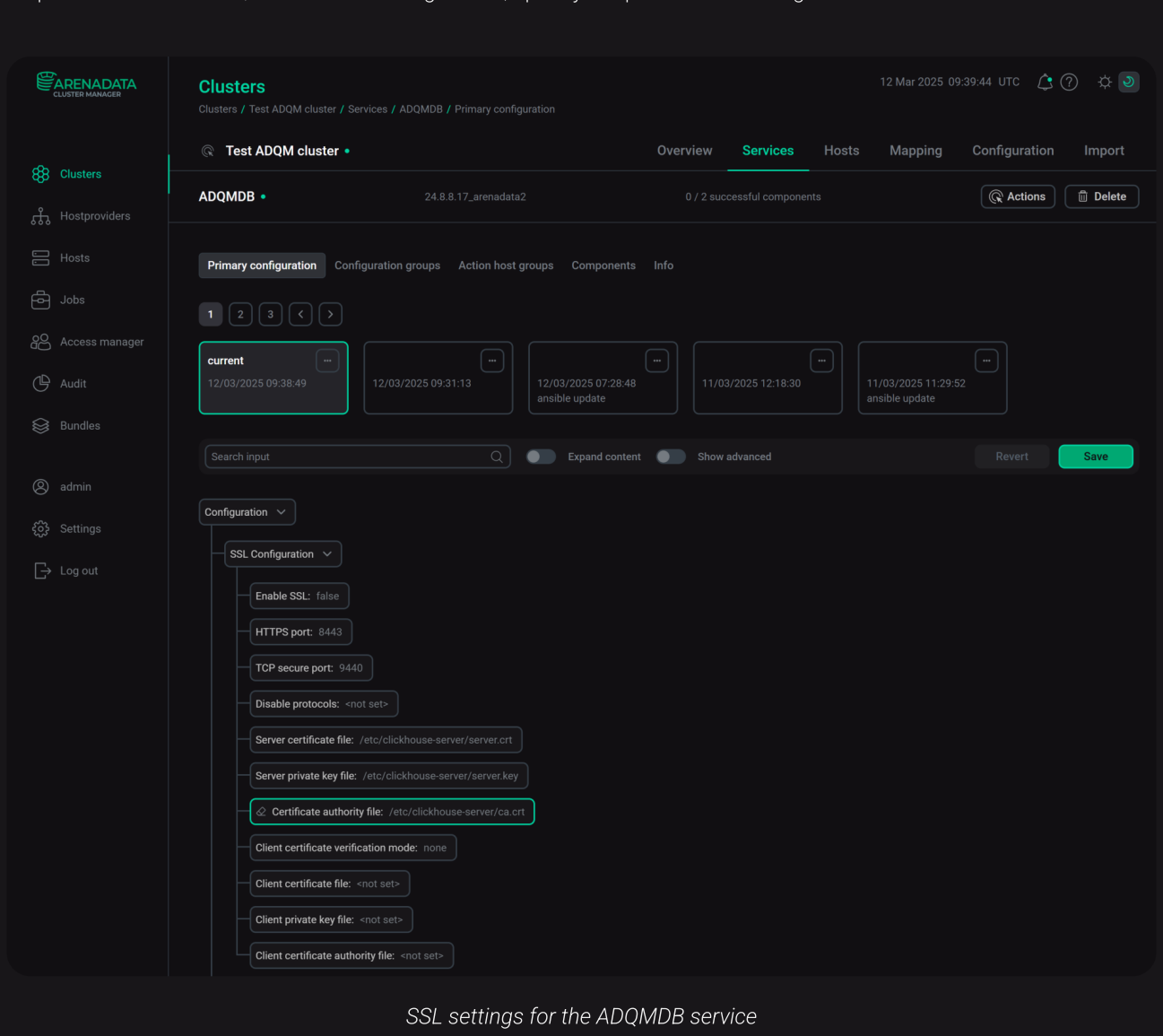
[Prometheus] -> Enable SSL: false
[Prometheus] -> Certificate file: /etc/admprom/prometheus/server.crt
[Prometheus] -> Private key file: /etc/admprom/prometheus/server.key
[Prometheus] -> Certificate authority file: /etc/admprom/prometheus/ca.crt
[Grafana] -> Enable SSL: false
[Grafana] -> Certificate file: /etc/admprom/grafana/server.crt
[Grafana] -> Private key file: /etc/admprom/grafana/server.key
[Grafana] -> Certificate authority file: /etc/admprom/grafana/ca.crt
[Node-exporter] -> Enable SSL: false
[Node-exporter] -> Certificate file: /etc/ssl/server.crt
[Node-exporter] -> Private key file: /etc/ssl/server.key
[Pushgateway] -> Enable SSL: false
[Pushgateway] -> Certificate file: /etc/admprom/pushgateway/server.crt
[Pushgateway] -> Private key file: /etc/admprom/pushgateway/server.key

Configuration parameters of the Monitoring service for SSL connections

3. After specifying all the necessary parameters to configure SSL connections, click ***Save*** and run an action that applies the changes to the service configuration.

Example

1. In the ***Certificate authority file*** parameter of the ADQMDB service configuration, specify the path to the self-signed CA certificate file — */etc/clickhouse-server/ca.crt*.

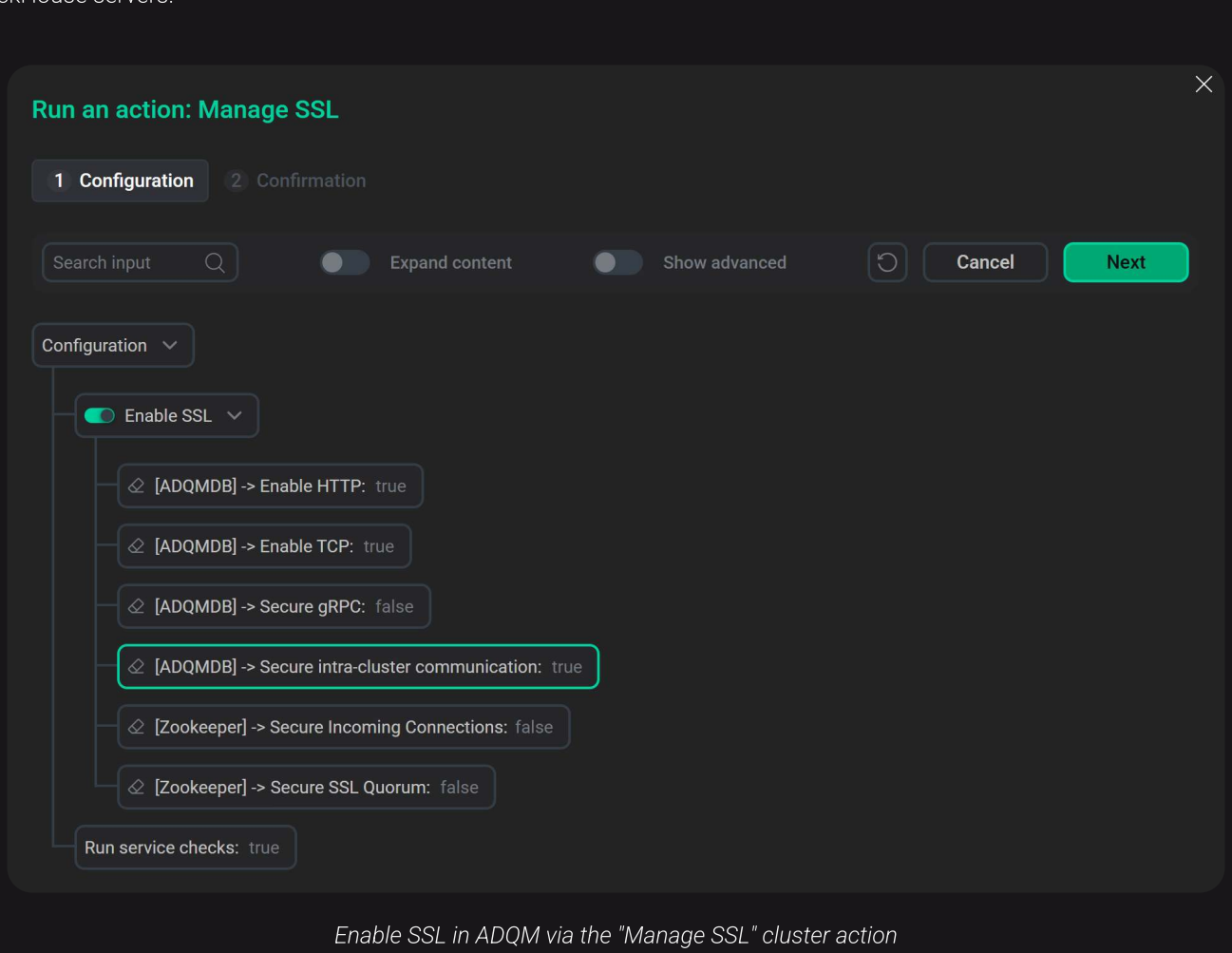

ARENADATA CLUSTER MANAGER
Clusters
Clusters / Test ADQM cluster / Services / ADQMDB / Primary configuration
12 Mar 2025 09:39:44 UTC
Test ADQM cluster
Overview
Services
Hosts
Mapping
Configuration
Import
ADQMDB
24.8.8.17_arenadata2
0 / 2 successful components
Actions
Delete
Primary configuration
Configuration groups
Action host groups
Components
Info
current
Search input
Expand content
Show advanced
Revert
Save
Configuration
SSL Configuration

Enable SSL: false
HTTPS port: 8443
TCP secure port: 9440
Disable protocols: <not set>
Server certificate file: /etc/clickhouse-server/server.crt
Server private key file: /etc/clickhouse-server/server.key
Certificate authority file: /etc/clickhouse-server/ca.crt
Client certificate verification mode: none
Client certificate file: <not set>
Client private key file: <not set>
Client certificate authority file: <not set>

SSL settings for the ADQMDB service

   Click ***Save***.

2. Run the ***Manage SSL*** action for the cluster. In the configuration window that appears, activate ***Enable SSL*** and also set ***Secure intra-cluster communication*** to `true` to encrypt internal traffic between ClickHouse servers.


Run an action: Manage SSL
1 Configuration
2 Confirmation
Search input
Expand content
Show advanced
Cancel
Next
Configuration
Enable SSL

[ADQMDB] -> Enable HTTP: true
[ADQMDB] -> Enable TCP: true
[ADQMDB] -> Secure gRPC: false
[ADQMDB] -> Secure intra-cluster communication: true
[Zookeeper] -> Secure Incoming Connections: false
[Zookeeper] -> Secure SSL Quorum: false
Run service checks: true

Enable SSL in ADQM via the "Manage SSL" cluster action

   Click ***Next*** and then ***Run*** to start the action execution.

Test connection

Connect via clickhouse-client

To ensure that the connection to ADQM is secured with the SSL encryption, you can connect to any server of the ADQM cluster via clickhouse-client with the `--secure` flag, specifying the TCP port for secure connection over SSL (`9440`). For example:

```
$ clickhouse-client --host adqm-host-1.arenadata.local --secure --port 9440 --user default
```

The output confirms the connection on the `9440` secure port:

```
ClickHouse client version 24.8.8.17.
Connecting to adqm-host-1.arenadata.local:9440 as user default.
Connected to ClickHouse server version 24.8.8.
```

Note that the ***Client certificate verification mode*** option in the ***SSL Configuration*** section of the ADQMDB service configuration should be disabled (set to `none`) as this test connection does not use the SSL certificate user authentication.

Check interserver connections

Run a distributed query, for example, that creates a new ADQM user on each host:

```
CREATE USER test_user ON CLUSTER default_cluster IDENTIFIED WITH sha256_password BY 'qwerty';
```

The query output shows that the communication between ADQM hosts also occurs via the `9440` port:

```
   ┌─host────────────────────────┬─port─┬─status─┬─error─┬─num_hosts_remaining─┬─num_hosts_active─┐
1. │ adqm-host-4.arenadata.local │ 9440 │      0 │       │                   3 │                0 │
2. │ adqm-host-1.arenadata.local │ 9440 │      0 │       │                   2 │                0 │
3. │ adqm-host-2.arenadata.local │ 9440 │      0 │       │                   1 │                0 │
4. │ adqm-host-3.arenadata.local │ 9440 │      0 │       │                   0 │                0 │
   └─────────────────────────────┴──────┴────────┴───────┴─────────────────────┴──────────────────┘
```

Connect via HTTPS

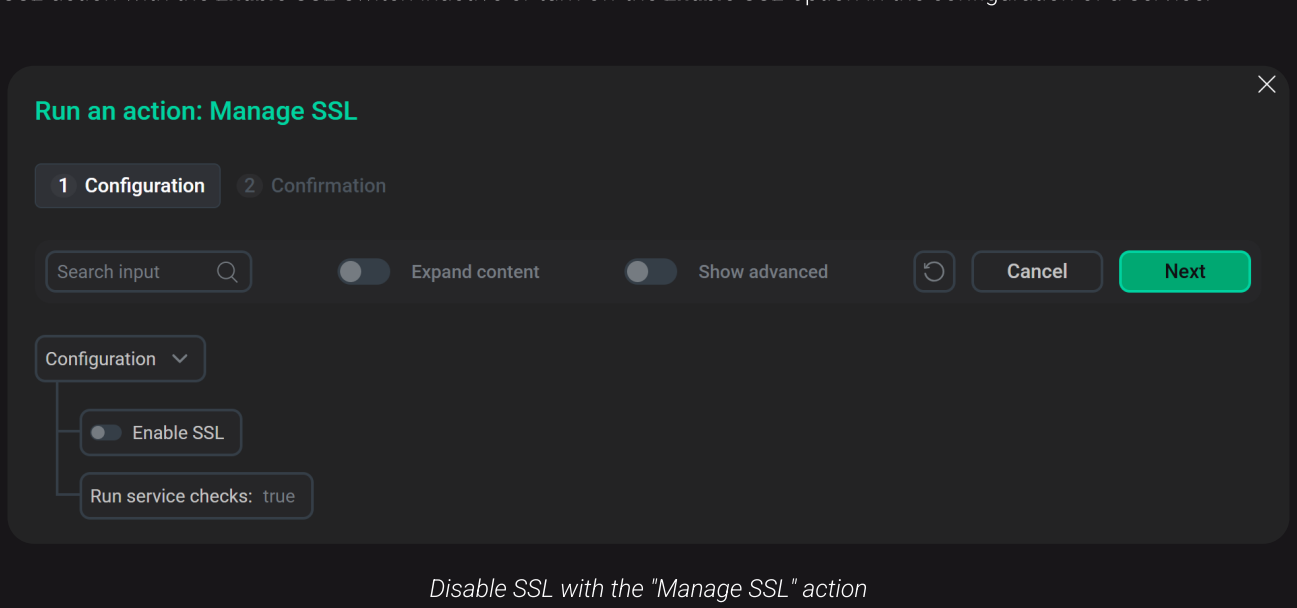
To test the connection to ADQM over HTTPS, you can run the following command on the cluster host:

```
$ echo 'SHOW TABLES' | curl 'https://adqm-host-1.arenadata.local:8443/?query=' --data-binary @- --cacert arenadata_ca.crt
```

If the connection is successful, the query will return a list of tables in the `default` database on the *adqm-host-1.arenadata.local* host.

Disable SSL

To disable SSL, run the ***Manage SSL*** action with the ***Enable SSL*** switch inactive or turn off the ***Enable SSL*** option in the configuration of a service.


Run an action: Manage SSL
1 Configuration
2 Confirmation
Search input
Expand content
Show advanced
Cancel
Next
Configuration
Enable SSL

Run service checks: true

Disable SSL with the "Manage SSL" action

Cluster actions

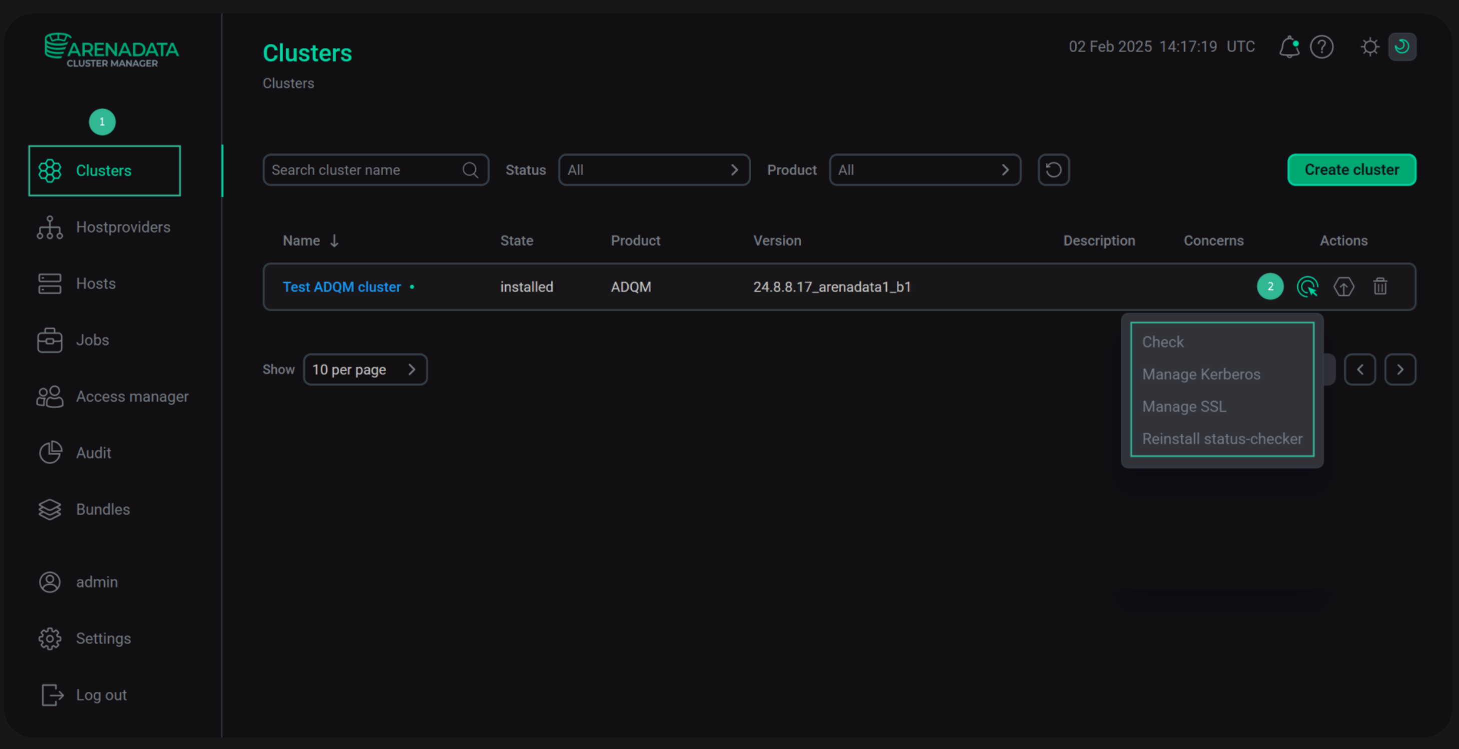
Elena Dvoryadkina

Contents

To Table of Contents

Manage a cluster
Upgrade a cluster

You can manage an ADQM cluster on the ***Clusters*** page in the ADCM UI.


ARENADATA
CLUSTER MANAGER
1
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Clusters
Clusters
02 Feb 2025 14:17:19 UTC
Search cluster name
Status
All
Product
All
Create cluster
Show
10 per page
Check
Manage Kerberos
Manage SSL
Reinstall status-checker

Name	State	Product	Version	Description	Concerns	Actions
Test ADQM cluster	installed	ADQM	24.8.8.17_arenadata1_b1		2	

ADQM cluster in the ADCM UI

Refer to ADCM documentation for an overview of the ***Clusters*** page and common actions.

Manage a cluster

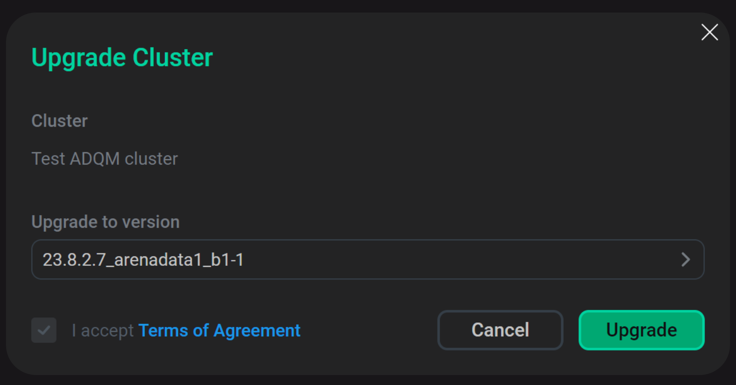
Click the icon to open a drop-down list with available actions and select an action to run it. A set of available actions depends on the cluster state:

- For a new cluster (the cluster state is `created`), an available action is:
  - ***Install*** — installs all cluster services.
- After a cluster is installed (the cluster state is `installed`), the following actions are available:
  - ***Check*** — checks if all components of the installed cluster work correctly.
  - ***Manage Kerberos*** — allows you to enable, reconfigure, and disable Kerberos authentication.
  - ***Manage SSL*** — allows you to configure and enable/disable SSL encryption for services of the ADQM cluster.
  - ***Reinstall status-checker*** — reconfigures and restarts the status checker for all cluster services. Use this action when migrating a cluster to a new ADCM server.
- For a cluster prepared for upgrade (the cluster state is `upgradable`), an available action is:
  - ***Upgrade*** — upgrades the cluster version.

Upgrade a cluster

To upgrade an ADQM cluster version, you need to upload a new bundle first — see the step ***Upload a cluster bundle to ADCM*** in the Create a cluster article.

After a new bundle is loaded, open the ***Clusters*** page. The icon becomes active for the ADQM cluster in the ***Actions*** column. Click this icon — in the ***Upgrade Cluster*** window that opens, select a version to upgrade ADQM and click ***Upgrade***.

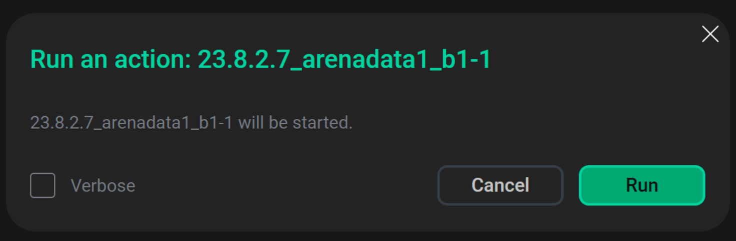

Upgrade Cluster
☑ I accept Terms of Agreement
Cancel
Upgrade

Cluster
Test ADQM cluster

Upgrade to version
23.8.2.7_arenadata1_b1-1

Start upgrade

In the confirmation dialog window that appears, click ***Run***.

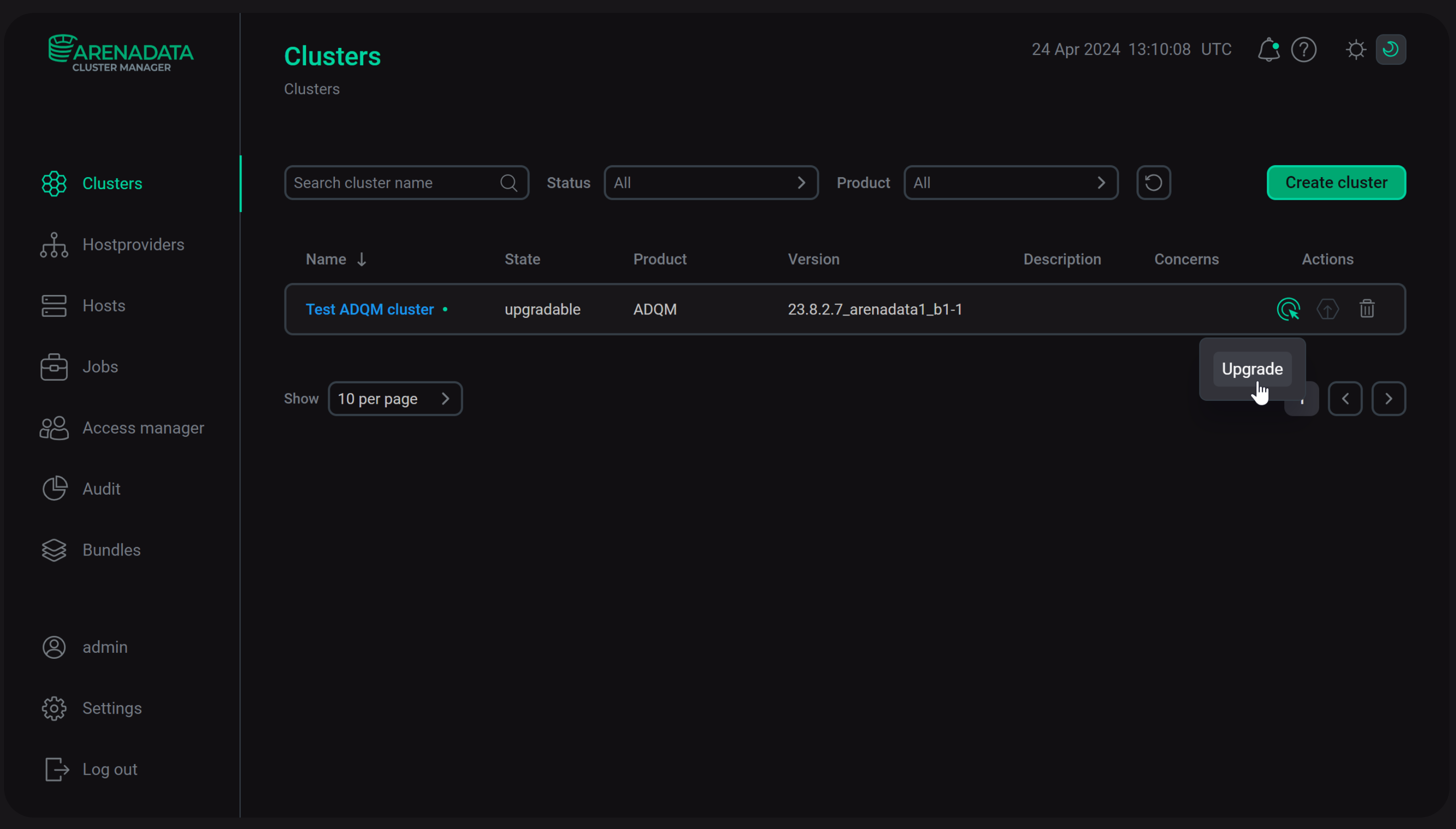

☐ Verbose
Cancel
Run

Run an action: 23.8.2.7_arenadata1_b1-1

23.8.2.7_arenadata1_b1-1 will be started.

Upgrade confirmation

As a result, ADCM changes the cluster state to `upgradable` — the cluster is ready for upgrade. Click the icon in the ***Actions*** column and select the ***Upgrade*** action.


ARENADATA
CLUSTER MANAGER
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Clusters
Clusters
24 Apr 2024 13:10:08 UTC
Search cluster name
Status
All
Product
All
Create cluster
Show
10 per page
Upgrade

Name	State	Product	Version	Description	Concerns	Actions
Test ADQM cluster	upgradable	ADQM	23.8.2.7_arenadata1_b1-1			

Run the upgrade action

In the dialog that opens, click ***Run*** to confirm the upgrade. Once the upgrade process is complete, ADCM will change the cluster state to `installed`.

NOTE

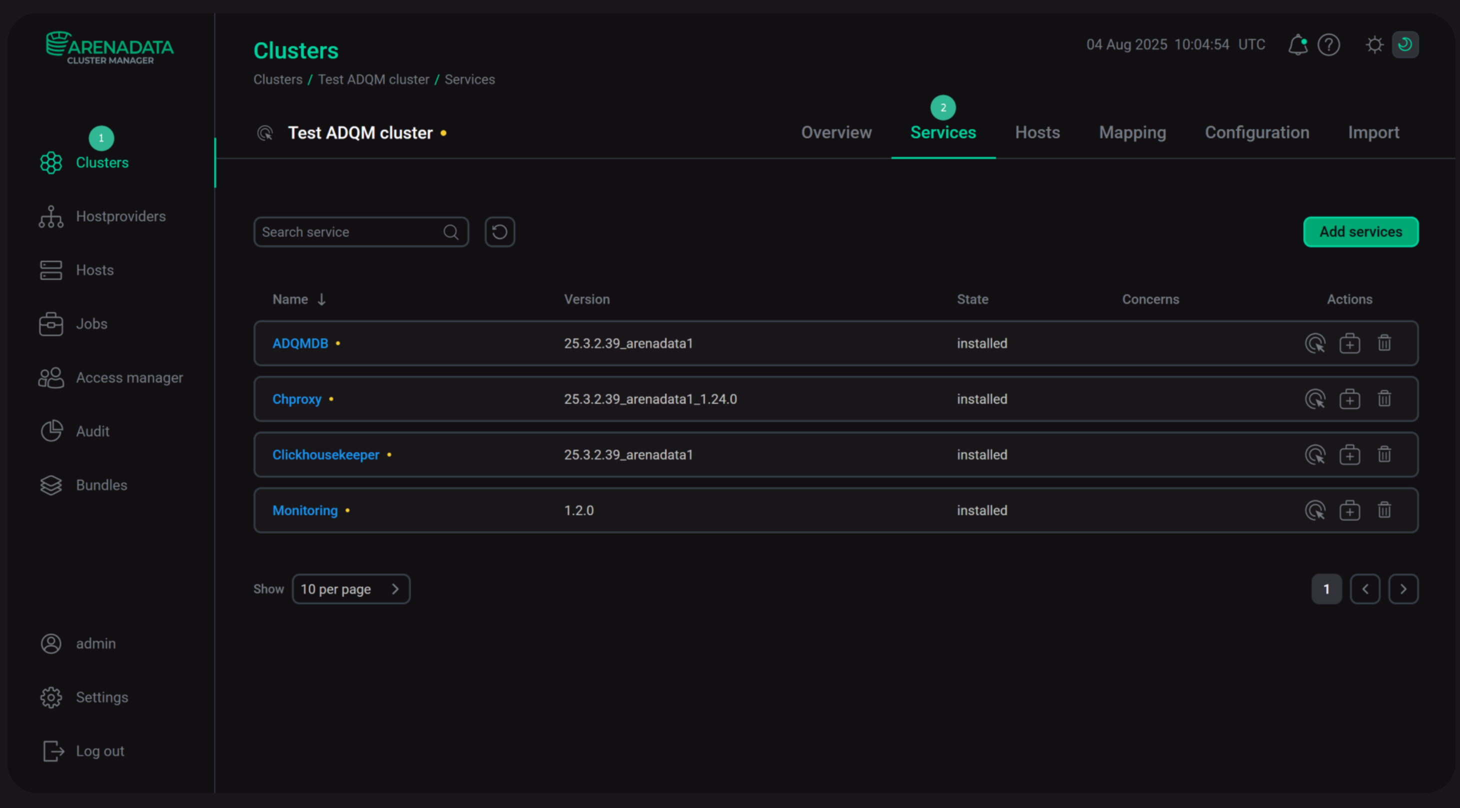
When upgrading ADQM from the version 23.3.2.37.1.b1 (minimum) to 24.3.2.23.2.b1 and higher in ADCM starting with the version 2.3, a cluster prepared for the upgrade has the `upgrading` status with two available actions — ***Complete the upgrade*** and ***Abort the upgrade***. You can upgrade such cluster without stopping it completely by sequentially upgrading the ADQMDB service on host groups and checking the correctness of the service's new version at intermediate stages of the cluster upgrade. If you detect any problems with the new version of ClickHouse, you can revert to the previous version before completing the cluster upgrade. See details of using this functionality in the Rolling upgrade article.

Service actions

Elena Dvoryadkina

To Table of Contents

ADCM UI provides actions to manage ADQM services. These actions are available on the ***Services*** tab. To open this tab, click an ADQM cluster name on the ***Clusters*** page and switch to the ***Services*** tab.



Clusters

Clusters / Test ADQM cluster / Services

Test ADQM cluster

Name	Version	State	Concerns	Actions
ADQMDB	25.3.2.39_arenadata1	installed		
Chproxy	25.3.2.39_arenadata1_1.24.0	installed		
Clickhousekeeper	25.3.2.39_arenadata1	installed		
Monitoring	1.2.0	installed		

ARENADATA CLUSTER MANAGER
04 Aug 2025 10:04:54 UTC
1
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
2
Overview
Services
Hosts
Mapping
Configuration
Import
Search service
Add services
Show 10 per page
1

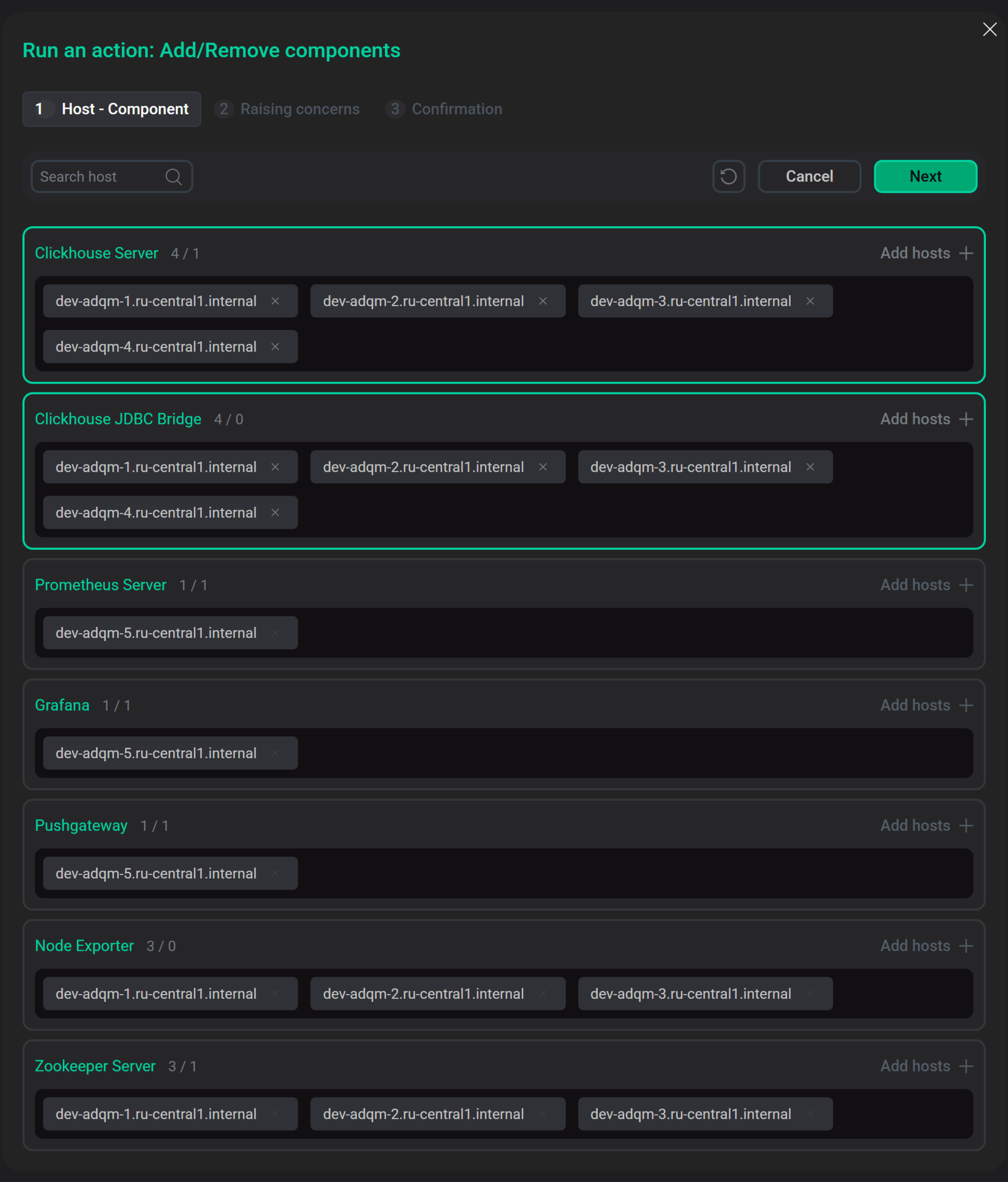
ADQM cluster services in the ADCM interface

Refer to ADCM documentation for an overview of the ***Services*** tab and common actions. A set of actions available for working with a service of an ADQM cluster depends on the service type.

Actions available for all services of ADQM:

- ***Install*** (if a service state is `created`) — installs a service.
- ***Add/Remove components*** — opens the component-host mapping interface where you can add, remove, and redistribute service components.

ADQMDB



Run an action: Add/Remove components

Clickhouse Server 4 / 1: dev-adqm-1.ru-central1.internal, dev-adqm-2.ru-central1.internal, dev-adqm-3.ru-central1.internal, dev-adqm-4.ru-central1.internal

Clickhouse JDBC Bridge 4 / 0: dev-adqm-1.ru-central1.internal, dev-adqm-2.ru-central1.internal, dev-adqm-3.ru-central1.internal, dev-adqm-4.ru-central1.internal

Prometheus Server 1 / 1: dev-adqm-5.ru-central1.internal

Grafana 1 / 1: dev-adqm-5.ru-central1.internal

Pushgateway 1 / 1: dev-adqm-5.ru-central1.internal

Node Exporter 3 / 0: dev-adqm-1.ru-central1.internal, dev-adqm-2.ru-central1.internal, dev-adqm-3.ru-central1.internal

Zookeeper Server 3 / 1: dev-adqm-1.ru-central1.internal, dev-adqm-2.ru-central1.internal, dev-adqm-3.ru-central1.internal

1 Host - Component
2 Raising concerns
3 Confirmation
Search host
Cancel
Next
Add hosts

The component-host mapping interface for the ADQMDB service

- ***Check*** — checks whether the settings of hosts, components, and related services match the requirements of a service, and whether the service itself works correctly.
- ***Start*** — starts a service.
- ***Stop*** — stops a service.

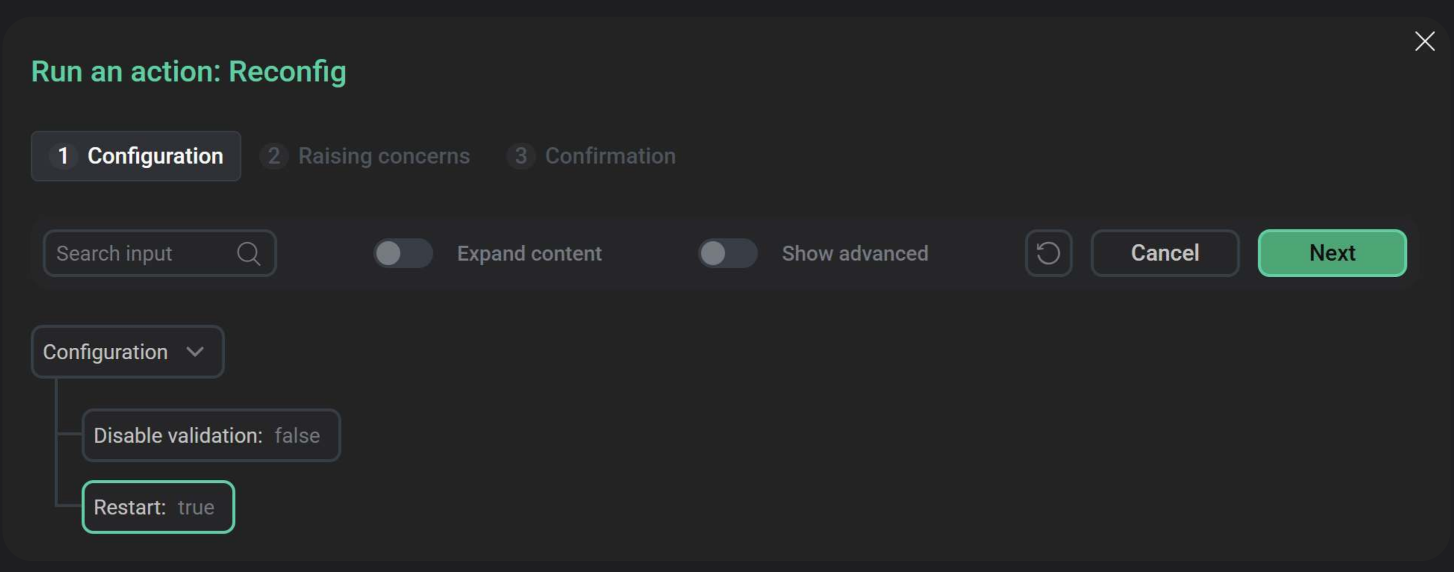
Service-specific actions:

- ***Reconfig*** (ADQMDB, Clickhousekeeper, Chproxy) — updates a service configuration according to settings specified on the service configuration page.

ADQMDB

On the ***Configuration*** tab of the ***Run an action*** window that opens when you select the action for the ADQMDB service, you can activate the options:

- ***Disable validation*** — to switch off the validation of the service configuration (in some cases, the configuration can be considered incorrect — for example, if it deletes storage policies that are still in use);
- ***Restart*** — to restart the service (applying changes to some parameters does not require restarting the service — in this case, it is enough to reload the configuration).



Run an action: Reconfig

Configuration

Disable validation: false

Restart: true

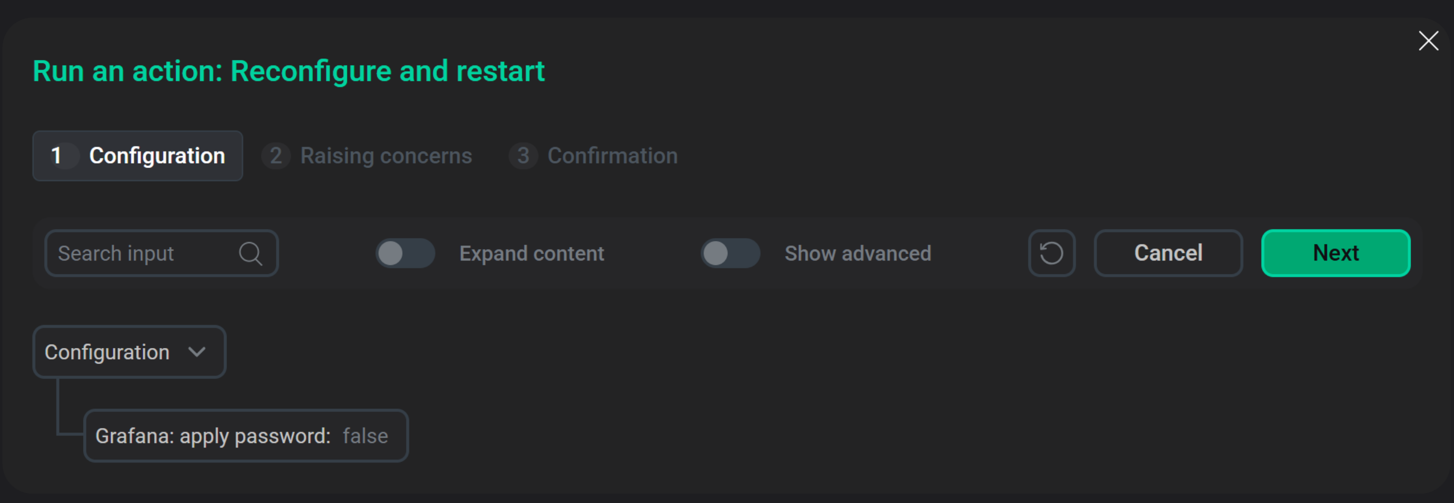
1 Configuration
2 Raising concerns
3 Confirmation
Search input
Expand content
Show advanced
Cancel
Next

Set up the "Reconfig" action for the ADQMDB service

- ***Restart*** (ADQMDB, Clickhousekeeper, Chproxy) — stops and then starts a service.
- ***Reconfig and restart*** (Zookeeper, Monitoring) — updates a service configuration according to the settings specified on the service configuration page, and restarts the service.

Monitoring

In the ***Run an action*** window that opens when you select the action for the Monitoring service, the ***Grafana: apply password*** parameter specifies whether the Grafana administrator password should be reset using the value of the ***Grafana settings → Grafana administrator's password*** parameter located on the ***Primary configuration*** tab of the Monitoring service configuration page.



Run an action: Reconfigure and restart

Configuration

Grafana: apply password: false

1 Configuration
2 Raising concerns
3 Confirmation
Search input
Expand content
Show advanced
Cancel
Next

Set up the "Reconfigure and restart" action for the Monitoring service

- ***Manage auto core dump*** (ADQMDB) — enables/disables an automatic core dump.
- ***Reinstall*** (Monitoring) — reinstalls a service.

Configure logical clusters in the ADCM interface

Elena Dvoryadkina

Contents

To Table of Contents

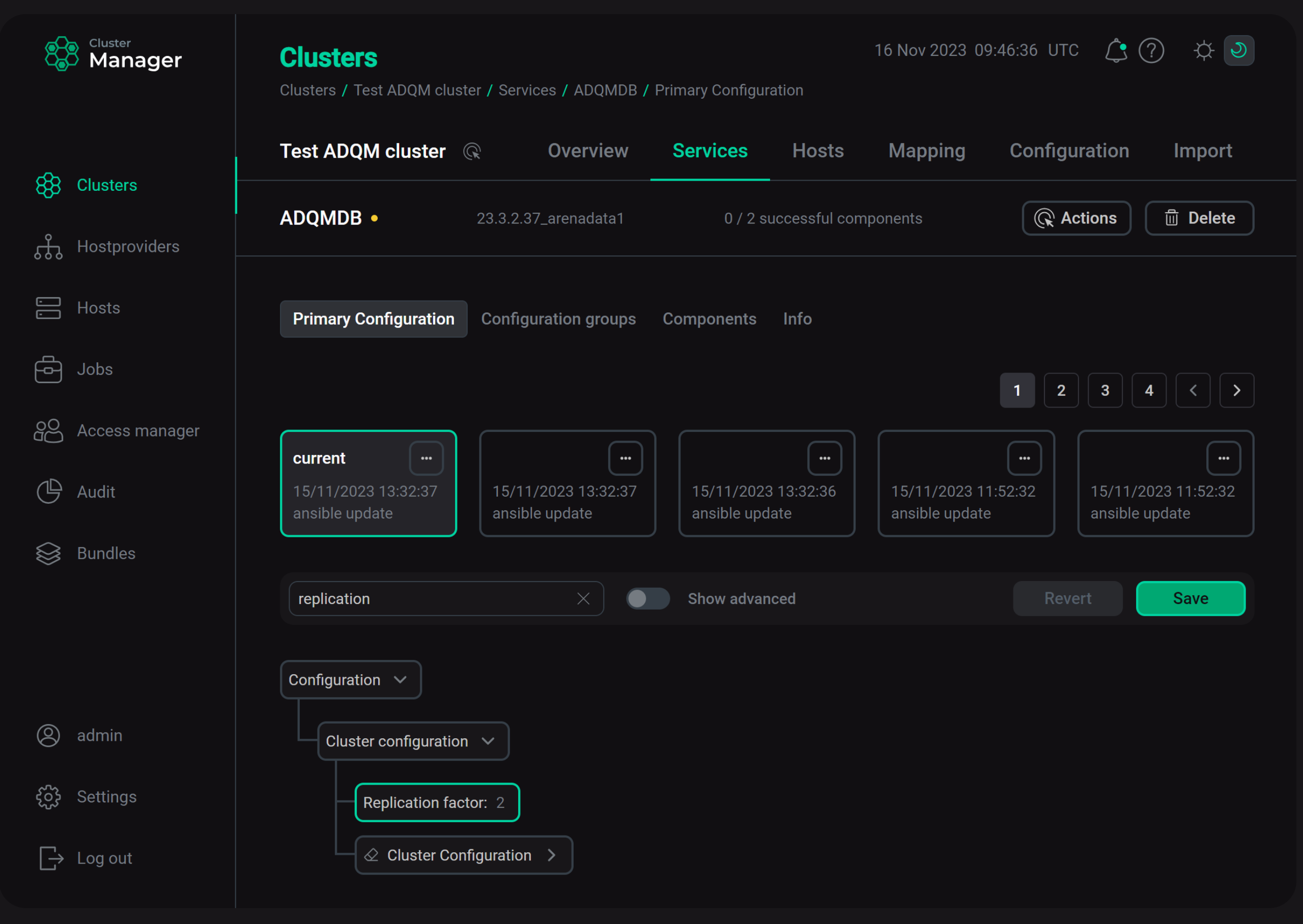
Auto generate logical cluster on ADQM installation

Add more logical clusters

To organize distributed storage and replication of data, as well as to allow executing distributed queries (`ON CLUSTER`) on multiple ADQM hosts, it is necessary to combine hosts into a logical cluster. You can configure a logical cluster within an ADQM cluster via the ADCM interface — use parameters in the ***Cluster configuration*** section on the configuration page of the ADQMDB service.

Auto generate logical cluster on ADQM installation

Before you install an ADQM cluster, specify the ***Replication factor*** parameter value to automatically group hosts into a logical cluster.



Cluster Manager

Clusters

16 Nov 2023 09:46:36 UTC

Clusters / Test ADQM cluster / Services / ADQMDB / Primary Configuration

Test ADQM cluster — Overview | Services | Hosts | Mapping | Configuration | Import

ADQMDB • 23.3.2.37_arenadata1 — 0 / 2 successful components — Actions | Delete

Primary Configuration | Configuration groups | Components | Info

current — 15/11/2023 13:32:37 ansible update

15/11/2023 13:32:37 ansible update

15/11/2023 13:32:36 ansible update

15/11/2023 11:52:32 ansible update

15/11/2023 11:52:32 ansible update

replication — Show advanced — Revert — Save

Configuration
  Cluster configuration
    Replication factor: 2
    Cluster Configuration

Set the replication factor

This will generate the cluster topology as follows:

- All ADQM hosts are split into groups — shards.
- Each host in a shard is a replica. The number of replicas in a shard corresponds to the value of the ***Replication factor*** parameter.
- If the total number of hosts is not exactly divisible by the specified replication factor, then the number of replicas in the last shard is determined as the remainder of the division.

After the ADQM cluster is installed, the `default_cluster` logical cluster will be added to the `remote_servers` section of the *config.xml* configuration file. This topology is created once and does not change when the cluster is further reconfigured.

For example, if an ADQM cluster contains 4 hosts (*host-1*, *host-2*, *host-3*, *host-4*) and the ***Replication factor*** parameter is set to `2`, a logical cluster generated based on the specified replication factor will include two shards with two replica hosts in each one:

```
<remote_servers>
    <default_cluster>
        <shard>
            <internal_replication>true</internal_replication>
            <weight>1</weight>
            <replica>
                <host>host-1</host>
                <port>9000</port>
            </replica>
            <replica>
                <host>host-2</host>
                <port>9000</port>
            </replica>
        </shard>
        <shard>
            <internal_replication>true</internal_replication>
            <weight>1</weight>
            <replica>
                <host>host-3</host>
                <port>9000</port>
            </replica>
            <replica>
                <host>host-4</host>
                <port>9000</port>
            </replica>
        </shard>
    </default_cluster>
</remote_servers>
```

Information on `default_cluster` will be added to the system.clusters system table:

```
SELECT cluster, shard_num, replica_num, host_name FROM system.clusters;
```

```
┌─cluster─────────┬─shard_num─┬─replica_num─┬─host_name─┐
│ default_cluster │         1 │           1 │ host-1    │
│ default_cluster │         1 │           2 │ host-2    │
│ default_cluster │         2 │           1 │ host-3    │
│ default_cluster │         2 │           2 │ host-4    │
└─────────────────┴───────────┴─────────────┴───────────┘
```

When creating a logical cluster, the `macros` section with shard and replica identifiers for the host is also added to the *config.xml* configuration file on each host. For example, macros for *host-4* are:

```
<macros>
    <replica>2</replica>
    <shard>2</shard>
</macros>
```

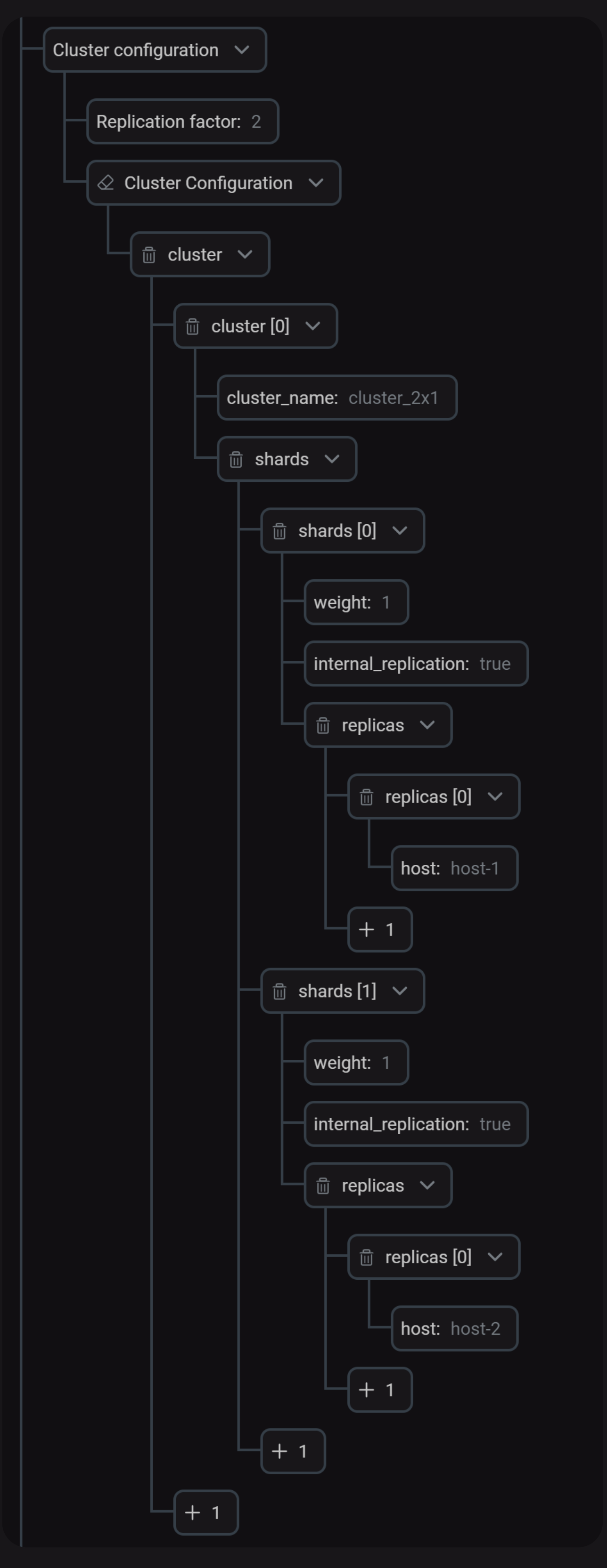
Macros are used to automatically substitute host-specific shard and replica identifiers when creating replicated tables on a cluster (for an example, see the Typical cluster article).

Add more logical clusters

ADQM hosts can be combined into multiple logical clusters with different topologies (one host can participate in several logical clusters).

To configure a logical cluster, use the ***Cluster Configuration*** parameter — add necessary elements using the icon `+ 1`:

1. Add a new cluster and specify its name.
2. Specify how many shards the cluster should contain — add items to the ***shards*** list.
   For each shard, specify the ***weight*** and ***internal_replication*** parameters (see the concept:architecture/sharding.adoc#configuration article for more information on these parameters), and add hosts to be replicas within the shard to the ***replicas*** list.



Cluster configuration
  Replication factor: 2
  Cluster Configuration
    cluster
      cluster [0]
        cluster_name: cluster_2x1
        shards
          shards [0]
            weight: 1
            internal_replication: true
            replicas
              replicas [0]
                host: host-1
              + 1
          shards [1]
            weight: 1
            internal_replication: true
            replicas
              replicas [0]
                host: host-2
              + 1
          + 1
      + 1

Configure logical cluster

3. Click ***Save*** and execute the ***Reconfig*** action for the ADQMDB service to save the settings.
   The cluster definition will appear in the `remote_servers` section of the *config.xml* file:

```
<remote_servers>
    <!-- default_cluster configuration is here -->

    <cluster_2x1>
        <shard>
            <internal_replication>true</internal_replication>
            <weight>1</weight>
            <replica>
                <host>host-1</host>
                <port>9000</port>
            </replica>
        </shard>
        <shard>
            <internal_replication>true</internal_replication>
            <weight>1</weight>
            <replica>
                <host>host-2</host>
                <port>9000</port>
            </replica>
        </shard>
    </cluster_2x1>
</remote_servers>
```

   Information on the cluster will be also added to the system.clusters table:

```
SELECT cluster, shard_num, replica_num, host_name FROM system.clusters;
```

```
┌─cluster─────────┬─shard_num─┬─replica_num─┬─host_name─┐
│ cluster_2x1     │         1 │           1 │ host-1    │
│ cluster_2x1     │         2 │           1 │ host-2    │
│ default_cluster │         1 │           1 │ host-1    │
│ default_cluster │         1 │           2 │ host-2    │
│ default_cluster │         2 │           1 │ host-3    │
│ default_cluster │         2 │           2 │ host-4    │
└─────────────────┴───────────┴─────────────┴───────────┘
```

Macros for all clusters except `default_cluster` are prefixed with `<cluster_name>_`. For example, for the `cluster_2x1` cluster configured above, the following macros will be added to the *config.xml* configuration file on *host-1*:

```
<macros>
    <!-- default_cluster macros are here -->

    <cluster_2x1_replica>1</cluster_2x1_replica>
    <cluster_2x1_shard>1</cluster_2x1_shard>
</macros>
```

Rolling upgrade

Elena Dvoryadkina

Contents

To Table of Contents

Step 1. Prepare for the update

Step 2. Update ADQMDB on host groups

Step 3. Complete the update

The **Rolling upgrade** functionality allows you to sequentially upgrade the ADQMDB service on separate ADQM hosts without a long shutdown of a cluster.

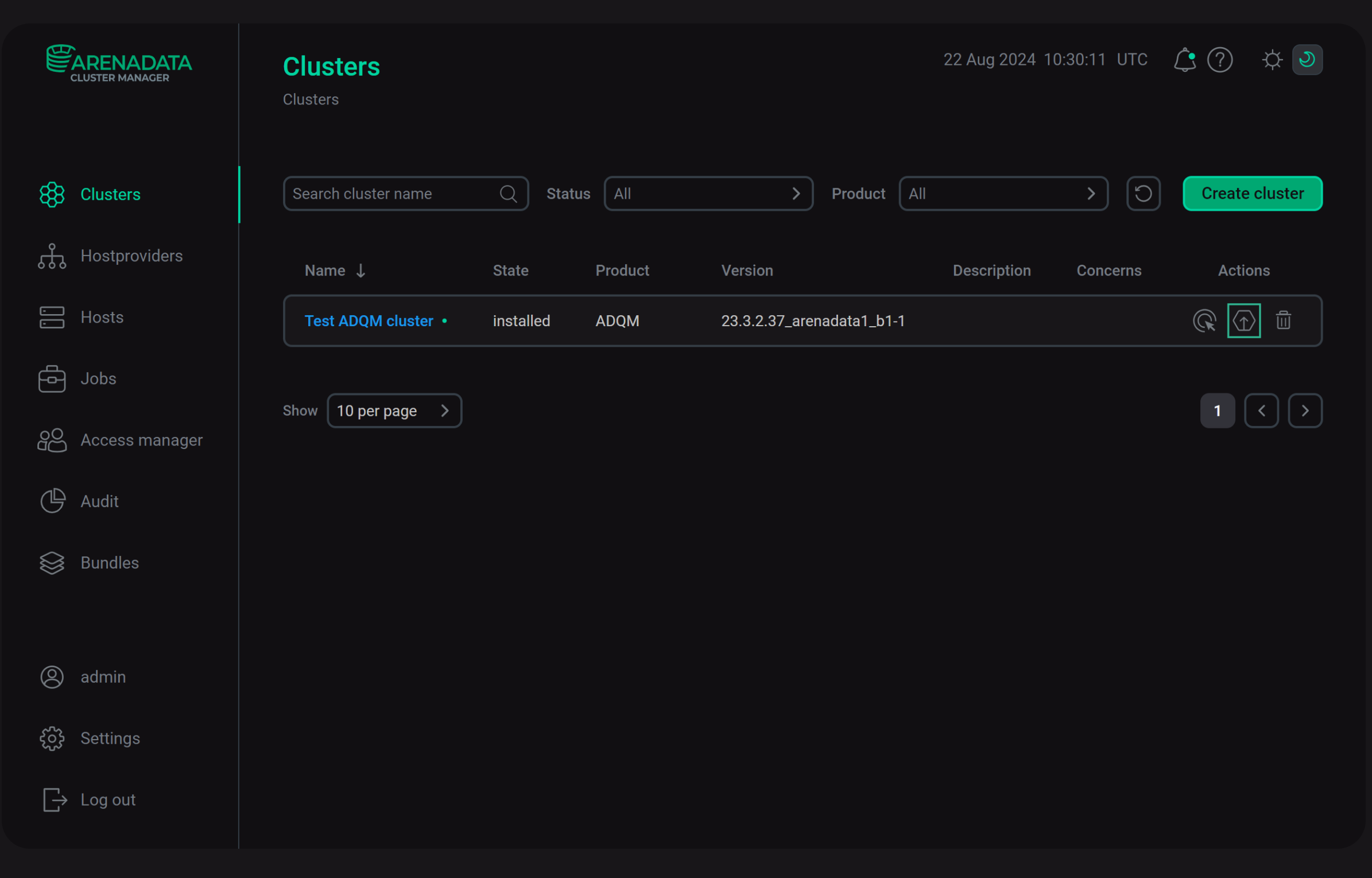
IMPORTANT

- Rolling upgrade is available for upgrading an ADQM version from 23.3.2.37.1.b1 (minimum) to 24.3.2.23.2.b1 (and higher) in ADCM starting with version 2.3.
- Rolling upgrade can only be applied to ADQM clusters in the `installed` state.

The process of ADQM rolling upgrade via the ADCM interface is described below.

Step 1. Prepare for the update

1. On the ***Bundles*** page, upload a new ADQM bundle — see the step ***Upload a cluster bundle to ADCM*** in the Create a cluster article.
2. Open the ***Clusters*** page, where the icon ⇧ becomes active for an ADQM cluster in the ***Actions*** column after a new bundle is loaded — click this icon.

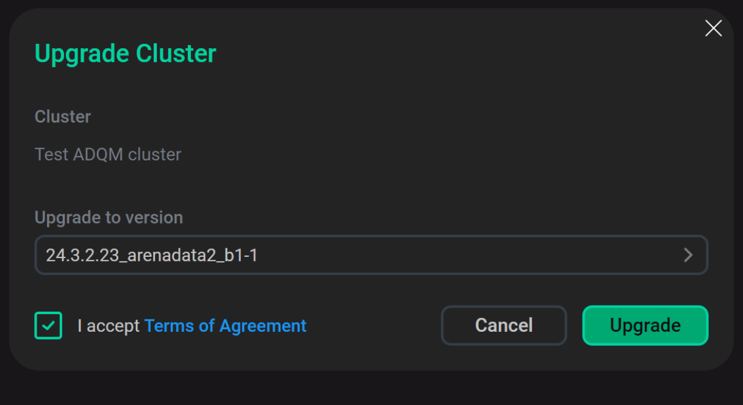


ARENADATA CLUSTER MANAGER
Clusters
22 Aug 2024 10:30:11 UTC
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Search cluster name
Status All
Product All
Create cluster

Name	State	Product	Version	Description	Concerns	Actions
Test ADQM cluster	installed	ADQM	23.3.2.37_arenadata1_b1-1			

Preparing a cluster for the upgrade

In the ***Upgrade Cluster*** window that opens, select a version you want to update ADQM to and click ***Upgrade***.



Upgrade Cluster
Cancel
Upgrade

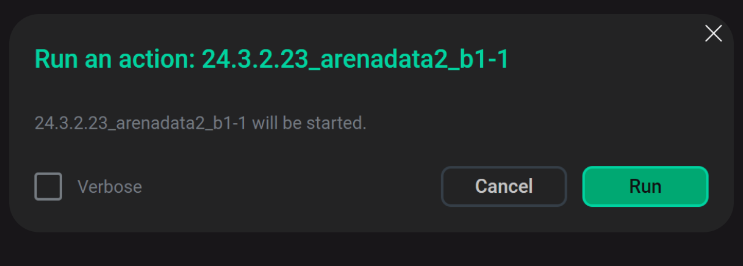
Cluster: Test ADQM cluster

Upgrade to version: 24.3.2.23_arenadata2_b1-1

☑ I accept Terms of Agreement

Selecting a version to update ADQM

In the confirmation window that appears, click ***Run***.



Run an action: 24.3.2.23_arenadata2_b1-1

24.3.2.23_arenadata2_b1-1 will be started.

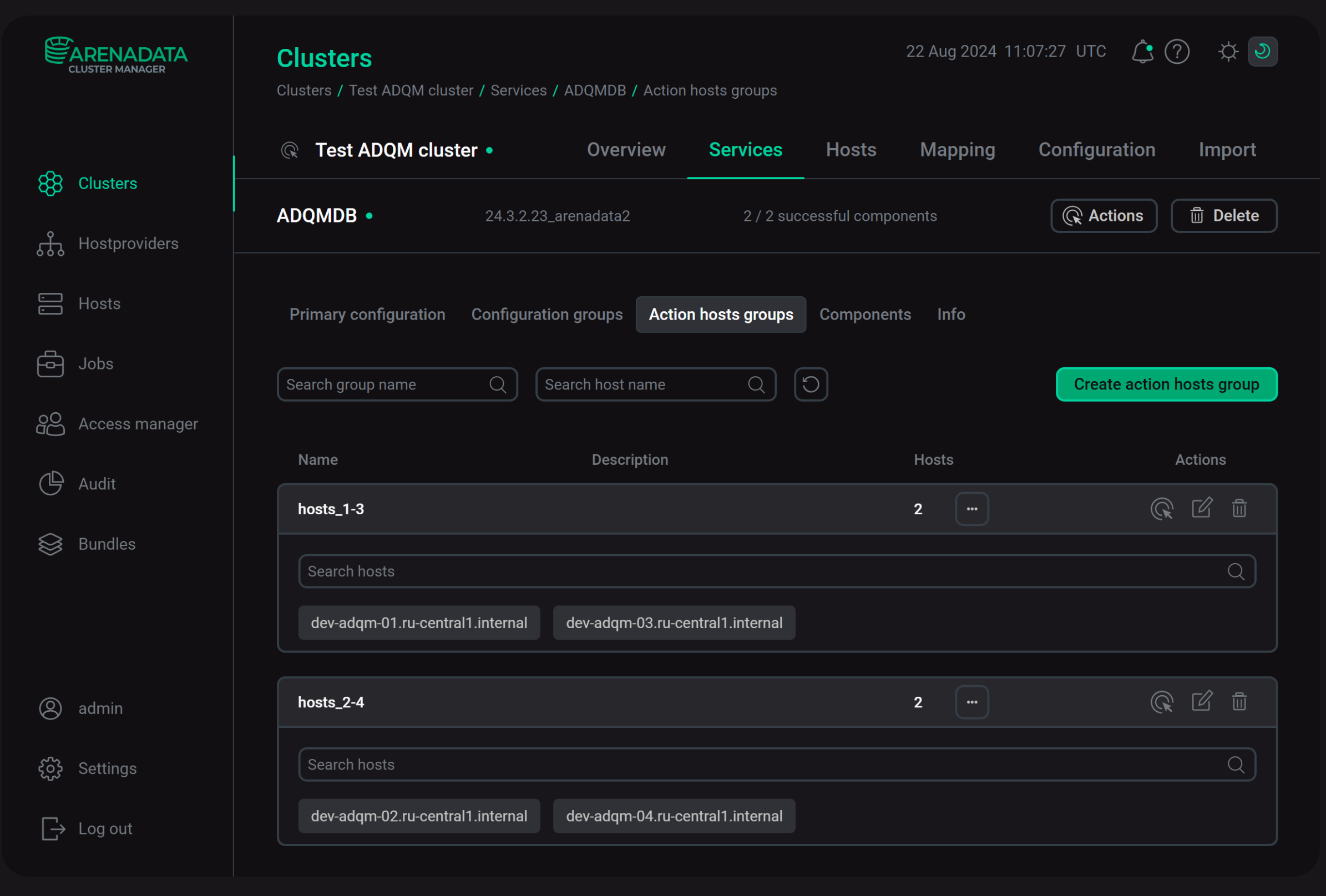
☐ Verbose

Confirming preparation for the update

When the action is completed, ADCM will change the cluster status to `upgrading` — the cluster is ready to be upgraded.

Step 2. Update ADQMDB on host groups

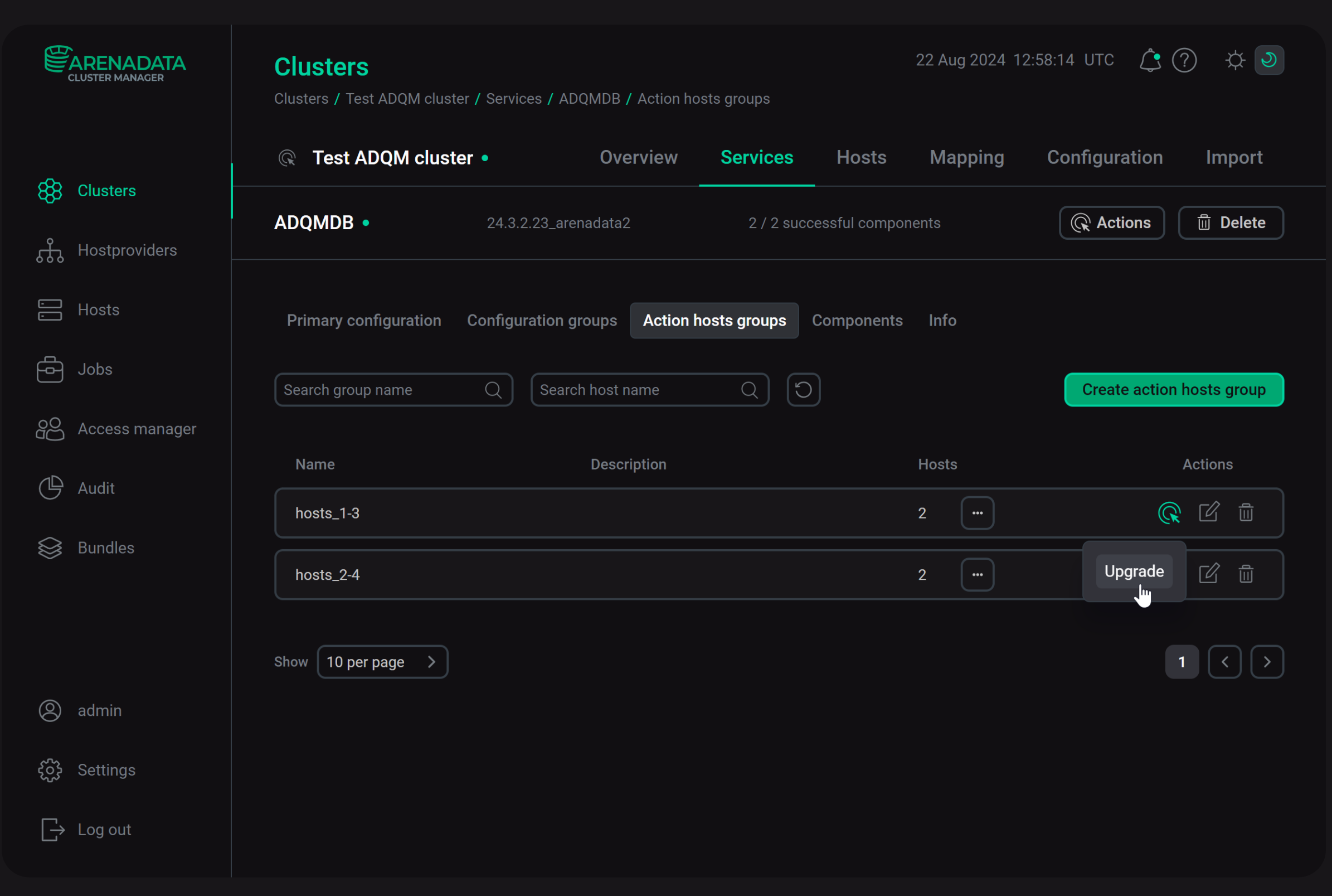
1. On the ADQMDB service configuration page (***Clusters*** → ***<ADQM_cluster>*** → ***Services*** → ***ADQMDB***), open the ***Action hosts groups*** tab and combine cluster hosts into groups to then upgrade the ADQMDB version for each host group sequentially. For example, you can create host groups so that each group contains one replica of each shard. The image below shows an example of dividing hosts into groups according to this principle for the following cluster topology:
   - shard 1 with two replicas — *dev-adqm-01.ru-central1.internal* and *dev-adqm-02.ru-central1.internal*;
   - shard 2 with two replicas — *dev-adqm-03.ru-central1.internal* and *dev-adqm-04.ru-central1.internal*.



ARENADATA CLUSTER MANAGER
Clusters
Clusters / Test ADQM cluster / Services / ADQMDB / Action hosts groups
22 Aug 2024 11:07:27 UTC
Test ADQM cluster
Overview
Services
Hosts
Mapping
Configuration
Import
ADQMDB
24.3.2.23_arenadata2
2 / 2 successful components
Actions
Delete
Primary configuration
Configuration groups
Action hosts groups
Components
Info
Search group name
Search host name
Create action hosts group
Name
Description
Hosts
Actions
hosts_1-3
2
Search hosts
dev-adqm-01.ru-central1.internal
dev-adqm-03.ru-central1.internal
hosts_2-4
2
Search hosts
dev-adqm-02.ru-central1.internal
dev-adqm-04.ru-central1.internal

Distributing hosts into groups for sequential ADQMDB service updates

2. Perform the ADQMDB service upgrade for one host group. To do this, in the host group row, click the icon in the ***Actions*** column and run the ***Upgrade*** action.



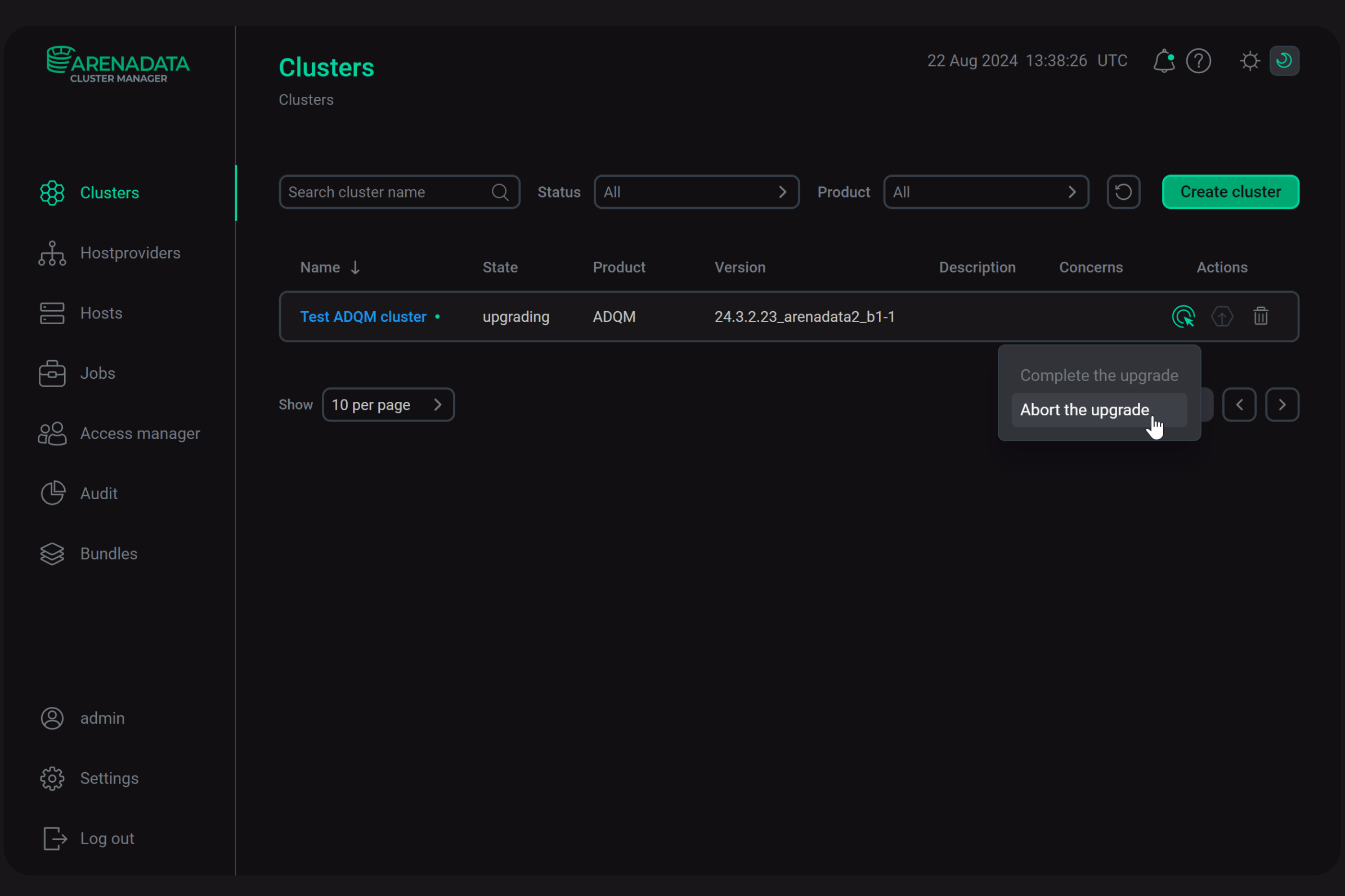
ARENADATA CLUSTER MANAGER
Clusters
Clusters / Test ADQM cluster / Services / ADQMDB / Action hosts groups
22 Aug 2024 12:58:14 UTC
Test ADQM cluster
Overview
Services
Hosts
Mapping
Configuration
Import
ADQMDB
24.3.2.23_arenadata2
2 / 2 successful components
Actions
Delete
Primary configuration
Configuration groups
Action hosts groups
Components
Info
Create action hosts group
Name
Description
Hosts
Actions
hosts_1-3
2
hosts_2-4
2
Upgrade
Show 10 per page

Initiating ADQMDB update for a group of hosts

In the window that opens, click ***Run*** to confirm the upgrade. After the ADQMDB service upgrade for a host group is complete, the cluster will remain in the `upgrading` state.

You can re-run the ***Upgrade*** action for a group of hosts if, for example, the upgrade failed on some hosts or new hosts were added to the group.

3. After the ***Upgrade*** action has been successfully completed, make sure that the new version of ClickHouse is installed on all hosts in the group (for example, you can run the `select version()` command in the clickhouse-client command-line client) and that databases work correctly (ClickHouse guarantees technical compatibility of two versions within a year, but business behavior may change, so it is recommended to check the applied result for each version separately). Further actions depend on the check result:
   - If the check is successful, you can continue upgrading ADQMDB on other hosts of the cluster (run the ***Upgrade*** action for the next group of hosts) or proceed to completing the cluster upgrade.
   - If you have detected any problems while testing the new version of ClickHouse, you can abort the cluster upgrade and roll back the previous version of the ADQMDB service on the previously upgraded hosts. To do this, switch to the ***Clusters*** page and run the ***Abort the upgrade*** action for the cluster.



ARENADATA CLUSTER MANAGER
Clusters
22 Aug 2024 13:38:26 UTC
Search cluster name
Status All
Product All
Create cluster
Complete the upgrade
Abort the upgrade
Show 10 per page

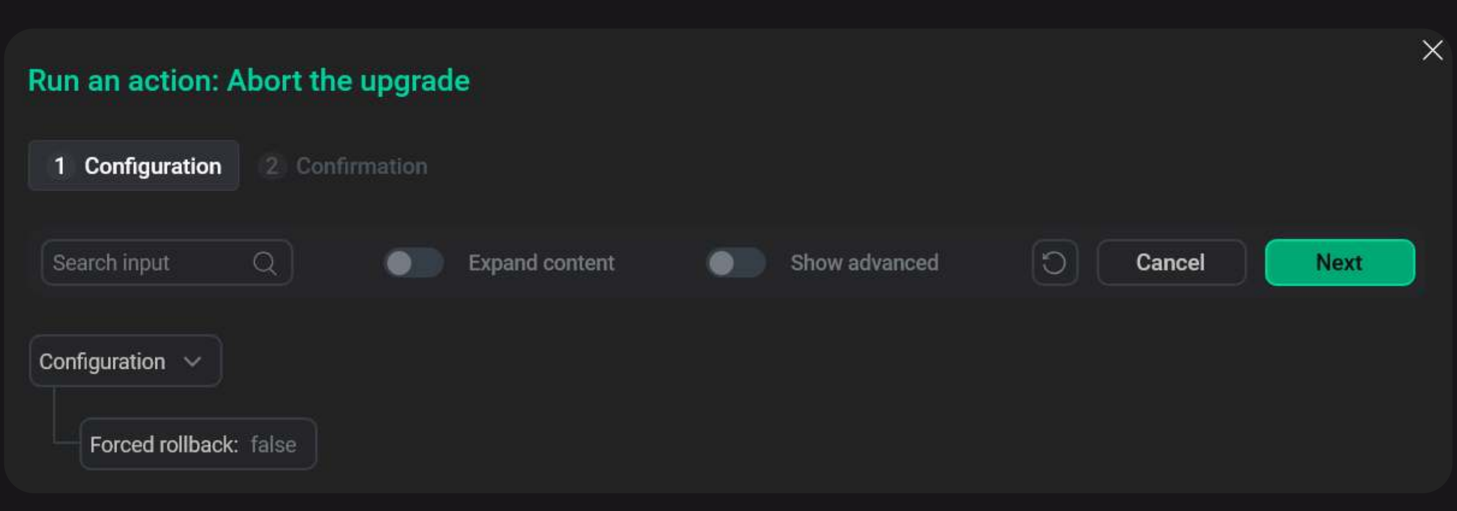
Name	State	Product	Version	Description	Concerns	Actions
Test ADQM cluster	upgrading	ADQM	24.3.2.23_arenadata2_b1-1			

Canceling the ADQMDB service update

After the update is canceled, the cluster bundle will revert to the previous version (and ClickHouse on all cluster hosts accordingly too), and the cluster state will be set to `installed`.

If you failed to cancel the upgrade (for example, because a file with information needed to roll back to the previous version is corrupted or lost, or the archive with the configuration files of the previous version of the ADQMDB service is missing), you can run the ***Abort the upgrade*** action with the ***Forced rollback*** option enabled. This will perform a forced rollback to the previous version of the ADQM bundle, after which you will need to run the ***Reconfig & restart*** action for the ADQMDB service.

Starting with ADQM 24.3.11.7.1.b1, the ***Abort the upgrade*** action with the ***Forced rollback*** flag set restores ADQM repositories and removes ADQM packages if they have been updated.

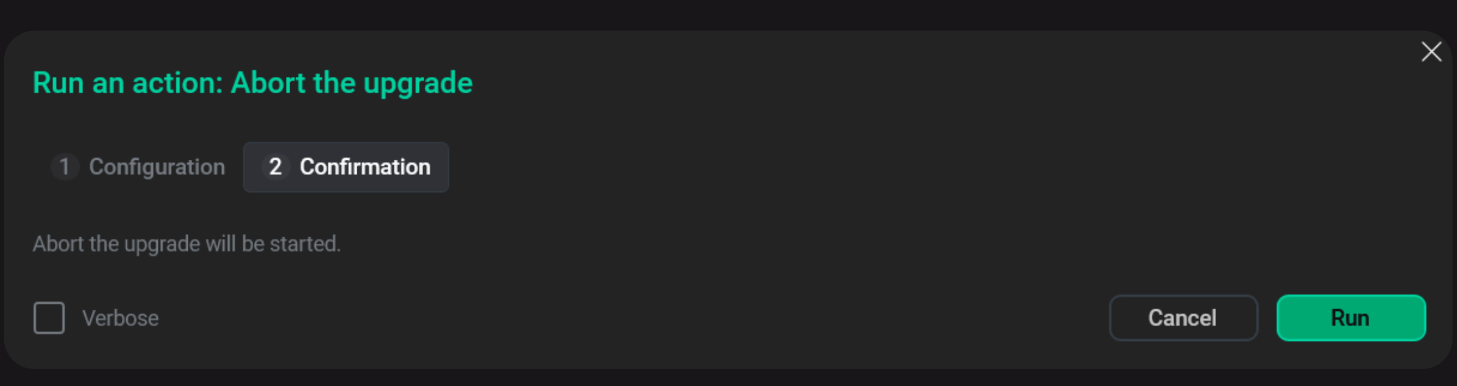


Run an action: Abort the upgrade
1 Configuration
2 Confirmation
Search input
Expand content
Show advanced
Cancel
Next
Configuration

Forced rollback: false

Canceling the update with the Forced rollback option

After you fill in the action parameters on the ***Configuration*** tab, click ***Next*** and confirm the action on the ***Confirmation*** tab by clicking ***Run***.



Run an action: Abort the upgrade
1 Configuration
2 Confirmation
Cancel
Run

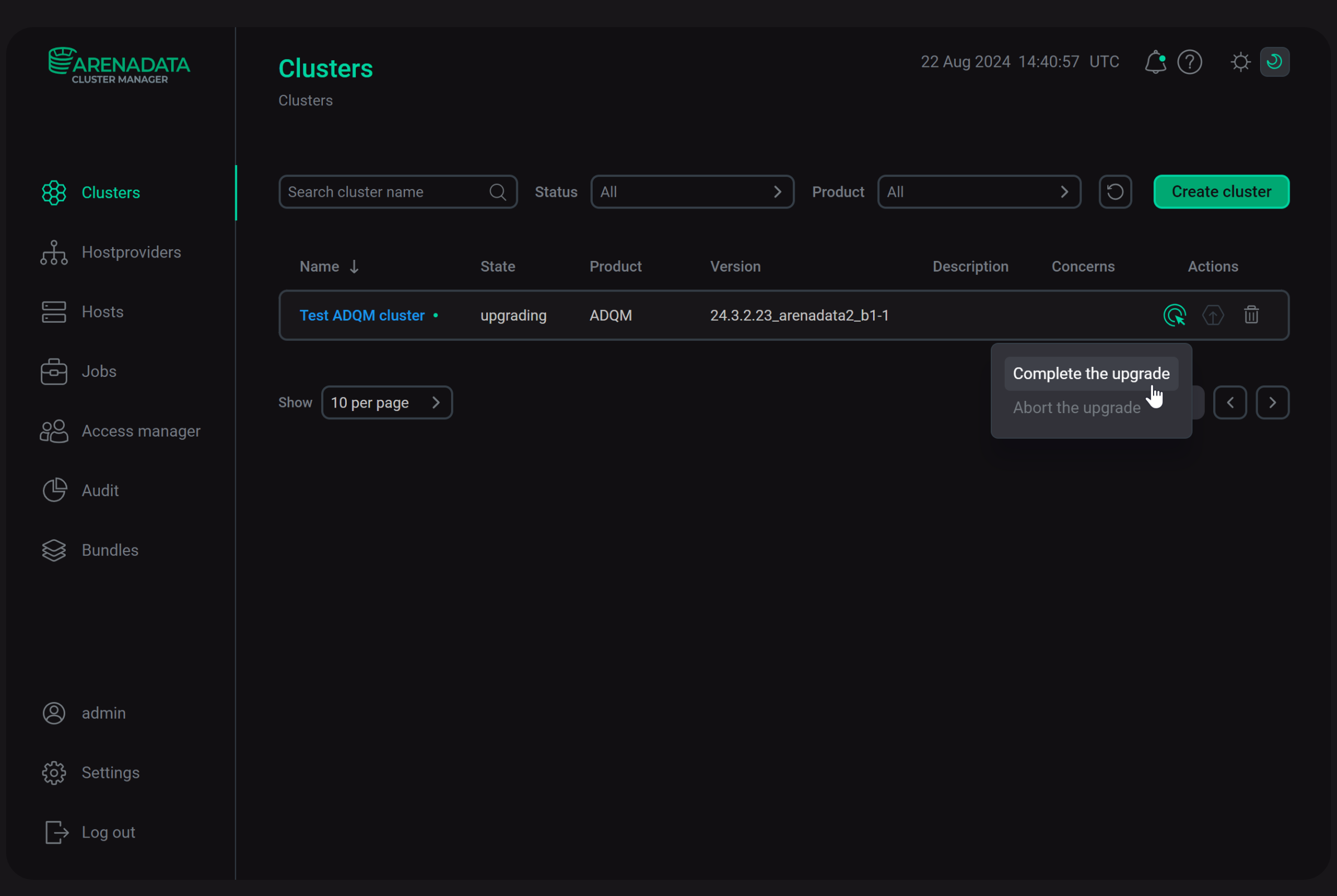
Abort the upgrade will be started.

☐ Verbose

Confirm canceling the update

Step 3. Complete the update

Once you have successfully upgraded the ADQMDB service to the new version on hosts where a preliminary upgrade was required, complete the cluster upgrade with the ***Complete the upgrade*** action (on the ***Clusters*** page) — it will upgrade ADQMDB on the remaining hosts, as well as all other ADQM services in the cluster. Once this action is performed, there is no option to revert to the previous version of ADQM.



ARENADATA CLUSTER MANAGER
Clusters
22 Aug 2024 14:40:57 UTC
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Search cluster name
Status All
Product All
Create cluster
Complete the upgrade
Abort the upgrade
Show 10 per page

Name	State	Product	Version	Description	Concerns	Actions
Test ADQM cluster	upgrading	ADQM	24.3.2.23_arenadata2_b1-1			

Completing the ADQM cluster upgrade

When the upgrade process is complete, ADCM will change the cluster state to `installed`.

NOTE

The ***Complete the upgrade*** action runs the usual upgrade of the cluster, i.e. the upgrade of all ADQM services on all hosts (as the Upgrade action does for a cluster being upgraded to an ADQM version up to and including 24.3.2.23.1.b1).

Maintenance mode

Elena Dvoryadkina

Contents

To Table of Contents

Overview

Examples

- Switch a host to the maintenance mode
- Remove a host from the cluster
- Replace an ADQMDB host

Overview

For hosts, ADQM supports maintenance mode implemented in the ADCM user interface. Temporarily unavailable or incorrectly functioning hosts can be switched to the maintenance mode to exclude their participation in cluster or service actions and perform actions without possible errors that may be associated with these hosts. The state of hosts in the maintenance mode does not affect the state of the cluster or services.

You can use this functionality for hardware or software maintenance, changing configuration settings, troubleshooting, decommissioning, or removing cluster nodes.

Specifics and limitations of the maintenance mode functionality supported by ADQM:

- It is not allowed to add any components of ADQM services to a host that is in the maintenance mode. However, you can logically remove components from a host in the maintenance mode if it is unavailable.
- If at least one host is switched to the maintenance mode, the ***Install*** and ***Upgrade*** cluster/service actions are not available.
- For the Zookeeper and Clickhousekeeper services, execution of the ***Reconfig and restart***/***Reconfig*** action may fail if the leader of the ZooKeeper/ClickHouse Keeper cluster is on a host switched to the maintenance mode.
- Maintenance mode is also supported to execute actions (except ***Install*** and Upgrade) of the ADQMDB service with integrated ClickHouse Keeper.

NOTE

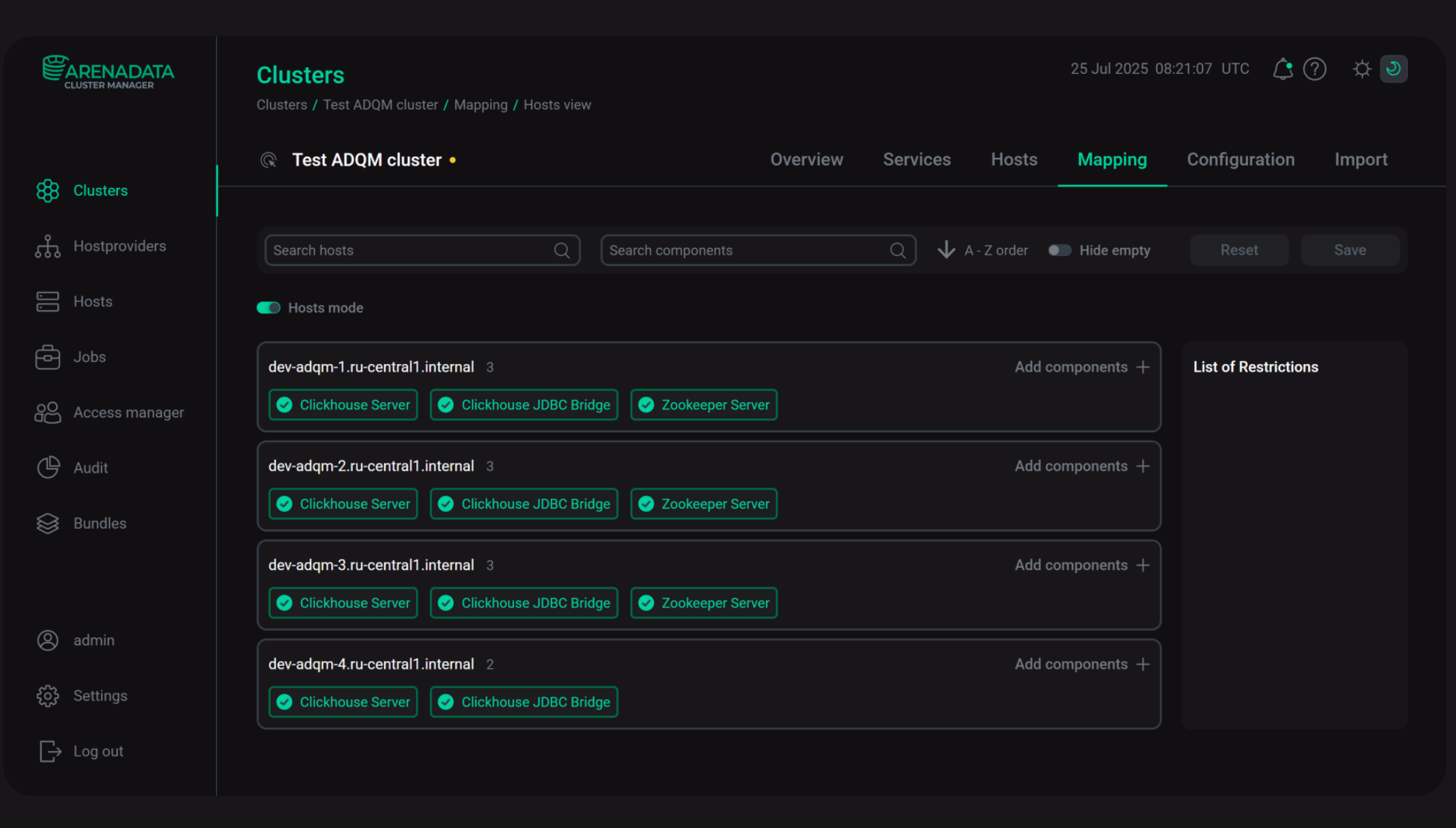
Maintenance mode in ADQM is currently supported for hosts only.

Examples

Examples below use an ADQM cluster deployed on four hosts, which has the following topology:

- shard 1 with two replicas — *dev-adqm-1.ru-central1.internal* and *dev-adqm-2.ru-central1.internal*;
- shard 2 with two replicas — *dev-adqm-3.ru-central1.internal* and *dev-adqm-4.ru-central1.internal*.

The ADQMDB and Zookeeper services are installed in the cluster with their components distributed across hosts as shown in the image below.



ARENADATA CLUSTER MANAGER
Clusters
Clusters / Test ADQM cluster / Mapping / Hosts view
25 Jul 2025 08:21:07 UTC
Test ADQM cluster
Overview
Services
Hosts
Mapping
Configuration
Import
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Search hosts
Search components
A - Z order
Hide empty
Reset
Save
Hosts mode
Add components
List of Restrictions

dev-adqm-1.ru-central1.internal 3: Clickhouse Server, Clickhouse JDBC Bridge, Zookeeper Server

dev-adqm-2.ru-central1.internal 3: Clickhouse Server, Clickhouse JDBC Bridge, Zookeeper Server

dev-adqm-3.ru-central1.internal 3: Clickhouse Server, Clickhouse JDBC Bridge, Zookeeper Server

dev-adqm-4.ru-central1.internal 2: Clickhouse Server, Clickhouse JDBC Bridge

Distribution of the cluster components across hosts

For testing purposes, one replicated table is added to the cluster:

```sql
CREATE TABLE test_repl_table on cluster 'default_cluster' (id UInt64)
ENGINE = ReplicatedMergeTree('/clickhouse/tables/{shard}/test_repl_table', '{replica}')
ORDER BY id;
```

The table contains a data row in the shard 2:

```sql
INSERT INTO test_repl_table VALUES ('1');
```

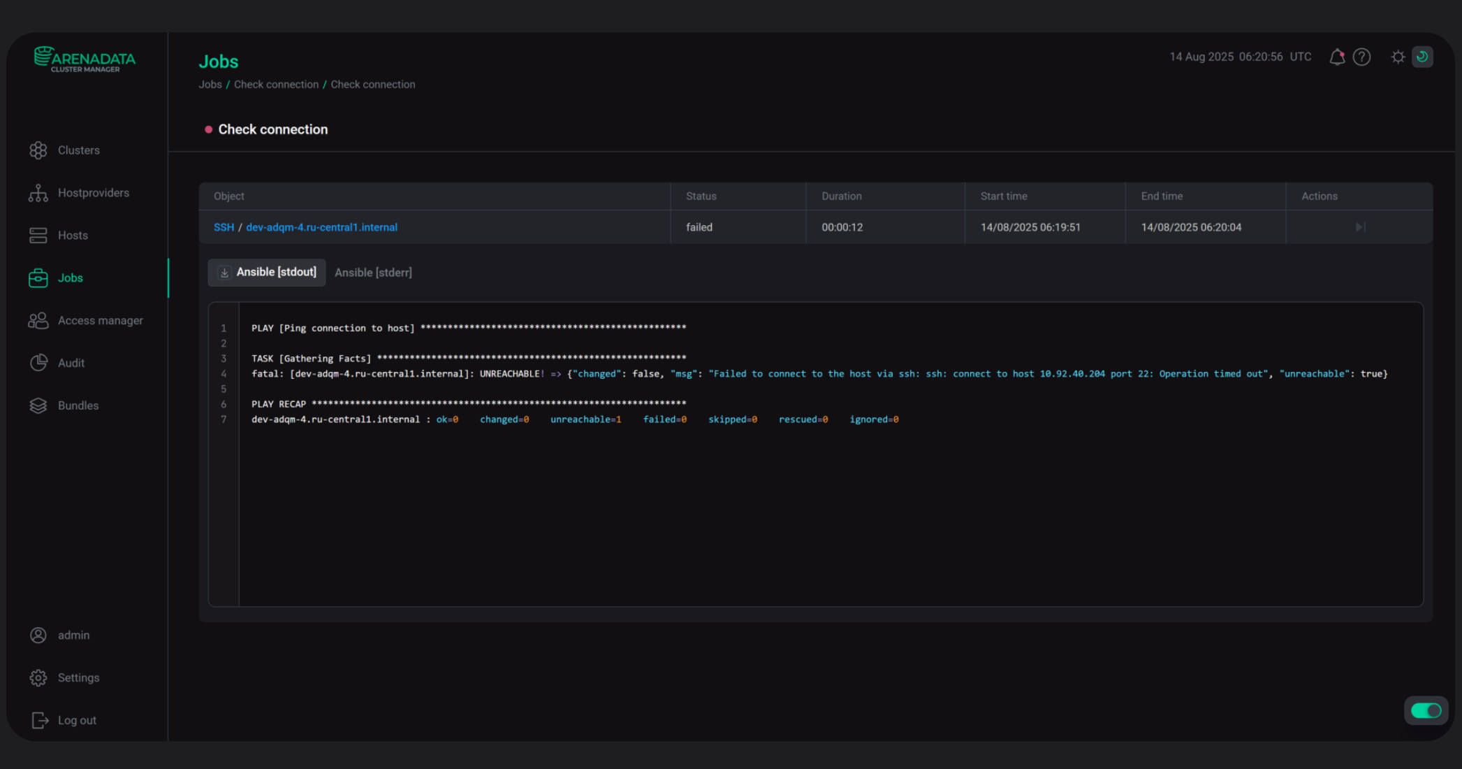
Switch a host to the maintenance mode

The following example shows how to enable the maintenance mode for an ADQMDB host to temporarily exclude it from processing.

1. Shut down a host on which the ADQMDB service's components are installed — for example, the *dev-adqm-4.ru-central1.internal* host.

Check connection to the host via SSH

The Ansible [stdout] of the ***Check connection*** action shows a message that the host is unreachable.



Jobs
Jobs / Check connection / Check connection
14 Aug 2025 06:20:56 UTC
Check connection
Ansible [stdout]
Ansible [stderr]

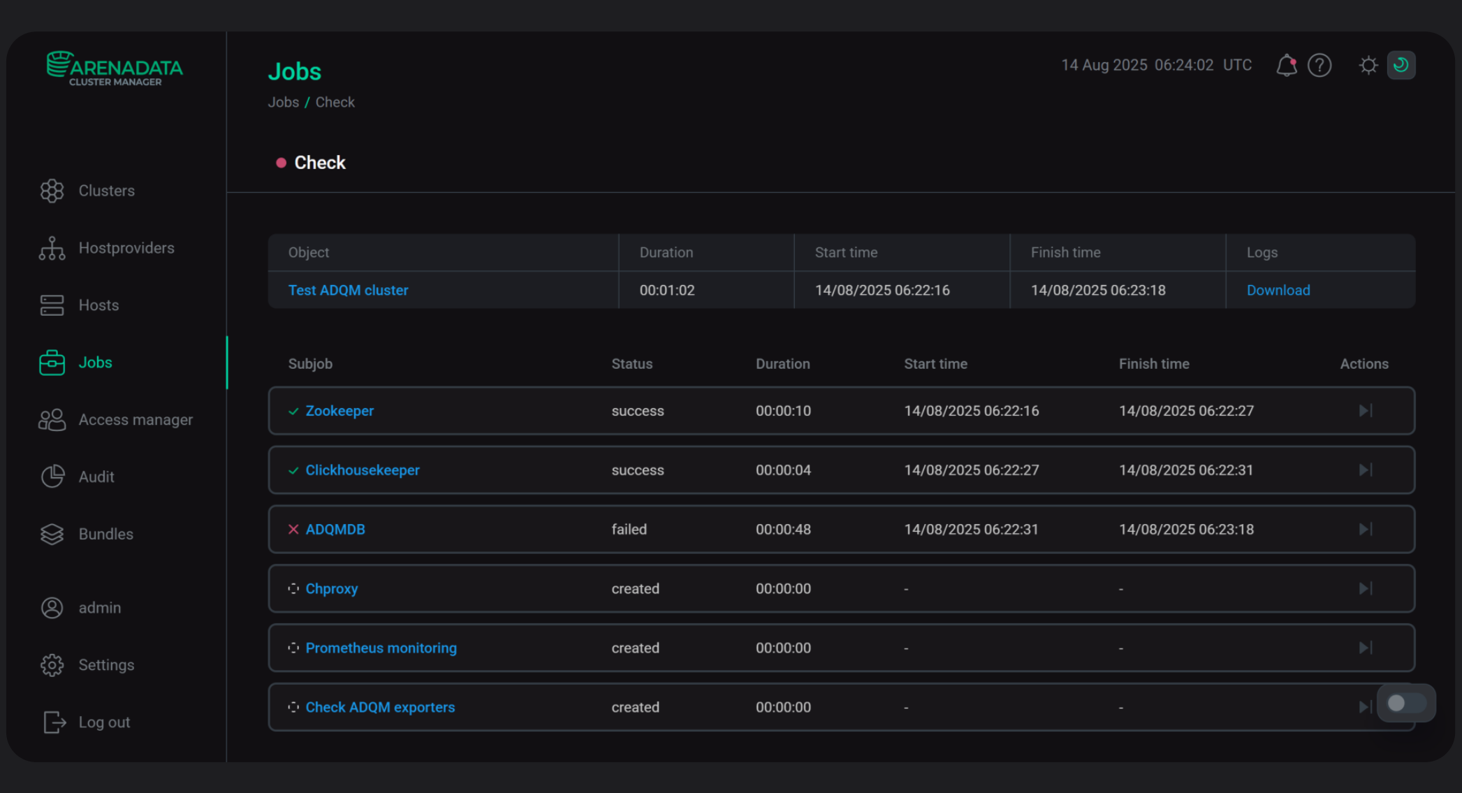
Object	Status	Duration	Start time	End time	Actions
SSH / dev-adqm-4.ru-central1.internal	failed	00:00:12	14/08/2025 06:19:51	14/08/2025 06:20:04	

```
PLAY [Ping connection to host] **************************************************

TASK [Gathering Facts] **********************************************************
fatal: [dev-adqm-4.ru-central1.internal]: UNREACHABLE! => {"changed": false, "msg": "Failed to connect to the host via ssh: ssh: connect to host 10.92.40.204 port 22: Operation timed out", "unreachable": true}

PLAY RECAP **********************************************************************
dev-adqm-4.ru-central1.internal : ok=0    changed=0    unreachable=1    failed=0    skipped=0    rescued=0    ignored=0
```

Result of the "Check connection" action

Check the cluster state

The ***Check*** cluster action fails because it finds one of the hosts to be unreachable while checking the ADQMDB service.

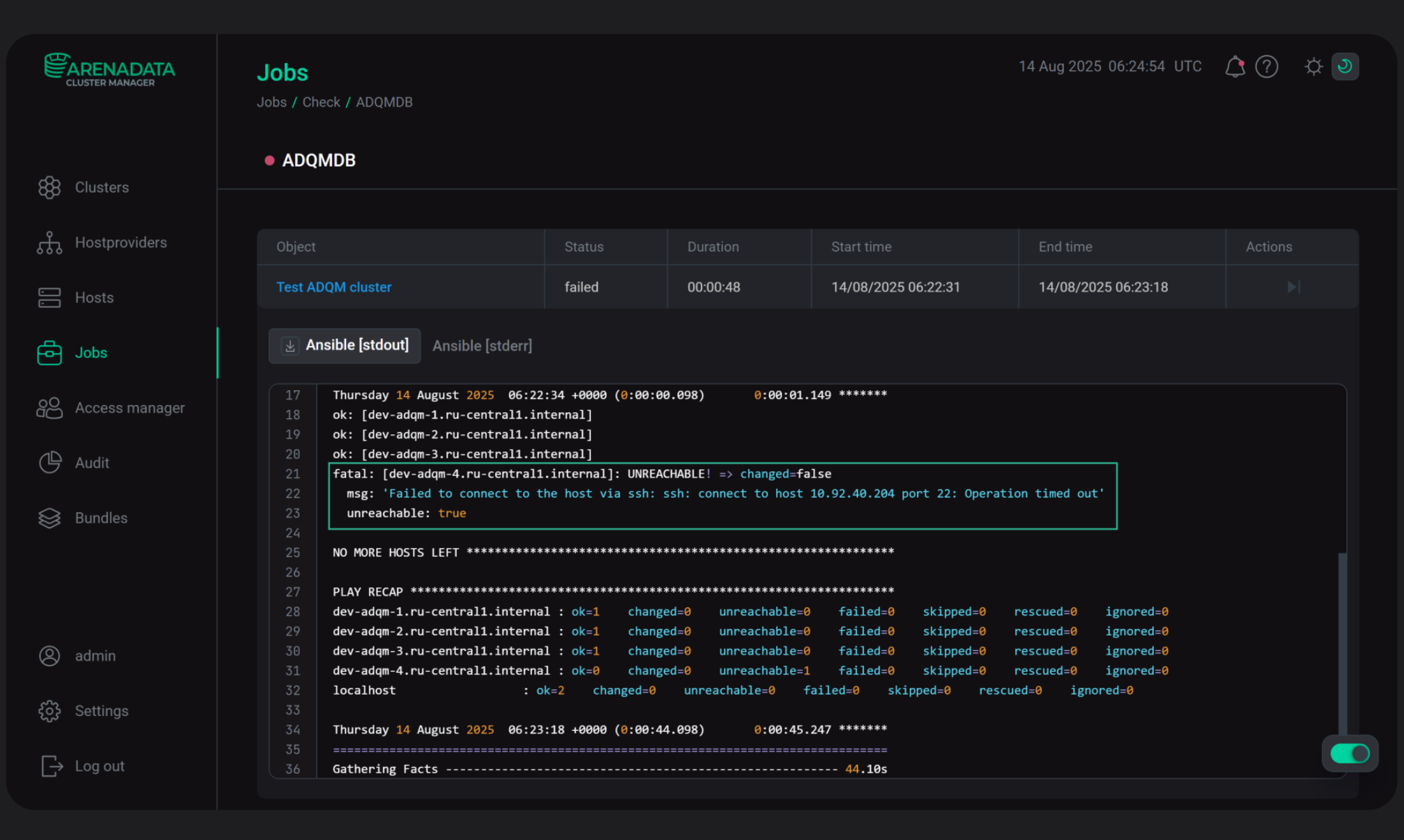


Jobs
Jobs / Check
14 Aug 2025 06:24:02 UTC
Check

Object	Duration	Start time	Finish time	Logs
Test ADQM cluster	00:01:02	14/08/2025 06:22:16	14/08/2025 06:23:18	Download

Subjob	Status	Duration	Start time	Finish time	Actions
✓ Zookeeper	success	00:00:10	14/08/2025 06:22:16	14/08/2025 06:22:27	
✓ Clickhousekeeper	success	00:00:04	14/08/2025 06:22:27	14/08/2025 06:22:31	
✕ ADQMDB	failed	00:00:48	14/08/2025 06:22:31	14/08/2025 06:23:18	
Chproxy	created	00:00:00	-	-	
Prometheus monitoring	created	00:00:00	-	-	
Check ADQM exporters	created	00:00:00	-	-	

Result of the "Check" cluster action



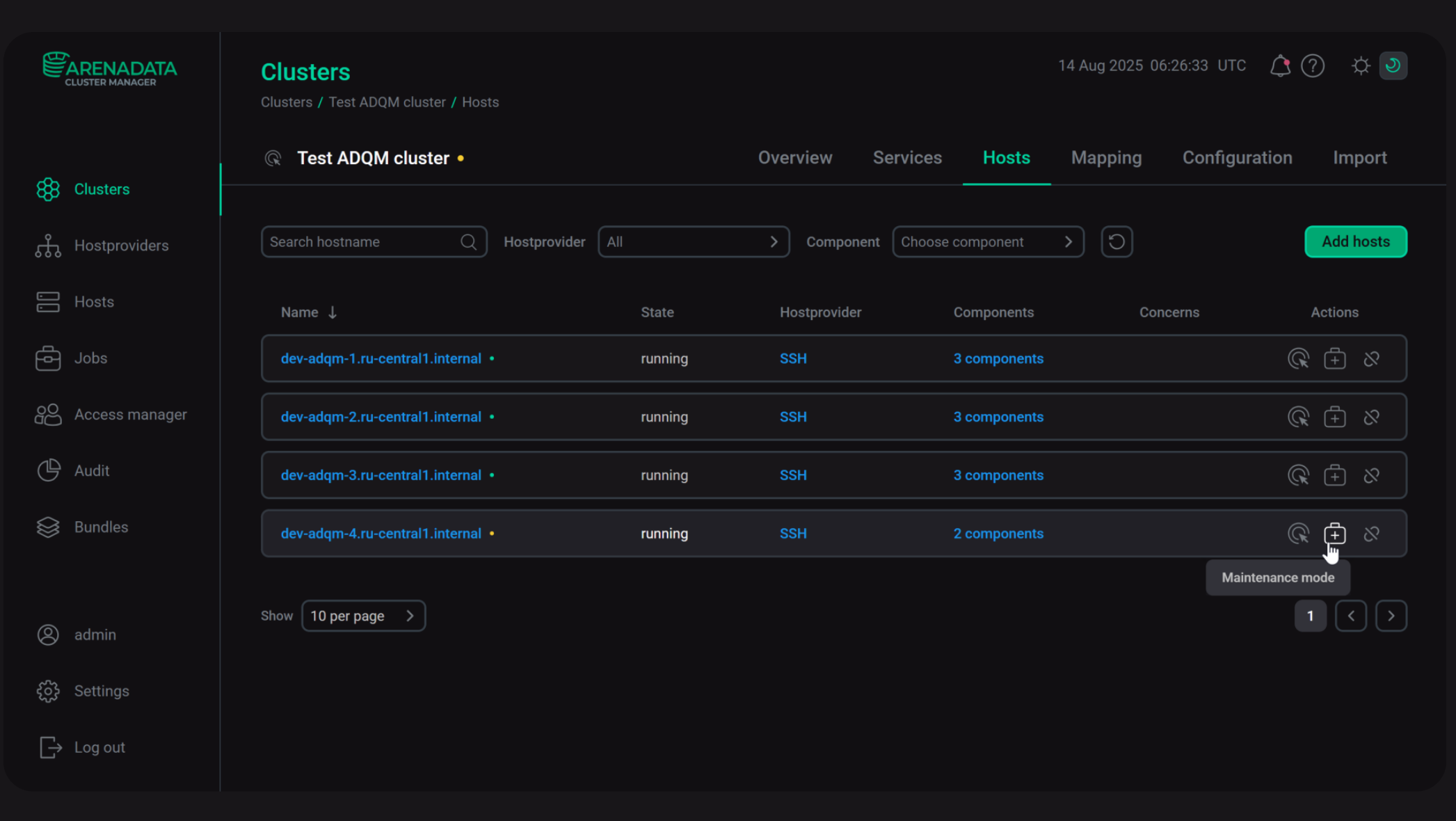
Jobs
Jobs / Check / ADQMDB
14 Aug 2025 06:24:54 UTC
ADQMDB
Ansible [stdout]
Ansible [stderr]

Object	Status	Duration	Start time	End time	Actions
Test ADQM cluster	failed	00:00:48	14/08/2025 06:22:31	14/08/2025 06:23:18	

```
Thursday 14 August 2025  06:22:34 +0000 (0:00:00.098)       0:00:01.149 *******
ok: [dev-adqm-1.ru-central1.internal]
ok: [dev-adqm-2.ru-central1.internal]
ok: [dev-adqm-3.ru-central1.internal]
fatal: [dev-adqm-4.ru-central1.internal]: UNREACHABLE! => changed=false
  msg: 'Failed to connect to the host via ssh: ssh: connect to host 10.92.40.204 port 22: Operation timed out'
  unreachable: true

NO MORE HOSTS LEFT *************************************************************

PLAY RECAP *********************************************************************
dev-adqm-1.ru-central1.internal : ok=1    changed=0    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
dev-adqm-2.ru-central1.internal : ok=1    changed=0    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
dev-adqm-3.ru-central1.internal : ok=1    changed=0    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
dev-adqm-4.ru-central1.internal : ok=0    changed=0    unreachable=1    failed=0    skipped=0    rescued=0    ignored=0
localhost                  : ok=2    changed=0    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

Thursday 14 August 2025  06:23:18 +0000 (0:00:44.098)       0:00:45.247 *******
===============================================================================
Gathering Facts ------------------------------------------------------- 44.10s
```

Message about the unavailable ADQMDB host

2. Enable the maintenance mode for the *dev-adqm-4.ru-central1.internal* host. To do this, click the icon ⊞ on the ***Hosts*** tab of the cluster page.



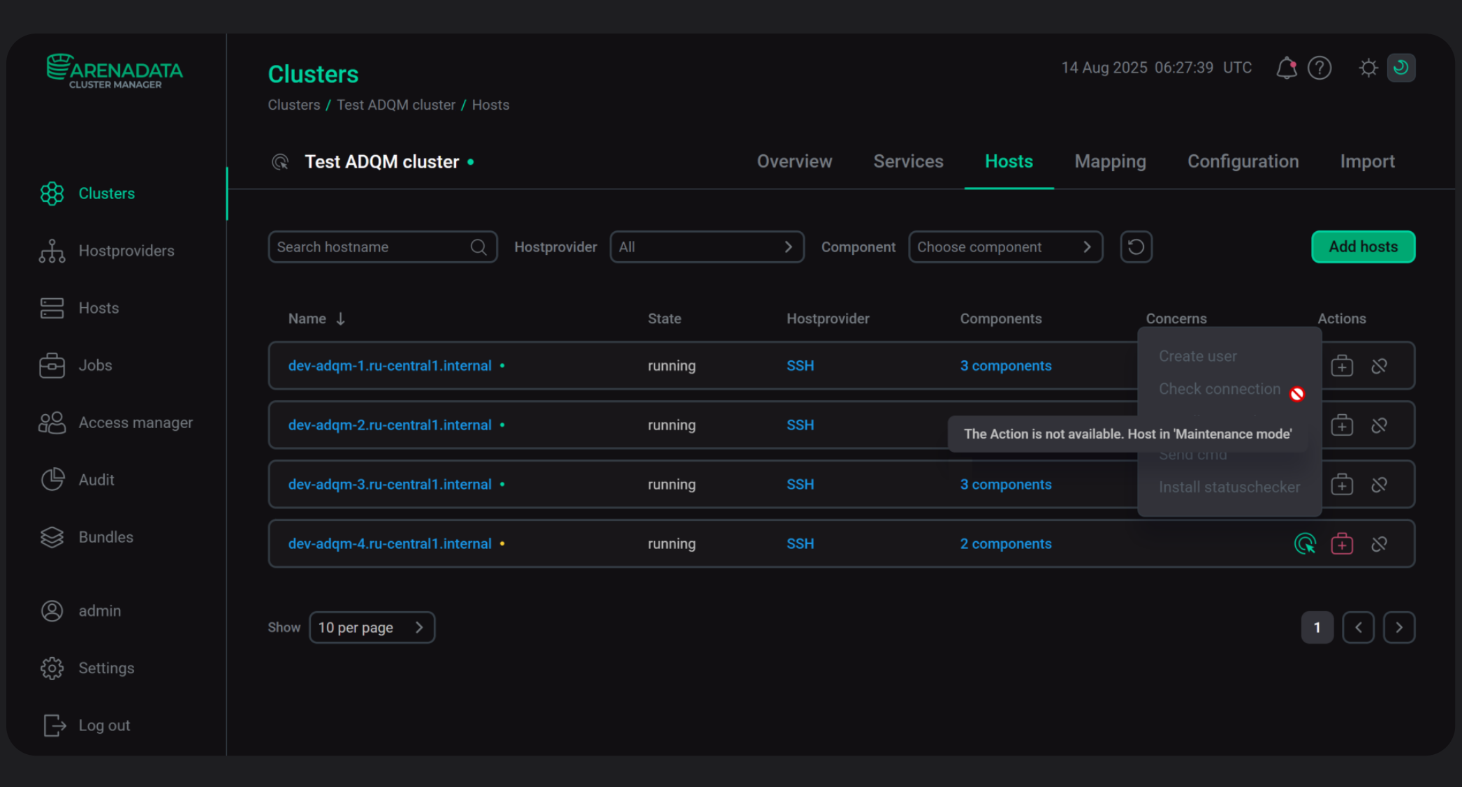
Clusters
Clusters / Test ADQM cluster / Hosts
14 Aug 2025 06:26:33 UTC
Test ADQM cluster
Overview
Services
Hosts
Mapping
Configuration
Import
Search hostname
Hostprovider
All
Component
Choose component
Add hosts
Maintenance mode
Show
10 per page

Name	State	Hostprovider	Components	Concerns	Actions
dev-adqm-1.ru-central1.internal	running	SSH	3 components		
dev-adqm-2.ru-central1.internal	running	SSH	3 components		
dev-adqm-3.ru-central1.internal	running	SSH	3 components		
dev-adqm-4.ru-central1.internal	running	SSH	2 components		

Enable the maintenance mode for the host

Host in the maintenance mode

Actions in the ADCM interface become unavailable to the host in the maintenance mode.

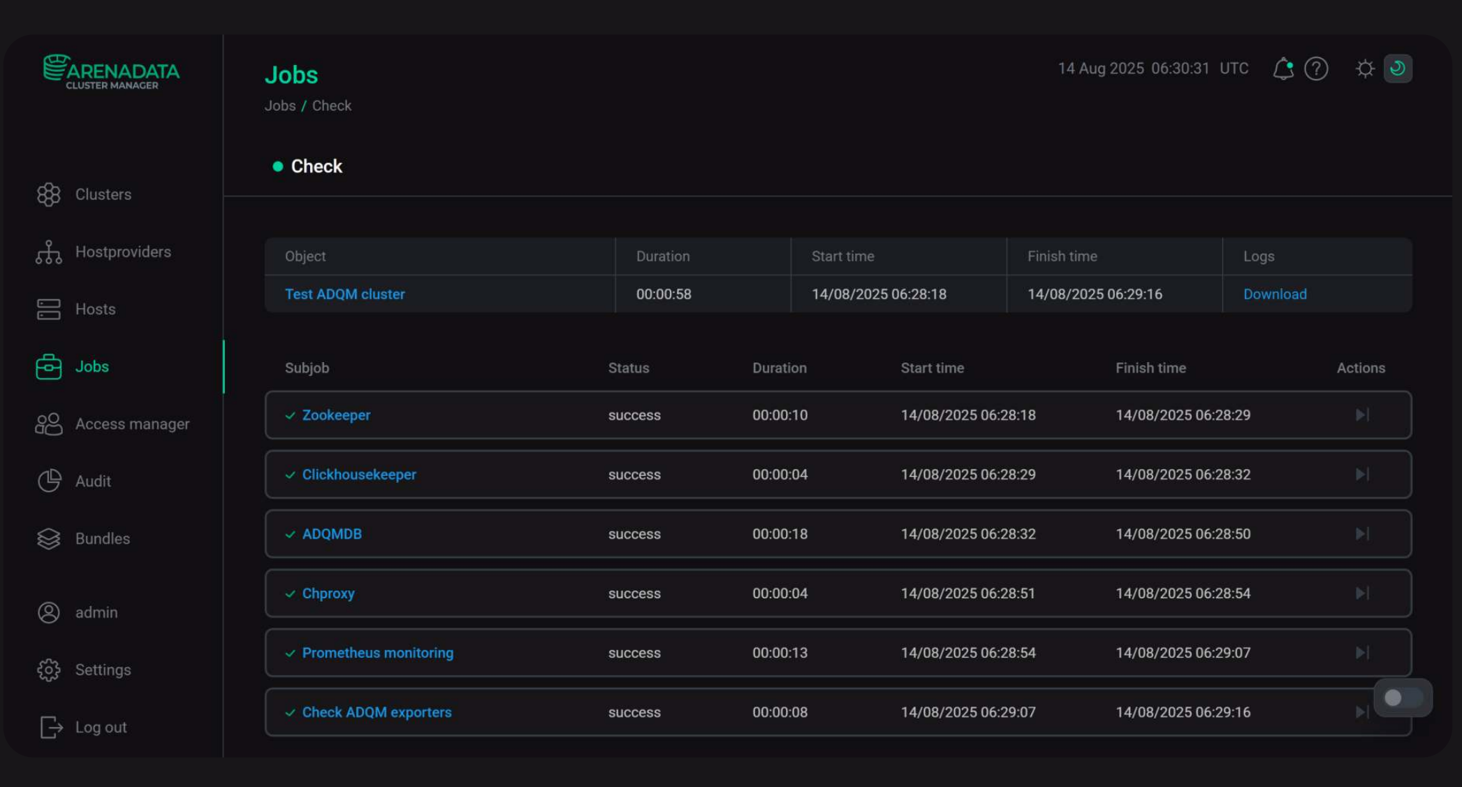


Clusters
Clusters / Test ADQM cluster / Hosts
14 Aug 2025 06:27:39 UTC
Test ADQM cluster
Create user
Check connection
Install statuschecker
Add hosts
Show
10 per page

The Action is not available. Host in 'Maintenance mode'

Host in the maintenance mode

Now, the complete unavailability of the host over the network does not lead to an error in executing cluster actions (for example, ***Check***) and ADQMDB service actions (***Check***, ***Start***, ***Stop***, ***Restart***, ***Reconfig***, ***Manage auto core dump***).



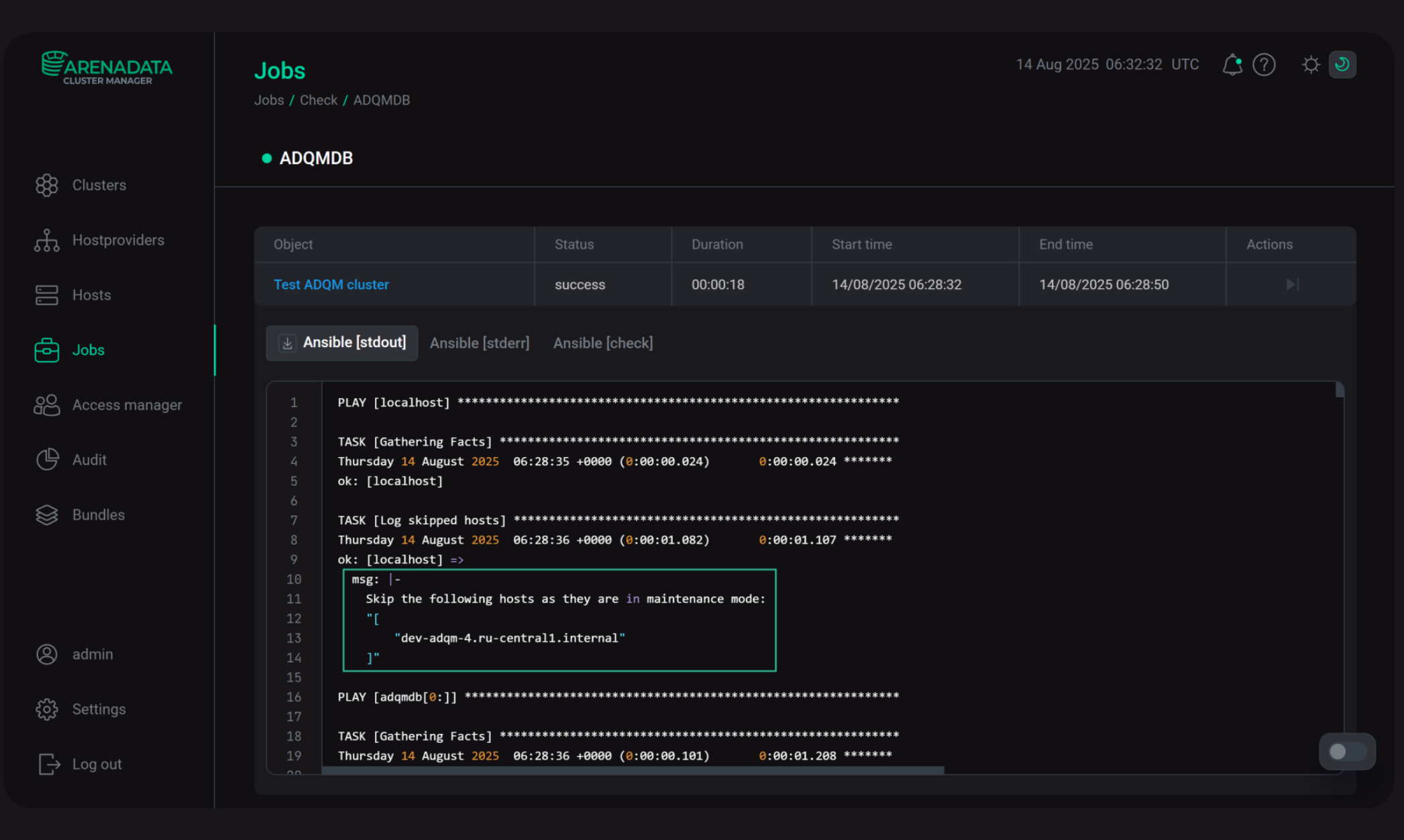
Jobs
Jobs / Check
14 Aug 2025 06:30:31 UTC
Check

Object	Duration	Start time	Finish time	Logs
Test ADQM cluster	00:00:58	14/08/2025 06:28:18	14/08/2025 06:29:16	Download

Subjob	Status	Duration	Start time	Finish time	Actions
✓ Zookeeper	success	00:00:10	14/08/2025 06:28:18	14/08/2025 06:28:29	
✓ Clickhousekeeper	success	00:00:04	14/08/2025 06:28:29	14/08/2025 06:28:32	
✓ ADQMDB	success	00:00:18	14/08/2025 06:28:32	14/08/2025 06:28:50	
✓ Chproxy	success	00:00:04	14/08/2025 06:28:51	14/08/2025 06:28:54	
✓ Prometheus monitoring	success	00:00:13	14/08/2025 06:28:54	14/08/2025 06:29:07	
✓ Check ADQM exporters	success	00:00:08	14/08/2025 06:29:07	14/08/2025 06:29:16	

Successful execution of the "Check" action for the cluster

Ansible [stdout] of the action step corresponding to the ADQMDB service check displays a message indicating that the host is not participating in processing because it is in the maintenance mode.



Jobs
Jobs / Check / ADQMDB
14 Aug 2025 06:32:32 UTC
ADQMDB
Ansible [stdout]
Ansible [stderr]
Ansible [check]

Object	Status	Duration	Start time	End time	Actions
Test ADQM cluster	success	00:00:18	14/08/2025 06:28:32	14/08/2025 06:28:50	

```
PLAY [localhost] ***************************************************************

TASK [Gathering Facts] *********************************************************
Thursday 14 August 2025  06:28:35 +0000 (0:00:00.024)       0:00:00.024 *******
ok: [localhost]

TASK [Log skipped hosts] *******************************************************
Thursday 14 August 2025  06:28:36 +0000 (0:00:01.082)       0:00:01.107 *******
ok: [localhost] =>
  msg: |-
    Skip the following hosts as they are in maintenance mode:
    "[
        "dev-adqm-4.ru-central1.internal"
    ]"

PLAY [adqmdb[0:]] **************************************************************

TASK [Gathering Facts] *********************************************************
Thursday 14 August 2025  06:28:36 +0000 (0:00:00.101)       0:00:01.208 *******
```

Message about the ADQMDB host in the maintenance mode

After performing the ***Reconfig*** action for the ADQMDB service, the host switched to the maintenance mode will be removed from the cluster topology definition in the `remote_servers` section of the */etc/clickhouse-server/config.xml* configuration file on all ADQMDB hosts except the host in the maintenance mode. At the same time, the cluster without this host remains operational — data is available on other hosts, and you can continue working with it.

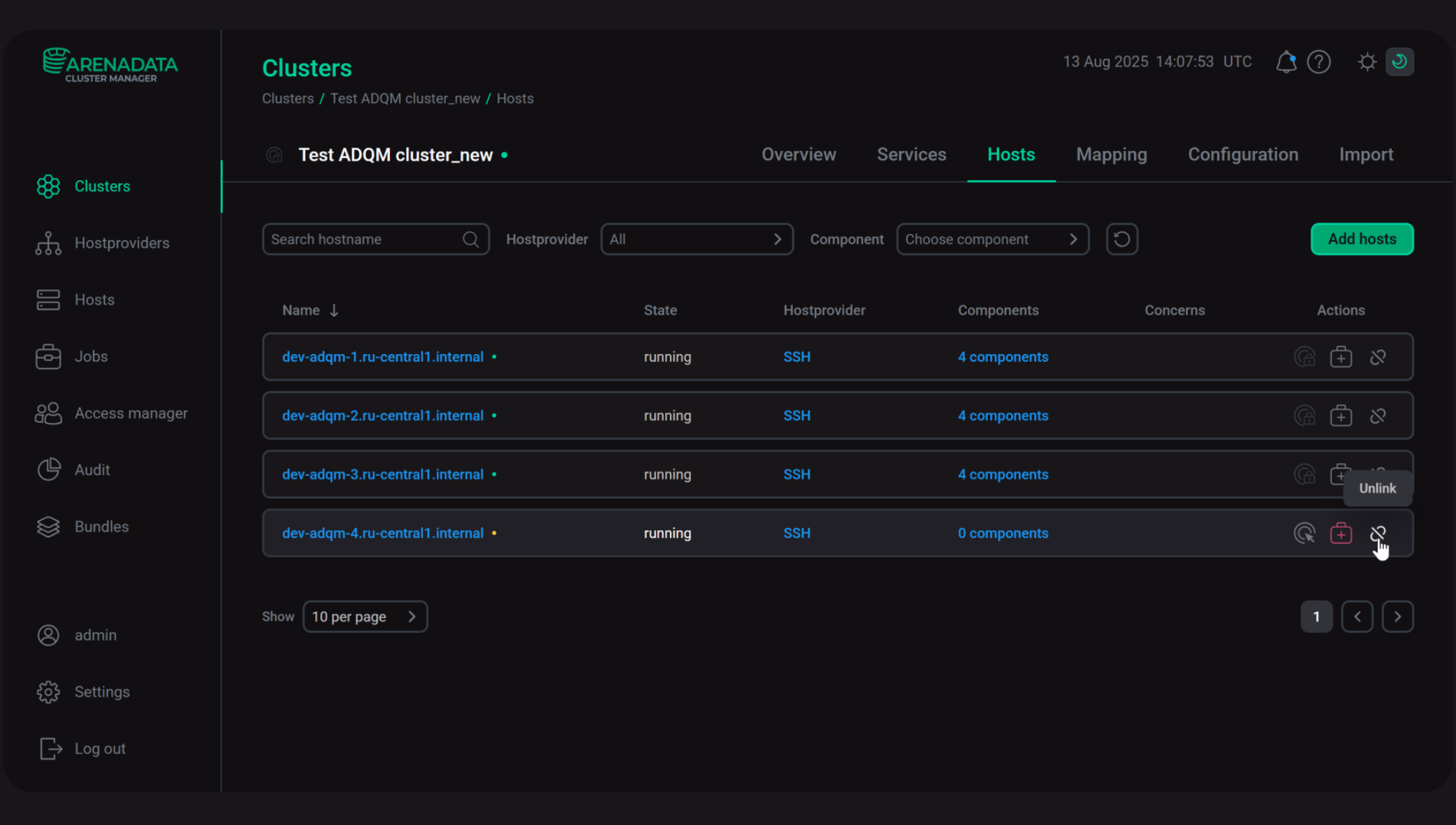
In order for the ADQMDB host to be processed by ADCM actions again, turn off the maintenance mode, and then run the ***Reconfig*** action on the ADQMDB service.

Remove a host from the cluster

If one of the ADQM hosts fails, it can be removed from the cluster (for example, this can be considered if replicas of data remain in the cluster and the presence of this host is not critical, or a new host will be added to replace the removed one).

In the following example, the *dev-adqm-4.ru-central1.internal* host with the ADQMDB service's components installed is deleted from the cluster.

1. Turn on the maintenance mode for the *dev-adqm-4.ru-central1.internal* host as described above.
2. Remove the replica corresponding to the *dev-adqm-4.ru-central1.internal* host from the logical cluster topology using the Cluster Configuration setting of the ADQMDB service. Click ***Save*** and perform the ***Reconfig*** action for the ADQMDB service to apply the new cluster topology.
3. Run the ***Add/Remove components*** action for the ADQMDB service and remove the Clickhouse Server and Clickhouse JDBC Bridge components from the *dev-adqm-4.ru-central1.internal* host.
4. Once all the components have been removed from the host, you can remove it from the cluster — on the ***Hosts*** page, click the icon ⛓ .



Clusters
Clusters / Test ADQM cluster_new / Hosts
13 Aug 2025 14:07:53 UTC
Test ADQM cluster_new
Overview
Services
Hosts
Mapping
Configuration
Import
Add hosts
Unlink
Show
10 per page

Name	State	Hostprovider	Components	Concerns	Actions
dev-adqm-1.ru-central1.internal	running	SSH	4 components		
dev-adqm-2.ru-central1.internal	running	SSH	4 components		
dev-adqm-3.ru-central1.internal	running	SSH	4 components		
dev-adqm-4.ru-central1.internal	running	SSH	0 components		

Remove a host from the cluster

Confirm the action by clicking ***Unlink*** in the window that appears.

CAUTION

When you need to delete a ZooKeeper or ClickHouse Keeper host, it is necessary to prepare a replacement host and then replace the failed host with the replacement host in a single step using the ***Add/Remove components*** action for the Zookeeper or Clickhousekeeper service (since the number of ZooKeeper/ClickHouse Keeper hosts should not be even).

Replace an ADQMDB host

The following example shows how to add a new host (with the same name — *dev-adqm-4.ru-central1.internal*) to the cluster instead of the removed ADQMDB host and restore data replication between the new host and the existing one (*dev-adqm-3.ru-central1.internal*).

1. Create a new *dev-adqm-4.ru-central1.internal* host in ADCM and add it to the ADQM cluster.
2. Add this host to the cluster topology via the ***Cluster Configuration*** parameter of the ADQMDB service, specifying it as a replica of the *dev-adqm-3.ru-central1.internal* host in the shard 2.
3. Run the ***Add/Remove components*** action for the ADQMDB service. Add the Clickhouse Server and Clickhouse JDBC Bridge components to the new host.
4. On the new host, create a table:

```sql
CREATE TABLE test_repl_table on cluster 'default_cluster' (id UInt64)
ENGINE = ReplicatedMergeTree('/clickhouse/tables/{shard}/test_repl_table', '{replica}')
ORDER BY id;
```

If the `REPLICA_ALREADY_EXISTS` error occurs, before creating the table on the new host, run the following command on the *dev-adqm-3.ru-central1.internal* host, which is a replica of *dev-adqm-4.ru-central1.internal*:

```sql
SYSTEM DROP REPLICA 'dev-adqm-4.ru-central1.internal';
```

Make sure that data from the *dev-adqm-3.ru-central1.internal* replica has been inserted into the table on the new host:

```sql
SELECT * FROM test_repl_table;
```

```
   ┌─id─┐
1. │  1 │
   └────┘
```

Coordination systems

Elena Dvoryadkina

To Table of Contents

For data replication and execution of distributed DDL queries (`ON CLUSTER`), ADQM requires you to configure and enable one of the coordination services for a cluster:

- Apache ZooKeeper — well-known open-source service for synchronization and coordination of distributed systems.
  ZooKeeper is implemented in Java and has a simple and powerful data model. The ZAB (ZooKeeper Atomic Broadcast) coordination algorithm used in ZooKeeper provides linearizable writes, but does not guarantee linearizability for reads, since each ZooKeeper node performs reads locally.

- ClickHouse Keeper — alternative coordination service that provides a ZooKeeper-compatible client-server protocol.
  ClickHouse Keeper is implemented in C++ and has the same data model as ZooKeeper.The used RAFT algorithm provides better consistency guarantees compared to ZooKeeper — optionally allows linearizable reads. ClickHouse Keeper can be used as a standalone replacement for ZooKeeper (when it is installed on separate servers) or as an internal part of a ClickHouse server.

General recommendations for choosing a coordination service depending on system requirements:

- For test or small clusters, you can use the integrated ClickHouse Keeper installed on the same servers as ClickHouse.

- For production environments, it is recommended to install ClickHouse Keeper or ZooKeeper on separate servers.

- If you already have an external ZooKeeper cluster configured, you can use it for an ADQM cluster.

IMPORTANT

- Since the format of ClickHouse Keeper snapshots and logs is incompatible with ZooKeeper, you should convert and copy data manually when migrating an ADQM cluster from ZooKeeper to ClickHouse Keeper.

- The interserver protocol in ClickHouse Keeper is also incompatible with ZooKeeper, so you cannot create a mixed cluster that uses both ZooKeeper and ClickHouse Keeper at the same time.

ZooKeeper

Elena Dvoryadkina

Contents

To Table of Contents

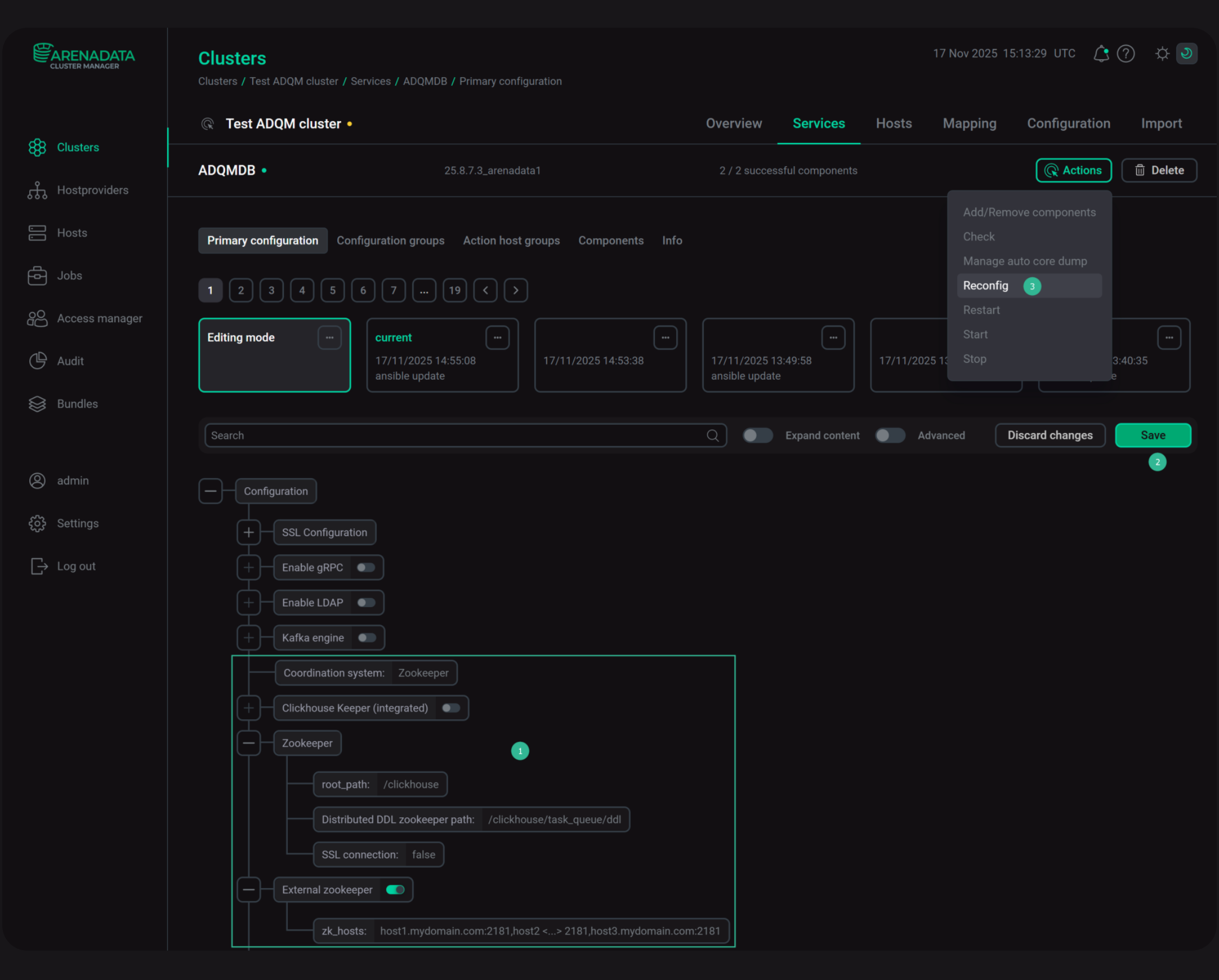
External ZooKeeper
ZooKeeper as a service of ADQM
Test replicated and distributed tables
  Configure ADQM cluster with ZooKeeper
  Replicated tables
  Distributed tables

To replicate data and execute distributed queries in ADQM through ZooKeeper, you can:

- use an external ZooKeeper cluster;
- install ZooKeeper as a service of the ADQM cluster.

External ZooKeeper

1. On the ADQMDB service's configuration page of the ADCM interface, perform the following:
   - activate the ***External zookeeper*** option and list all hosts of the ZooKeeper ensemble (indicating ports through which they communicate) separated by commas in the ***zk_hosts*** field;
   - specify paths to ZooKeeper nodes in the ***Zookeeper*** section;
   - set `Zookeeper` as the ***Coordination system*** parameter value.
2. Click ***Save*** and execute the ***Reconfig*** action for the ADQMDB service to use the external ZooKeeper as a coordination service for the ADQM cluster.



Coordination system: Zookeeper

root_path: /clickhouse

Distributed DDL zookeeper path: /clickhouse/task_queue/ddl

SSL connection: false

zk_hosts: host1.mydomain.com:2181,host2 <...> 2181,host3.mydomain.com:2181

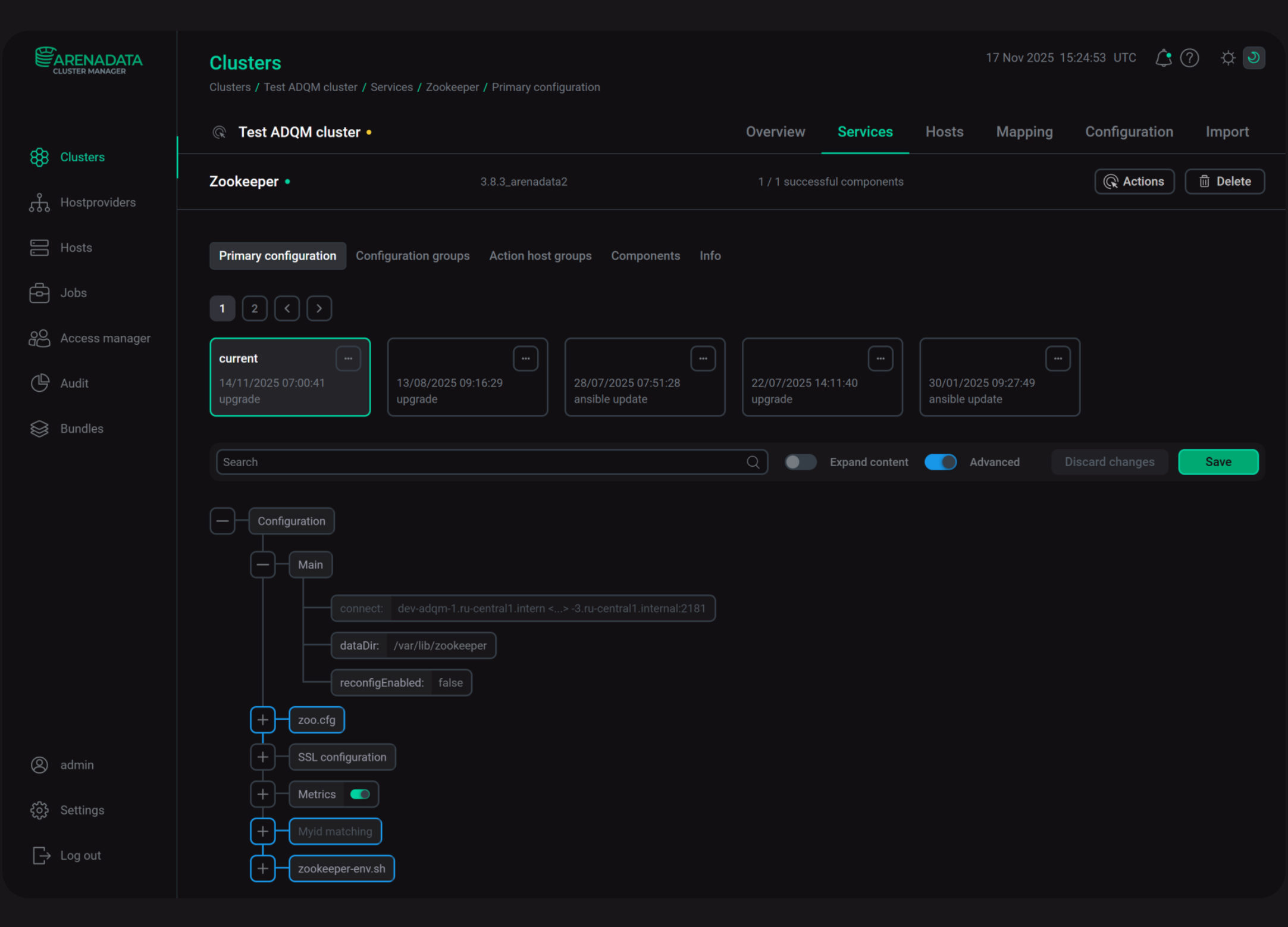
Clusters
Clusters / Test ADQM cluster / Services / ADQMDB / Primary configuration
Test ADQM cluster
Overview
Services
Hosts
Mapping
Configuration
Import
ADQMDB
Actions
Delete
Add/Remove components
Check
Manage auto core dump
Reconfig
Restart
Start
Stop
Primary configuration
Configuration groups
Action host groups
Components
Info
Editing mode
current
Discard changes
Save
Configuration
SSL Configuration
Enable gRPC
Enable LDAP
Kafka engine
Clickhouse Keeper (integrated)
Zookeeper
External zookeeper

Connect external ZooKeeper to ADQM

ZooKeeper as a service of ADQM

ADQM provides a special service that you can add and install on a cluster to coordinate replication and query distribution through ZooKeeper. To do this, perform the following steps in the ADCM interface:

1. Add the Zookeeper service to the ADQM cluster.
2. Install the Zookeeper Server component to the odd number of hosts.
3. Configure the Zookeeper service (see descriptions of parameters available for the service in the corresponding section of the Configuration parameters article).



connect: dev-adqm-1.ru-central1.intern <...> -3.ru-central1.internal:2181

dataDir: /var/lib/zookeeper

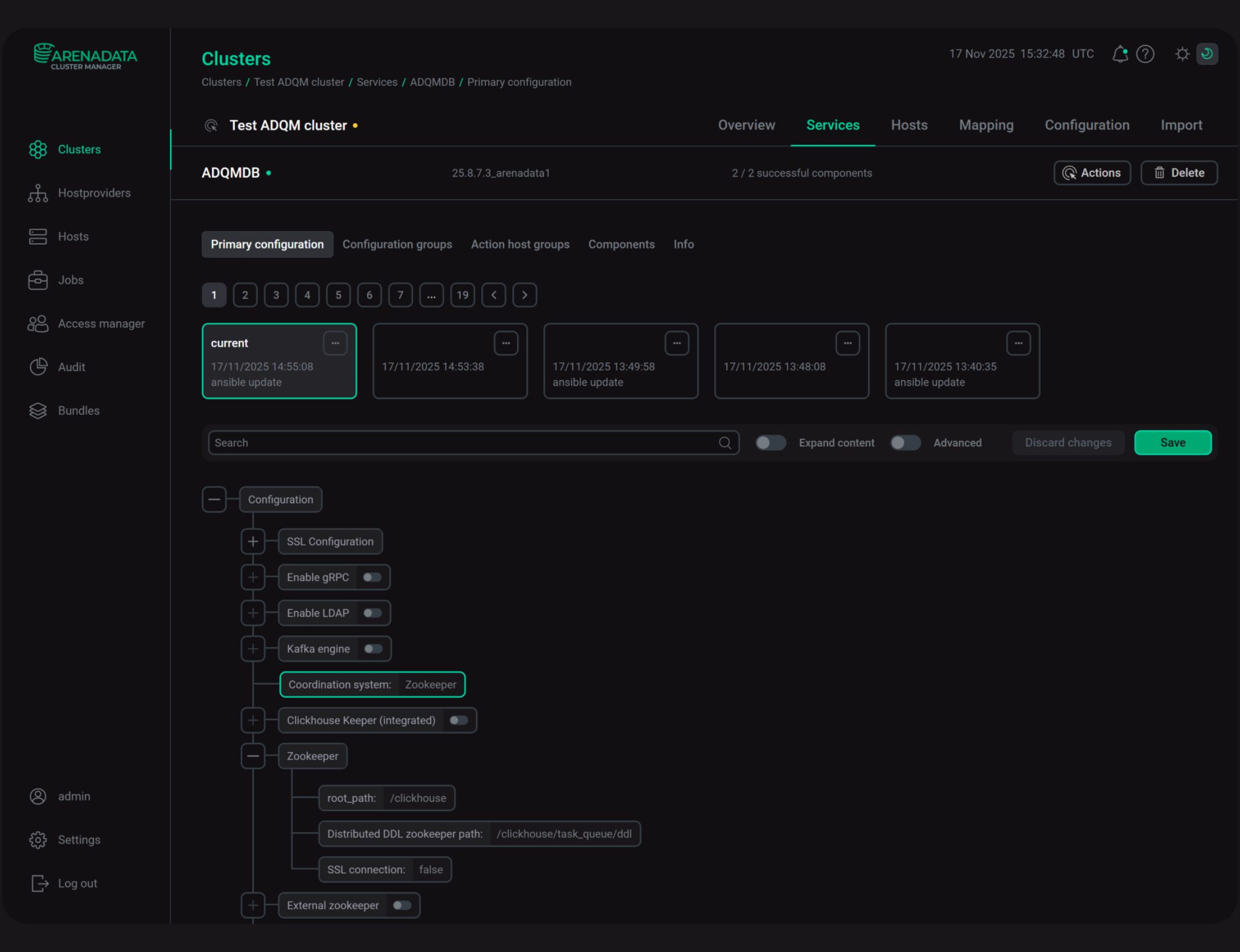
reconfigEnabled: false

Clusters
Clusters / Test ADQM cluster / Services / Zookeeper / Primary configuration
Test ADQM cluster
Zookeeper
Actions
Delete
Primary configuration
current
Save
Configuration
Main
zoo.cfg
SSL configuration
Metrics
Myid matching
zookeeper-env.sh

Set up the Zookeeper service

   Click ***Save*** to save settings.

4. Install the service.
5. On the ADQMDB service configuration page, set `Zookeeper` as the ***Coordination system*** parameter value.



Coordination system: Zookeeper

root_path: /clickhouse

Distributed DDL zookeeper path: /clickhouse/task_queue/ddl

SSL connection: false

Clusters
Clusters / Test ADQM cluster / Services / ADQMDB / Primary configuration
Test ADQM cluster
ADQMDB
Actions
Delete
Primary configuration
current
Save
Configuration
SSL Configuration
Enable gRPC
Enable LDAP
Kafka engine
Clickhouse Keeper (integrated)
Zookeeper
External zookeeper

Enable the Zookeeper service

   Configure also paths to ZooKeeper nodes in the ***Zookeeper*** section if needed.

6. Click ***Save*** and execute the ***Reconfig*** action for the ADQMDB service to use the installed ZooKeeper as a coordination service for the ADQM cluster.

> **NOTE**
>
> - If a cluster has the Zookeeper service installed and the ***External zookeeper*** option is enabled at the same time, the specified external ZooKeeper cluster will be used for data replication and query distribution.
> - To change hosts on which the Zookeeper Server component should be installed, you can use the ***Add/Remove components*** action of the Zookeeper service. After this action is completed, run the ***Reconfig*** action for the ADQMDB service to apply the changes.

Test replicated and distributed tables

Configure ADQM cluster with ZooKeeper

1. Configure a test ADQM cluster — two shards with two replicas in each one (hosts used in this example are *host-1*, *host-2*, *host-3*, and *host-4*). To learn how to do this via the ADCM interface, read Configure logical clusters in the ADCM interface. In this case, all necessary sections described below are added to the configuration file of each server automatically.
   - **Logical cluster configuration**
     In the *config.xml* file, the cluster configuration is defined in the `remote_servers` section:

```
<remote_servers>
    <default_cluster>
        <shard>
            <internal_replication>true</internal_replication>
            <weight>1</weight>
            <replica>
                <host>host-1</host>
                <port>9000</port>
            </replica>
            <replica>
                <host>host-2</host>
                <port>9000</port>
            </replica>
        </shard>
        <shard>
            <internal_replication>true</internal_replication>
            <weight>1</weight>
            <replica>
                <host>host-3</host>
                <port>9000</port>
            </replica>
            <replica>
                <host>host-4</host>
                <port>9000</port>
            </replica>
        </shard>
    </default_cluster>
</remote_servers>
```

     In this example, a ReplicatedMergeTree table is used, so the `internal_replication` parameter is set to `true` for each shard — in this case, the replicated table manages data replication itself (data is written to any available replica, and another replica receives data in the asynchronous mode). If using regular tables, set the `internal_replication` parameter to `false` — in this case, data is replicated by a Distributed table (data is written to all replicas of a shard).
   - **Macros**
     Each server configuration should also contain the `macros` section, which defines the shard and replica identifiers to be replaced with values corresponding to a specific host when creating replicated tables on the cluster (`ON CLUSTER`). For example, macros for *host-1* contained in the `default_cluster` cluster:

```
<macros>
    <replica>1</replica>
    <shard>1</shard>
</macros>
```

     If you configure a logical cluster in the ADCM interface via the ***Cluster Configuration*** setting and specify the cluster name, for example, `abc`, ADQM will write macros to *config.xml* as follows (for *host-1* in the `abc` cluster with the same topology as provided above):

```
<macros>
    <abc_replica>1</abc_replica>
    <abc_shard>1</abc_shard>
</macros>
```

     In this case, use the `{abc_shard}` and `{abc_replica}` variables within ReplicatedMergeTree parameters when creating replicated tables.

2. Install the Zookeeper service to three hosts (*host-1*, *host-2*, *host-3*). After that, the following sections will be added to the *config.xml* file:
   - `zookeeper` — list of ZooKeeper nodes;
   - `distributed_ddl` — path in ZooKeeper to the queue with DDL queries (if multiple clusters use the same ZooKeeper, this path should be unique for each cluster).

```
<zookeeper>
    <node>
        <host>host-1</host>
        <port>2181</port>
    </node>
    <node>
        <host>host-2</host>
        <port>2181</port>
    </node>
    <node>
        <host>host-3</host>
        <port>2181</port>
    </node>
    <root>/clickhouse</root>
</zookeeper>

<distributed_ddl>
    <path>/clickhouse/task_queue/ddl</path>
</distributed_ddl>
```

Replicated tables

1. Execute the following query on one of the hosts (for example, on *host-1*) to create a replicated table:

```
CREATE TABLE test_local ON CLUSTER default_cluster (id Int32, value_string String)
ENGINE = ReplicatedMergeTree('/clickhouse/tables/{shard}/test_local', '{replica}')
ORDER BY id;
```

   This query creates the `test_local` table on all hosts of the `default_cluster` cluster.

2. Insert data into the table on *host-2*:

```
INSERT INTO test_local VALUES (1, 'a');
```

3. Make sure data is replicated. To do this, select data from the `test_local` table on *host-1*:

```
SELECT * FROM test_local;
```

   If replication works correctly, the table on *host-1* will automatically receive data written to the replica on *host-2*:

```
┌─id─┬─value_string─┐
│  1 │ a            │
└────┴──────────────┘
```

Distributed tables

Note that the `SELECT` query returns data only from a table on a host on which the query is being run.

Insert data into any replica on the second shard (for example, into a table on *host-3*):

```
INSERT INTO test_local VALUES (2, 'b');
```

Repeat the `SELECT` query on *host-1* or *host-2* — the resulting selection will not include data from the second shard:

```
┌─id─┬─value_string─┐
│  1 │ a            │
└────┴──────────────┘
```

You can use distributed tables to get data from all shards (for details, see the Distributed tables section).

1. Create a table using the Distributed engine:

```
CREATE TABLE test_distr ON CLUSTER default_cluster AS default.test_local
ENGINE = Distributed(default_cluster, default, test_local, rand());
```

2. Run the following query on any host:

```
SELECT * FROM test_distr;
```

   The output includes data from both shards:

```
┌─id─┬─value_string─┐
│  1 │ a            │
└────┴──────────────┘
┌─id─┬─value_string─┐
│  2 │ b            │
└────┴──────────────┘
```

ClickHouse Keeper

Elena Dvoryadkina

Contents

To Table of Contents

ClickHouse Keeper as a service of ADQM

Integrated ClickHouse Keeper

Test replicated and distributed tables

- Configure ADQM cluster with ClickHouse Keeper
- Replicated tables
- Distributed tables

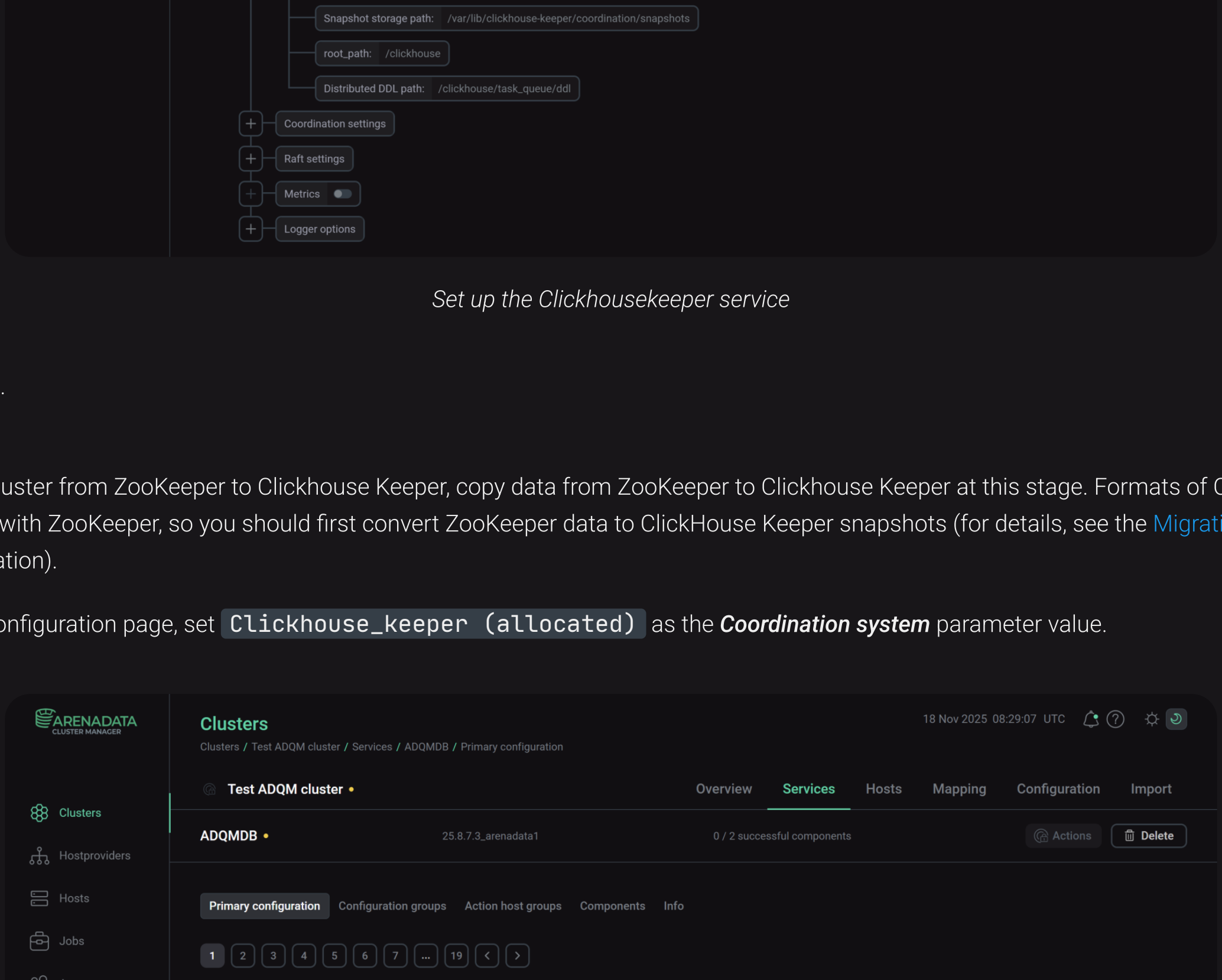
To use ClickHouse Keeper for data replication and query distribution in ADQM, you can:

- install ClickHouse Keeper in an ADQM cluster as a separate service;
- use ClickHouse Keeper integrated into a ClickHouse server.

ClickHouse Keeper as a service of ADQM

If you need to locate ClickHouse Keeper on dedicated hosts, install ClickHouse Keeper as a service in an ADQM cluster. For this, do the following in the ADCM interface:

1. Add the Clickhousekeeper service to the ADQM cluster.
2. Install the Clickhouse Keeper Server component to the odd number of hosts.
3. Configure the Clickhousekeeper service (for descriptions of available parameters, see the Configuration parameters article).

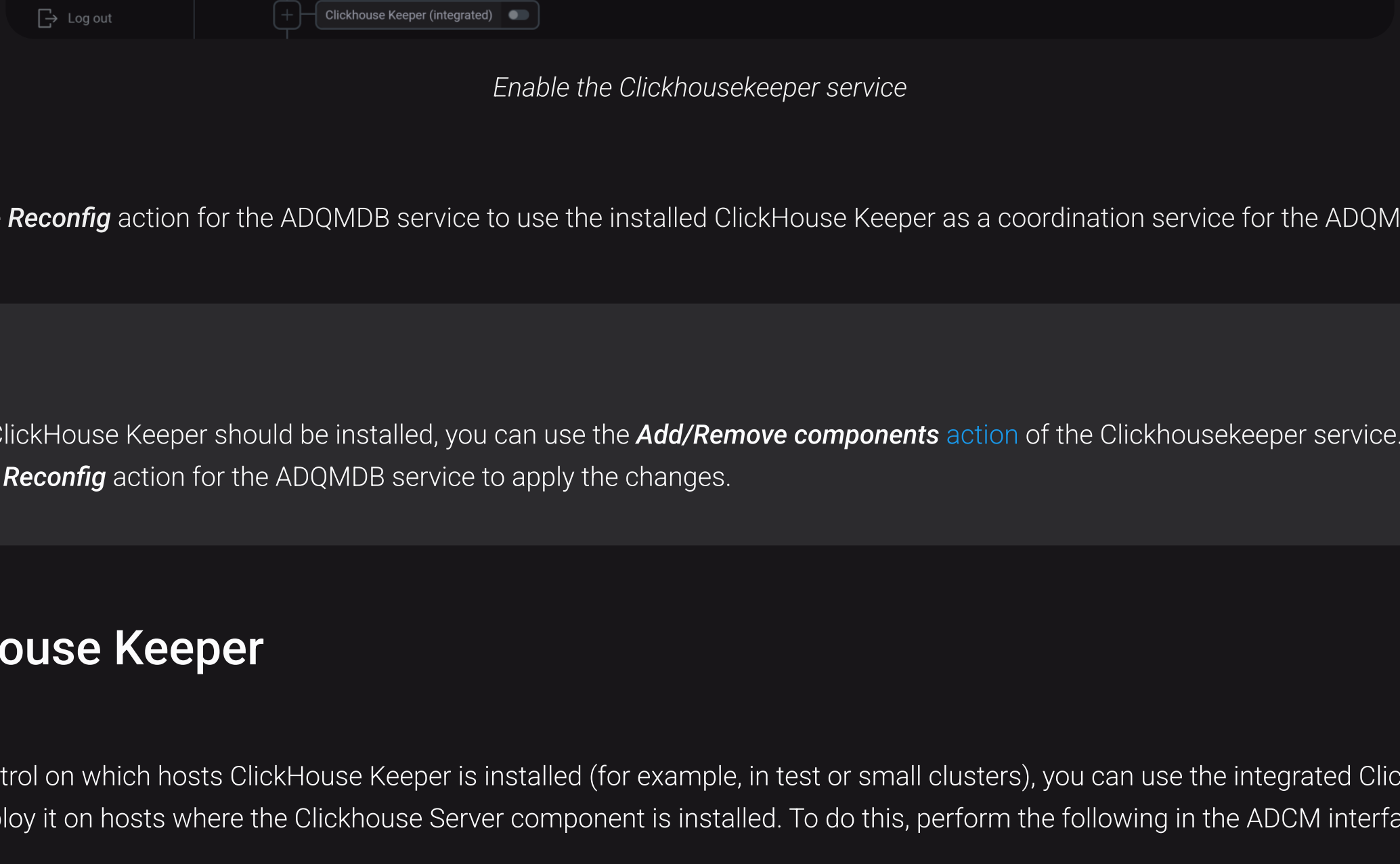

ARENADATA CLUSTER MANAGER
Clusters
Clusters / Test ADQM cluster / Services / Clickhousekeeper / Primary configuration
18 Nov 2025 08:24:21 UTC
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Test ADQM cluster
Overview
Services
Hosts
Mapping
Configuration
Import
Clickhousekeeper
25.8.7.3_arenadata1
0 / 1 successful components
Actions
Delete
Primary configuration
Configuration groups
Action host groups
Components
Info
current
18/11/2025 08:22:54
init
Search
Expand content
Advanced
Discard changes
Save
Configuration
General settings
Coordination settings
Raft settings
Metrics
Logger options

Listen host: ::
Enable IPv6: true
TCP port: 2129
Log storage path: /var/lib/clickhouse-keeper/coordination/log
Snapshot storage path: /var/lib/clickhouse-keeper/coordination/snapshots
root_path: /clickhouse
Distributed DDL path: /clickhouse/task_queue/ddl

Set up the Clickhousekeeper service

   Click ***Save*** to save settings.

4. Install the service.
   If you are migrating your cluster from ZooKeeper to Clickhouse Keeper, copy data from ZooKeeper to Clickhouse Keeper at this stage. Formats of ClickHouse Keeper snapshots and logs are incompatible with ZooKeeper, so you should first convert ZooKeeper data to ClickHouse Keeper snapshots (for details, see the Migration from ZooKeeper section in the ClickHouse documentation).
5. On the ADQMDB service configuration page, set `Clickhouse_keeper (allocated)` as the ***Coordination system*** parameter value.


ARENADATA CLUSTER MANAGER
Clusters
Clusters / Test ADQM cluster / Services / ADQMDB / Primary configuration
18 Nov 2025 08:29:07 UTC
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Test ADQM cluster
Overview
Services
Hosts
Mapping
Configuration
Import
ADQMDB
25.8.7.3_arenadata1
0 / 2 successful components
Actions
Delete
Primary configuration
Configuration groups
Action host groups
Components
Info
Editing mode
current
17/11/2025 14:55:08
ansible update
17/11/2025 14:53:38
17/11/2025 13:49:58
ansible update
17/11/2025 13:48:08
17/11/2025 13:40:35
ansible update
Search
Expand content
Advanced
Discard changes
Save
Configuration
SSL Configuration
Enable gRPC
Enable LDAP
Kafka engine
Clickhouse Keeper (integrated)

Coordination system: Clickhouse_keeper (allocated)

Enable the Clickhousekeeper service

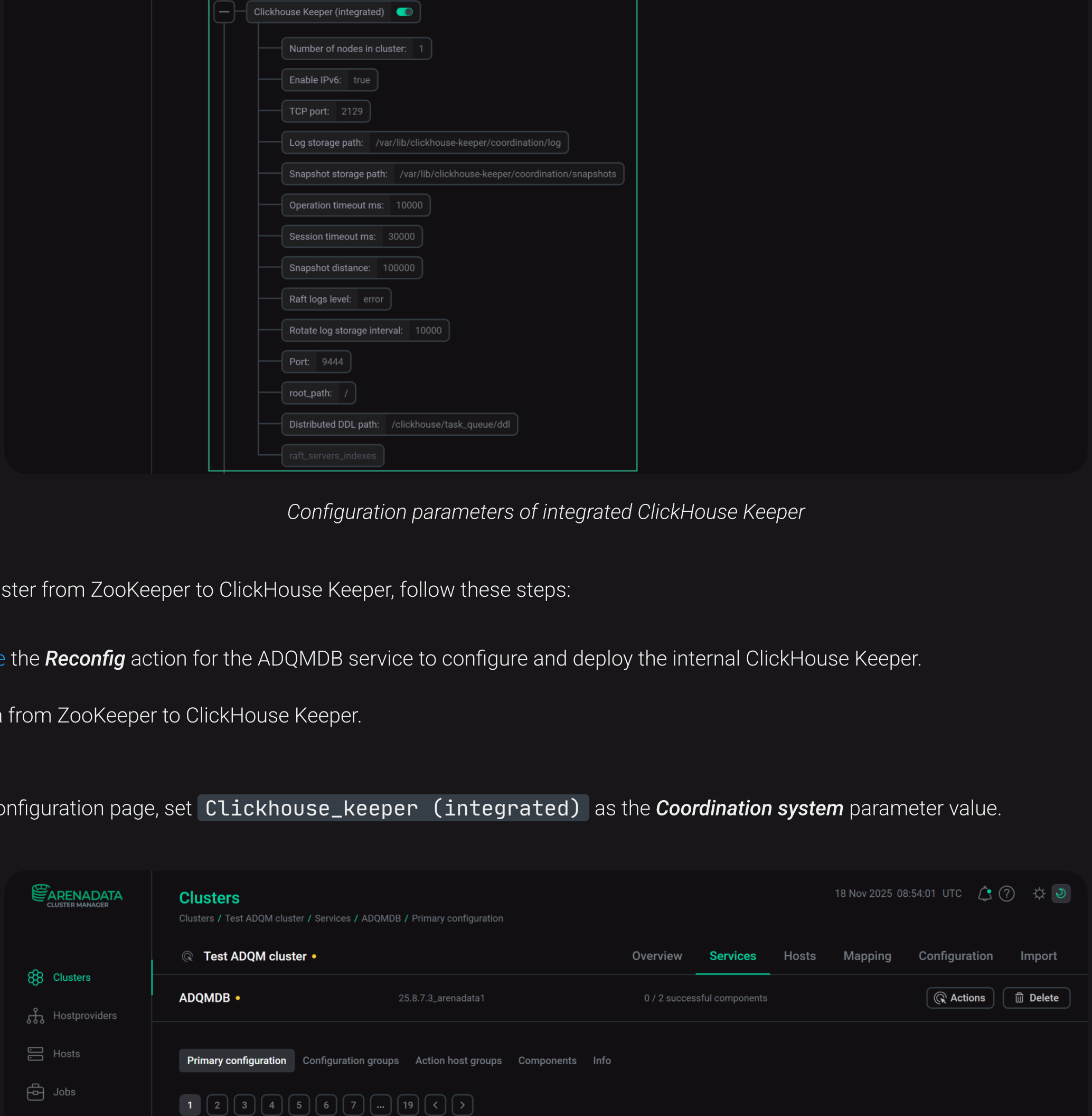
6. Click ***Save*** and execute the ***Reconfig*** action for the ADQMDB service to use the installed ClickHouse Keeper as a coordination service for the ADQM cluster.

> **NOTE**
>
> To change hosts on which ClickHouse Keeper should be installed, you can use the ***Add/Remove components*** action of the Clickhousekeeper service. After this action is completed, run the ***Reconfig*** action for the ADQMDB service to apply the changes.

Integrated ClickHouse Keeper

If there is no requirement to control on which hosts ClickHouse Keeper is installed (for example, in test or small clusters), you can use the integrated ClickHouse Keeper — ADQM will automatically configure and deploy it on hosts where the Clickhouse Server component is installed. To do this, perform the following in the ADCM interface:

1. Activate the ***Clickhouse Keeper (integrated)*** option on the configuration page of the ADQMDB service. In the expanded section, settings are filled in automatically based on the ClickHouse server parameters. You can change parameter values if necessary (see parameter descriptions in the Configuration parameters article).


ARENADATA CLUSTER MANAGER
Clusters
Clusters / Test ADQM cluster / Services / ADQMDB / Primary configuration
18 Nov 2025 08:47:45 UTC
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Test ADQM cluster
Overview
Services
Hosts
Mapping
Configuration
Import
ADQMDB
25.8.7.3_arenadata1
0 / 2 successful components
Actions
Delete
Primary configuration
Configuration groups
Action host groups
Components
Info
Editing mode
current
17/11/2025 14:55:08
ansible update
17/11/2025 14:53:38
17/11/2025 13:49:58
ansible update
17/11/2025 13:48:08
17/11/2025 13:40:35
ansible update
Search
Expand content
Advanced
Discard changes
Save
Configuration
SSL Configuration
Enable gRPC
Enable LDAP
Kafka engine
Clickhouse Keeper (integrated)
raft_servers_indexes

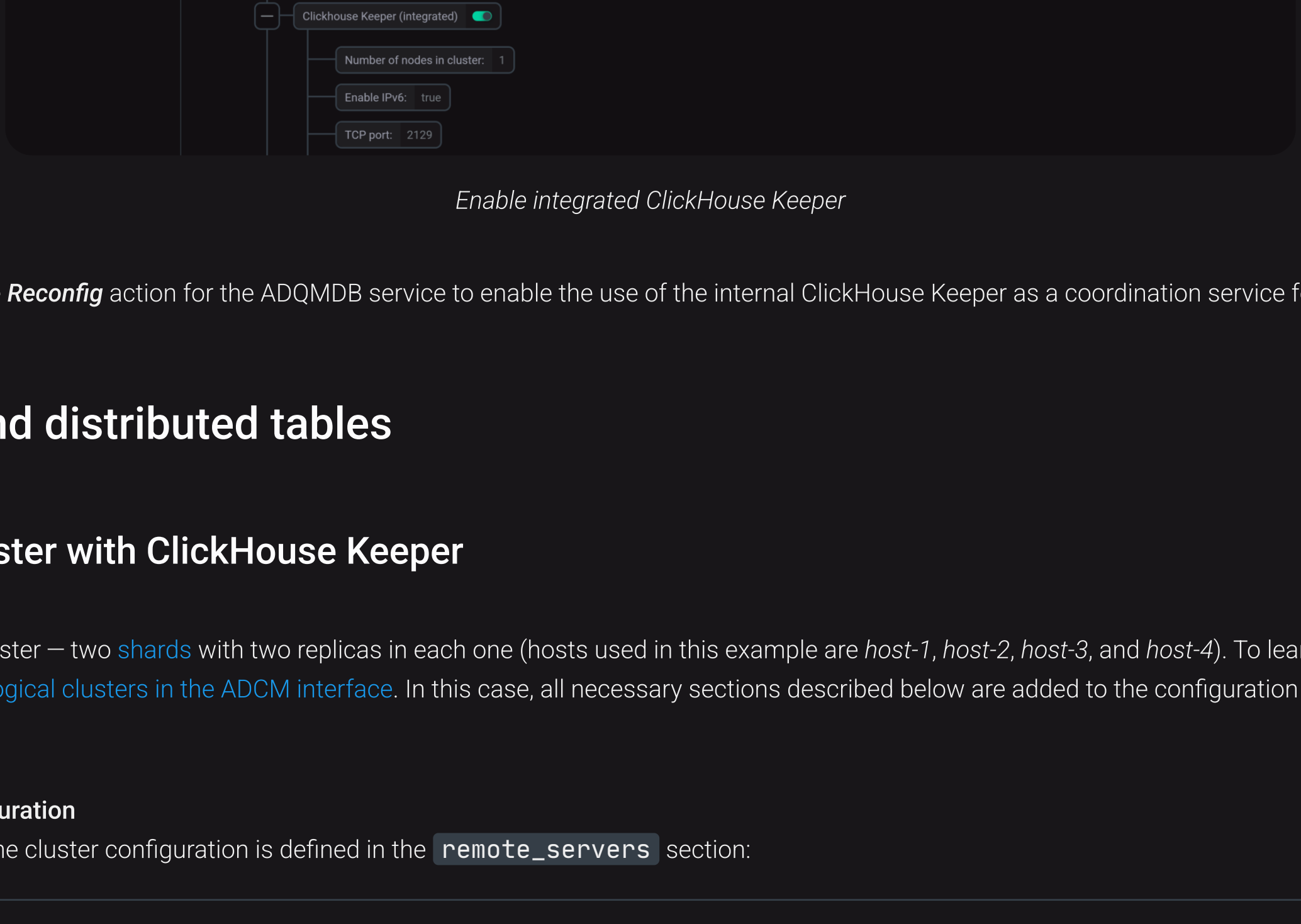
Coordination system: Zookeeper
Number of nodes in cluster: 1
Enable IPv6: true
TCP port: 2129
Log storage path: /var/lib/clickhouse-keeper/coordination/log
Snapshot storage path: /var/lib/clickhouse-keeper/coordination/snapshots
Operation timeout ms: 10000
Session timeout ms: 30000
Snapshot distance: 100000
Raft logs level: error
Rotate log storage interval: 10000
Port: 9444
root_path: /
Distributed DDL path: /clickhouse/task_queue/ddl

Configuration parameters of integrated ClickHouse Keeper

   If you are migrating the cluster from ZooKeeper to ClickHouse Keeper, follow these steps:

   - Click ***Save*** and execute the ***Reconfig*** action for the ADQMDB service to configure and deploy the internal ClickHouse Keeper.
   - Convert and copy data from ZooKeeper to ClickHouse Keeper.

2. On the ADQMDB service configuration page, set `Clickhouse_keeper (integrated)` as the ***Coordination system*** parameter value.


ARENADATA CLUSTER MANAGER
Clusters
Clusters / Test ADQM cluster / Services / ADQMDB / Primary configuration
18 Nov 2025 08:54:01 UTC
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
Test ADQM cluster
Overview
Services
Hosts
Mapping
Configuration
Import
ADQMDB
25.8.7.3_arenadata1
0 / 2 successful components
Actions
Delete
Primary configuration
Configuration groups
Action host groups
Components
Info
Editing mode
current
17/11/2025 14:55:08
ansible update
17/11/2025 14:53:38
17/11/2025 13:49:58
ansible update
17/11/2025 13:48:08
17/11/2025 13:40:35
ansible update
Search
Expand content
Advanced
Discard changes
Save
Configuration
SSL Configuration
Enable gRPC
Enable LDAP
Kafka engine
Clickhouse Keeper (integrated)

Coordination system: Clickhouse_keeper (integrated)
Number of nodes in cluster: 1
Enable IPv6: true
TCP port: 2129

Enable integrated ClickHouse Keeper

   Click ***Save*** and execute the ***Reconfig*** action for the ADQMDB service to enable the use of the internal ClickHouse Keeper as a coordination service for the ADQM cluster.

Test replicated and distributed tables

Configure ADQM cluster with ClickHouse Keeper

1. Configure a test ADQM cluster — two shards with two replicas in each one (hosts used in this example are *host-1*, *host-2*, *host-3*, and *host-4*). To learn how to do this via the ADCM interface, read Configure logical clusters in the ADCM interface. In this case, all necessary sections described below are added to the configuration file of each server automatically.

   - **Logical cluster configuration**
     In the *config.xml* file, the cluster configuration is defined in the `remote_servers` section:

```
<remote_servers>
    <default_cluster>
        <shard>
            <internal_replication>true</internal_replication>
            <weight>1</weight>
            <replica>
                <host>host-1</host>
                <port>9000</port>
            </replica>
            <replica>
                <host>host-2</host>
                <port>9000</port>
            </replica>
        </shard>
        <shard>
            <internal_replication>true</internal_replication>
            <weight>1</weight>
            <replica>
                <host>host-3</host>
                <port>9000</port>
            </replica>
            <replica>
                <host>host-4</host>
                <port>9000</port>
            </replica>
        </shard>
    </default_cluster>
</remote_servers>
```

     In this example, a ReplicatedMergeTree table is used, so the `internal_replication` parameter is set to `true` for each shard — in this case, the replicated table manages data replication itself (data is written to any available replica, and another replica receives data in the asynchronous mode). If using regular tables, set the `internal_replication` parameter to `false` — in this case, data is replicated by a Distributed table (data is written to all replicas of a shard).

   - **Macros**
     Each server configuration should also contain the `macros` section, which defines the shard and replica identifiers to be replaced with values corresponding to a specific host when creating replicated tables on the cluster (`ON CLUSTER`). For example, macros for *host-1* contained in the `default_cluster` cluster:

```
<macros>
    <replica>1</replica>
    <shard>1</shard>
</macros>
```

     If you configure a logical cluster in the ADCM interface via the ***Cluster Configuration*** setting and specify the cluster name, for example, `abc`, ADQM will write macros to *config.xml* as follows (for *host-1* in the `abc` cluster with the same topology as provided above):

```
<macros>
    <abc_replica>1</abc_replica>
    <abc_shard>1</abc_shard>
</macros>
```

     In this case, use the `{abc_shard}` and `{abc_replica}` variables within ReplicatedMergeTree parameters when creating replicated tables.

2. Install the Clickhousekeeper service to three hosts (*host-1*, *host-2*, *host-3*). After that, the following sections will be added to the *config.xml* file:

   - `zookeeper` — list of ClickHouse Keeper nodes;
   - `distributed_ddl` — path in ClickHouse Keeper to the queue with DDL queries (if multiple clusters use the same ClickHouse Keeper, this path should be unique for each cluster).

```
<zookeeper>
    <node>
        <host>host-1</host>
        <port>2129</port>
    </node>
    <node>
        <host>host-2</host>
        <port>2129</port>
    </node>
    <node>
        <host>host-3</host>
        <port>2129</port>
    </node>
    <root>/clickhouse</root>
</zookeeper>

<distributed_ddl>
    <path>/clickhouse/task_queue/ddl</path>
</distributed_ddl>
```

Replicated tables

1. Execute the following query on one of the hosts (for example, on *host-1*) to create a replicated table:

```
CREATE TABLE test_local ON CLUSTER default_cluster (id Int32, value_string String)
ENGINE = ReplicatedMergeTree('/clickhouse/tables/{shard}/test_local', '{replica}')
ORDER BY id;
```

   This query creates the `test_local` table on all hosts of the `default_cluster` cluster.

2. Insert data into the table on *host-2*:

```
INSERT INTO test_local VALUES (1, 'a');
```

3. Make sure data is replicated. To do this, select data from the `test_local` table on *host-1*:

```
SELECT * FROM test_local;
```

   If replication works correctly, the table on *host-1* will automatically receive data written to the replica on *host-2*:

```
┌─id─┬─value_string─┐
│  1 │ a            │
└────┴──────────────┘
```

Distributed tables

Note that the `SELECT` query returns data only from a table on a host on which the query is being run.

Insert data into any replica on the second shard (for example, into a table on *host-3*):

```
INSERT INTO test_local VALUES (2, 'b');
```

Repeat the `SELECT` query on *host-1* or *host-2* — the resulting selection will not include data from the second shard:

```
┌─id─┬─value_string─┐
│  1 │ a            │
└────┴──────────────┘
```

You can use distributed tables to get data from all shards (for details, see the Distributed tables section).

1. Create a table using the Distributed engine:

```
CREATE TABLE test_distr ON CLUSTER default_cluster AS default.test_local
ENGINE = Distributed(default_cluster, default, test_local, rand());
```

2. Run the following query on any host:

```
SELECT * FROM test_distr;
```

   The output includes data from both shards:

```
┌─id─┬─value_string─┐
│  1 │ a            │
└────┴──────────────┘
┌─id─┬─value_string─┐
│  2 │ b            │
└────┴──────────────┘
```

Usage of clickhouse-copier

Elena Dvoryadkina

ADQM

Contents

To Table of Contents

- How to use clickhouse-copier
  - Step 1. Configure connection to a coordination service
  - Step 2. Configure copy tasks
  - Step 3. Run clickhouse-copier
- Example 1. Sharding a table
  - Prepare source data
  - Copy data
  - Verify
- Example 2. Resharding
  - Prepare source data
  - Copy data
  - Verify

ADQM comes with the adqm-clickhouse-copier package, which is installed by default on all hosts with the ADQMDB service during ADQM installation. This package provides the functionality of clickhouse-copier allowing you to copy large amounts of data between clusters of different topologies and reshard data. This article describes how to create copy tasks and how to run clickhouse-copier to execute them, as well as provides some examples of using this tool.

> **CAUTION**
>
> Clickhouse-copier does not work with the Variant, Dynamic, and JSON data types.

How to use clickhouse-copier

To copy/reshard data via clickhouse-copier, follow the steps below.

Step 1. Configure connection to a coordination service

You can run multiple clickhouse-copier instances on different hosts to perform the same task. To synchronize the copy process across hosts, a coordination service is required — ZooKeeper or ClickHouse Keeper. Create an XML file with parameters for the connection to ZooKeeper/ClickHouse Keeper — copy the `zookeeper` section from the *config.xml* file. Optionally, you can also add the `logger` section with logging settings to it. Copy this file to all hosts where you will run clickhouse-copier.

Example of parameters configuring interaction with a ZooKeeper cluster

The *keeper.xml* file used in the examples within this article below:

```
<clickhouse>
    <zookeeper>
        <node>
            <host>dev-adqm1.ru-central1.internal</host>
            <port>2181</port>
        </node>
        <node>
            <host>dev-adqm2.ru-central1.internal</host>
            <port>2181</port>
        </node>
        <node>
            <host>dev-adqm3.ru-central1.internal</host>
            <port>2181</port>
        </node>
        <root>/clickhouse</root>
    </zookeeper>
</clickhouse>
```

Step 2. Configure copy tasks

Create an XML file with a description of copy tasks for clickhouse-copier (for example, *task.xml*).

Configuration template

```
<clickhouse>
    <remote_servers>
        <source_cluster>
            <shard>
                    <replica>
                        ...
                    </replica>
                    ...
            </shard>
            ...
        </source_cluster>

        <destination_cluster>
            ...
        </destination_cluster>
    </remote_servers>

    <max_workers>2</max_workers>

    <settings_pull>
        <readonly>1</readonly>
    </settings_pull>

    <settings_push>
        <readonly>0</readonly>
    </settings_push>

    <settings>
        <connect_timeout>3</connect_timeout>
        <distributed_foreground_insert>1</distributed_foreground_insert>
    </settings>

    <tables>
        <!-- A task to copy one table -->
        <table_1>
            <cluster_pull>source_cluster</cluster_pull>
            <database_pull>source_database</database_pull>
            <table_pull>test_table</table_pull>

            <cluster_push>destination_cluster</cluster_push>
            <database_push>destination_database</database_push>
            <table_push>test_table_copied</table_push>

            <engine>
            ENGINE=ReplicatedMergeTree('/clickhouse/tables/{shard}/test_table_copied', '{replica}')
            PARTITION BY partition_key
            ORDER BY sorting_key
            </engine>

            <sharding_key>sharding_key_expr</sharding_key>

            <where_condition>where_condition_expr</where_condition>

            <enabled_partitions>
                <partition>'2024-02-01'</partition>
                <partition>'2024-03-02'</partition>
                ...
            </enabled_partitions>
        </table_1>

        <!-- A task to copy the next table -->
        <table_2>
            ...
        </table_2>
        ...
    </tables>
</clickhouse>
```

Configuration parameters

Parameter	Description
remote_servers	Description of the source and destination clusters from the `remote_servers` section of the *config.xml* configuration file (`source_cluster` and `destination_cluster` in the above configuration are the names of these clusters). You can combine ADQM hosts into a logical cluster via ADCM — the cluster description will be automatically added to *config.xml*. For more details, refer to the article Configure logical clusters in the ADCM interface
max_workers	Maximum number of clickhouse-copier instances (workers) possible to perform a copy task simultaneously. If you start more instances, the extra ones will sleep
settings_pull, settings_push	Settings used to fetch (pull) data from tables of the source cluster and insert (push) data into tables of the destination cluster, respectively
settings	Common settings used for fetch and insert operations (overlaid by the `settings_pull` and `settings_push` settings, respectively)
tables	Description of copy tasks. You can specify multiple tasks in the same ZooKeeper/ClickHouse Keeper node — one task to copy one table. For example, there are two tasks specified in the above configuration — `table_1` and `table_2` (task names in the configuration can be arbitrary). The tasks will be executed sequentially, that is, copying the next table will not start until the previous one is copied. For each task, specify the following parameters: ▪ `cluster_pull`, `database_pull`, `table_pull` — source cluster (from the `remote_servers` section), its database, and tables that should be copied; ▪ `cluster_push`, `database_push`, `table_push` — destination cluster (from the `remote_servers` section), its database, and tables into which data should be inserted; ▪ `engine` — engine of destination tables (if a table is not found in the destination cluster, it will be created based on columns of the source table and the table engine specified here); ▪ `sharding_key` — sharding key used to insert data into the destination cluster; ▪ `where_condition` — optional expression to filter data when pulling it from hosts of the source cluster; ▪ `enabled_partitions` — partitions that should be copied (other partitions will be ignored). Partition names should have the same format as the `partition` column of the system.parts table (quoted text). As partition keys in the source and destination clusters can be different, these partition names specify destination partitions. This setting is optional (if it is not specified, all partitions will be copied), but it is strictly recommended to specify partitions explicitly. If there are already some ready partitions in the destination cluster, they will be removed at the start of the copying since they will be interpreted as incomplete data from the previous copying.

Once you have completed the task description, upload it to ZooKeeper/ClickHouse Keeper — create a znode (a path to it should be specific — *<znode-path-to-task>/description*) and save the contents of the *task.xml* file in it. For example, you can do this in the ZooKeeper command line interface (CLI) which is available on each host of a ZooKeeper cluster. Run the *zkCli.sh* script to open ZooKeeper CLI and use the `create` command:

```
$ create /clickhouse/clickhouse ""
```

```
$ create /clickhouse/clickhouse/copier ""
```

```
$ create /clickhouse/clickhouse/copier/task ""
```

```
$ create /clickhouse/clickhouse/copier/task/description "$(cat <path-to>/task.xml)"
```

where `<path-to>` is the path to the *task.xml* file.

Step 3. Run clickhouse-copier

To start a clickhouse-copier instance that will perform a copy task, run the following command:

```
$ clickhouse-copier --config keeper.xml --task-path /clickhouse/copier/task
```

where:

- `config` — path to the *keeper.xml* file with parameters for the connection to ZooKeeper/ClickHouse Keeper;
- `task-path` — path to a znode used to synchronize clickhouse-copier processes and store copy tasks.

The following optional parameters are also available for this command:

- `daemon` — start clickhouse-copier in daemon mode;
- `task-file` — path to a file with the task configuration for initial upload to ZooKeeper/ClickHouse Keeper;
- `task-upload-force` — force `task-file` to be uploaded even if a znode already exists (default is `false`);
- `base-dir` — path to logs and auxiliary files on a host where clickhouse-copier runs (if this parameter is omitted, subdirectories for these files are created in the directory from which clickhouse-copier was launched).

To reduce network traffic, it is recommended to run clickhouse-copier on the same host where the source data is located.

Example 1. Sharding a table

This example shows how to shard a table — copy data from a table replicated on two hosts within one shard to another cluster and distribute it between two shards with two replicas in each one.

Prepare source data

Create the `data_table1` replicated table (within the `default` database) in the `source_cluster1` cluster, which includes one shard with two replicas (this cluster is described in the configuration below):

```
CREATE TABLE data_table1 ON CLUSTER source_cluster1 (value UInt64)
ENGINE = ReplicatedMergeTree('/clickhouse/tables/{shard}/data_table1', '{replica}') ORDER BY value;
```

Generate test data for the table (1 million rows):

```
INSERT INTO data_table1 SELECT number FROM numbers(1000000);
```

In the examples of this article, a destination cluster is `default_cluster` — an example of a logical cluster that is automatically generated when you install ADQM on four hosts with a replication factor set to `2`. In this cluster, create a new database (`test_database`) to which clickhouse-copier will copy data:

```
CREATE DATABASE test_database ON CLUSTER default_cluster;
```

Copy data

Create the *task1.xml* configuration file that describes a task for clickhouse-copier to copy the `data_table1` table of the `source_cluster1` cluster to the `data_table1_copied` table of the `default_cluster` cluster.

task1.xml

```
<clickhouse>
    <remote_servers>
        <source_cluster1>
            <shard>
                <internal_replication>false</internal_replication>
                <weight>1</weight>
                <replica>
                    <host>dev-adqm1.ru-central1.internal</host>
                    <port>9000</port>
                </replica>
                <replica>
                    <host>dev-adqm2.ru-central1.internal</host>
                    <port>9000</port>
                </replica>
            </shard>
        </source_cluster1>
        <default_cluster>
            <shard>
                <internal_replication>true</internal_replication>
                <weight>1</weight>
                <replica>
                    <host>dev-adqm1.ru-central1.internal</host>
                    <port>9000</port>
                </replica>
                <replica>
                    <host>dev-adqm2.ru-central1.internal</host>
                    <port>9000</port>
                </replica>
            </shard>
            <shard>
                <internal_replication>true</internal_replication>
                <weight>1</weight>
                <replica>
                    <host>dev-adqm3.ru-central1.internal</host>
                    <port>9000</port>
                </replica>
                <replica>
                    <host>dev-adqm4.ru-central1.internal</host>
                    <port>9000</port>
                </replica>
            </shard>
        </default_cluster>
    </remote_servers>

    <max_workers>2</max_workers>

    <settings_pull>
        <readonly>1</readonly>
    </settings_pull>

    <settings_push>
        <readonly>0</readonly>
    </settings_push>

    <settings>
        <connect_timeout>3</connect_timeout>
        <distributed_foreground_insert>1</distributed_foreground_insert>
    </settings>

    <tables>
        <table_local>
            <cluster_pull>source_cluster1</cluster_pull>
            <database_pull>default</database_pull>
            <table_pull>data_table1</table_pull>

            <cluster_push>default_cluster</cluster_push>
            <database_push>test_database</database_push>
            <table_push>data_table1_copied</table_push>

            <engine>
            ENGINE=ReplicatedMergeTree('/clickhouse/tables/{shard}/data_table1_copied', '{replica}')
            ORDER BY (value)
            </engine>

            <sharding_key>rand()</sharding_key>
        </table_local>
    </tables>
</clickhouse>
```

Upload the *task1.xml* configuration to the `/clickhouse/clickhouse/copier/task1/description` znode of the ZooKeeper cluster specified in *keeper.xml*.

Run clickhouse-copier from the directory where the *keeper.xml* file is located:

```
$ clickhouse-copier --config keeper.xml --task-path /clickhouse/copier/task1
```

Verify

As a result, a replicated table is created in the `test_database` database on all hosts of the `default_cluster` cluster. You can check this as follows:

```
SELECT create_table_query FROM system.tables where name = 'data_table1_copied' and database ='test_database';
```

```
┌─create_table_query──────────────────────────────────────────────────────────────────────────────────────────
─────────────┐
│ CREATE TABLE test_database.data_table1_copied (`value` UInt64)
│
│ ENGINE = ReplicatedMergeTree('/clickhouse/tables/{shard}/data_table1_copied', '{replica}') ORDER BY value SETTINGS
index_granularity = 8192   │
└─────────────────────────────────────────────────────────────────────────────────────────────────────────────
─────────────┘
```

Data is distributed about equally between two shards of the `default_cluster` cluster — the table below shows queries for verification that you can execute on hosts of each shard.

Query	Shard 1	Shard 2
`SELECT count() FROM test_database.data_table1_copied;`	count() 499045	count() 500955
`SELECT * FROM test_database.data_table1_copied ORDER BY value ASC;`	value 2 4 5 7 9 10 ...	value 0 1 3 6 8 11 ...

To send `SELECT` queries to all shards at once, you can use a distributed table:

```
CREATE TABLE test_database.distr_table1 ON CLUSTER default_cluster AS test_database.data_table1_copied
ENGINE = Distributed(default_cluster, test_database, data_table1_copied, rand());
```

```
SELECT * FROM test_database.distr_table1 WHERE value < 10;
```

```
┌─value─┐
│     4 │
└───────┘
┌─value─┐
│     7 │
└───────┘
┌─value─┐
│     2 │
│     5 │
│     9 │
└───────┘
┌─value─┐
│     1 │
└───────┘
┌─value─┐
│     6 │
│     8 │
└───────┘
┌─value─┐
│     0 │
│     3 │
└───────┘
```

Example 2. Resharding

This example shows how to reshard data with clickhouse-copier — copy data distributed across three shards of the `source_cluster2` cluster (one replica in each shard) to the `default_cluster` cluster with two shards and two replicas in each shard.

Prepare source data

Create the `data_table2` table in the `source_cluster2` cluster (in the `default` database):

```
CREATE TABLE data_table2 ON CLUSTER source_cluster2 (value UInt64) ENGINE = MergeTree() ORDER BY value;
```

Create a distributed table that will access the `data_table2` tables and allow execution of distributed queries on multiple servers:

```
CREATE TABLE distr_table2 ON CLUSTER source_cluster2 AS default.data_table2
ENGINE = Distributed(source_cluster2, default, data_table2, rand());
```

Insert data to the `data_table2` tables. To do this, send the `INSERT` query to a distributed table — it will distribute data randomly across shards:

```
INSERT INTO distr_table2 SELECT number FROM numbers(1000000);
```

The following table shows the data distribution between three shards.

Query	Shard 1	Shard 2	Shard 3
`SELECT count() FROM data_table2;`	count() 332973	count() 334408	count() 332619

Copy data

Create an XML configuration of the copy task for clickhouse-copier (the *task2.xml* file).

task2.xml

```
<clickhouse>
    <remote_servers>
        <source_cluster2>
            <shard>
                <internal_replication>false</internal_replication>
                <weight>1</weight>
                <replica>
                    <host>dev-adqm1.ru-central1.internal</host>
                    <port>9000</port>
                </replica>
            </shard>
            <shard>
                <internal_replication>false</internal_replication>
                <weight>1</weight>
                <replica>
                    <host>dev-adqm2.ru-central1.internal</host>
                    <port>9000</port>
                </replica>
            </shard>
            <shard>
                <internal_replication>false</internal_replication>
                <weight>1</weight>
                <replica>
                    <host>dev-adqm3.ru-central1.internal</host>
                    <port>9000</port>
                </replica>
            </shard>
        </source_cluster2>
        <default_cluster>
            <shard>
                <internal_replication>true</internal_replication>
                <weight>1</weight>
                <replica>
                    <host>dev-adqm1.ru-central1.internal</host>
                    <port>9000</port>
                </replica>
                <replica>
                    <host>dev-adqm2.ru-central1.internal</host>
                    <port>9000</port>
                </replica>
            </shard>
            <shard>
                <internal_replication>true</internal_replication>
                <weight>1</weight>
                <replica>
                    <host>dev-adqm3.ru-central1.internal</host>
                    <port>9000</port>
                </replica>
                <replica>
                    <host>dev-adqm4.ru-central1.internal</host>
                    <port>9000</port>
                </replica>
            </shard>
        </default_cluster>
    </remote_servers>

    <max_workers>2</max_workers>

    <settings_pull>
        <readonly>1</readonly>
    </settings_pull>

    <settings_push>
        <readonly>0</readonly>
    </settings_push>

    <settings>
        <connect_timeout>3</connect_timeout>
        <distributed_foreground_insert>1</distributed_foreground_insert>
    </settings>

    <tables>
        <table_local>
            <cluster_pull>source_cluster2</cluster_pull>
            <database_pull>default</database_pull>
            <table_pull>data_table2</table_pull>

            <cluster_push>default_cluster</cluster_push>
            <database_push>test_database</database_push>
            <table_push>data_table2_copied</table_push>

            <engine>
            ENGINE=ReplicatedMergeTree('/clickhouse/tables/{shard}/data_table2_copied', '{replica}')
            ORDER BY (value)
            </engine>

            <sharding_key>rand()</sharding_key>
        </table_local>
    </tables>
</clickhouse>
```

Upload the *task2.xml* configuration to the `/clickhouse/clickhouse/copier/task2/description` znode.

Run clickhouse-copier from the directory where *keeper.xml* is stored:

```
$ clickhouse-copier --config keeper.xml --task-path /clickhouse/copier/task2
```

Verify

Check that the `data_table2_copied` table has been added to the `test_database` database:

```
SHOW tables FROM test_database;
```

```
┌─name───────────────┐
│ data_table1_copied │
│ data_table2_copied │
└────────────────────┘
```

See how 1 million data rows from the source table were distributed between two shards of the destination cluster.

Query	Shard 1	Shard 2
`SELECT count() FROM test_database.data_table2_copied;`	count() 500171	count() 499829
`SELECT * FROM test_database.data_table2_copied limit 10;`	value 3 5 7 8 11 13 14 18 20 22	value 0 1 2 4 6 9 10 12 15 16

Configure data storages

Elena Dvoryadkina

Contents

To Table of Contents

Overview

Disks

Storage policies

Assign storage policy to table

Example of tiered storage configuration

ADQM can store table data on multiple disks of various types, group them into volumes, and automatically move data between volumes according to the specified conditions.

Overview

ADQM supports the following types of storages for table data:

- **Local file system**. It is possible to store data on multiple block devices — for example, it can be useful when table data is divided into "hot" (the most recent data that is regularly queried and requires a small amount of space) and "cold" (historical data that is queried rarely). In this case, the "hot" data can be located on fast local disks (for example, on NVMe SSD disks or in memory), while the "cold" data — on slower ones (for example, HDD). The ability to add new devices also simplifies the storage extension process.
- **External S3 disks**. This storage option provides the ability to store huge amounts of information without worrying about the fault tolerance, reliability, and scalability of the storage.

It is also possible to create **cache** over disks that store data (to increase the speed of fetching data from external disks with high latency) and **encrypted disks** (to encrypt sensitive data of the ADQM cluster).

Basic concepts used when configuring a data storage in ADQM:

- **Disk** — additional block device mounted on the file system (local disk), external S3 storage, cache or encrypted disk.
- **Volume** — ordered set of equal disks.
- **Storage policy** — set of volumes and the specified rules to write table data.

General scheme of work with storages in ADQM:

1. Create and configure disks.
2. Create a storage policy that specifies disks to store data.
3. Assign a storage policy to a table.

To configure disks, volumes, and storage policies, use configuration parameters of the ADQMDB service (see the corresponding sections below). Specified parameter values are stored within the `<storage_configuration>` tag in the */etc/clickhouse-server/config.d/storage.xml* file.

Disks

To manage disks in ADQM, use the ADCM interface:

1. Open the ADQMDB service's configuration page (***ADQM cluster*** → ***Services*** → ***ADQMDB*** → ***Primary Configuration***).
2. Enable the ***Show advanced*** option.
3. Activate a group that corresponds to a particular disk type:
   - ***Enable additional local storage*** — local disks;
   - ***Enable S3 storage*** — S3 disks;
   - ***Enable cache*** — cache disks;
   - ***Enable encrypted storage*** — encrypted disks.
4. In the expanded section, you can add/remove and configure disks of the corresponding type. Parameters available for each disk type are described below.

   Local disks

Parameter	Description
disk_name	Disk name (should be unique)
disk_path	Absolute path to the directory where the block device is mounted. When specifying a relative path (without the initial `/`), an error occurs informing that the directory does not exist
keep_free_space_bytes	Amount of free disk space to be reserved (in bytes)

   If the ***Recursively create directories*** option is enabled, ADQM will automatically create the specified directories for local disks (if these directories do not exist) and assign the `clickhouse:clickhouse` owner with the `0770` permission mask to them.

   S3 disks

Parameter	Description
disk_name	Disk name (should be unique)
endpoint	Bucket URL with the path to the root directory on the server where the data is stored
access_key_id	S3 access key id
secret_access_key	S3 access secret key
proxy	List of proxy servers
custom_parameters	Other parameters (for a full list of parameters available for S3 disks, see Configuring the S3 disk in the ClickHouse documentation)

   Cache disks

Parameter	Description
parent_disk_name	Name of the parent disk (external S3 disk) over which the cache is created
cache_disk_name	Cache disk name (should be unique)
cache_disk_path	Absolute path to the directory to which the cache will be written
max_size_bytes	Maximum size of the cache (in bytes)
cache_on_write_operations	Enables caching data on any write operations (`INSERT` queries, background merges)
enable_filesystem_query_cache_limit	Allows limiting the size of the cache which is downloaded within each query (depends on the `max_query_cache_size` user setting)
enable_cache_hits_threshold	Number that defines how many times data needs to be read before it is cached
do_not_evict_index_and_mark_files	Allows you not to evict small frequently used files according to the cache policy
max_file_segment_size_bytes	Maximum size of a cache file (in bytes)
max_elements	Limit for a number of cache files

   If the ***Recursively create directories*** option is enabled, ADQM will automatically create the specified directories for storing cache (if these directories do not exist) and assign the `clickhouse:clickhouse` owner with the `0770` permission mask to them.

   Encrypted disks

Parameter	Description
parent_disk_name	Name of the parent disk over which the encrypted file is created
encrypted_disk_name	Encrypted disk name (should be unique)
encrypted_disk_path	Absolute or relative path to the directory on the disk where encrypted data will be saved. When specifying a relative path, a subdirectory is created on the parent disk
encryption_algorithm	Data encryption algorithm. Possible values: `AES_128_CTR`, `AES_192_CTR`, `AES_256_CTR`
encryption_key_hex	Key as a hexadecimal value used for encryption (for example, `00112233445566778899AABBCCDDEEFF`)

   If the ***Recursively create directories*** option is enabled, ADQM will automatically create the specified directories for storing encrypted data (if these directories do not exist) and assign the `clickhouse:clickhouse` owner with the `0770` permission mask to them.

   You can use the ***Enable…*** switch to temporarily disable a group of disks by the type, but keep in mind that a storage policy cannot refer to an unconfigured disk.
5. Click ***Save*** and execute the ***Reconfig*** action for the ADQMDB service to save the information about the added disks in the configuration (to modify existing disks, it is also required to restart the service — to do this, enable the ***Restart*** option in the ***Reconfig*** action configuration window).

To view information on disks configured in ADQM, you can use the system.disks system table:

```
SELECT
    name,
    path,
    formatReadableSize(free_space) AS free,
    formatReadableSize(total_space) AS total,
    formatReadableSize(keep_free_space) AS reserved
FROM system.disks;
```

When installing ADQM, one local disk (`default`) is configured by default. Data is saved to the directory specified via the ***Data path*** parameter in the ***Directories*** section on the ADQMDB service's configuration page (default is */var/lib/clickhouse/*).

```
┌─name────┬─path─────────────────┬─free──────┬─total─────┬─reserved─┐
│ default │ /var/lib/clickhouse/ │ 39.33 GiB │ 49.99 GiB │ 0.00 B   │
└─────────┴──────────────────────┴───────────┴───────────┴──────────┘
```

Storage policies

A storage policy groups disks into one or more volumes, defines rules to write table data on disks, and specifies how data should be moved between volumes or individual disks within volumes.

You can add and configure storage policies in the ADCM interface:

1. Go to the ADQMDB service's configuration page (***ADQM cluster*** → ***Services*** → ***ADQMDB*** → ***Primary Configuration***).
2. Enable the ***Show advanced*** option.
3. In the ***Storage policies*** section, add required storage policies to the ***Policies*** list.

   Storage policy settings

Parameter	Description
policy_name	Storage policy name (should be unique)
move_factor	Proportion of the free space to be available on a volume. When the amount of available space becomes less than the specified factor, data will automatically start moving to the next volume (if any). To move data parts, ADQM sorts existing data parts by size in the descending order and selects parts whose total size is sufficient to satisfy the `move_factor` condition. If the total size of all parts is insufficient, all parts will be moved
volumes	List of policy volumes (a policy should contain at least one volume). Volume configuration parameters are: ■ ***volume_name*** — volume name (should be unique). ■ ***max_data_part_size_bytes*** — maximum size of a data part that can be stored on any disk of the volume. If a merged part size is expected to be larger than the `max_data_part_size_bytes` value, this part will be written to the next volume. Basically this feature allows you to keep small data parts (new data) on a "hot" volume (SSD) and move them to a "cold" volume (HDD) when they become large (see the example below). Do not use this option if a policy has only one volume. ■ ***prefer_not_to_merge*** — disables merging data parts stored on the volume. This allows you to control how ADQM works with slow disks. ■ ***volume_disks*** — names of disks to be grouped into the volume (a volume should contain at least one disk). If a cache disk is created over a disk with data, then the cache disk should be specified for a volume. The order of volumes in the storage policy is important. Once a volume is overfilled, data is migrated to the next one. The order of disks in a volumes is also important — data is written to disks in turn.

4. Click ***Save*** and execute the ***Reconfig*** action for the ADQMDB service to save the information about the added storage policies in the configuration (to modify existing storage policies, it is also required to restart the service — to do this, enable the ***Restart*** option in the ***Reconfig*** action configuration window).

You can get information on storage policies configured in ADQM from the system.storage_policies system table:

```
SELECT policy_name, volume_name, disks FROM system.storage_policies;
```

By default, ADQM contains a single storage policy (`default`) with a single volume, which specifies that all table data is written to the `default` disk:

```
┌─policy_name─┬─volume_name─┬─disks───────┐
│ default     │ default     │ ['default'] │
└─────────────┴─────────────┴─────────────┘
```

Assign storage policy to table

After storage policies are configured, you can assign them to tables. To do this, specify a storage policy name as the `storage_policy` parameter value using the `SETTINGS` clause when creating a table. The basic syntax of a query to create a MergeTree table with a specified storage policy can be written as follows:

```
CREATE TABLE <table_name> (<column_definitions>)
ENGINE = MergeTree
ORDER BY <sorting_key>
SETTINGS storage_policy = '<policy_name>';
```

By default, a table uses the `default` storage policy.

You can see which storage policy is assigned to a table and where the table data is stored in the `storage_policy` and `data_paths` fields of the system.tables system table. For example, run the following queries to create a table without specifying a storage policy, populate it with test data, and display information about the assigned storage policy and directory where data is written:

```
CREATE TABLE test_table (id Int32) Engine=MergeTree ORDER BY id;
```

```
INSERT INTO test_table SELECT rand() FROM numbers(10000000);
```

```
SELECT name, storage_policy, data_paths FROM system.tables WHERE name = 'test_table' FORMAT Vertical;
```

```
Row 1:
──────
name:           test_table
storage_policy: default
data_paths:     ['/var/lib/clickhouse/store/50d/50d13b59-89d8-44ef-87f6-08ee8f71066c/']
```

To change the storage policy after the table has been created, use the `ALTER TABLE … MODIFY SETTING` query. Note that a new policy should contain all volumes and disks of the previous policy with the same names.

Example of tiered storage configuration

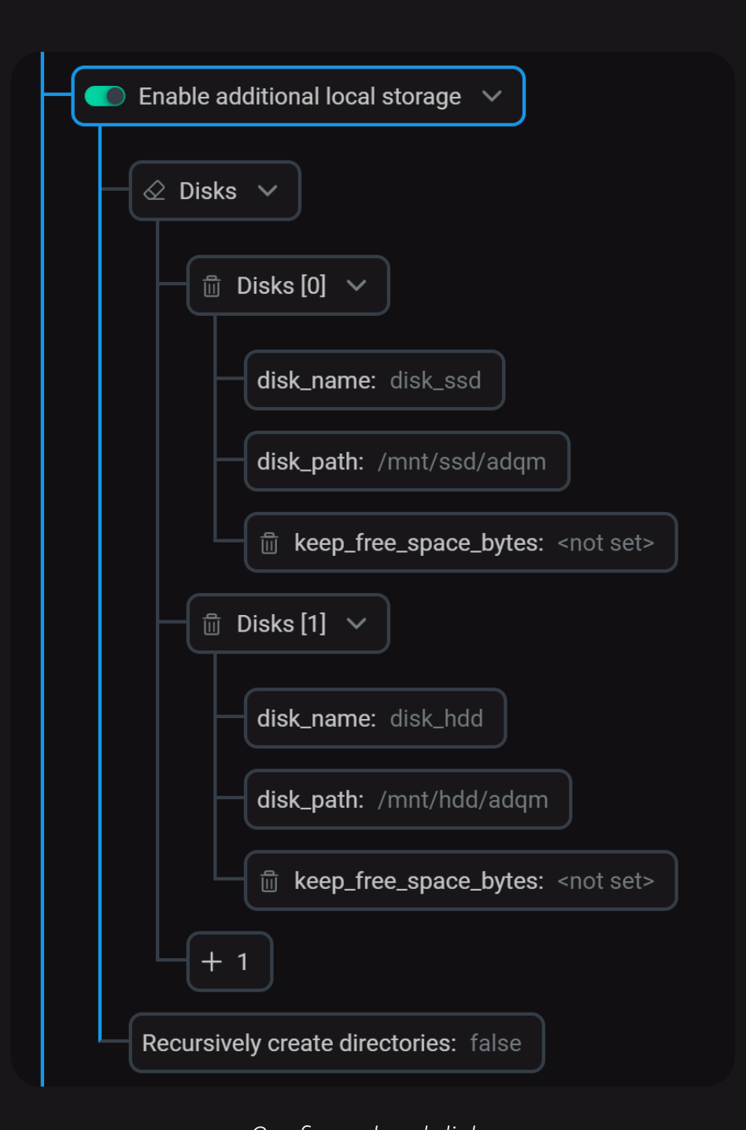
In this example, two block devices of different types are used for data storage — SSD and HDD disks mounted on the following directories:

- */mnt/ssd/adqm/*
- */mnt/hdd/adqm/*

Each directory is assigned the `clickhouse` owner:

```
$ sudo chown clickhouse:clickhouse -R /mnt/ssd/adqm/ /mnt/hdd/adqm/
```

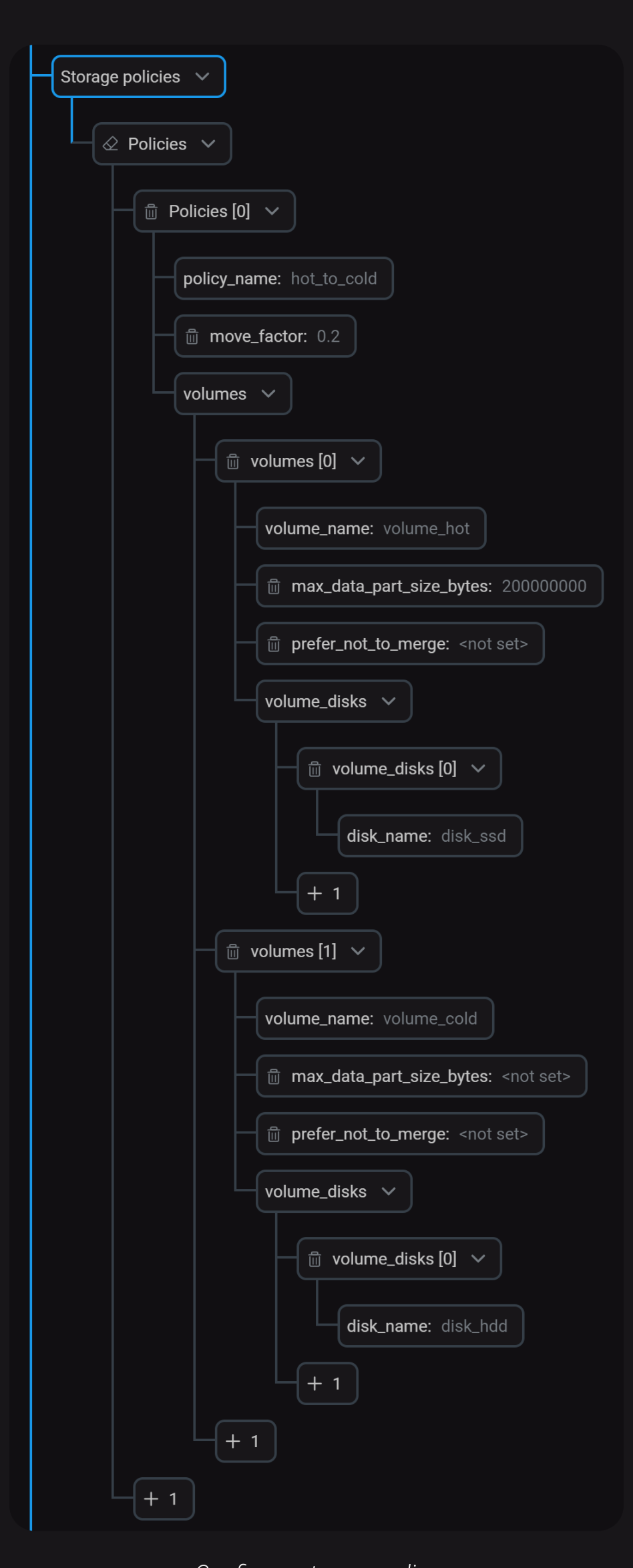
1. Specify local disks in the ***Enable additional local storage*** section (previously enabled) of the ADQMDB service's configuration page.



disk_name: disk_ssd
disk_path: /mnt/ssd/adqm
keep_free_space_bytes: <not set>
disk_name: disk_hdd
disk_path: /mnt/hdd/adqm
keep_free_space_bytes: <not set>
Recursively create directories: false

Configure local disks

2. In the ***Storage policies*** section, create a policy (`hot_to_cold`) that includes two volumes for storing "hot" and "cold" data.



policy_name: hot_to_cold
move_factor: 0.2
volume_name: volume_hot
max_data_part_size_bytes: 200000000
prefer_not_to_merge: <not set>
disk_name: disk_ssd
volume_name: volume_cold
max_data_part_size_bytes: <not set>
prefer_not_to_merge: <not set>
disk_name: disk_hdd

Configure storage policy

   The `volume_hot` volume contains one SSD disk (`disk_ssd`). The maximum size of a data part that can be stored on this volume is 200 MB. All data parts with the size larger than 200 MB will be moved to the `volume_cold` volume, which contains one HDD disk (`disk_hdd`). The MergeTree table engine constantly performs background data merges, merging newly inserted small data parts into larger parts over time. That is, large data parts appear after several merges, respectively, we can assume that the larger the part, the older it is.

   Also, once the `disk_ssd` disk is more than 80% full, data will be transferred to the `disk_hdd` disk in the background.

3. Click ***Save*** and run the ***Reconfig*** action for the service to apply the changes.
   After that, information about disks and policies will appear in the appropriate tags inside the `<storage_configuration>` section in the */etc/clickhouse-server/config.d/storage.xml* file.

```
<storage_configuration>
    <disks>
        <disk_ssd>
            <type>local</type>
            <path>/mnt/ssd/adqm/</path>
        </disk_ssd>
        <disk_hdd>
            <type>local</type>
            <path>/mnt/hdd/adqm/</path>
        </disk_hdd>
    </disks>
    <policies>
        <hot_to_cold>
            <volumes>
                <volume_hot>
                    <disk>disk_ssd</disk>
                    <max_data_part_size_bytes>200000000</max_data_part_size_bytes>
                </volume_hot>
                <volume_cold>
                    <disk>disk_hdd</disk>
                </volume_cold>
            </volumes>
            <move_factor>0.2</move_factor>
        </hot_to_cold>
    </policies>
</storage_configuration>
```

4. Create a table that uses the new storage policy:

```
CREATE TABLE test_table1 (id Int32) Engine=MergeTree ORDER BY id SETTINGS storage_policy = 'hot_to_cold';
```

   Populate the table with test data by repeating the following query for several times:

```
INSERT INTO test_table1 SELECT rand() FROM numbers(10000000);
```

5. Wait a bit and run the following query to see how the table data is distributed across disks:

```
SELECT
    disk_name,
    formatReadableSize(bytes_on_disk) AS size
FROM system.parts
WHERE (table = 'test_table1') AND active;
```

   As a result of merges, a large data part has been created and then was moved to the "cold" disk.

```
┌─disk_name─┬─size───────┐
│ disk_hdd  │ 530.49 MiB │
│ disk_ssd  │ 22.26 MiB  │
│ disk_ssd  │ 22.26 MiB  │
└───────────┴────────────┘
```

System tables

Elena Dvoryadkina

Contents

To Table of Contents

Overview

List of system tables

Overview

ADQM system tables are tables in the `system` database providing different kind of information that can be useful to administrators and developers for database management, query optimization, system performance monitoring, and error analysis and troubleshooting. For example, system tables contain:

- information on server states and internal processes;
- details on configured clusters, existing databases, tables, columns, and data parts;
- ClickHouse server metrics collected in real time and events occurred in the system;
- current values of configuration parameters;
- help information about options available for building queries.

Most of the system tables store their data in RAM. These tables are automatically created when the ClickHouse server starts, and they are read-only. They cannot be deleted or altered but can be detached.

System tables also include tables storing different logs — log tables named as `*_log`. If you activate logging, ADQM creates and starts populating the corresponding log table, which becomes accessible from the `system` database. Unlike other system tables, log tables are based on the MergeTree table engine. Data of these tables is stored in the file system and is not automatically deleted — this guarantees access to logs for analysis after the server restart. The how-to:logging.adoc#log_tables article lists types of logs that can be written to system tables and explains how to enable logging and configure log tables.

List of system tables

You can view the full list of available system tables using the query:

```sql
SHOW TABLES FROM system;
```

List of available system tables

For example, the output can look like:

```
    ┌─name───────────────────────────┐
 1. │ aggregate_function_combinators │
 2. │ asynchronous_inserts           │
 3. │ asynchronous_loader            │
 4. │ asynchronous_metrics           │
 5. │ backups                        │
 6. │ build_options                  │
 7. │ certificates                   │
 8. │ clusters                       │
 9. │ collations                     │
10. │ columns                        │
11. │ contributors                   │
12. │ current_roles                  │
13. │ dashboards                     │
14. │ data_skipping_indices          │
15. │ data_type_families             │
16. │ database_engines               │
17. │ databases                      │
18. │ detached_parts                 │
19. │ dictionaries                   │
20. │ disks                          │
21. │ distributed_ddl_queue          │
22. │ distribution_queue             │
23. │ dns_cache                      │
24. │ dropped_tables                 │
25. │ dropped_tables_parts           │
26. │ enabled_roles                  │
27. │ errors                         │
28. │ events                         │
29. │ filesystem_cache               │
30. │ formats                        │
31. │ functions                      │
32. │ generateSeries                 │
33. │ generate_series                │
34. │ grants                         │
35. │ graphite_retentions            │
36. │ jemalloc_bins                  │
37. │ kafka_consumers                │
38. │ keywords                       │
39. │ licenses                       │
40. │ macros                         │
41. │ merge_tree_settings            │
42. │ merges                         │
43. │ metrics                        │
44. │ models                         │
45. │ moves                          │
46. │ mutations                      │
47. │ mysql_binlogs                  │
48. │ named_collections              │
49. │ numbers                        │
50. │ numbers_mt                     │
51. │ one                            │
52. │ part_moves_between_shards      │
53. │ parts                          │
54. │ parts_columns                  │
55. │ privileges                     │
56. │ processes                      │
57. │ projection_parts               │
58. │ projection_parts_columns       │
59. │ query_cache                    │
60. │ quota_limits                   │
61. │ quota_usage                    │
62. │ quotas                         │
63. │ quotas_usage                   │
64. │ remote_data_paths              │
65. │ replicas                       │
66. │ replicated_fetches             │
67. │ replicated_merge_tree_settings │
68. │ replication_queue              │
69. │ rocksdb                        │
70. │ role_grants                    │
71. │ roles                          │
72. │ row_policies                   │
73. │ s3queue                        │
74. │ scheduler                      │
75. │ schema_inference_cache         │
76. │ server_settings                │
77. │ settings                       │
78. │ settings_changes               │
79. │ settings_profile_elements      │
80. │ settings_profiles              │
81. │ stack_trace                    │
82. │ storage_policies               │
83. │ symbols                        │
84. │ table_engines                  │
85. │ table_functions                │
86. │ tables                         │
87. │ time_zones                     │
88. │ user_directories               │
89. │ user_processes                 │
90. │ users                          │
91. │ view_refreshes                 │
92. │ warnings                       │
93. │ zeros                          │
94. │ zeros_mt                       │
95. │ zookeeper                      │
96. │ zookeeper_connection           │
    └────────────────────────────────┘
```

This list does not include `*_log` tables as logging has not been enabled.

Below are ADQM system tables grouped by the type of information they provide.

Server states | Current processes | Caches

System table	Table data
merges	Information about data part merges and mutations that are currently being performed for MergeTree tables
mutations	Information about mutations of MergeTree tables and their progress
processes	Information about all queries that are currently running
moves	Information about in-progress data part movements between disks or volumes of a storage (initiated, for example, by a TTL rule or the `ALTER TABLE … MOVE PARTITION|PART … TO DISK|VOLUME` query)
replicated_fetches	Information about currently running background fetches of data parts from replicas
stack_trace	Stack traces of all server threads (can be used to introspect the server state)
asynchronous_inserts	Information about pending asynchronous inserts in a queue
asynchronous_loader	Information and status for recent asynchronous jobs (for example, for loading tables)
query_cache	Information about the content of the query cache
dns_cache	Information about cached DNS records
filesystem_cache	Information about all entries in filesystem cache for remote objects

Clusters | Sharding | Replication | ZooKeeper

System table	Table data
clusters	Information about configured clusters
macros	Macros that will be used to automatically substitute host-specific shard and replica identifiers when creating replicated tables on clusters (macros for all clusters are defined in the `macros` section of the server configuration)
replicas	Information about replicated tables located on the server
replication_queue	Information about tasks from replication queues stored in ZooKeeper or ClickHouse Keeper for ReplicationMergeTree tables
distribution_queue	Information about local files that are in a queue to be sent to shards (these local files contain new data parts that are created by inserting new data into the Distributed table in asynchronous mode)
distributed_ddl_queue	Information about distributed DDL queries (with the `ON CLUSTER` clause) that were executed on the cluster
zookeeper	Data from the ZooKeeper or ClickHouse Keeper cluster defined in the configuration (allows you to get a list of children for a specific node or node value)
zookeeper_connection	Current connections to ZooKeeper or ClickHouse Keeper

Examples

- Obtain information on configured logical clusters:

```sql
SELECT cluster, shard_num, replica_num, host_name FROM system.clusters;
```

- Check that replicas are healthy:

```sql
SELECT database, table, last_queue_update_exception, zookeeper_exception FROM system.replicas WHERE is_readonly;
```

If a table is read-only (for example, because there is no ClickHouse Keeper/ZooKeeper configuration section in *config.xml* or an error occurred when re-initializing a session in ClickHouse Keeper/ZooKeeper), you can view error messages in the `last_queue_update_exception` and `zookeeper_exception` fields.

- Check the replication queue status:

```sql
SELECT database, table, replica_name, type, num_postponed, postpone_reason, num_tries, last_exception
FROM system.replication_queue FORMAT Vertical;
```

Databases | Tables | Columns | Data parts | Dictionaries

System table	Table data
databases	Information about databases available to the current user
tables	Metadata of tables the server "knows" about (tables detached via `DETACH TABLE` are not included)
columns	Information about all columns of all tables
parts	Information about currently existing data parts of all MergeTree tables
parts_columns	Information about columns of all currently existing data parts of MergeTree tables
detached_parts	Information about detached data parts of MergeTree tables
dropped_tables	Information about tables dropped from databases but not completely removed yet
dropped_tables_parts	Information about data parts of dropped MergeTree tables (listed in `system.dropped_tables`)
projection_parts	Information on currently existing data parts created for all projections of all tables within a cluster (copies of parent parts containing data aggregated or sorted in a different order). This table is similar to `system.parts`, but contains additional information about parent data parts in columns prefixed with `parent_`
projection_parts_columns	Information about columns of all currently existing projection data parts of all MergeTree tables
data_skipping_indices	Information about existing data skipping indexes in all tables
dictionaries	Information about dictionaries
named_collections	Configured named collections — collections of key/value pairs to be used to configure integrations with external sources (dictionaries, tables, table functions)

Examples

- View table columns with their data types:

```sql
SELECT name, type FROM system.columns WHERE table = '<table_name>';
```

- View information about table data parts:

```sql
SELECT name, active, rows, data_compressed_bytes, data_uncompressed_bytes
FROM system.parts WHERE table = '<table_name>'
FORMAT Vertical;
```

where `<table_name>` is a table name.

Access control system

System table	Table data
users	Information about user accounts configured on the server
grants	Privileges granted to user accounts (the list of privileges available for assignment can be found in the `system.privileges` table)
roles, role_grants, current_roles, enabled_roles	Information about roles and granting roles to users
row_policies	Information about row policies
settings_profiles	Configured settings profiles
settings_profile_elements	Contents of settings profiles (including settings and their values; constraints; roles and users for which the settings are applied; parent settings profiles)
quota_limits, quota_usage, quotas, quotas_usage	Information about quotas and quota usage

Data storage

System table	Table data
disks	Information about configured disks
storage_policies	Information about configured storage policies

Examples

- View information about data storage disks:

```sql
SELECT
    name,
    path,
    formatReadableSize(free_space) AS free,
    formatReadableSize(total_space) AS total,
    formatReadableSize(keep_free_space) AS reserved
FROM system.disks;
```

- View information about storage policies configured in ADQM:

```sql
SELECT policy_name, volume_name, disks FROM system.storage_policies;
```

ClickHouse server metrics | Errors and warnings

System table	Table data
metrics	Metrics that are calculated instantly and have up-to-date current values (see ***Metrics*** in the ***Metric types*** section of the ADQM monitoring metrics article)
events	Information about the number of events that have occurred in the system (see ***ProfileEvents*** in the ***Metric types*** section of the ADQM monitoring metrics article)
asynchronous_metrics	Metrics that are periodically calculated in the background (see ***AsynchronousMetrics*** in the ***Metric types*** section of the ADQM monitoring metrics article)
user_processes	Information about memory usage by processes a user has started and ProfileEvents for the user
errors	Information about all errors that have ever occurred (including error code and message, number of occurrences, the last occurrence time, and stack trace for debugging)
warnings	Warnings about server configuration (if the table is not empty, clickhouse-client displays warning messages stored in the table on connection to the ClickHouse server)
backups	Information about all `BACKUP` and `RESTORE` operations (the table is not persistent, and it contains only operations executed after the last server restart)
jemalloc_bins	Information about memory allocation made via jemalloc in different size classes (bins) aggregated from all arenas. These statistics may not be absolutely accurate because of thread local caching in jemalloc

Examples

- View errors:

```sql
SELECT name, code, value, last_error_time FROM system.errors;
```

- See a value of a specific metric (for example, `Query` — the number of queries being executed):

```sql
SELECT * FROM system.metrics WHERE metric = 'Query';
```

Settings

System table	Table data
settings	User-level settings (which can be changed for an individual query or session)
server_settings	Global server settings specified in *config.xml*
merge_tree_settings	List of all MergeTree table engine settings, their current and default values, as well as descriptions. You can configure these parameters when creating or modifying a table with the `SETTINGS` clause in the `CREATE TABLE` or `ALTER TABLE` query, respectively
replicated_merge_tree_settings	List of all ReplicatedMergeTree table engine settings, their current and default values, as well as descriptions. You can configure these parameters when creating or modifying a table with the `SETTINGS` clause in the `CREATE TABLE` or `ALTER TABLE` query, respectively
settings_changes	Information about changes of settings in different versions of ClickHouse
build_options	ClickHouse server build options

Examples

View which settings have been changed from default values (for example, during troubleshooting, it can be useful to analyze whether any changed settings could affect system behavior):

```sql
SELECT name, value FROM system.settings WHERE changed;
```

```sql
SELECT name, value FROM system.server_settings WHERE changed;
```

Information related to different table engines

System table	Table data
mysql_binlogs	List of active binlogs for the MaterializedMySQL database engine
kafka_consumers	Information about Kafka consumers (tables based on the Kafka engine)
rocksdb	RocksDB statistics (for the integration via the EmbeddedRocksDB table engine)
graphite_retentions	Information about the `graphite_rollup` parameters used in the GraphiteMergeTree tables
s3queue	In-memory state of S3Queue: which files are currently being processed, which files are processed or failed

Data for testing

ADQM provides a set of tables that can be useful for testing and generating successive values.

System table	Table data
numbers	The table contains a single `number` column (of the UInt64 type) with all natural numbers starting from `0` in sorted order. Reading from this table is not parallelized. You can get this table from the `system` database or using the numbers table function
numbers_mt	The table is similar to `system.numbers`, but reading is parallelized. Numbers can be returned in any order
zeros	The table contains a single `zero` column (of the Int64 type) with zeros. You can use it for performance tests instead of `system.numbers` as the fastest way to generate a large number of rows
zeros_mt	The table is similar to `system.zeros`, but reading is parallelized
generate_series, generateSeries	The table contains a single `generate_series` column (of the UInt64 type) with all natural numbers from `0` in sorted order. To get the same table with the arithmetic progression of sorted natural numbers in the specified range with the specified step, you can also use the generate_series or generateSeries table function
one	The table contains `0` in a single row with a single `dummy` column of the UInt8 type. It is used if the `SELECT` query does not specify the `FROM` clause — for example, `SELECT 1`. This table is similar to the `DUAL` table in other DBMSs

Help information

System table	Table data
functions	Supported functions (normal and aggregate)
aggregate_function_combinators	Combinators for aggregate functions
table_functions	Supported table functions
database_engines	Supported database engines
table_engines	Supported table engines and their functionality
data_type_families	Supported data types
formats	Supported input and output formats
privileges	Supported types of privileges that can be granted to users
keywords	Keywords that can be used in query syntax
time_zones	Time zones the ClickHouse server supports
collations	Supported collations for alphabetical comparison of strings that you can pass to queries as `ORDER BY … COLLATE <'collation_name'>`

Example

View supported aggregate functions:

```sql
SELECT name FROM system.functions WHERE is_aggregate = 1;
```

Configure user settings

Elena Dvoryadkina

Contents

To Table of Contents

Configure settings at the user account level

Configure settings at the session level

Configure settings for a single query

Default values of parameters

Examples

In ADQM/ClickHouse, there are two main groups of settings:

- **Server-level settings** — parameters that are specified in the *config.xml* configuration file and applied globally for the ClickHouse server. These settings cannot be changed at the user, session, or query level. You can find the list of these parameters and their values in the `system.server_settings` system table.
- **User-level settings** — parameters that can be configured for a particular user, session, or individual query, depending on a use case and how long the specified values should remain relevant. Available parameters with their values are listed in the `system.settings` table. The ways to configure these settings are described below.

Configure settings at the user account level

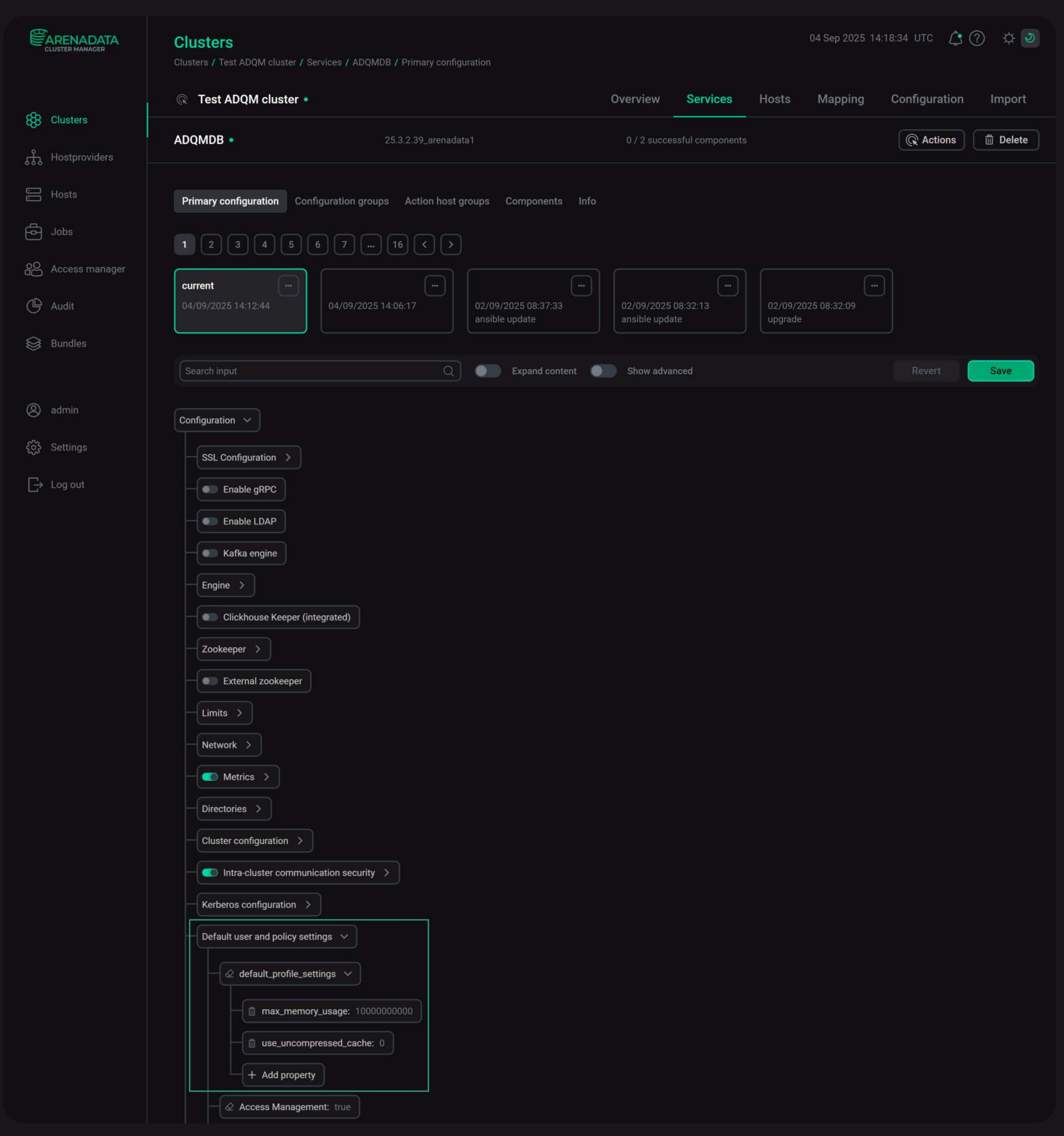
To specify values of parameters that will be applied when a user logs in, use the `SETTINGS` clause in the `CREATE USER` query when creating a new user or `ALTER USER` when modifying an existing user:

```
CREATE USER <user_name> ... SETTINGS <parameter_name> = <value>, ...;
```

You can also specify parameter values in a settings profile and then assign it to a user by running the `CREATE USER` or `ALTER USER` query:

```
CREATE SETTINGS PROFILE <profile_name> SETTINGS <parameter_name> = <value>, ...;
CREATE USER <user_name> SETTINGS PROFILE <profile_name>;
```

The settings profile of the default ADQM user can be managed through the ADCM interface — use the ***default_profile_settings*** parameter in the ***Default user and policy settings*** section on the configuration page of the ADQMDB service to edit the list of parameters for the `default` user.



ARENADATA
CLUSTER MANAGER
Clusters
04 Sep 2025 14:18:34 UTC
Clusters / Test ADQM cluster / Services / ADQMDB / Primary configuration
Test ADQM cluster
Overview
Services
Hosts
Mapping
Configuration
Import
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
ADQMDB
25.3.2.39_arenadata1
0 / 2 successful components
Actions
Delete
Primary configuration
Configuration groups
Action host groups
Components
Info
1 2 3 4 5 6 7 ... 16
current
04/09/2025 14:12:44
04/09/2025 14:06:17
02/09/2025 08:37:33
ansible update
02/09/2025 08:32:13
ansible update
02/09/2025 08:32:09
upgrade
Search input
Expand content
Show advanced
Revert
Save
Configuration
SSL Configuration
Enable gRPC
Enable LDAP
Kafka engine
Engine
Clickhouse Keeper (integrated)
Zookeeper
External zookeeper
Limits
Network
Metrics
Directories
Cluster configuration
Intra-cluster communication security
Kerberos configuration
Default user and policy settings
default_profile_settings
Add property

max_memory_usage: 10000000000

use_uncompressed_cache: 0

Access Management: true

Manage settings for the default ADQM user via ADCM

After you have set necessary parameters, do not forget to click ***Save*** and execute the ***Reconfig*** action for the ADQMDB service to save configuration changes and restart the service.

Configure settings at the session level

Use the `SET` operator to set a parameter value for the current session (this value will temporarily replace the value set at the user account level):

```
SET <parameter_name> = <value>;
```

The `SET` query also allows applying all values from the specified settings profile:

```
SET profile = '<profile_name>';
```

Session-level settings can be useful for ad-hoc testing or when you want certain parameter values to be in effect for queries within a single session, but not longer.

Configure settings for a single query

To configure settings for an individual query, list required parameters with their values after the `SETTINGS` clause:

```
SELECT <column_list> FROM <table_name> SETTINGS <parameter_name> = <value>, ...;
```

The specified settings will be applied only during this query execution — after it is completed, the parameters will use the values specified at the user/session level or the default values.

Default values of parameters

To view which parameters have been changed for a user/session and are different from the default settings, run the query:

```
SELECT name, value FROM system.settings WHERE changed;
```

To reset a parameter to its default value, you can use the command:

```
SET <parameter_name> = DEFAULT;
```

Examples

Log in to the clickhouse-client console client as the `default` user. See the `max_columns_to_read` parameter value:

```
SELECT name, value FROM system.settings WHERE name = 'max_columns_to_read';
```

```
   ┌─name────────────────┬─value─┐
1. │ max_columns_to_read │ 0     │
   └─────────────────────┴───────┘
```

Create a new user with a custom value for the `max_columns_to_read` parameter specified via the `SETTINGS` clause, for example:

```
CREATE USER test_user IDENTIFIED WITH sha256_password BY 'qwerty123' SETTINGS max_columns_to_read = 20;
```

Log in to `clickhouse-client` as the `test_user` user and run the following command:

```
SELECT name, value FROM system.settings WHERE changed;
```

The `max_columns_to_read` parameter appears in the list of modified settings with the value set for the current user:

```
    ┌─name──────────────────────────────┬─value───────┐
 1. │ max_columns_to_read               │ 20          │
      ...
    └───────────────────────────────────┴─────────────┘
```

Change the parameter value at the session level:

```
SET max_columns_to_read = 25;
```

Check the parameter value again:

```
SELECT name, value FROM system.settings WHERE changed;
```

```
    ┌─name──────────────────────────────┬─value───────┐
 1. │ max_columns_to_read               │ 25          │
      ...
    └───────────────────────────────────┴─────────────┘
```

If you log out of the console client (end the session) and log in back as `test_user`, you can make sure that the `max_columns_to_read` parameter is set to `20` again (the value assigned at the user account level).

Named collections

Elena Dvoryadkina

Contents

To Table of Contents

- Manage named collections
- Override parameters
- Create named collections
  - DDL queries
  - XML files
- Use named connections
- Modify and delete named collections
  - Modify DDL named collections
  - Drop DDL named collections

Named collections in ADQM/ClickHouse are predefined sets of parameters (specified as key/value pairs) that you can reuse in SQL queries when connecting to external data sources (such as PostgreSQL, Kafka, S3, MySQL, and others) via table functions, dictionaries, integration table and database engines.

This functionality allows you to:

- simplify SQL queries and make them easier to understand and manage by avoiding re-definition of the same parameters — you can specify values of parameters once within a named collection and then reference that collection in different queries;
- manage parameter sets centrally — a configuration for integration with an external system is stored in a single location where you can, for example, analyze connection errors if they occur and update the necessary parameters;
- protect sensitive credentials (for example, passwords) for integration with external systems by hiding them from users without administrator access rights.

> **NOTE**
>
> This article describes how to create and apply named collections using the example of integrating ADQM and ADPG. To run queries provided in examples below, first prepare ADPG as described in the Integration between ADQM and ADPG tutorial: create a PostgreSQL user for connecting to ADPG from ADQM and two PostgreSQL databases with test tables.
>
> You can also see an example of using a named collection to store parameters for connecting to an S3 storage in the Integration between ADQM and S3 article.

Manage named collections

To manage named collections (create/modify/delete), you can use:

- DDL queries. To be allowed to configure named collections with DDL, a user should have the `named_collection_control` privilege. To grant this permission, set the `named_collection_control` parameter to `1` in user settings within the */etc/clickhouse-server/users.xml* file (or another configuration file located in the */etc/clickhouse-server/users.d/* directory).

  Example

  Assign the `named_collection_control` privilege to the `default` user in the *users.xml* file:

  ```
  <users>
      <default>
          ...
          <named_collection_control>1</named_collection_control>
          ...
      </default>
  </users>
  ```

- XML files. Named collections can also be defined in the */etc/clickhouse-server/config.xml* file or other XML files located in the */etc/clickhouse-server/config.d/* directory. A named collection created or modified in a configuration file can be applied after the ClickHouse server restart.

You can view a list of existing named collections (both created via DDL queries and defined in XML files) in the `named_collections` system table.

Override parameters

Parameters set in a named collection can be overridden in an SQL query that uses this collection. For each parameter in a named collection, you can restrict the ability to change its value using the special flag:

- `OVERRIDABLE` (or `overridable="true"` in XML configuration) — parameter value can be overridden;
- `NOT OVERRIDABLE` (or `overridable="false"` in XML configuration) — parameter value cannot be overridden;
- no flag is set — the ability to override the parameter value is defined by the allow_named_collection_override_by_default setting.

> **CAUTION**
>
> If you use named collections to hide credentials for connecting to external systems from users without administrative access, it is recommended to limit the ability to override the corresponding parameters.

Create named collections

DDL queries

You can create a named collection using the `CREATE NAMED COLLECTION` query:

```
CREATE NAMED COLLECTION [IF NOT EXISTS] <collection_name> [ON CLUSTER <cluster_name>] AS
key1 = 'value1' [[NOT] OVERRIDABLE],
key2 = 'value2' [[NOT] OVERRIDABLE],
...;
```

The keys in a named collection should match the parameter names of the corresponding function, dictionary data source, table or database engine. You can see the exact names of the parameters in the ClickHouse documentation — for example, in the articles postgresql, mysql, s3, remote for the corresponding table functions or PostgreSQL, MySQL, Kafka for table engines.

When referencing a named collection that contains an invalid key, clickhouse-client displays an error message with a list of valid keys (required and optional). For example, if the `user_password` key is specified in a named collection and the `postgresql` function is called with a link to this collection, `clickhouse-client` reports that the function does not support this key:

```
DB::Exception: Unexpected key user_password in named collection. Required keys: database, db, password, table, user, username,
optional keys: addresses_expr, host, hostname, on_conflict, port, schema, use_table_cache. (BAD_ARGUMENTS)
```

Example

The following query creates the `my_adpg` named collection with parameters for connecting ADQM to ADPG:

```
CREATE NAMED COLLECTION my_adpg AS
host = '<adpg_host>',
port = 5432,
database = 'adpg_db1' NOT OVERRIDABLE,
user = 'adqm_user',
password = '<adqm_user_secret_password>';
```

Values of the parameters in this example (the `adpg_db1` database and `adqm_user` PostgreSQL user) are specified according to test data from the Integration between ADQM and ADPG article — see the ***Prepare ADPG for test examples*** section.

XML files

To define named collections in the XML configuration of the ClickHouse server, use the `named_collections` section:

```
<clickhouse>
    <named_collections>
        <named_collection1>
            <key1 overridable="true">value1</key1>
            <key2 overridable="false">value2</key2>
            <key3>value3</key3>
            ...
        </named_collection1>
        <named_collection2>
            ...
        </named_collection2>
        ...
    </named_collections>
</clickhouse>
```

Example

The following configuration defines the `my_adpg_xml` named collection with parameters for connection to the `adpg_db2` ADPG database (also see Integration between ADQM and ADPG for details on this database creation):

```
<clickhouse>
    <named_collections>
        <my_adpg_xml>
            <host>adpg_host</host>
            <port>5432</port>
            <database>adpg_db2</database>
            <user>adqm_user</user>
            <password>adqm_user_secret_password</password>
        </my_adpg_xml>
    </named_collections>
</clickhouse>
```

Use named connections

After you have created a named collection, you can use it in SQL queries to reference the predefined parameters when connecting to an external data source through a table function, integration table engine, database engine, or dictionary.

The following examples demonstrate different ways how you can access PostgreSQL data from ADQM using the named collections with ADPG connection parameters created above.

- **The postgresql table function**
  Select data from the `adpg_db1.table1` ADPG table using the postgresql table function (a table has not been defined in the named collection, so it should be specified separately in the table function parameters):

  ```
  SELECT id, name FROM postgresql(my_adpg, table = 'table1');
  ```

  Write data to the ADPG table:

  ```
  INSERT INTO TABLE FUNCTION postgresql(my_adpg, table = 'table1') (id, name) VALUES (3, 'three');
  ```

- **The PostgreSQL table engine**
  Create an ADQM table based on the PostgreSQL table engine:

  ```
  CREATE TABLE psql_table (id Int32, name String) ENGINE = PostgreSQL(my_adpg, table = 'table1');
  ```

  The engine copies data from the named collection when creating a table. Subsequent changes to the named collection do not affect existing tables.

- **The PostgreSQL database engine**
  Create a database in ADQM based on the PostgreSQL database engine (this database automatically gets tables from the `adpg_db1` PostgreSQL database — in this example, there is only one `table1`):

  ```
  CREATE DATABASE psql_database ENGINE = PostgreSQL(my_adpg);
  ```

- **The MaterializedPostgreSQL table engine**
  Create a MaterializedPostgreSQL table to replicate data of the `adpg_db2.table1` ADPG table using the my_adpg_xml named collection that lists connection parameters:

  ```
  SET allow_experimental_materialized_postgresql_table=1;
  ```

  ```
  CREATE TABLE psql_table_replica (id Int32, name String) ENGINE = MaterializedPostgreSQL(my_adpg_xml, table = 'table1');
  ```

- **The MaterializedPostgreSQL database engine**
  Create a MaterializedPostgreSQL database to which the `table1` and `table2` tables from the `adpg_db2` ADPG database are replicated, using the `my_adpg_xml` named collection to pass connection parameters for this replication:

  ```
  SET allow_experimental_database_materialized_postgresql=1;
  ```

  ```
  CREATE DATABASE psql_db_replica
  ENGINE = MaterializedPostgreSQL(my_adpg_xml)
  SETTINGS materialized_postgresql_tables_list = 'table1,table2';
  ```

- **The dictionary data source of the POSTGRESQL type**
  Create a dictionary in ADQM that reads data from the `adpg_db2.roles_dict` ADPG table (use the `name` parameter in the dictionary data source configuration to specify the named collection with settings for connecting to ADPG):

  ```
  CREATE DICTIONARY roles_dict_psql (id Int32, role String DEFAULT 'no role assigned')
  PRIMARY KEY id
  SOURCE(POSTGRESQL(name my_adpg_xml table roles_dict))
  LIFETIME(MIN 10 MAX 20)
  LAYOUT(FLAT());
  ```

Modify and delete named collections

Named collections created through DDL queries can also be altered or dropped with DDL. To modify or delete named collections defined in XML files, edit or remove their corresponding XML descriptions.

Modify DDL named collections

You can change an existing named collection with the `ALTER NAMED COLLECTION` query, using the `SET` keyword to add/change parameters and the `DELETE` keyword to delete existing ones. The basic syntax of a query is:

```
ALTER NAMED COLLECTION [IF EXISTS] <collection_name> [ON CLUSTER <cluster_name>]
[SET key1 = 'value1' [[NOT] OVERRIDABLE], key2 = 'value2' [[NOT] OVERRIDABLE], ... ] |
[DELETE key3, key4, ... ];
```

Below are examples of using the `ALTER NAMED COLLECTION` query to modify the my_adpg named collection.

- **Add/change parameters**
  Change the `database` parameter value and add the `table` parameter (this does not change the `NOT OVERRIDABLE` flag initially set for the `database` parameter, i.e. the database still cannot be overridden in an SQL query):

  ```
  ALTER NAMED COLLECTION my_adpg SET database = 'adpg_db2', table = 'table1';
  ```

  Change the `database` parameter value and allow overriding it in SQL:

  ```
  ALTER NAMED COLLECTION my_adpg SET database = 'adpg_db1' OVERRIDABLE;
  ```

- **Remove parameters**
  Remove the `table` parameter from the `my_adpg` collection:

  ```
  ALTER NAMED COLLECTION my_adpg DELETE table;
  ```

  If you need to reconfigure a parameter so that it uses the default setting (`allow_named_collection_override_by_default`) for the flag that allows/restricts overriding the parameter value, remove this parameter from the named collection and re-add it with no flag:

  ```
  ALTER NAMED COLLECTION my_adpg DELETE database;
  ```

  ```
  ALTER NAMED COLLECTION my_adpg SET database = 'adpg_db1';
  ```

  Change the `password` parameter value and remove the `database` parameter with a single query:

  ```
  ALTER NAMED COLLECTION my_adpg SET password = '<new_password>' DELETE database;
  ```

Drop DDL named collections

To remove a named collection created via a DDL query, use the `DROP NAMED COLLECTION` query:

```
DROP NAMED COLLECTION <collection_name>;
```

ADQM monitoring metrics

Elena Dvoryadkina

Contents

To Table of Contents

Metric types
View metrics

This article describes metrics for monitoring an ADQM cluster. For information on how to install monitoring, refer to the sections:

- Online installation
- Offline installation

Metric types

Two groups of metrics are available for an ADQM cluster: system metrics and ClickHouse server metrics. If monitoring is installed via the Monitoring service (available in ADQM starting with version 23.8.2.7), metrics are collected from all services of ADQM, including ZooKeeper, ClickHouse Keeper, Chproxy.

System metrics indicate general characteristics of cluster hosts, usually related to resource consumption. Available system metrics are listed in the table below.

System metrics

Metrics group	Description
cpu	CPU utilization
diskspace	Disk capacity
files	File statistics
iostat	Input/output operation performance
loadavg	System load averages
memory	Memory usage
netstat	Network connection statistics
network	Network interface performance

ClickHouse server metrics available for an ADQM cluster include:

- Metrics — metrics that are calculated instantly and have up-to-date current values (for example, the number of simultaneously processed queries or the current replica delay value).
- ProfileEvents — information about the number of events that have occurred in the system (for example, the number of `SELECT` queries processed since the start of the ClickHouse server).
- AsynchronousMetrics — metrics that are periodically calculated in the background (for example, the amount of RAM in use).

Tables below list ClickHouse server metrics.

Metrics

Metric name	Description
ActiveAsyncDrainedConnections	Number of active connections drained asynchronously
ActiveSyncDrainedConnections	Number of active connections drained synchronously
AsyncDrainedConnections	Number of connections drained asynchronously
AsynchronousReadWait	Number of threads waiting for asynchronous read
BackgroundBufferFlushSchedulePoolTask	Number of active tasks in BackgroundBufferFlushSchedulePool (this pool is used for periodic Buffer flushes)
BackgroundCommonPoolTask	Number of active tasks in an associated background pool
BackgroundDistributedSchedulePoolTask	Number of active tasks in BackgroundDistributedSchedulePool (this pool is used for distributed sends that are done in background)
BackgroundFetchesPoolTask	Number of active fetches in an associated background pool
BackgroundMergesAndMutationsPoolTask	Number of active merges and mutations in an associated background pool
BackgroundMessageBrokerSchedulePoolTask	Number of active tasks in BackgroundProcessingPool for message streaming
BackgroundMovePoolTask	Number of active tasks in BackgroundProcessingPool for moves
BackgroundSchedulePoolTask	Number of active tasks in BackgroundSchedulePool (this pool is used for periodic ReplicatedMergeTree tasks, like cleaning old data parts, altering data parts, replica re-initialization, etc.)
BrokenDistributedFilesToInsert	Number of files for asynchronous insertion into Distributed tables that has been marked as broken. This metric starts from `0` on start. Number of files for each shard is summed
CacheDetachedFileSegments	Number of existing detached cache file segments
CacheDictionaryUpdateQueueBatches	Number of "batches" (a set of keys) in update queue in CacheDictionaries
CacheDictionaryUpdateQueueKeys	Exact number of keys in update queue in CacheDictionaries
CacheFileSegments	Number of existing cache file segments
ContextLockWait	Number of threads waiting for lock in Context (this is global lock)
DelayedInserts	Number of `INSERT` queries that are throttled due to high number of active data parts for partition in a MergeTree table
DictCacheRequests	Number of requests in fly to data sources of dictionaries of cache type
DiskSpaceReservedForMerge	Disk space reserved for currently running background merges. It is slightly more than the total size of currently merging parts
DistributedFilesToInsert	Number of pending files to process for asynchronous insertion into Distributed tables. Number of files for each shard is summed
DistributedSend	Number of connections to remote servers sending data that was INSERTed into Distributed tables. Both synchronous and asynchronous mode
EphemeralNode	Number of ephemeral nodes hold in ZooKeeper
FilesystemCacheElements	Filesystem cache elements (file segments)
FilesystemCacheReadBuffers	Number of active cache buffers
FilesystemCacheSize	Filesystem cache size in bytes
GlobalThread	Number of threads in the global thread pool
GlobalThreadActive	Number of threads in the global thread pool running a task
HTTPConnection	Number of connections to an HTTP server
InterserverConnection	Number of connections from other replicas to fetch data parts
KafkaAssignedPartitions	Number of partitions Kafka tables currently assigned to
KafkaBackgroundReads	Number of background reads currently working (populating materialized views from Kafka)
KafkaConsumers	Number of active Kafka consumers
KafkaConsumersInUse	Number of consumers which are currently used by direct or background reads
KafkaConsumersWithAssignment	Number of active Kafka consumers which have some partitions assigned
KafkaLibrdkafkaThreads	Number of active librdkafka threads
KafkaProducers	Number of active Kafka producer created
KafkaWrites	Number of currently running inserts to Kafka
KeeperAliveConnections	Number of alive connections
KeeperOutstandingRequests	Number of outstanding requests
LocalThread	Number of threads in local thread pools. Threads in local thread pools are taken from the global thread pool
LocalThreadActive	Number of threads in local thread pools running a task
MMappedFileBytes	Sum size of mmapped file regions
MMappedFiles	Total number of mmapped files
MaxDDLEntryID	Max processed DDL entry of DDLWorker
MaxPushedDDLEntryID	Max DDL entry of DDLWorker pushed to ZooKeeper
MemoryTracking	Total amount of memory (in bytes) allocated by the server
Merge	Number of executing background merges
MySQLConnection	Number of client connections using the MySQL protocol
NetworkReceive	Number of threads receiving data from network. Only ClickHouse-related network interaction is included, not by third-party libraries
NetworkSend	Number of threads sending data to network. Only ClickHouse-related network interaction is included, not by third-party libraries
OpenFileForRead	Number of files open for reading
OpenFileForWrite	Number of files open for writing
PartMutation	Number of mutations (`ALTER DELETE`/`UPDATE`)
PartsActive	Active data parts (used by current and upcoming SELECTs)
PartsCommitted	Deprecated. See PartsActive
PartsCompact	Compact parts
PartsDeleteOnDestroy	Parts that were moved to another disk and should be deleted in own destructors
PartsDeleting	Not active data parts with identity refcounters, they are deleting right now by a cleaner
PartsInMemory	In-memory parts
PartsOutdated	Not active data parts, but could be used by only current SELECTs, could be deleted after SELECTs finish
PartsPreActive	Data part are in data_parts, but not used for SELECTs
PartsPreCommitted	Deprecated. See PartsPreActive
PartsTemporary	Data part are generating now, they are not in the data_parts list
PartsWide	Wide parts
PendingAsyncInsert	Number of asynchronous inserts that are waiting for flush
PostgreSQLConnection	Number of client connections using the PostgreSQL protocol
Query	Number of executing queries
QueryPreempted	Number of queries that are stopped and waiting due to the `priority` setting
QueryThread	Number of query processing threads
RWLockActiveReaders	Number of threads holding read lock in a table RWLock
RWLockActiveWriters	Number of threads holding write lock in a table RWLock
RWLockWaitingReaders	Number of threads waiting for read on a table RWLock
RWLockWaitingWriters	Number of threads waiting for write on a table RWLock
Read	Number of read (read, pread, io_getevents, etc.) syscalls in fly
ReadonlyReplica	Number of Replicated tables that are currently in the readonly state due to re-initialization after ZooKeeper session loss or due to startup without ZooKeeper configured
ReplicatedChecks	Number of data parts checking for consistency
ReplicatedFetch	Number of data parts being fetched from replica
ReplicatedSend	Number of data parts being sent to replicas
Revision	Revision of the server. It is a number incremented for every release or release candidate except patch releases
S3Requests	S3 requests
SendExternalTables	Number of connections that are sending data for external tables to remote servers. External tables are used to implement the `GLOBAL IN` and `GLOBAL JOIN` operators with distributed subqueries
SendScalars	Number of connections that are sending data for scalars to remote servers
StorageBufferBytes	Number of bytes in buffers of Buffer tables
StorageBufferRows	Number of rows in buffers of Buffer tables
SyncDrainedConnections	Number of connections drained synchronously
TCPConnection	Number of connections to a TCP server (clients with native interface), also includes server-server distributed query connections
TablesToDropQueueSize	Number of dropped tables, that are waiting for background data removal
VersionInteger	Version of the server as a single integer number in base-1000. For example, version `11.22.33` is translated to `11022033`
Write	Number of write (write, pwrite, io_getevents, etc.) syscalls in fly
ZooKeeperRequest	Number of requests to ZooKeeper in fly
ZooKeeperSession	Number of sessions (connections) to ZooKeeper. Should be no more than one, because using more than one connection to ZooKeeper may lead to bugs due to lack of linearizability (stale reads) that ZooKeeper consistency model allows
ZooKeeperWatch	Number of watches (event subscriptions) in ZooKeeper

ProfileEvents

Event	Description
AIORead	Number of reads with the Linux or FreeBSD AIO interface
AIOReadBytes	Number of bytes read with the Linux or FreeBSD AIO interface
AIOWrite	Number of writes with the Linux or FreeBSD AIO interface
AIOWriteBytes	Number of bytes written with the Linux or FreeBSD AIO interface
AggregationHashTablesInitializedAsTwoLevel	How many hash tables were inited as two-level for aggregation
AggregationPreallocatedElementsInHashTables	How many elements were preallocated in hash tables for aggregation
ArenaAllocBytes	Total bytes allocated in the internal arena used for small objects
ArenaAllocChunks	Total number of memory chunks allocated in the arena used for small objects
AsyncInsertBytes	Data size in bytes of asynchronous `INSERT` queries
AsyncInsertQuery	Same as InsertQuery, but only for asynchronous `INSERT` queries
AsynchronousReadWaitMicroseconds	Time spent in waiting for asynchronous reads
CachedReadBufferCacheWriteBytes	Bytes written from source (remote fs, etc.) to filesystem cache
CachedReadBufferCacheWriteMicroseconds	Time spent writing data into filesystem cache
CachedReadBufferReadFromCacheBytes	Bytes read from filesystem cache
CachedReadBufferReadFromCacheMicroseconds	Time spent reading from filesystem cache
CachedReadBufferReadFromSourceBytes	Bytes read from filesystem cache source (from remote fs, etc.)
CachedReadBufferReadFromSourceMicroseconds	Time spent reading from filesystem cache source (from remote filesystem, etc.)
CachedWriteBufferCacheWriteBytes	Bytes written from source (remote fs, etc.) to filesystem cache
CachedWriteBufferCacheWriteMicroseconds	Time spent writing data into filesystem cache
CannotRemoveEphemeralNode	Number of times an error happened while trying to remove ephemeral node
CannotWriteToWriteBufferDiscard	Number of stack traces dropped by a query profiler or signal handler because pipe is full or cannot write to pipe
CompileExpressionsBytes	Number of bytes used for compilation of expressions
CompileExpressionsMicroseconds	Total time spent for compilation of expressions to LLVM code
CompileFunction	Number of times a compilation of generated LLVM code (to create fused function for complex expressions) was initiated
CompiledFunctionExecute	Number of times a compiled function was executed
CompressedReadBufferBlocks	Number of compressed blocks (blocks of data that are compressed independent of each other) read from compressed sources (files, network)
CompressedReadBufferBytes	Number of uncompressed bytes (the number of bytes after decompression) read from compressed sources (files, network)
ContextLock	Number of times the lock of Context (global lock) was acquired or tried to acquire
CreatedHTTPConnections	Total amount of created HTTP connections (the counter increases each time a connection is created)
CreatedLogEntryForMerge	Number of successfully created log entries to merge parts in ReplicatedMergeTree tables
CreatedLogEntryForMutation	Number of successfully created log entries to mutate parts in ReplicatedMergeTree tables
CreatedReadBufferDirectIO	Number of times a read buffer with `O_DIRECT` was created for reading data (while choosing among other read methods)
CreatedReadBufferDirectIOFailed	Number of times a read buffer with `O_DIRECT` was attempted to be created for reading data (while choosing among other read methods), but the OS did not allow it (due to lack of filesystem support or other reasons) and we fallen back to the ordinary reading method
CreatedReadBufferMMap	Number of times a read buffer using `mmap` was created for reading data (while choosing among other read methods)
CreatedReadBufferMMapFailed	Number of times a read buffer with `mmap` was attempted to be created for reading data (while choosing among other read methods), but the OS did not allow it (due to lack of filesystem support or other reasons) and we fallen back to the ordinary reading method
CreatedReadBufferOrdinary	Number of times a read buffer for ordinary files was created
DNSError	Total count of errors in DNS resolution
DataAfterMergeDiffersFromReplica	Number of times data after merge is not byte-identical to the data on another replicas. There could be several reasons: - using newer version of a compression library after server update; - using another compression method; - non-deterministic compression algorithm (highly unlikely); - non-deterministic merge algorithm due to a logical error in code; - data corruption in memory due to a bug in code; - data corruption in memory due to a hardware issue; - manual modification of source data after server startup; - manual modification of checksums stored in ZooKeeper; - settings related to a data part format (for example, `enable_mixed_granularity_parts`) are different on different replicas. The server detects this situation and downloads a merged part from replica to force byte-identical result
DataAfterMutationDiffersFromReplica	Number of times data after mutation is not byte-identical to the data on another replicas. In addition to the reasons described in DataAfterMergeDiffersFromReplica, it is also possible due to non-deterministic mutation
DelayedInserts	Number of times the `INSERT` of a block to a MergeTree table was throttled due to the high number of active data parts for a partition
DelayedInsertsMilliseconds	Total number of milliseconds spent while the `INSERT` of a block to a MergeTree table was throttled due to the high number of active data parts for a partition
DictCacheKeysExpired	Number of keys looked up in the dictionaries of cache types and found in the cache but they were obsolete
DictCacheKeysHit	Number of keys looked up in the dictionaries of cache types and found in the cache
DictCacheKeysNotFound	Number of keys looked up in the dictionaries of cache types and not found
DictCacheKeysRequested	Number of keys requested from the data source for the dictionaries of cache types
DictCacheKeysRequestedFound	Number of keys requested from the data source for dictionaries of cache types and found in the data source
DictCacheKeysRequestedMiss	Number of keys requested from the data source for dictionaries of cache types but not found in the data source
DictCacheLockReadNs	Number of nanoseconds spend in waiting for read lock to lookup the data for the dictionaries of cache types
DictCacheLockWriteNs	Number of nanoseconds spend in waiting for write lock to update the data for the dictionaries of cache types
DictCacheRequestTimeNs	Number of nanoseconds spend in querying the external data sources for the dictionaries of cache types
DictCacheRequests	Number of bulk requests to the external data sources for the dictionaries of cache types
DirectorySync	Number of times the `F_FULLFSYNC`/`fsync`/`fdatasync` function was called for directories
DirectorySyncElapsedMicroseconds	Total time spent waiting for `F_FULLFSYNC`/`fsync`/`fdatasync` syscall for directories
DiskReadElapsedMicroseconds	Total time spent waiting for read syscall. This includes reads from page cache
DiskWriteElapsedMicroseconds	Total time spent waiting for write syscall. This includes writes to page cache
DistributedConnectionFailAtAll	Total count when distributed connection fails after all retries finished
DistributedConnectionFailTry	Total count when distributed connection fails with retry
DistributedConnectionMissingTable	Number of times a replica was rejected from a distributed query, because it did not contain a table needed for the query
DistributedConnectionStaleReplica	Number of times a replica was rejected from a distributed query, because some table needed for a query had replication lag higher than the configured threshold
DistributedDelayedInserts	Number of times the `INSERT` of a block to a Distributed table was throttled due to the high number of pending bytes
DistributedDelayedInsertsMilliseconds	Total number of milliseconds spent while the `INSERT` of a block to a Distributed table was throttled due to the high number of pending bytes
DistributedRejectedInserts	Number of times the `INSERT` of a block to a Distributed table was rejected with the `Too many bytes` exception due to the high number of pending bytes
DistributedSyncInsertionTimeoutExceeded	A timeout has exceeded while waiting for shards during synchronous insertion into a Distributed table (with `insert_distributed_sync = 1`)
DuplicatedInsertedBlocks	Number of times a block INSERTed to a ReplicatedMergeTree table was deduplicated
ExecuteShellCommand	Number of shell command executions
ExternalAggregationCompressedBytes	Number of bytes written to a disk for aggregation in external memory
ExternalAggregationMerge	Number of times temporary files were merged for aggregation in external memory
ExternalAggregationUncompressedBytes	Amount of data (uncompressed, before compression) written to a disk for aggregation in external memory
ExternalAggregationWritePart	Number of times a temporary file was written to a disk for aggregation in external memory
ExternalDataSourceLocalCacheReadBytes	Bytes read from local cache buffer in RemoteReadBufferCache
ExternalSortMerge	Number of times temporary files were merged for sorting in external memory
ExternalSortWritePart	Number of times a temporary file was written to a disk for sorting in external memory
FailedInsertQuery	Same as FailedQuery, but only for `INSERT` queries
FailedQuery	Number of failed queries
FailedSelectQuery	Same as FailedQuery, but only for `SELECT` queries
FileOpen	Number of files opened
FileSegmentCacheWriteMicroseconds	Metric per file segment. Time spend writing data to the cache
FileSegmentPredownloadMicroseconds	Metric per file segment. Time spent predownloading data to the cache. Predownloading — finishing file segment download (after someone who failed to do that) up to the point current thread was requested to do
FileSegmentReadMicroseconds	Metric per file segment. Time spend reading from a file
FileSegmentUsedBytes	Metric per file segment. How many bytes were actually used from the current file segment
FileSegmentWaitReadBufferMicroseconds	Metric per file segment. Time spend waiting for internal read buffer (includes cache waiting)
FileSync	Number of times the `F_FULLFSYNC`/`fsync`/`fdatasync` function was called for files
FileSyncElapsedMicroseconds	Total time spent waiting for `F_FULLFSYNC`/`fsync`/`fdatasync` syscall for files
FunctionExecute	Number of times a function was executed
HardPageFaults	Number of hard page faults. These are the page faults that required IO activity to handle
HedgedRequestsChangeReplica	Total count when timeout for changing replica expired in hedged requests
IOBufferAllocBytes	Total number of bytes allocated in IO buffers
IOBufferAllocs	Total number of allocations made in IO buffers
InsertQuery	Same as Query, but only for `INSERT` queries
InsertQueryTimeMicroseconds	Total time of `INSERT` queries
InsertedBytes	Number of bytes (uncompressed; for columns as they stored in memory) INSERTed to all tables
InsertedCompactParts	Number of parts inserted in the Compact format
InsertedInMemoryParts	Number of parts inserted in the InMemory format
InsertedRows	Number of rows INSERTed to all tables
InsertedWideParts	Number of parts inserted in the Wide format
KafkaBackgroundReads	Number of background reads populating materialized views from Kafka since server start
KafkaCommitFailures	Number of failed commits of consumed offsets to Kafka (usually is a sign of some data duplication)
KafkaCommits	Number of successful commits of consumed offsets to Kafka (normally should be the same as KafkaBackgroundReads)
KafkaConsumerErrors	Number of errors reported by librdkafka during polls
KafkaDirectReads	Number of direct selects from Kafka tables since server start
KafkaMessagesFailed	Number of Kafka messages ClickHouse failed to parse
KafkaMessagesPolled	Number of Kafka messages polled from librdkafka to ClickHouse
KafkaMessagesProduced	Number of messages produced to Kafka
KafkaMessagesRead	Number of Kafka messages already processed by ClickHouse
KafkaProducerErrors	Number of errors during producing the messages to Kafka
KafkaProducerFlushes	Number of explicit flushes to Kafka producer
KafkaRebalanceAssignments	Number of partition assignments (the final stage of consumer group rebalance)
KafkaRebalanceErrors	Number of failed consumer group rebalances
KafkaRebalanceRevocations	Number of partition revocations (the first stage of consumer group rebalance)
KafkaRowsRead	Number of rows parsed from Kafka messages
KafkaRowsRejected	Number of parsed rows which were later rejected (due to rebalances/errors or similar reasons). Those rows will be consumed again after the rebalance
KafkaRowsWritten	Number of rows inserted into Kafka tables
KafkaWrites	Number of writes (inserts) to Kafka tables
KeeperCommits	Number of successful commits
KeeperCommitsFailed	Number of failed commits
KeeperLatency	Keeper latency
KeeperPacketsReceived	Packets received by keeper server
KeeperPacketsSent	Packets sent by keeper server
KeeperReadSnapshot	Number of snapshot read (serialization)
KeeperRequestTotal	Total number of requests on keeper server
KeeperSaveSnapshot	Number of snapshot save
KeeperSnapshotApplys	Number of snapshot applying
KeeperSnapshotApplysFailed	Number of failed snapshot applying
KeeperSnapshotCreations	Number of snapshots creations
KeeperSnapshotCreationsFailed	Number of failed snapshot creations
MMappedFileCacheHits	Number of times a file has been found in the MMap cache (for the `mmap` read_method), so we didn't have to mmap it again
MMappedFileCacheMisses	Number of times a file has not been found in the MMap cache (for the `mmap` read_method), so we had to mmap it again
MainConfigLoads	Number of times the main configuration was reloaded
MarkCacheHits	Number of times an entry has been found in the mark cache, so we didn't have to load a mark file
MarkCacheMisses	Number of misses in the mark cache (this cache is used in the MergeTree engine for faster data reading)
MemoryOvercommitWaitTimeMicroseconds	Total time spent in waiting for memory to be freed in OvercommitTracker
Merge	Number of launched background merges
MergeTreeDataProjectionWriterBlocks	Number of blocks INSERTed to MergeTree tables projection. Each block forms a data part of level zero
MergeTreeDataProjectionWriterBlocksAlreadySorted	Number of blocks INSERTed to MergeTree tables projection that appeared to be already sorted
MergeTreeDataProjectionWriterCompressedBytes	Bytes written to filesystem for data INSERTed to MergeTree tables projection
MergeTreeDataProjectionWriterRows	Number of rows INSERTed to MergeTree tables projection
MergeTreeDataProjectionWriterUncompressedBytes	Uncompressed bytes (for columns as they stored in memory) INSERTed to MergeTree tables projection
MergeTreeDataWriterBlocks	Number of blocks INSERTed to MergeTree tables. Each block forms a data part of level zero
MergeTreeDataWriterBlocksAlreadySorted	Number of blocks INSERTed to MergeTree tables that appeared to be already sorted
MergeTreeDataWriterCompressedBytes	Bytes written to filesystem for data INSERTed to MergeTree tables
MergeTreeDataWriterRows	Number of rows INSERTed to MergeTree tables
MergeTreeDataWriterUncompressedBytes	Uncompressed bytes (for columns as they stored in memory) INSERTed to MergeTree tables
MergeTreeMetadataCacheDelete	Number of rocksdb deletes (used for the merge tree metadata cache)
MergeTreeMetadataCacheGet	Number of rocksdb reads (used for the merge tree metadata cache)
MergeTreeMetadataCacheHit	Number of times the read of meta file was done from the MergeTree metadata cache
MergeTreeMetadataCacheMiss	Number of times the read of meta file was not done from the MergeTree metadata cache
MergeTreeMetadataCachePut	Number of rocksdb puts (used for the merge tree metadata cache)
MergeTreeMetadataCacheSeek	Number of rocksdb seeks (used for the merge tree metadata cache)
MergedIntoCompactParts	Number of parts merged into the Compact format
MergedIntoInMemoryParts	Number of parts in merged into the InMemory format
MergedIntoWideParts	Number of parts merged into the Wide format
MergedRows	Rows read for background merges (the number of rows before merge)
MergedUncompressedBytes	Uncompressed bytes (for columns as they stored in memory) that was read for background merges. This is the number before merge
MergesTimeMilliseconds	Total time spent for background merges
NetworkReceiveBytes	Total number of bytes received from network. Only ClickHouse-related network interaction is included, not by third-party libraries
NetworkReceiveElapsedMicroseconds	Total time spent waiting for data to receive or receiving data from network. Only ClickHouse-related network interaction is included, not by third-party libraries
NetworkSendBytes	Total number of bytes send to network. Only ClickHouse-related network interaction is included, not by third-party libraries
NetworkSendElapsedMicroseconds	Total time spent waiting for data to send to network or sending data to network. Only ClickHouse-related network interaction is included, not by third-party libraries
NotCreatedLogEntryForMerge	Log entry to merge parts in ReplicatedMergeTree is not created due to concurrent log update by another replica
NotCreatedLogEntryForMutation	Log entry to mutate parts in ReplicatedMergeTree is not created due to concurrent log update by another replica
OSCPUVirtualTimeMicroseconds	CPU time spent seen by OS. Does not include involuntary waits due to virtualization
OSCPUWaitMicroseconds	Total time a thread was ready for execution but waiting to be scheduled by OS, from the OS point of view
OSIOWaitMicroseconds	Total time a thread spent waiting for a result of IO operation, from the OS point of view. This is real IO that doesn't include the page cache
OSReadBytes	Number of bytes read from disks or block devices. Doesn't include bytes read from page cache. May include excessive data due to block size, readahead, etc.
OSReadChars	Number of bytes read from filesystem, including the page cache
OSWriteBytes	Number of bytes written to disks or block devices. Doesn't include bytes that are in page cache dirty pages. May not include data that was written by OS asynchronously
OSWriteChars	Number of bytes written to filesystem, including the page cache
ObsoleteReplicatedParts	Number of times a data part was covered by another data part that has been fetched from a replica (so, we have marked a covered data part as obsolete and no longer needed)
OpenedFileCacheHits	Number of times a file has been found in the opened file cache, so we didn't have to open it again
OpenedFileCacheMisses	Number of misses in the opened file cache
OtherQueryTimeMicroseconds	Total time of queries that are not `SELECT` or `INSERT`
OverflowAny	Number of times approximate `GROUP BY` was in effect: when aggregation was performed only on top of first `max_rows_to_group_by` unique keys and other keys were ignored due to `group_by_overflow_mode = any`
OverflowBreak	Number of times when data processing was cancelled by query complexity limitation with the `overflow_mode = break` setting and the result was incomplete
OverflowThrow	Number of times when data processing was cancelled by query complexity limitation with the `overflow_mode = throw` setting and an exception was thrown
PerfAlignmentFaults	Number of alignment faults. These happen when unaligned memory accesses happen; the kernel can handle these but it reduces performance. This happens only on some architectures (never on x86)
PerfBranchInstructions	Retired branch instructions. Prior to Linux 2.6.35, this used the wrong event on AMD processors
PerfBranchMisses	Mispredicted branch instructions
PerfBusCycles	Bus cycles which can be different from total cycles
PerfCacheMisses	Cache misses. Usually this indicates Last Level Cache misses. This event is intended to be used in conjunction with the PerfCacheReferences event to calculate cache miss rates
PerfCacheReferences	Cache accesses. Usually this event indicates Last Level Cache accesses but may vary depending on your CPU. The metric may include prefetches and coherency messages, and this also depends on the design of your CPU
PerfContextSwitches	Number of context switches
PerfCpuClock	The CPU clock, a high-resolution per-CPU timer
PerfCpuCycles	Total cycles. Be wary of what happens during CPU frequency scaling
PerfCpuMigrations	Number of times the process has migrated to a new CPU
PerfDataTLBMisses	Data TLB misses
PerfDataTLBReferences	Data TLB references
PerfEmulationFaults	Number of emulation faults. The kernel sometimes traps on unimplemented instructions and emulates them for user space. This can negatively impact performance
PerfInstructionTLBMisses	Instruction TLB misses
PerfInstructionTLBReferences	Instruction TLB references
PerfInstructions	Retired instructions. Be careful, these can be affected by various issues, most notably hardware interrupt counts
PerfLocalMemoryMisses	Local NUMA node memory read misses
PerfLocalMemoryReferences	Local NUMA node memory reads
PerfMinEnabledRunningTime	Running time for event with minimum enabled time. Used to track the amount of event multiplexing
PerfMinEnabledTime	For all events, minimum time that an event was enabled. Used to track event multiplexing influence
PerfRefCpuCycles	Total cycles (not affected by CPU frequency scaling)
PerfStalledCyclesBackend	Stalled cycles during retirement
PerfStalledCyclesFrontend	Stalled cycles during issue
PerfTaskClock	A clock count specific to the task that is running
PolygonsAddedToPool	A polygon has been added to the cache (pool) for the `pointInPolygon` function
PolygonsInPoolAllocatedBytes	The number of bytes for polygons added to the cache (pool) for the `pointInPolygon` function
Query	Number of queries to be interpreted and potentially executed. Does not include queries that failed to parse or were rejected due to AST size limits, quota limits or limits on the number of simultaneously running queries. May include internal queries initiated by ClickHouse itself. Does not count subqueries
QueryMaskingRulesMatch	Number of times when query masking rules were successfully matched
QueryMemoryLimitExceeded	Number of times when memory limit exceeded for query
QueryProfilerRuns	Number of times QueryProfiler had been run
QueryProfilerSignalOverruns	Number of times processing of a query profiler signal was dropped due to overrun plus the number of signals that OS has not delivered due to overrun
QueryTimeMicroseconds	Total time of all queries
RWLockAcquiredReadLocks	Number of times a read lock was acquired on a RWLock
RWLockAcquiredWriteLocks	Number of times a write lock was acquired (in a heavy RWLock)
RWLockReadersWaitMilliseconds	Total time in milliseconds that readers waited for acquiring a lock on a RWLock
RWLockWritersWaitMilliseconds	Total time spent waiting for a write lock to be acquired (in a heavy RWLock)
ReadBackoff	Number of times the number of query processing threads was lowered due to slow reads
ReadBufferFromFileDescriptorRead	Number of reads (read/pread) from a file descriptor. Does not include sockets
ReadBufferFromFileDescriptorReadBytes	Number of bytes read from file descriptors. If the file is compressed, this will show the compressed data size
ReadBufferFromFileDescriptorReadFailed	Number of times the read (read/pread) from a file descriptor have failed
ReadBufferFromS3Bytes	Bytes read from S3
ReadBufferFromS3Microseconds	Time spent on reading from S3
ReadBufferFromS3RequestsErrors	Number of exceptions while reading from S3
ReadBufferSeekCancelConnection	Number of seeks which lead to new connection (s3, http)
ReadCompressedBytes	Number of bytes (before decompression) read from compressed sources (files, network)
RealTimeMicroseconds	Total (wall clock) time spent in processing (queries and other tasks) threads (not that this is a sum)
RegexpCreated	Compiled regular expressions. Identical regular expressions compiled just once and cached forever
RejectedInserts	Number of times the INSERT of a block to a MergeTree table was rejected with the `Too many parts` exception due to the high number of active data parts for a partition
RemoteFSBuffers	Number of buffers created for asynchronous reading from a remote filesystem
RemoteFSCacheDownloadBytes	Number of bytes downloaded from a remote filesystem cache
RemoteFSCacheReadBytes	Number of bytes read from a remote filesystem cache
RemoteFSCancelledPrefetches	Number of cancelled prefetches (because of seek)
RemoteFSLazySeeks	Number of lazy seeks
RemoteFSPrefetchedReads	Number of reads from prefetched buffer
RemoteFSPrefetches	Number of prefetches made with asynchronous reading from a remote filesystem
RemoteFSReadBytes	Number of bytes read from a remote filesystem
RemoteFSReadMicroseconds	Time spent reading from a remote filesystem
RemoteFSSeeks	Total number of seeks for async buffer
RemoteFSSeeksWithReset	Number of seeks which lead to a new connection
RemoteFSUnprefetchedReads	Number of reads from an unprefetched buffer
RemoteFSUnusedPrefetches	Number of prefetches pending at buffer destruction
ReplicaPartialShutdown	How many times a Replicated table has to deinitialize its state due to session expiration in ZooKeeper. The state is reinitialized every time when ZooKeeper is available again
ReplicatedDataLoss	Number of times a wanted data part doesn't exist on any replica (even on replicas that are offline right now). That data parts are definitely lost. This is normal due to asynchronous replication (if quorum inserts were not enabled), when the replica on which the data part was written was failed and when it became online after fail it doesn't contain that data part
ReplicatedPartChecks	Number of times we had to perform advanced search for a data part on replicas or to clarify the need of an existing data part
ReplicatedPartChecksFailed	Number of times the advanced search for a data part on replicas did not give result or when unexpected part has been found and moved away
ReplicatedPartFailedFetches	Number of times a data part was failed to download from replica of a ReplicatedMergeTree table
ReplicatedPartFetches	Number of times a data part was downloaded from a replica of a ReplicatedMergeTree table
ReplicatedPartFetchesOfMerged	Number of times we prefer to download already merged part from a replica of a ReplicatedMergeTree table instead of performing a merge ourself (usually we prefer doing a merge ourself to save network traffic). This happens when we have not all source parts to perform a merge or when the data part is old enough
ReplicatedPartMerges	Number of times data parts of ReplicatedMergeTree tables were successfully merged
ReplicatedPartMutations	Number of times data parts of ReplicatedMergeTree tables were successfully mutated
S3ReadBytes	Number of bytes read from S3 storage
S3ReadMicroseconds	Time of `GET` and `HEAD` requests to S3 storage
S3ReadRequestsCount	Number of `GET` and `HEAD` requests to S3 storage
S3ReadRequestsErrors	Number of non-throttling errors in `GET` and `HEAD` requests to S3 storage
S3ReadRequestsRedirects	Number of redirects in `GET` and `HEAD` requests to S3 storage
S3ReadRequestsThrottling	Number of 429 and 503 errors in `GET` and `HEAD` requests to S3 storage
S3WriteBytes	Number of bytes written to S3 storage
S3WriteMicroseconds	Time of `POST`, `DELETE`, `PUT`, and `PATCH` requests to S3 storage
S3WriteRequestsCount	Number of `POST`, `DELETE`, `PUT`, and `PATCH` requests to S3 storage
S3WriteRequestsErrors	Number of non-throttling errors in `POST`, `DELETE`, `PUT`, and `PATCH` requests to S3 storage
S3WriteRequestsRedirects	Number of redirects in `POST`, `DELETE`, `PUT`, and `PATCH` requests to S3 storage
S3WriteRequestsThrottling	Number of 429 and 503 errors in `POST`, `DELETE`, `PUT`, and `PATCH` requests to S3 storage
ScalarSubqueriesCacheMiss	Number of times a read from a scalar subquery was not cached and had to be calculated completely
ScalarSubqueriesGlobalCacheHit	Number of times a read from a scalar subquery was done using the global cache
ScalarSubqueriesLocalCacheHit	Number of times a read from a scalar subquery was done using the local cache
SchemaInferenceCacheEvictions	Number of times a schema from cache was evicted due to overflow
SchemaInferenceCacheHits	Number of times a schema from cache was used for schema inference
SchemaInferenceCacheInvalidations	Number of times a schema in cache became invalid due to changes in data
SchemaInferenceCacheMisses	Number of times a schema is not in cache while schema inference
Seek	Number of times the `lseek` function was called
SelectQuery	Same as Query, but only for `SELECT` queries
SelectQueryTimeMicroseconds	Total time of `SELECT` queries
SelectedBytes	Number of bytes (uncompressed; for columns as they stored in memory) SELECTed from all tables
SelectedMarks	Number of marks (index granules) selected to read from a MergeTree table
SelectedParts	Number of data parts selected to read from a MergeTree table
SelectedRanges	Number of non-adjacent ranges in all data parts selected to read from a MergeTree table
SelectedRows	Number of rows SELECTed from all tables
SleepFunctionCalls	Number of times a sleep function (`sleep`, `sleepEachRow`) has been called
SleepFunctionMicroseconds	Time spent sleeping due to a sleep function call
SlowRead	Number of reads from a file that were slow. This indicate system overload. Thresholds are controlled by `read_backoff_*` settings
SoftPageFaults	Number of soft page faults. These are the page faults that were resolved without disk IO activity
StorageBufferErrorOnFlush	Number of times a buffer in the Buffer table has not been able to flush due to error writing in the destination table
StorageBufferFlush	Number of times a buffer in a Buffer table was flushed
StorageBufferLayerLockReadersWaitMilliseconds	Time for waiting for Buffer layer during reading
StorageBufferLayerLockWritersWaitMilliseconds	Time for waiting free Buffer layer to write to (can be used to tune Buffer layers)
StorageBufferPassedAllMinThresholds	Number of times a criteria on min thresholds has been reached to flush a buffer in a Buffer table
StorageBufferPassedBytesFlushThreshold	Number of times background-only flush threshold on bytes has been reached to flush a buffer in a Buffer table (this is an expert-only metric)
StorageBufferPassedBytesMaxThreshold	Number of times a criteria on max bytes threshold has been reached to flush a buffer in a Buffer table
StorageBufferPassedRowsFlushThreshold	Number of times a background-only flush threshold on rows has been reached to flush a buffer in a Buffer table (this is expert-only metric)
StorageBufferPassedRowsMaxThreshold	Number of times a criteria on max rows threshold has been reached to flush a buffer in a Buffer table
StorageBufferPassedTimeFlushThreshold	Number of times a background-only flush threshold on time has been reached to flush a buffer in a Buffer table (this is expert-only metric)
StorageBufferPassedTimeMaxThreshold	Number of times a criteria on max time threshold has been reached to flush a buffer in a Buffer table
SystemTimeMicroseconds	Total time spent in processing (queries and other tasks) threads executing CPU instructions in OS kernel space. This includes time CPU pipeline was stalled due to cache misses, branch mispredictions, hyper-threading, etc.
TableFunctionExecute	Number of times a table function was executed
ThreadPoolReaderPageCacheHit	Number of times the read inside ThreadPoolReader was done from the page cache
ThreadPoolReaderPageCacheHitBytes	Number of bytes read inside ThreadPoolReader when it was done from the page cache
ThreadPoolReaderPageCacheHitElapsedMicroseconds	Time spent reading data from the page cache in ThreadPoolReader
ThreadPoolReaderPageCacheMiss	Number of times the read inside ThreadPoolReader was not done from the page cache and was hand off to the thread pool
ThreadPoolReaderPageCacheMissBytes	Number of bytes read inside ThreadPoolReader when read was not done from the page cache and was hand off to the thread pool
ThreadPoolReaderPageCacheMissElapsedMicroseconds	Time spent reading data inside the asynchronous job in ThreadPoolReader — when read was not done from the page cache
ThreadpoolReaderReadBytes	Bytes read from a threadpool task in asynchronous reading
ThreadpoolReaderTaskMicroseconds	Time spent getting the data in asynchronous reading
ThrottlerSleepMicroseconds	Total time a query was sleeping to conform all throttling settings
UncompressedCacheHits	Number of times a block of data has been found in the uncompressed cache (and decompression was avoided)
UncompressedCacheMisses	Number of times a block of data has not been found in the uncompressed cache (and decompression was required)
UncompressedCacheWeightLost	Number of bytes evicted from the uncompressed cache
UserTimeMicroseconds	Total time spent in processing (queries and other tasks) threads executing CPU instructions in user space. This includes time CPU pipeline was stalled due to cache misses, branch mispredictions, hyper-threading, etc.
WriteBufferFromFileDescriptorWrite	Number of writes (write/pwrite) to a file descriptor. Does not include sockets
WriteBufferFromFileDescriptorWriteBytes	Number of bytes written to file descriptors. If the file is compressed, this will show compressed data size
WriteBufferFromFileDescriptorWriteFailed	Number of times the write (write/pwrite) to a file descriptor have failed
WriteBufferFromS3Bytes	Bytes written to S3
ZooKeeperBytesReceived	Total number of bytes received from ZooKeeper
ZooKeeperBytesSent	Total number of bytes sent to ZooKeeper
ZooKeeperCheck	Number of `check` requests to ZooKeeper. Usually they don't make sense in isolation, only as part of a complex transaction
ZooKeeperClose	Number of times connection with ZooKeeper has been closed voluntary
ZooKeeperCreate	Number of times a node was created in ZooKeeper
ZooKeeperExists	Number of times an existence check was performed in ZooKeeper
ZooKeeperGet	Number of times data was fetched from ZooKeeper
ZooKeeperHardwareExceptions	Number of exceptions while working with ZooKeeper related to network (connection loss or similar)
ZooKeeperInit	Number of times a ZooKeeper session was initialized
ZooKeeperList	Number of times a `list` command was executed on ZooKeeper
ZooKeeperMulti	Number of times a `multi` command was executed on ZooKeeper
ZooKeeperOtherExceptions	Number of exceptions while working with ZooKeeper other than ZooKeeperUserExceptions and ZooKeeperHardwareExceptions
ZooKeeperRemove	Number of `remove` requests to ZooKeeper
ZooKeeperSet	Number of `set` requests to ZooKeeper
ZooKeeperSync	Number of `sync` requests to ZooKeeper. These requests are rarely needed or usable
ZooKeeperTransactions	Number of transactions performed on ZooKeeper
ZooKeeperUserExceptions	Number of exceptions while working with ZooKeeper related to the data (no node, bad version, or similar)
ZooKeeperWaitMicroseconds	Total time spent waiting for ZooKeeper in microseconds
ZooKeeperWatchResponse	Number of times watch notification has been received from ZooKeeper

AsynchronousMetrics

Metric name	Description
AsynchronousMetricsCalculationTimeSpent	Time in seconds spent for calculation of asynchronous metrics (this is the overhead of asynchronous metrics)
BlockActiveTime_vda	Time in seconds the block device had the IO requests queued. This is a system-wide metric, it includes all the processes on the host machine, not just clickhouse-server
BlockDiscardBytes_vda	Number of discarded bytes on the block device. These operations are relevant for SSD. Discard operations are not used by ClickHouse, but can be used by other processes on the system. This is a system-wide metric, it includes all the processes on the host machine, not just clickhouse-server
BlockDiscardMerges_vda	Number of discard operations requested from the block device and merged together by the OS IO scheduler. These operations are relevant for SSD. Discard operations are not used by ClickHouse, but can be used by other processes on the system. This is a system-wide metric, it includes all the processes on the host machine, not just clickhouse-server
BlockDiscardOps_vda	Number of discard operations requested from the block device. These operations are relevant for SSD. Discard operations are not used by ClickHouse, but can be used by other processes on the system. This is a system-wide metric, it includes all the processes on the host machine, not just clickhouse-server
BlockDiscardTime_vda	Time in seconds spend in discard operations requested from the block device, summed across all the operations. These operations are relevant for SSD. Discard operations are not used by ClickHouse, but can be used by other processes on the system. This is a system-wide metric, it includes all the processes on the host machine, not just clickhouse-server
BlockInFlightOps_vda	This value counts the number of I/O requests that have been issued to the device driver but have not yet completed. It does not include IO requests that are in the queue but not yet issued to the device driver. This is a system-wide metric, it includes all the processes on the host machine, not just clickhouse-server
BlockQueueTime_vda	This value counts the number of milliseconds that IO requests have waited on this block device. If there are multiple IO requests waiting, this value will increase as the product of the number of milliseconds times the number of requests waiting. This is a system-wide metric, it includes all the processes on the host machine, not just clickhouse-server
BlockReadBytes_vda	Number of bytes read from the block device. It can be lower than the number of bytes read from the filesystem due to the usage of the OS page cache, that saves IO. This is a system-wide metric, it includes all the processes on the host machine, not just clickhouse-server
BlockReadMerges_vda	Number of read operations requested from the block device and merged together by the OS IO scheduler. This is a system-wide metric, it includes all the processes on the host machine, not just clickhouse-server
BlockReadOps_vda	Number of read operations requested from the block device. This is a system-wide metric, it includes all the processes on the host machine, not just clickhouse-server
BlockReadTime_vda	Time in seconds spend in read operations requested from the block device, summed across all the operations. This is a system-wide metric, it includes all the processes on the host machine, not just clickhouse-server
BlockWriteBytes_vda	Number of bytes written to the block device. It can be lower than the number of bytes written to the filesystem due to the usage of the OS page cache, that saves IO. A write to the block device may happen later than the corresponding write to the filesystem due to write-through caching. This is a system-wide metric, it includes all the processes on the host machine, not just clickhouse-server
BlockWriteMerges_vda	Number of write operations requested from the block device and merged together by the OS IO scheduler. This is a system-wide metric, it includes all the processes on the host machine, not just clickhouse-server
BlockWriteOps_vda	Number of write operations requested from the block device. This is a system-wide metric, it includes all the processes on the host machine, not just clickhouse-server
BlockWriteTime_vda	Time in seconds spend in write operations requested from the block device, summed across all the operations. This is a system-wide metric, it includes all the processes on the host machine, not just clickhouse-server
CPUFrequencyMHz_N	The current frequency of the CPU, in MHz. Most of the modern CPUs adjust the frequency dynamically for power saving and Turbo Boosting
CompiledExpressionCacheBytes	Total bytes used for the cache of JIT-compiled code
CompiledExpressionCacheCount	Total entries in the cache of JIT-compiled code
DiskAvailable_default	Available bytes on the disk (virtual filesystem). Remote filesystems can show a large value like 16 EiB
DiskTotal_default	The total size in bytes of the disk (virtual filesystem). Remote filesystems can show a large value like 16 EiB
DiskUnreserved_default	Available bytes on the disk (virtual filesystem) without the reservations for merges, fetches, and moves. Remote filesystems can show a large value like 16 EiB
DiskUsed_default	Used bytes on the disk (virtual filesystem). Remote filesystems not always provide this information
FilesystemLogsPathAvailableBytes	Available bytes on the volume where ClickHouse logs path is mounted. If this value approaches zero, you should tune the log rotation in the configuration file
FilesystemLogsPathAvailableINodes	The number of available inodes on the volume where ClickHouse logs path is mounted
FilesystemLogsPathTotalBytes	The size of the volume where ClickHouse logs path is mounted, in bytes. It's recommended to have at least 10 GB for logs
FilesystemLogsPathTotalINodes	The total number of inodes on the volume where ClickHouse logs path is mounted
FilesystemLogsPathUsedBytes	Used bytes on the volume where ClickHouse logs path is mounted
FilesystemLogsPathUsedINodes	The number of used inodes on the volume where ClickHouse logs path is mounted
FilesystemMainPathAvailableBytes	Available bytes on the volume where the main ClickHouse path is mounted
FilesystemMainPathAvailableINodes	The number of available inodes on the volume where the main ClickHouse path is mounted. If it is close to zero, it indicates a misconfiguration, and you will get 'no space left on device' even when the disk is not full
FilesystemMainPathTotalBytes	The size of the volume where the main ClickHouse path is mounted, in bytes
FilesystemMainPathTotalINodes	The total number of inodes on the volume where the main ClickHouse path is mounted. If it is less than 25 million, it indicates a misconfiguration
FilesystemMainPathUsedBytes	Used bytes on the volume where the main ClickHouse path is mounted
FilesystemMainPathUsedINodes	The number of used inodes on the volume where the main ClickHouse path is mounted. This value mostly corresponds to the number of files
HTTPThreads	Number of threads in the server of the HTTP interface (without TLS)
InterserverThreads	Number of threads in the server of the replicas communication protocol (without TLS)
Jitter	The difference in time the thread for calculation of the asynchronous metrics was scheduled to wake up and the time it was in fact, woken up. A proxy-indicator of overall system latency and responsiveness
LoadAverageN (LoadAverage1, LoadAverage15, LoadAverage5)	The whole system load, averaged with exponential smoothing over 1 minute. The load represents the number of threads across all the processes (the scheduling entities of the OS kernel), that are currently running by CPU or waiting for IO, or ready to run but not being scheduled at this point of time. This number includes all the processes, not only clickhouse-server. The number can be greater than the number of CPU cores, if the system is overloaded, and many processes are ready to run but waiting for CPU or IO
MMapCacheCells	The number of files opened with `mmap` (mapped in memory). This is used for queries with the setting `local_filesystem_read_method` set to `mmap`. The files opened with `mmap` are kept in the cache to avoid costly TLB flushes.
MarkCacheBytes	Total size of mark cache in bytes
MarkCacheFiles	Total number of mark files cached in the mark cache
MaxPartCountForPartition	Maximum number of parts per partition across all partitions of all tables of MergeTree family. Values larger than 300 indicates misconfiguration, overload, or massive data loading
MemoryCode	The amount of virtual memory mapped for the pages of machine code of the server process, in bytes
MemoryDataAndStack	The amount of virtual memory mapped for the use of stack and for the allocated memory, in bytes. It is unspecified whether it includes the per-thread stacks and most of the allocated memory, that is allocated with the `mmap` system call. This metric exists only for completeness reasons. It is recommended to use the MemoryResident metric for monitoring
MemoryResident	The amount of physical memory used by the server process, in bytes
MemoryShared	The amount of memory used by the server process, that is also shared by another processes, in bytes. ClickHouse does not use shared memory, but some memory can be labeled by OS as shared for its own reasons. This metric does not make a lot of sense to watch, and it exists only for completeness reasons
MemoryVirtual	The size of the virtual address space allocated by the server process, in bytes. The size of the virtual address space is usually much greater than the physical memory consumption, and should not be used as an estimate for the memory consumption. The large values of this metric are totally normal, and make only technical sense
NetworkReceiveBytes_eth0	Number of bytes received via the network interface. This is a system-wide metric, it includes all the processes on the host machine, not just clickhouse-server
NetworkReceiveDrop_eth0	Number of bytes a packet was dropped while received via the network interface. This is a system-wide metric, it includes all the processes on the host machine, not just clickhouse-server
NetworkReceiveErrors_eth0	Number of times error happened receiving via the network interface. This is a system-wide metric, it includes all the processes on the host machine, not just clickhouse-server
NetworkReceivePackets_eth0	Number of network packets received via the network interface. This is a system-wide metric, it includes all the processes on the host machine, not just clickhouse-server
NetworkSendBytes_eth0	Number of bytes sent via the network interface. This is a system-wide metric, it includes all the processes on the host machine, not just clickhouse-server
NetworkSendDrop_eth0	Number of times a packed was dropped while sending via the network interface. This is a system-wide metric, it includes all the processes on the host machine, not just clickhouse-server
NetworkSendErrors_eth0	Number of times error (e.g. TCP retransmit) happened while sending via the network interface. This is a system-wide metric, it includes all the processes on the host machine, not just clickhouse-server
NetworkSendPackets_eth0	Number of network packets sent via the network interface. This is a system-wide metric, it includes all the processes on the host machine, not just clickhouse-server
NumberOfDatabases	Total number of databases on the server
NumberOfTables	Total number of tables summed across the databases on the server, excluding the databases that cannot contain MergeTree tables. The excluded database engines are those who generate the set of tables on the fly, like Lazy, MySQL, PostgreSQL, SQlite
OSContextSwitches	The number of context switches that the system underwent on the host machine. This is a system-wide metric, it includes all the processes on the host machine, not just clickhouse-server
OSGuestNiceTime	The ratio of time spent running a virtual CPU for guest operating systems under the control of the Linux kernel, when a guest was set to a higher priority (see `man procfs`). This is a system-wide metric, it includes all the processes on the host machine, not just clickhouse-server. This metric is irrelevant for ClickHouse, but still exists for completeness. The value for a single CPU core will be in the `[0..1]` interval. The value for all CPU cores is calculated as a sum across them `[0..num cores]`
OSGuestNiceTimeCPUN	The ratio of time spent running a virtual CPU for guest operating systems under the control of the Linux kernel, when a guest was set to a higher priority (see `man procfs`). This is a system-wide metric, it includes all the processes on the host machine, not just clickhouse-server. This metric is irrelevant for ClickHouse, but still exists for completeness. The value for a single CPU core will be in the `[0..1]` interval. The value for all CPU cores is calculated as a sum across them `[0..num cores]`
OSGuestNiceTimeNormalized	The value is similar to OSGuestNiceTime but divided to the number of CPU cores to be measured in the `[0..1]` interval regardless of the number of cores. This allows you to average the values of this metric across multiple servers in a cluster even if the number of cores is non-uniform, and still get the average resource utilization metric
OSGuestTime	The ratio of time spent running a virtual CPU for guest operating systems under the control of the Linux kernel (see `man procfs`). This is a system-wide metric, it includes all the processes on the host machine, not just clickhouse-server. This metric is irrelevant for ClickHouse, but still exists for completeness. The value for a single CPU core will be in the `[0..1]` interval. The value for all CPU cores is calculated as a sum across them `[0..num cores]`
OSGuestTimeCPUN	The ratio of time spent running a virtual CPU for guest operating systems under the control of the Linux kernel (see `man procfs`). This is a system-wide metric, it includes all the processes on the host machine, not just clickhouse-server. This metric is irrelevant for ClickHouse, but still exists for completeness. The value for a single CPU core will be in the `[0..1]` interval. The value for all CPU cores is calculated as a sum across them `[0..num cores]`
OSGuestTimeNormalized	The value is similar to OSGuestTime but divided to the number of CPU cores to be measured in the `[0..1]` interval regardless of the number of cores. This allows you to average the values of this metric across multiple servers in a cluster even if the number of cores is non-uniform, and still get the average resource utilization metric
OSIOWaitTime	The ratio of time the CPU core was not running the code but when the OS kernel did not run any other process on this CPU as the processes were waiting for IO. This is a system-wide metric, it includes all the processes on the host machine, not just clickhouse-server. The value for a single CPU core will be in the `[0..1]` interval. The value for all CPU cores is calculated as a sum across them `[0..num cores]`
OSIOWaitTimeCPUN	The ratio of time the CPU core was not running the code but when the OS kernel did not run any other process on this CPU as the processes were waiting for IO. This is a system-wide metric, it includes all the processes on the host machine, not just clickhouse-server. The value for a single CPU core will be in the `[0..1]` interval. The value for all CPU cores is calculated as a sum across them `[0..num cores]`
OSIOWaitTimeNormalized	The value is similar to OSIOWaitTime but divided to the number of CPU cores to be measured in the `[0..1]` interval regardless of the number of cores. This allows you to average the values of this metric across multiple servers in a cluster even if the number of cores is non-uniform, and still get the average resource utilization metric
OSIdleTime	The ratio of time the CPU core was idle (not even ready to run a process waiting for IO) from the OS kernel standpoint. This is a system-wide metric, it includes all the processes on the host machine, not just clickhouse-server. This does not include the time when the CPU was under-utilized due to the reasons internal to the CPU (memory loads, pipeline stalls, branch mispredictions, running another SMT core). The value for a single CPU core will be in the `[0..1]` interval. The value for all CPU cores is calculated as a sum across them `[0..num cores]`
OSIdleTimeCPUN	The ratio of time the CPU core was idle (not even ready to run a process waiting for IO) from the OS kernel standpoint. This is a system-wide metric, it includes all the processes on the host machine, not just clickhouse-server. This does not include the time when the CPU was under-utilized due to the reasons internal to the CPU (memory loads, pipeline stalls, branch mispredictions, running another SMT core). The value for a single CPU core will be in the `[0..1]` interval. The value for all CPU cores is calculated as a sum across them `[0..num cores]`
OSIdleTimeNormalized	The value is similar to OSIdleTime but divided to the number of CPU cores to be measured in the `[0..1]` interval regardless of the number of cores. This allows you to average the values of this metric across multiple servers in a cluster even if the number of cores is non-uniform, and still get the average resource utilization metric
OSInterrupts	The number of interrupts on the host machine. This is a system-wide metric, it includes all the processes on the host machine, not just clickhouse-server
OSIrqTime	The ratio of time spent for running hardware interrupt requests on the CPU. This is a system-wide metric, it includes all the processes on the host machine, not just clickhouse-server. A high number of this metric may indicate hardware misconfiguration or a very high network load. The value for a single CPU core will be in the `[0..1]` interval . The value for all CPU cores is calculated as a sum across them `[0..num cores]`
OSIrqTimeCPUN	The ratio of time spent for running hardware interrupt requests on the CPU. This is a system-wide metric, it includes all the processes on the host machine, not just clickhouse-server. A high number of this metric may indicate hardware misconfiguration or a very high network load. The value for a single CPU core will be in the `[0..1]` interval. The value for all CPU cores is calculated as a sum across them `[0..num cores]`
OSIrqTimeNormalized	The value is similar to OSIrqTime but divided to the number of CPU cores to be measured in the `[0..1]` interval regardless of the number of cores. This allows you to average the values of this metric across multiple servers in a cluster even if the number of cores is non-uniform, and still get the average resource utilization metric
OSMemoryAvailable	The amount of memory available to be used by programs, in bytes. This is very similar to the OSMemoryFreePlusCached metric. This is a system-wide metric, it includes all the processes on the host machine, not just clickhouse-server
OSMemoryBuffers	The amount of memory used by OS kernel buffers, in bytes. This should be typically small, and large values may indicate a misconfiguration of the OS. This is a system-wide metric, it includes all the processes on the host machine, not just clickhouse-server
OSMemoryCached	The amount of memory used by the OS page cache, in bytes. Typically, almost all available memory is used by the OS page cache — high values of this metric are normal and expected. This is a system-wide metric, it includes all the processes on the host machine, not just clickhouse-server
OSMemoryFreePlusCached	The amount of free memory plus OS page cache memory on the host system, in bytes. This memory is available to be used by programs. The value should be very similar to OSMemoryAvailable. This is a system-wide metric, it includes all the processes on the host machine, not just clickhouse-server
OSMemoryFreeWithoutCached	The amount of free memory on the host system, in bytes. This does not include the memory used by the OS page cache memory, in bytes. The page cache memory is also available for usage by programs, so the value of this metric can be confusing. See the OSMemoryAvailable metric instead. For convenience we also provide the OSMemoryFreePlusCached metric, that should be somewhat similar to OSMemoryAvailable. See also https://www.linuxatemyram.com/. This is a system-wide metric, it includes all the processes on the host machine, not just clickhouse-server
OSMemoryTotal	The total amount of memory on the host system, in bytes
OSNiceTime	The ratio of time the CPU core was running userspace code with higher priority. This is a system-wide metric, it includes all the processes on the host machine, not just clickhouse-server. The value for a single CPU core will be in the `[0..1]` interval. The value for all CPU cores is calculated as a sum across them `[0..num cores]`
OSNiceTimeCPUN	The ratio of time the CPU core was running userspace code with higher priority. This is a system-wide metric, it includes all the processes on the host machine, not just clickhouse-server. The value for a single CPU core will be in the `[0..1]` interval. The value for all CPU cores is calculated as a sum across them `[0..num cores]`
OSNiceTimeNormalized	The value is similar to OSNiceTime but divided to the number of CPU cores to be measured in the `[0..1]` interval regardless of the number of cores. This allows you to average the values of this metric across multiple servers in a cluster even if the number of cores is non-uniform, and still get the average resource utilization metric
OSOpenFiles	The total number of opened files on the host machine. This is a system-wide metric, it includes all the processes on the host machine, not just clickhouse-server
OSProcessesBlocked	Number of threads blocked waiting for IO to complete (`man procfs`). This is a system-wide metric, it includes all the processes on the host machine, not just clickhouse-server
OSProcessesCreated	The number of processes created. This is a system-wide metric, it includes all the processes on the host machine, not just clickhouse-server
OSProcessesRunning	The number of runnable (running or ready to run) threads by the operating system. This is a system-wide metric, it includes all the processes on the host machine, not just clickhouse-server
OSSoftIrqTime	The ratio of time spent for running software interrupt requests on the CPU. This is a system-wide metric, it includes all the processes on the host machine, not just clickhouse-server. A high number of this metric may indicate inefficient software running on the system. The value for a single CPU core will be in the `[0..1]` interval. The value for all CPU cores is calculated as a sum across them `[0..num cores]`
OSSoftIrqTimeCPUN	The ratio of time spent for running software interrupt requests on the CPU. This is a system-wide metric, it includes all the processes on the host machine, not just clickhouse-server. A high number of this metric may indicate inefficient software running on the system. The value for a single CPU core will be in the `[0..1]` interval. The value for all CPU cores is calculated as a sum across them `[0..num cores]`
OSSoftIrqTimeNormalized	The value is similar to OSSoftIrqTime but divided to the number of CPU cores to be measured in the `[0..1]` interval regardless of the number of cores. This allows you to average the values of this metric across multiple servers in a cluster even if the number of cores is non-uniform, and still get the average resource utilization metric
OSStealTime	The ratio of time spent in other operating systems by the CPU when running in a virtualized environment. This is a system-wide metric, it includes all the processes on the host machine, not just clickhouse-server. Not every virtualized environments present this metric, and most of them don't. The value for a single CPU core will be in the `[0..1]` interval. The value for all CPU cores is calculated as a sum across them `[0..num cores]`
OSStealTimeCPUN	The ratio of time spent in other operating systems by the CPU when running in a virtualized environment. This is a system-wide metric, it includes all the processes on the host machine, not just clickhouse-server. Not every virtualized environments present this metric, and most of them don't. The value for a single CPU core will be in the `[0..1]` interval. The value for all CPU cores is calculated as a sum across them `[0..num cores]`
OSStealTimeNormalized	The value is similar to OSStealTime but divided to the number of CPU cores to be measured in the `[0..1]` interval regardless of the number of cores. This allows you to average the values of this metric across multiple servers in a cluster even if the number of cores is non-uniform, and still get the average resource utilization metric
OSSystemTime	The ratio of time the CPU core was running OS kernel (system) code. This is a system-wide metric, it includes all the processes on the host machine, not just clickhouse-server. The value for a single CPU core will be in the `[0..1]` interval. The value for all CPU cores is calculated as a sum across them `[0..num cores]`
OSSystemTimeCPUN	The ratio of time the CPU core was running OS kernel (system) code. This is a system-wide metric, it includes all the processes on the host machine, not just clickhouse-server. The value for a single CPU core will be in the `[0..1]` interval. The value for all CPU cores is calculated as a sum across them `[0..num cores]`
OSSystemTimeNormalized	The value is similar to OSSystemTime but divided to the number of CPU cores to be measured in the `[0..1]` interval regardless of the number of cores. This allows you to average the values of this metric across multiple servers in a cluster even if the number of cores is non-uniform, and still get the average resource utilization metric
OSThreadsRunnable	The total number of runnable threads, as the OS kernel scheduler seeing it
OSThreadsTotal	The total number of threads, as the OS kernel scheduler seeing it
OSUptime	The uptime of the host server (the machine where ClickHouse is running), in seconds
OSUserTime	The ratio of time the CPU core was running userspace code. This is a system-wide metric, it includes all the processes on the host machine, not just clickhouse-server. This includes also the time when the CPU was under-utilized due to the reasons internal to the CPU (memory loads, pipeline stalls, branch mispredictions, running another SMT core). The value for a single CPU core will be in the `[0..1]` interval. The value for all CPU cores is calculated as a sum across them `[0..num cores]`
OSUserTimeCPUN	The ratio of time the CPU core was running userspace code. This is a system-wide metric, it includes all the processes on the host machine, not just clickhouse-server. This includes also the time when the CPU was under-utilized due to the reasons internal to the CPU (memory loads, pipeline stalls, branch mispredictions, running another SMT core). The value for a single CPU core will be in the `[0..1]` interval. The value for all CPU cores is calculated as a sum across them `[0..num cores]`
OSUserTimeNormalized	The value is similar to OSUserTime but divided to the number of CPU cores to be measured in the `[0..1]` interval regardless of the number of cores. This allows you to average the values of this metric across multiple servers in a cluster even if the number of cores is non-uniform, and still get the average resource utilization metric
ReplicasMaxAbsoluteDelay	Maximum difference in seconds between the most fresh replicated part and the most fresh data part still to be replicated, across Replicated tables. A very high value indicates a replica with no data
ReplicasMaxInsertsInQueue	Maximum number of INSERT operations in the queue (still to be replicated) across Replicated tables
ReplicasMaxMergesInQueue	Maximum number of merge operations in the queue (still to be applied) across Replicated tables
ReplicasMaxQueueSize	Maximum queue size (in the number of operations like get, merge) across Replicated tables
ReplicasMaxRelativeDelay	Maximum difference between the replica delay and the delay of the most up-to-date replica of the same table, across Replicated tables
ReplicasSumInsertsInQueue	Sum of INSERT operations in the queue (still to be replicated) across Replicated tables
ReplicasSumMergesInQueue	Sum of merge operations in the queue (still to be applied) across Replicated tables
ReplicasSumQueueSize	Sum queue size (in the number of operations like get, merge) across Replicated tables
TCPThreads	Number of threads in the server of the TCP protocol (without TLS)
TotalBytesOfMergeTreeTables	Total amount of bytes (compressed, including data and indices) stored in all tables of MergeTree family
TotalPartsOfMergeTreeTables	Total amount of data parts in all tables of MergeTree family. Numbers larger than `10000` will negatively affect the server startup time and it may indicate unreasonable choice of the partition key
TotalRowsOfMergeTreeTables	Total amount of rows (records) stored in all tables of MergeTree family
UncompressedCacheBytes	Total size of uncompressed cache in bytes. Uncompressed cache does not usually improve the performance and should be mostly avoided
UncompressedCacheCells	Total number of entries in the uncompressed cache. Each entry represents a decompressed block of data. Uncompressed cache does not usually improve performance and should be mostly avoided
Uptime	The server uptime in seconds. It includes the time spent for server initialization before accepting connections
jemalloc.arenas.all.dirty_purged	An internal metric of the low-level memory allocator (see jemalloc)
jemalloc.arenas.all.muzzy_purged	An internal metric of the low-level memory allocator (see jemalloc)
jemalloc.arenas.all.pactive	An internal metric of the low-level memory allocator (see jemalloc)
jemalloc.arenas.all.pdirty	An internal metric of the low-level memory allocator (see jemalloc)
jemalloc.arenas.all.pmuzzy	An internal metric of the low-level memory allocator (see jemalloc)
jemalloc.background_thread.num_runs	An internal metric of the low-level memory allocator (see jemalloc)
jemalloc.background_thread.num_threads	An internal metric of the low-level memory allocator (see jemalloc)
jemalloc.background_thread.run_intervals	An internal metric of the low-level memory allocator (see jemalloc)
jemalloc.active	An internal metric of the low-level memory allocator (see jemalloc)
jemalloc.allocated	An internal metric of the low-level memory allocator (see jemalloc)
jemalloc.epoch	An internal incremental update number of the statistics of jemalloc, used in all other jemalloc metrics
jemalloc.mapped	An internal metric of the low-level memory allocator (see jemalloc)
jemalloc.metadata	An internal metric of the low-level memory allocator (see jemalloc)
jemalloc.metadata_thp	An internal metric of the low-level memory allocator (see jemalloc)
jemalloc.resident	An internal metric of the low-level memory allocator (see jemalloc)
jemalloc.retained	An internal metric of the low-level memory allocator (see jemalloc)

View metrics

You can view monitoring metrics for ADQM in the system.metrics, system.events, and system.asynchronous_metrics system tables.

A query example:

```
SELECT * FROM system.metrics LIMIT 5;
```

The output:

```
┌─metric───────────┬─value─┬─description──────────────────────────────┐
│ Query            │     1 │ Number of executing queries               │
│ Merge            │     0 │ Number of executing background merges     │
│ PartMutation     │     0 │ Number of mutations (ALTER DELETE/UPDATE) │
│ ReplicatedFetch  │     0 │ Number of data parts being fetched from replica │
│ ReplicatedSend   │     0 │ Number of data parts being sent to replicas │
└──────────────────┴───────┴───────────────────────────────────────────┘
```

You can also view metrics in the Prometheus and Grafana web interfaces.

Logging

Elena Dvoryadkina

Contents

To Table of Contents

Log files

Log tables

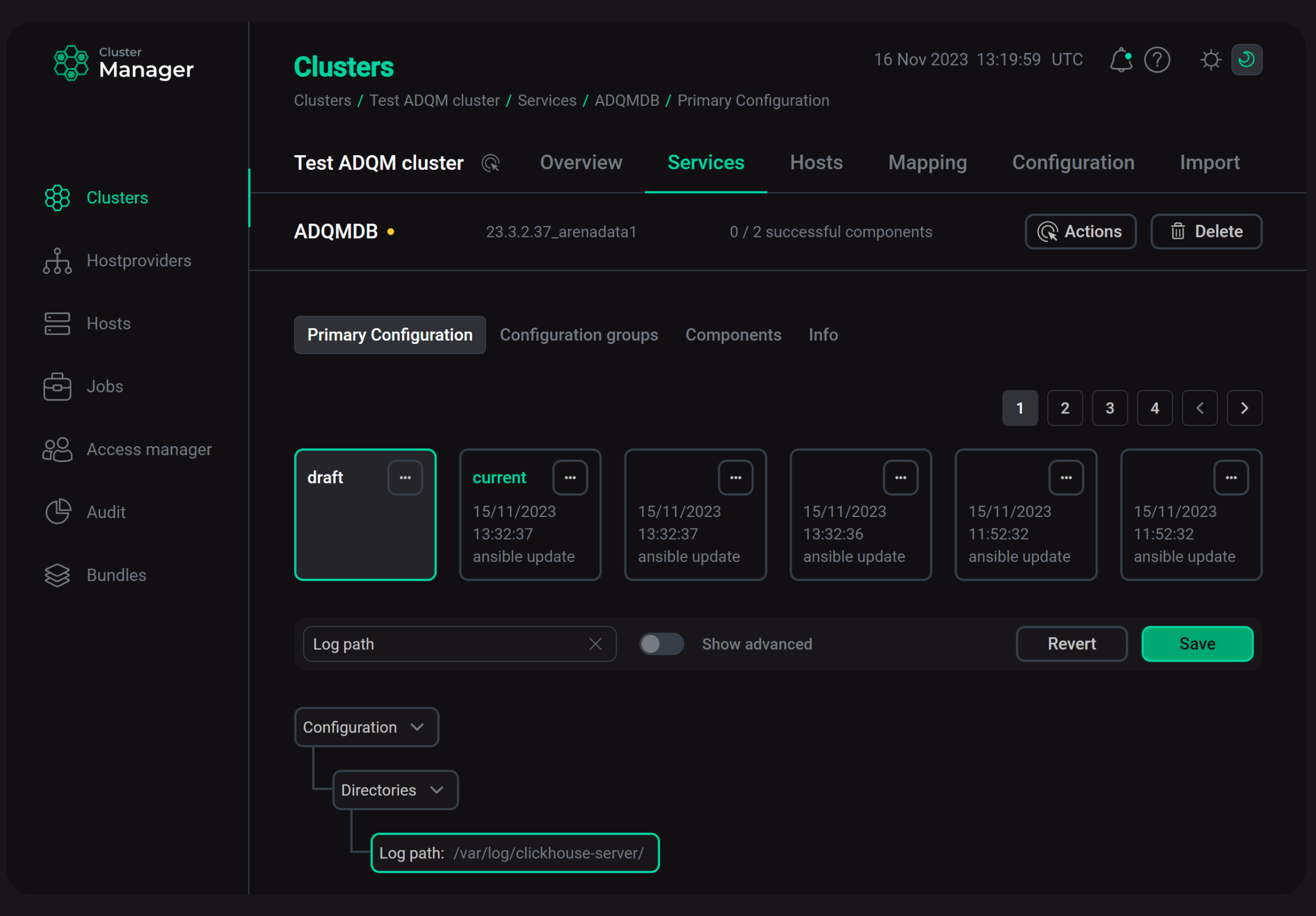
ADQM writes general and error text logs that can be useful for analyzing the causes of different errors. Additionally, you can enable logs of other types (for example, collect information about query execution, dependent views, threads, stack traces) that will be saved to special system tables in ADQM.

Log files

By default, ADQM saves the following logs to the */var/log/clickhouse-server* directory:

- *clickhouse-server.log* — main log of an ADQM server (the log verbosity level is configured by a special parameter described below);
- *clickhouse-server.err.log* — error log (contains records of the `warning`, `error`, `critical`, and `fatal` levels).

You can change the storage directory for log files using the ***Log path*** parameter in the ***Directories*** section on the ADQMDB service's configuration page of the ADCM interface.



Log path: /var/log/clickhouse-server/

Set the directory to store log files

On the same page, you can configure logging parameters in the ***Log settings*** section that is visible when the ***Show advanced*** option is activated:

- ***Logging level*** — verbosity level of messages written to the *clickhouse-server.log* file. Possible values: `trace`, `debug`, `information`, `notice`, `warning`, `error`, `critical`, `fatal`, `none`.
  The logging level for *clickhouse-server.err.log* cannot be separately configured.
- ***Log file size, MB*** — log file size (in MB). Once a file reaches the specified size, ADQM archives and renames it, and creates a new log file in its place.
- ***Log files count*** — number of archived log files that ADQM stores.

> **IMPORTANT**
>
> After you have set necessary parameters, do not forget to click ***Save*** and execute the ***Reconfig*** action for the ADQMDB service to save configuration changes.

To view the content of a log file, use the following command:

```
$ sudo less /var/log/clickhouse-server/clickhouse-server.log
```

The output can be similar to:

```
2023.09.08 10:59:14.549245 [ 31027 ] {} <Information> Application: Will watch for the process with pid 31036
2023.09.08 10:59:14.549357 [ 31036 ] {} <Information> Application: Forked a child process to watch
2023.09.08 10:59:14.549768 [ 31036 ] {} <Information> SentryWriter: Sending crash reports is disabled
2023.09.08 10:59:14.549851 [ 31036 ] {} <Trace> Pipe: Pipe capacity is 1.00 MiB
2023.09.08 10:59:14.674600 [ 31036 ] {} <Information> : Starting ClickHouse 22.3.7.28 with revision 54460, build id:
6F5FF3E355A49AA523CB708AFAC0EB6C5EE73278, PID 31036
2023.09.08 10:59:14.674685 [ 31036 ] {} <Information> Application: starting up
2023.09.08 10:59:14.674703 [ 31036 ] {} <Information> Application: OS name: Linux, version: 3.10.0-1160.66.1.el7.x86_64,
architecture: x86_64
2023.09.08 10:59:14.678336 [ 31036 ] {} <Trace> AsynchronousMetrics: Scanning /sys/class/thermal
2023.09.08 10:59:14.678369 [ 31036 ] {} <Trace> AsynchronousMetrics: Scanning /sys/block
2023.09.08 10:59:14.678516 [ 31036 ] {} <Trace> AsynchronousMetrics: Scanning /sys/devices/system/edac
2023.09.08 10:59:14.678536 [ 31036 ] {} <Trace> AsynchronousMetrics: Scanning /sys/class/hwmon
2023.09.08 10:59:14.981938 [ 31036 ] {} <Warning> Application: Calculated checksum of the binary:
1841F46AA9C864447AE24AC0D983E86C. There is no information about the reference checksum.
2023.09.08 10:59:14.991365 [ 31036 ] {} <Debug> Application: rlimit on number of file descriptors is 500000
2023.09.08 10:59:14.991394 [ 31036 ] {} <Debug> Application: Initializing DateLUT.
2023.09.08 10:59:14.991397 [ 31036 ] {} <Trace> Application: Initialized DateLUT with time zone 'UTC'.
2023.09.08 10:59:14.991426 [ 31036 ] {} <Debug> Application: Setting up /var/lib/clickhouse/tmp/ to store temporary data in it
2023.09.08 10:59:14.993872 [ 31036 ] {} <Debug> Application: Initiailizing interserver credentials.
2023.09.08 10:59:14.994558 [ 31036 ] {} <Debug> ConfigReloader: Loading config '/etc/clickhouse-server/config.xml'
2023.09.08 10:59:14.995658 [ 31036 ] {} <Debug> ConfigReloader: Loaded config '/etc/clickhouse-server/config.xml', performing
update on configuration
...
```

If an ADQM server has started successfully, the log contains the following messages:

- `<Information> Application: starting up` — server starts.
- `<Information> Application: Ready for connections.` — server is running and ready to handle client connections.

If an ADQM server start has failed with a configuration error, the log provides an `<Error>` string with an error description. For example:

```
...
2023.09.08 10:59:14.991426 [ 31036 ] {} <Error> Application: DB::Exception: A setting 'query_profiler_cpu_time_period_ns' appeared
at top level in config /etc/clickhouse-server/config.xml. But it is user-level setting that should be located in users.xml inside
<profiles> section for specific profile. You can add it to <profiles><default> if you want to change default value of this
setting. You can also disable the check - specify <skip_check_for_incorrect_settings>1</skip_check_for_incorrect_settings> in the
main configuration file.
...
```

If there is no an error message at the end of the log file, look through the entire file starting with the `<Information> Application: starting up` string.

Log tables

ADQM supports a set of system tables to store logs of different types. To start logging, activate the corresponding parameter in the ***Log settings*** section on the configuration page of the ADQMDB service — ADQM will create a system table with the same name as the parameter and will begin writing the log to it. Parameters responsible for enabling different logging types are listed in the table below. This table also describes which kind of information each log contains.

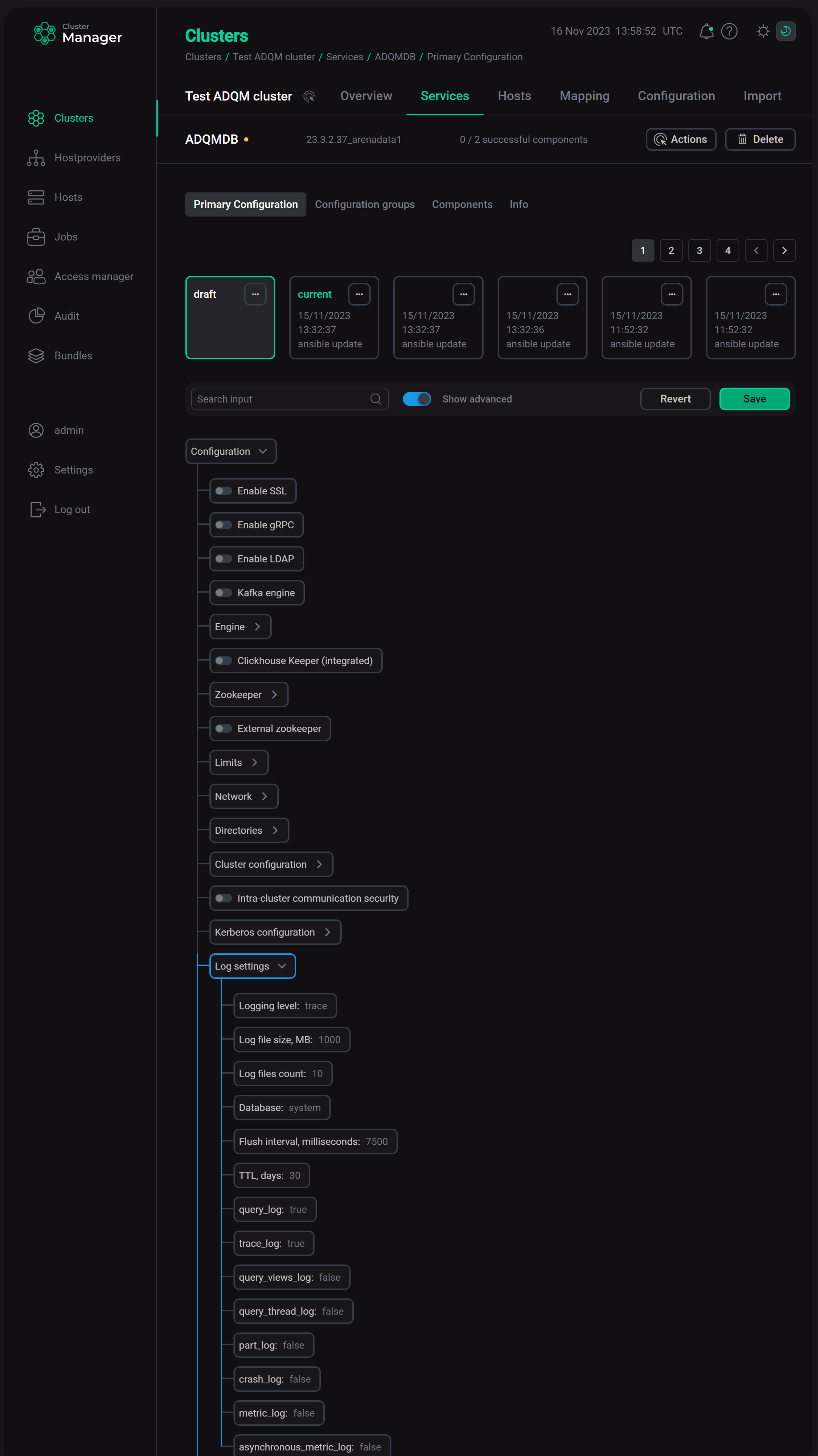
ADQM log types

Parameter to start logging	Log table	Log table data
query_log	query_log	Information about executed queries (for example, query start time, duration of query processing, error messages)
trace_log	trace_log	Stack traces collected by the sampling query profiler (useful for deep performance analysis)
query_views_log	query_views_log	Information about dependent views executed when processing queries received with the `log_query_views = 1` setting (for example, view type, duration of view execution)
query_thread_log	query_thread_log	Information about threads that execute queries received with the `log_query_threads = 1` setting (for example, thread name, time when a thread has started and finished query execution, duration of query execution, number of read and written bytes/rows)
part_log	part_log	Information about events related to data parts of MergeTree tables (for example, inserting/merging/deleting data parts, downloading data from replicas, moving data between disks)
crash_log	crash_log	Information about stack traces for fatal errors
session_log	session_log	Information about all successful and failed login and logout events
metric_log	metric_log	Historical values of metrics from the `system.metrics` and `system.events` tables
asynchronous_metric_log	asynchronous_metric_log	Historical values of metrics from the `system.asynchronous_metrics` table
asynchronous_insert_log	asynchronous_insert_log	Information about asynchronous inserts

> **NOTE**
>
> Data of tables with system logs is stored in a file system and is not deleted automatically. If you remove a table from a file system, ADQM will create an empty table again the next time when data is written to the log. If a system table schema changed when the ADQM server was updated, the current table is renamed and a new one is created.

In the ***Log settings*** section, you can set the following settings that are common to all log tables:

- ***Database*** — database to which log tables belong;
- ***Flush interval, milliseconds*** — interval for flushing data from the memory buffer to tables (in milliseconds);
- ***TTL, days*** — lifetime of logs (in days).



Logging level: trace
Log file size, MB: 1000
Log files count: 10
Database: system
Flush interval, milliseconds: 7500
TTL, days: 30
query_log: true
trace_log: true
query_views_log: false
query_thread_log: false
part_log: false
crash_log: false
metric_log: false
asynchronous_metric_log: false

Log settings

Databases

Elena Dvoryadkina

Contents

To Table of Contents

- Overview
- Create a database
- Atomic specifics
  - Table UUID
  - RENAME TABLE
  - EXCHANGE TABLES/DICTIONARIES
  - DROP/DETACH TABLE

Overview

ADQM supports a set of **database engines** optimized for specific tasks:

- Atomic — the main database engine that ADQM uses by default (see the Atomic specifics section below);
- MySQL — engine that allows connecting to databases on a remote MySQL server;
- MaterializedMySQL — experimental engine that allows replicating a MySQL database to ADQM;
- PostgreSQL — engine that allows connecting to databases on a remote PostgreSQL server;
- MaterializedPostgreSQL — experimental engine that allows replicating a PostgreSQL database to ADQM;
- SQLite — engine that allows connecting to SQLite databases;
- Lazy — in-memory database engine for working with Log tables;
- Replicated — experimental engine that is based on Atomic and supports replication of an entire database.

For detailed information on all database engines, see the ClickHouse documentation.

ADQM contains the following databases by default:

- `system` — a database with system tables that contain information about server states, processes, and environment, as well as system logs (`*_log` tables);
- `INFORMATION_SCHEMA` (`information_schema`) — a system database containing views that you can use to get metadata of database objects;
- `default` — an Atomic database that is set by default during ADQM installation (via the `Default database` configuration parameter of the ADQMDB service).

To get a list of all databases on the ADQM server, use the SHOW DATABASES query:

```
SHOW DATABASES;
```

For example, the output can look like:

```
--name----------------
│ INFORMATION_SCHEMA │
│ default            │
│ information_schema │
│ system             │
│ test_database      │
----------------------
```

Create a database

To create a database, use the CREATE DATABASE query:

```
CREATE DATABASE <database_name> [ENGINE = <database_engine>];
```

If you skip the `ENGINE` parameter, an Atomic database is created. Specifics of the Atomic database engine are listed below.

Atomic specifics

Table UUID

Each table in an Atomic database has UUID. UUID format — `xxxyyyyy-yyyy-yyyy-yyyy-yyyyyyyyyyyy`.

A table stores data in the */var/lib/clickhouse/store/xxx/xxxyyyyy-yyyy-yyyy-yyyy-yyyyyyyyyyyy* directory, to which the following link refers: */var/lib/clickhouse/data/<database_name>/<table_name>*, where `<database_name>` and `<table_name>` are database and table names.

As a table name is a link in the file system, the `RENAME TABLE` and `DROP TABLE` queries are quick and non-blocking. The `EXCHANGE TABLES` and `EXCHANGE DICTIONARIES` atomic operations are also available.

RENAME TABLE

The RENAME TABLE query renames a table. It is executed without changing UUID or moving table data. This query does not wait for queries that use the table to complete and is executed instantly — when a new name of a table is specified, the corresponding new link to a directory with table data appears, and the old link is then deleted.

EXCHANGE TABLES/DICTIONARIES

The EXCHANGE query exchanges the names of two tables or two dictionaries atomically (a dictionary is a key/value data store that is fully or partially stored in the RAM of the ADQM server and can be used as a reference to substitute data values by keys in the final sample).

For example, you can use a single atomic query:

```
EXCHANGE TABLES new_table AND old_table;
```

instead of the following non-atomic operation:

```
RENAME TABLE new_table TO tmp, old_table TO new_table, tmp TO old_table;
```

DROP/DETACH TABLE

You can use the DROP TABLE query to delete a table or the DETACH TABLE query to detach it (make the table "invisible" to the server).

The `DROP TABLE` query does not remove data immediately. An Atomic database marks a table as dropped (it cannot receive new queries), moves metadata to the */var/lib/clickhouse/metadata_dropped/* directory, and notifies the background thread about this. The database_atomic_delay_before_drop_table_sec setting defines a delay before the final deletion of data.

You can use the `SYNC` modifier to enable the synchronous mode (the database_atomic_wait_for_drop_and_detach_synchronously setting adds the `SYNC` modifier to all `DROP` and `DETACH` queries). In this case, a table will be deleted immediately after the completion of `SELECT`, `INSERT` and other queries that use the table.

Table engines

Elena Dvoryadkina

Contents

To Table of Contents

- Table engine families
- MergeTree family
  - Features of tables of the MergeTree family
  - Data insertion and storage
  - Special versions of the MergeTree engine
  - Replicated table engines

Table engine in ADQM/ClickHouse is a table type that determines:

- how and where data is stored;
- supported queries;
- concurrent data access;
- whether indexes can be used;
- whether multi-threaded query execution is supported;
- data replication parameters.

Table engine families

ADQM/ClickHouse supports a variety of table engines that are optimized for different tasks. All engines are grouped into families:

- **MergeTree** — the most universal and functional table engines for storing big data and high-performance search. The main feature of these engines is fast data insertion followed by background data processing. For details, see the MergeTree family section below.
- **Log** — simple engines for scenarios when it is required to quickly write a lot of small tables (up to 1 million rows) and then read them in full.
- **Integration** — engines for communicating with external data storages (ODBC, JDBC, MySQL, MongoDB, HDFS, S3, Kafka, EmbeddedRocksDB, RabbitMQ, PostgreSQL). You can find detailed information about these engines with examples in the Table Engines for Integrations section of the ClickHouse documentation.
- **Special** — engines for special tasks, for example:
  - Distributed — engine for distributed query processing on multiple servers. Read more about creating and using Distributed tables in the Sharding article.
  - Dictionary — engine that displays a dictionary data as a table.
  - Merge (not to be confused with the MergeTree engine) — engine that allows reading data from any number of other tables simultaneously. A Merge table does not store data itself and does not support writing data.
  - Other engines — read more in the Special Table Engines section of the ClickHouse documentation.

You can get information on features supported by table engines from the system.table_engines system table, for example:

```
SELECT * FROM system.table_engines WHERE name in ('MergeTree', 'TinyLog', 'MySQL');
```

The query result:

```
--name--------supports_settings---supports_skipping_indices---supports_projections---supports_sort_order---supports_ttl---
supports_replication---supports_deduplication---supports_parallel_insert--
| MySQL     |                 1 |                         0 |                    0 |                   0 |            0 |
0 |                      0 |                        0 |
| MergeTree |                 1 |                         1 |                    1 |                   1 |            1 |
0 |                      0 |                        1 |
| TinyLog   |                 1 |                         0 |                    0 |                   0 |            0 |
0 |                      0 |                        0 |
---------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------
```

MergeTree family

Features of tables of the MergeTree family

MergeTree is the main table engine for working with big data in ADQM. This engine and all its special versions support:

- partitioning — combining table data into **partitions** by the specified criterion (in other words, a partition is a logical group of data rows that you can manipulate as needed — detach/attach, move, etc.);
- data replication;
- data sampling;
- sparse index for optimized data search.

Data insertion and storage

Features related to data storage in tables of the MergeTree family:

- The MergeTree table engine inserts data as a set of separate **parts** and then merges them in the background.
- Data in each part is sorted by the primary key.
- Data that belongs to different partitions is split into different parts. Data parts of different partitions are not merged.
- The merge process is managed by the system (the system tends to a state when a partition stores only one sorted data part), but you can force merging with the OPTIMIZE query.
- Each data part is logically divided into granules. A **granule** is the smallest indivisible data set that is read when selecting data. The size of granules is set by the engine's `index_granularity` (the maximum number of data rows in a granule) and `index_granularity_bytes` (the maximum size of a granule in bytes) settings.
- Each data part is stored in a separate folder inside the table folder. Data parts can be stored both in the column format (each column is stored in a separate file) or in the compact format (all columns are stored in one file).
- At a particular moment in time, a table can store more than one copy of data — unmerged parts and their merged version.
- Old parts, whose data was merged into a new part, are removed after the time specified by the table's `old_parts_lifetime` setting.
- You can manipulate data parts as partitions — move, detach/attach, etc.

Special versions of the MergeTree engine

The MergeTree family also includes the following table engines that inherit the MergeTree functionality and also add extra functionality for specific use cases:

- ReplacingMergeTree — removes duplicate rows with the same sorting key value. This table engine is suitable for cleaning of duplicate data in the background to save space, however it does not guarantee the absence of duplicates.
- SummingMergeTree — replaces all rows with the same sorting key with one row that stores summarized values in numeric columns.
- AggregatingMergeTree — used for creating materialized views.
- CollapsingMergeTree and VersionedCollapsingMergeTree — add the logic of collapsing (deleting) rows to the algorithm for merging data parts.
- GraphiteMergeTree — thins and aggregates/averages Graphite data.

Replicated table engines

Engines of the MergeTree family with the `Replicated` prefix are used for creating replicated tables:

- ReplicatedMergeTree
- ReplicatedReplacingMergeTree
- ReplicatedSummingMergeTree
- ReplicatedAggregatingMergeTree
- ReplicatedCollapsingMergeTree
- ReplicatedVersionedCollapsingMergeTree
- ReplicatedGraphiteMergeTree

For an example on how to create replicated tables using the ReplicatedMergeTree engine, see the Typical cluster article.

Tables overview

Elena Dvoryadkina

Contents

To Table of Contents

- Create a MergeTree table
  - Syntax overview
  - Examples
- View the list of tables
- Get information on the specified table
  - Table structure
  - Table metadata
- Insert data into a table
- Read data from a table
- Modify a table
- Detach and attach a table
- Delete a table

ADQM supports an SQL language so you can run different queries to operate data tables. This article describes syntax and provides examples of queries to create/modify/delete tables, view table information, insert and read data.

Create a MergeTree table

Syntax overview

To create a new table, use the `CREATE TABLE` query with a table engine specified via the `ENGINE` clause. See ClickHouse documentation for more details on the `CREATE TABLE` query.

Below is the basic syntax of the `CREATE TABLE` query for creating a table based on the MergeTree engine — the main table engine for working with big data in ADQM.

```
CREATE TABLE [IF NOT EXISTS] [<db_name>.]<table_name> [ON CLUSTER <cluster_name>]
(   <column_name1> [<data_type1>] [NULL|NOT NULL] [DEFAULT|MATERIALIZED|EPHEMERAL|ALIAS <expr1>] [CODEC(<compression_codec1>)]
[TTL <expr1>],
    <column_name2> [<data_type2>] [NULL|NOT NULL] [DEFAULT|MATERIALIZED|EPHEMERAL|ALIAS <expr2>] [CODEC(<compression_codec2>)]
[TTL <expr2>],
    ...
    CONSTRAINT <constraint_name_1> CHECK <boolean_expr_1>,
    CONSTRAINT <constraint_name_2> CHECK <boolean_expr_2>,
    ...
    INDEX <index_name1> <expr1> TYPE <index_type1> GRANULARITY <granularity_value1>,
    INDEX <index_name2> <expr2> TYPE <index_type2> GRANULARITY <granularity_value2>)
ENGINE = MergeTree()
ORDER BY <expr>
[PARTITION BY <expr>]
[PRIMARY KEY <expr>]
[SAMPLE BY <expr>]
[TTL <expr>
    [DELETE|TO DISK 'xxx'|TO VOLUME 'xxx' [, ...] ]
    [WHERE <conditions>]
    [GROUP BY <key_expr> [SET v1 = aggr_func(v1) [, v2 = aggr_func(v2) ...]] ] ]
[SETTINGS name=value, ...];
```

Query parameters

In a `CREATE TABLE` query, you can specify a table structure explicitly. To do this, pass column descriptions, constraints, and secondary indexes as query parameters.

Column descriptions

MergeTree table columns are described as follows:

```
<column_name> [<data_type>] [NULL|NOT NULL] [DEFAULT|MATERIALIZED|EPHEMERAL|ALIAS <expr>]
[CODEC(<compression_codec>)] [TTL <expr>]
```

A column description includes:

- column name;
- data type (to check whether a data type name is case-sensitive, use the system.data_type_families system table);
- `NULL` or `NOT NULL` modifier that specifies whether a column data type is Nullable;
- expression for a default value of the specified type (`DEFAULT`, `MATERIALIZED`, `EPHEMERAL` or `ALIAS`);
- column compression codec;
- TTL expression that defines time to store column values.

Constraints

Constraints for column values are described as follows:

```
CONSTRAINT <constraint_name> CHECK <boolean_expr>
```

A constraint description includes:

- constraint name;
- boolean expression that data values to be inserted into the column should satisfy.

Data skipping indexes

For tables of the MergeTree family, you can set data skipping indexes that aggregate data into blocks according to the specified expressions. These aggregates allow skipping blocks that do not contain requested data when executing a `SELECT` query and reducing the amount of data to read from the disk.

Data skipping indexes are described in the following format:

```
INDEX <index_name> <expr> TYPE <index_type> GRANULARITY <granularity_value>
```

An index description contains:

- index name;
- data aggregation expression;
- index type. Most commonly used indexes are:
  - `minmax` — stores the minimum and maximum values of the specified expression;
  - `set(max_rows)` — stores unique values (no more than `max_rows`) of the specified expression;
  - `bloom_filter([false_positive])` — stores a Bloom filter for the specified columns. The optional `false_positive` parameter is the probability of receiving a false positive response from the filter. Possible values are `(0, 1)`. A default value is `0.025`.
- granularity — the number of data granules in a block to be skipped.

You can also add and remove columns, constraints, and indexes after the table has been created. To do this, use `ALTER TABLE` queries (see the Modify a table section below).

Basic clauses

ON CLUSTER	By default, a table is created only on the server on which the query is run. To create a table on all servers of the cluster, use the `ON CLUSTER` clause with the specified cluster name
ENGINE	Table engine with parameters. Names of table engines are case-sensitive. You can obtain them from the system.table_engines system table
ORDER BY	Sorting key defined as one or more columns. ADQM uses the sorting key as a primary key if the primary key is not specified explicitly via the `PRIMARY KEY` clause
PARTITION BY	Partitioning key. A partitioning key can be a table column or any expression from table columns. For example, to specify partitioning by months, use the `toYYYYMM(date_column)` expression, where `date_column` is a column of the Date type. The partition key can also be a tuple of columns or expressions
PRIMARY KEY	Primary key. By default, a primary key is the same as a sorting key (`ORDER BY`), and it is not necessary to specify the `PRIMARY KEY` clause separately in most cases. If the primary key differs from the sorting key, then the expression tuple of the primary key must be a prefix of the sorting key expression tuple
SAMPLE BY	Expression for sampling
TTL	List of TTL (time-to-live) rules that define how long rows should be stored in a table, as well as how data should be processed after the specified time has passed (for example, old data can be automatically deleted, moved to other disks/volumes, recompressed with another codec, or aggregated/grouped — for more information, see Use TTL to manage data)
SETTINGS	Additional parameters that manage the behavior of a MergeTree table. You can use the system.merge_tree_settings system table to access the list of settings for tables of the MergeTree family and their values

In the `CREATE TABLE` query (as well as in other queries listed in this article), `<table_name>` is the name of a table and `<db_name>` is the name of a database that contains the table. If the database is not specified explicitly, the query applies to the table in the current database of the session. To set the current database, use the following query:

```
USE <db_name>;
```

Examples

Create a MergeTree table

The following query creates a **MergeTree** table with three columns that store numeric IDs, string values, and dates. It sets a tuple of the column with identifiers and the column with dates as the sorting key (the primary key will match the sorting key), and specifies the key for partitioning data by days.

```
CREATE TABLE table_mergeTree
(
    id Int32,
    value String,
    date Date
)
ENGINE = MergeTree
ORDER BY (id, date)
PARTITION BY toYYYYMMDD(date);
```

The MergeTree engine has no parameters, so you can specify it as `ENGINE = MergeTree()` or `ENGINE = MergeTree`.

Create a ReplacingMergeTree table

A **ReplacingMergeTree** table removes duplicate rows with the same sorting key value (`ORDER BY`, not `PRIMARY KEY`) when merging data parts.

The ReplacingMergeTree engine has one optional parameter (`ver`) — a column (of the UInt*, Date, or DateTime type) that stores versions of data records.

When merging data parts, ReplacingMergeTree leaves only one row from all the rows with the same sorting key:

- **The most recently inserted row**, if the `ver` parameter is not specified.

```
CREATE TABLE table_replacing_1
(
    id Int32,
    value String,
    date Date,
    ver Int32
)
ENGINE = ReplacingMergeTree()
ORDER BY id;

INSERT INTO table_replacing_1 VALUES(1, 'value1', '2022-01-01', 1), (1, 'value2', '2022-01-01', 2), (1, 'value3', '2022-01-01',
-1);

SELECT * FROM table_replacing_1 FINAL;
```

After deduplication, the last inserted row remains in the table:

```
--id---value---------date---ver--
│  1 │ value3 │ 2022-01-01 │  -1 │
---------------------------------
```

- **A row with the maximum version**, if the `ver` parameter is specified.

```
CREATE TABLE table_replacing_2
(
    id Int32,
    value String,
    date Date,
    ver Int32
)
ENGINE = ReplacingMergeTree(ver)
ORDER BY id;

INSERT INTO table_replacing_2 VALUES(1, 'value1', '2022-01-01', 1), (1, 'value2', '2022-01-01', 2), (1, 'value3', '2022-01-01',
-1);

SELECT * FROM table_replacing_2 FINAL;
```

After deduplication, a row with the greatest value in the `ver` column remains in the table:

```
--id---value---------date---ver--
│  1 │ value2 │ 2022-01-01 │   2 │
---------------------------------
```

Deduplication is performed at the partition level. If a table is partitioned, duplicates between partitions are not removed.

ReplacingMergeTree deduplicates data during the data merge which is performed in the background. So, you can get some data unprocessed if you run the `SELECT` query between merges. To initiate an unscheduled merge and get a table or partition without duplicates, you can use the `FINAL` modifier after the table name in the `SELECT` query, or you can pre-execute the `OPTIMIZE` query for the table or partition.

Create a SummingMergeTree table

The **SummingMergeTree** engine finds rows with the same sorting keys and replaces these rows with a single row that contains summed values in numeric columns. Values in columns of other types are chosen arbitrarily from any row.

The SummingMergeTree engine has one optional parameter (`columns`) — names of numeric columns whose values will be summed (these columns should not be in the sorting key).

- If the `columns` parameter is not specified, ADQM sums data values in all columns of numeric types that are not parts of the sorting key.

```
CREATE TABLE table_sum_1
(
    id UInt32,
    value1 Int32,
    value2 Int32,
    value3 Int32
)
ENGINE = SummingMergeTree()
ORDER BY id;

INSERT INTO table_sum_1 VALUES(1, 1, 1, 1), (1, 2, 2, 2), (1, 3, 3, 3);

SELECT * FROM table_sum_1;
```

As a result, values in all numeric columns were summed up:

```
--id---value1---value2---value3--
│  1 │      6 │      6 │      6 │
---------------------------------
```

- If the `columns` parameter is specified, ADQM sums values in listed columns.

```
CREATE TABLE table_sum_2
(
    id UInt32,
    value1 Int32,
    value2 Int32,
    value3 Int32
)
ENGINE = SummingMergeTree(value1, value2)
ORDER BY id;

INSERT INTO table_sum_2 VALUES(1, 1, 1, 1), (1, 2, 2, 2), (1, 3, 3, 3);

SELECT * FROM table_sum_2;
```

As a result, values in the `value1` and `value2` columns were summed up, and a value of the `value3` column was selected from an arbitrary row:

```
--id---value1---value2---value3--
│  1 │      6 │      6 │      1 │
---------------------------------
```

Create an AggregatingMergeTree table

In **AggregatingMergeTree** tables, all rows with the same sorting key are replaced by a single row (within one data part) that stores a combination of aggregate function states. These tables can be used for incremental data aggregation, including for creation of aggregated materialized views.

The AggregatingMergeTree engine has no parameters. Below is an example of creating an aggregated materialized view.

1. Create the `table_sales` table of the MergeTree type to store raw sales data:

```
CREATE TABLE table_sales
(
    user_id Int64,
    store_id Int32,
    product_id Int32,
    date Date,
    sum Decimal(20, 2)
)
ENGINE = MergeTree
PARTITION BY date
ORDER BY (store_id, product_id, user_id);
```

2. Create an AggregatingMergeTree materialized view that "follows" the `table_sales` table and stores intermediate states of aggregations:

```
CREATE MATERIALIZED VIEW table_aggregating
(
    store_id Int32,
    product_id Int32,
    date Date,
    s AggregateFunction(sum, Decimal(20, 2)),
    c AggregateFunction(count, Decimal(20, 2))
)
ENGINE = AggregatingMergeTree
PARTITION BY date
ORDER BY (store_id, date) AS
SELECT
    store_id,
    product_id,
    date,
    sumState(sum) AS s,
    countState(sum) AS c
FROM table_sales
GROUP BY
    store_id,
    date,
    product_id;
```

When inserting data into `table_sales`, the `table_aggregating` materialized view will automatically update aggregate states for all changed keys listed in `GROUP BY`.

3. To get aggregated data, execute the `SELECT … GROUP BY …` query for the `table_aggregating` materialized view:

```
SELECT
    store_id,
    product_id,
    date,
    sumMerge(s),
    countMerge(c)
FROM table_aggregating
GROUP BY
    store_id,
    date,
    product_id
LIMIT 1;
```

Create a CollapsingMergeTree table

The **CollapsingMergeTree** table engine allows quick writing constantly changing object states and removes old object states in the background. This can significantly reduce the amount of storage and increase the efficiency of `SELECT` queries.

CollapsingMergeTree asynchronously removes (collapses) pairs of rows that contain the same values in all columns of the sorting key (`ORDER BY`), but different values in the special `sign` column of the Int8 type. This column values specify a row type:

- `1` — means that a row is a "state" row;
- `-1` — means that a row is a "cancel" row.

The `sign` column is passed to the CollapsingMergeTree engine as a parameter. Rows that do not have pairs remain in the table. For more information on the collapsing algorithm, see the CollapsingMergeTree article in the ClickHouse documentation.

An example below shows how the CollapsingMergeTree table engine processes data.

1. Create a CollapsingMergeTree table and specify a column whose values indicate types of rows:

```
CREATE TABLE table_collapsing
(
    id UInt32,
    pageViews UInt8,
    duration UInt8,
    sign Int8
)
ENGINE = CollapsingMergeTree(sign)
ORDER BY id;
```

2. Use two `INSERT` queries to create two different data parts. In the second query, the first value to be inserted cancels the previous state of the object, and the second value defines the current state.

```
INSERT INTO table_collapsing VALUES (1, 7, 100, 1);
INSERT INTO table_collapsing VALUES (1, 7, 100, -1), (1, 8, 150, 1);
```

3. Use the `SELECT` query to make sure that rows corresponding to the obsolete state of the object have been removed from the table, and only the latest state of the object remains. Since ADQM merges inserted data parts at an unknown moment, aggregation is required to get completely "collapsed" data from the CollapsingMergeTree table. If you need to get data without aggregation (for example, to check the presence of rows whose last values satisfy some conditions), you can use the `FINAL` modifier for the `FROM` clause.

```
SELECT * FROM table_collapsing FINAL;
```

The output is:

```
--id---pageViews---duration---sign--
│  1 │         8 │      150 │    1 │
------------------------------------
```

Create a VersionedCollapsingMergeTree table

The **VersionedCollapsingMergeTree** engine has the same purpose as CollapsingMergeTree, but uses a different algorithm for collapsing (deleting) rows. It allows inserting data in any order with multiple threads while CollapsingMergeTree requires strictly sequential data insertion.

The VersionedCollapsingMergeTree engine's parameters are:

- `sign` — name of a column (of the Int8 type) whose values indicate row types (`1` means that a row is a "state" row; `-1` indicates a "cancel" row that copies all fields of the corresponding "state" row except `sign`);
- `version` — name of a column (of the UInt type) that contains versions of object states (each state of an object is identified with a separate number).

An example below shows how the VersionedCollapsingMergeTree table engine processes data.

1. Create a VersionedCollapsingMergeTree table and specify columns whose values indicate a row type ("state" or "cancel") and object state version:

```
CREATE TABLE table_versioned_collapsing
(
    id UInt32,
    pageViews UInt8,
    duration UInt8,
    sign Int8,
    version UInt8
)
ENGINE = VersionedCollapsingMergeTree(sign, version)
ORDER BY id;
```

2. Use two `INSERT` queries to create two different data parts. In the second query, the first value to be inserted cancels the previous state of the object, and the second value defines the current state.

```
INSERT INTO table_versioned_collapsing VALUES (1, 7, 100, 1, 1);
INSERT INTO table_versioned_collapsing VALUES (1, 7, 100, -1, 1), (1, 8, 150, 1, 2);
```

3. Use the `SELECT` query to make sure that rows corresponding to the obsolete state of the object have been removed from the table, and only the latest state of the object remains. Since ADQM merges inserted data parts at an unknown moment, aggregation is required to get completely "collapsed" data from the CollapsingMergeTree table. If you need to get data without aggregation (for example, to check the presence of rows whose last values satisfy some conditions), you can use the `FINAL` modifier for the `FROM` clause.

```
SELECT * FROM table_versioned_collapsing FINAL;
```

The output is:

```
--id---pageViews---duration---sign---version--
│  1 │         8 │      150 │    1 │       2 │
----------------------------------------------
```

Create replicated tables

To create replicated tables, use engines of the MergeTree family with the Replicated prefix.

Meta information related to the replication process is stored in ZooKeeper, so the **Replicated*** engines require the following parameters to build a unique path to each replicated table in ZooKeeper:

- path to the table in ZooKeeper;
- replica name in ZooKeeper (it identifies different replicas of the same table).

Each **Replicated*** engine also has the same optional parameters as the engine to which the given replicated version corresponds. For example, the ReplicatedSummingMergeTree engine, like the SummingMergeTree engine, has the optional `columns` parameter — a list of columns whose values should be summed up.

For an example of creating replicated tables with the ReplicatedMergeTree engine, see the Typical cluster article.

View the list of tables

You can view names of all tables that exist in the specified database in the following ways:

- Use the SHOW TABLES query.

```
SHOW TABLES FROM <database_name>;
```

- Read values from the `name` column of the system.tables system table.

```
SELECT name FROM system.tables WHERE database='<database_name>';
```

Get information on the specified table

Table structure

To obtain the description of table columns, use the DESCRIBE TABLE query:

```
DESCRIBE TABLE [<db_name.]<table_name>;
```

This query returns a list of table columns with their names and data types, as well as the following information on each column:

- type of an expression for a default value (`DEFAULT`, `MATERIALIZED`, `EPHEMERAL`, or `ALIAS`);
- expression for a default value;
- comment;
- compression codec;
- TTL expression.

Example

Query:

```
CREATE TABLE test_table
(
    id Int32,
    description String DEFAULT 'unknown' COMMENT 'This column contains descriptions' CODEC(LZ4),
    date Date COMMENT 'This column contains event dates'
)
ENGINE = MergeTree
ORDER BY id;

DESCRIBE TABLE test_table;
```

Result:

```
--name---------type-----default_type---default_expression---comment-----------------------------codec_expression---
ttl_expression--
│ id          │ Int32  │              │                    │                                   │                  │
│
│ description │ String │ DEFAULT      │ 'unknown'          │ This column contains descriptions │ LZ4              │
│
│ date        │ Date   │              │                    │ This column contains event dates  │                  │
│
------------------------------------------------------------------------------------------------------------------------
------
```

Table metadata

To view table metadata, use the system.tables system table. A query syntax:

```
SELECT * FROM system.tables WHERE name = '<table_name>' [FORMAT <format_name>];
```

Example

Query:

```
SELECT * FROM system.tables WHERE name = 'test_table' FORMAT Vertical;
```

Result:

```
Row 1:
──────
database:                      default
name:                          test_table
uuid:                          7239d59f-9144-4600-b47f-c9dd02330242
engine:                        MergeTree
is_temporary:                  0
data_paths:                    ['/var/lib/clickhouse/store/723/7239d59f-9144-4600-b47f-c9dd02330242/']
metadata_path:                 /var/lib/clickhouse/store/463/4632cdc5-46bc-4d4e-ad36-27b8d101e2e1/test_table.sql
metadata_modification_time:    2022-12-21 07:04:25
dependencies_database:         []
dependencies_table:            []
create_table_query:            CREATE TABLE default.test_table (`id` Int32, `v` String, `d` Date, `v2` String CODEC(ZSTD(10)))
ENGINE = MergeTree PARTITION BY d ORDER BY id SETTINGS index_granularity = 8192 COMMENT 'This is a test table'
engine_full:                   MergeTree PARTITION BY d ORDER BY id SETTINGS index_granularity = 8192
as_select:
partition_key:                 d
sorting_key:                   id
primary_key:                   id
sampling_key:
storage_policy:                default
total_rows:                    1000000
total_bytes:                   56688997
lifetime_rows:                 NULL
lifetime_bytes:                NULL
comment:                       This is a test table
has_own_data:                  1
loading_dependencies_database: []
loading_dependencies_table:    []
loading_dependent_database:    []
loading_dependent_table:       []
```

Insert data into a table

To add data to a table, use the INSERT INTO query. After the `VALUES` clause, list column values (or data tuples, if you add more than one row) separated by commas:

```
INSERT INTO [<db_name>.]<table_name> [(<column_name1>, <column_name2>, <column_name3>)] VALUES (v11, v12, v13), (v21, v22, v23),
...;
```

Example

1. Create a table with three columns:

```
CREATE TABLE test_table
(
    a Int32,
    b String,
    c Int32
)
ENGINE = MergeTree
ORDER BY a;
```

2. Insert a couple of rows with the specified values for all columns of the table:

```
INSERT INTO test_table VALUES (1, 'a', 1), (2, 'b', 2);
```

Result:

```
--a---b---c--
│ 1 │ a │ 1 │
│ 2 │ b │ 2 │
-------------
```

3. Insert two rows with data values specified for the `a` and `b` columns only:

```
INSERT INTO test_table (a, b) VALUES (3, 'c'), (4, 'd');
```

As a result, data values from the query are inserted into the `a` and `b` columns, and the `c` column is filled with zeros since it does not have a default value specified:

```
--a---b---c--
│ 3 │ c │ 0 │
│ 4 │ d │ 0 │
-------------
```

The `INSERT INTO` query also allows you to copy data from one table to another one. To do this, use the combination of the `INSERT INTO` and `SELECT` clauses:

```
INSERT INTO [<db_name>.]<table_name> [(<column_name1>, <column_name2>, <column_name3>)] SELECT ... ;
```

Example

An example below copies the first three rows from the `source_table` table into the `new_table` table with the same structure. The `source_table` table contains the following data:

```
--a---b----c--
│ 1 │ a │  1 │
│ 1 │ a │  1 │
│ 2 │ b │  2 │
│ 3 │ c │  0 │
│ 6 │ e │ 12 │
│ 9 │ a │  9 │
--------------
```

1. Create the `new_table` table with three columns:

```
CREATE TABLE new_table
(
    d Int32,
    f String,
    g Int32
)
ENGINE = MergeTree
ORDER BY d;
```

2. Insert three rows from the `source_table` table into the new table with the following query:

```
INSERT INTO new_table SELECT * FROM source_table LIMIT 3;
```

The query execution result:

```
--d---f---g--
│ 1 │ a │ 1 │
│ 1 │ a │ 1 │
│ 2 │ b │ 2 │
-------------
```

Read data from a table

To retrieve data from a table, use the `SELECT` query. This query supports various modifiers and clauses that allow you to customize data selection. For example, you can filter/sort/group data that the query should return, limit the number of rows in the selection, set the output format, etc. For more information about all clauses available for `SELECT` queries, see the SELECT Query section of the ClickHouse documentation.

Select all data

To select all data from a table, use the asterisk symbol (`*`):

```
SELECT * FROM [<db_name.]<table_name>;
```

For example, the following query reads data from all columns of the `test_table` table:

```
SELECT * FROM test_table;
```

The output:

```
--a---b----c--
│ 1 │ a │  1 │
│ 1 │ a │  1 │
│ 2 │ b │  2 │
│ 3 │ c │  0 │
│ 6 │ e │ 12 │
│ 9 │ a │  9 │
--------------
```

Read data from the specified columns

To read data from individual columns, list column names after the `SELECT` clause:

```
SELECT <column_name1>[, ...] FROM [<db_name.]<table_name>;
```

For example, the following query reads data from the `b` and `c` columns of the `test_table` table:

```
SELECT b, c FROM test_table;
```

The output:

```
--b----c--
│ a │  1 │
│ a │  1 │
│ b │  2 │
│ c │  0 │
│ e │ 12 │
│ a │  9 │
----------
```

Select the specified number of rows

To read the specified number of data rows from a table, use the `LIMIT` clause in a `SELECT` query:

```
SELECT * FROM [<db_name>.]<table_name> LIMIT [n, ]m;
```

This query selects the first `m` rows from the table after skipping the first `n` rows.

For example, the following query reads the first three rows from the `test_table` table:

```
SELECT * FROM test_table LIMIT 3;
```

The output:

```
--a---b----c--
│ 1 │ a │  1 │
│ 1 │ a │  1 │
│ 2 │ b │  2 │
--------------
```

Read data that satisfy the specified condition

Use the `WHERE` clause to set a condition for filtering data to be read:

```
SELECT <column_name1>[, ...] FROM [<db_name.]<table_name> WHERE <filter_expr>;
```

For example, the following query returns values of the `c` column if they are greater than `1` but less than `10`:

```
SELECT c FROM test_table WHERE (c > 1) AND (c < 10);
```

The output:

```
---c--
│  2 │
│  9 │
------
```

Sort the results

Use the `ORDER BY` clause to sort the results of a `SELECT` query:

```
SELECT <column_name1>[, ...] FROM [<db_name.]<table_name> ORDER BY <expr_list>;
```

For example, the following query sorts data by values of the `c` column in ascending order:

```
SELECT * FROM test_table ORDER BY c;
```

The output:

```
--a---b----c--
│ 3 │ c │  0 │
│ 1 │ a │  1 │
│ 1 │ a │  1 │
│ 2 │ b │  2 │
│ 9 │ a │  9 │
│ 6 │ e │ 12 │
--------------
```

Modify a table

Table engines of the MergeTree family, as well as Merge and Distributed engines support `ALTER TABLE` queries that allow changing table structure, settings or data. For example, below is syntax of a query for modifying table columns:

```
ALTER TABLE [<db_name>].<table_name> [ON CLUSTER <cluster_name>] ADD|DROP|CLEAR|COMMENT|MODIFY COLUMN ...;
```

Manipulate columns

The following clauses in `ALTER TABLE` queries allow you to perform various actions on table columns.

Clause: ADD COLUMN

Description: Adds a new column to a table.

`AFTER | FIRST` clauses are used to indicate where to insert a new column — after the specified `after_column_name` column or at the beginning of the table. If none of these clauses is used, the column is added to the end of the table

Syntax:

```
ALTER TABLE [<db_name.]<table_name> [ON CLUSTER
<cluster_name>]
    ADD COLUMN [IF NOT EXISTS] <new_column_name>
[<data_type>] [<default_expr>]
[CODEC(<codec_name>)] [TTL <expr>] [AFTER
<after_column_name> | FIRST];
```

Clause: DROP COLUMN

Description: Deletes the specified column

Syntax:

```
ALTER TABLE [<db_name.]<table_name> [ON CLUSTER
<cluster_name>]
    DROP COLUMN [IF EXISTS] <column_name>;
```

Clause: RENAME COLUMN

Description: Renames an existing column.

Columns specified in the table's key expression (`PRIMARY KEY` or `ORDER BY`) cannot be renamed

Syntax:

```
ALTER TABLE [<db_name.]<table_name> [ON CLUSTER
<cluster_name>]
    RENAME COLUMN [IF EXISTS] <old_name> to
<new_name>;
```

Clause: CLEAR COLUMN

Description: Resets column values for the specified partition

Syntax:

```
ALTER TABLE [<db_name.]<table_name> [ON CLUSTER
<cluster_name>]
    CLEAR COLUMN [IF EXISTS] <column_name> IN
PARTITION <partition_name>;
```

Clause: COMMENT COLUMN

Description: Adds a comment to a column.

If a comment already exists, it is replaced with a new one, since a column can contain one comment only

Syntax:

```
ALTER TABLE [<db_name.]<table_name> [ON CLUSTER
<cluster_name>]
    COMMENT COLUMN [IF EXISTS] <column_name> 'Text
comment';
```

Clause: MODIFY COLUMN

Description: Changes column settings (data type, expression for default value, data compression codec, TTL, column position in a table)

Syntax:

```
ALTER TABLE [<db_name.]<table_name> [ON CLUSTER
<cluster_name>]
    MODIFY COLUMN [IF EXISTS] <column_name>
[<data_type>] [<default_expr>]
[CODEC(<codec_name>)] [TTL <expr>] [AFTER
<after_column_name> | FIRST];
```

Manipulate key expressions

Tables of the MergeTree family (including replicated tables) support a query to change the sorting key:

```
ALTER TABLE [<db_name.]<table_name> [ON CLUSTER <cluster_name>] MODIFY ORDER BY <new_expression>;
```

Manipulate data skipping indexes

You can use the following queries to add or remove indexes:

```
ALTER TABLE [<db_name.]<table_name> [ON CLUSTER <cluster_name>] ADD INDEX <index_name> <expression> TYPE <index_type>
GRANULARITY  <granularity_value> [FIRST|AFTER name];
ALTER TABLE [<db_name.]<table_name> [ON CLUSTER <cluster_name>] DROP INDEX <index_name>;
```

Only table engines of the MergeTree family support these operations.

Manipulate constraints

Use the following queries to add or delete constraints:

```
ALTER TABLE [<db_name.]<table_name> [ON CLUSTER <cluster_name>] ADD CONSTRAINT <constraint_name> CHECK <expression>;
ALTER TABLE [<db_name.]<table_name> [ON CLUSTER <cluster_name>] DROP CONSTRAINT <constraint_name>;
```

Manipulate partitions and data parts

To work with partitions and data parts, you can use the following clauses in `ALTER TABLE` queries.

Clause: DETACH PARTITION|PART

Description: Moves a partition/part to the *detached* directory (the server will not "see" the partition/part until the `ATTACH PARTITION|PART` query is executed).

The query is replicated — it moves data to the *detached* directory on all replicas

Syntax:

```
ALTER TABLE [<db_name.]<table_name> [ON CLUSTER
<cluster_name>]
    DETACH PARTITION|PART <partition_expr>;
```

Clause: DROP PARTITION|PART

Description: Deletes a partition/part.

The query is replicated — it deletes data on all replicas

Syntax:

```
ALTER TABLE [<db_name.]<table_name> [ON CLUSTER
<cluster_name>]
    DROP PARTITION|PART <partition_expr>;
```

Clause: DROP DETACHED PARTITION|PART

Description: Deletes the specified partition/part from the *detached* directory

Syntax:

```
ALTER TABLE [<db_name.]<table_name> [ON CLUSTER
<cluster_name>]
    DROP DETACHED PARTITION|PART <partition_expr>;
```

Clause: ATTACH PARTITION|PART

Description: Adds a partition/part from the *detached* directory to the table.

This query is replicated

Syntax:

```
ALTER TABLE [<db_name.]<table_name> [ON CLUSTER
<cluster_name>]
    ATTACH PARTITION|PART <partition_expr>;
```

Clause: ATTACH PARTITION FROM

Description: Copies a partition from one table to another one.

Both tables should have the same:

- structures;
- partitioning keys, sorting keys, and primary keys;
- data storage policies (a disk where the partition is stored should be available for both tables).

Syntax:

```
ALTER TABLE <dest_table_name> [ON CLUSTER
<cluster_name>]
    ATTACH PARTITION <partition_expr> FROM
<source_table_name>;
```

Clause: REPLACE PARTITION FROM

Description: Copies a partition from one table to another one and replaces existing partition in the destination table (data is not deleted from the source table).

Both tables should have the same:

- structures;
- partitioning keys, sorting keys, and primary keys;
- data storage policies (a disk where the partition is stored should be available for both tables).

Syntax:

```
ALTER TABLE <dest_table_name> [ON CLUSTER
<cluster_name>]
    REPLACE PARTITION <partition_expr> FROM
<source_table_name>;
```

Clause: MOVE PARTITION TO TABLE

Description: Moves a partition from one table to another one and deletes data from the source table.

Both tables should have the same:

- structures;
- partitioning keys, sorting keys, and primary keys;
- data storage policies (a disk where the partition is stored should be available for both tables);
- engine families (replicated or non-replicated).

Syntax:

```
ALTER TABLE <source_table_name> [ON CLUSTER
<cluster_name>]
    MOVE PARTITION <partition_expr> TO TABLE
<dest_table_name>;
```

Clause: CLEAR COLUMN IN PARTITION

Description: Removes all values in a column for the specified partition.

If a default value is defined for a column, that default value will be set

Syntax:

```
ALTER TABLE [<db_name.]<table_name> [ON CLUSTER
<cluster_name>]
    CLEAR COLUMN <column_name> IN PARTITION
<partition_expr>;
```

Clause: CLEAR INDEX IN PARTITION

Description: Resets a secondary index in the specified partition

Syntax:

```
ALTER TABLE [<db_name.]<table_name> [ON CLUSTER
<cluster_name>]
    CLEAR INDEX <index_name> IN PARTITION
<partition_expr>;
```

Clause: FREEZE PARTITION

Description: Creates a backup of the specified partition.

If you do not use the `PARTITION` clause in the query, all partitions will be backed up.

The query is not replicated — the backup is created on a local server only.

The query copies only table data, not metadata. To back up table metadata, copy the */var/lib/clickhouse/metadata/database/table.sql* file

Syntax:

```
ALTER TABLE [<db_name.]<table_name> [ON CLUSTER
<cluster_name>]
    FREEZE [PARTITION <partition_expr>] [WITH NAME
'backup_name'];
```

Clause: UNFREEZE PARTITION

Description: Removes a backup of the specified partition from the disk.

If you omit the `PARTITION` clause, the query removes the backup of all partitions at once

Syntax:

```
ALTER TABLE [<db_name.]<table_name> [ON CLUSTER
<cluster_name>]
    UNFREEZE [PARTITION <part_expr>] WITH NAME
'backup_name';
```

Clause: FETCH PARTITION|PART

Description: Downloads a partition/part from another server.

This query works for replicated tables only

Syntax:

```
ALTER TABLE [<db_name.]<table_name> [ON CLUSTER
<cluster_name>]
    FETCH PARTITION|PART <partition_expr> FROM
'path-in-zookeeper';
```

Clause: MOVE PARTITION|PART

Description: Moves partition/part to another disk or volume.

The query is not replicated as different replicas can have different storage policy configurations

Syntax:

```
ALTER TABLE [<db_name.]<table_name> [ON CLUSTER
<cluster_name>]
    MOVE PARTITION|PART <partition_expr> TO
DISK|VOLUME 'disk_name';
```

Clause: UPDATE IN PARTITION

Description: Updates partition data that matches the specified condition

Syntax:

```
ALTER TABLE [<db_name.]<table_name> [ON CLUSTER
<cluster_name>]
    UPDATE <column_name1> = <expr1> [, ...] [IN
PARTITION <partition_expr>] WHERE <filter_expr>;
```

Clause: DELETE IN PARTITION

Description: Deletes partition data that matches the specified condition

Syntax:

```
ALTER TABLE [<db_name.]<table_name> [ON CLUSTER
<cluster_name>]
    DELETE [IN PARTITION <partition_expr>] WHERE
<filter_expr>;
```

Specify partition in a query

To specify a partition in an `ALTER TABLE … PARTITION` query, you can use:

- Partition name in single quotes (quotes can be omitted if the partitioning expression is of the Int* type). For example, `ALTER TABLE test_table DROP PARTITION '2022-12-26';`.
- A tuple of constants or expressions that have the same types as constants or expressions in a tuple of the table's partitioning key. If the partitioning key consists of a single element, the expression should be wrapped in the `tuple(…)` function. For example, `ALTER TABLE test_table DROP PARTITION tuple(toYYYYMMDD(toDate('2022-12-26')));`.
- String identifier of the partition specified for the `PARTITION ID` expression in single quotes. For example, `ALTER TABLE test_table DROP PARTITION ID '20221226';`.

You can view names and identifiers of partitions in the `partition` and `partition_id` columns of the system.parts system table:

```
SELECT partition, partition_id FROM system.parts WHERE table = '<table_name>' and active = 1;
```

To set a name of a data part in the `ATTACH PART` and `DROP DETACHED PART` queries, use the value from the `name` column of the system.detached_parts system table in single quotes.

The Row update & deletion article describes how to update and delete data existing in a table.

Detach and attach a table

You can detach a table with the `DETACH TABLE` query:

```
DETACH TABLE [IF EXISTS] [<db_name.]<table_name> [ON CLUSTER <cluster_name>] [PERMANENTLY];
```

When this query is executed, the table becomes "invisible" to the server, but its data and metadata are not deleted. If the table was not detached permanently (the `PERMANENTLY` clause was not used in the query), the next time the server starts, it will read the table metadata and "see" the table again. If a table has been detached permanently, the server will not automatically re-enable it.

You can re-attach a table with the `ATTACH` query regardless of whether the table was detached permanently or not:

```
ATTACH TABLE [IF NOT EXISTS] [<db_name.]<table_name> [ON CLUSTER <cluster_name>];
```

Delete a table

To delete a table, use the `DROP TABLE` query:

```
DROP TABLE [IF EXISTS] [<db_name.]<table_name> [ON CLUSTER <cluster_name>];
```

Row update & deletion

Elena Dvoryadkina

Contents

To Table of Contents

Update data
Delete data
  ALTER TABLE...DELETE
  DELETE FROM

Although ADQM is not designed to update and delete data, it supports the ability to execute **mutations** — the `ALTER TABLE … UPDATE/DELETE` operations to update/delete existing data. The scope of mutations is changes that affect a large number of data rows in a table. These operations are heavyweight and not suitable for frequent use (best to avoid them if possible).

Once a query for data modification/deletion is submitted, it cannot be rolled back, and mutation execution will continue even after the ClickHouse server restart. To monitor the progress of mutations, use the system.mutations table. If a mutation is stuck and is not executing correctly for any reason, you can terminate it with the KILL MUTATION query (it will stop further processing, but all changes already applied to the data will remain).

Update data

To update data in a table, use the `ALTER TABLE…UPDATE` query:

```
ALTER TABLE [<db_name>.]<table_name>
[ON CLUSTER <cluster_name>]
UPDATE <column_name> = <expression>[, ...]
[IN PARTITION <partition_expr>]
WHERE <filter_expression>;
```

In this query:

- `[<db_name>.]<table_name>` — name of a table to be updated. If no database name is specified, the table is searched in the current database.
- `[ON CLUSTER <cluster_name>]` — name of a cluster to run the query on all servers of that cluster. If the `ON CLUSTER` clause is not used, the query affects only the server where it is executed.
- `<column_name> = <expression>` — command to update data values in a column. Values of the `<column_name>` column are replaced by the `<expression>` expression value in each row for which the `<filter_expression>` expression returns a non-null value.

  The `<expression>` expression value should be of the same data type as the `<column_name>` column, or it should be cast to the same type via the `CAST` operator.

  A single query can include multiple `<column_name> = <expression>` commands separated by commas. Modifying values of multiple columns with a single query can be more efficient than running multiple queries (one query for updating each column).

  Updating columns that are used for calculating the primary key or sorting key is not supported.
- `[IN PARTITION <partition_expr>]` — partition that contains data to be updated. The `IN PARTITION` clause helps speed up `UPDATE` operations when you need to perform targeted updates for a table divided into many partitions.
- `<filter_expression>` — expression that returns a value of the UInt8 type for each row of the table to determine in which rows the data should be updated.

Example

Initial table:

```
--a---b----c---------date--
│ 1 │ a │ 10 │ 2022-12-10 │
│ 2 │ b │ 20 │ 2022-12-11 │
│ 3 │ c │ 30 │ 2022-12-12 │
│ 4 │ d │ 40 │ 2022-12-13 │
│ 5 │ e │ 50 │ 2022-12-14 │
│ 6 │ f │ 60 │ 2022-12-15 │
---------------------------
```

The following query replaces values of the `b` and `c` columns in rows where the `date` column's value meets the specified condition:

```
ALTER TABLE test_table UPDATE b = repeat(b, 3), c = c/10 WHERE date > '2022-12-13';
```

The result is:

```
--a---b------c---------date--
│ 1 │ a   │ 10 │ 2022-12-10 │
│ 2 │ b   │ 20 │ 2022-12-11 │
│ 3 │ c   │ 30 │ 2022-12-12 │
│ 4 │ d   │ 40 │ 2022-12-13 │
│ 5 │ eee │  5 │ 2022-12-14 │
│ 6 │ fff │  6 │ 2022-12-15 │
-----------------------------
```

For replicated tables, `ALTER TABLE…UPDATE` queries are executed asynchronously. If necessary, you can use the mutations_sync setting to synchronize data updates on replicas.

Delete data

To delete data rows from a table, you can use the following queries:

- `ALTER TABLE…DELETE` — available for all table engines.
- `DELETE FROM` — available for table engines of the MergeTree family only (starting with ADQM 22.8).

In most cases, `DELETE FROM` is faster than `ALTER TABLE…DELETE` (this operation requires data to be merged before it is deleted). `DELETE FROM` is a lightweight operation that is effective for deleting a small number of data rows. For some scenarios (for example, for infrequent bulk deletes), `ALTER TABLE…DELETE` may be more suitable.

ALTER TABLE...DELETE

Query syntax:

```
ALTER TABLE [<db_name>.]<table_name>
[ON CLUSTER <cluster_name>]
DELETE [IN PARTITION <partition_expr>]
WHERE <filter_expression>;
```

The query removes data rows for which the `<filter_expression>` expression of the UInt8 type returns a non-zero value. To remove rows only from an individual partition, specify the partition using the `IN PARTITION` expression.

Example

Initial table:

```
--a---b------c---------date--
│ 1 │ a   │ 10 │ 2022-12-10 │
│ 2 │ b   │ 20 │ 2022-12-11 │
│ 3 │ c   │ 30 │ 2022-12-12 │
│ 4 │ d   │ 40 │ 2022-12-13 │
│ 5 │ eee │  5 │ 2022-12-14 │
│ 6 │ fff │  6 │ 2022-12-15 │
-----------------------------
```

The following query removes rows in which values of the `date` and `a` columns satisfy the specified conditions:

```
ALTER TABLE test_table DELETE WHERE date < '2022-12-13' AND a = 2;
```

The result is:

```
--a---b------c---------date--
│ 1 │ a   │ 10 │ 2022-12-10 │
│ 3 │ c   │ 30 │ 2022-12-12 │
│ 4 │ d   │ 40 │ 2022-12-13 │
│ 5 │ eee │  5 │ 2022-12-14 │
│ 6 │ fff │  6 │ 2022-12-15 │
-----------------------------
```

For replicated tables, `ALTER TABLE…DELETE` queries are asynchronous. If you need to synchronize the delete operation on replicas, change the mutations_sync parameter value.

DELETE FROM

Query syntax:

```
DELETE FROM [<db_name>.]<table_name> WHERE <filter_expression>;
```

In the `<table_name>` table, the query removes rows for which the `<filter_expression>` expression of the UInt8 type returns a non-zero value. Rows to be deleted are immediately marked as deleted and automatically excluded from all subsequent requests. Data deletion is performed asynchronously in the background (the mutations_sync setting controls the synchronicity of query processing).

Example

Initial table:

```
--a---b-------c---------date--
│ 1 │ a    │ 10 │ 2022-12-10 │
│ 3 │ c    │ 30 │ 2022-12-12 │
│ 4 │ d    │ 40 │ 2022-12-13 │
│ 5 │ eee  │  5 │ 2022-12-14 │
│ 6 │ fff  │  6 │ 2022-12-15 │
│ 7 │ aabb │ 10 │ 2022-12-16 │
│ 8 │ bbaa │ 20 │ 2022-12-17 │
------------------------------
```

The following query removes all rows from the `test_table` table where the `b` column value includes the `aa` substring:

```
DELETE FROM test_table WHERE b LIKE '%aa%';
```

The result is:

```
--a---b------c---------date--
│ 1 │ a   │ 10 │ 2022-12-10 │
│ 3 │ c   │ 30 │ 2022-12-12 │
│ 4 │ d   │ 40 │ 2022-12-13 │
│ 5 │ eee │  5 │ 2022-12-14 │
│ 6 │ fff │  6 │ 2022-12-15 │
-----------------------------
```

Data compression

Elena Dvoryadkina

Contents

To Table of Contents

- Data compression codecs
  - General-purpose codecs
  - Specialized codecs
- Apply compression codecs
  - At the server level
  - At the column level
- General recommendations
- Example

Data compression plays an important role in achieving high performance for ADQM. For example, you can use compression to reduce the amount of data to be read when executing a query and speed up this query — see the example in the Improve query performance article.

ADQM supports data compression for the following table engines:

- MergeTree — you can set a default compression algorithm for tables, as well as specify a compression method (or a combination of them) for an individual column when creating/modifying a table (see Apply compression codecs below).
- Log — uses the LZ4 compression method by default and supports column-level compression.
- Set, Join — support default compression only.

Data compression codecs

A **codec** defines a compression method applied to ADQM data. The following data compression codecs are available in ADQM:

- General-purpose codecs that allow you to find the optimal balance between the consumption of disk space and CPU resources.
- Specialized codecs for specific data types. Some of these codecs do not compress data themselves but preprocess it (taking into account data characteristics) for general-purpose codecs, which in turn compress prepared data more efficiently than unprepared data.

General-purpose codecs

Codec	Compression algorithm	Compression level
NONE	No compression	—
LZ4	The LZ4 lossless compression algorithm. Provides both high speed and good compression ratio. Used by default	—
LZ4HC[(level)]	The LZ4 HC (high compression) algorithm with a configurable compression level. In comparison with LZ4, it compresses data better, but slower (decompression is still fast)	Possible compression levels — `[1, 12]`, recommended level range — `[4, 9]`. Default compression level (if the level is not specified, or it is set to a value less than or equal to `0`) — `9`
ZSTD[(level)]	The ZSTD compression algorithm. Provides both speed and a high compression ratio that can be tuned. ZSTD used without a specialized codec often outperforms other compression methods or at least competitive	Possible compression levels — `[1, 22]`. Default compression level — `1`. Compression levels above `3` rarely result in significant improvement, but high compression levels (which mean better compression but higher CPU usage) can be useful for asymmetric scenarios — such as "compress once, decompress repeatedly". In any case, testing is recommended
ZSTD_QAT[(level)]	The ZSTD compression algorithm with a configurable compression level, implemented by Intel QATlib and Intel QAT ZSTD Plugin. Limitations: - The ZSTD_QAT codec is disabled by default and can only be used after you enable the parameter: `SET enable_zstd_qat_codec = 1`. - For compression, ZSTD_QAT tries to use an Intel QuickAssist Technology (QAT) offloading device. If no such device is found, compression is performed in software. - Decompression is always performed in software.	Possible compression levels — `[1, 12]`, recommended level range — `[6, 12]`. Default compression level — `1`
DEFLATE_QPL	The Deflate compression algorithm implemented by Intel Query Processing Library (Intel QPL). Limitations: - The DEFLATE_QPL codec is disabled by default and can only be used after you enable the parameter: `SET enable_deflate_qpl_codec = 1`. - DEFLATE_QPL works best if the system has an Intel In-Memory Analytics Accelerator (IAA) offloading device — see Accelerator Configuration and Benchmark with DEFLATE_QPL for more details. Technical support for this codec in ADQM is limited	—

Specialized codecs

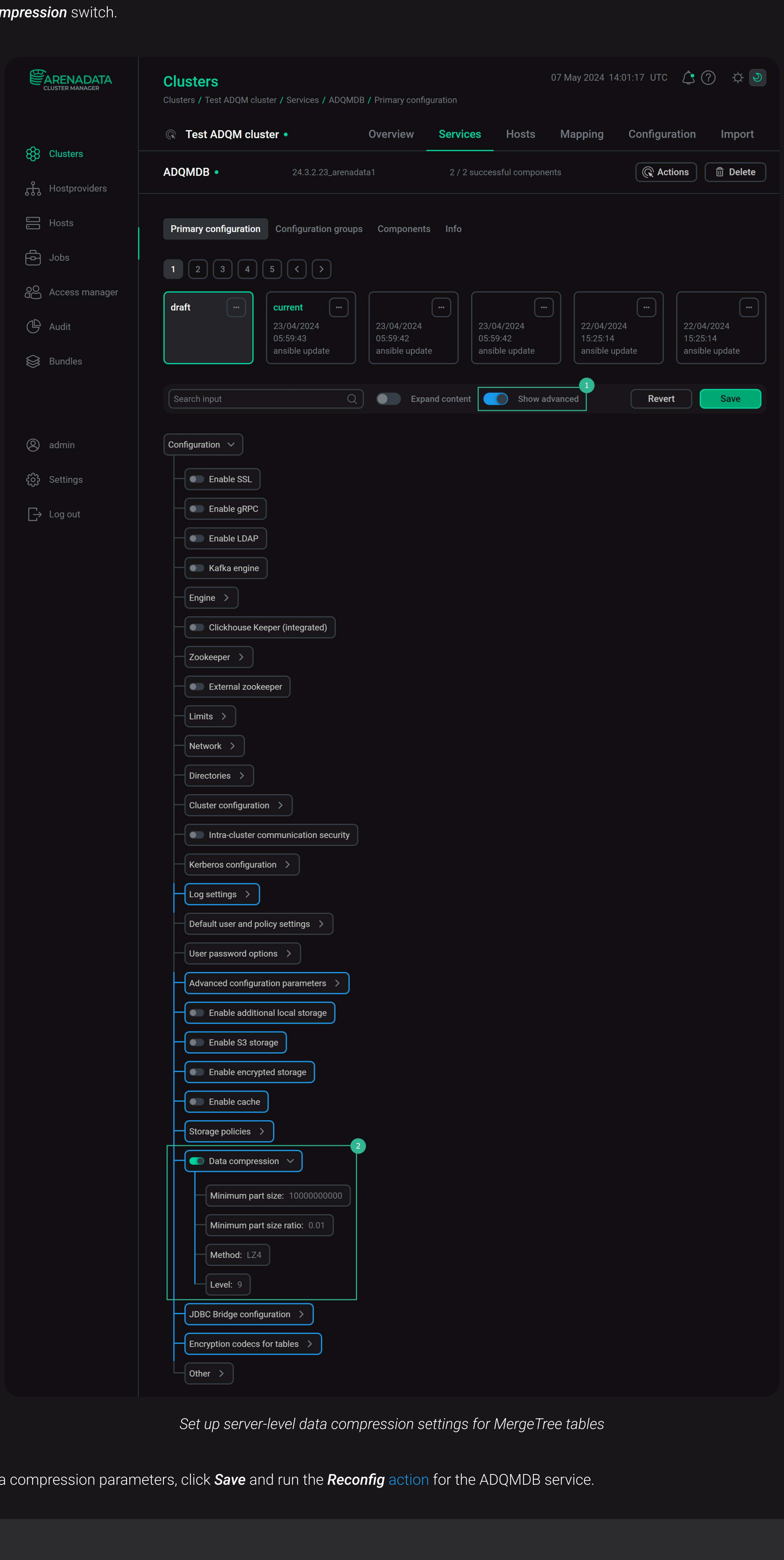
Codec	Description	Parameters
Delta(delta_bytes)	Replaces raw values with the difference (delta) of two neighboring values (except for the first value that remains unchanged). Works well with monotonically increasing data. Delta is a codec for data preparation, and it cannot be used stand-alone (in other words, it should be followed by some compression codec)	`delta_bytes` is the limit (in bytes) for storing delta values, that is, it is the maximum size of raw values. Possible values for `delta_bytes` are `1`, `2`, `4`, `8`. The default value is `sizeof(type)` if equal to `1`, `2`, `4`, or `8`. In all other cases, it is `1`
DoubleDelta(bytes_size)	Calculates the difference between neighboring delta values and stores it in compact binary form. An optimal compression ratio can be achieved for a monotonic sequence with a constant stride (such as time series data). Can be used with any fixed-length data type. Implements the algorithm used in Gorilla TSDB, extending it to support 64-bit data types. Uses 1 extra bit for 32-bit deltas: 5-bit prefixes instead of 4-bit prefixes. For more information, see Compressing Time Stamps in the ***Gorilla: A Fast, Scalable, In-Memory Time Series Database*** article. DoubleDelta is a codec for data preparation and it cannot be used stand-alone (in other words, it should be followed by some compression codec)	Possible values for `bytes_size` are `1`, `2`, `4`, `8`. The default value is `sizeof(type)` if equal to `1`, `2`, `4`, or `8`. In all other cases, it is `1`
GCD	Calculates the greatest common denominator (GCD) of the values in a column, then divides each value by the GCD. Can be used with integer, decimal, and date/time columns. The codec is well suited for columns whose values change (increase or decrease) in multiples of GCD — for example, `24`, `28`, `16`, `24`, `8`, `24` (`GCD = 4`). GCD is a codec for data preparation, and it cannot be used stand-alone (in other words, it should be followed by some compression codec)	—
Gorilla(bytes_size)	Calculates XOR between the current and previous floating point value and writes it in compact binary form. The smaller the difference between consecutive values (that is, the slower the values in a series change), the better the compression ratio. Implements the algorithm used in Gorilla TSDB, extending it to support 64-bit types. For more information, see Compressing values in the ***Gorilla: A Fast, Scalable, In-Memory Time Series Database*** article	Possible values for `bytes_size` are `1`, `2`, `4`, `8`. The default value is `sizeof(type)` if equal to `1`, `2`, `4`, or `8`. In all other cases, it is `1`
FPC(level, float_size)	Repeatedly predicts the next floating point value in the sequence using the better of two predictors, then calculates the XOR of the actual value with the predicted value, and compresses the result with a leading zero. Similar to Gorilla, FPC efficiently stores series of slowly changing floating point values. For 64-bit values, FPC is faster than Gorilla; for 32-bit values, the estimate may vary. For a detailed description of the algorithm, refer to High Throughput Compression of Double-Precision Floating-Point Data	Possible values for the `level` parameter — `[1, 28]`. The default value is `12`. Possible values for `float_size` — `4`, `8`. The default value is `sizeof(type)` if the type is Float. In all other cases, it is `4`
T64	Crops unused high bits of integer values (including Enum, Date, and DateTime). At each step of the algorithm, the codec places a block of 64 values into a 64x64 bit matrix, transposes it, crops the unused bits of values, and returns the rest as a sequence. Unused bits are the bits that do not differ between maximum and minimum values in the whole data part for which the compression is used	—

The DoubleDelta and Gorilla codecs are used in Gorilla TSDB as components of its compression algorithm. The Gorilla approach is effective in scenarios when data is a sequence of slowly changing values with their timestamps. Timestamps are effectively compressed by the DoubleDelta codec, and values are effectively compressed by the Gorilla codec.

Apply compression codecs

At the server level

By default, ADQM applies the compression method defined in the server configuration to columns of MergeTree tables. You can change the default compression settings in the ***Data compression*** section on the configuration page of the ADQMDB service. This section becomes visible when the ***Show advanced*** option is enabled. To access and apply configuration parameters, turn on the ***Data compression*** switch.



Minimum part size: 10000000000

Minimum part size ratio: 0.01

Method: LZ4

Level: 9

Set up server-level data compression settings for MergeTree tables

After you have specified the data compression parameters, click ***Save*** and run the ***Reconfig*** action for the ADQMDB service.

> **NOTE**
>
> Changing the default compression settings does not affect existing tables.

At the column level

You can also set up the compression method for each column separately in the CREATE TABLE query using the `CODEC` clause:

```
CREATE TABLE <table_name>
(   <column_name1> <data_type1> CODEC(<compression_codec1>) [...],
    <column_name2> <data_type2> CODEC(<compression_codec2>) [...],
    ...)
ENGINE = MergeTree()
...;
```

If the Default codec is assigned to a column (`CODEC(Default)`) or no codec is specified, the default compression algorithm set at the server level is used.

Codecs can be combined sequentially — for example, when using a specialized codec to prepare data for more efficient compression by a general-purpose codec (`CODEC(Delta, ZSTD)`).

For an existing table, you can change the compression codec of a column using the following query:

```
ALTER TABLE <table_name> MODIFY COLUMN <column_name> CODEC(<new_compression_codec>);
```

General recommendations

There are some recommendations on how to get the maximum performance when using data compression:

- Choose the most suitable compression algorithm for a specific task, taking into account the type and characteristics of your data (if known).
- When specifying a compression level, remember that the bigger that value is — the more (but slower) the compression is. In fact, data compression allows you to improve I/O performance and reduce data size at the expense of CPU. Try to find the optimal settings — to avoid long compression times or slow scan rates.
- The final performance of compressed append-optimized tables depends on many factors, such as hardware, query tuning parameters, etc. You should perform comparison testing to determine whether data compression is useful in your environment.

> **TIP**
>
> It is recommended to always balance between the available CPU resources and the required compression speed.

Example

Create a MergeTree table with Int64 and Float64 columns using the LZ4 and ZSTD data compression codecs in combination with various specialized codecs for preprocessing data:

```
CREATE TABLE compression_test (
    n Int32,
    i64 Int64 DEFAULT n CODEC(NONE),
    i64_lz4 Int64 DEFAULT n CODEC(LZ4),
    i64_delta_lz4 Int64 DEFAULT n CODEC(Delta, LZ4),
    i64_doubledelta_lz4 Int64 DEFAULT n CODEC(DoubleDelta, LZ4),
    i64_t64_lz4 Int64 DEFAULT n CODEC(T64, LZ4),
    i64_zstd Int64 DEFAULT n CODEC(ZSTD),
    i64_delta_zstd Int64 DEFAULT n CODEC(Delta, ZSTD),
    i64_doubledelta_zstd Int64 DEFAULT n CODEC(DoubleDelta, ZSTD),
    i64_t64_zstd Int64 DEFAULT n CODEC(T64, ZSTD),
    f64 Float64 DEFAULT n CODEC(NONE),
    f64_lz4 Float64 DEFAULT n CODEC(LZ4),
    f64_delta_lz4 Float64 DEFAULT n CODEC(Delta, LZ4),
    f64_doubledelta_lz4 Float64 DEFAULT n CODEC(DoubleDelta, LZ4),
    f64_gorilla_lz4 Float64 DEFAULT n CODEC(Gorilla, LZ4),
    f64_fpc_lz4 Float64 DEFAULT n CODEC(FPC, LZ4),
    f64_zstd Float64 DEFAULT n CODEC(ZSTD),
    f64_delta_zstd Float64 DEFAULT n CODEC(Delta, ZSTD),
    f64_doubledelta_zstd Float64 DEFAULT n CODEC(DoubleDelta, ZSTD),
    f64_gorilla_zstd Float64 DEFAULT n CODEC(Gorilla, ZSTD),
    f64_fpc_zstd Float64 DEFAULT n CODEC(FPC, ZSTD))
Engine = MergeTree()
ORDER BY tuple();
```

Generate test data for the table (1 million rows) so that each column contains a monotonic sequence of values with random increment:

```
INSERT INTO compression_test (n) SELECT number*1000+(rand()%100) FROM numbers(1000000);
```

Use the system.columns system table to view the size of compressed and uncompressed data in each column and column compression codecs:

```
SELECT
    name,
    type,
    formatReadableSize(data_uncompressed_bytes) AS uncompressed,
    formatReadableSize(data_compressed_bytes) AS compressed,
    round(data_uncompressed_bytes / data_compressed_bytes, 2) AS ratio,
compression_codec codec
FROM system.columns
WHERE table = 'compression_test' AND name != 'n';
```

The result of this testing shows that specialized codecs can significantly affect the level of data compression by general-purpose codecs, depending on a column data type. For example, for the Int64 type, both algorithms (LZ4 and ZSTD) provide good compression together with the DoubleDelta codec, and for the Float64 type — ZSTD together with Delta or FPC.

```
    ┌─name─────────────────┬─type────┬─uncompressed─┬─compressed─┬─ratio─┬─codec─────────────────────────┐
 1. │ i64                  │ Int64   │ 7.63 MiB     │ 7.63 MiB   │     1 │ CODEC(NONE)                   │
 2. │ i64_lz4              │ Int64   │ 7.63 MiB     │ 4.78 MiB   │   1.6 │ CODEC(LZ4)                    │
 3. │ i64_delta_lz4        │ Int64   │ 7.63 MiB     │ 2.89 MiB   │  2.64 │ CODEC(Delta(8), LZ4)          │
 4. │ i64_doubledelta_lz4  │ Int64   │ 7.63 MiB     │ 1.23 MiB   │  6.21 │ CODEC(DoubleDelta, LZ4)       │
 5. │ i64_t64_lz4          │ Int64   │ 7.63 MiB     │ 1.62 MiB   │  4.72 │ CODEC(T64, LZ4)               │
 6. │ i64_zstd             │ Int64   │ 7.63 MiB     │ 1.97 MiB   │  3.87 │ CODEC(ZSTD(1))                │
 7. │ i64_delta_zstd       │ Int64   │ 7.63 MiB     │ 1.36 MiB   │  5.62 │ CODEC(Delta(8), ZSTD(1))      │
 8. │ i64_doubledelta_zstd │ Int64   │ 7.63 MiB     │ 1.22 MiB   │  6.23 │ CODEC(DoubleDelta, ZSTD(1))   │
 9. │ i64_t64_zstd         │ Int64   │ 7.63 MiB     │ 1.42 MiB   │  5.36 │ CODEC(T64, ZSTD(1))           │
10. │ f64                  │ Float64 │ 7.63 MiB     │ 7.63 MiB   │     1 │ CODEC(NONE)                   │
11. │ f64_lz4              │ Float64 │ 7.63 MiB     │ 5.01 MiB   │  1.52 │ CODEC(LZ4)                    │
12. │ f64_delta_lz4        │ Float64 │ 7.63 MiB     │ 2.90 MiB   │  2.63 │ CODEC(Delta(8), LZ4)          │
13. │ f64_doubledelta_lz4  │ Float64 │ 7.63 MiB     │ 3.47 MiB   │   2.2 │ CODEC(DoubleDelta, LZ4)       │
14. │ f64_gorilla_lz4      │ Float64 │ 7.63 MiB     │ 2.66 MiB   │  2.87 │ CODEC(Gorilla, LZ4)           │
15. │ f64_fpc_lz4          │ Float64 │ 7.63 MiB     │ 3.11 MiB   │  2.45 │ CODEC(FPC(12), LZ4)           │
16. │ f64_zstd             │ Float64 │ 7.63 MiB     │ 2.22 MiB   │  3.44 │ CODEC(ZSTD(1))                │
17. │ f64_delta_zstd       │ Float64 │ 7.63 MiB     │ 1.49 MiB   │  5.11 │ CODEC(Delta(8), ZSTD(1))      │
18. │ f64_doubledelta_zstd │ Float64 │ 7.63 MiB     │ 2.05 MiB   │  3.72 │ CODEC(DoubleDelta, ZSTD(1))   │
19. │ f64_gorilla_zstd     │ Float64 │ 7.63 MiB     │ 2.32 MiB   │  3.29 │ CODEC(Gorilla, ZSTD(1))       │
20. │ f64_fpc_zstd         │ Float64 │ 7.63 MiB     │ 1.87 MiB   │  4.08 │ CODEC(FPC(12), ZSTD(1))       │
    └──────────────────────┴─────────┴──────────────┴────────────┴───────┴───────────────────────────────┘
```

For random data, Delta and DoubleDelta do not work very well, as you can see from the example below. Such data is better compressed with T64 (for integer values) and Gorilla (for floating point numbers) — so these codecs can be recommended for use when the data pattern is unknown.

```
CREATE TABLE compression_test_rand AS compression_test;
```

```
INSERT INTO compression_test_rand (n) SELECT rand()%(1000000) FROM numbers(1000000);
```

```
SELECT
    name,
    type,
    formatReadableSize(data_uncompressed_bytes) AS uncompressed,
    formatReadableSize(data_compressed_bytes) AS compressed,
    round(data_uncompressed_bytes / data_compressed_bytes, 2) AS ratio,
compression_codec codec
FROM system.columns
WHERE table = 'compression_test_rand' AND name != 'n';
```

```
    ┌─name─────────────────┬─type────┬─uncompressed─┬─compressed─┬─ratio─┬─codec─────────────────────────┐
 1. │ i64                  │ Int64   │ 7.63 MiB     │ 7.63 MiB   │     1 │ CODEC(NONE)                   │
 2. │ i64_lz4              │ Int64   │ 7.63 MiB     │ 4.40 MiB   │  1.73 │ CODEC(LZ4)                    │
 3. │ i64_delta_lz4        │ Int64   │ 7.63 MiB     │ 4.63 MiB   │  1.65 │ CODEC(Delta(8), LZ4)          │
 4. │ i64_doubledelta_lz4  │ Int64   │ 7.63 MiB     │ 4.43 MiB   │  1.72 │ CODEC(DoubleDelta, LZ4)       │
 5. │ i64_t64_lz4          │ Int64   │ 7.63 MiB     │ 2.40 MiB   │  3.18 │ CODEC(T64, LZ4)               │
 6. │ i64_zstd             │ Int64   │ 7.63 MiB     │ 3.32 MiB   │   2.3 │ CODEC(ZSTD(1))                │
 7. │ i64_delta_zstd       │ Int64   │ 7.63 MiB     │ 3.42 MiB   │  2.23 │ CODEC(Delta(8), ZSTD(1))      │
 8. │ i64_doubledelta_zstd │ Int64   │ 7.63 MiB     │ 3.84 MiB   │  1.99 │ CODEC(DoubleDelta, ZSTD(1))   │
 9. │ i64_t64_zstd         │ Int64   │ 7.63 MiB     │ 2.39 MiB   │  3.19 │ CODEC(T64, ZSTD(1))           │
10. │ f64                  │ Float64 │ 7.63 MiB     │ 7.63 MiB   │     1 │ CODEC(NONE)                   │
11. │ f64_lz4              │ Float64 │ 7.63 MiB     │ 4.67 MiB   │  1.63 │ CODEC(LZ4)                    │
12. │ f64_delta_lz4        │ Float64 │ 7.63 MiB     │ 4.85 MiB   │  1.57 │ CODEC(Delta(8), LZ4)          │
13. │ f64_doubledelta_lz4  │ Float64 │ 7.63 MiB     │ 5.94 MiB   │  1.28 │ CODEC(DoubleDelta, LZ4)       │
14. │ f64_gorilla_lz4      │ Float64 │ 7.63 MiB     │ 3.12 MiB   │  2.44 │ CODEC(Gorilla, LZ4)           │
15. │ f64_fpc_lz4          │ Float64 │ 7.63 MiB     │ 5.03 MiB   │  1.52 │ CODEC(FPC(12), LZ4)           │
16. │ f64_zstd             │ Float64 │ 7.63 MiB     │ 3.26 MiB   │  2.34 │ CODEC(ZSTD(1))                │
17. │ f64_delta_zstd       │ Float64 │ 7.63 MiB     │ 3.44 MiB   │  2.22 │ CODEC(Delta(8), ZSTD(1))      │
18. │ f64_doubledelta_zstd │ Float64 │ 7.63 MiB     │ 4.07 MiB   │  1.87 │ CODEC(DoubleDelta, ZSTD(1))   │
19. │ f64_gorilla_zstd     │ Float64 │ 7.63 MiB     │ 3.05 MiB   │   2.5 │ CODEC(Gorilla, ZSTD(1))       │
20. │ f64_fpc_zstd         │ Float64 │ 7.63 MiB     │ 4.20 MiB   │  1.82 │ CODEC(FPC(12), ZSTD(1))       │
    └──────────────────────┴─────────┴──────────────┴────────────┴───────┴───────────────────────────────┘
```

Use TTL to manage data

Elena Dvoryadkina

Contents

To Table of Contents

Overview
When TTL events are triggered
Table TTL
  - Create a table with TTL
  - Modify TTL
  - Remove TTL
  - Examples
Column TTL
  - Assign TTL to columns
  - Modify TTL
  - Remove TTL
  - Examples

Overview

For MergeTree tables, it is possible to set TTL (time-to-live) — time interval after which old data will be deleted from a table in the background. TTL can be set for a whole table (entire data rows with outdated values will be deleted) and for each individual column (outdated values in a column will be replaced with default values for the column data type).

Table-level TTL also allows you to schedule the following actions to be performed automatically on expired data rows:

- Move data between disks or volumes of an ADQM storage — this can be useful when you deploy an architecture to separate data into "hot", "warm", and "cold" (see Configure data storages).
- Rollup data into some aggregations and computations — you can store old data with a lower level of detail to save resources.
- Apply a higher level of compression to old data.

Thus, you can optimize a storage and improve query performance by managing your data lifecycle with the TTL mechanism.

When TTL events are triggered

ADQM deletes data with expired TTL when merging data parts (not immediately). It periodically performs "off-schedule" merges to delete or compress data according to TTL rules. You can control the frequency of such merges with the special parameters of a MergeTree table:

- `merge_with_ttl_timeout` — minimum delay in seconds between merges to delete data based TTL;
- `merge_with_recompression_ttl_timeout` — minimum delay in seconds between merges for data compression based on TTL.

By default, these parameters are set to 14400 seconds. This means that deletion/compression of outdated data will be performed every 4 hours or when a background merge of data parts occurs. If you want TTL rules to apply more frequently, change the corresponding setting above for your table. Note that too low value can lead to performing many "off-schedule" merges and consuming a lot of server resources — so it is not recommended to use values smaller than 300 seconds.

A MergeTree table also has the `ttl_only_drop_parts` setting that affects the expired data deletion as follows (depending on this parameter value):

- `0` (default) — ADQM deletes only rows expired according to their TTL;
- `1` — ADQM deletes a whole data part when all rows in it are expired.

Deleting whole parts instead of individual rows allows you to set a smaller `merge_with_ttl_timeout` timeout and reduce the impact of TTL operations on system performance.

Operations of moving data between disks/volumes or recompressing data are performed only when all rows in a data part are out of date.

When you run a `SELECT` query between merges, you can get expired data in a table. To avoid this, run the `OPTIMIZE … FINAL` query before selecting (to force merging). However, it is not recommended to use it frequently, especially for large tables.

Table TTL

Create a table with TTL

To set TTL for a table when creating it, use the `TTL` clause at the end of the table definition in the CREATE TABLE query. After the `TTL` clause, you can specify one rule to delete expired rows and/or multiple TTL rules to process data in a different way (for example, to move data to another disk/volume or recompress it).

Below is the basic syntax of a query for creating a table with TTL:

```
CREATE TABLE <table_name>
(   <timestamp_column> Date|DateTime ...,
    <column1> ...,
    <column2> ...,
    ...)
ENGINE = MergeTree()
TTL <ttl_expr> [<ttl_action>] [, <ttl_expr2> [<ttl_action2>]] [, ...];
```

where each TTL rule includes:

- `<ttl_expr>` — TTL expression that specifies when data to be considered outdated;
- `<ttl_action>` — action to be performed after the time specified by the TTL expression has passed.

A TTL expression that should return a value of the Date or DateTime type is usually specified in the following format:

```
<timestamp_column> [+ INTERVAL <time_interval> <interval_type>]
```

where:

- `<timestamp_column>` — name of a column of the Date or DateTime type with timestamps relative to which a time to live of data will be determined;
- `INTERVAL <time_interval> <interval_type>` — value of the Interval type that describes how long a data row should live, relative to the timestamp from the `<timestamp_column>` column. The structure of this value includes a time interval as a positive integer value and an interval type — for example, `1 MONTH`, `3 DAY`, `5 HOUR`.

To specify a time interval, you can use the toInterval conversion function instead of the `INTERVAL` operator. In this case, an example of a TTL expression can look like one of the following:

```
<timestamp_column> toIntervalMonth(1)
<timestamp_column> toIntervalDay(3)
<timestamp_column> toIntervalHour(5)
```

To specify actions to be performed on data rows that have expired according to their TTL expressions, use clauses listed in the table below.

Clause	Basic TTL syntax	Action description
DELETE	`TTL <ttl_expr> [DELETE] [WHERE <filter_expr>]`	Delete expired data rows. This is the default action (in other words, expired rows are removed if another action is not specified explicitly after the `TTL` clause)
TO DISK	`TTL <ttl_expr> TO DISK '<disk_name>'`	Move data to the specified disk
TO VOLUME	`TTL <ttl_expr> TO VOLUME '<volume_name>'`	Move data to the specified volume
RECOMPRESS	`TTL <ttl_expr> RECOMPRESS CODEC(<compression_codec>)`	Compress data using the specified codec
GROUP BY		Aggregate data. When defining a rule for aggregating old data, take into account the following points: - A `GROUP BY` expression (columns in `<key_expr>`) should be a prefix of the table primary key. - If a column is not part of the `GROUP BY` expression, and it is not explicitly specified after the `SET` clause, in a resulting row it will contain a value chosen randomly from one of the grouped rows (as if the `any` aggregate function is applied to it).

GROUP BY

```
TTL <ttl_expr>
    GROUP BY <key_expr>
    [SET <column_name> = aggr_func(<column_name>)[,
...]]
    [WHERE <filter_expr>]
```

For the `DELETE` or `GROUP BY` action, you can specify a filtering condition with the `WHERE` clause to delete/aggregate only some of the expired rows. For moving and compressing data, the `WHERE` clause is not applicable.

Modify TTL

You can change TTL of an existing table using the following query:

```
ALTER TABLE <table_name> MODIFY TTL <new_ttl_rules>;
```

After this query execution, each time new data is inserted or a new data part is created as a result of a background merge operation, TTL will be calculated according to a new rule.

To recalculate TTL for existing data, TTL materialization is required. The `materialize_ttl_after_modify` setting (enabled by default) controls whether TTL materialization should be performed automatically after the `ALTER TABLE … MODIFY TTL` query execution. You can also force TTL materialization with the following query:

```
ALTER TABLE <table_name> MATERIALIZE TTL;
```

TTL materialization can be a quite heavy operation (especially for large tables), as it recalculates TTL in all data parts (updates *ttl.txt* files) and rewrites data for all columns, which can result in a lot of read/write operations. To facilitate the TTL materialization operation, you can set `materialize_ttl_recalculate_only = 1` in the MergeTree table settings. In this case, only *ttl.txt* recalculation will be performed (it is a lightweight operation as only columns used in the TTL expression are read) and all columns will be copied via hard links. Some time later, expired data will be deleted during merge with `ttl_only_drop_parts = 1`.

In some cases, an optimal solution can be to disable TTL materialization (`materialize_ttl_after_modify = 0`) and delete/move old partitions manually (using the `ALTER TABLE … DROP|MOVE PARTITION` queries — see Manipulate partitions and data parts for details).

Remove TTL

To remove TTL rules from a table, use the query:

```
ALTER TABLE <table_name> REMOVE TTL;
```

Examples

Automatically delete expired data

1. Create a table with a TTL rule to remove rows that will be considered outdated when 10 minutes have passed since the time in the `time` column:

   ```
   CREATE TABLE test_table (time DateTime, value UInt64)
   ENGINE = MergeTree
   ORDER BY time
   TTL time + INTERVAL 10 MINUTE;
   ```

   Fill the table with test data:

   ```
   INSERT INTO test_table
   SELECT now() - toIntervalMinute(number), number
   FROM numbers(10);
   ```

   ```
   SELECT * FROM test_table;
   ```

   ```
       ┌────────────────time─┬─value─┐
    1. │ 2024-06-07 13:15:49 │     9 │
    2. │ 2024-06-07 13:16:49 │     8 │
    3. │ 2024-06-07 13:17:49 │     7 │
    4. │ 2024-06-07 13:18:49 │     6 │
    5. │ 2024-06-07 13:19:49 │     5 │
    6. │ 2024-06-07 13:20:49 │     4 │
    7. │ 2024-06-07 13:21:49 │     3 │
    8. │ 2024-06-07 13:22:49 │     2 │
    9. │ 2024-06-07 13:23:49 │     1 │
   10. │ 2024-06-07 13:24:49 │     0 │
       └─────────────────────┴───────┘
   ```

2. Wait for 5 minutes after inserting data and run the query:

   ```
   OPTIMIZE TABLE test_table FINAL;
   ```

   Repeat the query to select data from the table:

   ```
   SELECT * FROM test_table;
   ```

   You can see that the first five rows have been removed from the table (for each row, 10 minutes have passed since the time specified in the `time` column):

   ```
      ┌────────────────time─┬─value─┐
   1. │ 2024-06-07 13:20:49 │     4 │
   2. │ 2024-06-07 13:21:49 │     3 │
   3. │ 2024-06-07 13:22:49 │     2 │
   4. │ 2024-06-07 13:23:49 │     1 │
   5. │ 2024-06-07 13:24:49 │     0 │
      └─────────────────────┴───────┘
   ```

   If you do not use the `OPTIMIZE TABLE … FINAL` query and run `SELECT` between merges, outdated records can still exist in the table.

Filter rows to delete

1. Create the `alerts` table and define a TTL rule to remove rows with the `Critical` value in the `alert` column and a time value in the `time` column after which 10 minutes have passed:

   ```
   CREATE TABLE alerts (time DateTime, value UInt64, alert String)
   ENGINE = MergeTree
   ORDER BY time
   TTL time + INTERVAL 10 MINUTE DELETE WHERE alert = 'Critical';
   ```

   Insert test data into the table:

   ```
   INSERT INTO alerts
   SELECT now() - toIntervalMinute(number), number, ['Warning', 'Critical', 'Info'][(rand() % 3) + 1]
   FROM numbers(10);
   ```

   ```
   SELECT * FROM alerts;
   ```

   ```
       ┌────────────────time─┬─value─┬─alert────┐
    1. │ 2024-06-10 07:14:56 │     9 │ Critical │
    2. │ 2024-06-10 07:15:56 │     8 │ Warning  │
    3. │ 2024-06-10 07:16:56 │     7 │ Info     │
    4. │ 2024-06-10 07:17:56 │     6 │ Critical │
    5. │ 2024-06-10 07:18:56 │     5 │ Warning  │
    6. │ 2024-06-10 07:19:56 │     4 │ Info     │
    7. │ 2024-06-10 07:20:56 │     3 │ Critical │
    8. │ 2024-06-10 07:21:56 │     2 │ Warning  │
    9. │ 2024-06-10 07:22:56 │     1 │ Warning  │
   10. │ 2024-06-10 07:23:56 │     0 │ Critical │
       └─────────────────────┴───────┴──────────┘
   ```

2. 10 minutes after inserting the data, initiate a merge of data parts and look at the table contents:

   ```
   OPTIMIZE TABLE alerts FINAL;
   ```

   ```
   SELECT * FROM alerts;
   ```

   All old `Critical` alerts have been removed from the table:

   ```
      ┌────────────────time─┬─value─┬─alert───┐
   1. │ 2024-06-10 07:15:56 │     8 │ Warning │
   2. │ 2024-06-10 07:16:56 │     7 │ Info    │
   3. │ 2024-06-10 07:18:56 │     5 │ Warning │
   4. │ 2024-06-10 07:19:56 │     4 │ Info    │
   5. │ 2024-06-10 07:21:56 │     2 │ Warning │
   6. │ 2024-06-10 07:22:56 │     1 │ Warning │
      └─────────────────────┴───────┴─────────┘
   ```

Save data to another table before deleting

If you need to move data to another table before it is deleted from the main table, you can use TTL in combination with a materialized view.

1. Create:

   - the `alerts` table with a TTL rule to remove alerts after 10 minutes relative to the time from the `time` column;
   - the `alerts_history` table to store `Critical` alerts from the `alerts` table.

   ```
   DROP TABLE IF EXISTS alerts;
   ```

   ```
   CREATE TABLE alerts (time DateTime, value UInt64, alert String)
   ENGINE = MergeTree
   ORDER BY time
   TTL time + INTERVAL 10 MINUTE DELETE;
   ```

   ```
   CREATE TABLE alerts_history (time DateTime, value UInt64, alert String)
   ENGINE = MergeTree
   ORDER BY time;
   ```

   Note that the `alerts_history` table has the same structure as `alerts`, but do not use the `CREATE TABLE alerts_history AS alerts` query as it will also copy the TTL rule which is not needed.

2. Create a materialized view that will be triggered on inserting data into the `alerts` table and automatically load `Critical` alerts into `alerts_history`:

   ```
   CREATE MATERIALIZED VIEW alerts_history_mv TO alerts_history AS
   SELECT * FROM alerts WHERE alert = 'Critical';
   ```

3. Insert test data into the `alerts` table:

   ```
   INSERT INTO alerts
   SELECT now() - toIntervalMinute(number), number, ['Warning', 'Critical', 'Info'][(rand() % 3) + 1]
   FROM numbers(10);
   ```

   ```
   SELECT * FROM alerts;
   ```

   ```
       ┌────────────────time─┬─value─┬─alert────┐
    1. │ 2024-06-10 08:02:16 │     9 │ Warning  │
    2. │ 2024-06-10 08:03:16 │     8 │ Warning  │
    3. │ 2024-06-10 08:04:16 │     7 │ Info     │
    4. │ 2024-06-10 08:05:16 │     6 │ Critical │
    5. │ 2024-06-10 08:06:16 │     5 │ Warning  │
    6. │ 2024-06-10 08:07:16 │     4 │ Warning  │
    7. │ 2024-06-10 08:08:16 │     3 │ Critical │
    8. │ 2024-06-10 08:09:16 │     2 │ Warning  │
    9. │ 2024-06-10 08:10:16 │     1 │ Critical │
   10. │ 2024-06-10 08:11:16 │     0 │ Warning  │
       └─────────────────────┴───────┴──────────┘
   ```

4. Wait for 10 minutes and run the query:

   ```
   OPTIMIZE TABLE alerts FINAL;
   ```

   TTL will delete records from the `alerts` table, but rows with the `Critical` value in the `alert` column will be retained in the `alerts_history` table:

   ```
   SELECT * FROM alerts_history;
   ```

   ```
      ┌────────────────time─┬─value─┬─alert────┐
   1. │ 2024-06-10 08:05:16 │     6 │ Critical │
   2. │ 2024-06-10 08:08:16 │     3 │ Critical │
   3. │ 2024-06-10 08:10:16 │     1 │ Critical │
      └─────────────────────┴───────┴──────────┘
   ```

Aggregate outdated data to compact it

1. Create the `alerts` table and fill it with test data:

   ```
   DROP TABLE IF EXISTS alerts;
   ```

   ```
   CREATE TABLE alerts (time DateTime, value UInt64, alert String)
   ENGINE = MergeTree
   ORDER BY (toHour(time), alert);
   ```

   ```
   INSERT INTO alerts
   SELECT now() - toIntervalMinute(number), toInt64(randUniform(0, 100)), ['Warning', 'Critical', 'Info'][(rand() % 3) + 1]
   FROM numbers(15);
   ```

   ```
   SELECT * FROM alerts;
   ```

   ```
       ┌────────────────time─┬─value─┬─alert────┐
    1. │ 2024-07-26 21:51:41 │     5 │ Critical │
    2. │ 2024-07-26 21:48:41 │    18 │ Critical │
    3. │ 2024-07-26 21:46:41 │    53 │ Critical │
    4. │ 2024-07-26 21:43:41 │    33 │ Critical │
    5. │ 2024-07-26 21:50:41 │    98 │ Info     │
    6. │ 2024-07-26 21:49:41 │    74 │ Info     │
    7. │ 2024-07-26 21:47:41 │    73 │ Info     │
    8. │ 2024-07-26 21:42:41 │    52 │ Info     │
    9. │ 2024-07-26 21:41:41 │    92 │ Info     │
   10. │ 2024-07-26 21:39:41 │    70 │ Info     │
   11. │ 2024-07-26 21:38:41 │    81 │ Info     │
   12. │ 2024-07-26 21:37:41 │    72 │ Info     │
   13. │ 2024-07-26 21:45:41 │     7 │ Warning  │
   14. │ 2024-07-26 21:44:41 │    96 │ Warning  │
   15. │ 2024-07-26 21:40:41 │     0 │ Warning  │
       └─────────────────────┴───────┴──────────┘
   ```

2. Five minutes after inserting data (at `21:56:41` relative to timestamps generated above), modify the table to add a TTL rule that groups outdated data rows (if 10 minutes have passed since the time in the `time` column) by time and alert type, with each group storing the maximum value in the `value` column and the most recent time in the `time` column:

   ```
   ALTER TABLE alerts MODIFY TTL time + INTERVAL 10 MINUTE GROUP BY toHour(time), alert SET value = max(value), time =
   max(time);
   ```

   Note that `GROUP BY` columns in TTL should be a prefix of the table primary key (`ORDER BY` if `PRIMARY KEY` is not specified separately).

3. Perform TTL materialization to apply the TTL rule to existing data:

   ```
   ALTER TABLE alerts MATERIALIZE TTL;
   ```

4. Check the contents of the table:

   ```
   SELECT * FROM alerts;
   ```

   Rows for which the TTL rule was met (10 minutes have passed since their `time` values) were grouped, while the rest of the rows remained as before:

   ```
      ┌────────────────time─┬─value─┬─alert────┐
   1. │ 2024-07-26 21:51:41 │     5 │ Critical │
   2. │ 2024-07-26 21:48:41 │    18 │ Critical │
   3. │ 2024-07-26 21:46:41 │    53 │ Critical │
   4. │ 2024-07-26 21:50:41 │    98 │ Info     │
   5. │ 2024-07-26 21:49:41 │    74 │ Info     │
   6. │ 2024-07-26 21:47:41 │    73 │ Info     │
   7. │ 2024-07-26 21:42:41 │    92 │ Info     │
   8. │ 2024-07-26 21:45:41 │    96 │ Warning  │
      └─────────────────────┴───────┴──────────┘
   ```

   If you run optimization after another 10 minutes, the data will be aggregated further — the table will have one row for the most recent alert of each type:

   ```
   OPTIMIZE TABLE alerts FINAL;
   ```

   ```
      ┌────────────────time─┬─value─┬─alert────┐
   1. │ 2024-07-26 21:51:41 │    53 │ Critical │
   2. │ 2024-07-26 21:50:41 │    98 │ Info     │
   3. │ 2024-07-26 21:45:41 │    96 │ Warning  │
      └─────────────────────┴───────┴──────────┘
   ```

Change compression for outdated data

1. Create a table with a TTL rule to compress old data with the `LZ4HC(10)` codec and also set `merge_with_recompression_ttl_timeout = 1200` (in seconds) in the table settings to specify how often data should be recompressed according to TTL (in this example, every 20 minutes):

   ```
   DROP TABLE IF EXISTS alerts;
   ```

   ```
   CREATE TABLE alerts (time DateTime, value UInt64, alert String)
   ENGINE = MergeTree
   ORDER BY time
   TTL time + INTERVAL 10 MINUTE RECOMPRESS CODEC(LZ4HC(10))
   SETTINGS merge_with_recompression_ttl_timeout = 1200;
   ```

2. Insert data:

   ```
   INSERT INTO alerts
   SELECT now() - toIntervalMinute(number), number, ['Warning', 'Critical', 'Info'][(rand() % 3) + 1]
   FROM numbers(100);
   ```

3. View information about data parts of the `alerts` table in the `system.parts` system table:

   ```
   SELECT name, active, rows, data_compressed_bytes, data_uncompressed_bytes
   FROM system.parts WHERE table = 'alerts'
   FORMAT Vertical;
   ```

   Inserted data rows were written as one part:

   ```
   Row 1:
   ──────
   name:                    all_1_1_0
   active:                  1
   rows:                    100
   data_compressed_bytes:   1114
   data_uncompressed_bytes: 1933
   ```

4. Repeat the above `INSERT INTO` and `SELECT … FROM system.parts` queries. Now the table has two data parts:

   ```
   Row 1:
   ──────
   name:                    all_1_1_0
   active:                  1
   rows:                    100
   data_compressed_bytes:   1114
   data_uncompressed_bytes: 1933

   Row 2:
   ──────
   name:                    all_2_2_0
   active:                  1
   rows:                    100
   data_compressed_bytes:   1125
   data_uncompressed_bytes: 1931
   ```

5. Some time later, repeat the `SELECT … FROM system.parts` query. In the query results, you can see:

   - initial data parts have been merged into one part (`all_1_2_1`) containing 200 rows;
   - the size of uncompressed data in the resulting part is equal to the total amount of uncompressed data in two merged parts;
   - the size of compressed data in the resulting part is smaller than the size of compressed data in both initial parts because the outdated data has been recompressed with the LZ4HC(10) codec during merging.

   ```
   Row 1:
   ──────
   name:                    all_1_2_1
   active:                  1
   rows:                    200
   data_compressed_bytes:   1509
   data_uncompressed_bytes: 3864
   ```

Move data between hot and cold storages

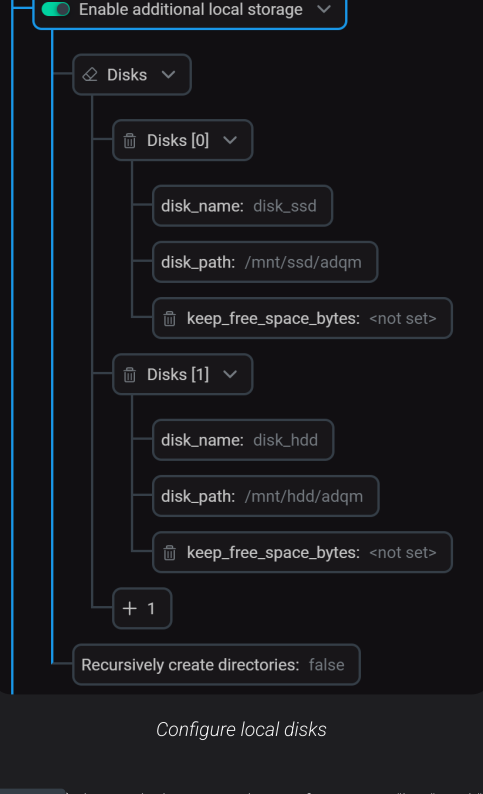
In this example, separate disks are used to store "hot" and "cold" data — SSD and HDD disks mounted on the following directories:

- */mnt/ssd/adqm/*
- */mnt/hdd/adqm/*

Each directory is assigned the `clickhouse` owner:

```
$ sudo chown clickhouse:clickhouse -R /mnt/ssd/adqm/ /mnt/hdd/adqm/
```

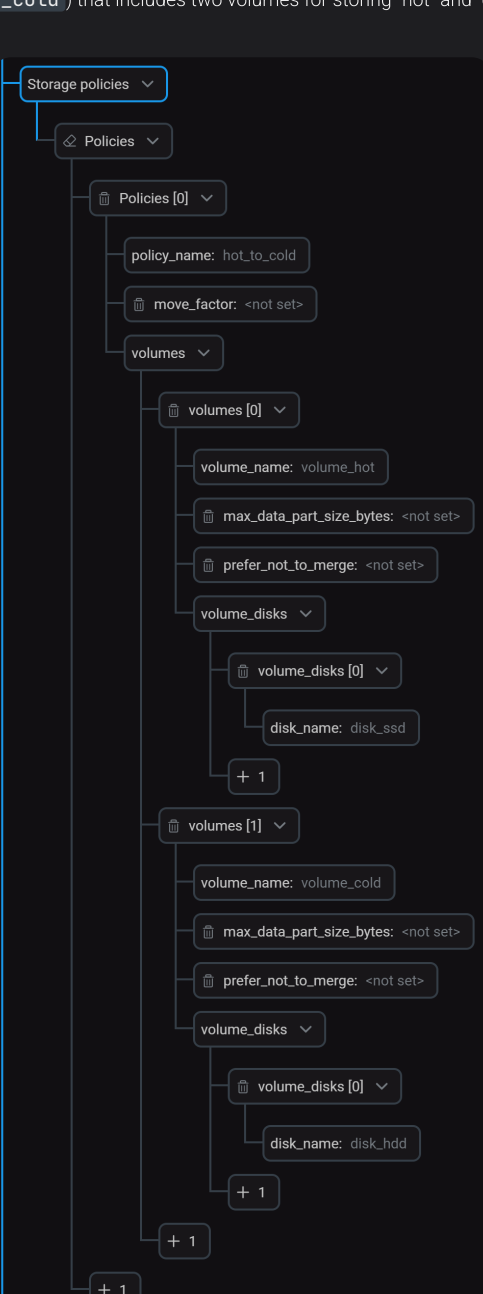
1. In the ADCM interface, configure a tiered storage for ADQM as described below.
   Specify local disks in the ***Enable additional local storage*** section (previously enabled) of the ADQMDB service's configuration page:

   Enable additional local storage
   Disks
   Disks [0]
   disk_name: disk_ssd
   disk_path: /mnt/ssd/adqm
   keep_free_space_bytes: <not set>
   Disks [1]
   disk_name: disk_hdd
   disk_path: /mnt/hdd/adqm
   keep_free_space_bytes: <not set>
   Recursively create directories: false

   Configure local disks

   In the ***Storage policies*** section, create a policy (`hot_to_cold`) that includes two volumes for storing "hot" and "cold" data.

   Storage policies
   Policies
   Policies [0]
   policy_name: hot_to_cold
   move_factor: <not set>
   volumes
   volumes [0]
   volume_name: volume_hot
   max_data_part_size_bytes: <not set>
   prefer_not_to_merge: <not set>
   volume_disks
   volume_disks [0]
   disk_name: disk_ssd
   volumes [1]
   volume_name: volume_cold
   max_data_part_size_bytes: <not set>
   prefer_not_to_merge: <not set>
   volume_disks
   volume_disks [0]
   disk_name: disk_hdd

   Configure storage policy

   Click ***Save*** and run the ***Reconfig*** action for the service to apply the changes.

   After that, information about disks and policy will appear in the appropriate tags inside the `<storage_configuration>` section in the */etc/clickhouse-server/config.d/storage.xml* file.

   storage.xml

   ```
   <storage_configuration>
       <disks>
           <disk_ssd>
               <type>local</type>
               <path>/mnt/ssd/adqm/</path>
           </disk_ssd>
           <disk_hdd>
               <type>local</type>
               <path>/mnt/hdd/adqm/</path>
           </disk_hdd>
       </disks>
       <policies>
           <hot_to_cold>
               <volumes>
                   <volume_hot>
                       <disk>disk_ssd</disk>
                   </volume_hot>
                   <volume_cold>
                       <disk>disk_hdd</disk>
                   </volume_cold>
               </volumes>
           </hot_to_cold>
       </policies>
   </storage_configuration>
   ```

   You can view available disks in the `system.disks` system table and storage policies in the `system.storage_policies` table:

   ```
   SELECT name, path, free_space, total_space FROM system.disks;
   ```

   ```
      ┌─name─────┬─path─────────────────┬──free_space─┬─total_space─┐
   1. │ default  │ /var/lib/clickhouse/ │ 45148381184 │ 53674487808 │
   2. │ disk_hdd │ /mnt/hdd/adqm/       │ 45148381184 │ 53674487808 │
   3. │ disk_ssd │ /mnt/ssd/adqm/       │ 45148381184 │ 53674487808 │
      └──────────┴──────────────────────┴─────────────┴─────────────┘
   ```

   ```
   SELECT policy_name, volume_name, disks FROM system.storage_policies;
   ```

   ```
      ┌─policy_name─┬─volume_name─┬─disks────────┐
   1. │ default     │ default     │ ['default']  │
   2. │ hot_to_cold │ volume_hot  │ ['disk_ssd'] │
   3. │ hot_to_cold │ volume_cold │ ['disk_hdd'] │
      └─────────────┴─────────────┴──────────────┘
   ```

2. Create a table with a TTL rule to move data between the `volume_hot` and `volume_cold` volumes:

   ```
   DROP TABLE IF EXISTS alerts;
   ```

   ```
   CREATE TABLE alerts (time DateTime, value UInt64, alert String)
   ENGINE = MergeTree
   ORDER BY time
   TTL time TO VOLUME 'volume_hot', time + INTERVAL 5 MINUTE TO VOLUME 'volume_cold'
   SETTINGS storage_policy = 'hot_to_cold';
   ```

3. Insert test data into the table:

   ```
   INSERT INTO alerts
       SELECT
       now() - toIntervalMinute(number),
       number,
       ['Warning', 'Critical', 'Info'][(rand() % 3) + 1]
   FROM numbers(5);
   ```

   Repeat the `INSERT INTO` query some time later.

   As a result, the table will have two data parts located on the `disk_ssd` disk (the `volume_hot` volume):

   ```
   SELECT name, rows, active, disk_name FROM system.parts WHERE table = 'alerts';
   ```

   ```
      ┌─name──────┬─rows─┬─active─┬─disk_name─┐
   1. │ all_1_1_0 │    5 │      1 │ disk_ssd  │
   2. │ all_2_2_0 │    5 │      1 │ disk_ssd  │
      └───────────┴──────┴────────┴───────────┘
   ```

4. 5 minutes after the first insertion, the first data part will be moved to the `disk_hdd` disk (the `volume_cold` volume). To verify this, repeat the `SELECT … FROM system.parts` query:

   ```
      ┌─name──────┬─rows─┬─active─┬─disk_name─┐
   1. │ all_1_1_0 │    5 │      1 │ disk_hdd  │
   2. │ all_2_2_0 │    5 │      1 │ disk_ssd  │
      └───────────┴──────┴────────┴───────────┘
   ```

Column TTL

When values in a column with TTL expire, ADQM replaces them with the default value for the column data type (or the value specified via `DEFAULT`). If all values in a column expire, the column is removed from the data part in the file system.

> **IMPORTANT**
>
> TTL cannot be used for key columns.

Assign TTL to columns

To specify a time to live for column values, use the `TTL` clause in the column description when creating a table with the `CREATE TABLE` query:

```
CREATE TABLE <table_name>
(   <timestamp_column> Date|DateTime,
    ...,
    <column_name1> [<data_type1>] TTL <timestamp_column> + INTERVAL <time_interval1> <interval_type1>,
    <column_name2> [<data_type2>] TTL <timestamp_column> + INTERVAL <time_interval2> <interval_type2>,
    ...)
ENGINE = MergeTree()
...;
```

A TTL expression specifying when column values should be deleted is described in the same way as a table-level TTL expression.

Modify TTL

To add a TTL to a column of an existing table or change the existing TTL of a column, use the `ALTER TABLE … MODIFY COLUMN` query:

```
ALTER TABLE <table_name> MODIFY COLUMN <column_name> <data_type> TTL <new_ttl_expr>;
```

Remove TTL

The following query allows you to delete TTL from a column:

```
ALTER TABLE <table_name> MODIFY COLUMN <column_name> REMOVE TTL;
```

Examples

Create a table with column-level TTL

1. Create a table and set the following TTL rules for the columns:

   - `value1` — a value in the column should be replaced with `100` if 5 minutes have passed since the time in the `time` column;
   - `value2` — a column value should be replaced with the default one (`0` for the Int data type) if 10 minutes have passed since the time in the `time` column.

   ```
   CREATE TABLE column_ttl_test
   (
       time DateTime,
       value1 Int DEFAULT 100 TTL time + INTERVAL 5 MINUTE,
       value2 Int TTL time + INTERVAL 10 MINUTE,
       alert String
   )
   ENGINE = MergeTree
   ORDER BY time
   SETTINGS min_rows_for_wide_part = 0, min_bytes_for_wide_part = 0;
   ```

   The `min_rows_for_wide_part` and `min_bytes_for_wide_part` settings are used to store data parts in the `Wide` format (for testing purposes) — each column in a separate file.

2. Insert data into the table:

   ```
   INSERT INTO column_ttl_test
   SELECT
           now() - toIntervalMinute(number),
           number,
           number + 10,
           ['Warning', 'Critical', 'Info'][(rand() % 3) + 1]
   FROM numbers(10);
   ```

   The first five rows in the `value1` column contain `100` because their values in the `time` column were already outdated at the time of insertion:

   ```
   SELECT * FROM column_ttl_test;
   ```

   ```
       ┌────────────────time─┬─value1─┬─value2─┬─alert────┐
    1. │ 2024-06-25 11:42:36 │    100 │     19 │ Warning  │
    2. │ 2024-06-25 11:43:36 │    100 │     18 │ Critical │
    3. │ 2024-06-25 11:44:36 │    100 │     17 │ Warning  │
    4. │ 2024-06-25 11:45:36 │    100 │     16 │ Info     │
    5. │ 2024-06-25 11:46:36 │    100 │     15 │ Info     │
    6. │ 2024-06-25 11:47:36 │      4 │     14 │ Info     │
    7. │ 2024-06-25 11:48:36 │      3 │     13 │ Warning  │
    8. │ 2024-06-25 11:49:36 │      2 │     12 │ Info     │
    9. │ 2024-06-25 11:50:36 │      1 │     11 │ Critical │
   10. │ 2024-06-25 11:51:36 │      0 │     10 │ Critical │
       └─────────────────────┴────────┴────────┴──────────┘
   ```

3. 3 minutes after inserting data into the table, run the query:

   ```
   OPTIMIZE TABLE column_ttl_test FINAL;
   ```

   Select data from the table again. In the results, you can see that values of the `value1` column in rows 6-8 and values of the `value2` column in rows 1-3 have been replaced:

   ```
       ┌────────────────time─┬─value1─┬─value2─┬─alert────┐
    1. │ 2024-06-25 11:42:36 │    100 │      0 │ Warning  │
    2. │ 2024-06-25 11:43:36 │    100 │      0 │ Critical │
    3. │ 2024-06-25 11:44:36 │    100 │      0 │ Warning  │
    4. │ 2024-06-25 11:45:36 │    100 │     16 │ Info     │
    5. │ 2024-06-25 11:46:36 │    100 │     15 │ Info     │
    6. │ 2024-06-25 11:47:36 │    100 │     14 │ Info     │
    7. │ 2024-06-25 11:48:36 │    100 │     13 │ Warning  │
    8. │ 2024-06-25 11:49:36 │    100 │     12 │ Info     │
    9. │ 2024-06-25 11:50:36 │      1 │     11 │ Critical │
   10. │ 2024-06-25 11:51:36 │      0 │     10 │ Critical │
       └─────────────────────┴────────┴────────┴──────────┘
   ```

4. After another 2 minutes (5 minutes after the initial data insert), all values in the `value1` column will be considered outdated and replaced with `100`:

   ```
   OPTIMIZE TABLE column_ttl_test FINAL;
   ```

   ```
   SELECT * FROM column_ttl_test;
   ```

   ```
       ┌────────────────time─┬─value1─┬─value2─┬─alert────┐
    1. │ 2024-06-25 11:42:36 │    100 │      0 │ Warning  │
    2. │ 2024-06-25 11:43:36 │    100 │      0 │ Critical │
    3. │ 2024-06-25 11:44:36 │    100 │      0 │ Warning  │
    4. │ 2024-06-25 11:45:36 │    100 │      0 │ Info     │
    5. │ 2024-06-25 11:46:36 │    100 │      0 │ Info     │
    6. │ 2024-06-25 11:47:36 │    100 │     14 │ Info     │
    7. │ 2024-06-25 11:48:36 │    100 │     13 │ Warning  │
    8. │ 2024-06-25 11:49:36 │    100 │     12 │ Info     │
    9. │ 2024-06-25 11:50:36 │    100 │     11 │ Critical │
   10. │ 2024-06-25 11:51:36 │    100 │     10 │ Critical │
       └─────────────────────┴────────┴────────┴──────────┘
   ```

   The `value1` column has been removed from the data part directory in the file system. You can check this as follows.

   Obtain the path to the directory with files of the active data part (`all_1_1_3`) from the `system.parts` table:

   ```
   SELECT name, active, path FROM system.parts WHERE table = 'column_ttl_test';
   ```

   ```
      ┌─name──────┬─active─┬─path──────────────────────────────────────────────────────────────────────────┐
   1. │ all_1_1_0 │      0 │ /var/lib/clickhouse/store/62a/62a16a55-c91c-4c6b-9386-f4347ffac137/all_1_1_0/ │
   2. │ all_1_1_1 │      0 │ /var/lib/clickhouse/store/62a/62a16a55-c91c-4c6b-9386-f4347ffac137/all_1_1_1/ │
   3. │ all_1_1_2 │      0 │ /var/lib/clickhouse/store/62a/62a16a55-c91c-4c6b-9386-f4347ffac137/all_1_1_2/ │
   4. │ all_1_1_3 │      1 │ /var/lib/clickhouse/store/62a/62a16a55-c91c-4c6b-9386-f4347ffac137/all_1_1_3/ │
      └───────────┴────────┴───────────────────────────────────────────────────────────────────────────────┘
   ```

   Make sure that there are no files with data of the `value1` column (*value1.bin* and *value1.cmrk2*) in this directory:

   ```
   $ sudo ls /var/lib/clickhouse/store/62a/62a16a55-c91c-4c6b-9386-f4347ffac137/all_1_1_3/
   ```

   ```
   alert.bin    checksums.txt  count.txt                      metadata_version.txt  serialization.json  time.cmrk2  value2.bin
   alert.cmrk2  columns.txt    default_compression_codec.txt  primary.cidx          time.bin            ttl.txt     value2.
   cmrk2
   ```

   The *columns.txt* file also has no information about the `value1` column:

   ```
   $ sudo cat /var/lib/clickhouse/store/62a/62a16a55-c91c-4c6b-9386-f4347ffac137/all_1_1_3/columns.txt
   ```

   ```
   columns format version: 1
   3 columns:
   `time` DateTime
   `value2` Int32
   `alert` String
   ```

Change a column TTL in an existing table

Use the `ALTER TABLE … MODIFY COLUMN … TTL` query to change the TTL for the `value2` column of the table created in the previous example:

```
ALTER TABLE column_ttl_test MODIFY COLUMN value2 Int TTL time + INTERVAL 1 HOUR;
```

Check the current TTL settings with the `SHOW CREATE TABLE` query:

```
SHOW CREATE TABLE column_ttl_test;
```

As you can see, the new TTL rule is applied to the `value2` column:

```
CREATE TABLE default.column_ttl_test
(
    `time` DateTime,
    `value1` Int32 DEFAULT 100 TTL time + toIntervalMinute(5),
    `value2` Int32 TTL time + toIntervalHour(1),
    `alert` String
)
ENGINE = MergeTree
ORDER BY time
SETTINGS min_rows_for_wide_part = 0, min_bytes_for_wide_part = 0, index_granularity = 8192
```

Indexes

Elena Dvoryadkina

Contents

To Table of Contents

- Overview
- Sparse primary index
  - Specify a primary key
  - Compound primary key
- Data skipping indexes
  - Manage indexes
  - Index types
- Get information on indexes

Overview

Indexing is a technique to improve database performance. Indexes are special data structures that allow a database server to quickly find requested rows by values of a key column (or column set) without a full table scan.

ADQM supports the following types of indexes for MergeTree tables:

- Primary index. A primary index in ADQM differs from the index in traditional relational databases to perform analytical queries on large amounts of data most efficiently — it stores key column values not per table row but per range of N rows ordered by key columns (each index entry stores key column values in the first row of the corresponding range). When a primary index is defined correctly for a table, it reduces the amount of data to be read from disk for query execution and provides quick searches for ranges of rows with the desired values. A key to build the primary index should be specified on a table creation and cannot be changed after that.
- Data skipping indexes (secondary indexes). When a primary index is not enough to ensure high query performance, you can add data skipping indexes to your table — secondary indexes that enable ADQM to skip reading blocks of data (they can be quite large) that are guaranteed not to contain the requested values. For example, such indexes can be effective for full-text search or for checking whether values are included in specific lists. Unlike a primary index, skipping indexes can be added to or removed from existing tables.

Each type of ADQM index has its own advantages and features. When designing table indexes, take into account which queries will be sent to a table most often, characteristics of columns used in queries (data type, cardinality), desired balance between read/write performance and system resource consumption. To select the most effective index, it is often necessary to test different index types, granularity sizes, and other parameters.

> **TIP**
>
> In the how-to:performance-tuning/improve-query-performance.adoc#indexes_example article, you can find examples of applying indexes of different types to tables and explore how they affect query execution speed.

Sparse primary index

MergeTree table data is divided into **data parts**. Rows within a data part are sorted by primary key columns. Each data part is logically divided into **granules**. A granule is the smallest indivisible data set (always contains an integer number of rows) that is read when selecting data. The first row of a granule is marked with values of primary key columns for that row — this is a **mark**. For each data part, ADQM creates an index file that stores marks. Thus, instead of indexing each row, the primary index for a data part has one index entry (mark) per group of rows (granule) — this technique is called **sparse index**. A sparse primary index allows it to avoid scanning the entire table for individual rows and quickly (with a binary search over index entries) identify granules that could possibly include rows matching a query filter. Found groups of potentially matching rows are then loaded into memory in parallel in order to find exact matches.

This type of indexing makes a primary index small (it completely fits into memory) and still allows data to be searched on large tables in seconds, skipping terabytes or even petabytes of irrelevant data (although a sparse index allows extra data to be read).

Specify a primary key

For a primary index to work most effectively, a primary key should include a column that will filter the majority of queries accessing the table. A primary key can contain multiple columns — when defining a compound primary key, follow the recommendations below.

To assign a primary key to a MergeTree table, use the `ORDER BY` or `PRIMARY KEY` clause in the `CREATE TABLE` query when creating the table:

```
CREATE TABLE <table_name> (<column_name> <column_type>, ...)
ENGINE = MergeTree
ORDER BY <sorting_key_expr>
[PRIMARY KEY <primary_key_expr>];
```

- `ORDER BY` — sorting key defined as one or more columns. ADQM uses the sorting key as a primary key if the primary key is not specified explicitly via the `PRIMARY KEY` clause.
- `PRIMARY KEY` — primary key. By default, a primary key is the same as a sorting key (`ORDER BY`), and it is not necessary to specify the `PRIMARY KEY` clause separately in most cases. However, it is possible to set a primary key (to write key column values as marks in the index file) that differs from the sorting key (to use columns for ordering rows in data parts). In this case, the tuple of primary key columns must be a prefix of the tuple of sorting key columns (for example, if the primary key is `(a, b)`, then the sorting key must be `(a, b, c…)`).

Each table can have only one primary key. You can also create a table without a primary key — to do this, use the `ORDER BY tuple()` syntax.

Values in primary key columns do not have to be unique — ADQM allows multiple rows with the same values in key columns to be inserted into a table.

Compound primary key

In a compound primary key, the order of key columns is important and can significantly affect both:

- efficiency of filtering by secondary key columns in queries;
- compression ratio of table's data files.

When a query filters data by the first key column, ADQM runs a binary search algorithm over the index marks of that column. Filtering by other key columns uses the generic exclusion search algorithm that is less efficient. For example, the `(a, b)` primary key is useful for speeding up queries filtering by the `a` column, but it does not improve the performance of queries filtering by the `b` column, despite the `b` column is a part of the compound primary key. Conversely, the primary index of a table with the `(b, a)` compound primary key speeds up queries filtering by the `b` column, but does not optimize queries filtering by the `a` column.

If you want to speed up both types of queries (filtering by the first key column and filtering by the secondary key column) and the cardinality (number of distinct values) of columns in the compound primary key is equally high, it makes sense to remove the second key column from the primary index (this will result in less memory consumption of the index), and use multiple primary indexes instead. To implement this, you can use one of the following approaches:

- create a second table with a different primary key;
- create a materialized view on the existing table;
- add a projection to the existing table.

If columns in a compound primary key have large differences in cardinality, it is recommended to order the primary key columns by cardinality in ascending order to speed up queries. The higher the difference in cardinality between key columns, the more important the order of those columns in the key is.

Data skipping indexes

For MergeTree tables, you can also set **data skipping indexes**. A data skipping index aggregates some information about the specified expression (column) on data blocks. This information is then used in `SELECT` queries to reduce the amount of data to be read from disk by skipping blocks of data that are guaranteed not to match the query's filter criteria. A skipped block consists of data granules in an amount equal to the granularity of the given index.

Manage indexes

Describe skipping indexes using the `INDEX` clause within the column description section of the `CREATE TABLE` query:

```
CREATE TABLE <table_name>
(   <column_name> <column_type>, ...
    INDEX <index_name> <index_expr> TYPE <index_type> [GRANULARITY <granularity_value>], ...)
ENGINE = MergeTree
...;
```

where:

- `<index_name>` — index name;
- `<index_expr>` — expression by which the information will be aggregated (often it is just a column name);
- `<index_type>` — index type that defines which information about a data block the index should store (this information will allow finding the necessary blocks quickly and skipping all the others);
- `<granularity_value>` — index granularity that is the number of data granules in a skipped block (the default value is `1`).

You can add a skipping index to an existing table using the following query:

```
ALTER TABLE <table_name> ADD INDEX <index_name> <index_expr> TYPE <index_type> [GRANULARITY <granularity_value>];
```

When an index is added to an existing table, it is not automatically updated. `ALTER TABLE` changes metadata, and the index will only be calculated for new data inserted into the table. To apply the index to existing data, run the additional command:

```
ALTER TABLE <table_name> MATERIALIZE INDEX <index_name>;
```

To delete indexes, use the following queries:

- `ALTER TABLE <table_name> DROP INDEX <index_name>` — removes index description from table metadata and deletes index files from disk;
- `ALTER TABLE <table_name> CLEAR INDEX <index_name>` — removes index files from disk without removing the index description from metadata.

Index types

MinMax

The `minmax` index stores the minimum and maximum values of the column (or expression) for each block.

Set

The `set(<max_rows>)` index stores unique values of the column (or expression) per block up to `<max_rows>` (`0` means an unlimited number of unique values). When an indexed column is used in the `WHERE` clause, ADQM reads a small set of values instead of the entire column. This type of index works well if a column has low cardinality (a small number of unique values) in each set of granules, but high cardinality overall.

Bloom Filter

ADQM supports three different types of Bloom filter index:

- `bloom_filter([<false_positive>])` — Bloom filter for the column. The optional `<false_positive>` parameter defines the probability of receiving a false positive response from the filter. Possible values are between `0` and `1`. A default value is `0.025`.
  This index type is supported for the following data types: Int*, UInt*, Float*, Enum, Date, DateTime, String, FixedString, Array, LowCardinality, Nullable, UUID and Map.
- `tokenbf_v1(<size_of_bloom_filter_in_bytes>, <number_of_hash_functions>, <random_seed>)` — an input string is split into alphanumeric tokens (sequences of characters separated by non-alphanumeric characters) and then tokens are stored in a Bloom filter. It is suitable when an exact match on a string is searched (for example, when you want to find a specific part of the URL or a query parameter in the `URL` column).
  All three index parameters are related to the Bloom filter configuration:
  - `<size_of_bloom_filter_in_bytes>` — Bloom filter size in bytes (larger filters have fewer false positives);
  - `<number_of_hash_functions>` — number of hash functions used in the Bloom filter;
  - `<random_seed>` — seed for the Bloom filter hash functions.

  This index type works with the String, FixedString, and Map data types only.
- `ngrambf_v1(<n>, <size_of_bloom_filter_in_bytes>, <number_of_hash_functions>, <random_seed>)` — an input string is split into n-grams (a substring of n characters) and then stored in a Bloom filter. It is suitable for full-text search, particularly in languages without word breaks, such as Chinese. The first parameter (`<n>`) sets the size of n-grams, the rest ones are similar to `tokenbf_v1`. This index type works with the String, FixedString, and Map data types.

Inverted

The inverted index `inverted([<n>], [<max_rows_per_posting_list>])` stores a mapping of unique words (or n-grams) of a text column with pointers to their location in the table (the identifiers of rows in which they are contained). Use it to optimize full-text search across columns of the types: String, FixedString, Array(String), Array(FixedString), Map(String), and Map(String). Currently, it is in the experimental state. For more detailed information and an example of the use, see the Full-text search article.

Get information on indexes

The following system tables provide information about existing indexes:

- `system.tables` — the `primary_key` field contains information about the table's primary key.

  Example

  ```
  SELECT primary_key FROM system.tables WHERE table='table_compound_primary_key';
  ```

  ```
  ┌─primary_key─┐
  │ a, b        │
  └─────────────┘
  ```

- `system.data_skipping_indices` — contains information about existing data skipping indexes in all tables.

  Example

  ```
  SELECT type_full,expr FROM system.data_skipping_indices WHERE table='table_skip_index';
  ```

  ```
  ┌─type_full─┬─expr─┐
  │ set(100)  │ b    │
  └───────────┴──────┘
  ```

You can also use queries:

- `SHOW CREATE TABLE <table_name>`.

  Example

  ```
  SHOW CREATE TABLE table_skip_index;
  ```

  ```
  CREATE TABLE default.table_skip_index
  (
      `a` UInt64,
      `b` UInt64,
      INDEX my_index b TYPE set(100) GRANULARITY 2
  )
  ENGINE = MergeTree
  PRIMARY KEY a
  ORDER BY a
  SETTINGS index_granularity = 10000
  ```

- `SHOW {INDEX|INDEXES|INDICES|KEYS} FROM <table_name>`.

  Example

  ```
  SHOW INDEXES FROM table_skip_index;
  ```

  ```
  ┌─table────────────┬─non_unique─┬─key_name─┬─seq_in_index─┬─column_name─┬
  ┬─index_type─┬─comment─┬─index_comment─┬─visible─┬─expression─┐
  │ table_skip_index │          1 │ PRIMARY  │            1 │ a           │ ... │ PRIMARY    │         │               │ YES
  │            │
  └──────────────────┴────────────┴──────────┴──────────────┴─────────────┴─
  ─┴────────────┴─────────┴───────────────┴─────────┴────────────┘
  ┌─table────────────┬─non_unique─┬─key_name─┬─seq_in_index─┬─column_name─┬─
  ┬─index_type─┬─comment─┬─index_comment─┬─visible─┬─expression─┐
  │ table_skip_index │          1 │ my_index │            1 │             │ ... │ SET        │         │               │ YES
  │ b          │
  └──────────────────┴────────────┴──────────┴──────────────┴─────────────┴─
  ┴────────────┴─────────┴───────────────┴─────────┴────────────┘
  ```

To examine the use of indexes for an individual query, you can analyze that query execution using the EXPLAIN command with the `indexes` parameter.

Example

```
EXPLAIN indexes = 1 SELECT * FROM table_skip_index WHERE b=100 OR b=555;
```

```
┌─explain─────────────────────────────────────┐
│ Expression ((Project names + Projection))   │
│   Expression                                │
│     ReadFromMergeTree (default.table_skip_index) │
│     Indexes:                                │
│       PrimaryKey                            │
│         Condition: true                     │
│         Parts: 6/6                          │
│         Granules: 10004/10004               │
│       Skip                                  │
│         Name: my_index                      │
│         Description: set GRANULARITY 2      │
│         Parts: 1/6                          │
│         Granules: 4/10004                   │
└─────────────────────────────────────────────┘
```

Dictionaries

Elena Dvoryadkina

Contents

To Table of Contents

- Create and configure a dictionary
  - Dictionary key and attributes
  - Dictionary source
  - Store dictionaries in memory
  - Dictionary updates
  - Example
- View information on dictionaries
- Load dictionaries
- Use dictionaries
  - dictGet, dictGetOrDefault, dictGetOrNull
  - dictHas
  - Example
- Detach and attach a dictionary
- Delete a dictionary

A **dictionary** is a key/value data store that is fully or partially stored in the RAM of the ADQM server and can be used as a reference to substitute data values by keys in the final sample. Dictionaries are an easier-to-use alternative to the JOIN operator.

A data source for a dictionary can be a local file (text or executable), an HTTP(s) resource, or another DBMS. ADQM automatically updates dictionaries and loads missing or changed values from the source. Access to dictionaries from ADQM is read-only.

You can configure ADQM to connect dictionaries from external sources using XML configuration files or DDL queries. This article contains details on the second approach (recommended) — it describes syntax and provides examples of queries to create/configure/delete dictionaries in ADQM, as well as to read data from dictionaries using special functions.

Create and configure a dictionary

To create and configure a dictionary in ADQM, use the `CREATE DICTIONARY` query. The basic syntax is:

```
CREATE DICTIONARY [OR REPLACE][IF NOT EXISTS] [<database_name>.]<dictionary_name> [ON CLUSTER <cluster_name>]
(   <key1_name> <key1_data_type>,
    <key2_name> <key2_data_type>,
    <attr1_name> <attr1_data_type> [DEFAULT <default_value1>] [EXPRESSION <expr1>] [HIERARCHICAL] [INJECTIVE] [IS_OBJECT_ID],
    <attr2_name> <attr2_data_type> [DEFAULT <default_value2>] [EXPRESSION <expr2>] [HIERARCHICAL] [INJECTIVE] [IS_OBJECT_ID])
PRIMARY KEY <key1_name>, <key2_name>
SOURCE(<SOURCE_TYPE>([<param1_name> <param1_value> ... <paramN_name> <paramN_value>]))
LAYOUT(<LAYOUT_NAME>([<param_name> <param_value>]))
LIFETIME({MIN <min_value> MAX <max_value> | <max_value>})
[SETTINGS(<setting_name> = <setting_value>, <setting_name> = <setting_value>, ...)]
[COMMENT '<comment_text>']
```

where:

- `PRIMARY KEY` — dictionary key;
- `SOURCE` — data source for a dictionary;
- `LAYOUT` — way to store a dictionary in memory;
- `LIFETIME` — frequency of dictionary updates;
- `SETTINGS` — additional settings that can be specified for a dictionary if its data source is a local file, HTTP(s) resource, or ClickHouse table;
- `COMMENT` — text comment.

Dictionary key and attributes

The dictionary structure is defined by a key and attributes specified as parameters of the `CREATE DICTIONARY` query.

- A **key** is a column in the dictionary source with unique values that are used to identify records and search for data values.

  ADQM supports the following types of keys:

  - Numeric key — a column of the UInt64 type.

    ```
    CREATE DICTIONARY <dictionary_name> (
        <key_name> UInt64,
        ...
        )
    PRIMARY KEY <key_name>
    ...
    ```

    where `<key_name>` is a name of a column with keys.

  - Composite key is a tuple of columns of arbitrary types. It can consist of a single element (for example, you can use a string as a key).

    ```
    CREATE DICTIONARY <dictionary_name> (
        <key1_name> <key1_data_type>,
        <key2_name> <key2_data_type>
        ...
        )
    PRIMARY KEY <key1_name>, <key2_name>
    ...
    ```

    where `<key1_name>`, `<key2_name>` are names of columns with keys, and `<key1_data_type>`, `<key2_data_type>` are key types.

- **Attributes** are columns in the dictionary source that store data values to be retrieved by a key. An attribute is described as follows:

```
CREATE DICTIONARY <dictionary_name> (
    ...
    <attribute_name> <attribute_data_type> [DEFAULT <default_value>] [EXPRESSION <expr>] [HIERARCHICAL] [INJECTIVE] [IS_OBJECT_ID]
    )
...
```

Parameters and basic clauses

<attribute_name>	Name of a column in the data source
<attribute_data_type>	Data type to which a value from the dictionary should be cast. Supported types: UInt8, UInt16, UInt32, UInt64, Int8, Int16, Int32, Int64, Float32, Float64, UUID, Decimal32, Decimal64, Decimal128, Decimal256, String, Array. The ability to specify Nullable types for attributes depends on how the dictionary is stored in memory (for example, Nullable types are currently not supported for attributes of `IP_TRIE` dictionaries)
DEFAULT	Default value for an element not found by key. NULL can only be specified if the attribute type is Nullable
EXPRESSION	Expression executed on a value
HIERARCHICAL	Indicates that the attribute contains a value of a parent key for the current key (use it to configure a hierarchical dictionary)
INJECTIVE	Indicates that the attribute is injective (different keys correspond to different values of the attribute). In this case, if `GROUP BY` uses a function that retrieves the attribute value by key, this function is automatically removed from `GROUP BY`. This can optimize the data aggregation process
IS_OBJECT_ID	Indicates that the query is executed for a MongoDB document by `ObjectID`

Dictionary source

Use the `SOURCE` clause in the `CREATE DICTIONARY` query to specify a data source for a dictionary:

```
CREATE DICTIONARY <dictionary_name> (...)
...
SOURCE(<SOURCE_TYPE>([<param1_name> <param1_value> ... <paramN_name> <paramN_value>]))
...
```

where `<SOURCE_TYPE>` is a data source type, `<paramX_name> <paramX_value>` — source settings.

Possible values of <SOURCE_TYPE>

Value	Data source
FILE	Local file (a source file should be located in the *user_files* folder)
HTTP	HTTP resource
MYSQL	Table from a MySQL database
CLICKHOUSE	Table from a ClickHouse database
MONGODB	Collection from a MongoDB database
REDIS	Redis database
CASSANDRA	Column family (table) from a Cassandra keyspace (database)
POSTGRESQL	Table from a PostgreSQL database (see the example in the Integration between ADQM and ADPG article)
ODBC	Table from any database that has the ODBC driver

You can find examples of connecting dictionaries to various types of sources in the Dictionary Sources section of the ClickHouse documentation.

Store dictionaries in memory

Use the `LAYOUT` clause in the `CREATE DICTIONARY` query to specify how to store a dictionary in memory:

```
CREATE DICTIONARY <dictionary_name> (...)
...
LAYOUT(<LAYOUT_TYPE>([<param_name> <param_value>]))
...
```

where `<LAYOUT_TYPE>` is a way to store a dictionary in memory.

Possible values of <LAYOUT_TYPE>

Value	Way to store a dictionary
FLAT	A dictionary is completely stored in RAM as flat arrays
HASHED, COMPLEX_KEY_HASHED	A dictionary is completely stored in RAM as hash tables
SPARSE_HASHED, COMPLEX_KEY_SPARSE_HASHED	A dictionary is completely stored in RAM as hash tables, similar to `HASHED` / `COMPLEX_KEY_HASHED`, but uses less memory in favor higher CPU usage
HASHED_ARRAY, COMPLEX_KEY_HASHED_ARRAY	A dictionary is completely stored in RAM. Each attribute is stored in an array. The key is stored as a hash table where a value is an index in an array of attributes
RANGE_HASHED, COMPLEX_KEY_RANGE_HASHED	The dictionary is stored in RAM as a hash table with an ordered array of ranges and their corresponding values
CACHE, COMPLEX_KEY_CACHE	A dictionary is stored in a cache that has a fixed number of cells containing frequently used elements
SSD_CACHE, COMPLEX_KEY_SSD_CACHE	A dictionary is stored in a cache, similar to `SSD_CACHE` / `COMPLEX_KEY_SSD_CACHE`, but data is stored on SSD and index is in RAM
DIRECT, COMPLEX_KEY_DIRECT	A dictionary does not store data locally and interacts with the source while processing a query
IP_TRIE	This type of dictionary storage is for retrieving metadata (such as `AS` number or country code) from network prefixes (IP addresses). A dictionary should have a composite key that contains one element of the String type — IP address

Layouts with the `COMPLEX_KEY` prefix are used for dictionaries with composite keys.

For more details on all ways of storing dictionaries in memory, refer to the Storing Dictionaries in Memory section of the ClickHouse documentation.

Dictionary updates

Use the `LIFETIME` clause in the `CREATE DICTIONARY` query to specify a time interval for dictionary updates in seconds.

```
CREATE DICTIONARY <dictionary_name> (...)
...
LIFETIME(<max_value> | MIN <min_value> MAX <max_value>)
...
```

where:

- `<max_value>` is a dictionary update interval. If you set `LIFETIME(0)`, dictionary updates are disabled.
- `MIN <min_value> MAX <max_value>` is an interval within which the time to update the dictionary is randomly selected. This setting can be useful to distribute the load on the dictionary source when updating on a large number of servers. If you set `MIN 0 MAX 0`, the dictionary is not reloaded by timeout. In this case, the dictionary data can be reloaded with the SYSTEM RELOAD DICTIONARY command.

Dictionary updates (except for loading on the first use) do not block queries — the old version of a dictionary is used during an update. If an error occurs during an update, the error is written to the server log, and queries continue using the old version of the dictionary.

The dictionary update logic depends on the data source type and how the dictionary is stored in memory — read the Dictionary Updates section of the ClickHouse documentation for details.

Example

1. Create a `roles_dict` table to be a data source for a dictionary:

```
CREATE TABLE roles_dict (`id` Int32, `role` String) ENGINE = MergeTree() ORDER BY id;
```

```
INSERT INTO roles_dict VALUES (10, 'admin'), (20, 'owner'), (30, 'author'), (40, 'reviewer'), (50, 'editor'), (60, 'view only');
```

```
--id┬─role-------
│ 10 │ admin     │
│ 20 │ owner     │
│ 30 │ author    │
│ 40 │ reviewer  │
│ 50 │ editor    │
│ 60 │ view only │
------------------
```

2. Create a dictionary that reads data from the `roles_dict` table:

```
CREATE DICTIONARY roles_dict_clickhouse (`id` Int32, `role` String DEFAULT 'no role assigned')
PRIMARY KEY id
SOURCE(CLICKHOUSE(HOST 'localhost' USER 'default' PASSWORD '' DATABASE 'default' TABLE 'roles_dict'))
LIFETIME(MIN 10 MAX 20)
LAYOUT(FLAT());
```

View information on dictionaries

You can obtain information about dictionaries configured on the server from the system.dictionaries system table:

```
SELECT * FROM system.dictionaries WHERE name = '<dictionary_name>' [FORMAT <format_name>];
```

Example

Query:

```
SELECT * FROM system.dictionaries WHERE name = 'roles_dict_clickhouse' FORMAT Vertical;
```

If a dictionary is configured but not loaded, the output can look like:

```
Row 1:
──────
database:                           default
name:                               roles_dict_clickhouse
uuid:                               4e28febc-b472-423c-bc14-bd29f195390f
status:                             NOT_LOADED
origin:                             4e28febc-b472-423c-bc14-bd29f195390f
type:
key.names:                          ['id']
key.types:                          ['UInt64']
attribute.names:                    ['role']
attribute.types:                    ['String']
bytes_allocated:                    0
hierarchical_index_bytes_allocated: 0
query_count:                        0
hit_rate:                           0
found_rate:                         0
element_count:                      0
load_factor:                        0
source:
lifetime_min:                       0
lifetime_max:                       0
loading_start_time:                 1970-01-01 00:00:00
last_successful_update_time:        1970-01-01 00:00:00
loading_duration:                   0
last_exception:
comment:
```

You can view names of all dictionaries that exist in the specified database with the SHOW DICTIONARIES query:

```
SHOW DICTIONARIES FROM <database_name>;
```

Load dictionaries

The dictionaries_lazy_load server configuration parameter defines whether dictionaries should be loaded on server startup or on the first use of a dictionary.

To force reloading of all dictionaries at once or a single dictionary, execute the corresponding query:

```
SYSTEM RELOAD DICTIONARIES;
```

```
SYSTEM RELOAD DICTIONARY <dictionary_name>;
```

After a dictionary is loaded, the system.dictionaries table changes the dictionary status to `LOADED`, and provides values of the dictionary configuration parameters and various metrics (such as the amount of RAM allocated or the number of queries since the dictionary was successfully loaded). For example:

```
SELECT * FROM system.dictionaries WHERE name = 'roles_dict_clickhouse' FORMAT Vertical;
```

```
Row 1:
──────
database:                           default
name:                               roles_dict_clickhouse
uuid:                               4e28febc-b472-423c-bc14-bd29f195390f
status:                             LOADED
origin:                             4e28febc-b472-423c-bc14-bd29f195390f
type:                               Flat
key.names:                          ['id']
key.types:                          ['UInt64']
attribute.names:                    ['role']
attribute.types:                    ['String']
bytes_allocated:                    21048
hierarchical_index_bytes_allocated: 0
query_count:                        0
hit_rate:                           1
found_rate:                         0
element_count:                      6
load_factor:                        0.005859375
source:                             ClickHouse: default.roles_dict
lifetime_min:                       10
lifetime_max:                       20
loading_start_time:                 2023-03-14 11:05:13
last_successful_update_time:        2023-03-14 11:05:13
loading_duration:                   0.001
last_exception:
comment:
```

Use dictionaries

ADQM supports a set of special functions to work with dictionaries.

dictGet, dictGetOrDefault, dictGetOrNull

The `dictGet`, `dictGetOrDefault`, and `dictGetOrNull` functions retrieve values from a dictionary.

```
dictGet('<dictionary_name>', <attr_name>, <id_expr>)
dictGetOrDefault('<dictionary_name>', <attr_name>, <id_expr>, <default_value_expr>)
dictGetOrNull('<dictionary_name>', <attr_name>, <id_expr>)
```

Arguments

<dictionary_name>	Name of a dictionary. Specify it as a string literal
<attr_name>	Name of a column in the dictionary source (string literal) or a tuple of column names (tuple)
<id_expr>	Key value. Specify it as an expression returning a value of the UInt64 type or a tuple of expressions (tuple), depending on whether the dictionary key is numeric or composite
<default_value_expr>	Value returned if the dictionary does not contain a row with the `<id_expr>` key. Specify it as an expression returning a value of the data type configured for the `<attr_name>` attribute, or a tuple of expressions

Returned value:

- value of the dictionary attribute corresponding to the `<id_expr>` key, if ADQM is able to cast this value to the attribute's data type specified in the dictionary configuration;
- if the `<id_expr>` key is not found in the dictionary:
  - `dictGet` returns the value specified for the attribute via the `DEFAULT` clause in the dictionary configuration;
  - `dictGetOrDefault` returns the `<default_value_expr>` parameter value;
  - `dictGetOrNull` returns NULL.

For example, the following query returns the role with the `20` identifier from the `roles_dict_clickhouse` dictionary:

```
SELECT dictGet('roles_dict_clickhouse', 'role', 20);
```

```
┌─dictGet('roles_dict_clickhouse', 'role', 20)─┐
│ owner                                        │
└──────────────────────────────────────────────┘
```

ADQM also supports specialized functions that cast dictionary attribute values to a specific data type regardless of the dictionary configuration:

- `dictGetInt8`, `dictGetInt16`, `dictGetInt32`, `dictGetInt64`;
- `dictGetUInt8`, `dictGetUInt16`, `dictGetUInt32`, `dictGetUInt64`;
- `dictGetFloat32`, `dictGetFloat64`;
- `dictGetDate`;
- `dictGetDateTime`;
- `dictGetUUID`;
- `dictGetString`.

The syntax of these functions is the same as `dictGet` syntax described above. You can use any of these functions with the `OrDefault` suffix (for example, `dictGetInt8OrDefault`) and specify the `<default_value_expr>` parameter value to be returned if the key is not found in the dictionary.

dictHas

The `dictHas` function checks whether the specified key exists in a dictionary.

```
dictHas('<dictionary_name>', <id_expr>)
```

Arguments

<dictionary_name>	Name of a dictionary. Specify it as a string literal
<id_expr>	Key value. Specify it as an expression returning a value of the UInt64 type or a tuple of expressions (tuple), depending on whether the dictionary key is numeric or composite

The function returns:

- `0` — if the key is not found in the dictionary;
- `1` — if the key exists.

For example, the following query checks if the `roles_dict_clickhouse` dictionary contains a role with the `100` identifier:

```
SELECT dictHas('roles_dict_clickhouse', 100);
```

```
┌─dictHas('roles_dict_clickhouse', 100)─┐
│                                     0 │
└───────────────────────────────────────┘
```

Example

1. Create the `users` table with users and identifiers of their assigned roles:

```
CREATE TABLE users (user_id Int32, name String, role_id Int32) ENGINE = MergeTree ORDER BY user_id;
```

```
INSERT INTO users VALUES (1, 'john', 10), (2, 'mary', 30), (3, 'andrew', 40), (4, 'harry', 70), (5, 'ann', 50);
```

```
--user_id---name-----role_id--
│       1 │ john   │      10 │
│       2 │ mary   │      30 │
│       3 │ andrew │      40 │
│       4 │ harry  │      70 │
│       5 │ ann    │      50 │
------------------------------
```

2. Run the following query to read usernames from the `users` table and find their role names in the `roles_dict_clickhouse` dictionary by `role_id`:

```
SELECT  name AS user, dictGet('roles_dict_clickhouse', 'role', toUInt64(role_id)) AS role FROM users;
```

Result:

```
--user-----role--------------
│ john   │ admin            │
│ mary   │ author           │
│ andrew │ reviewer         │
│ harry  │ no role assigned │
│ ann    │ editor           │
-----------------------------
```

Detach and attach a dictionary

You can detach a dictionary with the `DETACH DICTIONARY` query:

```
DETACH DICTIONARY [IF EXISTS] [<database_name.]<dictionary_name> [ON CLUSTER <cluster_name>] [PERMANENTLY];
```

When this query is executed, the dictionary becomes "invisible" to the server, but its metadata is not deleted. If the dictionary was not detached permanently (the `PERMANENTLY` clause was not used in the query), the next time the server starts, it will read the dictionary metadata and "see" the dictionary again. If a dictionary has been detached permanently, the server will not automatically re-enable it.

You can re-attach a dictionary with the `ATTACH` query regardless of whether the dictionary was detached permanently or not:

```
ATTACH DICTIONARY  [IF NOT EXISTS] [<database_name.]<dictionary_name> [ON CLUSTER <cluster_name>];
```

Delete a dictionary

To delete a dictionary, use the `DROP DICTIONARY` query:

```
DROP DICTIONARY [IF EXISTS] [<database_name>.]<dictionary_name> [ON CLUSTER <cluster_name>];
```

Materialized views

Elena Dvoryadkina

Contents

To Table of Contents

- Create materialized view
  - Syntax overview
  - Example
- Modify materialized view
  - Example

A **materialized view** calculates intermediate aggregate states for data from a source table according to the `SELECT` query, which is specified in the view definition, and saves the results to its internal table or to a separate target table. Aggregate states in the view are updated automatically each time the source table receives new data. Materialized views allow you to optimize the execution time of complex queries with aggregates that access a large number of data records — precomputation of totals reduces the amount of data needed to be processed for `SELECT` query execution.

Create materialized view

Syntax overview

To create a materialized view, use the `CREATE MATERIALIZED VIEW` query. The basic syntax is:

```
CREATE MATERIALIZED VIEW [IF NOT EXISTS] [<db_name>.]<mat_view_name> [ON CLUSTER <cluster_name>]
[TO [<db_name>.]<target_table>]
[ENGINE = <engine>]
[POPULATE]
AS SELECT
    ...
FROM [<db_name>.]<source_table>
...;
```

Below are clauses used to define a materialized view.

TO	Target table to which the materialized view should forward the data
ENGINE	Engine of the materialized view's internal table where aggregated data should be stored if a separate target table has not been explicitly specified in the materialized view definition via the `TO` clause
POPULATE	Indicates that data existing in the source table when the materialized view is being created should be automatically loaded into the view. It is not recommended to use `POPULATE` as data that arrives into the source table during the view creation will be missed and not inserted into the view. Also, the `POPULATE` keyword should be omitted if the materialized view definition includes the `TO` clause — in this case, the view will only process and store data added to the source table after the view has been created
AS SELECT … FROM …	Query that reads new data inserted into the source table and determines how to transform that data for loading into the materialized view (or into the target table, if any). The `SELECT` query can contain `DISTINCT`, `GROUP BY`, `ORDER BY`, `LIMIT` — in this case, the corresponding data conversions are performed independently for each inserted data block. For example, if `GROUP BY` is set, aggregation is performed during insertion within inserted data block only, and data will not be further aggregated. The exception is if you specify a materialized view's table engine (using the `ENGINE` clause), which performs data aggregation on its own (for example, `SummingMergeTree`). You can change the `SELECT` query after the materialized view has been created using the ALTER TABLE … MODIFY QUERY query. However, note that this query type is currently experimental and has some limitations

> **NOTE**
>
> Any changes to data existing in the source table (such as update, delete, etc.) do not affect the materialized view.

In ADQM, materialized views look the same as normal tables. For example, they are listed in the result of the `SHOW TABLES` query.

To delete a materialized view, run the `DROP VIEW` or `DROP TABLE` query.

Example

This example shows how to create a materialized view to pre-aggregate data and speed up subsequent `SELECT` query executions.

1. Create a source table to store pageview data:

```
CREATE TABLE page_views
(
    time DateTime,
    page_name String,
    time_on_page UInt64
)
ENGINE = MergeTree
ORDER BY time;
```

2. Create a target table to store pageview counts and average pageview time by month:

```
CREATE TABLE page_views_monthly
(
    page_name String,
    month Date,
    page_view_count UInt64,
    avg_time_on_page AggregateFunction(avg, UInt64)
)
ENGINE = SummingMergeTree()
ORDER BY (page_name, month);
```

A column with average pageview time is of the AggregateFunction data type that allows storing partially aggregated data. The SummingMergeTree table engine used in the example requires this data type for all aggregates except `sum` or `count`. To work with materialized views, you can also use other table engines. For example, use the AggregatingMergeTree engine that works with aggregate functions only (this means that if you set the AggregatingMergeTree engine in the query above, the `page_view_count` column should also be of the AggregateFunction type).

3. Create a materialized view:

```
CREATE MATERIALIZED VIEW page_views_monthly_mv
TO page_views_monthly
AS SELECT
    toDate(toStartOfMonth(time)) AS month,
    page_name,
    count() as page_view_count,
    avgState(time_on_page) AS avg_time_on_page
FROM page_views
GROUP BY page_name, month
ORDER BY page_name, month;
```

When data is inserted into the `page_views` source table, the materialized view will transform it as specified in the `SELECT` definition and send the result to the `page_views_monthly` target table. To insert intermediate aggregate states into a target table's `avg_time_on_page` column of the AggregateFunction type, use the appropriate aggregate function with the `State` suffix (`avgState`) in the view's `SELECT` definition.

If you run the `SHOW TABLES` query at this point, the output will include the created tables and materialized view:

```
┌─name──────────────────┐
│ page_views            │
│ page_views_monthly    │
│ page_views_monthly_mv │
└───────────────────────┘
```

4. Populate the source table with test data. For example, randomly generate 1 million views for 4 pages:

```
INSERT INTO page_views SELECT
    toDate('2023-01-01 00:00:00') + (rand() % 182) AS time,
    ['landing', 'products', 'docs', 'about'][toInt32(randBinomial(4, 0.01)) + 1],
    rand() % 15
FROM numbers(1000000);
```

5. Compare the speed of the `SELECT` query execution when fetching data from the source table with raw input data and from the table with pre-aggregated monthly data.

Select data from the `page_views` table:

```
SELECT
    page_name,
    toDate(toStartOfMonth(time)) AS month,
    count() AS page_view_count,
    round(avg(time_on_page), 2) AS avg_time_on_page
FROM page_views
GROUP BY page_name, month
ORDER BY page_name, month;
```

```
┌─page_name─┬──────month─┬─page_view_count─┬─avg_time_on_page─┐
│ about     │ 2023-02-01 │               2 │                7 │
│ about     │ 2023-03-01 │               1 │                7 │
│ docs      │ 2023-01-01 │              95 │             7.21 │
│ docs      │ 2023-02-01 │              93 │             6.42 │
│ docs      │ 2023-03-01 │             102 │             6.72 │
│ docs      │ 2023-04-01 │              84 │             7.35 │
│ docs      │ 2023-05-01 │             101 │             7.18 │
│ docs      │ 2023-06-01 │              92 │             7.72 │
│ docs      │ 2023-07-01 │               5 │              8.6 │
│ landing   │ 2023-01-01 │          163242 │             7.01 │
│ landing   │ 2023-02-01 │          147695 │                7 │
│ landing   │ 2023-03-01 │          163930 │             6.99 │
│ landing   │ 2023-04-01 │          158155 │             7.01 │
│ landing   │ 2023-05-01 │          163263 │             6.99 │
│ landing   │ 2023-06-01 │          158840 │                7 │
│ landing   │ 2023-07-01 │            5396 │             7.02 │
│ products  │ 2023-01-01 │            6667 │             7.05 │
│ products  │ 2023-02-01 │            6099 │             6.99 │
│ products  │ 2023-03-01 │            6589 │             7.02 │
│ products  │ 2023-04-01 │            6391 │             7.01 │
│ products  │ 2023-05-01 │            6428 │             6.98 │
│ products  │ 2023-06-01 │            6517 │             7.04 │
│ products  │ 2023-07-01 │             213 │             7.48 │
└───────────┴────────────┴─────────────────┴──────────────────┘

23 rows in set. Elapsed: 0.023 sec. Processed 1.00 million rows, 28.04 MB (43.43 million rows/s., 1.22 GB/s.)
```

Select data from the `page_views_monthly` table — reading from this table will be 7 times faster than from the source table (with more data for selection, the difference will be even more noticeable). Note that the `SELECT` query uses the same aggregate function as for data insertion, but with the `Merge` suffix — it is required to combine a set of intermediate aggregate states and return the result of complete data aggregation (see AggregateFunction):

```
SELECT
    page_name,
    month,
    sum(page_view_count) AS page_view_count,
    round(avgMerge(avg_time_on_page), 2) AS avg_time_on_page
FROM page_views_monthly
GROUP BY page_name, month
ORDER BY page_name, month;
```

```
┌─page_name─┬──────month─┬─page_view_count─┬─avg_time_on_page─┐
│ about     │ 2023-02-01 │               2 │                7 │
│ about     │ 2023-03-01 │               1 │                7 │
│ docs      │ 2023-01-01 │              95 │             7.21 │
│ docs      │ 2023-02-01 │              93 │             6.42 │
│ docs      │ 2023-03-01 │             102 │             6.72 │
│ docs      │ 2023-04-01 │              84 │             7.35 │
│ docs      │ 2023-05-01 │             101 │             7.18 │
│ docs      │ 2023-06-01 │              92 │             7.72 │
│ docs      │ 2023-07-01 │               5 │              8.6 │
│ landing   │ 2023-01-01 │          163242 │             7.01 │
│ landing   │ 2023-02-01 │          147695 │                7 │
│ landing   │ 2023-03-01 │          163930 │             6.99 │
│ landing   │ 2023-04-01 │          158155 │             7.01 │
│ landing   │ 2023-05-01 │          163263 │             6.99 │
│ landing   │ 2023-06-01 │          158840 │                7 │
│ landing   │ 2023-07-01 │            5396 │             7.02 │
│ products  │ 2023-01-01 │            6667 │             7.05 │
│ products  │ 2023-02-01 │            6099 │             6.99 │
│ products  │ 2023-03-01 │            6589 │             7.02 │
│ products  │ 2023-04-01 │            6391 │             7.01 │
│ products  │ 2023-05-01 │            6428 │             6.98 │
│ products  │ 2023-06-01 │            6517 │             7.04 │
│ products  │ 2023-07-01 │             213 │             7.48 │
└───────────┴────────────┴─────────────────┴──────────────────┘

23 rows in set. Elapsed: 0.003 sec.
```

Modify materialized view

Use the `ALTER TABLE…MODIFY QUERY` clause to modify a `SELECT` query that was specified when a materialized view was created. This command is not recommended for materialized views without target tables assigned via the `TO` clause as it does not change the structure and column definitions of a materialized view's internal table.

Before executing the `ALTER TABLE…MODIFY QUERY` query, enable the option: `SET allow_experimental_alter_materialized_view_structure=1`.

Example

1. Use the ALTER TABLE…ADD COLUMN clause to add the `browser` column to the `page_views` source table and to the `page_views_monthly` target table (columns of a materialized view and its target table do not have to match, so changing the target table's structure will not break data insertion):

```
ALTER TABLE page_views ADD COLUMN browser String;
```

```
ALTER TABLE page_views_monthly ADD COLUMN browser String after month,
MODIFY ORDER BY (page_name, month, browser);
```

2. Insert data into the source table:

```
INSERT INTO page_views SELECT
    toDate('2023-01-01 00:00:00') + (rand() % 182) AS time,
    ['landing', 'products', 'docs', 'about'][toInt32(randBinomial(4, 0.01)) + 1],
    rand() % 15,
    ['firefox', 'safary', 'chrome'][(number % 3) + 1]
FROM numbers(1000000);
```

If you read data from the target table at this stage, you will see that the `browser` column is empty:

```
SELECT
    page_name,
    month,
    browser,
    sum(page_view_count) AS page_view_count,
    round(avgMerge(avg_time_on_page), 2) AS avg_time_on_page
FROM page_views_monthly
GROUP BY page_name, month, browser
ORDER BY page_name, month ASC, browser;
```

```
┌─page_name─┬──────month─┬─browser─┬─page_view_count─┬─avg_time_on_page─┐
│ about     │ 2023-01-01 │         │               1 │               14 │
│ about     │ 2023-02-01 │         │               3 │             6.67 │
│ about     │ 2023-03-01 │         │               2 │              8.5 │
│ about     │ 2023-05-01 │         │               1 │               12 │
│ docs      │ 2023-01-01 │         │             203 │             7.13 │
│ docs      │ 2023-02-01 │         │             175 │             6.91 │
│ docs      │ 2023-03-01 │         │             185 │             7.21 │
│ docs      │ 2023-04-01 │         │             175 │             7.15 │
│ docs      │ 2023-05-01 │         │             182 │             7.53 │
│ docs      │ 2023-06-01 │         │             190 │             7.94 │
│ docs      │ 2023-07-01 │         │               8 │             9.88 │
│ landing   │ 2023-01-01 │         │          326986 │             7.01 │
│ landing   │ 2023-02-01 │         │          295510 │             6.99 │
│ landing   │ 2023-03-01 │         │          327021 │                7 │
│ landing   │ 2023-04-01 │         │          316761 │                7 │
│ landing   │ 2023-05-01 │         │          327207 │                7 │
│ landing   │ 2023-06-01 │         │          316732 │             6.99 │
│ landing   │ 2023-07-01 │         │           10702 │             7.05 │
│ products  │ 2023-01-01 │         │           13224 │             7.07 │
│ products  │ 2023-02-01 │         │           12103 │             7.02 │
│ products  │ 2023-03-01 │         │           13225 │             6.99 │
│ products  │ 2023-04-01 │         │           12807 │             6.98 │
│ products  │ 2023-05-01 │         │           13156 │             6.95 │
│ products  │ 2023-06-01 │         │           13000 │             7.01 │
│ products  │ 2023-07-01 │         │             441 │             6.95 │
└───────────┴────────────┴─────────┴─────────────────┴──────────────────┘
```

3. To send data from the new column of the source table to the corresponding column of the target table, you need to change the `SELECT` query of the materialized view. To do this, run the `ALTER TABLE…MODIFY QUERY` command:

```
SET allow_experimental_alter_materialized_view_structure=1;
```

```
ALTER TABLE page_views_monthly_mv
MODIFY QUERY
SELECT
    toDate(toStartOfMonth(time)) AS month,
    page_name,
    browser,
    count() AS page_view_count,
    avgState(time_on_page) AS avg_time_on_page
FROM page_views
GROUP BY page_name, month, browser
ORDER BY page_name, month, browser;
```

4. Add data to the source table again:

```
INSERT INTO page_views SELECT
    toDate('2023-01-01 00:00:00') + (rand() % 182) AS time,
    ['landing', 'products', 'docs', 'about'][toInt32(randBinomial(4, 0.01)) + 1],
    rand() % 15,
    ['firefox', 'safary', 'chrome'][(number % 3) + 1]
FROM numbers(1000000);
```

Make sure the data from the `browser` column is now inserted into the target table:

```
SELECT
    page_name,
    month,
    browser,
    sum(page_view_count) AS page_view_count,
    round(avgMerge(avg_time_on_page), 2) AS avg_time_on_page
FROM page_views_monthly
GROUP BY page_name, month, browser
ORDER BY page_name, month ASC, browser;
```

```
┌─page_name─┬──────month─┬─browser─┬─page_view_count─┬─avg_time_on_page─┐
│ about     │ 2023-01-01 │         │               1 │               14 │
│ about     │ 2023-02-01 │         │               3 │             6.67 │
│ about     │ 2023-02-01 │ safary  │               1 │                7 │
│ about     │ 2023-03-01 │         │               2 │              8.5 │
│ about     │ 2023-04-01 │ firefox │               1 │                1 │
│ about     │ 2023-05-01 │         │               1 │               12 │
│ about     │ 2023-05-01 │ chrome  │               1 │               14 │
│ about     │ 2023-05-01 │ safary  │               1 │                6 │
│ docs      │ 2023-01-01 │         │             203 │             7.13 │
│ docs      │ 2023-01-01 │ chrome  │              37 │             6.92 │
│ docs      │ 2023-01-01 │ firefox │              38 │             5.58 │
│ docs      │ 2023-01-01 │ safary  │              29 │             5.79 │
│ docs      │ 2023-02-01 │         │             175 │             6.91 │
│ docs      │ 2023-02-01 │ chrome  │              30 │             6.77 │
│ docs      │ 2023-02-01 │ firefox │              26 │             6.65 │
│ docs      │ 2023-02-01 │ safary  │              18 │             5.89 │
│ docs      │ 2023-03-01 │         │             185 │             7.21 │
│ docs      │ 2023-03-01 │ chrome  │              42 │             7.83 │
│ docs      │ 2023-03-01 │ firefox │              31 │             7.65 │
│ docs      │ 2023-03-01 │ safary  │              33 │             5.88 │
│ docs      │ 2023-04-01 │         │             175 │             7.15 │
│ docs      │ 2023-04-01 │ chrome  │              28 │             8.25 │
│ docs      │ 2023-04-01 │ firefox │              25 │             6.32 │
│ docs      │ 2023-04-01 │ safary  │              26 │             6.58 │
│ docs      │ 2023-05-01 │         │             182 │             7.53 │
│ docs      │ 2023-05-01 │ chrome  │              32 │             7.03 │
│ docs      │ 2023-05-01 │ firefox │              20 │             7.45 │
│ docs      │ 2023-05-01 │ safary  │              34 │             6.59 │
│ docs      │ 2023-06-01 │         │             190 │             7.94 │
│ docs      │ 2023-06-01 │ chrome  │              35 │             8.37 │
│ docs      │ 2023-06-01 │ firefox │              29 │             6.69 │
│ docs      │ 2023-06-01 │ safary  │              27 │             7.15 │
│ docs      │ 2023-07-01 │         │               8 │             9.88 │
│ docs      │ 2023-07-01 │ chrome  │               2 │                2 │
│ docs      │ 2023-07-01 │ firefox │               1 │                1 │
│ docs      │ 2023-07-01 │ safary  │               1 │                1 │
│ landing   │ 2023-01-01 │         │          326986 │             7.01 │
│ landing   │ 2023-01-01 │ chrome  │           54763 │             6.99 │
│ landing   │ 2023-01-01 │ firefox │           54665 │             7.02 │
│ landing   │ 2023-01-01 │ safary  │           54294 │                7 │
│ landing   │ 2023-02-01 │         │          295510 │             6.99 │
│ landing   │ 2023-02-01 │ chrome  │           48944 │             7.03 │
│ landing   │ 2023-02-01 │ firefox │           49031 │             7.03 │
│ landing   │ 2023-02-01 │ safary  │           49298 │                7 │
│ landing   │ 2023-03-01 │         │          327021 │                7 │
│ landing   │ 2023-03-01 │ chrome  │           54381 │             6.98 │
│ landing   │ 2023-03-01 │ firefox │           54433 │             7.03 │
│ landing   │ 2023-03-01 │ safary  │           54556 │             7.01 │
│ landing   │ 2023-04-01 │         │          316761 │                7 │
│ landing   │ 2023-04-01 │ chrome  │           52931 │             6.95 │
│ landing   │ 2023-04-01 │ firefox │           52706 │             6.99 │
│ landing   │ 2023-04-01 │ safary  │           53075 │             6.99 │
│ landing   │ 2023-05-01 │         │          327207 │                7 │
│ landing   │ 2023-05-01 │ chrome  │           54538 │             6.99 │
│ landing   │ 2023-05-01 │ firefox │           54733 │             7.02 │
│ landing   │ 2023-05-01 │ safary  │           54618 │             7.02 │
│ landing   │ 2023-06-01 │         │          316732 │             6.99 │
│ landing   │ 2023-06-01 │ chrome  │           52763 │             6.97 │
│ landing   │ 2023-06-01 │ firefox │           52955 │                7 │
│ landing   │ 2023-06-01 │ safary  │           52795 │             7.01 │
│ landing   │ 2023-07-01 │         │           10702 │             7.05 │
│ landing   │ 2023-07-01 │ chrome  │            1762 │             7.08 │
│ landing   │ 2023-07-01 │ firefox │            1747 │             6.98 │
│ landing   │ 2023-07-01 │ safary  │            1767 │             7.09 │
│ products  │ 2023-01-01 │         │           13224 │             7.07 │
│ products  │ 2023-01-01 │ chrome  │            2208 │             7.04 │
│ products  │ 2023-01-01 │ firefox │            2210 │             6.99 │
│ products  │ 2023-01-01 │ safary  │            2273 │             6.99 │
│ products  │ 2023-02-01 │         │           12103 │             7.02 │
│ products  │ 2023-02-01 │ chrome  │            1970 │              7.1 │
│ products  │ 2023-02-01 │ firefox │            1978 │             6.88 │
│ products  │ 2023-02-01 │ safary  │            1870 │             7.01 │
│ products  │ 2023-03-01 │         │           13225 │             6.99 │
│ products  │ 2023-03-01 │ chrome  │            2189 │             6.97 │
│ products  │ 2023-03-01 │ firefox │            2186 │             6.94 │
│ products  │ 2023-03-01 │ safary  │            2251 │             7.18 │
│ products  │ 2023-04-01 │         │           12807 │             6.98 │
│ products  │ 2023-04-01 │ chrome  │            2252 │             6.91 │
│ products  │ 2023-04-01 │ firefox │            2210 │              7.1 │
│ products  │ 2023-04-01 │ safary  │            2089 │             7.04 │
│ products  │ 2023-05-01 │         │           13156 │             6.95 │
│ products  │ 2023-05-01 │ chrome  │            2214 │                7 │
│ products  │ 2023-05-01 │ firefox │            2188 │             6.94 │
│ products  │ 2023-05-01 │ safary  │            2151 │             6.98 │
│ products  │ 2023-06-01 │         │           13000 │             7.01 │
│ products  │ 2023-06-01 │ chrome  │            2134 │             6.89 │
│ products  │ 2023-06-01 │ firefox │            2054 │             7.18 │
│ products  │ 2023-06-01 │ safary  │            2062 │             6.91 │
│ products  │ 2023-07-01 │         │             441 │             6.95 │
│ products  │ 2023-07-01 │ chrome  │              77 │             6.65 │
│ products  │ 2023-07-01 │ firefox │              67 │             7.27 │
│ products  │ 2023-07-01 │ safary  │              64 │             7.16 │
└───────────┴────────────┴─────────┴─────────────────┴──────────────────┘
```

Projections

Elena Dvoryadkina

Contents

To Table of Contents

- Overview
- Manage projections
  - Create/delete projections
  - Settings to enable projections
- View information on projections
- Examples
  - Filter without using a primary key
  - Pre-aggregate values
  - View information about projections

Overview

A **projection** is an additional hidden table that stores data from a source table in an alternative form to be optimal for executing some type of queries. A projection can store data rows sorted in a different order than the source table (and therefore have a different primary index) or pre-compute aggregate values across columns of the source table. So, this feature can be useful to optimize:

- queries with filters on source table columns that are not part of a primary key;
- queries that aggregate values of source table columns (data pre-aggregation performed by projections can reduce the consumption of both computing and I/O resources).

Projections are supported only for tables of the MergeTree family (including Replicated*). A table can have one or more projections — all of them will be automatically synchronized with the source table, i.e. updated each time new data is inserted into the source table.

Projections are similar to materialized views (which also allow pre-computing aggregations and using different row orders on data from another table), but they are defined at the level of data parts — for each data part, a projection of data in that part is created based on the projection query. This guarantees that projection data is consistent with the source table data and that ADQM can automatically choose and apply the most optimal projections when executing queries provided by users (a projection is considered optimal if it returns the correct result for a query with the least data to be scanned).

Unlike materialized views, projections do not require manual creation of separate additional tables with different structures, which you need to keep in mind and each time when building a query, you need to choose which table to send it to in order to ensure the high speed of the query execution. With projections, it is enough to send all queries to the source table — ADQM itself will determine whether the table has a projection that can speed up the query execution, and if such projection exists, it will redirect the query to it. In other words, a user does not need to know anything about projections.

CAUTION

As each projection creates a new hidden table (projection files are stored in the *<projection_name>.proj* subdirectories within directories of the source MergeTree table's data parts), it may require more I/O resources and disk space. For example, if a projection is created to define an alternate sorting key, all data from the source table will be duplicated in the projection.

Manage projections

Create/delete projections

You can specify a projection when creating a MergeTree table in the section with column descriptions using the `PROJECTION` clause. The basic syntax for creating a table with a projection is:

```
CREATE TABLE <table_name>
(   <column1> ...,
    <column2> ...,
    ...
    PROJECTION <projection_name> (SELECT <column_list> [ORDER BY <sorting_key>] [GROUP BY <grouping_key>])
)
ENGINE = MergeTree()
...;
```

After a projection name, there is a **projection query** (in parentheses), which specifies how data from the source table should be transformed for storing in the projection table. A projection query can include one of the following clauses:

- `ORDER BY` — key by which data rows will be sorted, and a primary key will be set in the projection;
- `GROUP BY` — key by which a source table data will be grouped for aggregation in the projection. In this case, a hidden projection table is based on the AggregatingMergeTree engine, and all aggregate functions are converted to AggregateFunction.

To manage projections of existing tables, you can use the `ALTER TABLE` queries listed below.

- Add a projection to an existing table:

  ```
  ALTER TABLE [<db_name.]<table_name> [ON CLUSTER <cluster_name>]
      ADD PROJECTION [IF NOT EXISTS] <projection_name> (SELECT <column_list> [ORDER BY <sorting_key>] [GROUP BY <grouping_key>]);
  ```

  This query adds a projection description to a MergeTree table metadata, but does not populate the projection with data (it will be rebuilt when new data is inserted into the source table or when materialized).

- Materialize a projection to fill it with existing data of the source table (you can select an individual partition):

  ```
  ALTER TABLE [<db_name.]<table_name> [ON CLUSTER <cluster_name>]
      MATERIALIZE PROJECTION [IF EXISTS] <projection_name> [IN PARTITION <partition_name>];
  ```

- Delete projection files from the disk without removing projection description from the source table's metadata (you can select an individual partition):

  ```
  ALTER TABLE [<db_name.]<table_name> [ON CLUSTER <cluster_name>]
      CLEAR PROJECTION [IF EXISTS] <projection_name> [IN PARTITION <partition_name>];
  ```

- Delete a projection description from the source table metadata and remove the projection files from the disk:

  ```
  ALTER TABLE [<db_name.]<table_name> [ON CLUSTER <cluster_name>] DROP PROJECTION [IF EXISTS] <partition_name>;
  ```

The `ADD PROJECTION`, `CLEAR PROJECTION`, and `DROP PROJECTION` commands are lightweight because they only change metadata or delete files.

Settings to enable projections

To tell ADQM whether it should use projections when executing queries, configure the following parameters (at the session level):

- `optimize_use_projections` — enables/disables automatic optimization of `SELECT` queries with projections (the default value is `1`, meaning the projection optimization is enabled);
- `force_optimize_projection` — enables/disables the obligatory use of projections for executing `SELECT` queries if the optimization via projections is enabled by the `optimize_use_projections` setting (the default value is `0`).

View information on projections

You can obtain information about projections from system tables:

- `system.projection_parts` — information about currently existing data parts for projections of all MergeTree tables in a cluster (information about the corresponding data parts of the source table is stored in columns prefixed with `parent_`);
- `system.projection_parts_columns` — information about columns of all currently existing projection data parts of all MergeTree tables.

Since projection tables are hidden, they are not included in the `SHOW TABLES` list.

Examples

Create a MergeTree table to store pageview data (the examples below will define projections for this table):

```
CREATE TABLE page_views (time DateTime, page_name String, time_on_page UInt64) ENGINE = MergeTree ORDER BY time;
```

Populate this table with test data. For example, randomly generate 1 million views for 4 pages over approximately the last 3 years relative to the current moment:

```
INSERT INTO page_views SELECT
    now() - toIntervalMinute(rand() % 500000 + number),
    ['landing', 'products', 'docs', 'about'][toInt32(randBinomial(4, 0.01)) + 1],
    rand() % 15
FROM numbers(1000000);
```

Filter without using a primary key

If you run a query filtering data by the `time` column, which is included in the primary key of the `page_views` table, a full table scan is not required:

```
SELECT count() FROM page_views WHERE toYear(time) = '2022';
```

```
   ┌─count()─┐
1. │  388601 │
   └─────────┘

1 row in set. Elapsed: 0.004 sec. Processed 393.22 thousand rows, 1.57 MB (99.18 million rows/s., 396.72 MB/s.)
```

To filter data not by a key column, ADQM performs a full table scan, for example:

```
SELECT count() FROM page_views WHERE page_name = 'products';
```

```
   ┌─count()─┐
1. │   38729 │
   └─────────┘

1 row in set. Elapsed: 0.006 sec. Processed 1.00 million rows, 16.04 MB (156.85 million rows/s., 2.52 GB/s.)
```

To optimize such queries, you can add a projection with a different primary index to the table. For example, create the `page_name_projection` projection which sorts rows by the `page_name` column:

```
ALTER TABLE page_views ADD PROJECTION page_name_projection (SELECT * ORDER BY page_name);
```

Materialize the projection to build it based on data currently existing in the `page_views` table:

```
ALTER TABLE page_views MATERIALIZE PROJECTION page_name_projection;
```

Repeat the query filtering by the `page_name` column:

```
SELECT count() FROM page_views WHERE page_name = 'products';
```

Now, ADQM has read 41.54 thousand rows instead of 1 million to execute the query:

```
   ┌─count()─┐
1. │   38729 │
   └─────────┘

1 row in set. Elapsed: 0.003 sec. Processed 41.54 thousand rows, 703.30 KB (12.48 million rows/s., 211.24 MB/s.)
```

Use the EXPLAIN clause to verify that ADQM uses the projection (`ReadFromMergeTree (page_name_projection)`) to execute the query:

```
EXPLAIN SELECT count() FROM page_views WHERE page_name = 'products';
```

```
   ┌─explain────────────────────────────────────┐
1. │ Expression ((Project names + Projection))  │
2. │   Aggregating                              │
3. │     Filter                                 │
4. │       ReadFromMergeTree (page_name_projection) │
   └────────────────────────────────────────────┘
```

Pre-aggregate values

Run a query that returns the number of pageviews and average pageview time by month:

```
SELECT
    page_name,
    toStartOfMonth(time) AS month,
    count() AS page_view_count,
    round(avg(time_on_page), 2) AS avg_time_on_page
FROM page_views
GROUP BY page_name, month
ORDER BY page_name, month;
```

To execute this query, ADQM reads all rows of the `page_views` table:

```
Query id: 303db76e-1e68-4454-91ea-a951c36d170b

     ┌─page_name─┬──────month─┬─page_view_count─┬─avg_time_on_page─┐
  1. │ about     │ 2022-03-01 │               1 │                9 │
  2. │ about     │ 2023-06-01 │               1 │                6 │

      ...

105. │ products  │ 2024-06-01 │             313 │             6.87 │
106. │ products  │ 2024-07-01 │             116 │             6.26 │
107. │ products  │ 2024-08-01 │               7 │                7 │
     └───────────┴────────────┴─────────────────┴──────────────────┘

107 rows in set. Elapsed: 0.018 sec. Processed 1.00 million rows, 28.04 MB (54.22 million rows/s., 1.52 GB/s.)
```

Create and materialize a projection that calculates the number of views and average view time for each page in the `page_views` table by month:

```
ALTER TABLE page_views ADD PROJECTION page_views_monthly_projection
    (
        SELECT
            page_name,
            toStartOfMonth(time),
            count(),
            round(avg(time_on_page), 2)
        GROUP BY
            page_name,
            toStartOfMonth(time)
    );
```

```
ALTER TABLE page_views MATERIALIZE PROJECTION page_views_monthly_projection;
```

Repeat the query:

```
SELECT
    page_name,
    toStartOfMonth(time) AS month,
    count() AS page_view_count,
    round(avg(time_on_page), 2) AS avg_time_on_page
FROM page_views
GROUP BY page_name, month
ORDER BY page_name, month;
```

The result is the same, but the performance is now better:

```
Query id: acb9a600-cbd7-44a4-aa61-ee5f8cdd941d

     ┌─page_name─┬──────month─┬─page_view_count─┬─avg_time_on_page─┐
  1. │ about     │ 2022-03-01 │               1 │                9 │
  2. │ about     │ 2023-06-01 │               1 │                6 │

      ...

105. │ products  │ 2024-06-01 │             313 │             6.87 │
106. │ products  │ 2024-07-01 │             116 │             6.26 │
107. │ products  │ 2024-08-01 │               7 │                7 │
     └───────────┴────────────┴─────────────────┴──────────────────┘

107 rows in set. Elapsed: 0.004 sec.
```

Use `EXPLAIN` to verify that ADQM now uses the projection to execute the query:

```
EXPLAIN SELECT
    page_name,
    toStartOfMonth(time) AS month,
    count() AS page_view_count,
    round(avg(time_on_page), 2) AS avg_time_on_page
FROM page_views
GROUP BY page_name, month
ORDER BY page_name, month;
```

```
   ┌─explain──────────────────────────────────────────────────────────────────┐
1. │ Expression ((Project names + (Before ORDER BY + Projection) [lifted up part])) │
2. │   Sorting (Sorting for ORDER BY)                                         │
3. │     Expression ((Before ORDER BY + Projection))                          │
4. │       Aggregating                                                        │
5. │         Expression                                                       │
6. │           ReadFromMergeTree (page_views_monthly_projection)              │
   └──────────────────────────────────────────────────────────────────────────┘
```

You can also inspect the `system.query_log` system table where the `projections` field shows the name of the used projection (or remains empty if no projection was used):

```
SELECT query, formatReadableQuantity(read_rows) AS read_rows, projections FROM system.query_log
WHERE type='QueryFinish' AND query_id='acb9a600-cbd7-44a4-aa61-ee5f8cdd941d';\G
```

```
Row 1:
──────
query:       SELECT
                 page_name,
                 toStartOfMonth(time) AS month,
                 count() AS page_view_count,
                 round(avg(time_on_page), 2) AS avg_time_on_page
             FROM page_views
             GROUP BY page_name, month
             ORDER BY page_name, month;
read_rows:   107.00
projections: ['default.page_views.page_views_monthly_projection']
```

A projection aggregating values of the source table is used not only for queries that completely match the projection query, but also for queries that select just some of aggregates with grouping by some `GROUP BY` columns of the projection query. For example:

```
EXPLAIN SELECT page_name, count() AS page_view_count FROM page_views GROUP BY page_name;
```

```
   ┌─explain─────────────────────────────────────────────┐
1. │ Expression ((Project names + Projection))           │
2. │   Aggregating                                       │
3. │     Expression                                      │
4. │       ReadFromMergeTree (page_views_monthly_projection) │
   └─────────────────────────────────────────────────────┘
```

View information about projections

Get information on active data parts of the source table from the `system.parts` table:

```
SELECT name, active, path FROM system.parts WHERE table='page_views';
```

```
   ┌─name────────┬─active─┬─path──────────────────────────────────────────────────────────────────┐
1. │ all_1_1_0_3 │      1 │ /var/lib/clickhouse/store/7f1/7f1aea44-35a3-4b96-a5c9-07640aec1340/all_1_1_0_3/ │
   └─────────────┴────────┴───────────────────────────────────────────────────────────────────────┘
```

The directory of the `page_views` table's data part contains subdirectories corresponding to projections (*page_name_projection.proj* and *page_views_monthly_projection.proj*):

```
$ sudo ls /var/lib/clickhouse/store/7f1/7f1aea44-35a3-4b96-a5c9-07640aec1340/all_1_1_0_3/
```

```
checksums.txt  default_compression_codec.txt  page_name.cmrk2                     primary.cidx        time.cmrk2
columns.txt    metadata_version.txt           page_name_projection.proj           serialization.json  time_on_page.bin
count.txt      page_name.bin                  page_views_monthly_projection.proj  time.bin            time_on_page.cmrk2
```

Each projection subdirectory contains column data files (*.bin*), column mark files (*.cmrk2*), and a primary index file (*primary.cidx*) of the projection hidden table:

```
$ sudo ls /var/lib/clickhouse/store/7f1/7f1aea44-35a3-4b96-a5c9-07640aec1340/all_1_1_0_3/page_name_projection.proj
```

```
checksums.txt  default_compression_codec.txt  page_name.cmrk2     time.bin          time_on_page.cmrk2
columns.txt    metadata_version.txt           primary.cidx        time.cmrk2
count.txt      page_name.bin                  serialization.json  time_on_page.bin
```

In the `system.projection_parts` table, see how much disk space the existing projections take up:

```
SELECT
    name,
    parent_name,
    formatReadableSize(bytes_on_disk) AS bytes,
    formatReadableSize(parent_bytes_on_disk) AS parent_bytes,
    bytes_on_disk / parent_bytes_on_disk AS ratio
FROM system.projection_parts;
```

```
   ┌─name──────────────────────────┬─parent_name─┬─bytes────┬─parent_bytes─┬────────────────ratio─┐
1. │ page_name_projection          │ all_1_1_0_3 │ 5.54 MiB │ 11.24 MiB    │    0.49312069529045727 │
2. │ page_views_monthly_projection │ all_1_1_0_3 │ 1.40 KiB │ 11.24 MiB    │ 0.00012159165770921657 │
   └───────────────────────────────┴─────────────┴──────────┴──────────────┴──────────────────────┘
```

From the `system.projection_parts_columns` table, get types of projection columns:

```
SELECT name as projection, column, type as column_type, column_position FROM system.projection_parts_columns;
```

```
   ┌─projection────────────────────┬─column───────────────┬─column_type─────────────────┬─column_position─┐
1. │ page_name_projection          │ time                 │ DateTime                    │               1 │
2. │ page_name_projection          │ page_name            │ String                      │               2 │
3. │ page_name_projection          │ time_on_page         │ UInt64                      │               3 │
4. │ page_views_monthly_projection │ page_name            │ String                      │               1 │
5. │ page_views_monthly_projection │ toStartOfMonth(time) │ Date                        │               2 │
6. │ page_views_monthly_projection │ count()              │ AggregateFunction(count)    │               3 │
7. │ page_views_monthly_projection │ avg(time_on_page)    │ AggregateFunction(avg, UInt64) │            4 │
   └───────────────────────────────┴──────────────────────┴─────────────────────────────┴─────────────────┘
```

User-defined functions and Executable tables

Elena Dvoryadkina

Contents

To Table of Contents

- SQL user-defined functions
  - Example
- Executable user-defined functions
  - Example
- The executable table function
  - Example
- The Executable and ExecutablePool table engines
  - Example
- Script execution parameters

This article describes ways to extend the built-in functionality of ADQM/ClickHouse to perform custom tasks:

- SQL user-defined functions — functions created from lambda expressions that can use SQL constructs and functions;
- executable user-defined functions | the executable table function | the Executable and ExecutablePool table engines — tools that allow you to run external executable programs or scripts (for example, Bash, Python, C++) to process data.

Create a table for test examples

Create the `sales` table with test data to be used in the examples this article provides:

```sql
CREATE TABLE sales (sale_id UInt32, product_name String, product_price UInt32, quantity UInt8)
ENGINE = MergeTree() ORDER BY (sale_id);
```

```sql
INSERT INTO sales VALUES
    (1, 'product_1', 100, 15),
    (2, 'product_2', 200, 20),
    (3, 'product_3', 300, 5),
    (4, 'product_2', 200, 25),
    (5, 'product_1', 100, 50);
```

```
   ┌─sale_id─┬─product_name─┬─product_price─┬─quantity─┐
1. │       1 │ product_1    │           100 │       15 │
2. │       2 │ product_2    │           200 │       20 │
3. │       3 │ product_3    │           300 │        5 │
4. │       4 │ product_2    │           200 │       25 │
5. │       5 │ product_1    │           100 │       50 │
   └─────────┴──────────────┴───────────────┴──────────┘
```

SQL user-defined functions

To create a function from a lambda expression, use the `CREATE FUNCTION` query:

```sql
CREATE FUNCTION <function_name> [ON CLUSTER <cluster_name>] AS (<parameter_0>, <parameter_1> ...) -> <expression>;
```

where:

- `<function_name>` — function name that should be unique among all user-defined and system functions;
- `<parameter_0>, <parameter_1> …` — list of parameters where all variables used by the function should be specified;
- `<expression>` — expression consisting of function parameters, constants, operators, and calls to other functions. Recursive functions are not allowed.

To delete an SQL user-defined function, use the `DROP FUNCTION <function_name>` query.

Example

Create a function that multiplies arguments:

```sql
CREATE FUNCTION sql_udf AS (a, b) -> a * b;
```

Check that the function is added to the list of ADQM functions:

```sql
SELECT name, create_query FROM system.functions WHERE origin = 'SQLUserDefined';
```

```
   ┌─name────┬─create_query─────────────────────────────┐
1. │ sql_udf │ CREATE FUNCTION sql_udf AS (a, b) -> (a * b) │
   └─────────┴──────────────────────────────────────────┘
```

Use the created function to calculate the total cost for each sale in the `sales` table:

```sql
SELECT sale_id, product_price, quantity, sql_udf(product_price, quantity) AS total_price FROM sales;
```

```
   ┌─sale_id─┬─product_price─┬─quantity─┬─total_price─┐
1. │       1 │           100 │       15 │        1500 │
2. │       2 │           200 │       20 │        4000 │
3. │       3 │           300 │        5 │        1500 │
4. │       4 │           200 │       25 │        5000 │
5. │       5 │           100 │       50 │        5000 │
   └─────────┴───────────────┴──────────┴─────────────┘
```

Executable user-defined functions

An executable user-defined function (UDF) calls an external program or script to process data. To create such a function, do the following:

1. In an XML file, describe the function configuration:

```xml
<functions>
    <function>
        <name>...</name>
        <type>executable</type>
        <return_type>...</return_type>
        <return_name>...</return_name>
        <argument>
            <type>...</type>
            <name>...</name>
        </argument>
        <argument>
            <type>...</type>
            <name>...</name>
        </argument>
        <format>...</format>
        <command>...</command>
    </function>
</functions>
```

Configuration parameters of executable user-defined function

name	Function name
type	Command execution type. Possible values: ▪ `executable` — a single command is started; ▪ `executable_pool` — a pool of processes is created.
return_type	Type of a returned value
return_name	Name of a returned value. It should be specified if a return value name is part of serialization for the user-defined function format (for example, Native or JSONEachRow). The default value is `result`
argument	Argument description that includes the type (`type`) and optionally name (`name`) of the argument. Argument names are required if they are part of serialization for the user-defined function format (for example, Native or JSONEachRow). The default argument name is `c` with an argument number. Each argument should be described in a separate `argument` tag
format	Format in which arguments are passed to a command
command	Name of a script to execute or (if `execute_direct` is set to `0`) command. The command should: ▪ read arguments from stdin and output the result to stdout; ▪ process arguments iteratively (i.e. after processing a group of arguments, it should wait for the next group).
lifetime	Function reload interval in seconds. If it is set to `0` (by default), the function is not reloaded
execute_direct	If `1` (by default), a command set in `command` will be searched for in the directory with user script files defined by the `user_scripts_path` parameter in the server configuration. Additional script arguments can be specified with whitespace separators (for example: `<script_name> <arg1> <arg2>`). If `execute_direct = 0` , the `command` value is passed as an argument for `bin/sh -c`

In the configuration of an executable user-defined function, you can also specify additional script execution parameters described below.

2. In the `user_defined_executable_functions_config` parameter of the server configuration (*config.xml*), specify the path to the file with the executable UDF configuration (the path can be specified as absolute or relative to the server configuration file). Configurations of executable user-defined functions can be located in one or more XML files — in the latter case, you can use the `*` and `?` wildcards to specify the path to multiple files, for example:

```xml
<user_defined_executable_functions_config>*_function.xml</user_defined_executable_functions_config>
```

3. In the `user_scripts_path` parameter of the server configuration, specify the directory to store scripts for the executable user-defined functions (the default path is */var/lib/clickhouse/user_scripts*), and place the script specified in the executable UDF configuration as `command` in this directory. Make sure that:

- An ADQM server has the software required to run the executable script installed.
- Permission to execute the script has been granted.

Example

This example shows how to create a user-defined function that executes a Python script. Before you begin, make sure that Python and all necessary libraries are installed on your ADQM server.

Describe the executable user-defined function configuration in the */etc/clickhouse-server/UDFs.xml* file:

```xml
<functions>
    <function>
        <type>executable</type>
        <name>total_price_function</name>
        <return_type>UInt32</return_type>
        <return_name>total_price</return_name>
        <argument>
            <type>UInt32</type>
            <name>product_price</name>
        </argument>
        <argument>
            <type>UInt8</type>
            <name>quantity</name>
        </argument>
        <format>JSONEachRow</format>
        <command>total_price.py</command>
    </function>
</functions>
```

In the *config.xml* file, specify the path to the file with this function configuration in the `user_defined_executable_functions_config` parameter:

```xml
<user_defined_executable_functions_config>UDFs.xml</user_defined_executable_functions_config>
```

In the */var/lib/clickhouse/user_scripts/total_price.py* file, write the Python script:

```python
#!/usr/bin/python3

import sys
import json

if __name__ == '__main__':
    for line in sys.stdin:
        data = json.loads(line)
        price = data['product_price']
        quantity = data['quantity']
        result = {'total_price': price * quantity}
        print(json.dumps(result), end='\n')
        sys.stdout.flush()
```

Give a permission to execute the script:

```
$ sudo chmod +x /var/lib/clickhouse/user_scripts/total_price.py
```

Make sure the function is added to ADQM:

```sql
SELECT name FROM system.functions WHERE origin = 'ExecutableUserDefined';
```

```
   ┌─name─────────────────┐
1. │ total_price_function │
   └──────────────────────┘
```

Run a query to select data from the `sales` table with the total price of each sale calculated via the `total_price_function` function:

```sql
SELECT sale_id, product_price, quantity, total_price_function(product_price, quantity) AS total_price FROM sales;
```

```
   ┌─sale_id─┬─product_price─┬─quantity─┬─total_price─┐
1. │       1 │           100 │       15 │        1500 │
2. │       2 │           200 │       20 │        4000 │
3. │       3 │           300 │        5 │        1500 │
4. │       4 │           200 │       25 │        5000 │
5. │       5 │           100 │       50 │        5000 │
   └─────────┴───────────────┴──────────┴─────────────┘
```

The executable table function

The `executable` table function creates a table based on the output of a function defined in a user script that prints rows to stdout.

The general syntax of the `executable` table function is:

```
executable(<script_name>, <format>, <table_structure>, [<input_query> ...] [,SETTINGS <parameter_name>=<value>, ...])
```

where:

- `<script_name>` — name of an executable script file stored in the directory that is set in the `user_scripts_path` parameter of the server configuration (the default directory is */var/lib/clickhouse/user_scripts/*). A script can read data from any source.
- `<format>` — format in which the function will receive data.
- `<table_structure>` — structure of a table to be generated (define it as `<column_name1> <data_type1>, <column_name2> <data_type2>, …`).
- `<input_query>` — query (or set of queries) that streams its results to stdin for the script to read.
- `SETTINGS` — additional settings to manage the script execution.

Before you call the `executable` table function, make sure that your ADQM server has all packages necessary to run the executable script installed, and that permission to execute the script has been granted.

NOTE

- If you are going to call the same script with the same input queries multiple times, consider using the Executable or ExecutablePool table engine.
- The key advantage between regular executable UDFs and the `executable` table function or Executable table engine is that regular executable UDFs cannot change the number of rows they return (e.g. if the input contains 100 rows, the result will return 100 rows). However, when used with the `executable` table function or Executable table engine, the script can perform any data transformations, including complex aggregations.

Example

Fill the */var/lib/clickhouse/user_scripts/add_tax.py* file with the text of the Python script that will add tax to a product price:

```python
#!/usr/bin/python3

import sys

if __name__ == '__main__':
     for line in sys.stdin:

          split_line = line.split()

          product = split_line[0]
          price = int(split_line[1])
          price_with_tax = price*1.2

          print(product + '\t' + str(price) + '\t' + str(price_with_tax) + '\n', end='')
          sys.stdout.flush()
```

Give a permission to execute the script:

```
$ sudo chmod +x /var/lib/clickhouse/user_scripts/add_tax.py
```

Use the `executable` table function to get a table with prices of unique products from the `sales` table and calculate these prices with tax using the *add_tax.py* script:

```sql
SELECT * FROM executable(
    'add_tax.py',
    TabSeparated,
    'product  String, price UInt32, price_with_tax Float32',
    (SELECT DISTINCT ON (product_name) product_name, product_price FROM sales));
```

Result:

```
   ┌─product───┬─price─┬─price_with_tax─┐
1. │ product_1 │   100 │            120 │
2. │ product_2 │   200 │            240 │
3. │ product_3 │   300 │            360 │
   └───────────┴───────┴────────────────┘
```

The Executable and ExecutablePool table engines

The Executable or ExecutablePool table engine allows you to create a table whose rows are generated from your executable script:

- Executable — runs a script on each data selection;
- ExecutablePool — maintains a pool of persistent processes and takes processes from the pool for reads.

The basic syntax of a query to create a table based on the Executable or ExecutablePool engine is:

```sql
CREATE TABLE <table_name> (<column_name1> <data_type1>, <column_name2> <data_type2>, ...)
ENGINE = Executable|ExecutablePool(<script_name>, <format>, [<input_query> ...])
[SETTINGS <parameter_name>=<value>, ...];
```

where:

- `<script_name>` — executable script whose output will be used to generate data for the `<table_name>` table each time a `SELECT` query is sent to it. The script can read data from any source. The script file should be saved in the directory defined by the `user_scripts_path` server configuration parameter (the default path is_/var/lib/clickhouse/user_scripts/_).
- `<format>` — data format in which the table will accept data.
- `<input_query>` — query (or set of queries) that streams its results to stdin for the script to read.
- `SETTINGS` — additional script execution parameters.

Example

Create an Executable table that will be populated with data generated by the *add_tax.py* script from the example above:

```sql
CREATE TABLE prices_with_tax (product  String, price UInt32, price_with_tax Float32)
ENGINE = Executable('add_tax.py', TabSeparated, (SELECT DISTINCT ON (product_name) product_name, product_price FROM sales));
```

The `CREATE TABLE` query creates a table but does not invoke the script. The script is called by a query to select data from the table:

```sql
SELECT * FROM prices_with_tax;
```

```
   ┌─product───┬─price─┬─price_with_tax─┐
1. │ product_1 │   100 │            120 │
2. │ product_2 │   200 │            240 │
3. │ product_3 │   300 │            360 │
   └───────────┴───────┴────────────────┘
```

Script execution parameters

When creating an executable user-defined function, calling the `executable` table function, and creating Executable/ExecutablePool tables (the basic syntax of corresponding queries is provided in the sections above), you can specify the following parameters that control a script execution.

send_chunk_header	Specifies whether to send the number of rows in each chunk of data before sending a chunk to process. The default value is `false`
command_termination_timeout	Maximum time (in seconds) during which a command should shut down after its pipe is closed. After that time, the `SIGTERM` signal is sent to the process executing the command. The default value is `10`
command_read_timeout	Timeout to read data from command stdout (in milliseconds). The default value is `10000`
command_write_timeout	Timeout to write data to command stdin (in milliseconds). The default value is `10000`
pool_size	Process pool size. If the value is `0` , there are no pool size restrictions. The default value is `16` . The parameter is relevant for: ▪ executable user-defined function with `type` set to `executable_pool` ; ▪ `executable` table function; ▪ ExecutablePool table engine.
max_command_execution_time	Maximum script command execution time for processing a block of data (in seconds). The default value is `10` The parameter is relevant for: ▪ executable user-defined function with `type` set to `executable_pool` ; ▪ `executable` table function; ▪ ExecutablePool table engine.

Example of working with tables via clickhouse-client

Elena Dvoryadkina

Contents

To Table of Contents

- Overview
- Step 1. Create a database
- Step 2. Create a table
- Step 3. Get information about a table
- Step 4. Insert new data into a table
- Step 5. Read data from a table
  - Select all data
  - Filter data
  - Sort the results
  - Group the results
  - Combine SELECT with CREATE and INSERT INTO
- Step 6. Update data in a table
- Step 7. Delete data from a table
- Step 8. Alter a table
- Step 9. Detach and attach a table
- Step 10. Delete a table

Overview

The easiest way to work with ADQM/ClickHouse tables is to use the clickhouse-client console client, which becomes available on each cluster host after ADQM installation. This client allows you to enter queries, pass them to ClickHouse, and view the results. Client parameters can be passed via the command line or configuration files.

To start working with `clickhouse-client`, run the command:

```
$ clickhouse-client -m
```

This command establishes a connection to the local ClickHouse server (`localhost:9000`) as the `default` user without a password and connects to the default database. The terminal switches to the interactive mode of operation with the ClickHouse server and displays the `:)` prompt, after which you can enter queries. The `-m` (or `--multiline`) parameter enables the multiline query input mode — to execute a query, you should end it with a semicolon and press `Enter`.

```
$ clickhouse-client -m
ClickHouse client version 24.3.11.7.
Connecting to localhost:9000 as user default.
Connected to ClickHouse server version 24.3.11.

:)
```

To exit `clickhouse-client`, press `Ctrl+D` or type `exit`, `quit`, `q`, or `logout` instead of a query.

The main operations for working with data in ADQM/ClickHouse are listed below (all examples are based on a simple database that stores sales information). For a full list of available commands, refer to the ClickHouse SQL Statements section of the ClickHouse documentation. Before you start executing the commands described in this article, it is also recommended that you read the ***Concepts*** → ***Data Model*** section of the current ADQM documentation.

Step 1. Create a database

A database to which `clickhouse-client` connects by default is specified by the ***Default database*** parameter on the ADQMDB service configuration page in the ADCM interface (***ADQM cluster*** → ***Services*** → ***ADQMDB*** → ***Primary Configuration*** → ***Other*** → ***Default database***). The default value of this parameter is the `default` database, which is created automatically when ADQM is installed.

To create a new database, use the CREATE DATABASE query.

The following query creates the `test_db` database (based on the Atomic engine) on all hosts in the `default_cluster` cluster:

```
CREATE DATABASE test_db ON CLUSTER default_cluster;
```

You can change the current database for the session with the `USE` command, for example:

```
USE test_db;
```

NOTE

- To select a database other than the default one when launching `clickhouse-client`, use the `-d` or `--database` parameter (for example, `clickhouse-client -d test_db`).
- If a query requires access to a table that is not in the current database of the session, specify the desired database explicitly using a dot `.` before a table name.

Step 2. Create a table

To create a new table, use the `CREATE TABLE` query — specify the table name and table engine, describe all columns (by defining their names, data types, modifiers, data compression codecs), set indexes, constraints, and other settings. You can find an overview of the `CREATE TABLE` query syntax with descriptions of its parameters and clauses in the section Create a MergeTree table of the article about basic operations with ADQM tables.

The following query creates the `sales` table with columns of different types to store sales data:

```
CREATE TABLE sales (
    id UInt32,
    product_name String,
    category String,
    sale_date Date DEFAULT toDate(now()),
    units_sold UInt32,
    price_per_unit Float32,
    region Enum('North America' = 1, 'Europe' = 2, 'Asia' = 3)
)
ENGINE = MergeTree()
ORDER BY id;
```

Step 3. Get information about a table

To check whether a table exists and get information about it, you can use the following commands:

- `EXISTS <table_name>` — checks if a table exists in a database (returns one column with a single value — `0` if the table does not exist or `1` if the table exists).

```
EXISTS sales;
```

The `sales` table exists in the current `test_db` database:

```
   ┌─result─┐
1. │      1 │
   └────────┘
```

- `DESCRIBE TABLE <table_name>` — returns information about each column of a table (name, data type, default value expression, comment, data compression codec, and TTL expression).

```
DESCRIBE TABLE sales;
```

```
   ┌─name───────────┬─type─────────────────────────────────────────────────┬─default_type─┬─default_expression─┬─comment─┬─codec_exp
ression─┬─ttl_expression─┐
1. │ id             │ UInt32                                               │              │                    │         │
│                │
2. │ product_name   │ String                                               │              │                    │         │
│                │
3. │ category       │ String                                               │              │                    │         │
│                │
4. │ sale_date      │ Date                                                 │ DEFAULT      │ toDate(now())      │         │
│                │
5. │ units_sold     │ UInt32                                               │              │                    │         │
│                │
6. │ price_per_unit │ Float32                                              │              │                    │         │
│                │
7. │ region         │ Enum8('North America' = 1, 'Europe' = 2, 'Asia' = 3) │              │                    │         │
│                │
   └────────────────┴──────────────────────────────────────────────────────┴──────────────┴────────────────────┴─────────┴──────────
────────┴────────────────┘
```

If the command output is too wide to fit in the console window, you can use the Vertical format for the output data — in this case, each value is printed on a separate line. To enable this, specify `\G` instead of or after the semicolon or use `FORMAT Vertical` at the end of a query:

```
DESCRIBE TABLE sales;\G
```

```
DESCRIBE TABLE sales FORMAT Vertical;
```

Command result in Vertical format

```
Row 1:
──────
name:               id
type:               UInt32
default_type:
default_expression:
comment:
codec_expression:
ttl_expression:

Row 2:
──────
name:               product_name
type:               String
default_type:
default_expression:
comment:
codec_expression:
ttl_expression:

Row 3:
──────
name:               category
type:               String
default_type:
default_expression:
comment:
codec_expression:
ttl_expression:

Row 4:
──────
name:               sale_date
type:               Date
default_type:       DEFAULT
default_expression: toDate(now())
comment:
codec_expression:
ttl_expression:

Row 5:
──────
name:               units_sold
type:               UInt32
default_type:
default_expression:
comment:
codec_expression:
ttl_expression:

Row 6:
──────
name:               price_per_unit
type:               Float32
default_type:
default_expression:
comment:
codec_expression:
ttl_expression:

Row 7:
──────
name:               region
type:               Enum8('North America' = 1, 'Europe' = 2, 'Asia' = 3)
default_type:
default_expression:
comment:
codec_expression:
ttl_expression:
```

To get a full list of formats ADQM/ClickHouse supports for input and output data, see the Formats for Input and Output Data article of ClickHouse documentation.

- `SELECT * FROM system.tables WHERE name = '<table_name>'` — returns metadata for a table with the specified name from the `system.tables` system table.

```
SELECT * FROM system.tables WHERE name = 'sales';\G
```

Command result

```
Row 1:
──────
database:                      test_db
name:                          sales
uuid:                          4a8690b9-e048-4b55-acea-27586cd2380f
engine:                        MergeTree
is_temporary:                  0
data_paths:                    ['/var/lib/clickhouse/store/4a8/4a8690b9-e048-4b55-acea-27586cd2380f/']
metadata_path:                 /var/lib/clickhouse/store/a8e/a8e6a2c0-ea2c-4c0b-8669-de931f53178d/sales.sql
metadata_modification_time:    2025-01-16 15:17:05
metadata_version:              0
dependencies_database:         []
dependencies_table:            []
create_table_query:            CREATE TABLE test_db.sales (`id` UInt32, `product_name` String, `category` String,
`sale_date` Date DEFAULT toDate(now()), `units_sold` UInt32, `price_per_unit` Float32, `region` Enum8('North America' = 1,
'Europe' = 2, 'Asia' = 3)) ENGINE = MergeTree ORDER BY id SETTINGS index_granularity = 8192
engine_full:                   MergeTree ORDER BY id SETTINGS index_granularity = 8192
as_select:
partition_key:
sorting_key:                   id
primary_key:                   id
sampling_key:
storage_policy:                default
total_rows:                    0
total_bytes:                   0
total_bytes_uncompressed:      0
parts:                         0
active_parts:                  0
total_marks:                   0
lifetime_rows:                 NULL
lifetime_bytes:                NULL
comment:
has_own_data:                  1
loading_dependencies_database: []
loading_dependencies_table:    []
loading_dependent_database:    []
loading_dependent_table:       []
```

- `SHOW CREATE TABLE <table_name>` — returns a single column containing the text of the CREATE TABLE query used to create the specified table.

```
SHOW CREATE TABLE sales;\G
```

```
Row 1:
──────
statement: CREATE TABLE test_db.sales
(
    `id` UInt32,
    `product_name` String,
    `category` String,
    `sale_date` Date DEFAULT toDate(now()),
    `units_sold` UInt32,
    `price_per_unit` Float32,
    `region` Enum8('North America' = 1, 'Europe' = 2, 'Asia' = 3)
)
ENGINE = MergeTree
ORDER BY id
SETTINGS index_granularity = 8192
```

- `SHOW TABLES` — lists all tables that exist in a database.

```
SHOW TABLES;
```

The current `test_db` database contains only one `sales` table:

```
   ┌─name──┐
1. │ sales │
   └───────┘
```

Step 4. Insert new data into a table

To add new data to a table, use the INSERT INTO query:

```
INSERT INTO <table_name> [(<c1>, <c2>, <c3>)] VALUES (v11, v12, v13), (v21, v22, v23), ...;
```

After a table name, you can optionally specify columns to insert data into — list the column names separated by commas or use an expression with `*` and modifiers such as `APPLY`, `EXCEPT`, `REPLACE`. After the `VALUES` keyword, list the column values separated by commas in the same order as the column names in the `(<c1>, <c2>, <c3>)` list. If columns to accept data are not defined explicitly, you should specify values of the corresponding types for all columns of the table — the given values will appear in the columns according to the order in which they were declared during the table creation.

If the specified list of columns does not include all existing columns of the table, the rest columns are filled with:

- values calculated from the `DEFAULT` expressions specified for columns in the table definition;
- zeros and empty strings, if the `DEFAULT` expressions for columns are not defined.

The following five separate queries each add one row to the `sales` table:

```
INSERT INTO sales VALUES (1, 'Laptop', 'Electronics', '2025-01-03', 5, 999.99, 1);
```

```
INSERT INTO sales VALUES (2, 'Smartphone', 'Electronics', '2025-01-04', 10, 699.99, 2);
```

```
INSERT INTO sales VALUES (3, 'Headphones', 'Electronics', '2025-01-04', 25, 49.99, 3);
```

```
INSERT INTO sales VALUES (4, 'Desk Chair', 'Furniture', '2025-01-05', 3, 149.99, 1);
```

```
INSERT INTO sales VALUES (5, 'Coffee Table', 'Furniture', '2025-01-05', 2, 249.99, 2);
```

To insert multiple rows via one `INSERT INTO` query, list all data tuples separated by commas after the `VALUES` keyword:

```
INSERT INTO sales VALUES
(6, 'Notebook', 'Stationery', '2025-01-06', 100, 1.99, 3),
(7, 'Pen', 'Stationery', '2025-01-06', 200, 0.49, 1),
(8, 'Water Bottle', 'Accessories', '2025-01-07', 15, 9.99, 2),
(9, 'Backpack', 'Accessories', '2025-01-07', 8, 59.99, 3),
(10, 'Monitor', 'Electronics', '2025-01-08', 6, 199.99, 1),
(11, 'Keyboard', 'Electronics', '2025-01-08', 12, 49.99, 2);
```

The following query adds data to all columns except `sale_date`, so as many values are passed as there are defined columns, and the `sale_date` column will be filled with the default value (the current date):

```
INSERT INTO sales (* EXCEPT(sale_date)) VALUES (12, 'Office Chair', 'Furniture',  4, 199.99, 3);
```

You can also use the `DEFAULT` keyword to insert default values:

```
INSERT INTO sales VALUES (13, 'Printer', 'Electronics', DEFAULT, 2, 299.99, 1);
```

You can also pass data in any format supported by ClickHouse to the `INSERT INTO` query (see the ***Input*** column in the Formats for Input and Output Data table). To do this, specify a format in a query explicitly using the `FORMAT` keyword, for example:

```
INSERT INTO sales FORMAT JSONEachRow
{"id":14,"product_name":"Cap","category":"Accessories","sale_date":"2025-01-09","units_sold":30,"price_per_unit":5.99,"region":2}
{"id":15,"product_name":"Binder","category":"Stationery","sale_date":"2025-01-09","units_sold":50,"price_per_unit":2.
99,"region":3};
```

NOTE

Data can also be imported into an ADQM/ClickHouse table from files of various formats — see Import from files to get examples.

Step 5. Read data from a table

To read data from a table, use the SELECT query. Here are some specifics of how to construct a query:

- After the `SELECT` statement, specify a list of columns to retrieve data from (list names of individual columns or use the `*` symbol to get data from all columns) and/or expressions to be evaluated over a set of rows (for example, aggregate or window functions). When creating a list of columns/expressions, you can use the `APPLY`, `EXCEPT`, `REPLACE` modifiers and their combinations.
- After the `FROM` keyword, specify the name of a table from which to retrieve data. To read data from two tables and combine them into a single result set, use the JOIN operator. A subquery or table function can also be used as a source for selecting data.
- In the `WHERE` clause, describe conditions for filtering rows to read.
- For additional processing of the output results (sorting, grouping), you can use `ORDER BY`, `GROUP BY`, `HAVING`, and other clauses.

Examples of various `SELECT` queries are listed below.

Select all data

The following query selects all data from the `sales` table:

```
SELECT * FROM sales;
```

Since this example uses the `*` symbol instead of column names and does not contain the `WHERE` condition, all columns and rows of the table are included in the results:

```
    ┌─id─┬─product_name─┬─category────┬──sale_date─┬─units_sold─┬─price_per_unit─┬─region────────┐
 1. │  1 │ Laptop       │ Electronics │ 2025-01-03 │          5 │         999.99 │ North America │
 2. │  2 │ Smartphone   │ Electronics │ 2025-01-04 │         10 │         699.99 │ Europe        │
 3. │  3 │ Headphones   │ Electronics │ 2025-01-04 │         25 │          49.99 │ Asia          │
 4. │  4 │ Desk Chair   │ Furniture   │ 2025-01-05 │          3 │         149.99 │ North America │
 5. │  5 │ Coffee Table │ Furniture   │ 2025-01-05 │          2 │         249.99 │ Europe        │
 6. │  6 │ Notebook     │ Stationery  │ 2025-01-06 │        100 │           1.99 │ Asia          │
 7. │  7 │ Pen          │ Stationery  │ 2025-01-06 │        200 │           0.49 │ North America │
 8. │  8 │ Water Bottle │ Accessories │ 2025-01-07 │         15 │           9.99 │ Europe        │
 9. │  9 │ Backpack     │ Accessories │ 2025-01-07 │          8 │          59.99 │ Asia          │
10. │ 10 │ Monitor      │ Electronics │ 2025-01-08 │          6 │         199.99 │ North America │
11. │ 11 │ Keyboard     │ Electronics │ 2025-01-08 │         12 │          49.99 │ Europe        │
12. │ 12 │ Office Chair │ Furniture   │ 2025-01-16 │          4 │         199.99 │ Asia          │
13. │ 13 │ Printer      │ Electronics │ 2025-01-16 │          2 │         299.99 │ North America │
14. │ 14 │ Cap          │ Accessories │ 2025-01-09 │         30 │           5.99 │ Europe        │
15. │ 15 │ Binder       │ Stationery  │ 2025-01-09 │         50 │           2.99 │ Asia          │
    └────┴──────────────┴─────────────┴────────────┴────────────┴────────────────┴───────────────┘
```

Filter data

The next query returns sales data only for January 7, 2025:

```
SELECT
    product_name,
    sale_date,
    units_sold
FROM sales
WHERE sale_date = '2025-01-07';
```

```
   ┌─product_name─┬──sale_date─┬─units_sold─┐
1. │ Water Bottle │ 2025-01-07 │         15 │
2. │ Backpack     │ 2025-01-07 │          8 │
   └──────────────┴────────────┴────────────┘
```

Sort the results

The following query returns the top 10 sales from the `sales` table in descending order:

```
SELECT
    sale_date,
    product_name,
    units_sold,
    price_per_unit,
    round(units_sold * price_per_unit, 2) AS revenue
FROM sales
ORDER BY revenue DESC
LIMIT 10;
```

```
    ┌──sale_date─┬─product_name─┬─units_sold─┬─price_per_unit─┬─revenue─┐
 1. │ 2025-01-04 │ Smartphone   │         10 │         699.99 │  6999.9 │
 2. │ 2025-01-03 │ Laptop       │          5 │         999.99 │ 4999.95 │
 3. │ 2025-01-04 │ Headphones   │         25 │          49.99 │ 1249.75 │
 4. │ 2025-01-08 │ Monitor      │          6 │         199.99 │ 1199.94 │
 5. │ 2025-01-16 │ Office Chair │          4 │         199.99 │  799.96 │
 6. │ 2025-01-16 │ Printer      │          2 │         299.99 │  599.98 │
 7. │ 2025-01-08 │ Keyboard     │         12 │          49.99 │  599.88 │
 8. │ 2025-01-05 │ Coffee Table │          2 │         249.99 │  499.98 │
 9. │ 2025-01-07 │ Backpack     │          8 │          59.99 │  479.92 │
10. │ 2025-01-05 │ Desk Chair   │          3 │         149.99 │  449.97 │
    └────────────┴──────────────┴────────────┴────────────────┴─────────┘
```

Group the results

The next query groups the sales stored in the `sales` table by product category and finds a product with the highest unit price in each category:

```
SELECT
    category,
    argMax(product_name, price_per_unit) AS most_expensive_product,
    max(price_per_unit) AS highest_price
FROM sales
GROUP BY category
ORDER BY highest_price DESC;
```

```
   ┌─category────┬─most_expensive_product─┬─highest_price─┐
1. │ Electronics │ Laptop                 │        999.99 │
2. │ Furniture   │ Coffee Table           │        249.99 │
3. │ Accessories │ Backpack               │         59.99 │
4. │ Stationery  │ Binder                 │          2.99 │
   └─────────────┴────────────────────────┴───────────────┘
```

The following query returns the daily revenue for each region:

```
SELECT
    sale_date,
    region,
    round(SUM(units_sold * price_per_unit), 2) AS daily_revenue
FROM sales
GROUP BY sale_date, region
ORDER BY sale_date, region;
```

```
    ┌──sale_date─┬─region────────┬─daily_revenue─┐
 1. │ 2025-01-03 │ North America │       4999.95 │
 2. │ 2025-01-04 │ Europe        │        6999.9 │
 3. │ 2025-01-04 │ Asia          │       1249.75 │
 4. │ 2025-01-05 │ North America │        449.97 │
 5. │ 2025-01-05 │ Europe        │        499.98 │
 6. │ 2025-01-06 │ North America │            98 │
 7. │ 2025-01-06 │ Asia          │           199 │
 8. │ 2025-01-07 │ Europe        │        149.85 │
 9. │ 2025-01-07 │ Asia          │        479.92 │
10. │ 2025-01-08 │ North America │       1199.94 │
11. │ 2025-01-08 │ Europe        │        599.88 │
12. │ 2025-01-09 │ Europe        │         179.7 │
13. │ 2025-01-09 │ Asia          │         149.5 │
14. │ 2025-01-14 │ North America │        599.98 │
15. │ 2025-01-14 │ Asia          │        799.96 │
    └────────────┴───────────────┴───────────────┘
```

Combine SELECT with CREATE and INSERT INTO

You can use the `SELECT` statement to copy data from one table to another via the `CREATE` and `INSERT INTO` commands.

The query below creates a new `sales_copy` table with the same structure as the `sales` table and also copies the rows corresponding to sales up to and including January 7, 2025 from the original table into the new one using the `SELECT` command:

```
CREATE TABLE sales_copy
ENGINE = MergeTree
ORDER BY id
AS SELECT * FROM sales
WHERE sale_date <= '2025-01-07';
```

The following query adds the rest rows (sales after January 7, 2025) from the source `sales` table to `sales_copy` using the `INSERT INTO` command:

```
INSERT INTO sales_copy SELECT * FROM sales WHERE sale_date > '2025-01-07';
```

To verify that the contents of these two tables are identical, select data from both tables:

```
SELECT * FROM sales;
```

```
    ┌─id─┬─product_name─┬─category────┬──sale_date─┬─units_sold─┬─price_per_unit─┬─region────────┐
 1. │  1 │ Laptop       │ Electronics │ 2025-01-03 │          5 │         999.99 │ North America │
 2. │  2 │ Smartphone   │ Electronics │ 2025-01-04 │         10 │         699.99 │ Europe        │
 3. │  3 │ Headphones   │ Electronics │ 2025-01-04 │         25 │          49.99 │ Asia          │
 4. │  4 │ Desk Chair   │ Furniture   │ 2025-01-05 │          3 │         149.99 │ North America │
 5. │  5 │ Coffee Table │ Furniture   │ 2025-01-05 │          2 │         249.99 │ Europe        │
 6. │  6 │ Notebook     │ Stationery  │ 2025-01-06 │        100 │           1.99 │ Asia          │
 7. │  7 │ Pen          │ Stationery  │ 2025-01-06 │        200 │           0.49 │ North America │
 8. │  8 │ Water Bottle │ Accessories │ 2025-01-07 │         15 │           9.99 │ Europe        │
 9. │  9 │ Backpack     │ Accessories │ 2025-01-07 │          8 │          59.99 │ Asia          │
10. │ 10 │ Monitor      │ Electronics │ 2025-01-08 │          6 │         199.99 │ North America │
11. │ 11 │ Keyboard     │ Electronics │ 2025-01-08 │         12 │          49.99 │ Europe        │
12. │ 12 │ Office Chair │ Furniture   │ 2025-01-16 │          4 │         199.99 │ Asia          │
13. │ 13 │ Printer      │ Electronics │ 2025-01-16 │          2 │         299.99 │ North America │
14. │ 14 │ Cap          │ Accessories │ 2025-01-09 │         30 │           5.99 │ Europe        │
15. │ 15 │ Binder       │ Stationery  │ 2025-01-09 │         50 │           2.99 │ Asia          │
    └────┴──────────────┴─────────────┴────────────┴────────────┴────────────────┴───────────────┘
```

```
SELECT * FROM sales_copy;
```

```
    ┌─id─┬─product_name─┬─category────┬──sale_date─┬─units_sold─┬─price_per_unit─┬─region────────┐
 1. │  1 │ Laptop       │ Electronics │ 2025-01-03 │          5 │         999.99 │ North America │
 2. │  2 │ Smartphone   │ Electronics │ 2025-01-04 │         10 │         699.99 │ Europe        │
 3. │  3 │ Headphones   │ Electronics │ 2025-01-04 │         25 │          49.99 │ Asia          │
 4. │  4 │ Desk Chair   │ Furniture   │ 2025-01-05 │          3 │         149.99 │ North America │
 5. │  5 │ Coffee Table │ Furniture   │ 2025-01-05 │          2 │         249.99 │ Europe        │
 6. │  6 │ Notebook     │ Stationery  │ 2025-01-06 │        100 │           1.99 │ Asia          │
 7. │  7 │ Pen          │ Stationery  │ 2025-01-06 │        200 │           0.49 │ North America │
 8. │  8 │ Water Bottle │ Accessories │ 2025-01-07 │         15 │           9.99 │ Europe        │
 9. │  9 │ Backpack     │ Accessories │ 2025-01-07 │          8 │          59.99 │ Asia          │
10. │ 10 │ Monitor      │ Electronics │ 2025-01-08 │          6 │         199.99 │ North America │
11. │ 11 │ Keyboard     │ Electronics │ 2025-01-08 │         12 │          49.99 │ Europe        │
12. │ 12 │ Office Chair │ Furniture   │ 2025-01-16 │          4 │         199.99 │ Asia          │
13. │ 13 │ Printer      │ Electronics │ 2025-01-16 │          2 │         299.99 │ North America │
14. │ 14 │ Cap          │ Accessories │ 2025-01-09 │         30 │           5.99 │ Europe        │
15. │ 15 │ Binder       │ Stationery  │ 2025-01-09 │         50 │           2.99 │ Asia          │
    └────┴──────────────┴─────────────┴────────────┴────────────┴────────────────┴───────────────┘
```

Step 6. Update data in a table

To update data in a table, use the ALTER TABLE…UPDATE query, specifying a list of new values for columns (or expressions to calculate values) after the `UPDATE` keyword and a condition for selecting rows in which you want to replace the values after `WHERE`:

```
ALTER TABLE <table_name> UPDATE <column_name> = <new_value>[, ...] WHERE <filter_expr>;
```

For example, the following query increases the price of each product in the `Accessories` category in the `sales_copy` table by 2 percent:

```
ALTER TABLE sales_copy
UPDATE price_per_unit = round(price_per_unit*1.02, 2)
WHERE category = 'Accessories';
```

To check the query results, you can output product data (name and price) from the `Accessories` category using the `SELECT` command:

```
SELECT product_name, price_per_unit FROM sales_copy WHERE category='Accessories';
```

```
   ┌─product_name─┬─price_per_unit─┐
1. │ Water Bottle │          10.19 │
2. │ Backpack     │          61.19 │
3. │ Cap          │           6.11 │
   └──────────────┴────────────────┘
```

Step 7. Delete data from a table

To delete data rows from a table according to the specified condition, you can use the `ALTER TABLE…DELETE` (for any tables) or `DELETE FROM` (for MergeTree tables) query — see Delete data for more details.

The following query deletes rows corresponding to sales up to and including January 5, 2025 from the `sales_copy` table:

```
ALTER TABLE sales_copy DELETE WHERE sale_date <= '2025-01-05';
```

To check the query results, select the table data and make sure that it contains only sales after January 5, 2025:

```
    ┌─id─┬─product_name─┬─category────┬──sale_date─┬─units_sold─┬─price_per_unit─┬─region────────┐
 1. │  6 │ Notebook     │ Stationery  │ 2025-01-06 │        100 │           1.99 │ Asia          │
 2. │  7 │ Pen          │ Stationery  │ 2025-01-06 │        200 │           0.49 │ North America │
 3. │  8 │ Water Bottle │ Accessories │ 2025-01-07 │         15 │          10.19 │ Europe        │
 4. │  9 │ Backpack     │ Accessories │ 2025-01-07 │          8 │          61.19 │ Asia          │
 5. │ 10 │ Monitor      │ Electronics │ 2025-01-08 │          6 │         199.99 │ North America │
 6. │ 11 │ Keyboard     │ Electronics │ 2025-01-08 │         12 │          49.99 │ Europe        │
 7. │ 12 │ Office Chair │ Furniture   │ 2025-01-16 │          4 │         199.99 │ Asia          │
 8. │ 13 │ Printer      │ Electronics │ 2025-01-16 │          2 │         299.99 │ North America │
 9. │ 14 │ Cap          │ Accessories │ 2025-01-09 │         30 │           6.11 │ Europe        │
10. │ 15 │ Binder       │ Stationery  │ 2025-01-09 │         50 │           2.99 │ Asia          │
    └────┴──────────────┴─────────────┴────────────┴────────────┴────────────────┴───────────────┘
```

The following query deletes a row related to a sale of the `Keyboard` product in the `Europe` region from the `sales_copy` table:

```
DELETE FROM sales_copy WHERE product_name = 'Keyboard' AND region = 'Europe';
```

The specified row was successfully deleted:

```
   ┌─id─┬─product_name─┬─category────┬──sale_date─┬─units_sold─┬─price_per_unit─┬─region────────┐
1. │  6 │ Notebook     │ Stationery  │ 2025-01-06 │        100 │           1.99 │ Asia          │
2. │  7 │ Pen          │ Stationery  │ 2025-01-06 │        200 │           0.49 │ North America │
3. │  8 │ Water Bottle │ Accessories │ 2025-01-07 │         15 │          10.19 │ Europe        │
4. │  9 │ Backpack     │ Accessories │ 2025-01-07 │          8 │          61.19 │ Asia          │
5. │ 10 │ Monitor      │ Electronics │ 2025-01-08 │          6 │         199.99 │ North America │
6. │ 12 │ Office Chair │ Furniture   │ 2025-01-16 │          4 │         199.99 │ Asia          │
7. │ 13 │ Printer      │ Electronics │ 2025-01-16 │          2 │         299.99 │ North America │
8. │ 14 │ Cap          │ Accessories │ 2025-01-09 │         30 │           6.11 │ Europe        │
9. │ 15 │ Binder       │ Stationery  │ 2025-01-09 │         50 │           2.99 │ Asia          │
   └────┴──────────────┴─────────────┴────────────┴────────────┴────────────────┴───────────────┘
```

Step 8. Alter a table

To change the structure (columns) of a table, use the `ALTER TABLE` query (supported by the MergeTree, Merge, and Distributed table engines):

```
ALTER TABLE <table_name> ADD|DROP|RENAME|CLEAR|COMMENT|{MODIFY|ALTER}|MATERIALIZE COLUMN <column_name> ...
```

In one query, you can specify multiple comma-separated actions to perform on table columns.

The following query renames the `product_name` column to `product` and adds a new `store_id` column:

```
ALTER TABLE sales_copy RENAME COLUMN product_name TO product, ADD COLUMN store_id UInt8;
```

To check that the column is renamed and the new `store_id` column is added, run the `DESCRIBE TABLE` query:

```
DESCRIBE TABLE sales_copy;
```

```
   ┌─name───────────┬─type─────────────────────────────────────────────────┬
1. │ id             │ UInt32                                               │
2. │ product        │ String                                               │
3. │ category       │ String                                               │
4. │ sale_date      │ Date                                                 │
5. │ units_sold     │ UInt32                                               │
6. │ price_per_unit │ Float32                                              │
7. │ region         │ Enum8('North America' = 1, 'Europe' = 2, 'Asia' = 3) │
8. │ store_id       │ UInt8                                                │
   └────────────────┴──────────────────────────────────────────────────────┴
```

NOTE

The `ALTER TABLE` query also allows you to perform various manipulations with partitions, data parts, constraints, indexes, table projections, add comments to tables, set TTL rules — see the detailed descriptions in the corresponding articles in the ALTER section of the ClickHouse documentation.

Step 9. Detach and attach a table

With the `DETACH TABLE` query, you can detach a table, which means it will be "invisible" to the server (but the table data and metadata will not be deleted).

The following query detaches the `sales_copy` table:

```
DETACH TABLE sales_copy;
```

Make sure the `sales_copy` table is not in the list of database tables:

```
SHOW tables;
```

```
   ┌─name──┐
1. │ sales │
   └───────┘
```

The next time the server starts, it will read the table metadata and "see" it again. To prevent the server from re-enabling the table automatically on restart, detach the table permanently — to do this, use the `PERMANENTLY` keyword at the end of the `DETACH TABLE` query.

To attach a table back (regardless of whether it was permanently detached or not), run the `ATTACH TABLE` query:

```
ATTACH TABLE sales_copy;
```

Check that the ClickHouse server can "see" the `sales_copy` table again:

```
SHOW tables;
```

```
   ┌─name───────┐
1. │ sales      │
2. │ sales_copy │
   └────────────┘
```

Step 10. Delete a table

To remove a table permanently, use the `DROP TABLE` query:

```
DROP TABLE sales_copy;
```

The table has been successfully removed from the database:

```
SHOW tables;
```

```
   ┌─name──┐
1. │ sales │
   └───────┘
```

JOIN clause

Elena Dvoryadkina

Contents

To Table of Contents

- Overview
- JOIN types
  - INNER JOIN
  - OUTER JOIN
  - CROSS JOIN
  - LEFT/RIGHT SEMI JOIN
  - LEFT/RIGHT ANTI JOIN
  - LEFT/RIGHT/INNER ANY JOIN
  - ASOF JOIN
- JOIN conditions
  - ON
  - USING

Overview

The `JOIN` clause combines columns from two tables into a new table. A set of data rows in a resulting table depends on the `JOIN` type and the specified join conditions.

The general `JOIN` syntax is:

```
FROM <left_table> <JOIN_type> <right_table> <join_condition>
```

where:

- `<JOIN_type>` — JOIN type. ADQM supports all standard SQL JOIN types and additional specialized JOIN types that can be useful for various analytical queries and time-series analysis.

  Standard SQL JOIN types:

  - INNER JOIN
  - LEFT/RIGHT/FULL OUTER JOIN
  - CROSS JOIN

  Additional types of JOIN available in ADQM:

  - LEFT/RIGHT SEMI JOIN
  - LEFT/RIGHT ANTI JOIN
  - LEFT/RIGHT/INNER ANY JOIN
  - ASOF JOIN

  If the JOIN type is not specified explicitly, `INNER JOIN` is used.

- `<join_condition>` — condition that determines which rows from two source tables match each other. A condition (or multiple conditions) should be specified as `ON <expr_list>` or `USING <column_list>` for all `JOIN` types except `CROSS JOIN` — see the JOIN conditions section for details.

The join_algorithm setting allows you to choose an algorithm for join execution. You can specify multiple algorithms (one of the available algorithms will be selected for each specific query based on its type/strictness and table engine) or allow the system to select and dynamically change the algorithm during query execution depending on resource availability and usage.

Create tables for test examples

To reproduce the examples provided below in this article and explore different types of `JOIN`, create two tables:

- `users`

```
CREATE TABLE users (user_id Int32, name String, role_id Int32) ENGINE = MergeTree ORDER BY user_id;
```

```
INSERT INTO users VALUES (1, 'john', 10), (1, 'john', 20), (1, 'john', 30), (2, 'mary', 30), (2, 'mary', 40), (3, 'andrew',
40), (4, 'harry', 70), (5, 'ann', 50);
```

```
┌─user_id─┬─name───┬─role_id─┐
│       1 │ john   │      10 │
│       2 │ mike   │      20 │
│       3 │ tom    │      30 │
│       4 │ mary   │      30 │
│       5 │ ada    │      40 │
│       6 │ andrew │      40 │
│       7 │ harry  │      70 │
│       8 │ ann    │      50 │
└─────────┴────────┴─────────┘
```

- `roles`

```
CREATE TABLE roles (id Int32, title String) ENGINE = MergeTree ORDER BY id;
```

```
INSERT INTO roles VALUES (10, 'admin'), (20, 'owner'), (30, 'author'), (40, 'reviewer'), (50, 'editor'), (60, 'view only');
```

```
┌─id─┬─title─────┐
│ 10 │ admin     │
│ 20 │ owner     │
│ 30 │ author    │
│ 40 │ reviewer  │
│ 50 │ editor    │
│ 60 │ view only │
└────┴───────────┘
```

The `users` table's `role_id` field is a key that refers to the `id` field in the `roles` table and serves for defining relationships between tables.

JOIN types

INNER JOIN

For each row in the left table, `INNER JOIN` checks whether the right table contains a row that satisfies the JOIN condition. If a corresponding row is found, a new row with columns from both tables is created and added to the result set. If there are more than one match for a row, then all matches are added to the resulting table (it means that the Cartesian product is produced for rows that match on the join condition).

The `INNER` keyword can be omitted in a query.

Example

```
SELECT users.name AS user, roles.title AS role FROM users INNER JOIN roles ON users.role_id=roles.id;
```

The query joins the `users` and `roles` tables and returns combined rows matching on identifiers of user roles. The `harry` user is not included in the result because the identifier of a role assigned to him was not found in the `roles` table. The result also does not include the `view only` role since it is not assigned to any user.

```
┌─user───┬─role─────┐
│ john   │ admin    │
│ mike   │ owner    │
│ tom    │ author   │
│ mary   │ author   │
│ ada    │ reviewer │
│ andrew │ reviewer │
│ ann    │ editor   │
└────────┴──────────┘
```

OUTER JOIN

`LEFT/RIGHT/FULL OUTER JOIN` queries extend `INNER JOIN` with rows from source tables that do not match the specified join condition. The `OUTER` keyword can be omitted.

- The `LEFT OUTER JOIN` clause performs the `INNER JOIN` operation first. Then, each row of the left table, which is not included in the `INNER JOIN` result, is added to the `LEFT OUTER JOIN` result with default values for the corresponding data types in columns of the right table. The joined table always has at least one row for each row of the left table.

  Example

  ```
  SELECT users.name AS user, roles.title AS role FROM users LEFT OUTER JOIN roles ON users.role_id=roles.id;
  ```

  The result contains role titles for all users from the `users` table, and also includes a row for the `harry` user whose role was not defined in the `roles` table. In this row, the `roles.title` column (with the `role` alias) contains an empty string — the default value for the `String` type:

  ```
  ┌─user───┬─role─────┐
  │ john   │ admin    │
  │ mike   │ owner    │
  │ tom    │ author   │
  │ mary   │ author   │
  │ ada    │ reviewer │
  │ andrew │ reviewer │
  │ harry  │          │
  │ ann    │ editor   │
  └────────┴──────────┘
  ```

- `RIGHT OUTER JOIN` works in a similar way — performs `INNER JOIN` first, and then adds all non-matching rows from the right table with default values in columns of the left table to the joined table, so that the joined table has at least one row for each row of the right table.

  Example

  ```
  SELECT users.name AS user, roles.title AS role FROM users RIGHT OUTER JOIN roles ON users.role_id=roles.id;
  ```

  The query returns all matches from the `users` and `roles` tables on role identifiers, as well as a row for the `view only` role that is not assigned to any user (with an empty string in the `user` column):

  ```
  ┌─user───┬─role─────┐
  │ john   │ admin    │
  │ mike   │ owner    │
  │ tom    │ author   │
  │ mary   │ author   │
  │ ada    │ reviewer │
  │ andrew │ reviewer │
  │ ann    │ editor   │
  └────────┴──────────┘
  ┌─user─┬─role──────┐
  │      │ view only │
  └──────┴───────────┘
  ```

- `FULL OUTER JOIN` combines `LEFT OUTER JOIN` and `RIGHT OUTER JOIN` — it returns the `INNER JOIN` result and all non-matching rows from both tables with default values in empty cells.

  Example

  ```
  SELECT users.name AS user, roles.title AS role FROM users FULL OUTER JOIN roles ON users.role_id=roles.id;
  ```

  The result includes all users from the `users` table with their roles (including the `harry` user whose role is not defined in the `roles` table), as well as a role from the `roles` table that is not assigned to any user:

  ```
  ┌─user───┬─role─────┐
  │ john   │ admin    │
  │ mike   │ owner    │
  │ tom    │ author   │
  │ mary   │ author   │
  │ ada    │ reviewer │
  │ andrew │ reviewer │
  │ harry  │          │
  │ ann    │ editor   │
  └────────┴──────────┘
  ┌─user─┬─role──────┐
  │      │ view only │
  └──────┴───────────┘
  ```

To use NULL values in empty cells instead of default values (as in standard SQL), change the join_use_nulls setting value.

CROSS JOIN

The `CROSS JOIN` query combines each row of the left table with each row of the right table (creates the Cartesian product). The column set of the joined table includes all columns of the left table followed by all columns of the right table. If tables have `N` and `M` rows respectively, a joined table has `N * M` rows. `CROSS JOIN` is equivalent to `FROM <left_table> INNER JOIN <right_table> ON TRUE` or `FROM <left_table>, <right_table>`.

Example

```
SELECT * FROM users CROSS JOIN roles;
```

The result contains all possible combinations of rows from the `users` and `roles` tables:

```
┌─user_id─┬─name───┬─role_id─┬─id─┬─title─────┐
│       1 │ john   │      10 │ 10 │ admin     │
│       1 │ john   │      10 │ 20 │ owner     │
│       1 │ john   │      10 │ 30 │ author    │
│       1 │ john   │      10 │ 40 │ reviewer  │
│       1 │ john   │      10 │ 50 │ editor    │
│       1 │ john   │      10 │ 60 │ view only │
│       2 │ mike   │      20 │ 10 │ admin     │
│       2 │ mike   │      20 │ 20 │ owner     │
│       2 │ mike   │      20 │ 30 │ author    │
│       2 │ mike   │      20 │ 40 │ reviewer  │
│       2 │ mike   │      20 │ 50 │ editor    │
│       2 │ mike   │      20 │ 60 │ view only │
│       3 │ tom    │      30 │ 10 │ admin     │
│       3 │ tom    │      30 │ 20 │ owner     │
│       3 │ tom    │      30 │ 30 │ author    │
│       3 │ tom    │      30 │ 40 │ reviewer  │
│       3 │ tom    │      30 │ 50 │ editor    │
│       3 │ tom    │      30 │ 60 │ view only │
│       4 │ mary   │      30 │ 10 │ admin     │
│       4 │ mary   │      30 │ 20 │ owner     │
│       4 │ mary   │      30 │ 30 │ author    │
│       4 │ mary   │      30 │ 40 │ reviewer  │
│       4 │ mary   │      30 │ 50 │ editor    │
│       4 │ mary   │      30 │ 60 │ view only │
│       5 │ ada    │      40 │ 10 │ admin     │
│       5 │ ada    │      40 │ 20 │ owner     │
│       5 │ ada    │      40 │ 30 │ author    │
│       5 │ ada    │      40 │ 40 │ reviewer  │
│       5 │ ada    │      40 │ 50 │ editor    │
│       5 │ ada    │      40 │ 60 │ view only │
│       6 │ andrew │      40 │ 10 │ admin     │
│       6 │ andrew │      40 │ 20 │ owner     │
│       6 │ andrew │      40 │ 30 │ author    │
│       6 │ andrew │      40 │ 40 │ reviewer  │
│       6 │ andrew │      40 │ 50 │ editor    │
│       6 │ andrew │      40 │ 60 │ view only │
│       7 │ harry  │      70 │ 10 │ admin     │
│       7 │ harry  │      70 │ 20 │ owner     │
│       7 │ harry  │      70 │ 30 │ author    │
│       7 │ harry  │      70 │ 40 │ reviewer  │
│       7 │ harry  │      70 │ 50 │ editor    │
│       7 │ harry  │      70 │ 60 │ view only │
│       8 │ ann    │      50 │ 10 │ admin     │
│       8 │ ann    │      50 │ 20 │ owner     │
│       8 │ ann    │      50 │ 30 │ author    │
│       8 │ ann    │      50 │ 40 │ reviewer  │
│       8 │ ann    │      50 │ 50 │ editor    │
│       8 │ ann    │      50 │ 60 │ view only │
└─────────┴────────┴─────────┴────┴───────────┘
```

LEFT/RIGHT SEMI JOIN

The `LEFT SEMI JOIN` query returns column values for each row from the left table that has at least one match on the join condition in the right table. If more than one match is found for a row, only the first found match is returned (Cartesian product is disabled).

`RIGHT SEMI JOIN` works similarly — it returns column values for all rows from the right table that have at least one match on the join condition in the left table. If multiple matches are found for a row, only the first found match is added to the result.

Example (RIGHT SEMI JOIN)

```
SELECT roles.title AS role, users.name AS user FROM users RIGHT SEMI JOIN roles ON users.role_id=roles.id;
```

For each role from the right `roles` table, the `RIGHT SEMI JOIN` query returns only one user that has been assigned the role. A `view only` role that has no match in the `users` table is not included in the resulting table:

```
┌─role─────┬─user─┐
│ admin    │ john │
│ owner    │ mike │
│ author   │ tom  │
│ reviewer │ ada  │
│ editor   │ ann  │
└──────────┴──────┘
```

LEFT/RIGHT ANTI JOIN

`LEFT ANTI JOIN` returns column values for all rows of the left table that do not match any rows in the right table on the specified join condition. Similarly, `RIGHT ANTI JOIN` returns column values for the right table's rows that have no matches in the left table.

Example (LEFT ANTI JOIN)

```
SELECT users.name AS user, roles.title AS role FROM users LEFT ANTI JOIN roles ON users.role_id=roles.id;
```

The `LEFT ANTI JOIN` query returns a data row only for the `harry` user, whose role is not found in the `roles` table on the specified identifier:

```
┌─user──┬─role─┐
│ harry │      │
└───────┴──────┘
```

LEFT/RIGHT/INNER ANY JOIN

The `LEFT ANY JOIN` query returns column values of each row from the left table, combining them either with column values of matching rows from the right table, or with default values in the right table columns (if no matches were found). If a row of the left table has more than one match in the right table, the query returns the first found match only. `RIGHT ANY JOIN` works similarly for the right table's rows.

In other words, the `ANY` keyword in queries indicates the strictness of the `JOIN` operation - the Cartesian product for rows of source tables matching on the join condition is not created and only one of the found matches for each row is included in the result. The default strictness is `ALL` — all matching rows from source tables are joined and written to the resulting table (see the LEFT/RIGHT OUTER JOIN and INNER JOIN query examples above). So, `LEFT/RIGHT ANY JOIN` joins tables as `LEFT/RIGHT OUTER JOIN`, and `INNER ANY JOIN` as `INNER JOIN`, but with disabled Cartesian product.

The default strictness value can be adjusted via the join_default_strictness setting.

Example (RIGHT ANY JOIN)

```
SELECT users.name AS user, roles.title AS role FROM users RIGHT ANY JOIN roles ON users.role_id=roles.id;
```

For each role from the right `roles` table, the `RIGHT ANY JOIN` query returns only one user (the first one found) to which this role is assigned. The result also includes the `view only` role that is not assigned to any user:

```
┌─user─┬─role─────┐
│ john │ admin    │
│ mike │ owner    │
│ tom  │ author   │
│ ada  │ reviewer │
│ ann  │ editor   │
└──────┴──────────┘
┌─user─┬─role──────┐
│      │ view only │
└──────┴───────────┘
```

ASOF JOIN

The `ASOF JOIN` and `LEFT ASOF JOIN` queries are used when it is necessary to join rows that do not have an exact match. If a row from the left table does not have an exact match with any row in the right table, then the closest matching row from the right table is used as a match instead. The closest match condition should be based on a special column (of the Int, UInt, Float, Date, DateTime, or Decimal type) that both tables contain.

General query syntax:

- `ASOF JOIN … ON`:

```
<left_table> [LEFT] ASOF JOIN <right_table> ON <exact_match_condition> AND <closest_match_condition>
```

  where:

  - `<exact_match_condition>` — condition for joining by equality (multiple conditions can be set);
  - `<closest_match_condition>` — condition for joining by the closest match (one condition should be set). The closest match condition supports the `>`, `>=`, `<`, and `<=` operators.

- `ASOF JOIN … USING`:

```
<left_table> [LEFT] ASOF JOIN <right_table> USING <exact_match_column1>, ... <exact_match_columnN>, <closest_match_column>
```

  where

  - `<exact_match_column1>, … <exact_match_columnN>` — columns for joining by equality of values;
  - `<closest_match_column>` — column (should be the last one in the `USING` list) for joining by the closest match. A join condition is `<left_table>.<closest_match_column> >= <right_table>.<closest_match_column>`.

Example (LEFT ASOF JOIN)

Create two tables for test examples:

- `table1`

```
CREATE TABLE table1 (id Int32, time DateTime, text String) ENGINE = MergeTree ORDER BY id;
```

```
INSERT INTO table1 VALUES
    (50, '2023-03-10 14:55:00', 'text1_0'),
    (50, '2023-03-10 15:00:00', 'text1_1'),
    (50, '2023-03-10 15:03:00', 'text1_2'),
    (50, '2023-03-10 15:10:00', 'text1_3'),
    (50, '2023-03-10 15:14:00', 'text1_4');
```

```
┌─id─┬────────────────time─┬─text────┐
│ 50 │ 2023-03-10 14:55:00 │ text1_0 │
│ 50 │ 2023-03-10 15:00:00 │ text1_1 │
│ 50 │ 2023-03-10 15:03:00 │ text1_2 │
│ 50 │ 2023-03-10 15:10:00 │ text1_3 │
│ 50 │ 2023-03-10 15:14:00 │ text1_4 │
└────┴─────────────────────┴─────────┘
```

- `table2`

```
CREATE TABLE table2 (id Int32, time DateTime, text String) ENGINE = MergeTree ORDER BY id;
```

```
INSERT INTO table2 VALUES
    (50, '2023-03-10 15:00:00', 'text2_1'),
    (50, '2023-03-10 15:07:00', 'text2_2'),
    (50, '2023-03-10 15:11:00', 'text2_3'),
    (50, '2023-03-10 15:20:00', 'text2_4');
```

```
┌─id─┬────────────────time─┬─text────┐
│ 50 │ 2023-03-10 15:00:00 │ text2_1 │
│ 50 │ 2023-03-10 15:07:00 │ text2_2 │
│ 50 │ 2023-03-10 15:11:00 │ text2_3 │
│ 50 │ 2023-03-10 15:20:00 │ text2_4 │
└────┴─────────────────────┴─────────┘
```

Run the `ASOF LEFT JOIN` query:

```
SELECT
    table1.text AS table1_text,
    table1.time AS table1_time,
    table2.text AS table2_text,
    table2.time AS table2_time
FROM table1
ASOF LEFT JOIN table2 ON (table1.id = table2.id) AND (table1.time >= table2.time);
```

In the resulting table, each row of `table1` is combined with a row from `table2` with the closest value in the `time` column according to the `table1.time >= table2.time` condition. For the first row (`text1_0`), no matching row was found in `table2`, so the `table2_text` and `table2_time` columns contain default values corresponding to column data types:

```
┌─table1_text─┬─────────table1_time─┬─table2_text─┬─────────table2_time─┐
│ text1_0     │ 2023-03-10 14:55:00 │             │ 1970-01-01 00:00:00 │
│ text1_1     │ 2023-03-10 15:00:00 │ text2_1     │ 2023-03-10 15:00:00 │
│ text1_2     │ 2023-03-10 15:03:00 │ text2_1     │ 2023-03-10 15:00:00 │
│ text1_3     │ 2023-03-10 15:10:00 │ text2_2     │ 2023-03-10 15:07:00 │
│ text1_4     │ 2023-03-10 15:14:00 │ text2_3     │ 2023-03-10 15:11:00 │
└─────────────┴─────────────────────┴─────────────┴─────────────────────┘
```

Run the `ASOF LEFT JOIN` query with the `USING` clause:

```
SELECT
    table1.text AS table1_text,
    table1.time AS table1_time,
    table2.text AS table2_text,
    table2.time AS table2_time
FROM table1
ASOF LEFT JOIN table2 USING (id, time);
```

The resulting table is the same:

```
┌─table1_text─┬─────────table1_time─┬─table2_text─┬─────────table2_time─┐
│ text1_0     │ 2023-03-10 14:55:00 │             │ 1970-01-01 00:00:00 │
│ text1_1     │ 2023-03-10 15:00:00 │ text2_1     │ 2023-03-10 15:00:00 │
│ text1_2     │ 2023-03-10 15:03:00 │ text2_1     │ 2023-03-10 15:00:00 │
│ text1_3     │ 2023-03-10 15:10:00 │ text2_2     │ 2023-03-10 15:07:00 │
│ text1_4     │ 2023-03-10 15:14:00 │ text2_3     │ 2023-03-10 15:11:00 │
└─────────────┴─────────────────────┴─────────────┴─────────────────────┘
```

JOIN conditions

`JOIN` combines rows of two tables based on the specified columns — **join keys**. Use the `ON` or `USING` clause for the `JOIN` operator to set join keys and additional join conditions if required.

CROSS JOIN queries do not require join conditions.

ON

The `ON` clause returns a boolean value. A row from one table matches a row from another table if the `ON` expression returns `true`.

You can specify multiple conditions using the `AND` and/or `OR` operators — in this case, rows are joined if the entire compound condition is met. If a condition is not met, rows can still be included in the result depending on the `JOIN` type .

A condition that determines a join key should use the equality operator (`=`) to compare columns of the left and right tables. Other conditions may use other logical operators, and each individual condition should be specified either for the left or right table.

Example

The following query specifies two conditions for joining tables:

- `users.role_id=roles.id` — join key;
- `roles.id>20` — additional condition for the `roles` table.

```
SELECT
    users.name AS user,
    roles.title AS role,
    roles.id AS role_id
FROM users
LEFT JOIN roles ON users.role_id = roles.id AND roles.id > 20;
```

The result:

```
┌─user───┬─role─────┬─role_id─┐
│ john   │          │       0 │
│ mike   │          │       0 │
│ tom    │ author   │      30 │
│ mary   │ author   │      30 │
│ ada    │ reviewer │      40 │
│ andrew │ reviewer │      40 │
│ harry  │          │       0 │
│ ann    │ editor   │      50 │
└────────┴──────────┴─────────┘
```

USING

You can utilize the `USING` clause instead of `ON` when columns in both tables have the same names. Pass a comma-separated list of column names as a parameter to the `USING` clause — the equality of these columns will be considered a condition for joining.

The result of the `JOIN` clause with `USING` contains a single column for each of the listed column pairs in the specified order, followed by all remaining columns from the left and right tables.

Example

Create two test tables:

- `test_table1`

```
CREATE TABLE test_table1 (num Int32, name String) ENGINE = MergeTree ORDER BY num;
```

```
INSERT INTO test_table1 VALUES (1, 'a'), (2, 'b'), (3, 'c');
```

- `test_table2`

```
CREATE TABLE test_table2 (num Int32, value String) ENGINE = MergeTree ORDER BY num;
```

```
INSERT INTO test_table2 VALUES (0, 'value1'), (1, 'value2'), (2, 'value3');
```

```
  test_table1        test_table2

┌─num─┬─name─┐     ┌─num─┬─value──┐
│   1 │ a    │     │   0 │ value1 │
│   2 │ b    │     │   1 │ value2 │
│   3 │ c    │     │   2 │ value3 │
└─────┴──────┘     └─────┴────────┘
```

Execute the following query with the `JOIN … USING` clause to join the `test_table2` and `test_table2` tables on the `num` column:

```
SELECT * FROM test_table1 INNER JOIN test_table2 USING num;
```

The output:

```
┌─num─┬─name─┬─value──┐
│   1 │ a    │ value2 │
│   2 │ b    │ value3 │
└─────┴──────┴────────┘
```

Subqueries

Elena Dvoryadkina

Contents

To Table of Contents

- Overview
- Subqueries in FROM
- Subqueries in JOIN
- Subqueries in WHERE
  - IN
  - NOT IN
  - ANY
  - ALL
  - EXISTS

Overview

A **subquery** is an SQL query nested within another (parent) query or subquery. A subquery is executed first, and then its results are passed to the parent query. Thus, subqueries can be useful when you need to use the results of one query as the input for another query. For example, you can use subqueries in `SELECT` queries: in the `FROM` and `JOIN` clauses to specify a source from which data should be read or in the `WHERE` clause to filter data.

Operations for which a subquery can be used depend on the type of data the subquery returns. The result of a subquery can be:

- Scalar value — when a subquery returns a value of one column from one row or a single calculated value (for example, an aggregate function result — sum, average/minimum/maximum value of a column). Such a subquery can be used with the comparison operators `>`, `<`, `=`, and others (see Comparison Operators in the ClickHouse documentation), passed as an argument to a function, or used as a column value in the `SELECT` clause.
- Table value — when a subquery returns multiple rows (possibly zero rows). In this case, a subquery can be used with operators such as `IN`, `NOT IN`, `ANY`, `ALL`, `EXISTS` to check the occurrence of a row from the parent query in the subquery results. Also, a table value from a subquery can be used in the `FROM` or `JOIN` clause to specify a source table.

NOTE

- Syntactically, a subquery is a `SELECT` query enclosed in parentheses.
- Aliases are not visible in subqueries and between subqueries.

Below are various examples of using subqueries in `SELECT` queries.

Create tables for test examples

To reproduce examples from the article, create the following tables first:

- `employees`

```
CREATE TABLE employees (
    id UInt16,
    full_name String,
    hire_date Date,
    position Enum('c developer' = 1,
                  'python developer' = 2,
                  'java developer' = 3,
                  'frontend developer' = 4,
                  'test automation engineer' = 5,
                  'manual testing engineer' = 6,
                  'load testing engineer' = 7,
                  'support engineer 1st line' = 8,
                  'support engineer 2nd line' = 9),
    department Enum('development' = 10, 'qa' = 20, 'tech support' = 30),
    location String,
    email String)
ENGINE = MergeTree
ORDER BY id;
```

```
INSERT INTO employees VALUES
    (1, 'James Wilson', '2020-11-23', 1, 10, 'New York', 'jwilson@example.com'),
    (2, 'Mary Burton', '2023-05-06', 1, 10, 'Chicago', 'mburton@example.com'),
    (3, 'Patricia Robinson', '2021-10-11', 2, 10, 'Los Angeles', 'probinson@example.com'),
    (4, 'Robert Gray', '2019-07-03', 2, 10, 'Seattle', 'rgray@example.com'),
    (5, 'Elizabeth Tucker', '2024-02-03', 4, 10, 'Chicago', 'etucker@example.com'),
    (6, 'Joseph Lewis', '2023-04-05', 4, 10, 'New York', 'jlewis@example.com'),
    (7, 'Paul Brown', '2024-02-03', 3, 10, 'Chicago', 'pbrown@example.com'),
    (8, 'Andrew Clarke', '2024-11-23', 3, 10, 'Los Angeles', 'aclarke@example.com')
    (9, 'William Ferguson', '2023-11-12', 5, 20, 'Chicago', 'wferguson@example.com'),
    (10, 'Linda Black', '2021-07-20', 6, 20, 'New York', 'lblack@example.com'),
    (11, 'David Green', '2019-10-10', 7, 20, 'Chicago', 'dgreen@example.com'),
    (12, 'Daniel Gray', '2020-08-03', 7, 20, 'New York', 'dgray@example.com'),
    (13, 'Mark Armstrong', '2020-04-05', 8, 30, 'New York', 'marmstrong@example.com'),
    (14, 'Donald Carter', '2021-07-07', 8, 30, 'Seattle', 'dcarter@example.com'),
    (15, 'Elizabeth Collins', '2022-01-22', 9, 30, 'Chicago', 'ecollins@example.com');
```

```
    ┌─id─┬─full_name─────────┬──hire_date─┬─position──────────────────┬─department───┬─location────┬─email──────────────────┐
 1. │  1 │ James Wilson      │ 2020-11-23 │ c developer               │ development  │ New York    │ jwilson@example.com    │
 2. │  2 │ Mary Burton       │ 2023-05-06 │ c developer               │ development  │ Chicago     │ mburton@example.com    │
 3. │  3 │ Patricia Robinson │ 2021-10-11 │ python developer          │ development  │ Los Angeles │ probinson@example.com  │
 4. │  4 │ Robert Gray       │ 2019-07-03 │ python developer          │ development  │ Seattle     │ rgray@example.com      │
 5. │  5 │ Elizabeth Tucker  │ 2024-02-03 │ frontend developer        │ development  │ Chicago     │ etucker@example.com    │
 6. │  6 │ Joseph Lewis      │ 2023-04-05 │ frontend developer        │ development  │ New York    │ jlewis@example.com     │
 7. │  7 │ Paul Brown        │ 2024-02-03 │ java developer            │ development  │ Chicago     │ pbrown@example.com     │
 8. │  8 │ Andrew Clarke     │ 2024-11-23 │ java developer            │ development  │ Los Angeles │ aclarke@example.com    │
 9. │  9 │ William Ferguson  │ 2023-11-12 │ test automation engineer  │ qa           │ Chicago     │ wferguson@example.com  │
10. │ 10 │ Linda Black       │ 2021-07-20 │ manual testing engineer   │ qa           │ New York    │ lblack@example.com     │
11. │ 11 │ David Green       │ 2019-10-10 │ load testing engineer     │ qa           │ Chicago     │ dgreen@example.com     │
12. │ 12 │ Daniel Gray       │ 2020-08-03 │ load testing engineer     │ qa           │ New York    │ dgray@example.com      │
13. │ 13 │ Mark Armstrong    │ 2020-04-05 │ support engineer 1st line │ tech support │ New York    │ marmstrong@example.com │
14. │ 14 │ Donald Carter     │ 2021-07-07 │ support engineer 1st line │ tech support │ Seattle     │ dcarter@example.com    │
15. │ 15 │ Elizabeth Collins │ 2022-01-22 │ support engineer 2nd line │ tech support │ Chicago     │ ecollins@example.com   │
    └────┴───────────────────┴────────────┴───────────────────────────┴──────────────┴─────────────┴────────────────────────┘
```

- `employee_bonus_oct_24`

```
CREATE TABLE employee_bonus_oct_24 (employee_id UInt16, bonus UInt32) ENGINE = MergeTree ORDER BY employee_id;
```

```
INSERT INTO employee_bonus_oct_24
VALUES (1, 700), (2, 450), (3, 500), (4, 950), (5, 370), (6, 600),(7, 1000), (8, 950),
       (9, 770), (10, 350), (11, 500), (12, 670), (13, 600), (14, 600), (15, 480);
```

```
    ┌─employee_id─┬─bonus─┐
 1. │           1 │   700 │
 2. │           2 │   450 │
 3. │           3 │   500 │
 4. │           4 │   950 │
 5. │           5 │   370 │
 6. │           6 │   600 │
 7. │           7 │  1000 │
 8. │           8 │   950 │
 9. │           9 │   770 │
10. │          10 │   350 │
11. │          11 │   500 │
12. │          12 │   670 │
13. │          13 │   600 │
14. │          14 │   600 │
15. │          15 │   480 │
    └─────────────┴───────┘
```

- `employee_bonus_oct_24_full`

```
CREATE TABLE employee_bonus_oct_24_full (
    employee_id UInt16,
    full_name String,
    position String,
    department String,
    bonus UInt32)
ENGINE = MergeTree
ORDER BY employee_id;
```

```
INSERT INTO employee_bonus_oct_24_full SELECT
    employee_bonus_oct_24.employee_id,
    employees.full_name,
    employees.position,
    employees.department,
    employee_bonus_oct_24.bonus
FROM employee_bonus_oct_24
INNER JOIN employees ON employee_bonus_oct_24.employee_id = employees.id;
```

```
    ┌─employee_id─┬─full_name─────────┬─position──────────────────┬─department───┬─bonus─┐
 1. │           1 │ James Wilson      │ c developer               │ development  │   700 │
 2. │           2 │ Mary Burton       │ c developer               │ development  │   450 │
 3. │           3 │ Patricia Robinson │ python developer          │ development  │   500 │
 4. │           4 │ Robert Gray       │ python developer          │ development  │   950 │
 5. │           5 │ Elizabeth Tucker  │ frontend developer        │ development  │   370 │
 6. │           6 │ Joseph Lewis      │ frontend developer        │ development  │   600 │
 7. │           7 │ Paul Brown        │ java developer            │ development  │  1000 │
 8. │           8 │ Andrew Clarke     │ java developer            │ development  │   950 │
 9. │           9 │ William Ferguson  │ test automation engineer  │ qa           │   770 │
10. │          10 │ Linda Black       │ manual testing engineer   │ qa           │   350 │
11. │          11 │ David Green       │ load testing engineer     │ qa           │   500 │
12. │          12 │ Daniel Gray       │ load testing engineer     │ qa           │   670 │
13. │          13 │ Mark Armstrong    │ support engineer 1st line │ tech support │   600 │
14. │          14 │ Donald Carter     │ support engineer 1st line │ tech support │   600 │
15. │          15 │ Elizabeth Collins │ support engineer 2nd line │ tech support │   480 │
    └─────────────┴───────────────────┴───────────────────────────┴──────────────┴───────┘
```

Subqueries in FROM

Instead of a table name or table function, you can use a subquery in the `FROM` clause to specify a data source for a `SELECT` query:

```
SELECT <expression> FROM (<subquery>) [AS <alias>]
```

In contrast to standard SQL, it is not necessary to specify an alias after a subquery (`AS <alias>`). You can do this for compatibility, but the specified name is not used anywhere.

A subquery in the following query gets bonuses of employees in the `development` department for October and calculates the value of each employee's bonus relative to the amount of the maximum bonus in the company:

```
SELECT employee, bonus, relative_to_max
FROM
(
    SELECT
        full_name AS employee,
        bonus,
        round((bonus/(SELECT max(bonus) FROM employee_bonus_oct_24_full)), 2) AS relative_to_max
    FROM employee_bonus_oct_24_full
    WHERE department = 'development'
)
WHERE relative_to_max < 0.5;
```

```
   ┌─employee─────────┬─bonus─┬─relative_to_max─┐
1. │ Mary Burton      │   450 │            0.45 │
2. │ Elizabeth Tucker │   370 │            0.37 │
   └──────────────────┴───────┴─────────────────┘
```

Subqueries in JOIN

In a `SELECT` query with the JOIN clause, there is no optimization of the order in which the join operation is executed relative to other stages of the query. The join operation (search in the right table) is performed before `WHERE` filtering and before aggregation. To explicitly specify the order of calculations and improve query performance, it is recommended to apply a filter as early as possible — for this purpose, you can join subqueries.

For example, the following query first evaluates the subqueries in the `FROM` (number of employees in each department) and `LEFT ANY JOIN` (number of employees in each department who received bonuses less than 500) clauses with grouping and filtering, and then joins the results of these subqueries:

```
SELECT department, number_of_employees, bonus_less_500
FROM
(
    SELECT department, count() AS number_of_employees
    FROM employees
    GROUP BY department
)
LEFT ANY JOIN
(
    SELECT department, count() AS bonus_less_500
    FROM employee_bonus_oct_24_full
    WHERE bonus < 500
    GROUP BY department
) USING (department)
SETTINGS joined_subquery_requires_alias = 0;
```

```
   ┌─department───┬─number_of_employees─┬─bonus_less_500─┐
1. │ development  │                   8 │              2 │
2. │ qa           │                   4 │              1 │
3. │ tech support │                   3 │              1 │
   └──────────────┴─────────────────────┴────────────────┘
```

Subqueries in WHERE

Subqueries are often used within the `WHERE` clause to filter data. As mentioned above, subqueries can be used with various comparison operators to define data filtering conditions:

- subqueries that return scalar values can be used before or after comparison operators such as `=` (`==`), `!=` (`<>`), `>`, `>=`, `<`, `<=`. For example, the following query finds all employees whose bonuses in October are greater than the average:

```
SELECT full_name AS employee, position, department
FROM employee_bonus_oct_24_full
WHERE bonus > (SELECT avg(bonus) FROM employee_bonus_oct_24_full);
```

```
   ┌─employee─────────┬─position─────────────────┬─department──┐
1. │ James Wilson     │ c developer              │ development │
2. │ Robert Gray      │ python developer         │ development │
3. │ Paul Brown       │ java developer           │ development │
4. │ Andrew Clarke    │ java developer           │ development │
5. │ William Ferguson │ test automation engineer │ qa          │
6. │ Daniel Gray      │ load testing engineer    │ qa          │
   └──────────────────┴──────────────────────────┴─────────────┘
```

- subqueries that return multiple data rows can be used with the `IN`, `NOT IN`, `ANY`, `ALL`, `EXISTS` operators, allowing you to check for the existence of a row in a set of rows. Examples of using subqueries with these operators are given in the sections below.

IN

Syntax of the `IN` operator with a subquery:

```
<IN_expression> IN (<IN_subquery>)
```

The `IN` operator evaluates the `IN_expression` expression, compares it with values in all rows of the `IN_subquery` result, and returns:

- `1` — if a row equal to `IN_expression` is found in the results of `IN_subquery`;
- `0` — in the following cases:
  - no rows equal to `IN_expression` were found in the results of `IN_subquery`;
  - `IN_subquery` does not return any rows;
  - there is a `NULL` value in the right or left part of the `IN` operator and the transform_null_in parameter is set to `0` (by default).

The following sample query uses the `IN` operator to display information about bonuses paid in October only to employees living in New York:

```
SELECT full_name AS employee, bonus FROM employee_bonus_oct_24_full
WHERE employee_id IN (SELECT id FROM employees WHERE location='New York');
```

```
   ┌─employee───────┬─bonus─┐
1. │ James Wilson   │   700 │
2. │ Joseph Lewis   │   600 │
3. │ Linda Black    │   350 │
4. │ Daniel Gray    │   670 │
5. │ Mark Armstrong │   600 │
   └────────────────┴───────┘
```

NOTE

- A subquery can specify more than one column to filter tuples (types of columns on the left and right of the `IN` operator should be the same):

```
(<column1>, <column2>) IN (SELECT <IN_subquery_result_column1>, <IN_subquery_result_column2> FROM ...)
```

- The `IN` operator with a subquery can be used in any part of the parent query, including in aggregate functions and lambda functions.
- For processing a distributed query, it is often more optimal to use the `GLOBAL IN` operator instead of `IN` with a subquery — see detailed information in the Distributed Subqueries section of the ClickHouse documentation.

NOT IN

Syntax of the `NOT IN` operator with a subquery:

```
<NOT_IN_expression> NOT IN (<NOT_IN_subquery>)
```

The `NOT IN` operator evaluates `NOT_IN_expression`, compares it with each row of the `NOT_IN_subquery` result, and returns:

- `1` — if no matches were found in the results of `NOT_IN_expression` and `NOT_IN_subquery`, or the subquery returns no rows;
- `0` — if the `NOT_IN_subquery` results contain at least one row equal to `NOT_IN_expression`.

For example, the following query returns data on bonuses received by employees who do not live in New York:

```
SELECT full_name AS employee, bonus FROM employee_bonus_oct_24_full
WHERE employee_id NOT IN (SELECT id FROM employees WHERE location='New York');
```

```
    ┌─employee──────────┬─bonus─┐
 1. │ Mary Burton       │   450 │
 2. │ Patricia Robinson │   500 │
 3. │ Robert Gray       │   950 │
 4. │ Elizabeth Tucker  │   370 │
 5. │ Paul Brown        │  1000 │
 6. │ Andrew Clarke     │   950 │
 7. │ William Ferguson  │   770 │
 8. │ David Green       │   500 │
 9. │ Donald Carter     │   600 │
10. │ Elizabeth Collins │   480 │
    └───────────────────┴───────┘
```

ANY

Syntax of the `ANY` operator with a subquery:

```
<ANY_expression> <operator> ANY (<ANY_subquery>)
```

where `operator` can be `=`, `<>`, `<`, `<=`, `>`, `>=`.

The `ANY` operator compares the results of `ANY_expression` and `ANY_subquery` using `operator` and returns `1` if at least one element from the subquery result satisfies the specified condition.

The following query finds employees not from the `development` department whose bonuses in October are greater than the bonus of at least one employee in the `development` department in the same month:

```
SELECT full_name AS employee, bonus, department FROM employee_bonus_oct_24_full
WHERE bonus > ANY (SELECT bonus FROM employee_bonus_oct_24_full WHERE department='development') AND department!='development';
```

```
   ┌─employee──────────┬─bonus─┬─department───┐
1. │ William Ferguson  │   770 │ qa           │
2. │ David Green       │   500 │ qa           │
3. │ Daniel Gray       │   670 │ qa           │
4. │ Mark Armstrong    │   600 │ tech support │
5. │ Donald Carter     │   600 │ tech support │
6. │ Elizabeth Collins │   480 │ tech support │
   └───────────────────┴───────┴──────────────┘
```

`= ANY` is equivalent to IN. If you rewrite the above example with `IN` as follows, the query result will be the same:

```
SELECT full_name AS employee, bonus FROM employee_bonus_oct_24_full
WHERE employee_id = ANY (SELECT id FROM employees WHERE location='New York');
```

```
   ┌─employee───────┬─bonus─┐
1. │ James Wilson   │   700 │
2. │ Joseph Lewis   │   600 │
3. │ Linda Black    │   350 │
4. │ Daniel Gray    │   670 │
5. │ Mark Armstrong │   600 │
   └────────────────┴───────┘
```

ALL

Syntax of the `ALL` operator with a subquery:

```
<ALL_expression> <operator> ALL (<ALL_subquery>)
```

where `operator` can be `=`, `<>`, `<`, `<=`, `>`, `>=`.

The `ALL` operator compares the value of `ALL_expression` with each value in the set returned by `ALL_subquery` and returns `1` if the comparison condition is true for all values in the set.

For example, the following query returns employees whose bonuses are less than the bonus of any employee in the `tech support` department:

```
SELECT full_name AS employee, bonus, department FROM employee_bonus_oct_24_full
WHERE bonus < ALL (SELECT bonus FROM employee_bonus_oct_24_full WHERE department='tech support');
```

```
   ┌─employee─────────┬─bonus─┬─department──┐
1. │ Mary Burton      │   450 │ development │
2. │ Elizabeth Tucker │   370 │ development │
3. │ Linda Black      │   350 │ qa          │
   └──────────────────┴───────┴─────────────┘
```

`!= ALL` is equivalent to NOT IN. The above example with `NOT IN` can be rewritten as follows:

```
SELECT full_name AS employee, bonus FROM employee_bonus_oct_24_full
WHERE employee_id != ALL (SELECT id FROM employees WHERE location='New York');
```

The result will be the same:

```
    ┌─employee──────────┬─bonus─┐
 1. │ Mary Burton       │   450 │
 2. │ Patricia Robinson │   500 │
 3. │ Robert Gray       │   950 │
 4. │ Elizabeth Tucker  │   370 │
 5. │ Paul Brown        │  1000 │
 6. │ Andrew Clarke     │   950 │
 7. │ William Ferguson  │   770 │
 8. │ David Green       │   500 │
 9. │ Donald Carter     │   600 │
10. │ Elizabeth Collins │   480 │
    └───────────────────┴───────┘
```

EXISTS

Syntax of the `EXISTS` operator:

```
EXISTS (<subquery>)
```

It checks how many rows the subquery result contains. If the result is empty, the operator returns `0`, otherwise — `1`.

CAUTION

References to tables and columns of a parent query are not supported in a subquery.

For example, the following query displays the `employees` table only if it contains at least one employee who lives in Chicago:

```
SELECT full_name AS employee, location AS city
FROM employees
WHERE EXISTS (SELECT * FROM employees WHERE location='Chicago');
```

```
    ┌─employee──────────┬─city────────┐
 1. │ James Wilson      │ New York    │
 2. │ Mary Burton       │ Chicago     │
 3. │ Patricia Robinson │ Los Angeles │
 4. │ Robert Gray       │ Seattle     │
 5. │ Elizabeth Tucker  │ Chicago     │
 6. │ Joseph Lewis      │ New York    │
 7. │ Paul Brown        │ Chicago     │
 8. │ Andrew Clarke     │ Los Angeles │
 9. │ William Ferguson  │ Chicago     │
10. │ Linda Black       │ New York    │
11. │ David Green       │ Chicago     │
12. │ Daniel Gray       │ New York    │
13. │ Mark Armstrong    │ New York    │
14. │ Donald Carter     │ Seattle     │
15. │ Elizabeth Collins │ Chicago     │
    └───────────────────┴─────────────┘
```

Combine queries (UNION, INTERSECT, EXCEPT)

Elena Dvoryadkina

Contents

To Table of Contents

UNION
- Examples

INTERSECT
- Examples

EXCEPT
- Examples

To combine results of two or more queries, you can use the UNION, INTERSECT, and EXCEPT set operators.

Create tables for test examples

To reproduce examples from this article and see how various set operators work in ClickHouse, first create the tables:

- `products_2023` — product catalog for 2023:

```
CREATE TABLE products_2023
(
    id UInt32,
    name String,
    category String,
    price Float32
) ENGINE = MergeTree()
ORDER BY id;
```

```
INSERT INTO products_2023 VALUES
(1, 'Office Chair', 'Furniture', 170),
(2, 'Backpack', 'Accessories', 60),
(3, 'Monitor', 'Electronics', 200),
(4, 'Tablet', 'Electronics', 750),
(5, 'Water Bottle', 'Accessories', 10),
(6, 'Office Desk', 'Furniture', 250);
```

```
   ┌─id─┬─name─────────┬─category────┬─price─┐
1. │  1 │ Office Chair │ Furniture   │   170 │
2. │  2 │ Backpack     │ Accessories │    60 │
3. │  3 │ Monitor      │ Electronics │   200 │
4. │  4 │ Tablet       │ Electronics │   750 │
5. │  5 │ Water Bottle │ Accessories │    10 │
6. │  6 │ Office Desk  │ Furniture   │   250 │
   └────┴──────────────┴─────────────┴───────┘
```

- `products_2024` — product catalog for 2024:

```
CREATE TABLE products_2024
(
    id UInt32,
    name String,
    category String,
    price Float32
) ENGINE = MergeTree()
ORDER BY id;
```

```
INSERT INTO products_2024 VALUES
(1, 'Office Chair', 'Furniture', 175),
(2, 'Backpack', 'Accessories', 60),
(3, 'Monitor', 'Electronics', 220),
(6, 'Office Desk', 'Furniture', 250),
(7, 'Laptop', 'Electronics', 1000),
(8, 'Smartphone', 'Electronics', 700),
(9, 'Cap', 'Accessories', 6),
(10, 'Coffee Table', 'Furniture', 180);
```

```
   ┌─id─┬─name─────────┬─category────┬─price─┐
1. │  1 │ Office Chair │ Furniture   │   175 │
2. │  2 │ Backpack     │ Accessories │    60 │
3. │  3 │ Monitor      │ Electronics │   220 │
4. │  6 │ Office Desk  │ Furniture   │   250 │
5. │  7 │ Laptop       │ Electronics │  1000 │
6. │  8 │ Smartphone   │ Electronics │   700 │
7. │  9 │ Cap          │ Accessories │     6 │
8. │ 10 │ Coffee Table │ Furniture   │   180 │
   └────┴──────────────┴─────────────┴───────┘
```

- `sales_2024_may` — sales data for May 2024:

```
CREATE TABLE sales_2024_may (
    sale_id UInt32,
    product_name String,
    sale_date Date DEFAULT toDate(now())
)
ENGINE = MergeTree()
ORDER BY sale_id;
```

```
INSERT INTO sales_2024_may VALUES
(1, 'Smartphone', '2024-05-06'),
(2, 'Smartphone', '2024-05-07'),
(3, 'Monitor', '2024-05-07'),
(4, 'Office Desk', '2024-05-08'),
(5, 'Monitor', '2024-05-09'),
(6, 'Office Chair', '2024-05-09'),
(7, 'Backpack', '2024-05-10'),
(8, 'Backpack', '2024-05-11'),
(9, 'Cap', '2024-05-11'),
(10, 'Cap', '2024-05-12');
```

```
    ┌─sale_id─┬─product_name─┬──sale_date─┐
 1. │       1 │ Smartphone   │ 2024-05-06 │
 2. │       2 │ Smartphone   │ 2024-05-07 │
 3. │       3 │ Monitor      │ 2024-05-07 │
 4. │       4 │ Office Desk  │ 2024-05-08 │
 5. │       5 │ Monitor      │ 2024-05-09 │
 6. │       6 │ Office Chair │ 2024-05-09 │
 7. │       7 │ Backpack     │ 2024-05-10 │
 8. │       8 │ Backpack     │ 2024-05-11 │
 9. │       9 │ Cap          │ 2024-05-11 │
10. │      10 │ Cap          │ 2024-05-12 │
    └─────────┴──────────────┴────────────┘
```

UNION

The `UNION` clause appends the result of the first query to the result of the second query. It can be used in two modes:

- `UNION ALL` — the result of the union contains all rows from the results of both queries, including duplicates;
- `UNION DISTINCT` — duplicate rows are excluded from the result of the union.

If the `ALL` or `DISTINCT` keyword is not explicitly specified, the behavior of `UNION` is determined by the `union_default_mode` setting — it can be set to `ALL`, `DISTINCT`, or an empty string (in this case, using `UNION` throws an exception).

The basic syntax is:

```
SELECT <column1> [, <column2> ]
FROM <table1>
[WHERE <condition>]

UNION [ALL | DISTINCT]

SELECT <column1> [, <column2> ]
FROM <table2>
[WHERE <condition>]
```

The resulting columns are matched by their index (order within `SELECT`). If column names do not match, names for the final result are taken from the first query.

Type casting is performed for unions. For example, if two queries being combined have the same field with compatible non-Nullable and Nullable types, the result of the union will have a Nullable type field.

Queries that are parts of `UNION` can be enclosed in parentheses. `ORDER BY` and `LIMIT` are applied to individual queries, not to the final result. If you need to apply the conversion to the final result, place all queries with `UNION` in a subquery in the `FROM` clause.

Examples

UNION ALL

The following query combines product catalogs for 2023 and 2024:

```
SELECT id, name FROM products_2023
UNION ALL
SELECT id, name FROM products_2024;
```

```
   ┌─id─┬─name─────────┐
1. │  1 │ Office Chair │
2. │  2 │ Backpack     │
3. │  3 │ Monitor      │
4. │  4 │ Tablet       │
5. │  5 │ Water Bottle │
6. │  6 │ Office Desk  │
   └────┴──────────────┘
    ┌─id─┬─name─────────┐
 7. │  1 │ Office Chair │
 8. │  2 │ Backpack     │
 9. │  3 │ Monitor      │
10. │  6 │ Office Desk  │
11. │  7 │ Laptop       │
12. │  8 │ Smartphone   │
13. │  9 │ Cap          │
14. │ 10 │ Coffee Table │
    └────┴──────────────┘
```

UNION DISTINCT

This query combines product catalogs for 2023 and 2024, eliminating duplicates:

```
SELECT id, name FROM products_2023
UNION DISTINCT
SELECT id, name FROM products_2024;
```

```
   ┌─id─┬─name─────────┐
1. │  1 │ Office Chair │
2. │  2 │ Backpack     │
3. │  3 │ Monitor      │
4. │  4 │ Tablet       │
5. │  5 │ Water Bottle │
6. │  6 │ Office Desk  │
   └────┴──────────────┘
    ┌─id─┬─name─────────┐
 7. │  7 │ Laptop       │
 8. │  8 │ Smartphone   │
 9. │  9 │ Cap          │
10. │ 10 │ Coffee Table │
    └────┴──────────────┘
```

INTERSECT

The `INTERSECT` clause returns all rows that appear in both the result sets of the first and second queries. The queries should match in the number of columns, their order, and data types. The `INTERSECT` result may contain duplicate rows. To eliminate duplicates, use `INTERSECT DISTINCT`.

If multiple `INTERSECT` statements are used, and no parentheses are specified, intersections are evaluated from left to right. `INTERSECT` has a higher execution priority than `UNION` and `EXCEPT`.

The basic syntax is:

```
SELECT <column1> [, <column2> ]
FROM <table1>
[WHERE <condition>]

INTERSECT [DISTINCT]

SELECT <column1> [, <column2> ]
FROM <table2>
[WHERE <condition>]
```

Examples

The following query returns the names of products sold in May 2024 that belong to the `Electronics` category, according to the `products_2024` table:

```
SELECT product_name FROM sales_2024_may
INTERSECT
SELECT name FROM products_2024 WHERE category='Electronics';
```

In the result, you can see that there were two sales of the `Smartphone` and `Monitor` products:

```
   ┌─product_name─┐
1. │ Smartphone   │
2. │ Smartphone   │
3. │ Monitor      │
4. │ Monitor      │
   └──────────────┘
```

The `DISTINCT` keyword excludes duplicate rows from the result of the previous query:

```
SELECT product_name FROM sales_2024_may
INTERSECT DISTINCT
SELECT name FROM products_2024 WHERE category='Electronics';
```

```
   ┌─product_name─┐
1. │ Smartphone   │
2. │ Monitor      │
   └──────────────┘
```

EXCEPT

The `EXCEPT` clause returns the result of the first query, excluding rows that are also included in the result of the second query. The queries should match in the number of columns, their order, and data types. The `EXCEPT` result may contain duplicate rows. To exclude duplicates, use `EXCEPT DISTINCT`.

If multiple `EXCEPT` statements are used with no parentheses specified, they are evaluated from left to right. `EXCEPT` has the same execution priority as `UNION` — lower than `INTERSECT`.

The basic syntax is:

```
SELECT <column1> [, <column2> ]
FROM <table1>
[WHERE <condition>]

EXCEPT [DISTINCT]

SELECT <column1> [, <column2> ]
FROM <table2>
[WHERE <condition>]
```

Examples

EXCEPT

The following query returns the names of products sold in May 2024 that are not in the `Electronics` category:

```
SELECT product_name FROM sales_2024_may
EXCEPT
SELECT name FROM products_2024 WHERE category='Electronics';
```

```
   ┌─product_name─┐
1. │ Office Desk  │
2. │ Office Chair │
3. │ Backpack     │
4. │ Backpack     │
5. │ Cap          │
6. │ Cap          │
   └──────────────┘
```

The `DISTINCT` keyword removes duplicates from the result of the previous query:

```
SELECT product_name FROM sales_2024_may
EXCEPT DISTINCT
SELECT name FROM products_2024 WHERE category='Electronics';
```

```
   ┌─product_name─┐
1. │ Office Desk  │
2. │ Office Chair │
3. │ Backpack     │
4. │ Cap          │
   └──────────────┘
```

Common table expressions

Elena Dvoryadkina

Contents

To Table of Contents

- CTE syntax
- Examples
  - Query with CTE
  - Multiple CTEs in a query
  - Nested CTEs
- Recursive CTEs
  - Overview
  - Search order
  - Cycle detection

Common table expression (CTE) is a named dataset that is defined within the scope of an SQL query, exists only during this query execution (CTE is not materialized and is not permanently stored anywhere in a database), and can be accessed multiple times within the main query.

CTE allows you to simplify and make a complex query more readable (you can divide the code into simple parts, reduce its size, avoid duplication), as well as to optimize the execution of a query that involves multiple access to the same datasets (the necessary data is temporarily stored in the cache so that it does not need to be searched for each time).

CTE syntax

To define CTE, use the `WITH` clause before the main (parent) query:

```
WITH <cte_name> AS (<cte_subquery>)
<primary_query>;
```

where:

- `<cte_name>` — identifier by which CTE can be referenced in other parts of the main query;
- `<cte_subquery>` — SQL query whose results populate CTE;
- `<primary_query>` — main SQL query that uses CTE.

CTE syntax for using results of a scalar subquery is slightly different:

```
WITH (<cte_scalar_subquery>) AS <cte_name>
<primary_query>;
```

You can define multiple CTEs within a single `WITH` clause, separating them by commas:

```
WITH
    <cte1>,
    <cte2>,
    <cte3>
<primary_query>;
```

Each CTE can reference any previous one within the `WITH` clause.

Examples

This section provides examples of how CTE can be used in queries to analyze data about bonuses given to employees of a chain of stores in two cities in September. The data is stored in the `employee_bonus_sept_24` table:

```
CREATE TABLE employee_bonus_sept_24 (
    empl_id UInt16,
    full_name String,
    position Enum('store manager' = 1, 'cashier' = 2, 'sales assistant' = 3),
    store_id UInt8,
    city Enum('New York' = 1, 'Chicago' = 2),
    bonus UInt32)
ENGINE = MergeTree
ORDER BY empl_id;
```

```
INSERT INTO employee_bonus_sept_24 VALUES
    (1, 'James Wilson', 1, 101, 1, 1000),
    (2, 'Mary Burton', 2, 101, 1, 450),
    (3, 'Patricia Robinson', 2, 101, 1, 500),
    (4, 'Robert Gray', 1, 102, 1, 1200),
    (5, 'Elizabeth Tucker', 2, 102, 1, 470),
    (6, 'Joseph Lewis', 3, 102, 1, 700),
    (7, 'William Ferguson', 1, 201, 2, 970),
    (8, 'Linda Black', 2, 201, 2, 350),
    (9, 'David Green', 2, 201, 2, 370),
    (10, 'Daniel Gray', 3, 201, 2, 770),
    (11, 'Mark Armstrong', 1, 202, 2, 1200),
    (12, 'Donald Carter', 2, 202, 2, 420),
    (13, 'Elizabeth Collins', 2, 202, 2, 400),
    (14, 'Paul Brown', 3, 202, 2, 660),
    (15, 'Andrew Clarke', 3, 202, 2, 680);
```

```
    ┌─empl_id─┬─full_name─────────┬─position────────┬─store_id─┬─city─────┬─bonus─┐
 1. │       1 │ James Wilson      │ store manager   │      101 │ New York │  1000 │
 2. │       2 │ Mary Burton       │ cashier         │      101 │ New York │   450 │
 3. │       3 │ Patricia Robinson │ cashier         │      101 │ New York │   500 │
 4. │       4 │ Robert Gray       │ store manager   │      102 │ New York │  1200 │
 5. │       5 │ Elizabeth Tucker  │ cashier         │      102 │ New York │   470 │
 6. │       6 │ Joseph Lewis      │ sales assistant │      102 │ New York │   700 │
 7. │       7 │ William Ferguson  │ store manager   │      201 │ Chicago  │   970 │
 8. │       8 │ Linda Black       │ cashier         │      201 │ Chicago  │   350 │
 9. │       9 │ David Green       │ cashier         │      201 │ Chicago  │   370 │
10. │      10 │ Daniel Gray       │ sales assistant │      201 │ Chicago  │   770 │
11. │      11 │ Mark Armstrong    │ store manager   │      202 │ Chicago  │  1200 │
12. │      12 │ Donald Carter     │ cashier         │      202 │ Chicago  │   420 │
13. │      13 │ Elizabeth Collins │ cashier         │      202 │ Chicago  │   400 │
14. │      14 │ Paul Brown        │ sales assistant │      202 │ Chicago  │   660 │
15. │      15 │ Andrew Clarke     │ sales assistant │      202 │ Chicago  │   680 │
    └─────────┴───────────────────┴─────────────────┴──────────┴──────────┴───────┘
```

Query with CTE

In the query below:

- `avg_bonus` is CTE that calculates the average bonus for each position across all stores in a given month;
- only those employees who received a bonus higher than the average are displayed.

```
WITH avg_bonus AS
    (
        SELECT position, round(avg(bonus)) AS position_avg_bonus
        FROM employee_bonus_sept_24
        GROUP BY position
    )
SELECT
    b.full_name AS employee,
    b.position,
    b.store_id AS store,
    b.bonus,
    ab.position_avg_bonus
FROM employee_bonus_sept_24 AS b
INNER JOIN avg_bonus AS ab ON b.position = ab.position
WHERE b.bonus > ab.position_avg_bonus
ORDER BY b.position, b.bonus DESC;
```

```
   ┌─employee──────────┬─position────────┬─store─┬─bonus─┬─position_avg_bonus─┐
1. │ Robert Gray       │ store manager   │   102 │  1200 │               1092 │
2. │ Mark Armstrong    │ store manager   │   202 │  1200 │               1092 │
3. │ Patricia Robinson │ cashier         │   101 │   500 │                423 │
4. │ Elizabeth Tucker  │ cashier         │   102 │   470 │                423 │
5. │ Mary Burton       │ cashier         │   101 │   450 │                423 │
6. │ Daniel Gray       │ sales assistant │   201 │   770 │                702 │
   └───────────────────┴─────────────────┴───────┴───────┴────────────────────┘
```

Window functions in CTE

If you need to filter query data by values of window functions, calculate these values within CTE (or a subquery).

For example, the following query:

- in the `bonus_rank_by_city` CTE, ranks employee bonuses using the `dense_rank()` window function with windows defined by cities;
- in the `WHERE` clause, filters rows by the calculated rank to return the 3 highest bonuses for employees in each city.

```
WITH bonus_rank_by_city AS
    (
        SELECT *, dense_rank() OVER (PARTITION BY city ORDER BY bonus DESC) AS rnk
        FROM employee_bonus_sept_24
        ORDER BY city, rnk
    )
SELECT *
FROM bonus_rank_by_city
WHERE rnk BETWEEN 1 AND 3;
```

```
   ┌─empl_id─┬─full_name────────┬─position────────┬─store_id─┬─city─────┬─bonus─┬─rnk─┐
1. │       4 │ Robert Gray      │ store manager   │      102 │ New York │  1200 │   1 │
2. │       1 │ James Wilson     │ store manager   │      101 │ New York │  1000 │   2 │
3. │       6 │ Joseph Lewis     │ sales assistant │      102 │ New York │   700 │   3 │
4. │      11 │ Mark Armstrong   │ store manager   │      202 │ Chicago  │  1200 │   1 │
5. │       7 │ William Ferguson │ store manager   │      201 │ Chicago  │   970 │   2 │
6. │      10 │ Daniel Gray      │ sales assistant │      201 │ Chicago  │   770 │   3 │
   └─────────┴──────────────────┴─────────────────┴──────────┴──────────┴───────┴─────┘
```

Multiple CTEs in a query

The following query uses two CTEs:

- `avg_bonus_position` — average monthly bonus for a position;
- `avg_bonus_city` — average monthly bonus for a city.

```
WITH
    avg_bonus_position AS
    (
        SELECT position, round(avg(bonus)) AS position_avg_bonus
        FROM employee_bonus_sept_24
        GROUP BY position
    ),
    avg_bonus_city AS
    (
        SELECT city, round(avg(bonus)) AS city_avg_bonus
        FROM employee_bonus_sept_24
        GROUP BY city
    )
SELECT
    b.full_name AS employee,
    b.position AS position,
    b.store_id AS store,
    b.city AS city,
    b.bonus,
    abp.position_avg_bonus,
    abc.city_avg_bonus
FROM employee_bonus_sept_24 AS b
INNER JOIN avg_bonus_position AS abp ON b.position = abp.position
INNER JOIN avg_bonus_city AS abc ON b.city = abc.city
ORDER BY b.position, b.bonus DESC;
```

```
    ┌─employee──────────┬─position────────┬─store─┬─city─────┬─bonus─┬─position_avg_bonus─┬─city_avg_bonus─┐
 1. │ Robert Gray       │ store manager   │   102 │ New York │  1200 │               1092 │            720 │
 2. │ Mark Armstrong    │ store manager   │   202 │ Chicago  │  1200 │               1092 │            647 │
 3. │ James Wilson      │ store manager   │   101 │ New York │  1000 │               1092 │            720 │
 4. │ William Ferguson  │ store manager   │   201 │ Chicago  │   970 │               1092 │            647 │
 5. │ Patricia Robinson │ cashier         │   101 │ New York │   500 │                423 │            720 │
 6. │ Elizabeth Tucker  │ cashier         │   102 │ New York │   470 │                423 │            720 │
 7. │ Mary Burton       │ cashier         │   101 │ New York │   450 │                423 │            720 │
 8. │ Donald Carter     │ cashier         │   202 │ Chicago  │   420 │                423 │            647 │
 9. │ Elizabeth Collins │ cashier         │   202 │ Chicago  │   400 │                423 │            647 │
10. │ David Green       │ cashier         │   201 │ Chicago  │   370 │                423 │            647 │
11. │ Linda Black       │ cashier         │   201 │ Chicago  │   350 │                423 │            647 │
12. │ Daniel Gray       │ sales assistant │   201 │ Chicago  │   770 │                702 │            647 │
13. │ Joseph Lewis      │ sales assistant │   102 │ New York │   700 │                702 │            720 │
14. │ Andrew Clarke     │ sales assistant │   202 │ Chicago  │   680 │                702 │            647 │
15. │ Paul Brown        │ sales assistant │   202 │ Chicago  │   660 │                702 │            647 │
    └───────────────────┴─────────────────┴───────┴──────────┴───────┴────────────────────┴────────────────┘
```

Nested CTEs

The following query allows you to compare the amount of funds allocated for employee bonuses in different stores. It shows what percentage the amount of bonuses paid to employees in each store takes up from the total amount of expenses on bonuses for employees of the entire chain. To do this, the query uses nested CTEs:

- `bonus_all_stores` — total amount of expenses on bonuses for employees of all stores;
- `store_bonus_totals_and_rates` — the amount of bonuses paid to employees in each store and its share relative to the total employee bonus expense in the month, calculated using the output of the `bonus_all_stores` CTE.

```
WITH
    (SELECT sum(bonus) AS all_stores_bonus FROM employee_bonus_sept_24) AS bonus_all_stores,
    store_bonus_totals_and_rates AS
    (
        SELECT
            store_id,
            count() AS employee_count,
            sum(bonus) AS store_bonus_total,
            store_bonus_total / bonus_all_stores AS store_bonus_rate
        FROM employee_bonus_sept_24
        GROUP BY store_id
    )
SELECT
    store_id,
    employee_count,
    store_bonus_total AS total_bonus,
    round(store_bonus_rate * 100, 2) AS bonus_percentage
FROM store_bonus_totals_and_rates;
```

```
   ┌─store_id─┬─employee_count─┬─total_bonus─┬─bonus_percentage─┐
1. │      101 │              3 │        1950 │            19.23 │
2. │      102 │              3 │        2370 │            23.37 │
3. │      201 │              4 │        2460 │            24.26 │
4. │      202 │              5 │        3360 │            33.14 │
   └──────────┴────────────────┴─────────────┴──────────────────┘
```

Recursive CTEs

Overview

ADQM/ClickHouse supports recursive CTEs that you can use to work with hierarchical or tree-structured data. This type of CTE allows a query to be executed repeatedly, each time referring to its own results obtained at the previous iteration.

To create a recursive CTE, use the `RECURSIVE` modifier after `WITH`. The basic syntax of a recursive CTE is:

```
WITH RECURSIVE <cte_name> AS (
    <cte_non_recursive_term>
    UNION ALL
    <cte_recursive_term>
)
SELECT <list_of_columns> FROM <cte_name>;
```

where:

- `<cte_non_recursive_term>` — non-recursive term of the CTE query that is executed first. Received data is written to the result set of the CTE and is also used as input values for the first call of the recursive term.
- `<cte_recursive_term>` — recursive term of the CTE query that refers to the result of its execution at the previous iteration (available under the CTE name — `<cte_name>`) and is executed until it returns any rows. The result of the recursive term, obtained at each iteration, is also written to the CTE's result set that will be available in the main query (under the same name — `<cte_name>`) after the entire CTE query has been processed.

For example, create a table to store information about positions in a company:

```
CREATE TABLE positions (
    id UInt32,
    position String,
    manager_id Nullable(UInt32)
) ENGINE = Memory;
```

Fill in the table with test data:

```
INSERT INTO positions VALUES
    (1, 'CEO', NULL),
    (2, 'CTO', 1),
    (3, 'Backend Team Lead', 2),
    (4, 'Frontend Team Lead', 2),
    (5, 'Backend Developer Senior', 3),
    (6, 'Backend Developer Middle', 3),
    (7, 'Frontend Developer Senior', 4),
    (8, 'Frontend Developer Middle', 4),
    (9, 'Frontend Developer Junior', 4),
    (10, 'UX/UI Designer', 2);
```

Run a query with a recursive CTE to see who employees at different positions in the company's organizational structure report to:

```
WITH RECURSIVE employee_hierarchy AS (
    -- Non-recursive term: start with CEO
    SELECT id, position, manager_id, toUInt16(0) AS level, [position] AS path
    FROM positions
    WHERE manager_id IS NULL

    UNION ALL

    -- Recursive term: add subordinates
    SELECT p.id, p.position, p.manager_id, eh.level + 1, arrayConcat(eh.path, [p.position]) AS path
    FROM positions p
    JOIN employee_hierarchy eh ON p.manager_id = eh.id
)
SELECT position, level, arrayStringConcat(path, ' -> ') AS full_path
FROM employee_hierarchy;
```

This query is executed in the following order:

1. The CTE's non-recursive term returns the initial data from the `positions` table — a position for which no manager is specified (by the `manager_id IS NULL` condition). In this example, it is `CEO` — the top position in the company hierarchy, which is assigned a level of `0`.
2. The output from the non-recursive term is used to evaluate the first iteration of the CTE's recursive term, which finds positions that report to `CEO` by joining the `positions` table and the `employee_hierarchy` CTE with the `positions.manager_id = employee_hierarchy.id` join key. Thus, the first iteration of the recursive term returns the `CTO` position (for which the `manager_id` value is equal to the `CEO` position's identifier) and assigns it a level of `1` in the company hierarchy. The output of this iteration is added to the CTE's result set and serves as input for the next iteration.
3. Two more iterations of the recursive term are performed and return the positions of the next two levels in the company hierarchy — `2` and `3`.
4. The fourth iteration returns an empty result, as the third-level positions in the `positions` table have no subordinates — the CTE execution ends here.
5. The main query calls the CTE to select data from its final result set, which includes the results of the non-recursive term and all iterations of the recursive term.

```
   ┌─position─┬─level─┬─full_path─┐
1. │ CEO      │     0 │ CEO       │
   └──────────┴───────┴───────────┘
   ┌─position─┬─level─┬─full_path──┐
2. │ CTO      │     1 │ CEO -> CTO │
   └──────────┴───────┴────────────┘
   ┌─position───────────┬─level─┬─full_path──────────────────────────┐
3. │ Backend Team Lead  │     2 │ CEO -> CTO -> Backend Team Lead  │
4. │ Frontend Team Lead │     2 │ CEO -> CTO -> Frontend Team Lead │
5. │ UX/UI Designer     │     2 │ CEO -> CTO -> UX/UI Designer     │
   └────────────────────┴───────┴──────────────────────────────────┘
    ┌─position──────────────────┬─level─┬─full_path─────────────────────────────────────────────────────┐
 6. │ Backend Developer Senior  │     3 │ CEO -> CTO -> Backend Team Lead -> Backend Developer Senior   │
 7. │ Backend Developer Middle  │     3 │ CEO -> CTO -> Backend Team Lead -> Backend Developer Middle   │
 8. │ Frontend Developer Senior │     3 │ CEO -> CTO -> Frontend Team Lead -> Frontend Developer Senior │
 9. │ Frontend Developer Middle │     3 │ CEO -> CTO -> Frontend Team Lead -> Frontend Developer Middle │
10. │ Frontend Developer Junior │     3 │ CEO -> CTO -> Frontend Team Lead -> Frontend Developer Junior │
    └───────────────────────────┴───────┴───────────────────────────────────────────────────────────────┘
```

Search order

The above query outputs data in the order it was added to the recursive CTE's result set — sorted in the breadth-first order by the nesting level of the recursive query, which corresponds to the level of a position in the company hierarchy (`level`) in this case.

To sort the results in the depth-first order, apply sorting by the `path` field that stores the path to each row as an array of already-visited rows (in this example, this is a sequence of all higher positions in the company hierarchy for each position).

```
WITH RECURSIVE employee_hierarchy AS (
    -- Non-recursive term: start with CEO
    SELECT id, position, manager_id, toUInt16(0) AS level, [position] AS path
    FROM positions
    WHERE manager_id IS NULL

    UNION ALL

    -- Recursive term: add subordinates
    SELECT p.id, p.position, p.manager_id, eh.level + 1, arrayConcat(eh.path, [p.position]) AS path
    FROM positions p
    JOIN employee_hierarchy eh ON p.manager_id = eh.id
)
SELECT repeat('    ', level) || position AS position, level, arrayStringConcat(path, ' -> ') AS full_path
FROM employee_hierarchy
ORDER BY path;
```

```
    ┌─position──────────────────────────────┬─level─┬─full_path─────────────────────────────────────────────────────┐
 1. │ CEO                                   │     0 │ CEO                                                           │
 2. │     CTO                               │     1 │ CEO -> CTO                                                    │
 3. │         Backend Team Lead             │     2 │ CEO -> CTO -> Backend Team Lead                               │
 4. │             Backend Developer Middle  │     3 │ CEO -> CTO -> Backend Team Lead -> Backend Developer Middle   │
 5. │             Backend Developer Senior  │     3 │ CEO -> CTO -> Backend Team Lead -> Backend Developer Senior   │
 6. │         Frontend Team Lead            │     2 │ CEO -> CTO -> Frontend Team Lead                              │
 7. │             Frontend Developer Junior │     3 │ CEO -> CTO -> Frontend Team Lead -> Frontend Developer Junior │
 8. │             Frontend Developer Middle │     3 │ CEO -> CTO -> Frontend Team Lead -> Frontend Developer Middle │
 9. │             Frontend Developer Senior │     3 │ CEO -> CTO -> Frontend Team Lead -> Frontend Developer Senior │
10. │         UX/UI Designer                │     2 │ CEO -> CTO -> UX/UI Designer                                  │
    └───────────────────────────────────────┴───────┴───────────────────────────────────────────────────────────────┘
```

Cycle detection

When working with a recursive CTE, it is important to check whether the recursive part of the query does not lead to an infinite loop and, if necessary, set conditions to stop recursion. This section provides examples of various mechanisms to prevent loops in queries:

- Track already-processed values
- Limit recursion depth
- Use LIMIT
- Limit by value

Track already-processed values

The standard way to detect cycles is to track one or more fields and check whether the current value has been processed before (in other words, whether the current point has already been reached by executing a previous sequence of recursions).

The following example shows how to traverse a graph with a cycle, avoiding infinite recursion.

Create the `graph` table:

```
CREATE TABLE graph (
    from_node UInt32,
    to_node UInt32
) ENGINE = MergeTree()
ORDER BY (from_node, to_node);
```

Fill the table with test data simulating a graph with some cycle (for example, `1 → 2 → 4 → 7 → 1`):

```
INSERT INTO graph VALUES
    (1, 2), (1, 3),
    (2, 4), (2, 5),
    (3, 6),
    (4, 7), (4, 8),
    (5, 9),
    (7, 1),
    (6, 10);
```

A graph traversal query without cycle detection will loop indefinitely and fail with the exception `Maximum recursive CTE evaluation depth (1000) exceeded`:

```
WITH RECURSIVE graph_traversal AS (
    SELECT from_node, to_node,
    FROM graph
    WHERE from_node = 1 -- Start with the node 1

    UNION ALL

    SELECT g.from_node, g.to_node,
    FROM graph_traversal AS gt
    JOIN graph AS g ON gt.to_node = g.from_node
)
SELECT from_node, to_node,
FROM graph_traversal;
```

To avoid looping the query, save the traversed paths to the graph nodes as arrays of already-visited nodes in the `path` field and use them to track the re-traversal of nodes, writing the cycle detection flag in the `has_cycle` field:

```
WITH RECURSIVE graph_traversal AS (
    SELECT from_node, to_node, 1 AS depth, [from_node, to_node] AS path, false AS has_cycle
    FROM graph
    WHERE from_node = 1 -- Start with the node 1

    UNION ALL

    SELECT g.from_node, g.to_node, gt.depth + 1, arrayPushBack(gt.path, g.to_node), has(gt.path, g.to_node) AS has_cycle
    FROM graph_traversal AS gt
    JOIN graph AS g ON gt.to_node = g.from_node
    WHERE NOT gt.has_cycle -- Stop recursion when a cycle is detected
)
SELECT from_node, to_node, depth, path, has_cycle, arrayStringConcat(path, ' -> ') AS path_str
FROM graph_traversal;
```

Query result

```
   ┌─from_node─┬─to_node─┬─depth─┬─path──┬─has_cycle─┬─path_str─┐
1. │         1 │       2 │     1 │ [1,2] │ false     │ 1 -> 2   │
2. │         1 │       3 │     1 │ [1,3] │ false     │ 1 -> 3   │
   └───────────┴─────────┴───────┴───────┴───────────┴──────────┘
   ┌─from_node─┬─to_node─┬─depth─┬─path────┬─has_cycle─┬─path_str────┐
3. │         2 │       4 │     2 │ [1,2,4] │ false     │ 1 -> 2 -> 4 │
4. │         2 │       5 │     2 │ [1,2,5] │ false     │ 1 -> 2 -> 5 │
5. │         3 │       6 │     2 │ [1,3,6] │ false     │ 1 -> 3 -> 6 │
   └───────────┴─────────┴───────┴─────────┴───────────┴─────────────┘
   ┌─from_node─┬─to_node─┬─depth─┬─path───────┬─has_cycle─┬─path_str──────────┐
6. │         4 │       7 │     3 │ [1,2,4,7]  │ false     │ 1 -> 2 -> 4 -> 7  │
7. │         4 │       8 │     3 │ [1,2,4,8]  │ false     │ 1 -> 2 -> 4 -> 8  │
8. │         5 │       9 │     3 │ [1,2,5,9]  │ false     │ 1 -> 2 -> 5 -> 9  │
9. │         6 │      10 │     3 │ [1,3,6,10] │ false     │ 1 -> 3 -> 6 -> 10 │
   └───────────┴─────────┴───────┴────────────┴───────────┴───────────────────┘
    ┌─from_node─┬─to_node─┬─depth─┬─path────────┬─has_cycle─┬─path_str──────────────┐
10. │         7 │       1 │     4 │ [1,2,4,7,1] │ true      │ 1 -> 2 -> 4 -> 7 -> 1 │
    └───────────┴─────────┴───────┴─────────────┴───────────┴───────────────────────┘
```

Limit recursion depth

When executing a recursive CTE, you can store the nesting level of the recursions in a separate column and use it to limit the depth of the search. In the example below, this is the `depth` column:

```
WITH RECURSIVE graph_traversal AS (
    SELECT from_node, to_node, 1 AS depth, [from_node, to_node] AS path
    FROM graph
    WHERE from_node = 1 -- Start with the node 1

    UNION ALL

    SELECT g.from_node, g.to_node, gt.depth + 1, arrayPushBack(gt.path, g.to_node)
    FROM graph_traversal AS gt
    JOIN graph AS g ON gt.to_node = g.from_node
    WHERE gt.depth < 3 -- Limiting the depth of recursion
)
SELECT from_node, to_node, depth, path, arrayStringConcat(path, ' -> ') AS path_str
FROM graph_traversal;
```

You can also configure the maximum depth of recursive CTEs in ADQM at the session level using the `max_recursive_cte_evaluation_depth` parameter (the default value is `1000`), for example:

```
SET max_recursive_cte_evaluation_depth = 3;
```

Use LIMIT

Another way to stop the infinite recursion of a CTE is to add `LIMIT` to the main query. For example, the following query calculates the factorial of numbers from `1` to `10`:

```
WITH RECURSIVE r AS (
    SELECT 1 AS i, toUInt32(1) AS factorial
    UNION ALL
    SELECT i+1 AS i, factorial*i as factorial
    FROM r
)
SELECT * FROM r LIMIT 10;
```

Query result

```
   ┌─i─┬─factorial─┐
1. │ 1 │         1 │
   └───┴───────────┘
   ┌─i─┬─factorial─┐
2. │ 2 │         2 │
   └───┴───────────┘
   ┌─i─┬─factorial─┐
3. │ 3 │         6 │
   └───┴───────────┘
   ┌─i─┬─factorial─┐
4. │ 4 │        24 │
   └───┴───────────┘
   ┌─i─┬─factorial─┐
5. │ 5 │       120 │
   └───┴───────────┘
   ┌─i─┬─factorial─┐
6. │ 6 │       720 │
   └───┴───────────┘
   ┌─i─┬─factorial─┐
7. │ 7 │      5040 │
   └───┴───────────┘
   ┌─i─┬─factorial─┐
8. │ 8 │     40320 │
   └───┴───────────┘
   ┌─i─┬─factorial─┐
9. │ 9 │    362880 │
   └───┴───────────┘
    ┌─i──┬─factorial─┐
10. │ 10 │   3628800 │
    └────┴───────────┘
```

NOTE

In this example, the `1` initial value in the `factorial` field is cast to the UInt32 type to avoid data type overflow during the factorial calculation, since the default type is UInt8.

Limit by value

You can also use a value of some field as a condition for stopping recursion. For example:

```
WITH RECURSIVE r AS (
    SELECT 1 AS i, toUInt32(1) AS factorial
    UNION ALL
    SELECT i+1 AS i, factorial*i as factorial
    FROM r
    WHERE factorial < 5000
)
SELECT * FROM r;
```

Query result

```
   ┌─i─┬─factorial─┐
1. │ 1 │         1 │
   └───┴───────────┘
   ┌─i─┬─factorial─┐
2. │ 2 │         2 │
   └───┴───────────┘
   ┌─i─┬─factorial─┐
3. │ 3 │         6 │
   └───┴───────────┘
   ┌─i─┬─factorial─┐
4. │ 4 │        24 │
   └───┴───────────┘
   ┌─i─┬─factorial─┐
5. │ 5 │       120 │
   └───┴───────────┘
   ┌─i─┬─factorial─┐
6. │ 6 │       720 │
   └───┴───────────┘
```

Aggregate functions

Elena Dvoryadkina

Contents

To Table of Contents

- Overview
- Combinators
  - Add conditions to aggregations
  - Aggregate on unique values
  - Manage aggregate values for empty inputs
  - Aggregate arrays
  - Aggregate values of the Map type
  - Work with aggregate states
  - Split data into groups before aggregation
- Process NULL

Overview

An **aggregate function** computes a single result from a set of input values. For example, you can calculate the sum, average, maximum, or minimum over a set of rows.

To obtain the full list of predefined aggregate functions with their descriptions, refer to the List of Aggregate Functions section of the ClickHouse documentation.

Create a table to run test queries

To reproduce examples provided below in this article and explore behavior of various aggregate functions, first create the `users` table:

```
CREATE TABLE users (user_id Int32, name String, age Int32, role String) ENGINE = MergeTree ORDER BY user_id;
```

```
INSERT INTO users VALUES
(1, 'john', 37, 'developer'),
(2, 'alex', 25, 'analyst'),
(3, 'mary', 30, 'manager'),
(4, 'david', 41, 'developer'),
(5, 'andrew', 41, 'analyst'),
(6, 'harry', 29, 'developer'),
(7, 'ann', 35, 'QA engineer'),
(8, 'jack', 27, 'designer');
```

```
┌─user_id─┬─name───┬─age─┬─role────────┐
│       1 │ john   │  37 │ developer   │
│       2 │ alex   │  25 │ analyst     │
│       3 │ mary   │  30 │ manager     │
│       4 │ david  │  41 │ developer   │
│       5 │ andrew │  41 │ analyst     │
│       6 │ harry  │  29 │ developer   │
│       7 │ ann    │  35 │ QA engineer │
│       8 │ jack   │  27 │ designer    │
└─────────┴────────┴─────┴─────────────┘
```

For example, the following aggregate function returns the age of the youngest user from the `users` table:

```
SELECT min(age) FROM users;
```

The output:

```
┌─min(age)─┐
│       25 │
└──────────┘
```

To utilize an aggregate function in the `WHERE` clause, use a subquery. The following example returns the name of the youngest user:

```
SELECT name FROM users WHERE age = (SELECT min(age) FROM users);
```

The output:

```
┌─name─┐
│ alex │
└──────┘
```

You can also combine an aggregate function with the `GROUP BY` clause. For example, the following query returns the number of users for each role:

```
SELECT role, count() AS number_of_users FROM users GROUP BY role;
```

The output:

```
┌─role────────┬─number_of_users─┐
│ manager     │               1 │
│ designer    │               1 │
│ analyst     │               2 │
│ developer   │               3 │
│ QA engineer │               1 │
└─────────────┴─────────────────┘
```

To filter the result set, use the `HAVING` clause. For example, the following query returns roles that are assigned to more than one user:

```
SELECT role, count() AS number_of_users FROM users GROUP BY role HAVING count()>1;
```

The output:

```
┌─role──────┬─number_of_users─┐
│ analyst   │               2 │
│ developer │               3 │
└───────────┴─────────────────┘
```

NOTE

There is a fundamental difference between the `WHERE` and `HAVING` clauses. `WHERE` selects input rows before evaluating groups and aggregate functions — it determines which rows are included in aggregate calculations. `HAVING` selects rows after groups and aggregate functions are calculated. The `WHERE` clause cannot contain aggregate functions, and the `HAVING` clause always contains aggregate functions. If you need to use additional conditions, add them to the `WHERE` clause, not to `HAVING`. This approach avoids aggregation calculations for rows that do not meet the additional conditions.

Combinators

To extend the capabilities of aggregate functions and address a wide range of tasks without the need to change data structures, ADQM supports various combinators. A **combinator** is a special suffix that is added to an aggregate function name and modifies that function's behavior. You can use multiple combinators for one function simultaneously.

Add conditions to aggregations

An aggregate function with the `If` combinator accepts an additional argument — a condition of the UInt8 type that determines which rows the aggregate function should process.

Aggregate expression syntax is `<aggr_func_name>If(<expr>, <if_condition>)`.

Example

The following query returns the number of users older than 30:

```
SELECT countIf(age>30) AS over_30 FROM users;
```

The output:

```
┌─over_30─┐
│       4 │
└─────────┘
```

Aggregate on unique values

An aggregate function with the `Distinct` combinator is invoked once for each unique value of an argument (or unique set of values, for multiple arguments).

Aggregate expression syntax is `<aggr_func_name>Distinct(<expr>)` or `<aggr_func_name>(DISTINCT <expr>)`.

Example

The following query returns the number of unique roles in the `users` table:

```
SELECT count(DISTINCT role) AS unique_roles FROM users;
```

The output:

```
┌─unique_roles─┐
│            5 │
└──────────────┘
```

Manage aggregate values for empty inputs

You can use one of the following combinators to define a value that an aggregate function should return in the case when it does not have input data:

- `OrDefault` — returns a default value of the aggregate function's return type;
- `OrNull` — converts the aggregate function's return type to Nullable, and returns NULL if there is nothing to aggregate.

Example:

```
SELECT avg(number), avgOrDefault(number), avgOrNull(number) FROM numbers(0);
```

The output:

```
┌─avg(number)─┬─avgOrDefault(number)─┬─avgOrNull(number)─┐
│         nan │                    0 │              NULL │
└─────────────┴──────────────────────┴───────────────────┘
```

Aggregate arrays

Array

Aggregate functions with the `Array` combinator accept arguments of the Array(T) type (arrays) instead of T-type arguments. When processing arrays, an aggregate function works like the original aggregate function over all array elements.

Aggregate expression syntax is `<aggr_func_name>Array(<arr>)`, where `<arr>` is a set of arrays (for example, a column that contains arrays). If a function accepts multiple arguments, array lengths should be equal.

Example

1. Create a table with one of the columns containing arrays:

```
CREATE TABLE test_arrays (id Int32, array Array(UInt32)) ENGINE = MergeTree ORDER BY id;
```

```
INSERT INTO test_arrays VALUES (1, [1, 2, 3]), (2, [2,3,7]), (3, [1,1,1]);
```

```
┌─id─┬─array───┐
│  1 │ [1,2,3] │
│  2 │ [2,3,7] │
│  3 │ [1,1,1] │
└────┴─────────┘
```

2. Use the `sumArray` function to calculate the sum of elements of all arrays in a column of the Array(UInt32) type:

```
SELECT sumArray(array) AS sum FROM test_arrays;
```

```
┌─sum─┐
│  21 │
└─────┘
```

ForEach

The `ForEach` combination converts an aggregate function for tables into an aggregate function for arrays that aggregates corresponding values from multiple arrays and returns an array of results.

Aggregate expression syntax is `<aggr_func_name>ForEach(<arr>)`, where `<arr>` is a set of arrays (for example, a table column with arrays).

Example

Use the `maxForEach` function to get maximum values among the corresponding elements of arrays:

```
SELECT maxForEach(array) FROM test_arrays;
```

The aggregate function finds the maximum value among the first elements of arrays and writes it to the first position of the resulting array. Then it repeats the same for the second and third elements of arrays:

```
┌─maxForEach(array)─┐
│ [2,3,7]           │
└───────────────────┘
```

Aggregate values of the Map type

To aggregate values of the Map type, use the `Map` combinator for aggregate functions. An aggregate function with this combinator accepts a Map type as an argument, aggregates values of each key separately using the appropriate aggregate function, and returns the result of the Map type.

Example

1. Create a table with one of the columns containing values of the Map type:

```
CREATE TABLE test_map_table (a Int32, b Map(String, UInt64)) ENGINE = MergeTree ORDER BY a;
```

```
INSERT INTO test_map_table VALUES (1, {'key1':1, 'key2':10}), (2, {'key1':2,'key2':20}), (3, {'key1':3,'key2':30});
```

```
┌─a─┬─b─────────────────────┐
│ 1 │ {'key1':1,'key2':10} │
│ 2 │ {'key1':2,'key2':20} │
│ 3 │ {'key1':3,'key2':30} │
└───┴───────────────────────┘
```

2. Use the `Map` combinator for the `sum` aggregate function to calculate the sum of values by the `key2` key:

```
select sumMap(b)['key2'] from test_map_table;
```

The output:

```
┌─arrayElement(sumMap(b), 'key2')─┐
│                              60 │
└─────────────────────────────────┘
```

Work with aggregate states

ADQM allows working with intermediate aggregate states instead of resulting values which take up more space. Use the `State` combinator for aggregate functions to store intermediate states of aggregations as values of the AggregateFunction type. To get the final result of aggregation in the future, pass intermediate states as arguments to the same aggregate functions with combinators:

- `Merge` — combines states to finish aggregation, and returns the resulting value;
- `MergeState` — merges intermediate aggregation states, similar to the `Merge` combinator, but returns an intermediate aggregation state (not the resulting value), similar to the `State` combinator.

For some aggregate functions (for example, `sum`, `max`, `min`, `any`), you can use the `SimpleState` combinator. An aggregate function with this combinator returns the same value as the original function, but of the SimpleAggregateFunction type. `SimpleAggregateFunction` has better performance than `AggregateFunction` for the same aggregate function.

To store aggregation states and automatically merge them by primary key, you can use the AggregatingMergeTree table engine.

Split data into groups before aggregation

The `Resample` combinator allows you to divide data into groups (for example, split column values into intervals) and perform aggregation for each group separately.

Aggregate expression syntax is `<aggr_func_name>Resample(<start>, <end>, <step>)(<aggr_func_params>, <resample_key>)`, where:

- `<resample_key>` — column which values are used to split data into intervals;
- `<start>` — start value of the whole interval for `<resample_key>` values;
- `<stop>` — end value of the whole interval for `<resample_key>` values (the interval does not include the end value — `[<start>, <stop>)`);
- `<step>` — step for dividing the whole interval into subintervals (the `aggr_func_name` function is executed over each of subintervals independently);
- `<aggr_func_params>` — aggregate function parameters.

An aggregate function with the `Resample` combinator returns an array of the `aggr_func_name` function results for each data group.

Example

The source `users` table:

```
┌─name───┬─age─┐
│ john   │  37 │
│ alex   │  25 │
│ mary   │  30 │
│ david  │  41 │
│ andrew │  41 │
│ harry  │  29 │
│ ann    │  35 │
│ jack   │  27 │
└────────┴─────┘
```

The following query returns names of users whose age is in the `[20,35)` and `[35,50)` intervals:

```
SELECT groupArrayResample(20, 50, 15)(name, age) from users;
```

where:

- `name` — column with user names (an argument of the groupArray aggregate function);
- `age` — column to aggregate user names by age.

The output:

```
┌─groupArrayResample(20, 50, 15)(name, age)─────────────────┐
│ [['alex','mary','harry','jack'],['john','david','andrew','ann']] │
└────────────────────────────────────────────────────────────┘
```

Process NULL

Most aggregate functions ignore NULL input values. The exception is the `count()` function that returns the total number of rows in the selection.

Example

Create a table with a column of the Nullable type:

```
CREATE TABLE table_nulls (x Int8, y Nullable(Int8)) ENGINE = MergeTree ORDER BY x;
```

```
INSERT INTO table_nulls VALUES (1, NULL), (2, 1), (3, 2), (4, 2), (5, 3);
```

```
┌─x─┬────y─┐
│ 1 │ NULL │
│ 2 │    1 │
│ 3 │    2 │
│ 4 │    2 │
│ 5 │    3 │
└───┴──────┘
```

The `groupArray(y)` function returns an array of the `y` column's values, ignoring NULL:

```
SELECT groupArray(y) FROM table_nulls;
```

```
┌─groupArray(y)─┐
│ [1,2,2,3]     │
└───────────────┘
```

The `count()` function returns the total number of rows in the table:

```
SELECT count() FROM table_nulls;
```

```
┌─count()─┐
│       5 │
└─────────┘
```

The `count(y)` function returns the number of rows where `y` values are not NULL:

```
SELECT count(y) FROM table_nulls;
```

```
┌─count(y)─┐
│        4 │
└──────────┘
```

The `count(DISTINCT y)` function returns the number of unique rows with non-NULL `y` values:

```
SELECT count(DISTINCT y) FROM table_nulls;
```

```
┌─uniqExact(y)─┐
│            3 │
└──────────────┘
```

You can use the `coalesce` function to replace NULL values with some suitable value that will be taken into account when evaluating an aggregate function. For example, `avg(coalesce(y, 0))` uses `0` instead of NULL values in the `y` column when calculating an average value for the column:

```
SELECT avg(y), avg(coalesce(y, 0)) FROM table_nulls;
```

```
┌─avg(y)─┬─avg(coalesce(y, 0))─┐
│      2 │                 1.6 │
└────────┴─────────────────────┘
```

To work around skipping NULL values, you can also use the `tuple` function, for example:

```
SELECT groupArray(y), groupArray(tuple(y)).1 FROM table_nulls;
```

```
┌─groupArray(y)─┬─tupleElement(groupArray(tuple(y)), 1)─┐
│ [1,2,2,3]     │ [NULL,1,2,2,3]                        │
└───────────────┴───────────────────────────────────────┘
```

Use flameGraph

Elena Dvoryadkina

Contents

To Table of Contents

The flameGraph function syntax
Build a flame graph
Example

For in-depth performance analysis and query debugging, ADQM/ClickHouse provides the system.trace_log system table, which stores stack traces collected by the query profiler. For example, it can help you understand why a query is running slowly or not as expected. By examining the sequence of operations in the trace, you can detect specific query functions that cause problems (requiring the most time or resources to execute). This information can be useful for optimizing your query and improving its performance.

To visually represent stack traces as a flame graph, you can use the `flameGraph` aggregate function. This function returns an array of strings based on data from the `system.trace_log` table, which you can then pass to the flamegraph.pl utility (or a similar tool) to create a flame graph.

The flameGraph function syntax

The basic syntax of the `flameGraph` function:

```
flameGraph(<trace>, [<size>], [<ptr>])
```

where:

- `<trace>` — stack trace;
- `<size>` — allocation size for memory profiling (the `Memory`, `MemorySample`, or `MemoryPeak` trace type), the default value is `1`;
- `<ptr>` — allocation address, the default value is `0`.

Build a flame graph

To get a query profile as a flame graph based on information from the `system.trace_log` table, follow the instructions below.

First, set up the ADQM query profiler and prepare tools for rendering flame graphs on your ADQM hosts:

1. Enable the query profiler and configure it using the `query_profiler_*` and `memory_profiler_*` parameters (see Session Settings in the ClickHouse documentation). You can set these parameters at the ADQM user level, the clickhouse-client session level, or for an individual query to be profiled (for details, refer to the Configure user settings article). For example, it is recommended to set `query_profiler_real_time_period_ns` and/or `query_profiler_cpu_time_period_ns` to `10000000` to trace C++ functions that are running.

2. Install the `adqm-clickhouse-debuginfo` package. Make sure the `allow_introspection_functions` setting is enabled for a user who will connect to ADQM and use the `flameGraph` function. You can also enable it at the `clickhouse-client` session level:

    - in the batch mode:

    ```
    $ clickhouse-client --allow_introspection_functions=1 -q "..." ...
    ```

    - in the interactive mode:

    ```
    SET allow_introspection_functions = 1;
    ```

3. Download the flamegraph.pl script to ADQM hosts and grant it execution permission (with the `chmod +x` command). Check also that Perl is installed on the hosts.

Now, you can use the `flameGraph` function and the `flamegraph.pl` utility to generate a flame graph for a query to profile it. For example, the following command creates a flame graph (an interactive SVG file named `<flame_cpu>`) visualizing stack traces by CPU time for the query with the `<query_id>` identifier (`<path_to>` in this example is a path to the `flamegraph.pl` script on the ADQM host):

```
$ clickhouse-client \
    -q "SELECT arrayJoin(flameGraph(arrayReverse(trace))) \
        FROM system.trace_log \
        WHERE trace_type = 'CPU' AND query_id = '<query_id>'" \
    | <path_to>/flamegraph.pl  > <flame_cpu>.svg
```

Example

This example shows how you can use a flame graph to identify which functions are executed and in what order during the execution of a query.

1. Create a table without a primary key that has two columns — the first column is compressed with the LZ4 algorithm, and the second one stores data without compression:

    ```
    CREATE TABLE test_table (a Int64 CODEC(LZ4), b Int64 CODEC(NONE)) ENGINE = MergeTree ORDER BY tuple();
    ```

2. Insert 100 million rows with test values into the table:

    ```
    INSERT INTO test_table SELECT 1, 1 FROM numbers(100000000);
    ```

    Using the following query, you can see the difference between columns in the size of stored data:

    ```
    SELECT
        name,
        formatReadableSize(data_uncompressed_bytes) AS uncompressed_size,
        formatReadableSize(data_compressed_bytes) AS compressed_size,
        round(data_uncompressed_bytes / data_compressed_bytes, 2) AS ratio
    FROM system.columns
    WHERE table = 'test_table';
    ```

    ```
       ┌─name─┬─uncompressed_size─┬─compressed_size─┬─ratio─┐
    1. │ a    │ 762.94 MiB        │ 3.45 MiB        │ 221.4 │
    2. │ b    │ 762.94 MiB        │ 763.23 MiB      │     1 │
       └──────┴───────────────────┴─────────────────┴───────┘
    ```

3. Run two separate queries to calculate the sum of the values in each column:

    ```
    SELECT sum(a) FROM test_table;
    ```

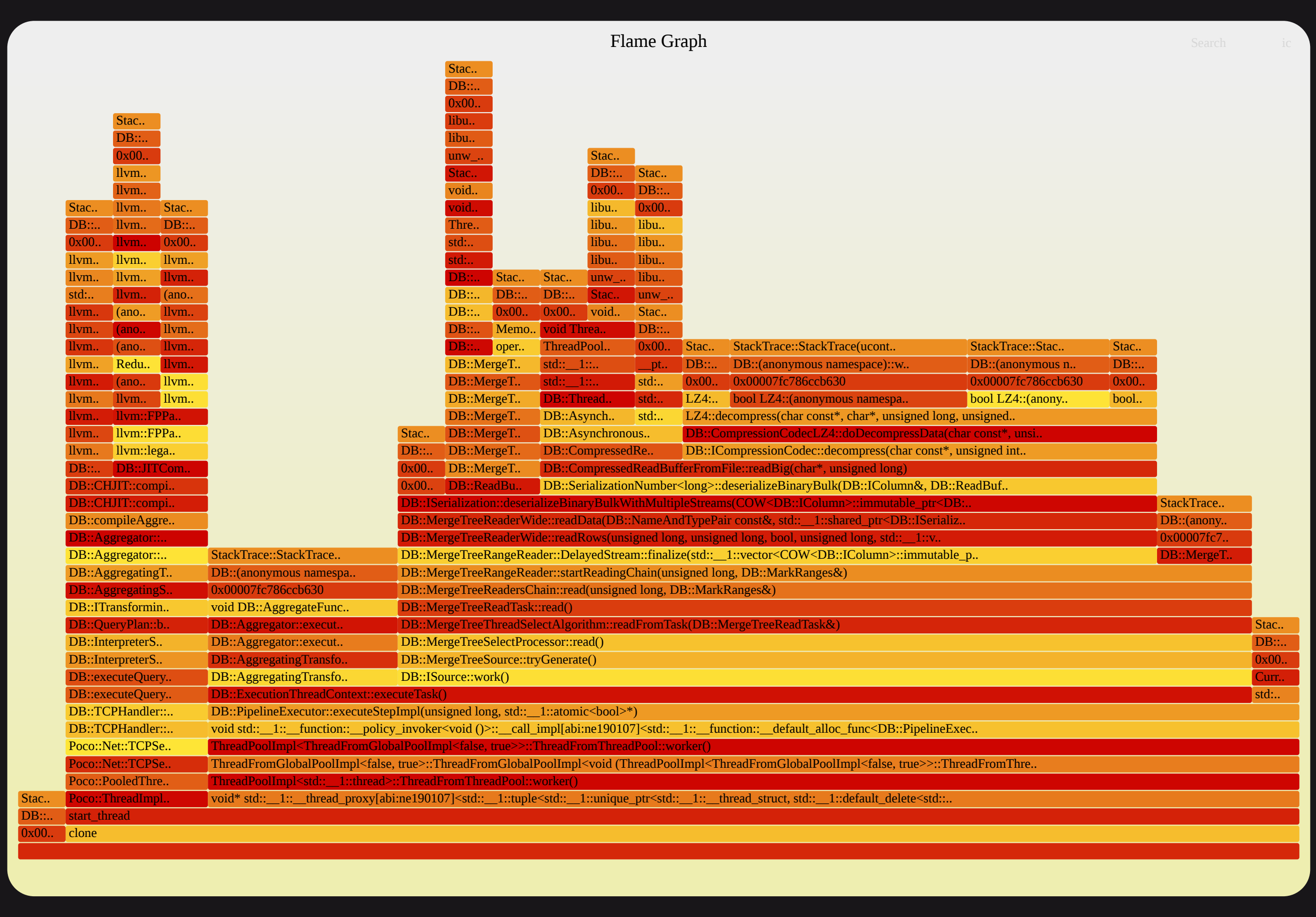
    ```
    Query id: adde6caf-60c5-4820-8bb3-5f75a8edb98e

       ┌────sum(a)─┐
    1. │ 100000000 │ -- 100.00 million
       └───────────┘

    1 row in set. Elapsed: 0.072 sec. Processed 100.00 million rows, 800.00 MB (1.39 billion rows/s., 11.14 GB/s.)
    Peak memory usage: 7.92 MiB.
    ```

    ```
    SELECT sum(b) FROM test_table;
    ```

    ```
    Query id: 8491a039-be0d-4a17-9130-ec286dfee039

       ┌────sum(b)─┐
    1. │ 100000000 │ -- 100.00 million
       └───────────┘

    1 row in set. Elapsed: 0.363 sec. Processed 100.00 million rows, 800.00 MB (275.73 million rows/s., 2.21 GB/s.)
    Peak memory usage: 14.51 MiB.
    ```

4. For the first query, build a flame graph based on a CPU query profiler:

    ```
    $ clickhouse-client \
        -q "SELECT arrayJoin(flameGraph(arrayReverse(trace))) \
            FROM system.trace_log \
            WHERE trace_type = 'CPU' AND query_id = 'adde6caf-60c5-4820-8bb3-5f75a8edb98e'" \
        | flamegraph.pl  > flamegraph_cpu_1.svg
    ```



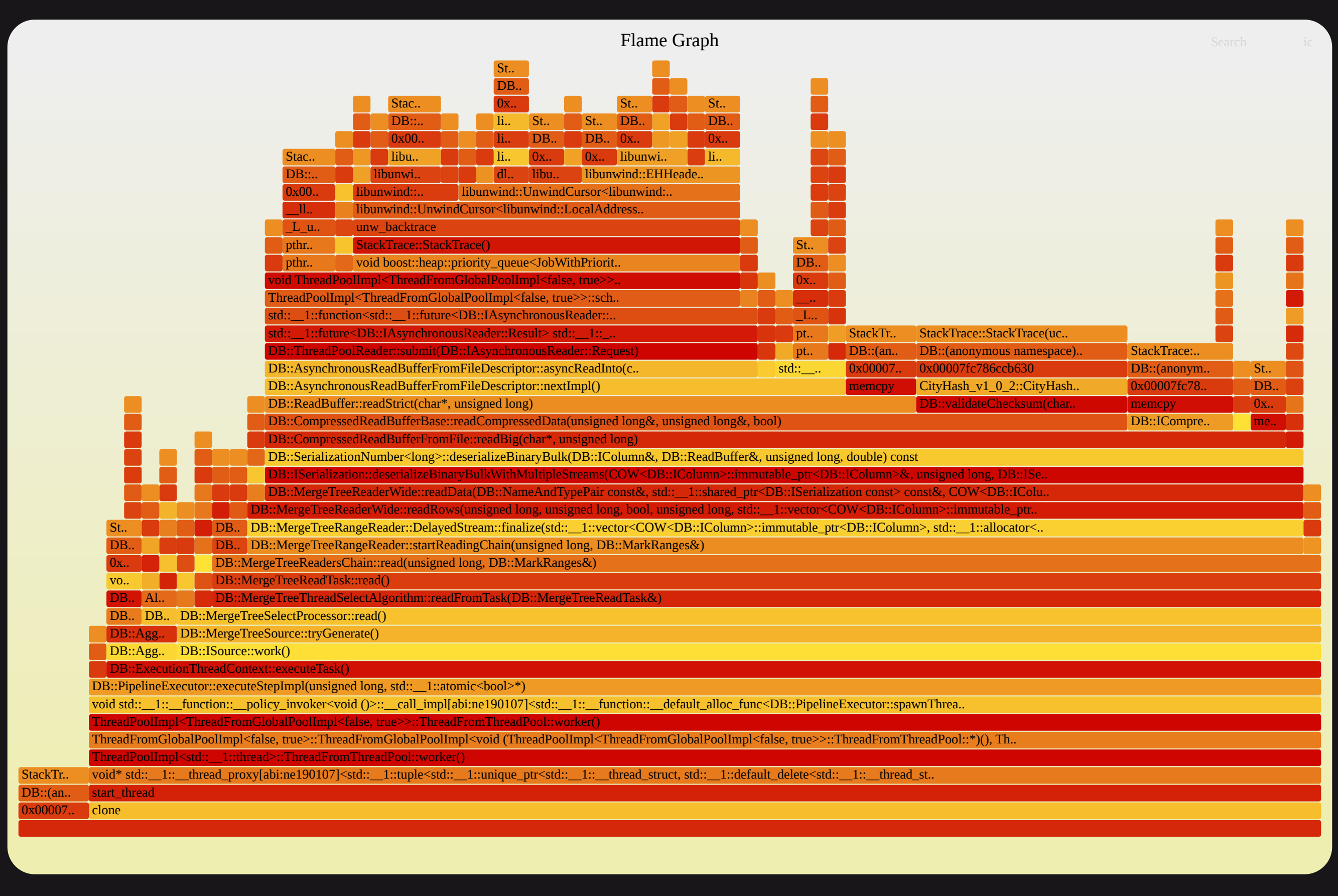
CPU flame graph for the query calculating the sum of the values in a column for which a data compression codec is used

    The flame graph shows the full stack of C++ functions that were called. For example, note that decompression was performed during data reading (the `decompress` functions were called).

5. Create a flame graph for the second query:

    ```
    $ clickhouse-client \
        -q "SELECT arrayJoin(flameGraph(arrayReverse(trace))) \
            FROM system.trace_log \
            WHERE trace_type = 'CPU' AND query_id = '8491a039-be0d-4a17-9130-ec286dfee039'" \
        | flamegraph.pl  > flamegraph_cpu_2.svg
    ```

    The graph shows that the `CompressedReadBufferBase` class is also used in this case, but it does not call the decompression functions (since data is read from a column where compression is not applied).



CPU flame graph for the query that reads data from the column without compression

Window functions

Elena Dvoryadkina

Contents

To Table of Contents

Syntax overview
  - Window specification
  - ROWS
  - RANGE

Function types

A **window function** performs a calculation across a set of rows: all rows in a query are divided into groups (windows), and each group has its own aggregates. It does not return a single output row as an aggregate function. A window function adds an aggregated value to each row of the selection result in a separate column.

Create a table to run test queries

To follow the examples in this article and see how window functions work, first create the `wnd_func_table` table:

```
CREATE TABLE wnd_func_table (group_id Int32, sort_id Int32, value Int32) ENGINE = MergeTree ORDER BY group_id;
```

```
INSERT INTO wnd_func_table FORMAT Values
(1,1,10), (1,2,20), (1,3,30), (1,4,40), (1,5,50),
(2,1,1), (2,2,2), (2,3,3), (2,4,4), (2,4,5), (2,4,6), (2,5,7), (2,6,8);
```

```
┌─group_id─┬─sort_id─┬─value─┐
│        1 │       1 │    10 │
│        1 │       2 │    20 │
│        1 │       3 │    30 │
│        1 │       4 │    40 │
│        1 │       5 │    50 │
│        2 │       1 │     1 │
│        2 │       2 │     2 │
│        2 │       3 │     3 │
│        2 │       4 │     4 │
│        2 │       4 │     5 │
│        2 │       4 │     6 │
│        2 │       5 │     7 │
│        2 │       6 │     8 │
└──────────┴─────────┴───────┘
```

Syntax overview

The basic syntax for using a window function can be written as follows:

```
SELECT <window_function_name>([<argument_list>]) OVER {() | <window_name> | (<window_spec>)}
FROM <table_name>
WINDOW <window_name> AS (<window_spec>);
```

Use the `OVER` clause to define a **window** for a function — a set of rows over which the function is calculated. After the `OVER` keyword, you can specify:

- `()` — empty parentheses, if you want to pass all rows from a dataset to a window function;

  Example

  The following query reads all rows from the `wnd_func_table` table and adds row numbers and the sum of all values in the `value` column to the additional `number` and `sum` columns:

```
SELECT
    group_id,
    sort_id,
    value,
    row_number() OVER() AS number,
    sum(value) OVER() AS sum
FROM wnd_func_table;
```

```
┌─group_id─┬─sort_id─┬─value─┬─number─┬─sum─┐
│        1 │       1 │    10 │      1 │ 186 │
│        1 │       2 │    20 │      2 │ 186 │
│        1 │       3 │    30 │      3 │ 186 │
│        1 │       4 │    40 │      4 │ 186 │
│        1 │       5 │    50 │      5 │ 186 │
│        2 │       1 │     1 │      6 │ 186 │
│        2 │       2 │     2 │      7 │ 186 │
│        2 │       3 │     3 │      8 │ 186 │
│        2 │       4 │     4 │      9 │ 186 │
│        2 │       4 │     5 │     10 │ 186 │
│        2 │       4 │     6 │     11 │ 186 │
│        2 │       5 │     7 │     12 │ 186 │
│        2 │       6 │     8 │     13 │ 186 │
└──────────┴─────────┴───────┴────────┴─────┘
```

- `<window_name>` — name assigned to a window defined via the `WINDOW` clause within a query (in other words, the `WINDOW` clause allows you to describe a window once and then reuse it for multiple window functions in the `SELECT` query);
- `<window_spec>` — window specification in parentheses as a set of optional clauses (see the syntax below).

TIP

If you need to filter your query data based on values of window functions, calculate window functions within CTE or subquery.

Window specification

Below is the basic syntax of the window specification (`<window_spec>`):

```
[<window_name>]
[PARTITION BY <partition_column> [, ...]]
[ORDER BY <order_column> [ASC|DESC] [, ...]]
[{RANGE | ROWS} {<frame_start> | BETWEEN <frame_start> AND <frame_end>}]
```

where:

- `PARTITION BY` — column that contains values used to divide rows into windows. A window function calculates a value for each row of the source dataset over rows of the window to which this row belongs. If you omit `PARTITION BY` in a query, a window function is evaluated over all rows.

  Example

  The following query uses the `PARTITION BY` clause to divide data rows into windows by values of the `group_id` column:

```
SELECT
    group_id,
    sort_id,
    value,
    row_number() OVER (w) AS number,
    sum(value) OVER (w) AS sum
FROM wnd_func_table
WINDOW w AS (PARTITION BY group_id);
```

  The `row_number` and `sum` functions are calculated from data of each window separately (in this example, a window is a set of rows with the same values in the `group_id` column):

```
┌─group_id─┬─sort_id─┬─value─┬─number─┬─sum─┐
│        1 │       1 │    10 │      1 │ 150 │
│        1 │       2 │    20 │      2 │ 150 │
│        1 │       3 │    30 │      3 │ 150 │
│        1 │       4 │    40 │      4 │ 150 │
│        1 │       5 │    50 │      5 │ 150 │
│        2 │       1 │     1 │      1 │  36 │
│        2 │       2 │     2 │      2 │  36 │
│        2 │       3 │     3 │      3 │  36 │
│        2 │       4 │     4 │      4 │  36 │
│        2 │       4 │     5 │      5 │  36 │
│        2 │       4 │     6 │      6 │  36 │
│        2 │       5 │     7 │      7 │  36 │
│        2 │       6 │     8 │      8 │  36 │
└──────────┴─────────┴───────┴────────┴─────┘
```

- `ORDER BY` — column that contains values used to order rows inside a window during a window function calculation.
  If `ORDER BY` is specified, the function is calculated for rows from the beginning of the window to the current row (including all rows with the same value in the field specified in the `ORDER BY` clause) according to the specified sort order. That is, for each row in the source dataset, it is selected a **frame** — a set of rows over which the function should be calculated (for example, aggregate functions used as window functions operate with frame rows, rather than with the entire window). You can use the RANGE or ROWS clause to specify another rule for defining a frame.

  Example

  The following query sets sorting by values of the `sort_id` column:

```
SELECT
    group_id,
    sort_id,
    value,
    row_number() OVER (w) AS number,
    groupArray(value) OVER (w) AS frame_values,
    sum(value) OVER (w) AS sum
FROM wnd_func_table
WINDOW w AS (PARTITION BY group_id ORDER BY sort_id ASC);
```

  The output table includes the following additional columns with results of window functions:

  - `number` — row number within a window;
  - `frame_values` — the `value` column's values in rows that belong to the frame for the current row;
  - `sum` — sum of the `value` column's values in rows included into the frame to which the current row belongs (in this example — each value is summed up with all previous ones).

```
┌─group_id─┬─sort_id─┬─value─┬─number─┬─frame_values──────┬─sum─┐
│        1 │       1 │    10 │      1 │ [10]              │  10 │
│        1 │       2 │    20 │      2 │ [10,20]           │  30 │
│        1 │       3 │    30 │      3 │ [10,20,30]        │  60 │
│        1 │       4 │    40 │      4 │ [10,20,30,40]     │ 100 │
│        1 │       5 │    50 │      5 │ [10,20,30,40,50]  │ 150 │
│        2 │       1 │     1 │      1 │ [1]               │   1 │
│        2 │       2 │     2 │      2 │ [1,2]             │   3 │
│        2 │       3 │     3 │      3 │ [1,2,3]           │   6 │
│        2 │       4 │     4 │      4 │ [1,2,3,4,5,6]     │  21 │
│        2 │       4 │     5 │      5 │ [1,2,3,4,5,6]     │  21 │
│        2 │       4 │     6 │      6 │ [1,2,3,4,5,6]     │  21 │
│        2 │       5 │     7 │      7 │ [1,2,3,4,5,6,7]   │  28 │
│        2 │       6 │     8 │      8 │ [1,2,3,4,5,6,7,8] │  36 │
└──────────┴─────────┴───────┴────────┴───────────────────┴─────┘
```

  The sort order of rows in a window is not related to the query-level sort order.

ROWS

The `ROWS` clause allows you to define a frame by specifying a fixed number of rows preceding or following the current row in one of the following ways:

```
ROWS <frame_start>
ROWS BETWEEN <frame_start> AND <frame_end>
```

Bounds of a frame (`<frame_start>` and `<frame_end>`) can be defined with the following expressions.

CURRENT ROW	Frame starts or ends with the current row
UNBOUNDED PRECEDING	Frame starts with the first row of the window
UNBOUNDED FOLLOWING	Frame ends with the last row of the window
<N> PRECEDING	Frame includes `<N>` rows before the current row
<M> FOLLOWING	Frame includes `<M>` rows after the current row

If `<frame_end>` is omitted, a frame ends with `CURRENT ROW` by default.

Example — ROWS <frame_start>

A frame includes the current row and two previous ones:

```
SELECT
    group_id,
    sort_id,
    value,
    groupArray(value) OVER (PARTITION BY group_id ORDER BY sort_id ASC
            ROWS 2 PRECEDING) AS frame_values
FROM wnd_func_table;
```

The `frame_values` column contains values of the `value` column in all rows that fall into a frame defined for each row separately:

```
┌─group_id─┬─sort_id─┬─value─┬─frame_values─┐
│        1 │       1 │    10 │ [10]         │
│        1 │       2 │    20 │ [10,20]      │
│        1 │       3 │    30 │ [10,20,30]   │
│        1 │       4 │    40 │ [20,30,40]   │
│        1 │       5 │    50 │ [30,40,50]   │
│        2 │       1 │     1 │ [1]          │
│        2 │       2 │     2 │ [1,2]        │
│        2 │       3 │     3 │ [1,2,3]      │
│        2 │       4 │     4 │ [2,3,4]      │
│        2 │       4 │     5 │ [3,4,5]      │
│        2 │       4 │     6 │ [4,5,6]      │
│        2 │       5 │     7 │ [5,6,7]      │
│        2 │       6 │     8 │ [6,7,8]      │
└──────────┴─────────┴───────┴──────────────┘
```

Example — ROWS BETWEEN <frame_start> AND <frame_end>

A frame includes the current row and all subsequent rows until the end of the window:

```
SELECT
    group_id,
    sort_id,
    value,
    groupArray(value) OVER (PARTITION BY group_id ORDER BY sort_id ASC
        ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING) AS frame_values
FROM wnd_func_table;
```

```
┌─group_id─┬─sort_id─┬─value─┬─frame_values──────┐
│        1 │       1 │    10 │ [10,20,30,40,50]  │
│        1 │       2 │    20 │ [20,30,40,50]     │
│        1 │       3 │    30 │ [30,40,50]        │
│        1 │       4 │    40 │ [40,50]           │
│        1 │       5 │    50 │ [50]              │
│        2 │       1 │     1 │ [1,2,3,4,5,6,7,8] │
│        2 │       2 │     2 │ [2,3,4,5,6,7,8]   │
│        2 │       3 │     3 │ [3,4,5,6,7,8]     │
│        2 │       4 │     4 │ [4,5,6,7,8]       │
│        2 │       4 │     5 │ [5,6,7,8]         │
│        2 │       4 │     6 │ [6,7,8]           │
│        2 │       5 │     7 │ [7,8]             │
│        2 │       6 │     8 │ [8]               │
└──────────┴─────────┴───────┴───────────────────┘
```

RANGE

The `RANGE` clause is also intended to specify a window frame. Unlike `ROWS` that operates with physical rows, `RANGE` operates with row ranges defined by the `ORDER BY` clause. This means that rows with the same values in `ORDER BY` columns are counted as one row. For the `ROWS` clause, each physical record in the dataset is considered a separate row.

There are two ways to use the `RANGE` clause:

```
RANGE <frame_start>
RANGE BETWEEN <frame_start> AND <frame_end>
```

You can use expressions listed in the table below to set frame bounds (`<frame_start>` and `<frame_end>`).

CURRENT ROW	Frame starts or ends with a range of rows with the same values in `ORDER BY` columns as the current row (all rows of the range are included in the frame)
UNBOUNDED PRECEDING	Frame starts with the first row of the window
UNBOUNDED FOLLOWING	Frame ends with the last row of the window
<offset> PRECEDING	Frame includes rows with the `ORDER BY` column's values that are at most `<offset>` less than the value of the `ORDER BY` column in the current row. The `<offset> PRECEDING` expression can only be used if a single column is set in the `ORDER BY` clause
<offset> FOLLOWING	Frame includes rows with the `ORDER BY` column's values that are at most `<offset>` greater than the value of the `ORDER BY` column in the current row. The `<offset> FOLLOWING` expression can only be used if a single column is set in the `ORDER BY` clause

If `<frame_end>` is omitted, a frame ends with `CURRENT ROW` by default.

If `ORDER BY` is specified in a query, but the frame is not explicitly described, the default frame is applied: `RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW`.

Example — CURRENT ROW

A frame includes only the current row:

```
SELECT
    group_id,
    sort_id,
    value,
    groupArray(value) OVER (PARTITION BY group_id ORDER BY sort_id ASC
        RANGE BETWEEN CURRENT ROW AND CURRENT ROW) AS frame_values
FROM wnd_func_table;
```

In the second window (`group_id=2`), a range of rows with the same value in the `sort_id` field is treated as a single row:

```
┌─group_id─┬─sort_id─┬─value─┬─frame_values─┐
│        1 │       1 │    10 │ [10]         │
│        1 │       2 │    20 │ [20]         │
│        1 │       3 │    30 │ [30]         │
│        1 │       4 │    40 │ [40]         │
│        1 │       5 │    50 │ [50]         │
│        2 │       1 │     1 │ [1]          │
│        2 │       2 │     2 │ [2]          │
│        2 │       3 │     3 │ [3]          │
│        2 │       4 │     4 │ [4,5,6]      │
│        2 │       4 │     5 │ [4,5,6]      │
│        2 │       4 │     6 │ [4,5,6]      │
│        2 │       5 │     7 │ [7]          │
│        2 │       6 │     8 │ [8]          │
└──────────┴─────────┴───────┴──────────────┘
```

Example — <offset> PRECEDING/FOLLOWING

A frame for each row includes rows where the `number` value (the `ORDER BY` sort field) is at most `10` less than and at most `5` greater than the `number` value in the current row:

```
SELECT
    number,
    groupArray(number) OVER (ORDER BY number ASC
        RANGE BETWEEN 10 PRECEDING AND 5 FOLLOWING) AS frame_values
FROM values('number Int8', 10, 20, 25, 27, 30, 40, 15, 50, 60, 7, 5, 2);
```

In the resulting table, the `frame_values` column contains arrays of `number` values for the frame of each row:

```
┌─number─┬─frame_values──────┐
│      2 │ [2,5,7]           │
│      5 │ [2,5,7,10]        │
│      7 │ [2,5,7,10]        │
│     10 │ [2,5,7,10,15]     │
│     15 │ [5,7,10,15,20]    │
│     20 │ [10,15,20,25]     │
│     25 │ [15,20,25,27,30]  │
│     27 │ [20,25,27,30]     │
│     30 │ [20,25,27,30]     │
│     40 │ [30,40]           │
│     50 │ [40,50]           │
│     60 │ [50,60]           │
└────────┴───────────────────┘
```

Function types

Window functions can be combined into the following groups:

- **Aggregate** — aggregate functions (for example, `sum`, `avg`, `count`, `min`, `max`) can be used as window functions.
- **Ranking** — functions that rank rows in a window. For example, such functions can be used to assign a sequence number to a row or to rank rows.

Function	Return value
row_number()	Row number within a window. Numbering starts from `1` and does not depend on whether rows have the same values in the `ORDER BY` column
rank()	Row rank. The same rank is returned for rows with the same values in the `ORDER BY` column, and `N-1` rank values are skipped for the following rows, where `N` is the number of rows in the previous group of rows with the same values in the `ORDER BY` column (that is, ranking is with gaps)
dense_rank()	Row rank. The same rank is returned for rows with the same values in the `ORDER BY` column, and the numeric rank for the following rows is not skipped (that is, ranking is sequential)

- **Value functions** — functions that allow you to access different rows of a frame relative to the current row, as well as values of rows at the beginning or end of the frame.

Function	Return value
lagInFrame(value, [N]) leadInFrame(value, [N])	Value of the `value` column from a row that precedes or follows the current row by `N` rows within the frame. The default value of `N` is `1`
first_value(value) last_value(value)	Value of the `value` column from the first or last row of the frame
nth_value(value, N)	Value of the `value` column from the N-th row of the frame

Table functions

Elena Dvoryadkina

Contents

To Table of Contents

Functions

Examples

Table functions create table objects in ADQM based on data from external sources and allow you to read and write data using the following queries:

- `SELECT … FROM <table_function>` — creates a temporary table available only in the current query (the table will be dropped after the query completes);
- `CREATE TABLE [<db_name>.]<table_name> AS <table_function>` — creates the same table as a table function returns (the created table will work in the same way as the corresponding table function);
- `INSERT INTO TABLE FUNCTION <table_function> …` — inserts data into a table referenced by the table function.

IMPORTANT

To use table functions, enable the allow_ddl setting.

Functions

ADQM supports a set of table functions for integration with various external systems. The following table lists examples.

Function	Return value
file	Table to select/insert local file data
url	Table to select/insert data on a remote HTTP/HTTPS server at the specified URL
hdfs, hdfsCluster	Table to select/insert data of a file in HDFS
mysql	Table to select/insert data stored on a remote MySQL server
postgresql	Table to select/insert data of a table in a remote PostgreSQL database
mongodb	Table to select data from a remote MongoDB server
s3, s3Cluster	Table to select/insert data of a file in Amazon S3
jdbc	Table connected to a data source via a JDBC driver
odbc	Table connected to a data source via ODBC
cluster	Dataset from a cluster configured in the `remote_servers` section (*config.xml*)
remote	Dataset from remote ClickHouse servers

To get the full list of supported table functions with their descriptions, refer to the Table Functions section of the ClickHouse documentation.

Examples

Queries below show how to use table functions on the example of the `file` function that allows you to read or write local file data. Before executing test requests, prepare a source file:

1. In the *config.xml* configuration file, set the */var/lib/clickhouse/user_files* folder as the `user_files_path` parameter value. Check the setting:

```
$ sudo grep user_files_path /etc/clickhouse-server/config.xml
    <user_files_path>/var/lib/clickhouse/user_files/</user_files_path>
```

2. Create a CSV file with test data in the */var/lib/clickhouse/user_files* folder:

```
$ sudo cat > /var/lib/clickhouse/user_files/file_func_source.csv
1,2,3
4,5,6
7,8,9
```

Run the following queries to view how the `file` function works:

1. `CREATE TABLE`

```
CREATE TABLE file_table AS file('file_func_source.csv', 'CSV', 'column_1 UInt32, column_2 UInt32, column_3 UInt32');
```

   As a result, a table with the specified structure and data from the *file_func_source.csv* file is added to the current database.

2. `SELECT … FROM`
   Both of the following queries return a table containing data from the specified file:

```
SELECT * FROM file_table;
```

```
SELECT * FROM file('file_func_source.csv', 'CSV', 'column_1 UInt32, column_2 UInt32, column_3 UInt32');
```

   The output:

```
┌─column_1─┬─column_2─┬─column_3─┐
│        1 │        2 │        3 │
│        4 │        5 │        6 │
│        7 │        8 │        9 │
└──────────┴──────────┴──────────┘
```

3. `INSERT INTO TABLE FUNCTION`
   The following query inserts data into the source file:

```
INSERT INTO FUNCTION file('file_func_source.csv', 'CSV', 'column_1 UInt32, column_2 UInt32, column_3 UInt32')
    VALUES (10, 20, 30), (40, 50, 60);
```

   Make sure the data is written to the file:

```
$ sudo cat /var/lib/clickhouse/user_files/file_func_source.csv
1,2,3
4,5,6
7,8,9
10,20,30
40,50,60
```

   When calling the `file` table function, new data rows are also included in the selection:

```
SELECT * FROM file_table;
```

```
┌─column_1─┬─column_2─┬─column_3─┐
│        1 │        2 │        3 │
│        4 │        5 │        6 │
│        7 │        8 │        9 │
│       10 │       20 │       30 │
│       40 │       50 │       60 │
└──────────┴──────────┴──────────┘
```

NOTE

For information on basic queries to operate data tables in ADQM, see Tables overview.

The JSON data type

Elena Dvoryadkina

Contents

To Table of Contents

- JSON data type
  - Select data from dynamic columns
  - Handle data changes
  - Unstructured data
  - Add primary keys
- Map table columns to JSON fields
- Functions to work with JSON
- Import JSON data
  - JSON import formats
  - Import JSON objects as strings
  - Skip unknown fields
  - Import JSON with nested objects
  - Auto detect data types on JSON import
- Export JSON data
  - JSON output formats
  - Export JSON to a file

To work with JSON data in ADQM, you can:

- use JSON data type (available since version 22.3);
- explicitly define table columns mapping them to JSON fields;
- store JSON objects as strings and use special functions to extract values.

The current article describes these approaches using examples based on a test data set that includes information about persons (with fields of different types — `name`, `age`, `hire_date`, `address`) and can be structured differently in JSON format.

JSON data type

Starting with version 22.3, ADQM supports the **Object(JSON)** data type (an alias is **JSON**) that allows automatic creation of columns for a table based on incoming JSON data. That is, the table structure can dynamically change depending on data changes/updates.

> IMPORTANT
>
> The JSON data type is currently under development, but you can use it experimentally. To do this, enable the following option before creating a table: `SET allow_experimental_object_type = 1`.

Example

Create the *persons.json* file with JSON data in the */var/lib/clickhouse/user_files* folder:

```
[
  {"name":"john", "age":42, "hire_date":"2021-01-01", "address": {"country":"USA", "city":"New York"}},
  {"name":"mary", "age":25, "hire_date":"2022-02-02", "address": {"country":"UK", "city":"London"}},
  {"name":"andrew", "age":33, "hire_date":"2023-03-03", "address": {"country":"Germany", "city":"Berlin"}}
]
```

When importing data, use the `JSONAsObject` format to specify that each JSON object should be stored as a separate table row in a JSON column.

For example, the following query specifies that JSON objects from the *persons.json* file are stored in the `person` column of the JSON type (if the column is not explicitly specified, the `json` column will be used):

```
SELECT * FROM file('persons.json', 'JSONAsObject', 'person JSON');
```

Query execution result:

```
┌─person──────────────────────────────────────────────────────────────────────────────────────────────────┐
│ {"address.city":"New York","address.country":"USA","age":42,"hire_date":"2021-01-01","name":"john"}     │
│ {"address.city":"London","address.country":"UK","age":25,"hire_date":"2022-02-02","name":"mary"}        │
│ {"address.city":"Berlin","address.country":"Germany","age":33,"hire_date":"2023-03-03","name":"andrew"} │
└─────────────────────────────────────────────────────────────────────────────────────────────────────────┘
```

To query this data effectively, it is recommended to create a MergeTree table and insert data into it via the `INSERT INTO` query. Run the following commands sequentially:

```
SET allow_experimental_object_type = 1;
```

```
CREATE TABLE json_table (person JSON) ENGINE = MergeTree ORDER BY tuple();
```

```
INSERT INTO json_table SELECT * FROM file('persons.json', 'JSONAsObject', 'person JSON');
```

Run the `DESCRIBE` query to verify that the table contains a single JSON column:

```
DESCRIBE json_table;
```

```
┌─name───┬─type───────────┐
│ person │ Object('json') │
└────────┴────────────────┘
```

With the `describe_extend_object_types` setting, you can see which columns will be dynamically created in the table according to fields of the JSON objects:

```
DESCRIBE json_table SETTINGS describe_extend_object_types=1;
```

```
┌─name───┬─type──────────────────────────────────────────────────────────────────────────────────┐
│ person │ Tuple(address Tuple(city String, country String), age Int8, hire_date String, name String) │
└────────┴───────────────────────────────────────────────────────────────────────────────────┘
```

> NOTE
>
> Note that dynamically created columns cannot be used as primary keys or sort keys, as ADQM does not know about these columns when creating a table. Therefore, the `CREATE TABLE` query uses the `ORDER BY tuple()` clause, i.e. the table does not have a primary key. The Add primary keys section describes how to work around this limitation.

Select data from dynamic columns

You can access dynamically created columns of a table in queries. For example:

```
SELECT person.name, person.address.country FROM json_table;
```

```
┌─person.name─┬─person.address.country─┐
│ john        │ USA                    │
│ mary        │ UK                     │
│ andrew      │ Germany                │
└─────────────┴────────────────────────┘
```

When fetching data from JSON-based columns, you can use them to sort and aggregate data.

For example, the following query returns rows sorted by person names:

```
SELECT person.name, person.address.city FROM json_table ORDER BY person.name;
```

```
┌─person.name─┬─person.address.city─┐
│ andrew      │ Berlin              │
│ john        │ New York            │
│ mary        │ London              │
└─────────────┴─────────────────────┘
```

Handle data changes

If JSON object fields change in a source file (for example, new fields are added or types of existing ones change), ADQM will automatically adjust table columns according to a new structure of JSON objects when inserting data.

Add new columns

1. Add an `id` field to the *persons.json* source file:

```
[
  {"id":1, "name":"john", "age":42, "hire_date":"2021-01-01", "address": {"country":"USA", "city":"New York"}},
  {"id":2, "name":"mary", "age":25, "hire_date":"2022-02-02", "address": {"country":"UK", "city":"London"}},
  {"id":3, "name":"andrew", "age":33, "hire_date":"2023-03-03", "address": {"country":"Germany", "city":"Berlin"}}
]
```

2. Insert data into the table from the JSON file:

```
INSERT INTO json_table SELECT * FROM file('persons.json', 'JSONAsObject', 'person JSON');
```

3. Make sure a new `id` column of the Int8 type is added to the table schema:

```
DESCRIBE json_table SETTINGS describe_extend_object_types=1;
```

```
┌─name───┬─type───────────────────────────────────────────────────────────────────────────────────────────┐
│ person │ Tuple(address Tuple(city String, country String), age Int8, hire_date String, id Int8, name String) │
└────────┴────────────────────────────────────────────────────────────────────────────────────────────┘
```

Change columns

1. Change the `id` field values in the original JSON file from numbers to strings, for example:

```
[
  {"id":"user_1", "name":"john", "age":42, "hire_date":"2021-01-01", "address": {"country":"USA", "city":"New York"}},
  {"id":"user_2", "name":"mary", "age":25, "hire_date":"2022-02-02", "address": {"country":"UK", "city":"London"}},
  {"id":"user_3", "name":"andrew", "age":33, "hire_date":"2023-03-03", "address": {"country":"Germany", "city":"Berlin"}}
]
```

2. Insert data into the table:

```
INSERT INTO json_table SELECT * FROM file('persons.json', 'JSONAsObject', 'person JSON');
```

3. Make sure the `id` column type is automatically changed to String:

```
DESCRIBE json_table SETTINGS describe_extend_object_types=1;
```

```
┌─name───┬─type─────────────────────────────────────────────────────────────────────────────────────────────┐
│ person │ Tuple(address Tuple(city String, country String), age Int8, hire_date String, id String, name String) │
└────────┴──────────────────────────────────────────────────────────────────────────────────────────────┘
```

Unstructured data

If JSON data is unstructured (for example, JSON objects have different sets of fields), ADQM will anyway consider all fields and fill in fields for which no values are defined with default values for the corresponding data types (for example, `0` for numbers, empty strings for strings).

Example

In the *persons.json* source file, change the structure of the first JSON object — remove the `address.city` field and add a new `position` field:

```
[
  {"name":"john", "age":42, "hire_date":"2021-01-01", "address": {"country":"USA"}, "position":"manager"},
  {"name":"mary", "age":25, "hire_date":"2022-02-02", "address": {"country":"UK", "city":"London"}},
  {"name":"andrew", "age":33, "hire_date":"2023-03-03", "address": {"country":"Germany", "city":"Berlin"}}
]
```

Create a new table and paste data from the source file into it:

```
SET allow_experimental_object_type = 1;
```

```
CREATE TABLE unstruct_json_table (person JSON) ENGINE = MergeTree ORDER BY tuple();
```

```
INSERT INTO unstruct_json_table SELECT * FROM file('persons.json', 'JSONAsObject', 'person JSON');
```

Check values of table columns:

```
SELECT person.name, person.position, person.address.city FROM unstruct_json_table;
```

```
┌─person.name─┬─person.position─┬─person.address.city─┐
│ john        │ manager         │                     │
│ mary        │                 │ London              │
│ andrew      │                 │ Berlin              │
└─────────────┴─────────────────┴─────────────────────┘
```

Default values in table columns are taken into account when calculating aggregate functions. For example, the following query with the `groupArray` function returns an array of values of the `person.address.city` column, where the first value is an empty string as the first JSON object does not contain the `city` field:

```
SELECT groupArray(person.address.city) FROM  unstruct_json_table;
```

```
┌─groupArray(person.address.city)─┐
│ ['','London','Berlin']          │
└─────────────────────────────────┘
```

Add primary keys

As noted above, dynamically created columns cannot be used as primary or sort keys. When creating a table, it is recommended that you use the JSON type for semi-structured parts of data that can potentially be modified in the JSON source, but explicitly specify separate columns for data which type and structure can be clearly declared, and use those columns as primary keys.

Example

In the *persons.json* source data file, add the `person` dedicated field for JSON objects:

```
[
  {"person":{"address":{"city":"New York","country":"USA"},"age":42,"hire_date":"2021-01-01","name":"john"}},
  {"person":{"address":{"city":"London","country":"UK"},"age":25,"hire_date":"2022-02-02","name":"mary"}},
  {"person":{"address":{"city":"Berlin","country":"Germany"},"age":33,"hire_date":"2023-03-03","name":"andrew"}}
]
```

Create a table with the following columns:

- `name` — to store values of the JSON objects' `name` field extracted with the `JSONExtractString` function (this column is used as a sort key);
- `person` — to store JSON objects.

```
SET allow_experimental_object_type = 1;
```

```
CREATE TABLE sort_key_json_table (name String DEFAULT JSONExtractString(toJSONString(person), 'name'), person JSON)
ENGINE = MergeTree
ORDER BY name;
```

Insert data into the table from the *persons.json* file:

```
INSERT INTO sort_key_json_table (name, person) SELECT
    JSONExtractString(toJSONString(person), 'person.name'),
    person
FROM file('persons.json', 'JSONAsObject', 'person JSON');
```

In the resulting table, rows are sorted by the `name` column:

```
SELECT * FROM sort_key_json_table;
```

```
┌─name───┬─person───────────────────────────────────────────┐
│ andrew │ ((('Berlin','Germany'),33,'2023-03-03','andrew')) │
│ john   │ ((('New York','USA'),42,'2021-01-01','john'))     │
│ mary   │ ((('London','UK'),25,'2022-02-02','mary'))        │
└────────┴───────────────────────────────────────────────────┘
```

Map table columns to JSON fields

ADQM can work with JSON data without using the Object(JSON) data type — this is relevant for all versions of ADQM (including versions earlier than 22.3). For this, it is recommended to create a table with explicitly defined columns mapping them to all JSON fields. This requires you to know the JSON data structure and field types, and make sure the table schema is maintained if JSON fields are added/changed. For nested JSON objects, you can use the Tuple, Map, and Nested data types when defining table columns.

This approach is optimal when JSON data has a known fixed structure — it can provide better performance and the ability to control codecs and flexibly configure primary and sort keys.

Example

Prepare the */var/lib/clickhouse/user_files/persons.json* file with JSON data without nested structures:

```
[
  {"name":"john", "age":42, "hire_date":"2021-01-01"},
  {"name":"mary", "age":25, "hire_date":"2022-02-02"},
  {"name":"andrew", "age":33, "hire_date":"2023-03-03"}
]
```

Create a table with appropriate columns for all JSON fields:

```
CREATE TABLE struct_json_table (name String, age Int32, hire_date Date) ENGINE = MergeTree ORDER BY name;
```

Insert data from the file into the table using the `file` table function:

```
INSERT INTO struct_json_table SELECT * FROM file('persons.json', 'JSONEachRow');
```

As a result, data from JSON fields are written to the corresponding table columns:

```
SELECT * FROM struct_json_table;
```

```
┌─name───┬─age─┬──hire_date─┐
│ andrew │  33 │ 2023-03-03 │
│ john   │  42 │ 2021-01-01 │
│ mary   │  25 │ 2022-02-02 │
└────────┴─────┴────────────┘
```

This example uses `JSONEachRow` — one of the supported input formats to import JSON data. The JSON import formats section describes features of other formats available in ADQM.

For examples of creating Nested, Tuple, and Map columns to work with nested JSON structures, see the Handle as Structured Data article in the ClickHouse documentation.

Functions to work with JSON

ADQM can store JSON objects as strings and provides special functions for parsing JSON-based strings and extracting field values.

For example, the following query interprets JSON objects from the specified file as strings (the `JSONAsString` format is used), extracts values of the `name` and `age` fields, and writes them to the appropriate columns:

```
SELECT
    JSONExtractString(json, 'name') AS name,
    JSONExtractUInt(json, 'age') AS age
FROM (SELECT * FROM file('persons.json', JSONAsString) AS json);
```

The output:

```
┌─name───┬─age─┐
│ john   │  42 │
│ mary   │  25 │
│ andrew │  33 │
└────────┴─────┘
```

Refer to the json-functions article in the ClickHouse documentation for the full list of supported functions.

Import JSON data

ADQM provides a set of formats and settings for importing JSON data. To choose appropriate options, you first need to determine how input data is structured.

JSON import formats

The table below lists examples of choosing formats to parse incoming JSON data depending on its structure. To run these data import examples, first create a table:

```
CREATE TABLE import_json_table(name String, age Int32, hire_date Date) ENGINE = MergeTree ORDER BY hire_date;
```

Queries in the ***Import data*** column insert data to the `import_json_table` table from the *persons.json* file with the content described in the ***Input JSON*** column (the *persons.json* file should be located in a folder from which clickhouse-client is started).

JSON import formats

Input JSON	Import data
Array of JSON objects:	Use the `JSONEachRow` format:
NDJSON (newline-delimited JSON):	Use the `JSONEachRow` format:
JSON arrays:	Use the `JSONCompactEachRow` format:
A single JSON object that contains fields with arrays of values:	Use the `JSONColumns` format:
Array of JSON arrays:	Use the `JSONCompactColumns` format:

Array of JSON objects:

```
[
  {"name":"john", "age":42, "hire_date":"2021-01-
01"},
  {"name":"mary", "age":25, "hire_date":"2022-02-
02"},
  {"name":"andrew", "age":33, "hire_date":"2023-03-
03"}
]
```

Use the `JSONEachRow` format:

```
INSERT INTO import_json_table
FROM INFILE 'persons.json' FORMAT JSONEachRow;
```

NDJSON (newline-delimited JSON):

```
{"name":"john","age":42,"hire_date":"2021-01-01"}
{"name":"mary","age":25,"hire_date":"2022-02-02"}
{"name":"andrew","age":33,"hire_date":"2023-03-03"}
```

Use the `JSONEachRow` format:

```
INSERT INTO import_json_table
FROM INFILE 'persons.json' FORMAT JSONEachRow;
```

JSON arrays:

```
["john", 42, "2021-01-01"],
["mary", 25, "2022-02-02"],
["andrew", 33, "2023-03-03"]
```

Use the `JSONCompactEachRow` format:

```
INSERT INTO import_json_table
FROM INFILE 'persons.json' FORMAT JSONCompactEachRow;
```

A single JSON object that contains fields with arrays of values:

```
{
  "name": ["john", "mary", "andrew"],
  "age": ["42", "25", "33"],
  "hire_date": ["2021-01-01", "2022-02-02", "2023-03-
03"]
}
```

Use the `JSONColumns` format:

```
INSERT INTO import_json_table
FROM INFILE 'persons.json' FORMAT JSONColumns;
```

Array of JSON arrays:

```
[
  ["john", "mary", "andrew"],
  ["42", "25", "33"],
  ["2021-01-01", "2022-02-02", "2023-03-03"]
]
```

Use the `JSONCompactColumns` format:

```
INSERT INTO import_json_table
FROM INFILE 'persons.json' FORMAT JSONCompactColumns;
```

Each of the examples above imports data into the table as follows:

```
┌─name───┬─age─┬──hire_date─┐
│ john   │  42 │ 2021-01-01 │
│ mary   │  25 │ 2022-02-02 │
│ andrew │  33 │ 2023-03-03 │
└────────┴─────┴────────────┘
```

The full list of formats for JSON data import is available in the Formats for Input and Output Data article (see formats prefixed with `JSON`).

Import JSON objects as strings

Use the `JSONAsString` format to import JSON objects into an ADQM table as strings (without parsing). In this case, data is written to a single column of the String type.

If there are several comma-separated JSON objects in the source file, each of them is considered a separate row. The input in square brackets is considered an array of JSON objects.

Example

1. Prepare JSON data in the *persons.json* local file:

```
[
  {"name":"john", "age":42, "hire_date":"2021-01-01"},
  {"name":"mary", "age":25, "hire_date":"2022-02-02"},
  {"name":"andrew", "age":33, "hire_date":"2023-03-03"}
]
```

2. Create a table with a string column:

```
CREATE TABLE json_as_str_table(data String) ENGINE = MergeTree ORDER BY tuple();
```

3. Insert data into the table using the `JSONAsString` format:

```
INSERT INTO json_as_str_table FROM INFILE 'persons.json' FORMAT JSONAsString;
```

4. Resulting table:

```
SELECT * FROM json_as_str_table;
```

```
┌─data───────────────────────────────────────────────┐
│ {"name":"john", "age":42, "hire_date":"2021-01-01"}   │
│ {"name":"mary", "age":25, "hire_date":"2022-02-02"}   │
│ {"name":"andrew", "age":33, "hire_date":"2023-03-03"} │
└───────────────────────────────────────────────────────┘
```

To further process JSON objects stored as strings, use JSON functions.

In the following query, the `JSONExtractString` function parses strings representing JSON objects from the `data` column and returns string values of the `name` field:

```
SELECT JSONExtractString(data, 'name') AS person_name, data FROM json_as_str_table;
```

```
┌─person_name─┬─data──────────────────────────────────────────────────┐
│ john        │ {"name":"john", "age":42, "hire_date":"2021-01-01"}   │
│ mary        │ {"name":"mary", "age":25, "hire_date":"2022-02-02"}   │
│ andrew      │ {"name":"andrew", "age":33, "hire_date":"2023-03-03"} │
└─────────────┴───────────────────────────────────────────────────────┘
```

Skip unknown fields

By default, ADQM ignores unknown columns when importing JSON data.

For example, create a table with two columns (`name` and `hire_date`) and import JSON with three fields (`name`, `hire_date`, and `age` — the same JSON as in the previous section):

```
CREATE TABLE short_table(name String, hire_date Date) ENGINE = MergeTree ORDER BY hire_date;
```

```
INSERT INTO short_table FROM INFILE 'persons.json' FORMAT JSONEachRow;
```

Ensure that the `age` field is ignored:

```
SELECT * FROM short_table;
```

```
┌─name───┬──hire_date─┐
│ john   │ 2021-01-01 │
│ mary   │ 2022-02-02 │
│ andrew │ 2023-03-03 │
└────────┴────────────┘
```

To change this behavior, disable the `input_format_skip_unknown_fields` setting:

```
SET input_format_skip_unknown_fields = 0;
```

ADQM will throw an exception if table and JSON structures are inconsistent:

```
INSERT INTO short_table FROM INFILE 'persons.json' FORMAT JSONEachRow;
```

```
Exception on client:
Code: 117. DB::Exception: Unknown field found while parsing JSONEachRow format: age: (at row 1)
```

Import JSON with nested objects

To import JSON objects with nested structures, use complex data types (JSON, Array, or Tuple) for columns to store nested object.

Example

1. Prepare the */var/lib/clickhouse/user_files/persons.json* file:

```
[
  {"name":"john", "age":42, "hire_date":"2021-01-01", "address": {"country":"USA", "city":"New York"}},
  {"name":"mary", "age":25, "hire_date":"2022-02-02", "address": {"country":"UK", "city":"London"}},
  {"name":"andrew", "age":33, "hire_date":"2023-03-03", "address": {"country":"Germany", "city":"Berlin"}}
]
```

2. Select data using the `file` table function with a table structure passed as a parameter:

```
SELECT * FROM file('persons.json', JSONEachRow, 'name String, age Int32, hire_date Date, address JSON');
```

3. In the resulting table, the `address` column contains nested JSON objects:

```
┌─name───┬─age─┬──hire_date─┬─address─────────────────────────────────┐
│ john   │  42 │ 2021-01-01 │ {"city":"New York","country":"USA"}     │
│ mary   │  25 │ 2022-02-02 │ {"city":"London","country":"UK"}        │
│ andrew │  33 │ 2023-03-03 │ {"city":"Berlin","country":"Germany"}   │
└────────┴─────┴────────────┴─────────────────────────────────────────┘
```

Fields of nested JSONs

To access nested object fields, enable the `input_format_import_nested_json` setting:

```
SET input_format_import_nested_json = 1
```

After that, you can use the dot notation to specify fields of nested JSON objects in queries, wrapping them in backquotes.

For example, to display cities from `address` nested objects in a separate column, use the query:

```
SELECT * FROM file('persons.json', JSONEachRow, 'name String, `address.city` String');
```

```
┌─name───┬─address.city─┐
│ john   │ New York     │
│ mary   │ London       │
│ andrew │ Berlin       │
└────────┴──────────────┘
```

Auto detect data types on JSON import

ADQM can automatically detect data types when importing JSON data. This functionality allows you to quickly create a table with columns which data types match types of fields in the JSON source.

Example

1. Prepare the */var/lib/clickhouse/user_files/persons.json* source file:

```
{"name":"john","age":42,"hire_date":"2021-01-01"}
{"name":"mary","age":25,"hire_date":"2022-02-02"}
{"name":"andrew","age":33,"hire_date":"2023-03-03"}
```

2. Run the `DESCRIBE` query to check which data types are defined for values in the *persons.json* file:

```
DESCRIBE TABLE file('persons.json', JSONEachRow);
```

```
┌─name──────┬─type─────────────┬─default_type─┬─default_expression─┬─comment─┬─codec_expression─┬─ttl_expression─┐
│ name      │ Nullable(String) │              │                    │         │                  │                │
│ age       │ Nullable(Int64)  │              │                    │         │                  │                │
│ hire_date │ Nullable(Date)   │              │                    │         │                  │                │
└───────────┴──────────────────┴──────────────┴────────────────────┴─────────┴──────────────────┴────────────────┘
```

3. Create a table using the `file` table function:

```
CREATE TABLE persons ENGINE = MergeTree ORDER BY tuple() AS SELECT * FROM file('persons.json', JSONEachRow);
```

4. Make sure the table columns match types of fields from the imported JSON:

```
DESCRIBE TABLE persons;
```

```
┌─name──────┬─type─────────────┬─default_type─┬─default_expression─┬─comment─┬─codec_expression─┬─ttl_expression─┐
│ name      │ Nullable(String) │              │                    │         │                  │                │
│ age       │ Nullable(Int64)  │              │                    │         │                  │                │
│ hire_date │ Nullable(Date)   │              │                    │         │                  │                │
└───────────┴──────────────────┴──────────────┴────────────────────┴─────────┴──────────────────┴────────────────┘
```

Export JSON data

Most of the JSON formats that ADQM supports can be used to arrange results of `SELECT` queries.

JSON output formats

Below are examples of formats for exporting data in JSON format from the `export_json_table` table:

```
┌─name───┬─age─┬──hire_date─┐
│ john   │  42 │ 2021-01-01 │
│ mary   │  25 │ 2022-02-02 │
│ andrew │  33 │ 2023-03-03 │
└────────┴─────┴────────────┘
```

Table creation

```
CREATE TABLE export_json_table (`name` String, `age` Int32, `hire_date` Date) ENGINE = MergeTree ORDER BY hire_date;
```

```
INSERT INTO export_json_table VALUES ('john', 42, '2021-01-01'), ('mary', 25, '2022-02-02'), ('andrew', 33, '2023-03-03');
```

- **JSONEachRow**
  Outputs each table row as a separate JSON object on a new line.

```
SELECT * FROM export_json_table FORMAT JSONEachRow;
```

```
{"name":"john","age":42,"hire_date":"2021-01-01"}
{"name":"mary","age":25,"hire_date":"2022-02-02"}
{"name":"andrew","age":33,"hire_date":"2023-03-03"}
```

- **JSONStringsEachRow**
  Differs from `JSONEachRow` only in that it outputs field values as strings rather than typed values of JSON objects.

  For example, the following query outputs the `age` field values as strings instead of numbers:

```
SELECT * FROM export_json_table FORMAT JSONStringsEachRow;
```

```
{"name":"john","age":"42","hire_date":"2021-01-01"}
{"name":"mary","age":"25","hire_date":"2022-02-02"}
{"name":"andrew","age":"33","hire_date":"2023-03-03"}
```

- **JSONCompactEachRow**
  Outputs table rows as JSON arrays — arrays of values without column names.

```
SELECT * FROM export_json_table FORMAT JSONCompactEachRow;
```

```
["john", 42, "2021-01-01"]
["mary", 25, "2022-02-02"]
["andrew", 33, "2023-03-03"]
```

- **JSON**
  Outputs data in JSON format. In addition to the table data, it also outputs column names and types, as well as some additional information (for example, the number of output rows and the total number of rows that could have been output if there were not `LIMIT` in a query).

```
SELECT * FROM export_json_table LIMIT 2 FORMAT JSON;
```

```
{
        "meta":
        [
                {
                        "name": "name",
                        "type": "String"
                },
                {
                        "name": "age",
                        "type": "Int32"
                },
                {
                        "name": "hire_date",
                        "type": "Date"
                }
        ],

        "data":
        [
                {
                        "name": "john",
                        "age": 42,
                        "hire_date": "2021-01-01"
                },
                {
                        "name": "mary",
                        "age": 25,
                        "hire_date": "2022-02-02"
                }
        ],

        "rows": 2,

        "rows_before_limit_at_least": 3,

        "statistics":
        {
                "elapsed": 0.000559985,
                "rows_read": 3,
                "bytes_read": 59
        }
}
```

- **JSONCompact**
  Outputs the same metadata, but uses a compressed form for data itself.

```
SELECT * FROM export_json_table LIMIT 2 FORMAT JSONCompact;
```

```
{
        "meta":
        [
                {
                        "name": "name",
                        "type": "String"
                },
                {
                        "name": "age",
                        "type": "Int32"
                },
                {
                        "name": "hire_date",
                        "type": "Date"
                }
        ],

        "data":
        [
                ["john", 42, "2021-01-01"],
                ["mary", 25, "2022-02-02"]
        ],

        "rows": 2,

        "rows_before_limit_at_least": 3,

        "statistics":
        {
                "elapsed": 0.000571299,
                "rows_read": 3,
                "bytes_read": 59
        }
}
```

The full list of formats to export data as JSON is available in the Formats for Input and Output Data article of the ClickHouse documentation.

Export JSON to a file

To save JSON data from an ADQM table to a file, you can use the `INTO OUTFILE` clause in the `SELECT` query, for example:

```
SELECT * FROM export_json_table INTO OUTFILE 'out.json' FORMAT JSONEachRow;
```

Check that the specified file with data has been created in a folder from which clickhouse-client is started:

```
$ cat out.json
{"name":"john","age":42,"hire_date":"2021-01-01"}
{"name":"mary","age":25,"hire_date":"2022-02-02"}
{"name":"andrew","age":33,"hire_date":"2023-03-03"}
```

Nested data structures

Elena Dvoryadkina

Contents

To Table of Contents

Overview
- flatten_nested = 1
- flatten_nested = 0

Example
- Create a table
- Insert data
- Select data

ADQM/ClickHouse provides the Nested data type, which can be used to store and process nested data structures and can serve as an alternative to the Tuple and Array(Tuple) types. It allows modeling complex data structures without the need to create/store additional tables and use complex queries.

A nested data structure is a table inside a cell of the main table. Each row of the main table may correspond to any number of rows in the nested data structure.

Overview

To define a Nested column in a table, it is necessary to specify parameters of a nested structure — column names with corresponding data types.

Below is an example of the CREATE TABLE query syntax for creating a table with a nested data structure in a column (for simplicity, the example uses short column names and specific data types):

```
CREATE TABLE test_table (
    a String,
    b UInt32,
    ...
    n Nested (
        c String,
        d Int64,
        ...
    )
)
ENGINE = MergeTree ORDER BY tuple();
```

To access columns of nested data structures, use the `<nested_structure>.<nested_column>` format in queries (where `<nested_structure>` is the name of the table's Nested column and `<nested_column>` is the name of a column within the nested structure).

Data format of a nested structure's columns is specified by the `flatten_nested` setting, which can be set to `1` (by default) or `0`.

flatten_nested = 1

Columns of a nested structure are of the Array(Type) type — they store arrays of values of the corresponding data types, and all arrays have the same length. In this case, when reading and processing values of nested columns, you can use functions for working with arrays, including the ARRAY JOIN clause.

In the `SHOW CREATE TABLE` query output, you can see which data types are defined for table columns:

```
SHOW CREATE TABLE test_table;
```

```
   ┌─statement──────────────────────┐
1. │ CREATE TABLE default.test_table │
   │ (                              │
   │     `a` String,                │
   │     `b` UInt32,                │
   │     `n.c` Array(String),       │
   │     `n.d` Array(Int64)         │
   │ )                              │
   │ ENGINE = MergeTree             │
   │ ORDER BY tuple()               │
   │ SETTINGS index_granularity = 8192 │
   └────────────────────────────────┘
```

Insert data

To insert data into a Nested column, pass an array of values separately for each column of the nested structure (the system automatically checks whether all passed arrays of values are of the same length). For example:

```
INSERT INTO test_table VALUES ('a', 1, ['c1', 'c2', 'c3'], [100, 200, 300]);
```

When working with a table that has columns containing nested structures, you can also insert JSON data with the same structure. For example, use the JSONEachRow format in queries:

```
INSERT INTO test_table FORMAT JSONEachRow {"a":"a1", "b":2, "n.c":["c4", "c5"], "n.d":[101, 202]};
```

To pass data as a hierarchical JSON object, first set `input_format_import_nested_json=1`:

```
SET input_format_import_nested_json=1;
```

```
INSERT INTO test_table FORMAT JSONEachRow
{
    "a":"a2",
    "b":"3",
    "n": {
        "c": ["c6", "c7"],
        "d": [111, 222]
    }
};
```

Select data

Querying all data from a table outputs each column of the nested structure separately:

```
SELECT * FROM test_table;
```

```
   ┌─a──┬─b─┬─n.c─────────────┬─n.d───────────┐
1. │ a  │ 1 │ ['c1','c2','c3'] │ [100,200,300] │
2. │ a1 │ 2 │ ['c4','c5']      │ [101,202]     │
3. │ a2 │ 3 │ ['c6','c7']      │ [111,222]     │
   └────┴───┴─────────────────┴───────────────┘
```

flatten_nested = 0

Data of all columns of a nested structure is stored as a single array of tuples — in fact, a Nested column is of the Array(Tuple) type. In this case, an arbitrary level of nesting is supported.

If you recreate the test_table table after setting `flatten_nested = 0`, the output of `SHOW CREATE TABLE` will show the Nested type for the `n` column, although it is actually the same as Array(Tuple(...)):

```
   ┌─statement──────────────────────┐
1. │ CREATE TABLE default.test_table │
   │ (                              │
   │     `a` String,                │
   │     `b` UInt32,                │
   │     `n` Nested(c String, d Int64) │
   │ )                              │
   │ ENGINE = MergeTree             │
   │ ORDER BY tuple()               │
   │ SETTINGS index_granularity = 8192 │
   └────────────────────────────────┘
```

Insert data

Accordingly, data should be inserted into a nested structure as an array of tuples. For example:

```
INSERT INTO test_table VALUES ('a', 1, [('c1', 100), ('c2', 200), ('c3', 300)]);
```

To insert a hierarchical JSON object into a table with a Nested column when the `flatten_nested` parameter is set to `0`, it is not necessary to activate the `input_format_import_nested_json` option, and data for nested columns should be passed as an array:

```
INSERT INTO test_table FORMAT JSONEachRow
{
    "a":"a1",
    "b":"2",
    "n": [
        {"c": "c4", "d": 101},
        {"c": "c5", "d": 202}
        ]
};
```

Select data

Querying all data from a table outputs a column of the Nested type, where each row is an array of tuples of nested columns' values:

```
SELECT * FROM test_table;
```

```
   ┌─a──┬─b─┬─n───────────────────────────────┐
1. │ a  │ 1 │ [('c1',100),('c2',200),('c3',300)] │
2. │ a1 │ 2 │ [('c4',101),('c5',202)]            │
   └────┴───┴─────────────────────────────────┘
```

Data of individual columns can also be accessed via the dot notation:

```
SELECT n.c, n.d FROM test_table;
```

```
   ┌─n.c─────────────┬─n.d───────────┐
1. │ ['c1','c2','c3'] │ [100,200,300] │
2. │ ['c4','c5']      │ [101,202]     │
   └─────────────────┴───────────────┘
```

Example

Create a table

Create the `orders` table to store order data in columns of the appropriate types:

- `order_id` (UInt64) — order identifier;
- `customer_id` (UInt32) — customer identifier;
- `order_date` (Date) — order date.

Also, add a column to the table to store information about products each order includes — the `items` column of the Nested type that defines the following fields of the nested structure:

- `product_id` (UInt32) — product identifier;
- `product_name` (String) — product name;
- `quantity` (UInt16) — number of product units in an order;
- `price_per_unit` (Decimal) — cost of a product unit.

The value of the `flatten_nested` parameter is `1` (default).

```
CREATE TABLE orders (
    order_id UInt64,
    customer_id UInt32,
    order_date Date,
    items Nested (
        product_id UInt32,
        product_name String,
        quantity UInt16,
        price_per_unit Decimal(10, 2)
    )
) ENGINE = MergeTree()
ORDER BY (order_date, order_id);
```

Insert data

Insert test data into the table (for example, some orders over two days):

```
INSERT INTO orders VALUES
(1001, 42, '2025-08-07',
    [101, 102, 103],
    ['Laptop', 'Mouse', 'Keyboard'],
    [1, 2, 1],
    [999.99, 19.99, 49.99]),
(1002, 77, '2025-08-07',
    [104, 105],
    ['Monitor', 'Headphones'],
    [2, 1],
    [199.99, 89.99]),
(1003, 55, '2025-08-07',
    [101, 104, 102],
    ['Laptop', 'Monitor', 'Mouse'],
    [1, 1, 1],
    [999.99, 199.99, 19.99]),
(1004, 24, '2025-08-08',
    [102, 105],
    ['Mouse', 'Headphones'],
    [3, 4],
    [19.99, 89.99]),
(1005, 38, '2025-08-08',
    [101, 103, 104],
    ['Laptop', 'Keyboard', 'Monitor'],
    [1, 1, 1],
    [999.99, 49.99, 199.99]);
```

Select data

Select all data from the table:

```
SELECT * FROM orders;
```

```
   ┌─order_id─┬─customer_id─┬─order_date─┬─items.product_id─┬─items.product_name────────────┬─items.quantity─┬─items.
price_per_unit─┐
1. │     1001 │          42 │ 2025-08-07 │ [101,102,103]    │ ['Laptop','Mouse','Keyboard']   │ [1,2,1]        │ [999.99,19.99,49.
99]  │
2. │     1002 │          77 │ 2025-08-07 │ [104,105]        │ ['Monitor','Headphones']        │ [2,1]          │ [199.99,89.99]
│
3. │     1003 │          55 │ 2025-08-07 │ [101,104,102]    │ ['Laptop','Monitor','Mouse']    │ [1,1,1]        │ [999.99,199.
99,19.99] │
4. │     1004 │          24 │ 2025-08-08 │ [102,105]        │ ['Mouse','Headphones']          │ [3,4]          │ [19.99,89.99]
│
5. │     1005 │          38 │ 2025-08-08 │ [101,103,104]    │ ['Laptop','Keyboard','Monitor'] │ [1,1,1]        │ [999.99,49.
99,199.99] │

   └──────────┴─────────────┴────────────┴──────────────────┴─────────────────────────────────┴────────────────┴──────────────
───┘
```

To display data in a usual tabular form, expanding the Nested structure into several rows, use `ARRAY JOIN` in a query:

```
SELECT * FROM orders ARRAY JOIN items;
```

```
    ┌─order_id─┬─customer_id─┬─order_date─┬─items.product_id─┬─items.product_name─┬─items.quantity─┬─items.price_per_unit─┐
 1. │     1001 │          42 │ 2025-08-07 │              101 │ Laptop             │              1 │               999.99 │
 2. │     1001 │          42 │ 2025-08-07 │              102 │ Mouse              │              2 │                19.99 │
 3. │     1001 │          42 │ 2025-08-07 │              103 │ Keyboard           │              1 │                49.99 │
 4. │     1002 │          77 │ 2025-08-07 │              104 │ Monitor            │              2 │               199.99 │
 5. │     1002 │          77 │ 2025-08-07 │              105 │ Headphones         │              1 │                89.99 │
 6. │     1003 │          55 │ 2025-08-07 │              101 │ Laptop             │              1 │               999.99 │
 7. │     1003 │          55 │ 2025-08-07 │              104 │ Monitor            │              1 │               199.99 │
 8. │     1003 │          55 │ 2025-08-07 │              102 │ Mouse              │              1 │                19.99 │
 9. │     1004 │          24 │ 2025-08-08 │              102 │ Mouse              │              3 │                19.99 │
10. │     1004 │          24 │ 2025-08-08 │              105 │ Headphones         │              4 │                89.99 │
11. │     1005 │          38 │ 2025-08-08 │              101 │ Laptop             │              1 │               999.99 │
12. │     1005 │          38 │ 2025-08-08 │              103 │ Keyboard           │              1 │                49.99 │
13. │     1005 │          38 │ 2025-08-08 │              104 │ Monitor            │              1 │               199.99 │
    └──────────┴─────────────┴────────────┴──────────────────┴────────────────────┴────────────────┴──────────────────────┘
```

Apply aggregation by columns of a nested structure. For example, run the following query to calculate how many units of each product were sold in a month (quantity across all orders):

```
SELECT
    formatDateTime(order_date, '%M') as month,
    items.product_name as product,
    SUM(items.quantity) as units_sold
FROM orders
ARRAY JOIN items
GROUP BY
    formatDateTime(order_date, '%M'),
    items.product_name;
```

```
   ┌─month──┬─product────┬─units_sold─┐
1. │ August │ Keyboard   │          2 │
2. │ August │ Headphones │          5 │
3. │ August │ Laptop     │          3 │
4. │ August │ Mouse      │          6 │
5. │ August │ Monitor    │          4 │
   └────────┴────────────┴────────────┘
```

You can also filter data by a field of a nested structure. For example, the following query outputs sales of the `Laptop` product for each day:

```
SELECT
    order_date,
    items.product_name AS product ,
    SUM(items.quantity) AS quantity
FROM orders
ARRAY JOIN items
WHERE items.product_name = 'Laptop'
GROUP BY
    order_date,
    items.product_name;
```

```
   ┌─order_date─┬─product─┬─quantity─┐
1. │ 2025-08-07 │ Laptop  │        2 │
2. │ 2025-08-08 │ Laptop  │        1 │
   └────────────┴─────────┴──────────┘
```

To return only one row with the number of units sold for each product per date, you can use a subquery and the `groupArray` function:

```
SELECT
    order_date,
    groupArray((product, units_sold))
FROM
(
    SELECT
        order_date,
        product,
        sum(product_sales) AS units_sold
    FROM orders
    ARRAY JOIN
        items.product_name AS product,
        items.quantity AS product_sales
    GROUP BY
        order_date,
        product
)
GROUP BY order_date;
```

```
   ┌─order_date─┬─groupArray((product, units_sold))──────────────────────────────────────┐
1. │ 2025-08-07 │ [('Laptop',2),('Mouse',3),('Monitor',3),('Keyboard',1),('Headphones',1)] │
2. │ 2025-08-08 │ [('Keyboard',1),('Mouse',3),('Monitor',1),('Headphones',4),('Laptop',1)] │
   └────────────┴────────────────────────────────────────────────────────────────────────┘
```

Full-text search

Elena Dvoryadkina

Contents

To Table of Contents

Overview
Manage indexes
Example

Overview

For efficient full-text search on String or FixedString columns (the Array(String), Array(FixedString), Map(String), and Map(String) types are also supported) in tables with very large numbers of records, you can use **inverted indexes** — a special type of data skipping index.

ADQM/ClickHouse builds an inverted index as follows:

- Text of each cell in a column on which the index is compiled is split into tokens — **terms**. By default, a string is split based on spaces, but it is also possible to split text into **n-grams**. All terms are written into a sorted list — **dictionary**.
- Each term is associated with a sorted list of identifiers of rows which contain this term — a **posting list**.

In other words, an inverted index stores a mapping of all terms to their locations in a table (row numbers). If a table has a defined inverted index on a text column, ADQM automatically applies it when executing a query that searches for a term in that column (for example, a query like `SELECT … WHERE <column_name> == | IN | LIKE '<term>'`) — it quickly finds a term in the dictionary and returns the required table rows using posting list.

For example, consider a table with the `text` column of the String type:

```
┌─row_id─┬─text────────────┐
│      0 │ Lower East Side │
│      1 │ Upper West Side │
│      2 │ West Village    │
│      3 │ East Village    │
└────────┴─────────────────┘
```

An inverted index on the `text` column will generally be composed as follows.

Dictionary (terms)	Posting lists
East	0, 3
Lower	0
Side	0, 1
Upper	1
Village	2, 3
West	1, 2

IMPORTANT

Inverted indexes are currently supported as an experimental functionality — it is not recommended to use them on production environments.

Manage indexes

Before creating an inverted index, enable the following option:

```
SET allow_experimental_inverted_index = true;
```

Describe an inverted index in the same way as other types of data skipping index — use the `INDEX` clause within the column description section of the `CREATE TABLE` query :

```
CREATE TABLE <table_name>
(   <column_name> <column_type>, ...
    INDEX <index_name> <index_expr> TYPE inverted([<n>], [<max_rows_per_posting_list>]), ...)
ENGINE = MergeTree
...;
```

where:

- `<index_name>` — index name.
- `<index_expr>` — expression based on which the index will be built (often it is just a text column name).
- `<n>` — tokenizer that specifies how a string in a text column cell is split into terms. For example:
  - `inverted(0)` (or shorter: `inverted()`, `inverted`) — a string is split into words by spaces;
  - `inverted(n)` (where `n` is a number in the `[2, 8]` range) — a string is split into n-grams (substrings of `n` characters).
- `<max_rows_per_posting_list>` — the maximum number of row identifiers per posting list (if `0` — the size of a posting list for a term is not limited, if the parameter is not specified — the default limit set to `64K` is used).

Since inverted indexes map terms to row identifiers (instead of granule identifiers as other data skipping indexes), the `GRANULARITY` parameter supplied to index creation is not relevant.

You can add an inverted index to an existing table using the following query:

```
ALTER TABLE <table_name> ADD INDEX <index_name> <index_expr> TYPE inverted([<n>], [<max_rows_per_posting_list>]);
```

When an index is added to an existing table, it is not automatically updated. `ALTER TABLE` changes metadata, and the index will only be calculated for new data inserted into the table. To apply the index to existing data, run the additional command:

```
ALTER TABLE <table_name> MATERIALIZE INDEX <index_name>;
```

To delete indexes, use the following queries:

- `ALTER TABLE <table_name> DROP INDEX <index_name>` — removes index description from table metadata and deletes index files from disk;
- `ALTER TABLE <table_name> CLEAR INDEX <index_name>` — removes index files from disk without removing the index description from metadata.

Example

Create the `trips` table with New York City taxi data as the ClickHouse documentation describes:

```
CREATE TABLE trips (
    trip_id             UInt32,
    pickup_datetime     DateTime,
    dropoff_datetime    DateTime,
    pickup_longitude    Nullable(Float64),
    pickup_latitude     Nullable(Float64),
    dropoff_longitude   Nullable(Float64),
    dropoff_latitude    Nullable(Float64),
    passenger_count     UInt8,
    trip_distance       Float32,
    fare_amount         Float32,
    extra               Float32,
    tip_amount          Float32,
    tolls_amount        Float32,
    total_amount        Float32,
    payment_type        Enum('CSH' = 1, 'CRE' = 2, 'NOC' = 3, 'DIS' = 4, 'UNK' = 5),
    pickup_ntaname      LowCardinality(String),
    dropoff_ntaname     LowCardinality(String)
)
ENGINE = MergeTree
PRIMARY KEY (pickup_datetime, dropoff_datetime);
```

```
INSERT INTO trips
SELECT
    trip_id,
    pickup_datetime,
    dropoff_datetime,
    pickup_longitude,
    pickup_latitude,
    dropoff_longitude,
    dropoff_latitude,
    passenger_count,
    trip_distance,
    fare_amount,
    extra,
    tip_amount,
    tolls_amount,
    total_amount,
    payment_type,
    pickup_ntaname,
    dropoff_ntaname
FROM s3(
    'https://datasets-documentation.s3.eu-west-3.amazonaws.com/nyc-taxi/trips_{0..2}.gz',
    'TabSeparatedWithNames');
```

Execute a test query for data selection with a condition on the `pickup_ntaname` column:

```
SELECT pickup_datetime,pickup_datetime,pickup_ntaname,dropoff_ntaname
FROM trips
WHERE (pickup_ntaname LIKE '%Village%') OR (pickup_ntaname LIKE '%Square%');
```

```
┌──────pickup_datetime─┬──────pickup_datetime─┬─pickup_ntaname────────────────────────────┬─dropoff_ntaname───────┐
│ 2015-07-01 00:02:24 │ 2015-07-01 00:02:24 │ East Village                              │ Gramercy              │
│ 2015-07-01 00:03:15 │ 2015-07-01 00:03:15 │ Hudson Yards-Chelsea-Flatiron-Union Square │ West Village          │

...

│ 2015-07-09 00:32:13 │ 2015-07-09 00:32:13 │ West Village                              │ East Village          │
└─────────────────────┴─────────────────────┴───────────────────────────────────────────┴───────────────────────┘

  Showed first 10000.

714664 rows in set. Elapsed: 0.831 sec. Processed 3.00 million rows, 18.00 MB (3.61 million rows/s., 21.66 MB/s.)
Peak memory usage: 13.04 MiB.
```

Add an inverted index to the table:

```
SET allow_experimental_inverted_index = true;
```

```
ALTER TABLE trips ADD INDEX inv_idx(pickup_ntaname) TYPE inverted;
```

```
ALTER TABLE trips MATERIALIZE INDEX inv_idx;
```

Repeat the query — note that its execution speed has increased:

```
SELECT pickup_datetime,pickup_datetime,pickup_ntaname,dropoff_ntaname
FROM trips
WHERE (pickup_ntaname LIKE '%Village%') OR (pickup_ntaname LIKE '%Square%');
```

```
┌──────pickup_datetime─┬──────pickup_datetime─┬─pickup_ntaname────────────────────────────┬─dropoff_ntaname───────┐
│ 2015-07-01 00:02:24 │ 2015-07-01 00:02:24 │ East Village                              │ Gramercy              │
│ 2015-07-01 00:03:15 │ 2015-07-01 00:03:15 │ Hudson Yards-Chelsea-Flatiron-Union Square │ West Village          │

...

│ 2015-07-09 00:32:13 │ 2015-07-09 00:32:13 │ West Village                              │ East Village          │
└─────────────────────┴─────────────────────┴───────────────────────────────────────────┴───────────────────────┘

  Showed first 10000.

714664 rows in set. Elapsed: 0.129 sec. Processed 3.00 million rows, 18.00 MB (23.27 million rows/s., 139.63 MB/s.)
Peak memory usage: 13.04 MiB.
```

If you apply the EXPLAIN clause with the `indexes` parameter to the query, you can ensure that the inverted index was used:

```
EXPLAIN indexes = 1 SELECT pickup_datetime,pickup_datetime,pickup_ntaname,dropoff_ntaname
FROM trips
WHERE (pickup_ntaname LIKE '%Village%') OR (pickup_ntaname LIKE '%Square%');
```

```
┌─explain─────────────────────────────────────┐
│ Expression ((Project names + Projection))   │
│   Expression                                │
│     ReadFromMergeTree (default.trips)       │
│     Indexes:                                │
│       PrimaryKey                            │
│         Condition: true                     │
│         Parts: 2/2                          │
│         Granules: 368/368                   │
│       Skip                                  │
│         Name: inv_idx                       │
│         Description: inverted GRANULARITY 1 │
│         Parts: 2/2                          │
│         Granules: 368/368                   │
└─────────────────────────────────────────────┘
```

Improve query performance

Elena Dvoryadkina

Contents

To Table of Contents

- Indexes
  - Primary index
  - Compound key of a primary index
  - Data skipping index (Set)
- Strict data types
  - Example
- Compression and encoding
  - Example
- Query cache
  - Manage query caching
  - Configure query cache
  - View information on query cache
  - Example
- PREWHERE
  - Example
- Optimize JOIN operations

This article describes tools to improve the performance of queries in ADQM.

Indexes

High performance of analytical queries in ADQM is usually achieved due to indexing tables — when table indexes are defined correctly, they can significantly reduce the amount of data to be read from disk for query execution.

For MergeTree tables, ADQM supports the following index types:

- **Primary index** — provides quick searches for ranges of rows that potentially contain the desired values. A key to build a primary index can be compound, i.e. include multiple columns.
- **Data skipping indexes** — store some additional information about the specified column on data blocks and, when executing read queries, allow skipping data blocks that do not contain the requested values. The type of information about a data block that the index aggregates and stores is defined by the index type: MinMax, Set, Bloom Filter, or Inverted.

> TIP
>
> For detailed information on how ADQM indexes work and how to manage them, refer to the Indexes article.

Below are examples that demonstrate how a primary index or a data skipping index (for example, of the Set type) can affect the speed of reading data from a MergeTree table.

Primary index

1. Create a table without a primary key (with one numeric column and one string column) and populate it with test data:

   ```
   CREATE TABLE table_no_primary_key (a Int64, b String) ENGINE = MergeTree ORDER BY tuple();
   ```

   ```
   INSERT INTO table_no_primary_key
   SELECT toInt64(randUniform(0, 1000)), ['a', 'b', 'c', 'd', 'e'][(rand() % 5) + 1]
   FROM numbers(100000000);
   ```

2. Run a query that calculates the number of rows with the `555` value in the `a` column:

   ```
   SELECT count() FROM table_no_primary_key WHERE a = 555;
   ```

   The output shows that the full-table scan was performed — ADQM processed each of the 100 million rows, and this took about 0.5 seconds:

   ```
   ┌─count()─┐
   │  100162 │
   └─────────┘

   1 row in set. Elapsed: 0.512 sec. Processed 100.00 million rows, 800.00 MB (195.30 million rows/s., 1.56 GB/s.)
   ```

3. To improve this query performance, redefine the table to set the `a` column as a primary key:

   ```
   CREATE TABLE table_with_primary_key (a Int64, b String) ENGINE = MergeTree ORDER BY (a);
   ```

   Populate the new table with data from the first table:

   ```
   INSERT INTO table_with_primary_key SELECT * FROM table_no_primary_key;
   ```

4. Repeat the query filtering data by the `a` column:

   ```
   SELECT count() FROM table_with_primary_key WHERE a = 555;
   ```

   In this case, ADQM has processes 172 thousand rows instead of 100 million and completed the query much faster:

   ```
   ┌─count()─┐
   │  100162 │
   └─────────┘

   1 row in set. Elapsed: 0.004 sec. Processed 172.03 thousand rows, 1.38 MB (43.90 million rows/s., 351.17 MB/s.)
   ```

   If you apply the EXPLAIN clause with the `indexes` parameter to the query, you can see that only 21 granules out of 12209 were selected as possibly containing rows with the `555` value in the `a` column during the query execution:

   ```
   EXPLAIN indexes = 1 SELECT count() FROM table_with_primary_key WHERE a = 555;
   ```

   ```
   ┌─explain────────────────────────────────────────────────┐
   │ Expression ((Projection + Before ORDER BY))            │
   │   Aggregating                                          │
   │     Expression (Before GROUP BY)                       │
   │       Filter (WHERE)                                   │
   │         ReadFromMergeTree (default.table_with_primary_key) │
   │         Indexes:                                       │
   │           PrimaryKey                                   │
   │             Keys:                                      │
   │               a                                        │
   │             Condition: (a in [555, 555])               │
   │             Parts: 8/8                                 │
   │             Granules: 21/12209                         │
   └────────────────────────────────────────────────────────┘
   ```

Compound key of a primary index

1. Create a table with a compound primary key:

   ```
   CREATE TABLE table_compound_primary_key (a Int64, b String) ENGINE = MergeTree ORDER BY (a, b);
   ```

   Fill in the table with data:

   ```
   INSERT INTO table_compound_primary_key
   SELECT toInt64(randUniform(0, 1000)), ['a', 'b', 'c', 'd', 'e'][(rand() % 5) + 1]
   FROM numbers(100000000);
   ```

   Note that table columns differ greatly in cardinality — the first column is filled with integers from the `[0, 1000)` range, while the second one contains only five possible string values.

2. Check the performance of some queries filtering by the first column and by the second column of the primary key:

   Filter by the a column

   ```
   SELECT count()
   FROM table_compound_primary_key
   WHERE a = 555;
   ```

   ```
   ┌─count()─┐
   │  100162 │
   └─────────┘

   1 row in set. Elapsed: 0.004 sec.
   Processed 155.65 thousand rows, 1.25 MB
   (38.72 million rows/s., 309.78 MB/s.)
   ```

   Filter by the b column

   ```
   SELECT count()
   FROM table_compound_primary_key
   WHERE b = 'c';
   ```

   ```
   ┌──count()─┐
   │ 19997249 │
   └──────────┘

   1 row in set. Elapsed: 1.050 sec.
   Processed 44.17 million rows, 441.66 MB
   (42.07 million rows/s., 420.70 MB/s.)
   ```

3. Swap columns in the primary key ordering them from low to high cardinality, according to the recommendations:

   ```
   CREATE TABLE table_compound_primary_key1 (a Int64, b String) ENGINE = MergeTree ORDER BY (b, a);
   ```

   ```
   INSERT INTO table_compound_primary_key1 SELECT * from table_compound_primary_key;
   ```

4. See how the performance of the same queries with filters by each of key columns has changed. The query filtering by the `b` column now runs two times faster, while the execution time of the query filtering by the `a` column remains the same:

   Filter by the a column

   ```
   SELECT count()
   FROM table_compound_primary_key1
   WHERE a = 555;
   ```

   ```
   ┌─count()─┐
   │  100162 │
   └─────────┘

   1 row in set. Elapsed: 0.004 sec.
   Processed 540.67 thousand rows, 4.33 MB
   (123.51 million rows/s., 988.06 MB/s.)
   ```

   Filter by the b column

   ```
   SELECT count()
   FROM table_compound_primary_key1
   WHERE b = 'c';
   ```

   ```
   ┌──count()─┐
   │ 19997249 │
   └──────────┘

   1 row in set. Elapsed: 0.566 sec.
   Processed 20.01 million rows, 200.13 MB
   (35.38 million rows/s., 353.83 MB/s.)
   ```

Data skipping index (Set)

1. Create a test table with 10000 rows in each granule and add 100 million data rows to it:

   ```
   CREATE TABLE table_skip_index (a UInt64, b UInt64)
   ENGINE MergeTree
   PRIMARY KEY a
   SETTINGS index_granularity=10000;
   ```

   ```
   INSERT INTO table_skip_index SELECT number, intDiv(number,5000) FROM numbers(100000000);
   ```

2. When running a query with a filter by the `b` column, which is not a part of the primary key, all 100 million records are scanned:

   ```
   SELECT * FROM table_skip_index WHERE b=100 OR b=555;
   ```

   ```
   ┌──────a─┬───b─┐
   │ 500000 │ 100 │
   │ 500001 │ 100 │
   │ 500002 │ 100 │
   │    ... │ ... │
   └────────┴─────┘
   10000 rows in set. Elapsed: 0.042 sec. Processed 100.00 million rows, 800.12 MB (2.38 billion rows/s., 19.01 GB/s.)
   ```

3. Add a data skipping index:

   ```
   ALTER TABLE table_skip_index ADD INDEX my_index b TYPE set(100) GRANULARITY 2;
   ```

   At this point, the skipping index will only be applied to newly inserted data. To index data that already exists in the table, run the following query:

   ```
   ALTER TABLE table_skip_index MATERIALIZE INDEX my_index;
   ```

4. Re-execute the query that selects data with the same filter:

   ```
   SELECT * FROM table_skip_index WHERE b=100 OR b=555;
   ```

   Instead of processing 100 million rows (800 MB), ADQM reads and processes only 40 thousand rows (440 KB) — four granules of 10000 rows each one.

   ```
   ┌──────a─┬───b─┐
   │ 500000 │ 100 │
   │ 500001 │ 100 │
   │ 500002 │ 100 │
   │    ... │ ... │
   └────────┴─────┘
   10000 rows in set. Elapsed: 0.018 sec. Processed 40.00 thousand rows, 440.00 KB (2.19 million rows/s., 24.06 MB/s.)
   ```

Strict data types

To improve query performance and minimize your storage, it is important to analyze data to be stored in a table and assign each column the most appropriate type. For example, if a column with string values has low cardinality (contains a limited set of repeated values), you can assign the LowCardinality type to this column instead of String for the column storage optimization.

Example

1. Create a table with two columns of the Int64 and String types:

   ```
   CREATE TABLE table_types (a Int64, b String) ENGINE = MergeTree ORDER BY a;
   ```

2. Insert 100 million rows of test data into the table (the first column stores random integers from the `[0,1000)` range, each row of the second column stores one of five predefined strings):

   ```
   INSERT INTO table_types
   SELECT toInt64(randUniform(0, 1000)), ['a', 'b', 'c', 'd', 'e'][(rand() % 5) + 1]
   FROM numbers(100000000);
   ```

3. Check the size of uncompressed data:

   ```
   SELECT
       name,
       type,
       formatReadableSize(data_uncompressed_bytes) AS uncompressed_size
   FROM system.columns
   WHERE table = 'table_types';
   ```

   ```
   ┌─name─┬─type───┬─uncompressed_size─┐
   │ a    │ Int64  │ 759.91 MiB        │
   │ b    │ String │ 189.98 MiB        │
   └──────┴────────┴───────────────────┘
   ```

4. Knowing the range of values in the `a` column, you can define a more appropriate type for integer data in the table schema — UInt16 instead of Int64 (see value ranges that different data types support in the UInt* article). The `b` column storing strings can also be optimized if you change the String type to LowCardinality. Create a table similar to the table created earlier, but with strict data types for columns:

   ```
   CREATE TABLE table_strict_types (a UInt16, b LowCardinality(String)) ENGINE = MergeTree ORDER BY a;
   ```

   ```
   INSERT INTO table_strict_types SELECT a, toLowCardinality(b) from table_types;
   ```

5. Ensure that the size of columns has decreased:

   ```
   SELECT
       name,
       type,
       formatReadableSize(data_uncompressed_bytes) AS uncompressed_size
   FROM system.columns
   WHERE table = 'table_strict_types';
   ```

   ```
   ┌─name─┬─type───────────────────┬─uncompressed_size─┐
   │ a    │ UInt16                 │ 189.98 MiB        │
   │ b    │ LowCardinality(String) │ 95.32 MiB         │
   └──────┴────────────────────────┴───────────────────┘
   ```

6. Compare how the same test query runs when selecting data from the initial `table_types` table and the `table_strict_types` table with optimal column types:

   ```
   SELECT count() FROM table_types WHERE b = 'a';
   ```

   ```
   ┌──count()─┐
   │ 20003609 │
   └──────────┘

   1 row in set. Elapsed: 5.330 sec. Processed 100.00 million rows, 1.00 GB (18.76 million rows/s., 187.61 MB/s.)
   ```

   ```
   SELECT count() FROM table_strict_types WHERE b = 'a';
   ```

   ```
   ┌──count()─┐
   │ 20003609 │
   └──────────┘

   1 row in set. Elapsed: 0.390 sec. Processed 100.00 million rows, 100.12 MB (256.40 million rows/s., 256.71 MB/s.)
   ```

   The output shows that both queries scan the same number of rows, but the query accessing the first table reads almost 10 times more data.

Compression and encoding

When a query runs, it needs to read data from disk — the more data it reads, the slower it is. You can reduce the amount of data to be read using compression and encryption. ADQM supports general-purpose compression codecs that optimize the balance between disk space and CPU consumption, a set of specialized compression codecs for specific data types, and encryption codecs. For a complete list of available compression and encryption codecs, see the Column Compression Codecs section of the ClickHouse documentation.

By default, ADQM applies the compression method defined in the server configuration to columns. You can also specify the compression method for each individual column in the `CREATE TABLE` query using the `CODEC` clause:

```
CREATE TABLE <table_name>
(   <column_name1> <data_type1> CODEC(<compression_codec1>) [...],
    <column_name2> <data_type2> CODEC(<compression_codec2>) [...],
    ...)
ENGINE = MergeTree
...;
```

Codecs can be combined sequentially (for example, `CODEC(Delta, Default)`).

Example

1. Create a test table with three columns (the first and second columns are compressed with the LZ4 and ZSTD algorithms respectively, the third one stores data without compression):

   ```
   CREATE TABLE table_sizes (a Int32 CODEC(LZ4), b Int32 CODEC(ZSTD), c Int32 CODEC(NONE))
   ENGINE = MergeTree
   ORDER BY a;
   ```

2. Insert 100 million rows with the same values (for example, `1`) in all columns:

   ```
   INSERT INTO table_sizes SELECT 1, 1, 1 FROM numbers(100000000);
   ```

3. In the `system.columns` table, you can see the difference between columns in the size of stored data. The first two columns (compressed with different codecs) store 1.7 MB and 279 KB of data respectively, and the size of the uncompressed column is about 380 MB:

   ```
   SELECT
       name,
       formatReadableSize(data_uncompressed_bytes) AS uncompressed_size,
       formatReadableSize(data_compressed_bytes) AS compressed_size,
       round(data_uncompressed_bytes / data_compressed_bytes, 2) AS ratio
   FROM system.columns
   WHERE table = 'table_sizes';
   ```

   ```
   ┌─name─┬─uncompressed_size─┬─compressed_size─┬───ratio─┐
   │ a    │ 379.95 MiB        │ 1.71 MiB        │  222.15 │
   │ b    │ 379.95 MiB        │ 279.06 KiB      │ 1394.21 │
   │ c    │ 379.95 MiB        │ 380.10 MiB      │       1 │
   └──────┴───────────────────┴─────────────────┴─────────┘
   ```

4. Run a query that calculates the sum by each column to see the difference between compressed and uncompressed columns:

   ```
   SELECT sum(a) FROM table_sizes;
   ```

   ```
   ┌────sum(a)─┐
   │ 100000000 │
   └───────────┘

   1 row in set. Elapsed: 0.197 sec. Processed 100.00 million rows, 400.00 MB (507.58 million rows/s., 2.03 GB/s.)
   ```

   ```
   SELECT sum(b) FROM table_sizes;
   ```

   ```
   ┌────sum(b)─┐
   │ 100000000 │
   └───────────┘

   1 row in set. Elapsed: 0.091 sec. Processed 100.00 million rows, 400.00 MB (1.09 billion rows/s., 4.38 GB/s.)
   ```

   ```
   SELECT sum(c) FROM table_sizes;
   ```

   ```
   ┌────sum(c)─┐
   │ 100000000 │
   └───────────┘

   1 row in set. Elapsed: 0.389 sec. Processed 100.00 million rows, 400.00 MB (257.00 million rows/s., 1.03 GB/s.)
   ```

   Each of three queries scans 100 million rows, but scanning compressed data is faster: 0.197 and 0.091 seconds vs 0.389 seconds for uncompressed data.

Query cache

Another way to reduce the execution time of `SELECT` queries, as well as reduce the load on the ADQM server and optimize resource consumption is to cache queries. ADQM can compute a query just once (when executing it for the first time), save the result to the **query cache**, and then read this result from the cache when executing the same query again. If a query fails due to an error or user cancellation, no entry is added to the query cache.

Manage query caching

1. Enable the query cache support in ADQM at the session level:

   ```
   SET allow_experimental_query_cache = 1;
   ```

2. Set the `use_query_cache` setting value to `1` to cache the result of a specific query or all queries of the current session:

   - `SELECT … SETTINGS use_query_cache = 1;` — caching an individual query;
   - `SET use_query_cache = 1;` — caching queries at the session level.

   Subsequent executions of the same query (also with the `use_query_cache = 1` parameter) will read the previously computed result from the cache and return it immediately.

   To set up caching in more detail, you can also use the following parameters (both values are `1` by default):

   - `enable_writes_to_query_cache` — specifies whether results of `SELECT` queries should be stored in the query cache;
   - `enable_reads_from_query_cache` — specifies whether results of `SELECT` queries should be retrieved from the query cache.

> NOTE
>
> The `use_query_cache` setting and other options related to the query cache affect only standalone `SELECT` queries.

To clear the query cache, use the `SYSTEM DROP QUERY CACHE` command.

Configure query cache

To provide more accurate control over the query cache behavior, ADQM supports a set of configuration options.

Server-level setting

The size of the query cache (in bytes), the maximum number of cache entries, and the maximum size of individual cache entries (in bytes and in records) can be configured using the corresponding parameters in the `query_cache` section of the ADQM server's configuration file (*config.xml*).

```
<query_cache>
    <max_size>1073741824</max_size>
    <max_entries>1024</max_entries>
    <max_entry_size>1048576</max_entry_size>
    <max_entry_rows>30000000</max_entry_rows>
</query_cache>
```

Settings per query or per session

- To cache only the most frequent or time-consuming queries, you can configure the following settings:
  - `query_cache_min_query_runs` — specifies how many times (minimum) a `SELECT` query must be executed before its result is cached;
  - `query_cache_min_query_duration` — specifies how long a query should run at least for its result to be cached.
- Entries in the query cache expire after a certain period of time (`60` seconds, by default). To change this period, use the `query_cache_ttl` parameter.
- Entries in the query cache are not shared between users by default for security reasons. However, if necessary, you can mark cache entries as available to other users (i.e. as shared) by setting the `query_cache_share_between_users` parameter to `1`.
- Results of queries with non-deterministic functions (for example, `rand()`, `now()`) are not cached by default. To change this, use the `query_cache_store_results_of_queries_with_nondeterministic_functions` option.

View information on query cache

You can get information about the content of the query cache from the system.query_cache system table.

```
SELECT * FROM system.query_cache;
```

```
┌─query──────────────────────────────────────────────────┬────────────key_hash─┬──────────expires_at─┬─stale─┬─shared─┬─result_s
ize─┐
│ SELECT count() FROM table_types WHERE b = 'a' SETTINGS  │ 3145443392323842376 │ 2023-10-18 12:58:43 │     1 │      0 │
256 │
└────────────────────────────────────────────────────────┴─────────────────────┴─────────────────────┴───────┴────────┴─────────
────┘
```

In the example above, the cache entry is marked as stale because the time elapsed since the query result was cached exceeds the lifetime of the cache entry. This means that the next time the query is run, the result of the cached query will not be used, but the cache entry will be updated instead.

The `SETTINGS` expression in the `query` field is shown partially because all parameters associated with the query cache are internally cleared before the query is used as a key for the query cache.

The number of query cache hits and misses since the server startup is counted as the `QueryCacheHits` and `QueryCacheMisses` events in the `system.events` system table. Both counters are updated only for `SELECT` queries that are executed with the `use_query_cache = 1` setting.

```
SELECT * FROM system.events WHERE (event = 'QueryCacheHits' OR event = 'QueryCacheMisses');
```

```
┌─event────────────┬─value─┬─description──────────────────────────────────────────────────────────────────────────────
───┐
│ QueryCacheHits   │     2 │ Number of times a query result has been found in the query cache (and query computation was
avoided).  │
│ QueryCacheMisses │     4 │ Number of times a query result has not been found in the query cache (and required query
computation). │
└──────────────────┴───────┴──────────────────────────────────────────────────────────────────────────────────────────
───┘
```

Example

1. Run the `SELECT` query accessing the non-optimized `table_types` table created above:

   ```
   SELECT count() FROM table_types WHERE b = 'a';
   ```

   The query takes about 2.5 seconds to complete:

   ```
   ┌──count()─┐
   │ 19999789 │
   └──────────┘

   1 row in set. Elapsed: 2.584 sec. Processed 100.00 million rows, 1.00 GB (38.70 million rows/s., 387.00 MB/s.)
   ```

2. Turn on the query cache feature and run the query again with the `use_query_cache` setting enabled:

   ```
   SET allow_experimental_query_cache = 1;
   ```

   ```
   SELECT count() FROM table_types WHERE b = 'a' SETTINGS use_query_cache = 1;
   ```

   ```
   ┌──count()─┐
   │ 19999789 │
   └──────────┘

   1 row in set. Elapsed: 2.924 sec. Processed 100.00 million rows, 1.00 GB (34.20 million rows/s., 341.98 MB/s.)
   ```

   As you can see, the query took about the same amount of time to complete (even a little more), since the first time the query runs with the `use_query_cache = 1` setting the result of its execution will be written to the cache. Subsequent executions of the same query (also with the `use_query_cache = 1` setting) within the time interval specified by `query_cache_ttl` will read the result from the cache and return it immediately:

   ```
   SELECT count() FROM table_types WHERE b = 'a' SETTINGS use_query_cache = 1;
   ```

   ```
   ┌──count()─┐
   │ 19999789 │
   └──────────┘

   1 row in set. Elapsed: 0.003 sec.
   ```

PREWHERE

`PREWHERE` is a way to optimize `SELECT … WHERE` queries. It allows filtering data before the read operation, so the amount of data to be read from the table is reduced, and the query runs faster. In ADQM, `PREWHERE` filtering optimization is applied automatically (with the `optimize_move_to_prewhere` setting enabled by default) — even if the `PREWHERE` clause is not explicitly specified in the query, a part of filtering expressions is moved from `WHERE` to `PREWHERE` to be checked first.

You can explicitly specify the `PREWHERE` clause in the query to manually adjust the filtering optimization. First, columns necessary for checking the `PREWHERE` expression are read from the queried table. Then other columns needed to complete the rest of the query are read, but only those blocks where the `PREWHERE` expression is true at least for some rows. If there are a lot of blocks where the `PREWHERE` condition is not satisfied for all rows, and the `PREWHERE` expression uses fewer columns than other parts of the query, `PREWHERE` optimization can significantly reduce the amount of data read from disk for query execution.

You can use both the `PREWHERE` and `WHERE` clauses in the `SELECT` query simultaneously (`PREWHERE` is executed before `WHERE` in this case).

If a query has the `FINAL` modifier, the `PREWHERE` optimization is not always correct. It is enabled only if both the `optimize_move_to_prewhere` and `optimize_move_to_prewhere_if_final` settings are turned on.

Example

1. Create a test table with three columns and add 10 million rows to it (the first column contains integers in order, the second column — random numbers from the `[0, 1000)` range, and the third one — the `test string` value in 100 rows, and values for the remaining rows are converted from the corresponding integer identifiers in the first column):

   ```
   CREATE TABLE table_prewhere (id UInt64, int_value Int64, string_value String) ENGINE = MergeTree ORDER BY id;
   ```

   ```
   INSERT INTO table_prewhere SELECT
       number,
       toInt64(randUniform(0, 1000)),
       if(number between 11 and 110, 'test string', toString(number))
   FROM numbers(10000000);
   ```

2. Run a query that filters data by columns that are not included into the primary key:

   ```
   SELECT count()
   FROM table_prewhere
   WHERE (int_value > 550) AND (string_value = 'test string');
   ```

   ```
   ┌─count()─┐
   │      43 │
   └─────────┘
   1 row in set. Elapsed: 0.638 sec. Processed 10.00 million rows, 238.89 MB (15.68 million rows/s., 374.58 MB/s.)
   ```

3. If you enable tracing with the `SET send_logs_level='debug'` command before executing the query, you can see that ADQM has automatically moved the first filtering condition (`int_value > 550`) to `PREWHERE`:

   ```
   InterpreterSelectQuery: MergeTreeWhereOptimizer: condition "int_value > 550" moved to PREWHERE
   ```

   Thus, the query first receives a large set of rows based on the `int_value > 550` condition, and then filters out the most of these rows according to the `string_value = 'test string'` filter. To optimize the query, change the order in which filter conditions are checked — pass the filter by the `string_value` column to `PREWHERE` explicitly:

   ```
   SELECT count()
   FROM table_prewhere
   PREWHERE string_value = 'test string'
   WHERE int_value > 550;
   ```

   Now the query first receives a small set of rows that meet the `string_value = 'test string'` condition, which is then checked against the second `int_value > 550` filter. As a result, less data is processed (158.89 MB instead of 238.89 MB) and the query runs faster:

   ```
   ┌─count()─┐
   │      43 │
   └─────────┘
   1 row in set. Elapsed: 0.249 sec. Processed 10.00 million rows, 158.89 MB (40.12 million rows/s., 637.45 MB/s.)
   ```

Optimize JOIN operations

JOIN operations in ADQM can be optimized in several ways to improve query performance. Here are some recommendations:

- ADQM supports multiple join algorithms — choose an algorithm that is the most optimal for your query based on the join type/strictness, and the engine of tables being joined. To set the join algorithm, use the join_algorithm parameter. You can specify one or multiple algorithms (one of the available algorithms will be selected for each specific query based on its type/strictness and table engine) or allow the query planner to select and dynamically change the algorithm during query execution depending on resource availability. See the Choosing the Right Join Algorithm blog post by ClickHouse for the comparison of join algorithms and recommendations on how to choose the join algorithm that fits best into your scenario.
- Use appropriate join keys (columns used to match rows between two tables). Ideally, join keys should be indexed, and they should have low cardinality. The data type of a key can also affect performance (for example, using a string join key may be slower than using an integer key).
- In a query with the `JOIN` clause, there is no optimization of the order in which the join operation is executed relative to other steps of the query. The join operation (lookup in the right table) is performed before `WHERE` filtering and before aggregation. To explicitly specify the order of calculations and improve query performance, it is recommended to apply a filter as early as possible — join subqueries.
- Results of joins are not cached — each time a query with the same `JOIN` is executed, the join subquery is executed again. To avoid this, you can use the special Join table engine, which is a prepared data structure for usage in join operations.
- For small amounts of data (up to ~200 GB in compressed form), it is recommended to use memory no less than the amount of data.
- Sometimes it is more efficient to use `IN` or dictionaries instead of `JOIN`. It is not recommended to use `JOIN` with dictionaries as dictGet functions work more efficiently.
- In some scenarios, it is more effective to denormalize data by combining multiple tables into one to completely eliminate the use of join operations and improve query performance.

Analyze queries

Elena Dvoryadkina

Contents

To Table of Contents

- EXPLAIN AST
  - Example
  - Settings
- EXPLAIN SYNTAX
  - Example
- EXPLAIN QUERY TREE
  - Example
  - Settings
- EXPLAIN PLAN
  - Example
  - Settings
- EXPLAIN PIPELINE
  - Example
  - Settings
- EXPLAIN ESTIMATE
  - Example
- EXPLAIN TABLE OVERRIDE
  - Example

To analyze how ADQM transforms and executes queries, use the `EXPLAIN` clause. Basic syntax:

```
EXPLAIN [AST | SYNTAX | QUERY TREE | PLAN | PIPELINE | ESTIMATE | TABLE OVERRIDE] [<setting_name> = <setting_value>, ...]
<query_to_analyze>
[FORMAT <output_format>]
```

where:

- `EXPLAIN [<explain_type>] [<explain_settings>]` — type of the `EXPLAIN` analyzer (see the table below) and its additional settings;
- `<query_to_analyze>` — query to be inspected;
- `FORMAT <output_format>` — output format for analysis result.

EXPLAIN analyzer types

EXPLAIN type	EXPLAIN result
AST	Abstract syntax tree of the query
SYNTAX	Optimized version of the query syntax with which ADQM will replace the original text of the query when executing it
QUERY TREE	Query tree after optimization
PLAN	Query execution plan. This `EXPLAIN` type is used by default
PIPELINE	Query execution pipeline
ESTIMATE	Estimated number of rows, marks, and data parts that ADQM should process when executing the query
TABLE OVERRIDE	Pre-validation of table schema overrides to be specified for a MaterializedMySQL database

EXPLAIN AST

The `EXPLAIN AST` query returns an abstract syntax tree (AST) of the analyzed query. This analyzer supports all types of queries, not only `SELECT`.

Example

```
EXPLAIN AST SELECT sum(number) AS test_sums FROM numbers(5) GROUP BY number % 2;
```

```
┌─explain─────────────────────────────────────┐
│ SelectWithUnionQuery (children 1)           │
│  ExpressionList (children 1)                │
│   SelectQuery (children 3)                  │
│    ExpressionList (children 1)              │
│     Function sum (alias test_sums) (children 1) │
│      ExpressionList (children 1)            │
│       Identifier number                     │
│    TablesInSelectQuery (children 1)         │
│     TablesInSelectQueryElement (children 1) │
│      TableExpression (children 1)           │
│       Function numbers (children 1)         │
│        ExpressionList (children 1)          │
│         Literal UInt64_5                    │
│    ExpressionList (children 1)              │
│     Function modulo (children 1)            │
│      ExpressionList (children 2)            │
│       Identifier number                     │
│       Literal UInt64_2                      │
└─────────────────────────────────────────────┘
```

Settings

Enable the `graph` option for an `EXPLAIN AST` query to get an abstract syntax tree of the query as a graph described in the DOT graph description language. Then you can load this graph description into a special application that is able to draw it as a diagram (for example, use Graphviz — open-source software for graph visualization).

Below is an example of a query abstract syntax tree presented as a graph:

```
EXPLAIN AST graph=1 SELECT sum(number) AS test_sums FROM numbers(5) GROUP BY number % 2;
```

```
┌─explain──────────────────────────────────────────────────────────────────┐
│ digraph {                                                                │
│     rankdir="UD";                                                        │
│     n140208872268568[label="SelectWithUnionQuery (children 1)"];         │
│     n140208877065784[label="ExpressionList (children 1)"];               │
│     n140205632168472[label="SelectQuery (children 3)"];                  │
│     n140206404082776[label="ExpressionList (children 1)"];               │
│     n140206025898776[label="Function sum (alias test_sums) (children 1)"]; │
│     n140206026032952[label="ExpressionList (children 1)"];               │
│     n140205801742360[label="Identifier number"];                         │
│     ...                                                                 │
│ }                                                                        │
└──────────────────────────────────────────────────────────────────────────┘
```

EXPLAIN SYNTAX

When ADQM executes queries, it can automatically rewrite their syntax to improve performance. The `EXPLAIN SYNTAX` analyzer allows you to view syntax optimizations that ADQM applies — then you can use them to manually optimize complex queries for faster execution.

Example

1. Create a MergeTree table with 1 million rows and specify the partition key so that data is divided into 10 partitions:

```
CREATE TABLE test_table (value UInt64)
ENGINE = MergeTree
PARTITION BY value % 10
ORDER BY value AS
SELECT number
FROM numbers(1000000);
```

2. Run a query that calculates the arithmetic mean in the `value` column for the first partition:

```
SELECT avg(value)
FROM (SELECT value FROM test_table)
WHERE (value % 10) = 1;
```

As you can see in the query result, ADQM processes only 100 thousand rows (not a million):

```
┌─avg(value)─┐
│     499996 │
└────────────┘

1 row in set. Elapsed: 0.006 sec. Processed 100.00 thousand rows, 800.00 KB (16.40 million rows/s., 131.16 MB/s.)
```

3. Use `EXPLAIN SYNTAX` to analyze the query above:

```
EXPLAIN SYNTAX
SELECT avg(value)
FROM (SELECT value FROM test_table)
WHERE (value % 10) = 1;
```

4. The analysis shows that ADQM automatically optimizes the query by moving the `WHERE` filter inside the subquery — so it is enough to process only 1/10 part of the table data:

```
┌─explain──────────────────────┐
│ SELECT avg(value)            │
│ FROM                         │
│ (                            │
│     SELECT value             │
│     FROM test_table          │
│     WHERE (value % 10) = 1   │
│ )                            │
│ WHERE (value % 10) = 1       │
└──────────────────────────────┘
```

Thus, you can use this optimization in more complex queries and manually specify the necessary filters in subqueries.

EXPLAIN QUERY TREE

The `EXPLAIN QUERY TREE` analyzer dumps the query tree. This analyzer type is currently experimental. To use it, enable the option: `SET allow_experimental_analyzer = 1`.

Create a test table to run examples below

Examples of `EXPLAIN QUERY TREE` queries below use a test table that stores randomly generated pageview data. To reproduce these examples, prepare the `page_views` table:

```
CREATE TABLE page_views (time DateTime, page_name String, time_on_page UInt64)
ENGINE = MergeTree
ORDER BY time;
```

```
INSERT INTO page_views
SELECT
    toDate('2023-01-01 00:00:00') + (rand() % 182) AS time,
    ['landing', 'products', 'docs', 'about'][toInt32(randBinomial(4, 0.01)) + 1],
    rand() % 15
FROM numbers(1000000);
```

Example

```
SET allow_experimental_analyzer = 1;
```

```
EXPLAIN QUERY TREE
SELECT page_name, toDate(toStartOfMonth(time)) AS month, count() AS page_view_count
FROM page_views
WHERE page_name = 'landing'
GROUP BY page_name, month
ORDER BY page_name, month
LIMIT 3;
```

```
┌─explain──────────────────────────────────────────────────────────────────────────────────────────┐
│ QUERY id: 0                                                                                      │
│   PROJECTION COLUMNS                                                                             │
│     page_name String                                                                             │
│     month Date                                                                                   │
│     page_view_count UInt64                                                                       │
│   PROJECTION                                                                                     │
│     LIST id: 1, nodes: 3                                                                         │
│       COLUMN id: 2, column_name: page_name, result_type: String, source_id: 3                    │
│       FUNCTION id: 4, function_name: toDate, function_type: ordinary, result_type: Date          │
│         ARGUMENTS                                                                                │
│           LIST id: 5, nodes: 1                                                                   │
│             FUNCTION id: 6, function_name: toStartOfMonth, function_type: ordinary, result_type: Date │
│               ARGUMENTS                                                                          │
│                 LIST id: 7, nodes: 1                                                             │
│                   COLUMN id: 8, column_name: time, result_type: DateTime, source_id: 3           │
│       FUNCTION id: 9, function_name: count, function_type: aggregate, result_type: UInt64        │
│   JOIN TREE                                                                                      │
│     TABLE id: 3, table_name: default.page_views                                                  │
│   WHERE                                                                                          │
│     FUNCTION id: 10, function_name: equals, function_type: ordinary, result_type: UInt8          │
│       ARGUMENTS                                                                                  │
│         LIST id: 11, nodes: 2                                                                    │
│           COLUMN id: 2, column_name: page_name, result_type: String, source_id: 3                │
│           CONSTANT id: 12, constant_value: 'landing', constant_value_type: String                │
│   GROUP BY                                                                                       │
│     LIST id: 13, nodes: 2                                                                        │
│       COLUMN id: 2, column_name: page_name, result_type: String, source_id: 3                    │
│       FUNCTION id: 4, function_name: toDate, function_type: ordinary, result_type: Date          │
│         ARGUMENTS                                                                                │
│           LIST id: 5, nodes: 1                                                                   │
│             FUNCTION id: 6, function_name: toStartOfMonth, function_type: ordinary, result_type: Date │
│               ARGUMENTS                                                                          │
│                 LIST id: 7, nodes: 1                                                             │
│                   COLUMN id: 8, column_name: time, result_type: DateTime, source_id: 3           │
│   ORDER BY                                                                                       │
│     LIST id: 14, nodes: 2                                                                        │
│       SORT id: 15, sort_direction: ASCENDING, with_fill: 0                                       │
│         EXPRESSION                                                                               │
│           COLUMN id: 2, column_name: page_name, result_type: String, source_id: 3                │
│       SORT id: 16, sort_direction: ASCENDING, with_fill: 0                                       │
│         EXPRESSION                                                                               │
│           FUNCTION id: 4, function_name: toDate, function_type: ordinary, result_type: Date      │
│             ARGUMENTS                                                                            │
│               LIST id: 5, nodes: 1                                                               │
│                 FUNCTION id: 6, function_name: toStartOfMonth, function_type: ordinary, result_type: Date │
│                   ARGUMENTS                                                                      │
│                     LIST id: 7, nodes: 1                                                         │
│                       COLUMN id: 8, column_name: time, result_type: DateTime, source_id: 3       │
│   LIMIT                                                                                          │
│     CONSTANT id: 17, constant_value: UInt64_3, constant_value_type: UInt64                       │
└──────────────────────────────────────────────────────────────────────────────────────────────────┘
```

> **TIP**
>
> To speed up the execution of queries that calculate aggregates (for example, like the query above calculates the number of page views by month), you can use materialized views that automatically calculate intermediate data aggregate states — see an example in the Use materialized views article.

Settings

To customize the `EXPLAIN QUERY TREE` output, use the following parameters (set a parameter value to `1` or `0` to enable or disable the corresponding setting respectively):

- `run_passes` — runs all query tree passes before dumping the query tree (the default is `1`);
- `passes` — specifies how many passes to run (the default is `-1` that runs all the passes);
- `dump_passes` — dumps information about used passes before dumping the query tree (the default is `0`).

Example

The following analysis runs three query tree passes and outputs information about them before dumping the query tree:

```
EXPLAIN QUERY TREE passes=3, dump_passes=1
SELECT page_name, toDate(toStartOfMonth(time)) AS month, count() AS page_view_count
FROM page_views
WHERE page_name = 'landing'
GROUP BY page_name, month
ORDER BY page_name, month
LIMIT 3
FORMAT TSVRaw;
```

```
Pass 1 QueryAnalysis - Resolve type for each query expression. Replace identifiers, matchers with query expressions. Perform
constant folding. Evaluate scalar subqueries.
Pass 2 FunctionToSubcolumns - Rewrite function to subcolumns, for example tupleElement(column, subcolumn) into column.
subcolumn
Pass 3 CountDistinct - Optimize single countDistinct into count over subquery
QUERY id: 0
  PROJECTION COLUMNS
    page_name String
    month Date
    page_view_count UInt64
  PROJECTION
    LIST id: 1, nodes: 3
      COLUMN id: 2, column_name: page_name, result_type: String, source_id: 3
      FUNCTION id: 4, function_name: toDate, function_type: ordinary, result_type: Date
        ARGUMENTS
          LIST id: 5, nodes: 1
            FUNCTION id: 6, function_name: toStartOfMonth, function_type: ordinary, result_type: Date
              ARGUMENTS
                LIST id: 7, nodes: 1
                  COLUMN id: 8, column_name: time, result_type: DateTime, source_id: 3
      FUNCTION id: 9, function_name: count, function_type: aggregate, result_type: UInt64
  JOIN TREE
    TABLE id: 3, table_name: default.page_views
  WHERE
    FUNCTION id: 10, function_name: equals, function_type: ordinary, result_type: UInt8
      ARGUMENTS
        LIST id: 11, nodes: 2
          COLUMN id: 2, column_name: page_name, result_type: String, source_id: 3
          CONSTANT id: 12, constant_value: 'landing', constant_value_type: String
  GROUP BY
    LIST id: 13, nodes: 2
      COLUMN id: 2, column_name: page_name, result_type: String, source_id: 3
      FUNCTION id: 4, function_name: toDate, function_type: ordinary, result_type: Date
        ARGUMENTS
          LIST id: 5, nodes: 1
            FUNCTION id: 6, function_name: toStartOfMonth, function_type: ordinary, result_type: Date
              ARGUMENTS
                LIST id: 7, nodes: 1
                  COLUMN id: 8, column_name: time, result_type: DateTime, source_id: 3
  ORDER BY
    LIST id: 14, nodes: 2
      SORT id: 15, sort_direction: ASCENDING, with_fill: 0
        EXPRESSION
          COLUMN id: 2, column_name: page_name, result_type: String, source_id: 3
      SORT id: 16, sort_direction: ASCENDING, with_fill: 0
        EXPRESSION
          FUNCTION id: 4, function_name: toDate, function_type: ordinary, result_type: Date
            ARGUMENTS
              LIST id: 5, nodes: 1
                FUNCTION id: 6, function_name: toStartOfMonth, function_type: ordinary, result_type: Date
                  ARGUMENTS
                    LIST id: 7, nodes: 1
                      COLUMN id: 8, column_name: time, result_type: DateTime, source_id: 3
  LIMIT
    CONSTANT id: 17, constant_value: UInt64_3, constant_value_type: UInt64
```

EXPLAIN PLAN

The `EXPLAIN PLAN` analyzer displays the query execution plan as a tree that shows in which order clauses from the query are actually executed. Read query execution plan stages from bottom to top. This analyzer type is used by default if another type is not explicitly specified for `EXPLAIN`.

Example

The following analysis shows that when executing a `SELECT` query, ADQM first reads data (`ReadFromStorage`), then filters (`Filter`) and aggregates (`Aggregating`) it.

```
EXPLAIN SELECT sum(number) AS test_sums FROM numbers(10000) WHERE number > 100 GROUP BY number % 2;
```

```
┌─explain─────────────────────────────────┐
│ Expression ((Projection + Before ORDER BY)) │
│   Aggregating                           │
│     Expression (Before GROUP BY)        │
│       Filter (WHERE)                    │
│         ReadFromStorage (SystemNumbers) │
└─────────────────────────────────────────┘
```

Settings

In `EXPLAIN PLAN` queries, you can use the following parameters to specify information the analysis output should include (if parameter value is `1` — the corresponding setting is enabled, if `0` — it is disabled):

- `header` — adds the `Header` key to the query execution plan as an array of columns (column name and type are printed) for which values are calculated at each step.

Example

EXPLAIN header = 0 (default)

```
EXPLAIN
SELECT sum(number) AS test_sums
FROM numbers(5) GROUP BY number % 2;
```

```
┌─explain─────────────────────────────────────┐
│ Expression ((Projection + Before ORDER BY)) │
│   Aggregating                               │
│     Expression (Before GROUP BY)            │
│       ReadFromStorage (SystemNumbers)       │
└─────────────────────────────────────────────┘
```

EXPLAIN header = 1

```
EXPLAIN header = 1
SELECT sum(number) AS test_sums
FROM numbers(5) GROUP BY number % 2;
```

```
┌─explain─────────────────────────────────────┐
│ Expression ((Projection + Before ORDER BY)) │
│ Header: test_sums UInt64                    │
│   Aggregating                               │
│   Header: modulo(number, 2) UInt8           │
│           sum(number) UInt64                │
│     Expression (Before GROUP BY)            │
│     Header: number UInt64                   │
│             modulo(number, 2) UInt8         │
│       ReadFromStorage (SystemNumbers)       │
│       Header: number UInt64                 │
└─────────────────────────────────────────────┘
```

- `description` — prints a description for each step of the query execution plan.

Example

EXPLAIN description = 0

```
EXPLAIN description = 0
SELECT sum(number) AS test_sums
FROM numbers(5) GROUP BY number % 2;
```

```
┌─explain───────────┐
│ Expression        │
│   Aggregating     │
│     Expression    │
│       ReadFromStorage │
└───────────────────┘
```

EXPLAIN description = 1 (default)

```
EXPLAIN
SELECT sum(number) AS test_sums
FROM numbers(5) GROUP BY number % 2;
```

```
┌─explain─────────────────────────────────────┐
│ Expression ((Projection + Before ORDER BY)) │
│   Aggregating                               │
│     Expression (Before GROUP BY)            │
│       ReadFromStorage (SystemNumbers)       │
└─────────────────────────────────────────────┘
```

- `indexes` — adds the `Indexes` key to the query execution plan as an array of used indexes. Each index contains the `Type` key (`MinMax`, `Partition`, `PrimaryKey`, or `Skip`) and optional keys:
  - `Name` — index name (for `Skip` indexes only);
  - `Keys` — array of columns used by the index;
  - `Condition` — used condition;
  - `Description` — index description (for `Skip` indexes only);
  - `Parts` — number of parts filtered by the applied index;
  - `Granules` — number of granules filtered by the applied index.

  This parameter is only supported for analyzing queries that access MergeTree tables.

Example

EXPLAIN actions = 0 (default)

```
EXPLAIN indexes = 0
SELECT sum(value)
FROM test_table
WHERE (value % 10) = 1;
```

```
┌─explain────────────────────────────────────────
┐
│ Expression ((Projection + Before ORDER BY))
│
│   Aggregating
│
│     Expression (Before GROUP BY)
│
│       Filter (WHERE)
│
│         ReadFromMergeTree (default.test_table)
│
└────────────────────────────────────────────────
┘
```

EXPLAIN actions = 1

```
EXPLAIN indexes = 1
SELECT sum(value)
FROM test_table
WHERE (value % 10) = 1;
```

```
┌─explain────────────────────────────────────────
──────────┐
│ Expression ((Projection + Before ORDER BY))
│
│   Aggregating
│
│     Expression (Before GROUP BY)
│
│       Filter (WHERE)
│
│         ReadFromMergeTree (default.test_table)
│
│         Indexes:
│
│           MinMax
│
│             Condition: true
│
│             Parts: 10/10
│
│             Granules: 120/120
│
│           Partition
│
│             Keys:
│
│               moduloLegacy(value, 10)
│
│             Condition: (moduloLegacy(value, 10)
in [1, 1]) │
│             Parts: 1/10
│
│             Granules: 12/120
│
│           PrimaryKey
│
│             Condition: true
│
│             Parts: 1/1
│
│             Granules: 12/12
│
└────────────────────────────────────────────────
──────────┘
```

- `actions` — prints detailed information about the actions performed at each step of the query execution plan (the `Actions` key).

Example

EXPLAIN actions = 0 (default)

```
EXPLAIN
SELECT sum(number) AS test_sums
FROM numbers(5) GROUP BY number % 2;
```

```
┌─explain─────────────────────────────────────┐
│ Expression ((Projection + Before ORDER BY)) │
│   Aggregating                               │
│     Expression (Before GROUP BY)            │
│       ReadFromStorage (SystemNumbers)       │
└─────────────────────────────────────────────┘
```

EXPLAIN actions = 1

```
EXPLAIN actions = 1
SELECT sum(number) AS test_sums
FROM numbers(5) GROUP BY number % 2;
```

```
┌─explain────────────────────────────────────────
──────────────────────────┐
│ Expression ((Projection + Before ORDER BY))
│
│ Actions: INPUT : 0 -> sum(number) UInt64 : 0
│
│          ALIAS sum(number) :: 0 -> test_sums
UInt64 : 1                                │
│ Positions: 1
│
│   Aggregating
│
│   Keys: modulo(number, 2)
│
│   Aggregates:
│
│       sum(number)
│
│         Function: sum(UInt64) → UInt64
│
│         Arguments: number
│
│   Skip merging: 0
│
│     Expression (Before GROUP BY)
│
│     Actions: INPUT : 0 -> number UInt64 : 0
│
│              COLUMN Const(UInt8) -> 2 UInt8 : 1
│
│              FUNCTION modulo(number : 0, 2 ::
1) -> modulo(number, 2) UInt8 : 2 │
│     Positions: 0 2
│
│       ReadFromStorage (SystemNumbers)
│
└────────────────────────────────────────────────
──────────────────────────┘
```

- `json` — outputs the query execution plan in JSON format. Each node is a dictionary, which always contains the `Node Type` (a step name) and `Plans` (an array with child step descriptions) keys. Other optional keys may be added depending on a node type and settings.
  To avoid unnecessary escaping, it is recommended to use the TSVRaw format for analysis result output.

Example

EXPLAIN json = 0 (default)

```
EXPLAIN
SELECT sum(number) AS test_sums
FROM numbers(5) GROUP BY number % 2;
```

```
┌─explain─────────────────────────────────────┐
│ Expression ((Projection + Before ORDER BY)) │
│   Aggregating                               │
│     Expression (Before GROUP BY)            │
│       ReadFromStorage (SystemNumbers)       │
└─────────────────────────────────────────────┘
```

EXPLAIN json = 1

```
EXPLAIN json = 1
SELECT sum(number) AS test_sums
FROM numbers(5) GROUP BY number % 2 FORMAT TSVRaw;
```

```
[
  {
    "Plan": {
      "Node Type": "Expression",
      "Description": "(Projection + Before ORDER
BY)",
      "Plans": [
        {
          "Node Type": "Aggregating",
          "Plans": [
            {
              "Node Type": "Expression",
              "Description": "Before GROUP BY",
              "Plans": [
                {
                  "Node Type": "ReadFromStorage",
                  "Description": "SystemNumbers"
                }
              ]
            }
          ]
        }
      ]
    }
  }
]
```

EXPLAIN PIPELINE

The `EXPLAIN PIPELINE` analyzer displays the query execution pipeline that shows how many parallel threads are used to execute each step of the query plan.

Example

```
EXPLAIN PIPELINE SELECT sum(number) AS test_sums FROM numbers(5) GROUP BY number % 2;
```

```
┌─explain───────────────────┐
│ (Expression)              │
│ ExpressionTransform × 8   │
│   (Aggregating)           │
│   Resize 1 → 8            │
│     AggregatingTransform  │
│       (Expression)        │
│       ExpressionTransform │
│         (ReadFromStorage) │
│         Limit             │
│           Numbers 0 → 1   │
└───────────────────────────┘
```

Settings

To configure the `EXPLAIN PIPELINE` query output, use the following parameters (set the parameter value to `1` or `0` to enable or disable the corresponding setting respectively):

- `header` — prints a header for each output port (the default is `0`).

Example

```
EXPLAIN PIPELINE header = 1
SELECT sum(number) AS test_sums FROM numbers(5) GROUP BY number % 2;
```

```
┌─explain─────────────────────────────────────────────────────────────────────┐
│ (Expression)                                                                │
│ ExpressionTransform × 8                                                     │
│ Header: test_sums UInt64: test_sums UInt64 UInt64(size = 0)                 │
│   (Aggregating)                                                             │
│   Resize 1 → 8                                                              │
│   Header × 8 : modulo(number, 2) UInt8: modulo(number, 2) UInt8 UInt8(size = 0) │
│                sum(number) UInt64: sum(number) UInt64 UInt64(size = 0)      │
│     AggregatingTransform                                                    │
│     Header: modulo(number, 2) UInt8: modulo(number, 2) UInt8 UInt8(size = 0) │
│             sum(number) UInt64: sum(number) UInt64 UInt64(size = 0)         │
│       (Expression)                                                          │
│       ExpressionTransform                                                   │
│       Header: number UInt64: number UInt64 UInt64(size = 0)                 │
│               modulo(number, 2) UInt8: modulo(number, 2) UInt8 UInt8(size = 0) │
│         (ReadFromStorage)                                                   │
│         Limit                                                               │
│         Header: number UInt64: number UInt64 UInt64(size = 0)               │
│           Numbers 0 → 1                                                     │
│           Header: number UInt64: number UInt64 UInt64(size = 0)             │
└─────────────────────────────────────────────────────────────────────────────┘
```

- `graph` — prints the query execution pipeline as a graph described in the DOT graph description language (the default is `0`). Load the provided syntax into an application for graph visualization (for example, Graphviz) to view the visual representation of the graph.

Example

```
EXPLAIN PIPELINE graph = 1
SELECT sum(number) AS test_sums FROM numbers(5) GROUP BY number % 2 FORMAT TSV;
```

```
digraph
{
  rankdir="LR";
  { node [shape = rect]
    subgraph cluster_0 {
      label ="Expression";
      style=filled;
      color=lightgrey;
      node [style=filled,color=white];
      { rank = same;
        n6 [label="ExpressionTransform × 8"];
      }
    }
    subgraph cluster_1 {
      label ="ReadFromStorage";
      style=filled;
      color=lightgrey;
      node [style=filled,color=white];
      { rank = same;
        n2 [label="Limit"];
        n1 [label="Numbers"];
      }
    }
    subgraph cluster_2 {
      label ="Expression";
      style=filled;
      color=lightgrey;
      node [style=filled,color=white];
      { rank = same;
        n3 [label="ExpressionTransform"];
      }
    }
    subgraph cluster_3 {
      label ="Aggregating";
      style=filled;
      color=lightgrey;
      node [style=filled,color=white];
      { rank = same;
        n4 [label="AggregatingTransform"];
        n5 [label="Resize"];
      }
    }
  }
  n2 -> n3 [label=""];
  n1 -> n2 [label=""];
  n3 -> n4 [label=""];
  n4 -> n5 [label=""];
  n5 -> n6 [label="× 8"];
}
```

- `compact` — prints a graph in the compact mode if the `graph` setting is enabled (the default is `1`).

Example

```
EXPLAIN PIPELINE graph = 1, compact = 0
SELECT sum(number) AS test_sums FROM numbers(5) GROUP BY number % 2 FORMAT TSV;
```

```
digraph
{
  rankdir="LR";
  { node [shape = rect]
    n140206024061464[label="Numbers"];
    n140205465170584[label="Limit"];
    n140205620507928[label="ExpressionTransform"];
    n140205801688600[label="AggregatingTransform"];
    n140205649781464[label="Resize"];
    n140205630520728[label="ExpressionTransform"];
    n140205788048536[label="ExpressionTransform"];
    n140205620512152[label="ExpressionTransform"];
    n140206023882392[label="ExpressionTransform"];
    n140206024060696[label="ExpressionTransform"];
    n140205620509080[label="ExpressionTransform"];
    n140205717371544[label="ExpressionTransform"];
    n140206023882008[label="ExpressionTransform"];
  }
  n140206024061464 -> n140205465170584;
  n140205465170584 -> n140205620507928;
  n140205620507928 -> n140205801688600;
  n140205801688600 -> n140205649781464;
  n140205649781464 -> n140205630520728;
  n140205649781464 -> n140205788048536;
  n140205649781464 -> n140205620512152;
  n140205649781464 -> n140206023882392;
  n140205649781464 -> n140206024060696;
  n140205649781464 -> n140205620509080;
  n140205649781464 -> n140205717371544;
  n140205649781464 -> n140206023882008;
}
```

EXPLAIN ESTIMATE

The `EXPLAIN ESTIMATE` analyzer shows the estimated number of rows, marks, and data parts to be read from a table when processing the query. This type of analysis can only be applied to queries that access MergeTree tables.

Example

1. Create a MergeTree table using options that specify how to divide data parts into granules:
   - `index_granularity` — specifies the maximum number of data rows between marks of an index;
   - `write_final_mark` — indicates whether to write the last index mark at the end of a data part (after the last byte).

```
CREATE TABLE test_table_estimate (i Int64) ENGINE = MergeTree ORDER BY i SETTINGS index_granularity = 16, write_final_mark = 0;
```

2. Insert data into the table by running the following query for three times:

```
INSERT INTO test_table_estimate SELECT number FROM numbers(128);
```

3. Run `EXPLAIN ESTIMATE` for a query that reads data from the table:

```
EXPLAIN ESTIMATE SELECT * FROM test_table_estimate;
```

The result of the query analysis:

```
┌─database─┬─table───────────────┬─parts─┬─rows─┬─marks─┐
│ default  │ test_table_estimate │     3 │  384 │    24 │
└──────────┴─────────────────────┴───────┴──────┴───────┘
```

EXPLAIN TABLE OVERRIDE

The `EXPLAIN TABLE OVERRIDE` query allows you to pre-check the correctness of table schema overrides, which can then be specified for a MaterializedMySQL database creation (see Table Overrides for details). If a table schema override would result in any error, an appropriate message will be displayed in the `EXPLAIN TABLE OVERRIDE` output (for example, when using Nullable columns for the `PARTITION BY` and `ORDER BY` clauses, for which it is not recommended).

To execute an `EXPLAIN TABLE OVERRIDE` query, use the mysql table function to specify a connection to a remote MySQL table.

Example

1. On a remote MySQL server, create a table in the `mysql_db` database:

```
CREATE TABLE mysql_table (id INT PRIMARY KEY, date DATETIME DEFAULT now());
```

2. In ADQM, run the following query to check if the table can be properly partitioned by the `date` column as specified (replace `<host>:<port>` with the MySQL server address, and `<user_name>` and `<user_password>` with a name and password of a user that accesses the MySQL database):

```
EXPLAIN TABLE OVERRIDE mysql('<host>:<port>', 'mysql_db', 'mysql_table', '<user_name>', '<user_password>')
PARTITION BY toYYYYMM(date);
```

In the analysis output, ADQM will throw an exception with a message about incorrect partitioning key override and recommendation on how to fix it:

```
Exception: PARTITION BY override refers to nullable column `date` (use assumeNotNull() if the column does not in fact contain
NULL values). (INVALID_TABLE_OVERRIDE)
....`
```

3. Change the `PARTITION BY` override according to the provided recommendation:

```
EXPLAIN TABLE OVERRIDE mysql('<host>:<port>', 'mysql_db', 'mysql_table', 'root', 'my-secret-pw')
PARTITION BY toYYYYMM(assumeNotNull(date));
```

Now the validation is passed:

```
┌─explain─────────────────────────────────────────┐
│ PARTITION BY uses columns: `date` Nullable(DateTime) │
└─────────────────────────────────────────────────┘
```

Sampling query profiler

Elena Dvoryadkina

Contents

To Table of Contents

- Enable query profiler
- Analyze profiler reports
- Introspection functions
  - addressToLine
  - addressToLineWithInlines
  - addressToSymbol
  - demangle
  - tid
  - logTrace

To introspect query performance in ADQM, you can run a sampling query profiler. For example, it can help you to find source code routines that are most frequently called during a query execution, or trace the CPU and real time spent, including idle time.

Enable query profiler

1. On the ADQMDB service's configuration page of the ADCM interface, enable the ***Show advanced*** option, navigate to the ***Log settings*** section, and activate the ***trace_log*** setting to run the query profiler so that it can start collecting stack traces and saving them in the `trace_log` system table. In the same section, configure parameters of the `trace_log` table:

   - ***Database*** — name of a database where the table should be stored;
   - ***Flush interval, milliseconds*** — interval for flushing data from the memory buffer to the table (in milliseconds);
   - ***TTL, days*** — lifetime of rows in the table (in days).

   Click ***Save*** and execute the ***Reconfig*** action with the enabled ***Restart*** option for the ADQMDB service to save the configuration changes and restart the service.

   NOTE

   - A partitioning key for the `trace_log` table is automatically set to the `event_date` column — date at the moment of sampling. See the trace_log article of the ClickHouse documentation for more details on table columns.
   - Data in the `trace_log` table is valid only for a running server. After a server restart, ADQM does not clean up the table and all the stored virtual memory address may become invalid.

2. Use the `query_profiler_real_time_period_ns` and `query_profiler_cpu_time_period_ns` parameters to configure the real-clock and CPU-clock timers, which determine how often the query profiler should sample stack traces and send them to the `trace_log` table.

   By default, both timers are set to one sample per second. This sampling frequency allows collecting enough information about an ADQM cluster without affecting the performance of ADQM servers. If you need to profile each query individually, try to use higher sampling frequency — for example, 100 times per second:

   ```
   SET query_profiler_real_time_period_ns=10000000;
   ```

   ```
   SET query_profiler_cpu_time_period_ns=10000000;
   ```

   Collecting stack traces with a frequency higher than a few thousand samples per second is usually not possible.

Analyze profiler reports

1. Install the adqm-clickhouse-debuginfo package.
2. Enable introspection functions for query profiling (for security reasons, introspection functions are disabled by default):

   ```
   SET allow_introspection_functions = 1;
   ```

3. Use introspection functions to analyze the `trace_log` table — see the next section for examples on how to do this.

Introspection functions

This section describes functions that can be used to introspect the ELF and DWARF formats for query profiling.

Presets for running examples

1. Before you reproduce examples of using introspection functions, make sure you have the adqm-clickhouse-debuginfo package installed and the `allow_introspection_functions` setting enabled.
2. In the examples below, introspection functions analyze the first stack trace instance from the `trace_log` table. The following query returns the first row of this table:

   ```
   SELECT * FROM system.trace_log LIMIT 1 FORMAT Vertical;
   ```

   ```
   Row 1:
   ──────
   event_date:              2023-09-04
   event_time:              2023-09-04 10:18:01
   event_time_microseconds: 2023-09-04 10:18:01.858935
   timestamp_ns:            1693822681858935963
   revision:                54472
   trace_type:              Memory
   thread_id:               11216
   query_id:                7b53b488-6e38-495a-ab63-e0ed985bf2de
   trace:
   [282096402,282019952,282019801,282176985,373506363,387689543,282729773,282744526,140712295374501,140712292395789]
   size:                    8760
   event:
   increment:               0
   ```

   The `trace` field contains the stack trace at the moment of sampling. Each element of the array is a virtual memory address inside an ADQM server process. Introspection functions can be applied to a single address or to the whole stack trace.

addressToLine

The `addressToLine` function converts a virtual memory address inside an ADQM server process to a filename and a line number in ADQM source code.

Function syntax:

```
addressToLine(<address_of_binary_instruction>)
```

where `<address_of_binary_instruction>` is an address of an instruction in a running process (a value of the UInt64 type).

The function returns a String value:

- Name of a source code file and a line number in that file, delimited by a colon.
- Name of a binary, if the function could not find the debug information.
- Empty string, if the passed address is not valid.

Example

1. Get a source code file name and a line number for a single address:

   ```
   SELECT addressToLine(282096402::UInt64);
   ```

   In the function execution result, `./build/src/Common/StackTrace.cpp` is the file name and `287` is the line number:

   ```
   ┌─addressToLine(CAST('282096402', 'UInt64'))─┐
   │ ./build/src/Common/StackTrace.cpp:287      │
   └────────────────────────────────────────────┘
   ```

2. Apply the function to the whole stack trace, using the arrayMap function to process each individual element of the `trace` array by the `addressToLine` function:

   ```
   SELECT arrayStringConcat(arrayMap(x -> addressToLine(x), trace), '\n') AS trace_source_code_lines
   FROM system.trace_log
   LIMIT 1
   FORMAT Vertical;
   ```

   ```
   Row 1:
   ──────
   trace_source_code_lines: ./build/src/Common/StackTrace.cpp:287
   ./build/src/Common/MemoryTracker.cpp:202
   ./build/src/Common/MemoryTracker.cpp:338
   ./build/src/Common/ThreadStatus.cpp:185
   ./build/contrib/llvm-project/libcxx/include/__memory/unique_ptr.h:302
   ./build/contrib/llvm-project/libcxx/include/__memory/shared_ptr.h:701
   ./build/base/base/wide_integer_impl.h:789
   ./build/contrib/llvm-project/libcxx/include/__memory/unique_ptr.h:302
   /usr/lib64/libpthread-2.17.so
   /usr/lib64/libc-2.17.so
   ```

addressToLineWithInlines

The `addressToLineWithInlines` function is similar to `addressToLine`, but returns an array with all inline functions (so it evaluates more slowly).

Function syntax:

```
addressToLineWithInlines(<address_of_binary_instruction>)
```

where `<address_of_binary_instruction>` is an address of an instruction in a running process (a value of the UInt64 type).

The function returns an array of String values:

- Array where the first element contains a name of a source code file and a line number in that file (delimited by a colon), and other elements include information on inline functions — source code filename, line number, and function name.
- Array with a single element that is a binary name, if the function could not find the debug information.
- Empty array, if the passed address is not valid.

Example

1. Apply the function to a single address:

   ```
   SELECT addressToLineWithInlines(373506363::UInt64) FORMAT Vertical;
   ```

   ```
   Row 1:
   ──────
   addressToLineWithInlines(CAST('373506363', 'UInt64')): ['./build/src/Interpreters/ThreadStatusExt.
   cpp:185','./build/contrib/llvm-project/libcxx/include/__memory/unique_ptr.h:302:std::__1::unique_ptr<DB::QueryProfilerReal,
   std::__1::default_delete<DB::QueryProfilerReal>>::reset[abi:v15000]
   (DB::QueryProfilerReal*)','./build/src/Interpreters/ThreadStatusExt.cpp:415:DB::ThreadStatus::finalizeQueryProfiler()']
   ```

2. Apply the function to the whole stack trace, using the arrayJoin function that splits an array into rows:

   ```
   SELECT address, addressToLineWithInlines(arrayJoin(trace) AS address)
   FROM system.trace_log
   WHERE query_id = '7b53b488-6e38-495a-ab63-e0ed985bf2de'
   FORMAT Vertical;
   ```

   ```
   Row 1:
   ──────
   address:                                    282096402
   addressToLineWithInlines(arrayJoin(trace)): ['./build/src/Common/StackTrace.cpp:287']

   Row 2:
   ──────
   address:                                    282019952
   addressToLineWithInlines(arrayJoin(trace)): ['./build/src/Common/MemoryTracker.cpp:202']

   Row 3:
   ──────
   address:                                    282019801
   addressToLineWithInlines(arrayJoin(trace)): ['./build/src/Common/MemoryTracker.cpp:338']

   Row 4:
   ──────
   address:                                    282176985
   addressToLineWithInlines(arrayJoin(trace)): ['./build/src/Common/ThreadStatus.cpp:185']

   Row 5:
   ──────
   address:                                    373506363
   addressToLineWithInlines(arrayJoin(trace)): ['./build/src/Interpreters/ThreadStatusExt.cpp:185','./build/contrib/llvm-
   project/libcxx/include/__memory/unique_ptr.h:302:std::__1::unique_ptr<DB::QueryProfilerReal,
   std::__1::default_delete<DB::QueryProfilerReal>>::reset[abi:v15000]
   (DB::QueryProfilerReal*)','./build/src/Interpreters/ThreadStatusExt.cpp:415:DB::ThreadStatus::finalizeQueryProfiler()']

   ...
   ```

addressToSymbol

The `addressToSymbol` function converts a virtual memory address inside an ADQM server process to a symbol from an object file.

Function syntax:

```
addressToSymbol(<address_of_binary_instruction>)
```

where `<address_of_binary_instruction>` is an address of an instruction in a running process (a value of the UInt64 type).

The function returns a String value:

- Symbol from an object file.
- Empty string, if the passed address is not valid.

Example

1. Get a symbol for a single address:

   ```
   SELECT addressToSymbol(282096402::UInt64);
   ```

   ```
   ┌─addressToSymbol(CAST('282096402', 'UInt64'))─┐
   │ _ZN10StackTrace10tryCaptureEv                │
   └──────────────────────────────────────────────┘
   ```

2. Apply the function to the whole stack trace, using the arrayMap function to process each individual element of the `trace` array by the `addressToSymbol` function:

   ```
   SELECT arrayStringConcat(arrayMap(x -> addressToSymbol(x), trace), '\n') AS trace_symbols
   FROM system.trace_log
   LIMIT 1
   FORMAT Vertical;
   ```

   ```
   Row 1:
   ──────
   trace_symbols: _ZN10StackTrace10tryCaptureEv
   _ZN13MemoryTracker9allocImplElbPS_
   _ZN13MemoryTracker9allocImplElbPS_
   _ZN2DB12ThreadStatus20flushUntrackedMemoryEv
   _ZN2DB12ThreadStatus15detachFromGroupEv
   _ZNSt3__110__function16__policy_invokerIFvvEE11__call_implINS0_20__default_alloc_funcIZN24ThreadFromGlobalPoolImplILb1EEC1I
   ZN2DB28PullingAsyncPipelineExecutor4pullERNS9_5ChunkEmE3$_0JEEEOT_DpOT0_EUlvE_S2_EEEEvPKNS0_16__policy_storageE
   _ZN14ThreadPoolImplINSt3__16threadEE6workerENS0_15__list_iteratorIS1_PvEE
   _ZNSt3__114__thread_proxyB6v15000INS_5tupleIJNS_10unique_ptrINS_15__thread_structENS_14default_deleteIS3_EEEEZN14ThreadPool
   ImplINS_6threadEE12scheduleImplIvEET_NS_8functionIFvvEEElNS_8optionalImEEbEUlvE0_EEEEEPvSJ_
   start_thread
   __clone
   ```

demangle

The `demangle` function converts a symbol that you can get using the `addressToSymbol` function into a C++ function name.

Function syntax:

```
demangle(<symbol>)
```

where `<symbol>` is a symbol from an object file (a value of the String type).

The function returns a String value:

- C++ function name.
- Empty string, if the passed symbol is not valid.

Example

1. Get a function name for a single address:

   ```
   SELECT demangle(addressToSymbol(282096402::UInt64));
   ```

   ```
   ┌─demangle(addressToSymbol(CAST('282096402', 'UInt64')))─┐
   │ StackTrace::tryCapture()                               │
   └────────────────────────────────────────────────────────┘
   ```

2. Apply the function to the whole stack trace, using the arrayMap function to process each individual element of the `trace` array by the `demangle` function:

   ```
   SELECT arrayStringConcat(arrayMap(x -> demangle(addressToSymbol(x)), trace), '\n') AS trace_functions
   FROM system.trace_log
   LIMIT 1
   FORMAT Vertical;
   ```

   ```
   Row 1:
   ──────
   trace_functions: StackTrace::tryCapture()
   MemoryTracker::allocImpl(long, bool, MemoryTracker*)
   MemoryTracker::allocImpl(long, bool, MemoryTracker*)
   DB::ThreadStatus::flushUntrackedMemory()
   DB::ThreadStatus::detachFromGroup()
   void std::__1::__function::__policy_invoker<void
   ()>::__call_impl<std::__1::__function::__default_alloc_func<ThreadFromGlobalPoolImpl<true>::ThreadFromGlobalPoolImpl<DB::Pu
   llingAsyncPipelineExecutor::pull(DB::Chunk&, unsigned long)::$_0>(DB::PullingAsyncPipelineExecutor::pull(DB::Chunk&,
   unsigned long)::$_0&&)::'lambda'(), void ()>>(std::__1::__function::__policy_storage const*)
   ThreadPoolImpl<std::__1::thread>::worker(std::__1::__list_iterator<std::__1::thread, void*>)
   void* std::__1::__thread_proxy[abi:v15000]<std::__1::tuple<std::__1::unique_ptr<std::__1::__thread_struct,
   std::__1::default_delete<std::__1::__thread_struct>>, void ThreadPoolImpl<std::__1::thread>::scheduleImpl<void>
   (std::__1::function<void ()>, long, std::__1::optional<unsigned long>, bool)::'lambda0'()>>(void*)
   start_thread
   __clone
   ```

tid

The `tid` function returns id of a thread, in which the current Block is processed.

Function syntax:

```
tid()
```

The function returns a value of the Uint64 type.

Example

```
SELECT tid();
```

```
┌─tid()─┐
│ 10959 │
└───────┘
```

logTrace

The `logTrace` function sends a trace log message to a server log for each Block.

Function syntax:

```
logTrace('<message>')
```

where `<message>` is a message that is emitted to a server log (a String value).

The function always returns `0`.

Example

```
SELECT logTrace('logTrace message');
```

```
┌─logTrace('logTrace message')─┐
│                            0 │
└──────────────────────────────┘
```

Kill a query

Elena Dvoryadkina

Contents

To Table of Contents

KILL QUERY
  Example
KILL MUTATION
  Example
Known issue

To stop query execution and interrupt mutations, ADQM/ClickHouse provides the KILL QUERY and KILL MUTATION commands.

KILL QUERY

Although ADQM/ClickHouse is a high-performance DBMS, poorly optimized queries (for example, full scans of large data sets, use of inefficient indexes, or non-optimized JOIN operations) can take a long time to execute, consume a lot of resources, and block critical processes. In such cases, it may be necessary to stop the query execution — to do this, use the `KILL QUERY` command.

TIP

See the best practices to avoid long-running queries in the Improve query performance article.

The basic syntax of `KILL QUERY`:

```
KILL QUERY [ON CLUSTER <cluster_name>] WHERE <where_expr> [ASYNC|SYNC|TEST] [FORMAT <output_format>];
```

where:

- `[ON CLUSTER <cluster_name>]` — optional clause used to specify a cluster name when a query should be terminated on all cluster nodes.
- `WHERE <where_expr>` — mandatory clause that defines the criteria for selecting queries to be terminated.
  To find queries that are taking too long to execute and/or consuming too much memory, you can use the system.processes system table, which contains information about currently running queries and their states. For example, you can use the `query_id`, `user`, `elapsed`, `memory_usage`, and other relevant fields of this table to identify a target query to be stopped:

```
SELECT query_id, user, elapsed, memory_usage FROM system.processes ORDER BY elapsed DESC;
```

  As an alternative to `SELECT * FROM system.processes`, you can use the command:

```
SHOW PROCESSLIST;
```

- `[ASYNC|SYNC|TEST]` — keyword indicating whether the `KILL QUERY` command should be executed asynchronously, synchronously, or just as a test run:
  - `ASYNC` (by default) — the command does not wait for confirmation that execution of target queries has stopped.
  - `SYNC` — the command waits until all target queries are actually terminated and displays information about each process as it stops.
  - `TEST` — test run of the `KILL QUERY` command, which allows you to check user permissions and to view which queries would be affected by the command without actually terminating them.
- `[FORMAT <output_format>]` — optional clause specifying the format in which the `KILL QUERY` operation result should be displayed.

The `KILL QUERY` query response contains the `kill_status` column, the possible values of which include:

- `finished` — a query has been successfully stopped;
- `waiting` — waiting for a query to stop after sending it a signal to terminate;
- `cant_cancel` — a signal to terminate cannot be sent;
- `pending` — a query has not been initialized yet;
- `unknown_status` — status of the query termination operation is unknown.

In order for a user to cancel queries of other users, they should have the `KILL QUERY` privilege. Read-only users can stop only their own queries.

Example

NOTE

This example uses two ADQM user accounts:

- `default` — default user with comprehensive rights;
- `adqm_user` — user with rights to read data (`SELECT`) and cancel queries of other users (`KILL QUERY`).

1. In two separate clickhouse-client sessions, launch an infinitely running test query as the default user, for example:

```
SELECT count() FROM system.numbers;
```

   While the query is running, the client will display the query ID and progress.

   - Session 1:

```
SELECT count()
FROM system.numbers

Query id: 6d522fb4-97db-45e4-9565-3158edb5a281

↖ Progress: 398.17 billion rows, 3.19 TB (3.40 billion rows/s., 27.22 GB/s.)   (1.0 CPU)
Press the space key to toggle the display of the progress table.
```

   - Session 2:

```
SELECT count()
FROM system.numbers

Query id: f38506fd-4923-4bd2-8668-e6affdaa61c4

↖ Progress: 403.64 billion rows, 3.23 TB (3.41 billion rows/s., 27.24 GB/s.)   (1.0 CPU)
Press the space key to toggle the display of the progress table.
```

2. Open another `clickhouse-client` session as the `adqm_user` user. View the list of incomplete queries started by the `default` user and their current execution time:

```
SELECT query_id, user, elapsed, memory_usage, query
FROM system.processes
WHERE user='default'
ORDER BY elapsed DESC;
```

```
   ┌─query_id─────────────────────────────┬─user────┬────elapsed─┬─memory_usage─┬─query──────────────────────────┐
1. │ 6d522fb4-97db-45e4-9565-3158edb5a281 │ default │ 133.361736 │            0 │ SELECT count() FROM system.numbers; │
2. │ f38506fd-4923-4bd2-8668-e6affdaa61c4 │ default │ 125.132542 │            0 │ SELECT count() FROM system.numbers; │
   └──────────────────────────────────────┴─────────┴────────────┴──────────────┴────────────────────────────────┘
```

3. Also on behalf of the `adqm_user` user, force the queries to stop using one of the methods listed below.

   - Stop a query by its ID:

```
KILL QUERY WHERE query_id = '6d522fb4-97db-45e4-9565-3158edb5a281';
```

     The `KILL QUERY` command output:

```
   ┌─kill_status─┬─query_id─────────────────────────────┬─user────┬─query──────────────────────────────┐
1. │ waiting     │ 6d522fb4-97db-45e4-9565-3158edb5a281│ default │ SELECT count() FROM system.numbers; │
   └─────────────┴──────────────────────────────────────┴─────────┴────────────────────────────────────┘
```

     The `kill_status` column contains the `waiting` value because the `KILL QUERY` command is executed asynchronously by default, that is, it returns the result immediately, without waiting for the actual completion of the target query.

   - Stop all queries started by the `default` user:

```
KILL QUERY WHERE user = 'default';
```

```
   ┌─kill_status─┬─query_id─────────────────────────────┬─user────┬─query──────────────────────────────┐
1. │ waiting     │ 6d522fb4-97db-45e4-9565-3158edb5a281 │ default │ SELECT count() FROM system.numbers; │
2. │ waiting     │ f38506fd-4923-4bd2-8668-e6affdaa61c4 │ default │ SELECT count() FROM system.numbers; │
   └─────────────┴──────────────────────────────────────┴─────────┴────────────────────────────────────┘
```

     Before calling the command to cancel queries, you can run `KILL QUERY` with the `TEST` keyword to make sure that the `adqm_user` user has permissions to stop queries started by another user, and to see which queries this command will kill:

```
KILL QUERY WHERE user = 'default' TEST;
```

```
   ┌─kill_status────┬─query_id─────────────────────────────┬─user────┬─query──────────────────────────────┐
1. │ unknown_status │ 6d522fb4-97db-45e4-9565-3158edb5a281 │ default │ SELECT count() FROM system.numbers; │
2. │ unknown_status │ f38506fd-4923-4bd2-8668-e6affdaa61c4 │ default │ SELECT count() FROM system.numbers; │
   └────────────────┴──────────────────────────────────────┴─────────┴────────────────────────────────────┘
```

     If a user does not have the necessary permissions to interrupt another user's queries, the client will display a corresponding message, for example: `DB::Exception: User adqm_user attempts to kill query created by default. (ACCESS_DENIED)`.

   - Stop all currently running queries that are longer than 90 seconds:

```
KILL QUERY WHERE elapsed > 90;
```

KILL MUTATION

Mutations in ADQM/ClickHouse (the `ALTER TABLE … UPDATE/DELETE` operations to update/delete data) involve overwriting large amounts of data and can be quite resource-intensive. Once a mutation has started, it cannot be rolled back, and its execution will continue even after a server restart. However, you can explicitly stop the mutation operation with the `KILL MUTATION` command (for example, if the mutation has been running for too long or is no longer needed). Killing a mutation cancels further processing, but all changes already applied to the data will remain. Therefore, it is recommended to use `KILL MUTATION` carefully, especially in production systems, as it may affect data integrity.

The basic syntax of the `KILL MUTATION` query:

```
KILL MUTATION [ON CLUSTER <cluster_name>] WHERE <where_expr> [ASYNC|SYNC|TEST] [FORMAT <output_format>];
```

where:

- `[ON CLUSTER <cluster_name>]` — name of a cluster on which the mutation execution should be stopped.
- `WHERE <where_expr>` — mandatory clause for selecting mutations to be terminated. You can view mutations that are currently in progress and could potentially be problematic in the system.mutations table (rows with `is_done = 0`). Use values of the `database`, `table`, `mutation_id`, and other columns of this table in the `<where_expr>` expression to specify mutations the `KILL MUTATION` command should stop.
- `[ASYNC|SYNC|TEST]` — keyword that specifies whether the `KILL MUTATION` command should be executed asynchronously, synchronously, or as a test run (similar to KILL QUERY).
- `[FORMAT <output_format>]` — optional clause that specifies the output format for the command result.

Just like `KILL QUERY`, the `KILL MUTATION` command outputs the kill_status column that shows the status of the mutation termination operation.

To terminate a mutation with the `KILL MUTATION` command, a user should have the same privilege that is required to start this mutation. For example, to stop an `ALTER DELETE` mutation, the `ALTER DELETE`, `ALTER TABLE`, or `ALTER` privilege is required. To execute `KILL MUTATION ON CLUSTER`, a user should have the privileges to perform all mutation types (`ALTER UPDATE`, `ALTER DELETE`, `ALTER MATERIALIZE COLUMN`, `ALTER MATERIALIZE INDEX`, `ALTER MATERIALIZE TTL`) across all databases and tables (`ON *.*`). This is because when running a query on a cluster, the system first checks permissions and does not know which specific mutations the `KILL MUTATION` command will terminate until it begins.

Example

1. View mutations that are in progress:

```
SELECT * FROM system.mutations WHERE is_done = 0;
```

   The following query returns active mutations in a cluster (replace `<cluster_name>` with your cluster name):

```
SELECT * FROM clusterAllReplicas('<cluster_name>', system.mutations) WHERE is_done = 0;
```

2. Stop mutations by filtering them in the `KILL MUTATION` query by values from the `system.mutations` table.

   - Stop all unfinished mutations on the `my_table` table of the `my_database` database in the `my_cluster` cluster:

```
KILL MUTATION ON CLUSTER my_cluster WHERE database = 'my_database' AND table = 'my_table' AND is_done = 0;
```

   - Stop a specific mutation with the `mutation_2.txt` identifier on the `my_table` table of the `my_database` database:

```
KILL MUTATION WHERE database = 'my_database' AND table = 'my_table' AND mutation_id = 'mutation_2.txt' AND is_done = 0;
```

     An example of the `KILL MUTATION` query output (where `<mutation_query_text>` is the text of the command to modify/delete data, `UPDATE…` or `DELETE…`):

```
   ┌─kill_status─┬─database────┬─table────┬─mutation_id────┬─command───────────────┐
1. │ waiting     │ my_database │ my_table │ mutation_2.txt │ (<mutation_query_text>) │
   └─────────────┴─────────────┴──────────┴────────────────┴───────────────────────┘
```

Known issue

Some types of queries cannot be stopped by the `KILL QUERY` command. This command doesn't actually terminate query execution — it only sets a flag, which should be checked by the query at certain points of its execution (usually after completing processing one block of the query execution pipeline and before starting the next one). A query cannot be terminated if it is stuck at a stage where this flag is not checked (for example, during a scalar subquery evaluation, which is performed at the query analysis stage, outside the main execution pipeline of the query). In a case when a query cannot be killed with `KILL QUERY`, the only way to stop it may be to restart the ClickHouse server.

Encryption codecs

Elena Dvoryadkina

Contents

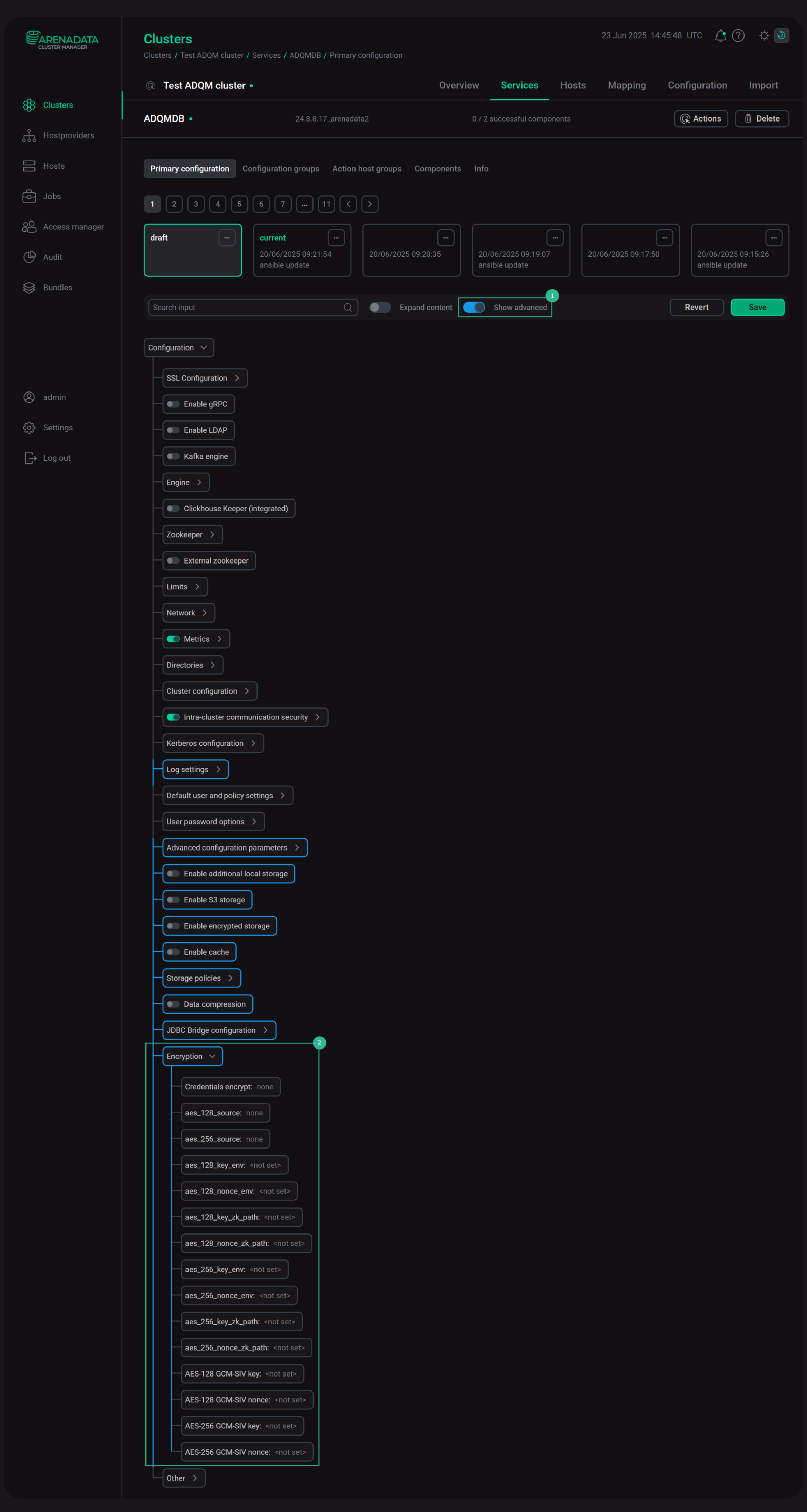
To Table of Contents

Configure encryption keys
Apply data encryption algorithms
- Encrypt passwords
- Encrypt data of table columns
Example

ADQM allows you to encrypt sensitive data (passwords stored in ClickHouse configuration files) and table column data on disk using the AES-128 or AES-256 algorithm in GCM-SIV mode.

Configure encryption keys

Configuration parameters for the AES-128 and AES-256 encryption algorithms are located in the ***Encryption*** section on the configuration page of the ADQMDB service. This section becomes visible when the ***Show advanced*** option is enabled.



ARENADATA CLUSTER MANAGER
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
Clusters
Clusters / Test ADQM cluster / Services / ADQMDB / Primary configuration
23 Jun 2025 14:45:48 UTC
Test ADQM cluster
Overview
Services
Hosts
Mapping
Configuration
Import
ADQMDB
24.8.8.17_arenadata2
0 / 2 successful components
Actions
Delete
Primary configuration
Configuration groups
Action host groups
Components
Info
1 2 3 4 5 6 7 ... 11
draft
current
20/06/2025 09:21:54 ansible update
20/06/2025 09:20:35
20/06/2025 09:19:07 ansible update
20/06/2025 09:17:50
20/06/2025 09:15:26 ansible update
Search input
Expand content
Show advanced
1
Revert
Save
Configuration
SSL Configuration
Enable gRPC
Enable LDAP
Kafka engine
Engine
Clickhouse Keeper (integrated)
Zookeeper
External zookeeper
Limits
Network
Metrics
Directories
Cluster configuration
Intra-cluster communication security
Kerberos configuration
Log settings
Default user and policy settings
User password options
Advanced configuration parameters
Enable additional local storage
Enable S3 storage
Enable encrypted storage
Enable cache
Storage policies
Data compression
JDBC Bridge configuration
2
Encryption
Other

Credentials encrypt: none
aes_128_source: none
aes_256_source: none
aes_128_key_env: <not set>
aes_128_nonce_env: <not set>
aes_128_key_zk_path: <not set>
aes_128_nonce_zk_path: <not set>
aes_256_key_env: <not set>
aes_256_nonce_env: <not set>
aes_256_key_zk_path: <not set>
aes_256_nonce_zk_path: <not set>
AES-128 GCM-SIV key: <not set>
AES-128 GCM-SIV nonce: <not set>
AES-256 GCM-SIV key: <not set>
AES-256 GCM-SIV nonce: <not set>

Configuration of encryption algorithms

Use parameters in this section to set a key and nonce for the required encryption algorithm or to specify where to load them from (it is recommended to store encryption keys in environment variables or ZooKeeper nodes instead of configuration files). For parameter descriptions, see the corresponding section of the Configuration parameters article.

AES-128

- Load keys from environment variables:
  - In the ***aes_128_key_env*** and ***aes_128_nonce_env*** parameters, specify environment variables that contain a key and nonce for the encryption algorithm, respectively.
  - Set the ***aes_128_source*** parameter to `from_env`.
- Load keys from ZooKeeper nodes:
  - In the ***aes_128_key_zk_path*** and ***aes_128_nonce_zk_path*** parameters, specify ZooKeeper nodes that contain a key and nonce for the encryption algorithm, respectively.
  - Set the ***aes_128_source*** parameter to `from_zk`.
- Store keys in a configuration file (not recommended):
  - In the ***AES-128 GCM-SIV key*** and ***AES-128 GCM-SIV nonce*** parameters, specify a key and nonce for the encryption algorithm, respectively.
  - Set the ***aes_128_source*** parameter to `none`.

AES-256

- Load keys from environment variables:
  - In the ***aes_256_key_env*** and ***aes_256_nonce_env*** parameters, specify environment variables that contain a key and nonce for the encryption algorithm, respectively.
  - Set the ***aes_256_source*** parameter to `from_env`.
- Load keys from ZooKeeper nodes:
  - In the ***aes_256_key_zk_path*** and ***aes_256_nonce_zk_path*** parameters, specify ZooKeeper nodes that contain a key and nonce for the encryption algorithm, respectively.
  - Set the ***aes_256_source*** parameter to `from_zk`.
- Store keys in a configuration file (not recommended):
  - In the ***AES-256 GCM-SIV key*** and ***AES-256 GCM-SIV nonce*** parameters, specify a key and nonce for the encryption algorithm, respectively.
  - Set the ***aes_256_source*** parameter to `none`.

After you have defined the parameters, click ***Save*** and run the ***Reconfig*** action for the ADQMDB service.

Encryption algorithm settings are saved to the `encryption_codecs` section of the */etc/clickhouse-server/config.d/encryption.xml* configuration file.

Apply data encryption algorithms

Once the keys for the AES-128/AES-256 algorithms have been specified, you can enable encryption for:

- passwords stored in a configuration file;
- table column data on disk.

Encrypt passwords

In the ***Encryption*** section of the ADQMDB service configuration, set the ***Credentials encrypt*** parameter to `AES-128` or `AES-128`. Save the configuration and execute the ***Reconfig*** service action.

After this, values of the following parameters (if set up) will be hidden in the */etc/clickhouse-server/credentials.xml* configuration file:

- `interserver_http_password` — password for authentication between replicas;
- `shard_secret` — cluster secret to validate distributed queries on shards;
- `default_user_password` — password of the default user.

Example of credentials.xml

```
<clickhouse>
    <interserver_http_password>
        <password hide_in_preprocessed="true" encrypted_by="AES_256_GCM_SIV">...encrypted value...</password>
    </interserver_http_password>
    <shard_secret>
        <secret hide_in_preprocessed="true" encrypted_by="AES_256_GCM_SIV">...encrypted value...</secret>
    </shard_secret>
    <default_user_password>
        <password hide_in_preprocessed="true" encrypted_by="AES_256_GCM_SIV">...encrypted value...</password>
    </default_user_password>
</clickhouse>
```

CAUTION

Loading keys from ZooKeeper nodes is currently not supported for password encryption if SSL is enabled for connections between ClickHouse and ZooKeeper.

Encrypt data of table columns

In ClickHouse, a data encryption algorithm applied to a table column is defined by an **encryption codec**. To assign an encryption codec to a column, use the `CODEC` clause in the CREATE TABLE query:

```
CREATE TABLE <table_name>
(   <column_name1> <data_type1> CODEC(AES_128_GCM_SIV | AES_256_GCM_SIV) [...],
    <column_name2> <data_type2> CODEC(AES_128_GCM_SIV | AES_256_GCM_SIV) [...],
    ...)
ENGINE = MergeTree()
...;
```

Encryption codecs actually only encrypt column data on disk but do not compress it. If compression is required, specify it explicitly for a column — to do this, list the desired data compression codecs separated by commas (specialized codecs first, and then general-purpose codecs) before an encryption codec in the `CODEC` clause. For example:

```
CREATE TABLE <table_name>
(   <column_name> <data_type> CODEC(Delta, LZ4, AES_256_GCM_SIV) [...],
    ...)
ENGINE = MergeTree()
...;
```

CAUTION

- Most table engines, including MergeTree family engines, create index files on disk without applying encryption codecs. This means that plaintext may appear on disk if an encrypted column is indexed.
- If a `SELECT` query mentions a specific value from an encrypted column (for example, within the `WHERE` expression), this value may appear in the `system.query_log` log table. In this case, you may want to disable logging.

Example

The following example shows how to set up a key stored in ZooKeeper for the AES-256 encryption algorithm in ADQM, and how to apply this algorithm to encrypt the `default` user password (which should be set — see Configure authorization for a new cluster) and data of a test table column.

1. Create two ZooKeeper nodes to store a key and nonce for the AES-256 encryption algorithm (for this, you can use the ZooKeeper CLI available on each cluster host where ZooKeeper is installed):

   ```
   $ sudo /usr/lib/zookeeper/bin/zkCli.sh
   ```

   ```
   create /clickhouse/clickhouse/aes256_key_hex '00112233445566778899AABBCCDDEEFF00112233445566778899AABBCCDDEEFF'
   create /clickhouse/clickhouse/aes256_nonce_hex 'ABCDEFABCDEFABCDEFABCDEF'
   ```

2. In the ***Encryption*** section of the ADQMDB service configuration, set the following parameter values:

   - ***aes_256_key_zk_path***: `/clickhouse/aes256_key_hex`;
   - ***es_256_nonce_zk_path***: `/clickhouse/aes256_nonce_hex`;
   - ***aes_256_source***: `from_zk`;
   - ***Credentials encrypt***: `AES-256`.

   Save the configuration and run the ***Reconfig*** action for the ADQMDB service.

   Make sure that the encryption codec settings are saved in the */etc/clickhouse-server/config.d/encryption.xml* file:

   ```
   <clickhouse>
       <encryption_codecs>
           <aes_256_gcm_siv>
               <key_hex hide_in_preprocessed="true" from_zk="/clickhouse/aes256_key_hex"/>
               <nonce_hex hide_in_preprocessed="true" from_zk="/clickhouse/aes256_nonce_hex"/>
           </aes_256_gcm_siv>
       </encryption_codecs>
   </clickhouse>
   ```

3. In the */etc/clickhouse-server/credentials.xml* configuration file, check that the password of the `default` user is now encrypted:

   ```
   <clickhouse>
       <default_user_password>
           <password hide_in_preprocessed="true" encrypted_by="AES_256_GCM_SIV">...encrypted password...</password>
       </default_user_password>
   </clickhouse>
   ```

4. Create an ADQM table and insert test data into it:

   ```
   CREATE TABLE encryption_test (x String CODEC(AES_256_GCM_SIV), y String) ENGINE = MergeTree ORDER BY x;
   INSERT INTO encryption_test VALUES ('1', '10'), ('2', '20'), ('3', '30');
   ```

   Check the contents of */var/lib/clickhouse/data/default/encryption_test/all_1_1_0/data.bin* to ensure that the first column data is stored encrypted on disk.

ADQM functions for machine learning

Elena Dvoryadkina

Contents

To Table of Contents

Basic concepts

Machine learning algorithms

- Simple linear regression
- Stochastic regression

How to use machine learning functions

- Fit a model
- Predict new values

Example

- Prepare data
- Create a machine learning model
- Evaluate the model

ADQM/ClickHouse provides a set of machine learning functions — aggregate functions implementing machine learning algorithms that you can train on known data sets and then use to predict unknown values.

Basic concepts

In this article, the following basic concepts are used to describe machine learning algorithms and ADQM functions that implement them:

- **Independent and dependent variables.** Independent variables are the input to a machine learning algorithm. Dependent variables (also known as target variables) are the output that depend on independent variables. In other words, independent variables are features of an object (or event) whose values can be used to predict a value of a target variable using a trained machine learning model.
- **Training and test data sets.** Training data is used to train a machine learning model. It should include columns with values of independent variables and a column with target values that the model should learn to predict. Test data is another data set (independent of the training data set) used to evaluate the model fit on the training data.
- **Model training.** A machine learning algorithm finds patterns in the provided training data that map values of independent variables to target variable values, and outputs a machine learning model that captures these patterns. The model can then be used to make predictions based on new data for which values of the target variable are unknown.
- **Model parameters or regression weights (coefficients).** Model parameters (or regression weights) define how independent variables affect a dependent one. For example, if the linear regression equation is $y = a_1 \cdot x_1 + a_2 \cdot x_2 + ... + b$, the model parameters are the $a_1, a_2, ..., b$ constants (b is the bias). Model parameters are automatically estimated during model training based on the training data and returned by an ADQM machine learning function.
- **Model hyperparameters.** Hyperparameters are preset configurations that are manually adjusted before the training process starts and determine the model structure and the way it is trained. Properly selected hyperparameters can significantly affect the model efficiency and accuracy.

Machine learning algorithms

ADQM functions for machine learning implement the following regression algorithms.

Simple linear regression

The `simpleLinearRegression(<feature>, <target>)` function performs simple linear regression, i.e. calculates a linear relationship between two variables — one independent variable and a target one. Pass columns with independent and target variable values from a training data set as the `<feature>` and `<target>` function parameters, respectively.

Stochastic regression

To predict values of a target variable based on two or more independent variables, ADQM provides the following multiple-regression functions that use the **stochastic gradient descent** method to train a model:

- `stochasticLinearRegression([<hyperparameters>])(<target>, <feature1>, <feature2>, …)` — estimates a linear relationship between a target variable (`<target>`) and a set of independent variables (`<feature1>, <feature2>, …`) and allows prediction of continuous numerical values of the dependent variable based on the observed values of the independent variables.
- `stochasticLogisticRegression([<hyperparameters>])(<target>, <feature1>, <feature2>, …)` — allows estimating the probability of an event occurrence based on values of multiple input features. Since the outcome is a probability, the dependent variable can take one of two values — usually `0` (the event did not occur) and `1` (the event did occur). This algorithm is used for binary classification tasks.

When using a stochastic regression function to build a machine learning model, you can customize four hyperparameters (`<hyperparameters>` in the function syntax descriptions above). The following table lists the hyperparameters in the order they should be passed to a function sequentially.

Hyperparameter	Description	Default value
learning rate	Step length coefficient, when the gradient descent step is performed. Too high learning rate may cause infinite weights of the model	0.00001
l2 regularization coefficient	L2 regularization coefficient, which may help to avoid overfitting	0.1
mini-batch size	Number of elements of the training set whose gradients will be computed and summed to perform one step of gradient descent. Pure stochastic descent uses a single element, however, using a mini-batch of a fixed small number of elements (about 10 elements) makes the gradient steps more stable	15
method for updating weights	Method for updating weights. Possible values: `Adam` (default), `SGD`, `Momentum`, and `Nesterov`. The last two methods are more demanding on computing resources and memory, however, they have a high convergence speed and stability of stochastic gradient methods	Adam

How to use machine learning functions

Machine learning functions are used to fit a model and predict new data.

Fit a model

To fit a model, follow the steps:

1. Create a table based on a training data set. A sample query:

   ```
   CREATE TABLE train_data
   (
       feature1 Float64,
       feature2 Float64,
       target Float64
   )
   ENGINE = MergeTree;
   ```

   For linear regression, independent (`feature1`, `feature2` ...) and target (`target`) variables should be numeric. For logistic regression, the target variable should be binary (takes only two values — for example, `0` and `1`) and predicted values (probabilities) will range between `0` and `1`.

2. Use a machine learning function with the `State` combinator to fit a model and save its state for later use. An example of a query with the `stochasticLinearRegression` function:

   ```
   CREATE TABLE my_model ENGINE = Memory AS SELECT
   stochasticLinearRegressionState(0.001, 0.1, 10, 'SGD')(target, feature1, feature2)
   AS state FROM train_data;
   ```

   If you don't apply the `State` combinator to a machine learning function, you can fetch the parameters of the created model without saving it. For example, the following query fits a model and returns a column with its parameters (the first values correspond to the regression weights and the last one is bias):

   ```
   stochasticLinearRegression(0.001, 0.1, 10, 'SGD')(target, feature1, feature2) FROM train_data;
   ```

Predict new values

After saving a model state into a table, you can use it multiple times to make predictions or merge it with other states to create new (improved) models.

1. Create a table based on a test data set (`test_data` in the example below). This table should be similar in structure to `train_data`, but may not contain a target value (it will be predicted).
2. To make predictions, use the `evalMLMethod` function, which accepts a model state and features to predict on as parameters. For example, the following query returns a column with target variable values predicted based on values of the `feature1` and `feature2` features in the `test_data` table:

   ```
   WITH (SELECT state FROM my_model) AS model SELECT
   evalMLMethod(model, feature1, feature2) FROM test_data;
   ```

Example

Prepare data

1. Create tables with data for training (`trips_train`) and testing (`trips_test`) as described in the New York Taxi Data example that the ClickHouse documentation provides:

   ```
   CREATE TABLE trips_train (
       trip_id             UInt32,
       pickup_datetime     DateTime,
       dropoff_datetime    DateTime,
       pickup_longitude    Nullable(Float64),
       pickup_latitude     Nullable(Float64),
       dropoff_longitude   Nullable(Float64),
       dropoff_latitude    Nullable(Float64),
       passenger_count     UInt8,
       trip_distance       Float32,
       fare_amount         Float32,
       extra               Float32,
       tip_amount          Float32,
       tolls_amount        Float32,
       total_amount        Float32,
       payment_type        Enum('CSH' = 1, 'CRE' = 2, 'NOC' = 3, 'DIS' = 4, 'UNK' = 5),
       pickup_ntaname      LowCardinality(String),
       dropoff_ntaname     LowCardinality(String)
   )
   ENGINE = MergeTree
   PRIMARY KEY (pickup_datetime, dropoff_datetime);
   ```

   ```
   CREATE TABLE trips_test (
       trip_id             UInt32,
       pickup_datetime     DateTime,
       dropoff_datetime    DateTime,
       pickup_longitude    Nullable(Float64),
       pickup_latitude     Nullable(Float64),
       dropoff_longitude   Nullable(Float64),
       dropoff_latitude    Nullable(Float64),
       passenger_count     UInt8,
       trip_distance       Float32,
       fare_amount         Float32,
       extra               Float32,
       tip_amount          Float32,
       tolls_amount        Float32,
       total_amount        Float32,
       payment_type        Enum('CSH' = 1, 'CRE' = 2, 'NOC' = 3, 'DIS' = 4, 'UNK' = 5),
       pickup_ntaname      LowCardinality(String),
       dropoff_ntaname     LowCardinality(String)
   )
   ENGINE = MergeTree
   PRIMARY KEY (pickup_datetime, dropoff_datetime);
   ```

2. Fill the `trips_train` and `trips_test` tables with data from the *trips_0.gz* and *trips_1.gz* files in the S3 bucket, respectively:

   ```
   INSERT INTO trips_train
   SELECT
       trip_id,
       pickup_datetime,
       dropoff_datetime,
       pickup_longitude,
       pickup_latitude,
       dropoff_longitude,
       dropoff_latitude,
       passenger_count,
       trip_distance,
       fare_amount,
       extra,
       tip_amount,
       tolls_amount,
       total_amount,
       payment_type,
       pickup_ntaname,
       dropoff_ntaname
   FROM s3('https://datasets-documentation.s3.eu-west-3.amazonaws.com/nyc-taxi/trips_0.gz', 'TabSeparatedWithNames');
   ```

   ```
   INSERT INTO trips_test
   SELECT
       trip_id,
       pickup_datetime,
       dropoff_datetime,
       pickup_longitude,
       pickup_latitude,
       dropoff_longitude,
       dropoff_latitude,
       passenger_count,
       trip_distance,
       fare_amount,
       extra,
       tip_amount,
       tolls_amount,
       total_amount,
       payment_type,
       pickup_ntaname,
       dropoff_ntaname
   FROM s3('https://datasets-documentation.s3.eu-west-3.amazonaws.com/nyc-taxi/trips_1.gz', 'TabSeparatedWithNames');
   ```

Create a machine learning model

1. Create a machine learning model to predict taxi fare (a target variable is `fare_amount`) depending on the number of passengers and trip distance (the `passenger_count` and `trip_distance` independent variables):

   ```
   SELECT stochasticLinearRegression(0.001, 0.1, 5)(fare_amount, passenger_count, trip_distance)
   FROM trips_train;
   ```

   This query returns model parameters that determine the weights of the linear regression equation ($\mathrm{fare\_amount} = 0.08 \cdot \mathrm{passenger\_count} + 2.7 \cdot \mathrm{trip\_distance} + 4.9$):

   ```
   ┌─stochasticLinearRegression(0.001, 0.1, 5)(fare_amount, passenger_count, trip_distance)─┐
   │ [0.08194997204590103,2.706812913667359,4.897836872083942]                              │
   └────────────────────────────────────────────────────────────────────────────────────────┘

   1 row in set. Elapsed: 0.036 sec. Processed 1.00 million rows, 9.01 MB (27.71 million rows/s., 249.35 MB/s.)
   Peak memory usage: 2.38 MiB.
   ```

   As you can see, the fare is not much affected by the number of passengers.

2. Save the model state to the `taxi_trips_model` table based on the Memory engine:

   ```
   CREATE TABLE taxi_trips_model ENGINE = Memory AS
   SELECT stochasticLinearRegressionState(0.001, 0.1, 5)(fare_amount, passenger_count, trip_distance)
   AS state FROM trips_train;
   ```

Evaluate the model

Use the trained model to make predictions based on the test data set and compare those predictions to the actual values (note that the `evalMLMethod` function accepts the model state and the same set of independent variables that were used to train the model — `passenger_count` and `trip_distance`):

```
WITH (SELECT state FROM taxi_trips_model) AS model
SELECT
    fare_amount,
    evalMLMethod(model, passenger_count, trip_distance) AS prediction
FROM trips_test LIMIT 10;
```

In the output table, the resulting predictions are not too far from the actual data:

```
┌─fare_amount─┬─────────prediction─┐
│         8.5 │   9.20394916259993 │
│        13.5 │ 13.714548983540645 │
│        41.5 │  41.78716293769729 │
│          24 │ 22.230999964014817 │
│         8.5 │  9.018605114531145 │
│         6.5 │ 7.6682400731869595 │
│        19.5 │ 21.287184808552087 │
│          19 │ 20.377558220769647 │
│         7.5 │ 7.8100114849842495 │
│           9 │  8.954756513157905 │
└─────────────┴────────────────────┘
```

For more accurate predictions, you can select another set of hyperparameters.

JDBC Bridge

Elena Dvoryadkina

Contents

To Table of Contents

Installation
Configuration
Usage
- jdbc table function
- JDBC table engine
Example

ClickHouse JDBC Bridge allows ADQM to access data from external data sources that support the JDBC interface.

Installation

ClickHouse JDBC Bridge in ADQM is a component of the ADQMDB service that you can install in the same way as other components. The Add components article explains how to deploy ADQM components on cluster hosts from the ADCM interface.

> **IMPORTANT**
>
> If you install the ClickHouse JDBC Bridge component, add it to all hosts of the cluster where the Clickhouse Server component is installed.

Configuration

To be able to connect ADQM to an external data source via JDBC Bridge, define this data source under a certain name in the ***datasource.json*** field located on the ADQMDB service's configuration page (in the ***JDBC Bridge configuration*** section that is visible when the ***Show advanced*** option is activated).

By default, the JDBC Bridge configuration contains definitions of two data sources (`postgres13` for PostgreSQL and `ch-server` for ClickHouse):

```
[
    {
        "$schema": "./datasource.jschema",
        "postgres13": {
            "converter": {
                "mappings": [
                    {
                        "nativeType": "bool",
                        "toType": "String"
                    }
                ]
            },
            "driverUrls": [
                "drivers/postgresql/postgresql-jdbc.jar"
            ],
            "driverClassName": "org.postgresql.Driver",
            "jdbcUrl": "jdbc:postgresql://localhost/test",
            "username": "",
            "password": "",
            "initializationFailTimeout": 0,
            "minimumIdle": 0,
            "maximumPoolSize": 10
        }
    },
    {
        "$schema": "./datasource.jschema",
        "ch-server": {
            "driverUrls": [
                "drivers/clickhouse/clickhouse-jdbc.jar"
            ],
            "driverClassName": "ru.yandex.clickhouse.ClickHouseDriver",
            "jdbcUrl": "jdbc:clickhouse://ch-server:8123/system",
            "username": "default",
            "password": "",
            "initializationFailTimeout": 0,
            "minimumIdle": 0,
            "maximumPoolSize": 10
        }
    }
]
```

You can modify these configurations or use them as templates to describe new data sources, following the JSON format. For example, for the PostgreSQL data source configuration above, you need to set the real `username` and `password` values, and also change the `jdbcUrl` key value — replace `localhost` with the host on which PostgreSQL is installed and type a name of the required database instead of `test`.

Below is a general data source configuration template for convenience — you can copy and paste it into the ***datasource.json*** field, replacing placeholders with real data that corresponds to your external DBMS:

```
{
  "<datasource_name>": {
  "driverUrls": [
    "<datasource_jdbc_driver_url>"
  ],
  "jdbcUrl": "jdbc:<driver_name>://<host>:<port>/<database>",
  "username": "<username>",
  "password": "<password>"
  }
}
```

where:

- `<datasource_name>` — arbitrary name of a data source (you can pass this name to the jdbc table function or to the JDBC table engine to specify the data source);
- `<datasource_jdbc_driver_url>` — URL from which ADQM can download a JDBC driver for the specified DBMS;
- `<driver_name>` — JDBC driver name;
- `<host>:<port>` — host and port where the DBMS is installed;
- `<database>` — name of a database in the DBMS (this element is optional — if you skip it, specify a database explicitly each time you connect to the external data source from ADQM);
- `<username>` and `<password>` — credentials for access to the DBMS.

> **IMPORTANT**
>
> After you have set necessary data source configurations, do not forget to click ***Save*** and execute the ***Reconfig*** action with the enabled ***Restart*** option for the ADQMDB service to save changes and restart the service.

The following query returns a list of named data sources configured for JDBC Bridge:

```
SELECT name FROM jdbc('', 'show datasources');
```

The output can look like:

```
┌─name────────┐
│ ch-server   │
│ self_server │
│ postgres13  │
│ script      │
└─────────────┘
```

Usage

ADQM can interact with an external DBMS via the JDBC interface through the jdbc table function or JDBC table engine.

jdbc table function

The `jdbc` table function returns a table connected to an external data source via a JDBC driver. The basic syntax of a query that calls the `jdbc` function can be written as follows:

```
SELECT ... FROM jdbc('<datasource>', ['<external_database_name | schema>'], '<external_table_name | query>');
```

where:

- `<datasource>` — external data source name set in the JDBC Bridge configuration;
- `<external_database_name | schema>` — name of a database in the external data source or table schema (this parameter is optional);
- `<external_table_name | query>` — name of a table in the external database or query that selects data from the external table (the query should be in the native format of the specified external data source).

JDBC table engine

ADQM also provides the JDBC table engine for integration with external data sources via the JDBC interface. The general query syntax for creating a JDBC table is:

```
CREATE TABLE [IF NOT EXISTS] [<db_name>.]<jdbc_table_name> (<column_name> <column_type>, ...)
ENGINE = JDBC('<datasource>', '<external_database_name>', '<external_table_name | query>');
```

where:

- `<column_name> <column_type>, …` — column definitions for a table to be created (the schema of the JDBC table in ADQM should be consistent with the schema of the connected `<external_table_name>` table or with a table returned by the `<query>` query — i.e. names and order of columns should be the same, and column data types should be compatible);
- `<datasource>` — external data source name specified in the JDBC Bridge configuration;
- `<external_database_name>` — name of a database in the external data source (leave this parameter empty — `''`, if the data source configuration already includes the database name or you explicitly specify database for the source table within the query passed as the third parameter of the JDBC engine);
- `<external_table_name | query>` — name of a table in the `external_db_name` database or query that selects data from the external table (for example, `SELECT * FROM test_table WHERE column_1 = 1`).

Example

1. On the MariaDB server, create a table in the `mariadb_db` database :

   ```
   CREATE TABLE mariadb_table (id INT PRIMARY KEY, date DATETIME);
   ```

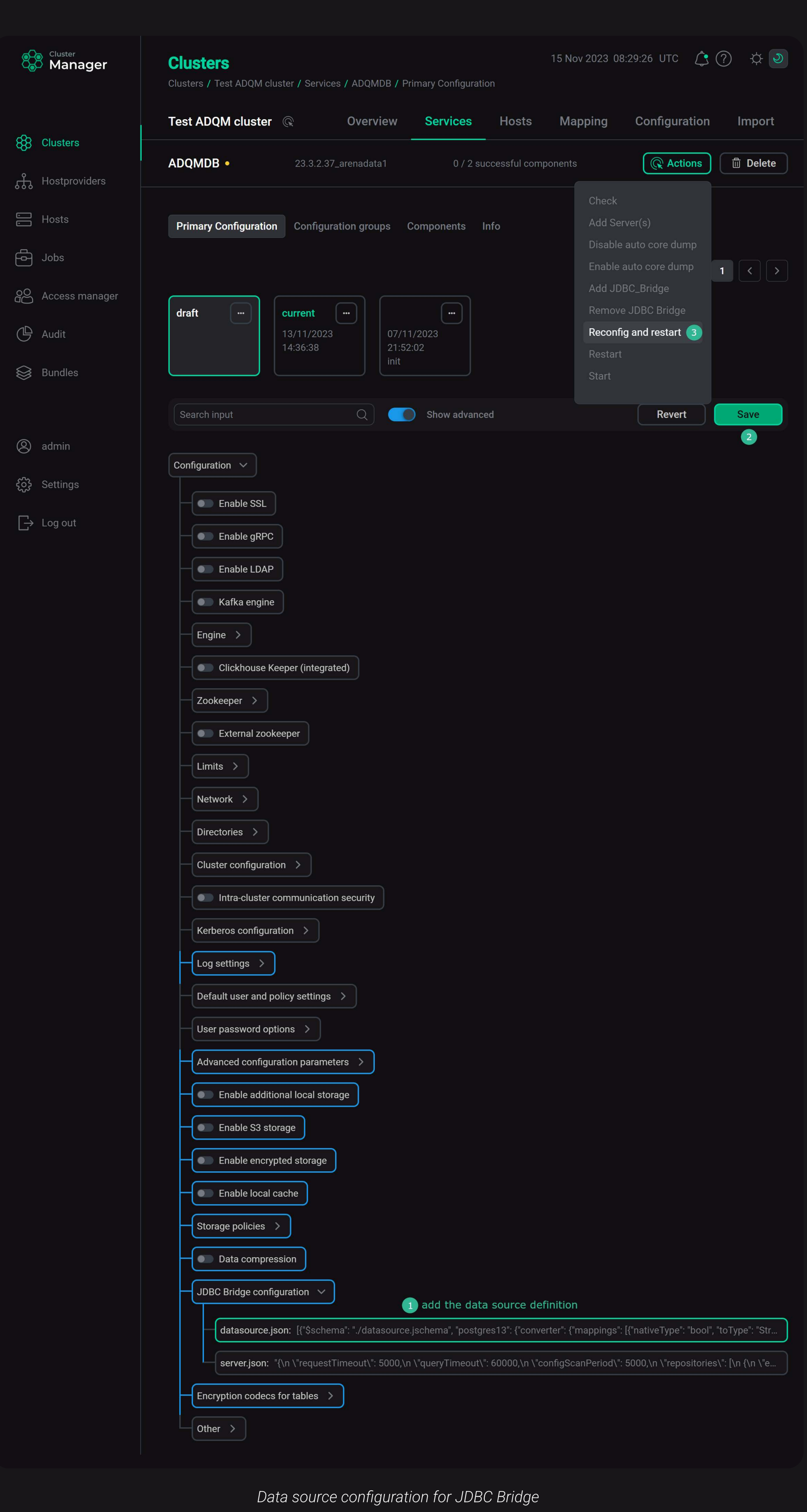
   Insert test data into the table:

   ```
   INSERT INTO mariadb_table VALUES (1, "2023-09-25 11:12:13"), (2, "2023-09-26 12:13:14"), (3, "2023-09-27 13:14:15");
   ```

2. In ADCM, add the following definition of the MariaDB data source to the ***datasource.json*** parameter of the JDBC Bridge configuration (replace `<host>:<port>` with the MariaDB server address, and `<user_name>` and `<user_password>` with a name and password of a user that accesses the MariaDB database):

   ```
   {
       "$schema": "./datasource.jschema",
       "mariadb10": {
           "driverUrls": [
           "https://repo1.maven.org/maven2/org/mariadb/jdbc/mariadb-java-client/2.7.4/mariadb-java-client-2.7.4.jar"
           ],
           "driverClassName": "org.mariadb.jdbc.Driver",
           "jdbcUrl": "jdbc:mariadb://<host>:<port>",
           "dataSource": {
           "user": "<user_name>",
           "password": "<user_password>"
           },
           "initializationFailTimeout": 0,
           "minimumIdle": 0,
           "maximumPoolSize": 10
       }
   }
   ```

   Click ***Save*** and execute the ***Reconfig*** action with the ***Restart*** option for the ADQMDB service to save information on the MariaDB data source in the JDBC Bridge configuration and restart the service.



Cluster Manager
Clusters
15 Nov 2023 08:29:26 UTC
Clusters / Test ADQM cluster / Services / ADQMDB / Primary Configuration
Test ADQM cluster
Overview
Services
Hosts
Mapping
Configuration
Import
Clusters
Hostproviders
Hosts
Jobs
Access manager
Audit
Bundles
admin
Settings
Log out
ADQMDB
23.3.2.37_arenadata1
0 / 2 successful components
Actions
Delete
Check
Add Server(s)
Disable auto core dump
Enable auto core dump
Add JDBC_Bridge
Remove JDBC Bridge
Reconfig and restart
3
Restart
Start
Primary Configuration
Configuration groups
Components
Info
draft
current
13/11/2023
14:36:38
07/11/2023
21:52:02
init
1
Search input
Show advanced
Revert
Save
2
Configuration
Enable SSL
Enable gRPC
Enable LDAP
Kafka engine
Engine
Clickhouse Keeper (integrated)
Zookeeper
External zookeeper
Limits
Network
Directories
Cluster configuration
Intra-cluster communication security
Kerberos configuration
Log settings
Default user and policy settings
User password options
Advanced configuration parameters
Enable additional local storage
Enable S3 storage
Enable encrypted storage
Enable local cache
Storage policies
Data compression
JDBC Bridge configuration
Encryption codecs for tables
Other

1 add the data source definition

datasource.json: [{"$schema": "./datasource.jschema", "postgres13": {"converter": {"mappings": [{"nativeType": "bool", "toType": "Str...

server.json: "{\n \"requestTimeout\": 5000,\n \"queryTimeout\": 60000,\n \"configScanPeriod\": 5000,\n \"repositories\": [\n {\n \"e...

Data source configuration for JDBC Bridge

3. Connect ADQM to the `mariadb_table` table of the external `mariadb_db` database through the jdbc table function or the JDBC table engine — in both cases, pass the name of the data source configured on the previous step as the first parameter.

   - jdbc table function:

     ```
     SELECT * FROM jdbc('mariadb10', 'mariadb_db', 'mariadb_table');
     ```

     ```
     ┌─id─┬────────────────date─┐
     │  1 │ 2023-09-25 11:12:13 │
     │  2 │ 2023-09-26 12:13:14 │
     │  3 │ 2023-09-27 13:14:15 │
     └────┴─────────────────────┘
     ```

     ```
     SELECT * FROM jdbc('mariadb10', 'SELECT * FROM mariadb_db.mariadb_table WHERE id >= 2');
     ```

     ```
     ┌─id─┬────────────────date─┐
     │  2 │ 2023-09-26 12:13:14 │
     │  3 │ 2023-09-27 13:14:15 │
     └────┴─────────────────────┘
     ```

     ```
     SELECT * FROM jdbc('mariadb10', 'test_dates DateTime', 'SELECT date as test_dates FROM mariadb_db.mariadb_table');
     ```

     ```
     ┌──────────test_dates─┐
     │ 2023-09-25 11:12:13 │
     │ 2023-09-26 12:13:14 │
     │ 2023-09-27 13:14:15 │
     └─────────────────────┘
     ```

   - JDBC table engine:

     ```
     CREATE TABLE jdbc_table (id Int32, date DateTime) ENGINE JDBC('mariadb10', 'mariadb_db', 'mariadb_table');
     ```

     ```
     SELECT * FROM jdbc_table;
     ```

     ```
     ┌─id─┬────────────────date─┐
     │  1 │ 2023-09-25 11:12:13 │
     │  2 │ 2023-09-26 12:13:14 │
     │  3 │ 2023-09-27 13:14:15 │
     └────┴─────────────────────┘
     ```

     ```
     CREATE TABLE jdbc_table_1 (test_IDs Int32, test_dates DateTime)
     ENGINE JDBC('mariadb10', '', 'SELECT id as test_IDs, date as test_dates FROM mariadb_db.mariadb_table');
     ```

     ```
     SELECT * FROM jdbc_table_1;
     ```

     ```
     ┌─test_IDs─┬──────────test_dates─┐
     │        1 │ 2023-09-25 11:12:13 │
     │        2 │ 2023-09-26 12:13:14 │
     │        3 │ 2023-09-27 13:14:15 │
     └──────────┴─────────────────────┘
     ```

     Make sure that JDBC Bridge allows also writing data from ADQM to an external database. To do this, add a new data record to the ADQM table:

     ```
     INSERT INTO jdbc_table VALUES (5, '2023-09-28 14:15:16');
     ```

     Then check the table content on the MariaDB server:

     ```
     SELECT * FROM mariadb_table;
     ```

     ```
     +----+---------------------+
     | id | date                |
     +----+---------------------+
     |  1 | 2023-09-25 11:12:13 |
     |  2 | 2023-09-26 12:13:14 |
     |  3 | 2023-09-27 13:14:15 |
     |  5 | 2023-09-28 14:15:16 |
     +----+---------------------+
     ```

Integration between ADQM and HDFS

Elena Dvoryadkina

Contents

To Table of Contents

- HDFS table engine
  - Example
- Table functions
  - Example
- Wildcards in path
- HDFS high availability (HA)
- Example of connecting ADQM to HDFS of a kerberized ADH cluster
  - Verify authentication and generate test data in HDFS
  - Create a principal for connecting ADQM to HDFS
  - Set up ADQM
  - Create a table in ADQM

For integration with the Hadoop Distributed File System (HDFS), ClickHouse implements the HDFS table engine and special table functions, which allow reading and writing HDFS data from ClickHouse. This article describes ways to use these tools on the example of ADQM and ADH (Arenadata Hadoop) that provides the HDFS service.

To integrate with HDFS, send ADQM queries to an ADH host with a NameNode in the Active state (in examples below, replace `<namenode_ip>` in URI to HDFS files with the IP address of the corresponding host that belongs to your ADH cluster). When connecting to a NameNode in the Standby status, ADQM can return the following error message:

```
Exception: Unable to connect to HDFS: Operation category READ is not supported in state standby.
```

HDFS table engine

The HDFS table engine allows you to manage data stored in HDFS files from ADQM. This table type supports parallel reads and writes, but has a number of limitations (replication, indexes, `ALTER` and `SELECT…SAMPLE` operations are not supported).

Below is the basic syntax of a query that creates an HDFS table:

```
CREATE TABLE <table_name> (<column_name> <column_type>, ...) ENGINE = HDFS(<URI>, <format>);
```

HDFS table engine parameters:

- `<URI>` — whole URI to access a file in HDFS. The file path part of the URI (the directory and file names) may include wildcards to address multiple files — in this case, the table will be read-only.
- `<format>` — format that ADQM will use to receive table data when executing `SELECT` queries and return data when executing `INSERT` queries (see available formats in the ***Input*** and ***Output*** columns of the Formats for Input and Output Data table, respectively).

When exporting data from an ADQM table to the HDFS file system, use the `hdfs_create_new_file_on_insert` and `hdfs_truncate_on_insert` parameters that allow you to control how data should be saved to HDFS files — whether new files should be created or existing files should be overwritten. By default, both options are set to `0`, and the following message is shown in an attempt to insert data into an ADQM table:

```
Exception: File with path <file_path> already exists. If you want to overwrite it, enable setting hdfs_truncate_on_insert,
if you want to create new file on each insert, enable setting hdfs_create_new_file_on_insert.
```

where `<file_path>` is an HDFS path to a file, based on which an HDFS table was created in ADQM.

In the ADQM configuration file, you can set up extended configuration for HDFS tables: specify global settings in the `<hdfs>` section and user-level settings in the `hdfs_<user_name>` section (replace `<user_name>` with a user name). See the list of parameters that can be configured for the HDFS table engine in the corresponding section of the HDFS article in the ClickHouse documentation.

Example

Create an HDFS table

1. In the *adqm_data_sources* HDFS directory, create the *test_1.txt* file with test data to be later inserted into ADQM tables:

   ```
   "one",1
   "two",2
   "three",3
   ```

   This file URI to access from ADQM is *hdfs://<namenode_ip>:8020/adqm_data_sources/test_1.txt*, where `<namenode_ip>` is an IP address of an ADH host on which an active NameNode runs.

2. Run the clickhouse-client console client and create a table based on the HDFS engine with a structure corresponding to data to be imported from HDFS:

   ```
   CREATE TABLE hdfs_table (name String, value UInt32) ENGINE=HDFS('hdfs://<namenode_ip>:8020/adqm_data_sources/test_1.txt',
   'CSV');
   ```

3. Select data from the table:

   ```
   SELECT * FROM hdfs_table;
   ```

   ```
   ┌─name──┬─value─┐
   │ one   │     1 │
   │ two   │     2 │
   │ three │     3 │
   └───────┴───────┘
   ```

Insert data into an HDFS table

1. Enable the `hdfs_create_new_file_on_insert` setting to create a new file in HDFS on each data insert in an ADQM table which is based on the HDFS engine (for example, if an HDFS table has been created for the *file_name.txt* file, names of new files will be generated according to the pattern — *file_name.1.txt*, *file_name.2.txt*, and so on):

   ```
   SET hdfs_create_new_file_on_insert = 1;
   ```

2. Insert data into the table by repeating the `INSERT INTO` query twice:

   ```
   INSERT INTO hdfs_table VALUES('four', 4);
   ```

   ```
   INSERT INTO hdfs_table VALUES('five', 5);
   ```

3. Check the content of the *adqm_data_sources* directory in HDFS:

   ```
   $ hdfs dfs -ls /adqm_data_sources
   ```

   As you can see, two new files (*test_1.1.txt* and *test_1.2.txt*) have been created in the directory:

   ```
   Found 3 items
   -rwxrwxrwx   3 clickhouse hadoop          9 2023-10-25 13:40 /adqm_data_sources/test_1.1.txt
   -rwxrwxrwx   3 clickhouse hadoop          9 2023-10-25 13:40 /adqm_data_sources/test_1.2.txt
   -rwxrwxrwx   3 clickhouse hadoop         23 2023-10-25 13:25 /adqm_data_sources/test_1.txt
   ```

   View the content of these files:

   ```
   $ hdfs dfs -cat /adqm_data_sources/test_1.1.txt
   ```

   ```
   "four",4
   ```

   ```
   $ hdfs dfs -cat /adqm_data_sources/test_1.2.txt
   ```

   ```
   "five",5
   ```

   On subsequent reading the table, you can make sure that a sample includes data from all files:

   ```
   SELECT * FROM hdfs_table;
   ```

   ```
   ┌─name──┬─value─┐
   │ one   │     1 │
   │ two   │     2 │
   │ three │     3 │
   └───────┴───────┘
   ┌─name─┬─value─┐
   │ five │     5 │
   └──────┴───────┘
   ┌─name─┬─value─┐
   │ four │     4 │
   └──────┴───────┘
   ```

4. Activate the `hdfs_truncate_on_insert` option and perform another data insertion into the table — this will overwrite the last data file :

   ```
   SET hdfs_truncate_on_insert = 1;
   ```

   ```
   INSERT INTO hdfs_table VALUES('six', 6);
   ```

5. Check the content of the *adqm_data_sources* directory in HDFS:

   ```
   $ hdfs dfs -ls /adqm_data_sources
   ```

   No new file was added, but the *test_1.2.txt* file was overwritten:

   ```
   Found 3 items
   -rwxrwxrwx   3 clickhouse hadoop          9 2023-10-25 13:40 /adqm_data_sources/test_1.1.txt
   -rwxrwxrwx   3 clickhouse hadoop          9 2023-10-25 13:45 /adqm_data_sources/test_1.2.txt
   -rwxrwxrwx   3 clickhouse hadoop         23 2023-10-25 13:25 /adqm_data_sources/test_1.txt
   ```

   ```
   $ hdfs dfs -cat /adqm_data_sources/test_1.2.txt
   ```

   ```
   "six",6
   ```

   If the `hdfs_truncate_on_insert` setting is enabled, new data will replace the current contents of an existing file in any case, regardless of the `hdfs_create_new_file_on_insert` value.

Table functions

ADQM also provides two table functions for integration with HDFS:

- **hdfs** — creates a table for reading/writing data in HDFS.
  Function syntax:

  ```
  hdfs(<URI>, <format>, <structure>)
  ```

  where:

  - `<URI>` — URI to a file in HDFS (when using the function in the read-only mode, you can specify the path to multiple files with wildcards);
  - `<format>` — format in which the function will receive or output data (see detailed information about formats to import and export ADQM/ClickHouse data in the Formats for Input and Output Data article of the ClickHouse documentation);
  - `<structure>` — table structure in the `column_name1 data_type1, column_name2 data_type2, …` format.

- **hdfsCluster** — allows processing HDFS files in parallel from multiple nodes in a specified cluster. On an initiator node, the function creates a connection to all nodes in the cluster, discloses asterisks in the HDFS file path, and dispatches each file dynamically. On a worker node, it requests the next task from the initiator and processes it. This is repeated until all tasks are completed.
  Function syntax:

  ```
  hdfs(<cluster_name>, <URI>, <format>, <structure>)
  ```

  where `<cluster_name>` is a name of a cluster used to build a set of addresses and connection parameters to remote and local servers, and other parameters are similar to parameters of the `hdfs` function.

Example

Read data from an HDFS file

Run the following query to read data from the *test_1.txt* HDFS file with the `hdfs` function:

```
SELECT * FROM hdfs('hdfs://<namenode_ip>:8020/adqm_data_sources/test_1.txt', 'CSV', 'column1 String, column2 UInt32');
```

```
┌─column1─┬─column2─┐
│ one     │       1 │
│ two     │       2 │
│ three   │       3 │
└─────────┴─────────┘
```

Write data to HDFS

When inserting data through the `hdfs` function with the `hdfs_truncate_on_insert` setting enabled, existing data in a file referenced by the function will be replaced with new data. In this case, the `clickhouse` user in HDFS should have permissions to modify this file.

For example, run the following query (the `hdfs_truncate_on_insert` value was set to `1` in the above example for the HDFS table engine):

```
INSERT INTO FUNCTION hdfs('hdfs://<namenode_ip>:8020/adqm_data_sources/test_1.txt', 'CSV', 'column1 String, column2 UInt32')
VALUES ('hdfs_func_test_string_1', 100);
```

In HDFS, check that the *test_1.txt* file has been updated:

```
$ hdfs dfs -cat /adqm_data_sources/test_1.txt
```

```
"hdfs_func_test_string_1",100
```

If the `hdfs_truncate_on_insert` parameter value is set to `0` and the `hdfs_create_new_file_on_insert` setting is enabled, a new file will be created when inserting data into HDFS through the `hdfs` function. For example, if the function inserts data into the *file_name.txt* file, it will create a file named *file_name.1.txt* to save new data or overwrite the *file_name.1.txt* file if it already exists.

Run the following query to export data to the *test_1.txt* file:

```
SET hdfs_truncate_on_insert = 0, hdfs_create_new_file_on_insert = 1;
```

```
INSERT INTO FUNCTION hdfs('hdfs://<namenode_ip>:8020/adqm_data_sources/test_1.txt', 'CSV', 'column1 String, column2 UInt32')
VALUES ('hdfs_func_test_string_2', 200);
```

As a result, new data will be written to the *test_1.1.txt* file:

```
$ hdfs dfs -cat /adqm_data_sources/test_1.1.txt
```

```
"hdfs_func_test_string_2",200
```

Wildcards in path

If an ADQM table should accept data from a bunch of HDFS files and be read-only, you can use the following wildcards to specify the path to multiple HDFS files in the `URI` parameter of the HDFS table engine or the `hdfs`/`hdfsCluster` table function:

- `*` — substitutes any number of any characters except `/`, including an empty string;
- `?` — substitutes any single character;
- `{first_string,second_string,third_one}` — substitutes any of strings `'first_string'`, `'second_string'`, `'third_one'` (you can also use numbers in this pattern — for example, `{1,3,5}`);
- `{n..m}` — substitutes any number in the `[n, m]` range.

Wildcard characters can appear in multiple path components (for example, in a directory name and in a file name). Only files that exist in the file system and match to the whole path pattern are processed. The list of files is determined during the `SELECT` (not `CREATE`) operation.

Example

1. Create multiple files with the following URIs in HDFS:

   - *hdfs://<namenode_ip>:8020/first_dir/file_1.txt*
   - *hdfs://<namenode_ip>:8020/first_dir/file_2.txt*
   - *hdfs://<namenode_ip>:8020/second_dir/file_3.txt*
   - *hdfs://<namenode_ip>:8020/second_dir/file_4.txt*

   Data in all files should comply with the format and scheme that will be specified in queries connecting from ADQM.

2. You can import data from all the above files into ADQM with the `hdfs` function in one of the following ways:

   ```
   SELECT * FROM hdfs('hdfs://<namenode_ip>:8020/{first,second}_dir/file_{1..4}.txt', 'CSV', 'column1 String, column2 UInt32');
   ```

   ```
   SELECT * FROM hdfs('hdfs://<namenode_ip>:8020/{first,second}_dir/file_?.txt', 'CSV', 'column1 String, column2 UInt32');
   ```

   The following query reads data from all files in the *first_dir* and *second_dir* directories:

   ```
   SELECT * FROM hdfs('hdfs://<namenode_ip>:8020/{first,second}_dir/*', 'CSV', 'column1 String, column2 UInt32');
   ```

3. If the list of files contains a number range with leading zeros in file names, you can use the general pattern `{0n..0m}`, or the separate pattern `{n..m}` (or the sign `?`) for each digit in a file name. For example, you can create a table with data from files named *file_000.txt*, *file_001.txt*, …, *file_999.txt* as follows:

   ```
   CREATE TABLE hdfs_table_1 (column1 String, column2 UInt32)
   ENGINE=HDFS('hdfs://<namenode_ip>:8020/<dir_name>/file_{001..999}.txt', 'CSV');
   ```

   ```
   CREATE TABLE hdfs_table_2 (column1 String, column2 UInt32)
   ENGINE=HDFS('hdfs://<namenode_ip>:8020/<dir_name>/file_{0..9}{0..9}{0..9}.txt', 'CSV');
   ```

   ```
   CREATE TABLE hdfs_table_3 (column1 String, column2 UInt32)
   ENGINE=HDFS('hdfs://<namenode_ip>:8020/<dir_name>/file_???.txt', 'CSV');
   ```

HDFS high availability (HA)

If the high availability (HA) mode is enabled for HDFS on an ADH cluster, perform the following configuration:

1. Copy the /etc/hadoop/conf/hdfs-site.xml file from an ADH host to the */etc/clickhouse-server/* folder on an ADQM host.
2. Add the following setting to the */etc/clickhouse-server/config.xml* configuration file of ClickHouse:

   ```
   <hdfs>
       <libhdfs3_conf>/etc/clickhouse-server/hdfs-site.xml</libhdfs3_conf>
   </hdfs>
   ```

3. Restart the ClickHouse server:

   ```
   $ sudo systemctl restart clickhouse-server
   ```

Now, when connecting to HDFS via the HDFS table engine or table functions, you can use a value of the `dfs.nameservices` tag in the *hdfs-site.xml* file instead of `<namenode_ip>:8020` in an HDFS file URI. For example, if the value of the `dfs.nameservices` tag is `adhcluster`, then a query creating the hdfs_table table and the hdfs function call from examples above can be modified as follows:

```
CREATE TABLE hdfs_table (name String, value UInt32) ENGINE=HDFS('hdfs://adhcluster/adqm_data_sources/test_1.txt', 'CSV');
```

```
SELECT * FROM hdfs('hdfs://adhcluster/adqm_data_sources/test_1.txt', 'CSV', 'column1 String, column2 UInt32');
```

Example of connecting ADQM to HDFS of a kerberized ADH cluster

Verify authentication and generate test data in HDFS

1. Configure MIT Kerberos KDC and kerberize your ADH cluster according to the instructions from the MIT Kerberos section of the ADH documentation. Use `ADREALM.IO` as a realm name.
2. Verify that Kerberos authentication in HDFS works correctly. To do this, create a ticket-granting ticket (TGT) for the `hdfs/<adh_node_hostname>@ADREALM.IO` principal on an ADH host:

   ```
   $ sudo -u hdfs kinit -k -t /etc/security/keytabs/hdfs.service.keytab hdfs/<adh_node_hostname>@ADREALM.IO
   ```

   Then, run the command, for example, to view the contents of the HDFS root directory:

   ```
   $ sudo -u hdfs hdfs dfs -ls /
   ```

   If Kerberos authentication is successful, the current contents of the root directory will be displayed on the screen.

3. Create a new HDFS directory (*/adqm_test_kerb*) and add a file (*test_2.txt*) to it with some data rows to test ADQM connection via Kerberos, for example:

   ```
   "ten",10
   "twenty",20
   "thirty",30
   ```

4. To allow inserting new data on the ADQM side, set this directory owner to the `adqm_user` user, under which ADQM will connect to HDFS (a principal of this user will be created in the next step):

   ```
   $ sudo -u hdfs hdfs dfs -chown -R adqm_user /adqm_test_kerb
   ```

> **IMPORTANT**
>
> Due to the libhdfs3 library limitations, ADQM requires setting up the following HDFS service parameters in order to access HDFS directories of a kerberized ADH cluster:
>
> - the `hadoop.rpc.protection` parameter in the *core-site.xml* file should not be set to `privacy`;
> - the `dfs.encrypt.data.transfer` parameter in the *hdfs-site.xml* file should be set to `false`.
>
> You can change the values of these parameters on the HDFS service's configuration page in the ADCM interface, after which it is required to run the ***Restart*** action for the service.

Create a principal for connecting ADQM to HDFS

1. On the host where the MIT Kerberos KDC is deployed, create a principal for the `adqm_user` user that will be used by ADQM to connect to HDFS:

   ```
   $ sudo kadmin.local -q "add_principal -randkey adqm_user@ADREALM.IO"
   ```

2. Create a keytab file for this principal:

   ```
   $ sudo kadmin.local -q "xst -kt /tmp/adqm_user.keytab adqm_user@ADREALM.IO"
   ```

3. Copy the */tmp/adqm_user.keytab* file to ADQM hosts (for example, to the */tmp* directory).
   On an ADQM host, set `clickhouse` as the owner of the keytab file and restrict access to this file for security purposes:

   ```
   $ sudo chown clickhouse:clickhouse /tmp/adqm_user.keytab
   $ sudo chmod 0600 /tmp/adqm_user.keytab
   ```

Set up ADQM

1. In the */etc/clickhouse-server/config.d* directory, create a file (for example, *hdfs_kerberos.xml*) with the following settings:

   ```
   <?xml version="1.0"?>
   <clickhouse>
       <hdfs>
           <hadoop_security_authentication>kerberos</hadoop_security_authentication>
           <hadoop_kerberos_keytab>/tmp/adqm_user.keytab</hadoop_kerberos_keytab>
           <hadoop_kerberos_principal>adqm_user@ADREALM.IO</hadoop_kerberos_principal>
           ...
       </hdfs>
   </clickhouse>
   ```

   Execute the ***Reconfig*** action with the ***Restart*** option for the ADQMDB service.

2. Install the following packages on an ADQM host:

   ```
   $ sudo yum install krb5-libs krb5-workstation
   ```

3. Edit the */etc/krb5.conf* file as follows (replace `<kdc_host>` with the name of the host where the MIT Kerberos KDC is installed):

   ```
   # Configuration snippets may be placed in this directory as well
   includedir /etc/krb5.conf.d/

   [logging]
    default = FILE:/var/log/krb5libs.log
    kdc = FILE:/var/log/krb5kdc.log
    admin_server = FILE:/var/log/kadmind.log

   [libdefaults]
    dns_lookup_realm = false
    ticket_lifetime = 24h
    forwardable = true
    rdns = false
    pkinit_anchors = FILE:/etc/pki/tls/certs/ca-bundle.crt
    default_realm = ADREALM.IO

   [realms]
    ADREALM.IO = {
     admin_server = <kdc_host>
     kdc = <kdc_host>
    }

   [domain_realm]
    .adrealm.io = ADREALM.IO
    adrealm.io = ADREALM.IO
   ```

Create a table in ADQM

1. On an ADQM host, open clickhouse-client and create a table based on the HDFS engine:

   ```
   CREATE TABLE hdfs_table_kerb (name String, value UInt32)
   ENGINE=HDFS('hdfs://<namenode_ip>:8020/adqm_test_kerb/test_2.txt', 'CSV');
   ```

2. Read data from the table:

   ```
   SELECT * FROM hdfs_table_kerb;
   ```

   ```
      ┌─name───┬─value─┐
   1. │ ten    │    10 │
   2. │ twenty │    20 │
   3. │ thirty │    30 │
      └────────┴───────┘
   ```

3. Insert a new row into the table:

   ```
   SET hdfs_create_new_file_on_insert = 1;
   ```

   ```
   INSERT INTO hdfs_table_kerb VALUES('forty', 40);
   ```

4. Check the contents of the *adqm_test_kerb* directory in HDFS:

   ```
   $ sudo -u hdfs hdfs dfs -ls /adqm_test_kerb
   ```

   ```
   -rwxrwxrwx   3 adqm_user hadoop         12 2025-02-24 09:04 /adqm_test_kerb/test_2.1.txt
   -rw-r--r--   3 adqm_user hadoop         36 2025-02-24 08:53 /adqm_test_kerb/test_2.txt
   ```

   Make sure the new *test_2.1.txt* file contains the row inserted into the HDFS table on the ADQM side:

   ```
   $ sudo -u hdfs hdfs dfs -cat /adqm_test_kerb/test_2.1.txt
   ```

   ```
   "forty",40
   ```

Read Hive data from ADQM

Elena Dvoryadkina

Contents

To Table of Contents

Create a Hive table

HDFS high availability (HA)

Example

To read data from remote Hive tables via `SELECT` queries, ClickHouse provides the Hive table engine. It currently supports input formats as follows:

- Text — supports only simple scalar column types except `binary`;
- ORC — supports simple scalar columns types except `char` and the only `array` complex type;
- Parquet — supports all simple scalar columns types and the only `array` complex type.

This article describes how to work with the Hive table engine on the example of ADQM and ADH (Arenadata Hadoop) that provides the Hive service.

Create a Hive table

The basic syntax of a query that creates a Hive table in ADQM is:

```
CREATE TABLE <table_name> (<column_name> <column_type> [ALIAS <expr>], ...)
ENGINE = Hive('thrift://<host>:<port>', '<hive_database>', '<hive_table>')
PARTITION BY <expr>;
```

Hive table engine parameters:

- `thrift://<host>:<port>` — address of an ADH host where the Hive Metastore component is installed;
- `<hive_database>` — name of a remote Hive database;
- `<hive_table>` — name of a remote Hive table.

An ADQM table's structure can differ from the structure of an original Hive table you are connecting to:

- Column names in the ADQM table should be the same as in the original Hive table, but you can use just some of these columns and in any order. In the ADQM table, you can also use the `ALIAS` expression to specify columns to be calculated from other columns.
- Column types in the ADQM table should be the same as types in the original Hive table.
- The `PARTITION BY` key of the ADQM table should be consistent with the `PARTITIONED BY` key of the original Hive table, and columns used in the `PARTITION BY` expression should be defined in the table structure.

HDFS high availability (HA)

If the high availability (HA) mode for HDFS is enabled on an ADH cluster, perform the following configuration to be able to connect to Hive tables from ADQM:

1. Copy the */etc/hadoop/conf/hdfs-site.xml* file from an ADH host to the */etc/clickhouse-server/* folder on an ADQM host.
2. Add the following setting to the */etc/clickhouse-server/config.xml* configuration file of ClickHouse:

   ```
   <hdfs>
       <libhdfs3_conf>/etc/clickhouse-server/hdfs-site.xml</libhdfs3_conf>
   </hdfs>
   ```

3. Restart the ClickHouse server:

   ```
   $ sudo systemctl restart clickhouse-server
   ```

An internal name of an HDFS cluster specified as internal nameservice should be in lower case. Otherwise, when attempting to retrieve data from a remote Hive table, ADQM may return the following error message:

```
Unable to connect to HDFS: InvalidParameter: Cannot parse URI: hdfs://<internal_nameservice_in_lower_case>, missing port or
invalid HA configuration
Caused by: HdfsConfigNotFound: Config key: dfs.ha.namenodes.<internal_nameservice_in_lower_case> not found. (NETWORK_ERROR)
```

where `<internal_nameservice_in_lower_case>` is an internal nameservice specified in the `dfs.internal.nameservices` parameter of the *hdfs-site.xml* configuration file, but in lower case.

Example

The example below shows how use the Hive table engine in ADQM to select data from a remote Hive table located on an ADH cluster (high availability for HDFS is not enabled).

1. On an ADH host with the Hive HiveServer2 component installed, create a Hive table (for example, `test_table`) in the `default` database:

   ```
   CREATE TABLE test_table(`id` int, `value` int, `name` string) PARTITIONED BY (`date` date);
   ```

2. Insert test data into the table:

   ```
   INSERT INTO test_table VALUES (0, 1, 'one', '2023-11-23');
   ```

3. On an ADQM host, run the clickhouse-client console client and create a table based on the Hive engine (replace `<adh_host>` with an IP address of an ADH host with the Hive Metastore component):

   ```
   CREATE TABLE hive_table(id Int8, value Int16, name String, date Date)
   ENGINE = Hive('thrift://<adh_host>:9083', 'default', 'test_table')
   PARTITION BY date;
   ```

4. Select data from the table:

   ```
   SELECT * FROM hive_table;
   ```

   ```
   ┌─id─┬─value─┬─name─┬───────date─┐
   │  0 │     1 │ one  │ 2023-11-23 │
   └────┴───────┴──────┴────────────┘
   ```

Integration between ADQM and Kafka

Elena Dvoryadkina

Contents

To Table of Contents

- Overview
- Kafka table engine
  - Create a Kafka table
  - Extended configuration
  - Kerberos support
  - Virtual columns
- Common operations
  - Stop and restart message consumption
  - Modify a target table
  - Modify a Kafka table
- Example
  - Create a Kafka topic
  - Create ADQM tables for integration with Kafka
  - Read data from Kafka
  - Insert data from ADQM to Kafka
  - Change the target table
- Example of connecting ADQM to Kafka with Kerberos SASL
  - Set up ADS
  - Create a topic in ADS
  - Create a principal for connecting ADQM to Kafka
  - Set up ADQM
  - Create tables in ADQM
  - Check the connection between ADQM and Kafka

ClickHouse supports integration with Apache Kafka through a special table engine — Kafka. It allows you to:

- subscribe to Kafka topics to load streaming data from Kafka to ClickHouse (it is the most common use case) or publish data from ClickHouse to Kafka;
- create fault-tolerant storage;
- process streams as they become available.

This article explains how to use the Kafka table engine on the example of ADQM and ADS (Arenadata Streaming) that provides the Kafka service.

Overview

The Kafka table engine allows ClickHouse to read data from a Kafka topic directly and track message streaming, but each message can be retrieved only once from a Kafka-engine table. When data is queried from a Kafka table, it is considered already consumed from the queue. In this case, an offset for a consumer is increased, and data cannot be re-read without resetting these offsets. Thus, it is not recommended to select data from a Kafka table directly (via the `SELECT` query) — use a materialized view instead. This approach requires you to create the following tables on the ClickHouse side:

- Table based on the Kafka engine to be a Kafka consumer that reads data flow.
- Target table with the desired structure to be a persistent storage of data received from Kafka. Typically, a target table is implemented based on an engine from the MergeTree family.
- Materialized view to collect data from the Kafka table in the background (it will receive new records in blocks when a trigger fires on inserting data into the Kafka table), convert it into the required format, and transfer it to the previously created target table.
  Sometimes it is necessary to apply various transformations to data coming from Kafka — for example, to store raw and aggregated data. In this case, you can bind multiple materialized views to one Kafka table to store data with different levels of detail across multiple tables.

To write data from ClickHouse to Kafka, insert it directly into a Kafka-engine table.

Kafka table engine

Create a Kafka table

The basic syntax of a query that creates a Kafka table in ADQM is:

```
CREATE TABLE <table_name> (<column_name> <column_type> [ALIAS <expr>], ...)
ENGINE = Kafka()
SETTINGS
    kafka_broker_list = '<host_name>:9092,...',
    kafka_topic_list = '<topic_name>,...',
    kafka_group_name = '<consumer_group_name>',
    kafka_format = '<data_format>'[,]
    [kafka_schema = '',]
    [kafka_num_consumers = <N>,]
    [kafka_max_block_size = 0,]
    [kafka_skip_broken_messages = <N>,]
    [kafka_commit_every_batch = 0,]
    [kafka_client_id = '',]
    [kafka_poll_timeout_ms = 0,]
    [kafka_poll_max_batch_size = 0,]
    [kafka_flush_interval_ms = 0,]
    [kafka_thread_per_consumer = 0,]
    [kafka_handle_error_mode = 'default',]
    [kafka_commit_on_select = false,]
    [kafka_max_rows_per_message = 1];
```

The Kafka table engine has the following required parameters. You can specify these parameters in parentheses when creating a table (see an example).

kafka_broker_list	List of Kafka brokers separated by commas
kafka_topic_list	List of Kafka topics separated by commas
kafka_group_name	Group of Kafka consumers. Offsets to read messages are tracked for each group separately. If you want to avoid duplicate messages in your cluster, use the same consumer group name for all Kafka tables
kafka_format	Kafka message format. For this parameter, use the same format names as for the `FORMAT` clause in `INSERT` and `SELECT` queries — see available formats in the Formats for Input and Output Data article of the ClickHouse documentation

Optional parameters

kafka_schema	Use this parameter if the format requires a schema definition. For example, CapnProto requires the path to the schema file and the name of the root `schema.capnp:Message` object
kafka_num_consumers	Number of consumers per table. Default value is `1`. Specify more consumers if the throughput of one consumer is insufficient. The total number of consumers should not exceed the number of partitions in a topic (only one consumer can be assigned to one partition) and should not be greater than the number of cores on a server where ClickHouse is deployed
kafka_max_block_size	Maximum batch size (in messages) for a poll. Default value is equal to a value of the max_insert_block_size parameter
kafka_skip_broken_messages	Maximum number of allowed schema-incompatible messages per block. If `kafka_skip_broken_messages = N`, the engine skips `N` Kafka messages that cannot be parsed (one message corresponds to one data row). Default value is `0`
kafka_commit_every_batch	Specifies whether to commit each consumed and handled batch instead of the whole block. Default value is `0`
kafka_client_id	Client identifier. The parameter is empty by default
kafka_poll_timeout_ms	Timeout for a single poll from Kafka. Default value is equal to a value of the stream_poll_timeout_ms parameter
kafka_poll_max_batch_size	Maximum number of messages to be polled in a single Kafka poll. Default value is equal to a value of the max_block_size parameter
kafka_flush_interval_ms	Timeout for flushing data from Kafka. Default value is equal to a value of the stream_flush_interval_ms parameter
kafka_thread_per_consumer	Specifies whether to provide an independent thread for each consumer. When this option is enabled, each consumer flushes data independently, in parallel (otherwise — rows from several consumers are squashed to form one block). Default value is `0`
kafka_handle_error_mode	Specifies how to handle errors for the Kafka table engine. Possible values: - `default` — an exception will be thrown if a message cannot be parsed; - `stream` — an exception text and raw message will be saved in the `_error` and `_raw_message` virtual columns
kafka_commit_on_select	Specifies whether to commit messages when executing the `SELECT` query. Default value is `false`
kafka_max_rows_per_message	Maximum number of rows in one Kafka message for row-based formats. Default value is `1`

TIP

For production environments, it is recommended to pass parameters using named collections.

Extended configuration

In the ClickHouse configuration file (*config.xml*), you can specify additional settings that the Kafka table engine supports:

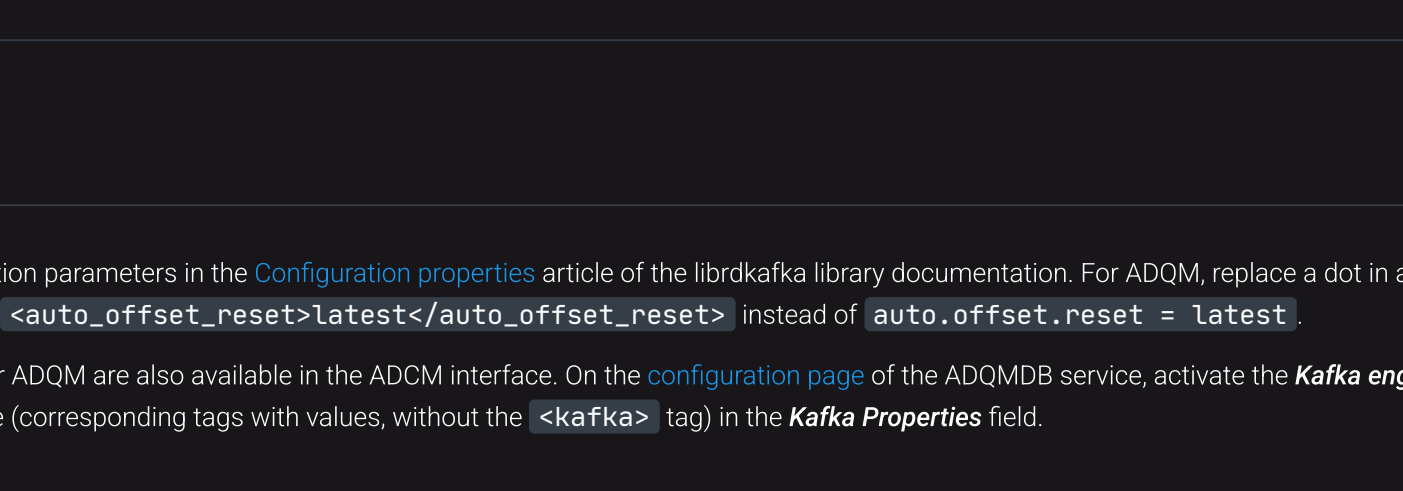
- global settings in the `<kafka>` section;
- topic-level settings in the `<kafka_topic>` section inside the `<kafka>` tag (specify a topic name via the `<name>` parameter).

For example, the following parameter allows you to enable debug logging for the Kafka library (librdkafka):

```
<kafka>
   <debug>all</debug>
</kafka>
```

See a list of possible configuration parameters in the Configuration properties article of the librdkafka library documentation. For ADQM, replace a dot in a parameter name with an underscore — for example, use `<auto_offset_reset>latest</auto_offset_reset>` instead of `auto.offset.reset = latest`.

Settings of the Kafka engine for ADQM are also available in the ADCM interface. On the configuration page of the ADQMDB service, activate the ***Kafka engine*** option and list required parameters of the Kafka engine (corresponding tags with values, without the `<kafka>` tag) in the ***Kafka Properties*** field.



Kafka Properties: <debug>all</debug> <auto_offset_reset>latest</auto_offset_reset>

Extended configuration of the Kafka engine

Click ***Save*** and execute the ***Reconfig*** action with the ***Restart*** option for the ADQMDB service — after the action is completed, the `<kafka>` section with the specified parameters will be added to the *config.xml* file.

Kerberos support

For integration with Kerberos-aware Kafka, set the `security_protocol` parameter to `sasl_plaintext`. It is enough if the Kerberos ticket-granting ticket is obtained and cached by the operating system. ClickHouse can maintain Kerberos credentials using a keytab file. Use also the `sasl_kerberos_service_name`, `sasl_kerberos_keytab`, and `sasl_kerberos_principal` parameters to set up Kerberos authentication — see the example below.

Virtual columns

It can be useful to track metadata of Kafka messages being loaded into ClickHouse. For example, you may need to have coordinates of a consumed message or to know how many messages were received from a certain topic or partition. For this purpose, the Kafka engine provides the following virtual columns.

Column name	Data type	Description
_topic	LowCardinality(String)	Kafka topic
_key	String	Message key
_offset	UInt64	Message offset
_timestamp	Nullable(DateTime)	Message timestamp
_timestamp_ms	Nullable(DateTime64(3))	Message timestamp in milliseconds
_partition	UInt64	Partition of a Kafka topic
_headers.name	Array(String)	Array of message header keys
_headers.value	Array(String)	Array of message header values
_error	String	Text of an exception thrown when the parsing of a message failed. The column is filled if the Kafka engine's `kafka_handle_error_mode` parameter is set to `stream` and an exception has occurred during parsing a message (if the message was parsed successfully, the column is empty)
_raw_message	String	Raw message that could not be parsed successfully. The column is filled if the Kafka engine's `kafka_handle_error_mode` parameter is set to `stream` and an exception has occurred during parsing a message (if the message was parsed successfully, the column is empty)

To get values of virtual columns in ADQM, add the corresponding columns to a target table and specify them in the `SELECT` clause of a materialized view — see the example below. There is no need to create virtual columns in a Kafka table as they are available automatically.

Common operations

Stop and restart message consumption

To stop receiving messages from a topic, detach a Kafka-engine table:

```
DETACH TABLE <kafka_engine_table>;
```

Detaching a Kafka table does not affect an offset of the consumer group. To restart consumption and continue it from the previous offset, reattach the table:

```
ATTACH TABLE <kafka_engine_table>;
```

Modify a target table

If it is needed to change a target table, the following sequence of steps is recommended:

1. Detach the Kafka table (`DETACH TABLE`).
2. Change the target table (`ALTER TABLE`).
3. Delete the materialized view to avoid discrepancies between the modified target table and data from the view (`DROP VIEW`).
4. Re-attach the Kafka table (`ATTACH TABLE`).
5. Re-create the materialized view (`CREATE MATERIALIZED VIEW`).

Modify a Kafka table

To change settings for a Kafka-engine table, drop it and re-create it with new settings. In this case, there is no need to modify the materialized view — message consumption will resume once the Kafka table is re-created.

Example

Create a Kafka topic

Connect to any ADS host where the Kafka service is installed. Go to the root directory with scripts for executing commands in the Kafka environment (*.sh* files) and follow the steps below.

1. Create the `topic-kafka-to-adqm` topic via the following command (replace `<kafka_host>` with a name of an ADS host where the topic is created):

   ```
   $ bin/kafka-topics.sh --create --topic topic-kafka-to-adqm --bootstrap-server <kafka_host>:9092
   ```

   The result:

   ```
   Created topic topic-kafka-to-adqm.
   ```

   NOTE

   For more information on creating and reading Kafka topics, see Quick start with Kafka in the ADS documentation.

2. Write test messages to the topic as follows.
   Run the command that starts the message recording mode:

   ```
   $ bin/kafka-console-producer.sh --topic topic-kafka-to-adqm --bootstrap-server <kafka_host>:9092
   ```

   On the next line after entering the command, enter messages — each message on a new line (use `Enter` to move to a new line):

   ```
   >1,"one"
   >2,"two"
   >3,"three"
   ```

   To exit the message recording mode, go to the next line after writing the last message and press `Ctrl+C`.

Create ADQM tables for integration with Kafka

On an ADQM host, run the clickhouse-client console client and follow the steps below.

1. Create a table based on the Kafka engine:

   ```
   CREATE TABLE kafka_table (id Int32, name String) ENGINE = Kafka ('<kafka_host>:9092', 'topic-kafka-to-adqm', 'clickhouse',
   'CSV');
   ```

2. Create a target table to store data from Kafka:

   ```
   CREATE TABLE kafka_data (id Int32, name String) ENGINE = MergeTree() ORDER BY id;
   ```

3. Create a materialized view that will receive data from the Kafka table and put it into the target MergeTree table:

   ```
   CREATE MATERIALIZED VIEW kafka_data_mv TO kafka_data AS SELECT id, name FROM kafka_table;
   ```

   Now the materialized view is connected to the Kafka-engine table — it starts reading Kafka data and inserting corresponding data rows into the target table. This process will continue indefinitely — all subsequent insertions of messages into the Kafka topic will be consumed.

Read data from Kafka

1. Verify that data from Kafka is inserted into the target table:

   ```
   SELECT * FROM kafka_data;
   ```

   ```
   ┌─id─┬─name──┐
   │  1 │ one   │
   │  2 │ two   │
   │  3 │ three │
   └────┴───────┘
   ```

2. Insert more messages into the Kafka topic (on the ADS host):

   ```
   $ bin/kafka-console-producer.sh --topic topic-kafka-to-adqm --bootstrap-server <kafka_host>:9092
   ```

   ```
   >4,"four"
   >5,"five"
   ```

3. Check that new data is in the ADQM table:

   ```
   SELECT * FROM kafka_data;
   ```

   ```
   ┌─id─┬─name──┐
   │  1 │ one   │
   │  2 │ two   │
   │  3 │ three │
   │  4 │ four  │
   │  5 │ five  │
   └────┴───────┘
   ```

Insert data from ADQM to Kafka

1. On the ADQM host, insert data into the Kafka-engine table:

   ```
   INSERT INTO kafka_table VALUES (6, 'six');
   ```

2. On the ADS host, check that the corresponding message is sent to the Kafka topic:

   ```
   $ bin/kafka-console-consumer.sh --topic topic-kafka-to-adqm --from-beginning --bootstrap-server <kafka_host>:9092
   ```

   ```
   1,"one"
   2,"two"
   3,"three"
   4,"four"
   5,"five"
   6,"six"
   ```

3. Ensure also that new data is written to the target ADQM table with Kafka data:

   ```
   SELECT * FROM kafka_data;
   ```

   ```
   ┌─id─┬─name──┐
   │  1 │ one   │
   │  2 │ two   │
   │  3 │ three │
   │  4 │ four  │
   │  5 │ five  │
   │  6 │ six   │
   └────┴───────┘
   ```

Change the target table

To include meta information about Kafka messages (for example, the topic name and message offset) into the target table, make the following changes:

1. Detach the Kafka table:

   ```
   DETACH TABLE kafka_table;
   ```

2. Add the `topic` and `offset` columns to the target table:

   ```
   ALTER TABLE kafka_data
       ADD COLUMN topic String,
       ADD COLUMN offset UInt64;
   ```

3. Delete the materialized view:

   ```
   DROP VIEW kafka_data_mv;
   ```

4. Re-attach the Kafka-engine table:

   ```
   ATTACH TABLE kafka_table;
   ```

5. Re-create the materialized view:

   ```
   CREATE MATERIALIZED VIEW kafka_data_mv TO kafka_data AS SELECT id, name, _topic as topic, _offset as offset FROM kafka_table;
   ```

Now, for each newly consumed message, the `topic` and `offset` columns of the target ADQM table will contain a name of the Kafka topic and the message offset in the partition. To check this:

1. Insert a new message into the Kafka topic on the ADS host:

   ```
   $ bin/kafka-console-producer.sh --topic topic-kafka-to-adqm --bootstrap-server <kafka_host>:9092
   ```

   ```
   >7,"seven"
   ```

2. Query data from the `kafka_data` table in ADQM:

   ```
   SELECT * FROM kafka_data;
   ```

   ```
   ┌─id─┬─name──┐
   │  1 │ one   │
   │  2 │ two   │
   │  3 │ three │
   │  4 │ four  │
   │  5 │ five  │
   │  6 │ six   │
   └────┴───────┘
   ┌─id─┬─name──┬─topic──────────────┬─offset─┐
   │  7 │ seven │ topic-kafka-to-adqm │      6 │
   └────┴───────┴─────────────────────┴────────┘
   ```

Example of connecting ADQM to Kafka with Kerberos SASL

Set up ADS

1. Configure MIT Kerberos KDC on a separate host and kerberize your ADS cluster according to the instructions from the MIT Kerberos article. Use `ADS-KAFKA.LOCAL` as a realm name.
2. Ensure that Kafka authentication is configured correctly by following the steps from the Check installed Kerberos SASL section.
3. Create a user principal for authentication in Kafka (for example, `ads_user`) by running the following command on the host where the MIT Kerberos KDC is deployed:

   ```
   $ sudo kadmin.local -q "add_principal -pw PASSWORD ads_user@ADS-KAFKA.LOCAL"
   ```

4. On the ADS side, configure the JAAS file (*/tmp/client.jaas*) and *.properties* configuration file (*/tmp/client.properties*) for the user as described in the corresponding sections of the Use MIT Kerberos in Kafka article.

Create a topic in ADS

Create a test topic to which ADQM will connect to read/write messages. To do this, follow the steps below on any ADS host where the Kafka service is installed:

1. Create a ticket-granting ticket (TGT) for the `ads_user` principal:

   ```
   $ kinit -p ads_user@ADS-KAFKA.LOCAL
   ```

   Enter the password specified during the principal creation — in this example, `PASSWORD`.

2. Export the generated *client.jaas* file as a JVM option for the given user using the `KAFKA_OPTS` environment variable:

   ```
   $ export KAFKA_OPTS="-Djava.security.auth.login.config=/tmp/client.jaas"
   ```

3. Create a topic (for example, `topic-adqm-to-kafka-kerberos`) specifying the path to the created *client.properties* file:

   ```
   $ /usr/lib/kafka/bin/kafka-topics.sh \
     --create --topic topic-adqm-to-kafka-kerberos  \
     --bootstrap-server <kafka_host1>:9092,<kafka_host2>:9092,<kafka_host3>:9092  \
     --command-config /tmp/client.properties
   ```

   Replace `<kafka_host1>`, `<kafka_host2>`, `<kafka_host3>` with names of ADS hosts where the topic should be created.

4. Write some messages to the topic and also specify the path to the *client.properties* file:

   ```
   $ /usr/lib/kafka/bin/kafka-console-producer.sh  \
     --topic topic-adqm-to-kafka-kerberos   \
     --bootstrap-server <kafka_host1>:9092,<kafka_host2>:9092,<kafka_host3>:9092  \
     --producer.config /tmp/client.properties
   ```

   ```
   >1,"one"
   >2,"two"
   >3,"three"
   ```

   To exit the message recording mode, go to the next line after writing the last message and press `Ctrl+C`.

Create a principal for connecting ADQM to Kafka

1. On the host where the MIT Kerberos KDC is deployed, create a new principal `adqm_user` that will be used by ADQM to connect to Kafka:

   ```
   $ sudo kadmin.local -q "add_principal -randkey adqm_user@ADS-KAFKA.LOCAL"
   ```

2. Create a keytab file for this principal:

   ```
   $ sudo kadmin.local -q "xst -kt /tmp/adqm_user.keytab adqm_user@ADS-KAFKA.LOCAL"
   ```

3. Copy the */tmp/adqm_user.keytab* file to ADQM hosts (for example, to the */tmp* directory).
   On an ADQM host, set `clickhouse` as the owner of the keytab file and restrict access to this file for security purposes:

   ```
   $ sudo chown clickhouse:clickhouse /tmp/adqm_user.keytab
   $ sudo chmod 0600 /tmp/adqm_user.keytab
   ```

Set up ADQM

1. On the configuration page of the ADQMDB service, enable ***Kafka engine*** and enter the following parameters in the ***Kafka Properties*** field:

   ```
   <security_protocol>SASL_PLAINTEXT</security_protocol>
   <sasl_mechanism>GSSAPI</sasl_mechanism>
   <sasl_kerberos_service_name>kafka</sasl_kerberos_service_name>
   <sasl_kerberos_keytab>/tmp/adqm_user.keytab</sasl_kerberos_keytab>
   <sasl_kerberos_principal>adqm_user@ADS-KAFKA.LOCAL</sasl_kerberos_principal>
   ```

   Click ***Save*** and execute the ***Reconfig*** action with the ***Restart*** option for the ADQMDB service.

2. Install the following packages on an ADQM host:

   ```
   $ sudo yum install krb5-libs krb5-workstation
   ```

3. Also, on the ADQM host, edit the */etc/krb5.conf* file and change its contents as follows (replace `<kdc_host>` with the name of the host with the MIT Kerberos KDC installed):

   ```
   # Configuration snippets may be placed in this directory as well
   includedir /etc/krb5.conf.d/

   [logging]
    default = FILE:/var/log/krb5libs.log
    kdc = FILE:/var/log/krb5kdc.log
    admin_server = FILE:/var/log/kadmind.log

   [libdefaults]
    dns_lookup_realm = false
    ticket_lifetime = 24h
    forwardable = true
    rdns = false
    pkinit_anchors = FILE:/etc/pki/tls/certs/ca-bundle.crt
    default_realm = ADS-KAFKA.LOCAL

   [realms]
    ADS-KAFKA.LOCAL = {
    admin_server = <kdc_host>
    kdc = <kdc_host>
    }

   [domain_realm]
    .ads-kafka.local = ADS-KAFKA.LOCAL
    ads-kafka.local = ADS-KAFKA.LOCAL
   ```

Create tables in ADQM

On an ADQM host, open clickhouse-client and create tables for integration with Kafka:

1. Table based on the Kafka engine:

   ```
   CREATE TABLE adqm_to_ads_kerberos_kafka (id Int32, name String)
   ENGINE = Kafka ('<kafka_host1>:9092,<kafka_host2>:9092,<kafka_host3>:9092', 'topic-adqm-to-kafka-kerberos', 'clickhouse_group',
   'CSV');
   ```

2. Target table to store data from Kafka:

   ```
   CREATE TABLE adqm_to_ads_kerberos_data (id Int32, name String) ENGINE = MergeTree() ORDER BY id;
   ```

3. Materialized view that will receive data from the Kafka table and put it into the target MergeTree table:

   ```
   CREATE MATERIALIZED VIEW adqm_to_ads_kerberos_data_mv TO adqm_to_ads_kerberos_data
   AS SELECT id, name FROM adqm_to_ads_kerberos_kafka;
   ```

Check the connection between ADQM and Kafka

1. Verify that the data from the Kafka topic is inserted into the ADQM target table:

   ```
   SELECT * FROM adqm_to_ads_kerberos_data;
   ```

   ```
      ┌─id─┬─name──┐
   1. │  1 │ one   │
   2. │  2 │ two   │
   3. │  3 │ three │
      └────┴───────┘
   ```

2. Insert a data row into the ADQM Kafka-engine table:

   ```
   INSERT INTO adqm_to_ads_kerberos_kafka VALUES (4, 'four');
   ```

3. Check that the corresponding message has been successfully received on the ADS cluster side:

   ```
   $ /usr/lib/kafka/bin/kafka-console-consumer.sh \
     --topic topic-adqm-to-kafka-kerberos \
     --from-beginning  \
     --bootstrap-server <kafka_host1>:9092,<kafka_host2>:9092,<kafka_host3>:9092  \
     --consumer.config /tmp/client.properties
   ```

   ```
   1,"one"
   2,"two"
   3,"three"
   4,"four"
   ```

Integration between ADQM and ADPG

Elena Dvoryadkina

Contents

To Table of Contents

- Overview
- Basic parameters for integration
- The postgresql table function
  - Example
- The PostgreSQL table engine
  - Create a table
  - Example
- The PostgreSQL database engine
  - Create a database
  - Example
- The MaterializedPostgreSQL table engine
  - Requirements
  - Create a table
  - Virtual columns
  - Example
- The MaterializedPostgreSQL database engine
  - Requirements
  - Create a database
  - Settings
  - Add/remove tables to/from replication dynamically
  - Example
- Dictionary data source of the POSTGRESQL type
  - Example

Overview

ClickHouse supports a set of tools to integrate with PostgreSQL:

- the postgresql table function | the PostgreSQL table engine | the PostgreSQL database engine — provide the ability to send queries from ClickHouse to tables/databases on a PostgreSQL server for real-time data exchange. They support read and write operations on PostgreSQL data. When executing a `SELECT` or `INSERT` query, ClickHouse connects to a remote PostgreSQL table/database and retrieves up-to-date data from PostgreSQL or exports new data to it. In other words, the data is stored in PostgreSQL, and you can access it from ClickHouse.
- the MaterializedPostgreSQL table engine | the MaterializedPostgreSQL database engine — allow using ClickHouse as a PostgreSQL replica. When you create a MaterializedPostgreSQL table/database, ClickHouse creates an initial data dump of a table or tables of the PostgreSQL database and starts the replication process — in the background, it applies data changes performed on the PostgreSQL side to the corresponding MaterializedPostgreSQL table/database. Thus, when MaterializedPostgreSQL engines are used, unlike PostgreSQL engines, data is stored in both PostgreSQL and ClickHouse.
- the POSTGRESQL type of a data source for dictionaries — allows you to assign a PostgreSQL table to be a data source for a dictionary in ClickHouse.

This article describes ways to use these tools on the example of ADQM and ADPG (Arenadata Postgres — a relational database management system based on PostgreSQL). It provides the syntax of queries to call a table function and create tables/databases based on the mentioned integration engines, as well as sample queries to read/write ADPG data from ADQM.

Prepare ADPG for test examples

On the ADPG host, run the `psql` terminal client as the default user (`postgres`) using the following commands:

```
$ sudo su - postgres
```

```
$ psql
```

Now you can enter commands in `psql` to work with PostgreSQL databases and tables:

```
psql (14.1)
Type "help" for help.

postgres=#
```

> **NOTE**
>
> For an overview on how to work with PostgreSQL tables in `psql`, see Example of working with tables via psql in the ADPG documentation.

Create a PostgreSQL user

Create a user for connecting to ADPG from ADQM (for demonstration purposes, grant superuser privileges):

```
CREATE ROLE adqm_user SUPERUSER LOGIN PASSWORD '<adqm_user_secret_password>';
```

Create an ADPG database and table to use in examples with the table function and PostgreSQL engines

1. Create the `adpg_db1` database in ADPG:

   ```
   CREATE DATABASE adpg_db1;
   ```

   Switch to this database:

   ```
   \c adpg_db1
   ```

   The command result:

   ```
   You are now connected to database "adpg_db1" as user "postgres".
   adpg_db1=#
   ```

2. Create a table:

   ```
   CREATE TABLE table1 (id INTEGER PRIMARY KEY, name VARCHAR(10));
   ```

3. Add test data to the table:

   ```
   INSERT INTO table1 (id, name) VALUES (1, 'one'), (2, 'two');
   ```

Create an ADPG database and tables to use in examples with MaterializedPostgreSQL engines

1. Create a new database in ADPG (`adpg_db2`) and connect to it:

   ```
   CREATE DATABASE adpg_db2;
   ```

   ```
   \c adpg_db2
   ```

2. In the `adpg_db2` database, create three tables with the same structure:

   ```
   CREATE TABLE table1 (id INTEGER PRIMARY KEY, name VARCHAR(10));
   ```

   ```
   CREATE TABLE table2 (id INTEGER PRIMARY KEY, name VARCHAR(10));
   ```

   ```
   CREATE TABLE table3 (id INTEGER PRIMARY KEY, name VARCHAR(10));
   ```

3. Add test data to these tables:

   ```
   INSERT INTO table1 (id, name) VALUES (1, 'one'), (2, 'two');
   ```

   ```
   INSERT INTO table2 (id, name) VALUES (3, 'three'), (4, 'four');
   ```

   ```
   INSERT INTO table3 (id, name) VALUES (5, 'five'), (6, 'six');
   ```

Create an ADPG table to be a data source for a dictionary

In the `adpg_db2` database, create the `roles_dict` table to be a data source for a dictionary:

```
CREATE TABLE roles_dict (id INTEGER PRIMARY KEY, role VARCHAR(10));
```

```
INSERT INTO roles_dict (id, role) VALUES (10, 'admin'), (20, 'owner'), (30, 'author'), (40, 'reviewer'), (50, 'editor'), (60,
'view only');
```

Set up authentication for connecting to ADPG

To allow the `adqm_user` user to connect to the `adpg_db1` and `adpg_db2` databases from an ADQM host and set password authentication, add the following records to the *pg_hba.conf* file (for ADPG, you can do this in the ADCM interface — see the Password authentication article for details):

```
host adpg_db1 adqm_user <adqm_host_address> password
host adpg_db2 adqm_user <adqm_host_address> password
```

where `<adqm_host_address>` is an address of the ADQM host (for example, if an IPv4 address of the ADQM host is `10.92.40.25`, replace `<adqm_host_address>` with `10.92.40.25/32`). You can find description of the `address` field and other fields of records in the *pg_hba.conf* file in the article The pg_hba.conf file.

Basic parameters for integration

To call the `postgresql` table function and create tables/databases based on engines for integration with PostgreSQL (the basic syntax of corresponding queries is provided in sections below), it is required to specify the following parameters.

<postgres_host>	PostgreSQL server address (IP or host name)
<postgres_database>	PostgreSQL database name
<postgres_table>	Name of a table in the PostgreSQL database (specify it for the `postgresql` table function and for the PostgreSQL/MaterializedPostgreSQL table engines)
<postgres_user>	PostgreSQL user name used to connect to PostgreSQL from ClickHouse
<user_password>	PostgreSQL user password
<schema>	PostgreSQL schema (you can optionally specify it for the `postgresql` table function and for the PostgreSQL table/database engines)

> **TIP**
>
> For production environments, it is recommended to pass parameters using named collections.

The postgresql table function

The `postgresql` table function allows you to send `SELECT` and `INSERT` queries to tables in PostgreSQL databases.

The function syntax is:

```
postgresql('<postgres_host>:5432', '<postgres_database>', '<postgres_table>', '<postgres_user>', '<user_password>'[, '<schema>'])
```

Function parameters are described above.

Example

Read data from ADPG

Run the following query to get ADPG data from the `adpg_db1.table1` table in ADQM using the `postgresql` function:

```
SELECT id, name FROM postgresql('<adpg_host>:5432', 'adpg_db1', 'table1', 'adqm_user', '<adqm_user_secret_password>');
```

where `<adpg_host>` is an ADPG host address (IP or host name).

The result:

```
┌─id─┬─name─┐
│  1 │ one  │
│  2 │ two  │
└────┴──────┘
```

Write data to ADPG

Run the following query to send a new record to the ADPG table from ADQM:

```
INSERT INTO TABLE FUNCTION postgresql('<adpg_host>:5432', 'adpg_db1', 'table1', 'adqm_user', '<adqm_user_secret_password>') (id,
name)
VALUES (3, 'three');
```

Make sure that data is written to the destination table. To do this, run the following command on the ADPG host (connect to the `adpg_db1` database first):

```
SELECT * FROM table1;
```

```
 id | name
----+-------
  1 | one
  2 | two
  3 | three
(3 rows)
```

The PostgreSQL table engine

The PostgreSQL table engine allows you to send `SELECT` and `INSERT` queries to tables on a remote PostgreSQL server.

Create a table

Below is the basic syntax of a query that creates a PostgreSQL table (see parameter descriptions above):

```
CREATE TABLE <table_name> (<column_name> <column_type> [DEFAULT|MATERIALIZED|ALIAS <expr>] [TTL <expr>], ...)
ENGINE = PostgreSQL('<postgres_host>:5432', '<postgres_database>', '<postgres_table>', '<postgres_user>', '<user_password>'[,
'<schema>']);
```

The structure of an ADQM table can differ from the structure of an original PostgreSQL table you are connecting to:

- Column names in the ADQM table should be the same as in the original PostgreSQL table, but you can use just some of these columns and in any order.
- Column types in the ADQM table may differ from types in the original PostgreSQL table — ADQM tries to cast values to ClickHouse data types.
- The external_table_functions_use_nulls setting defines how Nullable columns are handled. The default value is `1`. If the value is `0`, columns are not converted to Nullable — default values of corresponding data types are inserted instead of NULL values. This also applies to NULL values in arrays.

Example

In ADQM, create a table based on the PostgreSQL engine and read its data:

```
CREATE TABLE psql_table (id Int32, name String)
ENGINE = PostgreSQL('<adpg_host>:5432', 'adpg_db1', 'table1', 'adqm_user', '<adqm_user_secret_password>');
```

```
SELECT * FROM psql_table;
```

```
┌─id─┬─name──┐
│  1 │ one   │
│  2 │ two   │
│  3 │ three │
└────┴───────┘
```

Insert data into the ADQM table:

```
INSERT INTO psql_table VALUES (4, 'four');
```

On the ADPG host, query the `adpg_db1.table1` table to check that data from ADQM was successfully imported into it:

```
SELECT * FROM table1;
```

```
 id | name
----+-------
  1 | one
  2 | two
  3 | three
  4 | four
(4 rows)
```

The PostgreSQL database engine

The PostgreSQL database engine allows you to connect to a database on a PostgreSQL server and perform read/write operations (`SELECT`/`INSERT` queries) to exchange data between ClickHouse and PostgreSQL, as well as receive real-time information about database tables from the PostgreSQL server and their structures using the `SHOW TABLES` and `DESCRIBE TABLE` queries.

Create a database

The syntax of a query that creates a PostgreSQL database in ADQM is:

```
CREATE DATABASE <database_name>
ENGINE = PostgreSQL('<postgres_host>:5432', '<postgres_database>', '<postgres_user>', '<user_password>'[, `<schema>`,
`use_table_cache`]);
```

See descriptions of basic parameters above. The optional `use_table_cache` parameter specifies whether structures of tables are cached. If the parameter is set to `1`, a table structure is cached and its changes are not tracked — you can update the table structure manually running the `DETACH TABLE` and `ATTACH TABLE` queries sequentially.

Example

Create a database based on the PostgreSQL engine:

```
CREATE DATABASE psql_database
ENGINE = PostgreSQL('<adpg_host>:5432', 'adpg_db1', 'adqm_user', '<adqm_user_secret_password>', 1);
```

Check that this database has been added to the ADQM database list:

```
SHOW DATABASES;
```

```
┌─name───────────────┐
│ INFORMATION_SCHEMA │
│ default            │
│ information_schema │
│ psql_database      │
│ system             │
└────────────────────┘
```

View tables that are in the `psql_database` database:

```
SHOW TABLES FROM psql_database;
```

```
┌─name───┐
│ table1 │
└────────┘
```

Query data from a table:

```
SELECT * FROM psql_database.table1;
```

```
┌─id─┬─name──┐
│  1 │ one   │
│  2 │ two   │
│  3 │ three │
│  4 │ four  │
└────┴───────┘
```

Add new data to the table on the ADPG side:

```
INSERT INTO table1 (id, name) VALUES (5, 'five');
```

Check that this data is available from the ADQM table:

```
SELECT * FROM psql_database.table1;
```

```
┌─id─┬─name──┐
│  1 │ one   │
│  2 │ two   │
│  3 │ three │
│  4 │ four  │
│  5 │ five  │
└────┴───────┘
```

Add a new column to the `adpg_db1.table1` table in ADPG to change its structure. To do this, run the following command on the ADPG host:

```
ALTER TABLE table1 ADD COLUMN value SMALLINT;
```

The table structure on the ADQM side will not change, since the database's `use_table_cache` parameter is set to `1` and ADQM caches table structures:

```
DESCRIBE TABLE psql_database.table1;
```

```
┌─name─┬─type─────────────┬─
│ id   │ Int32            │
│ name │ Nullable(String) │
└──────┴──────────────────┴─
```

To update the table structure in ADQM, detach and re-attach the table:

```
DETACH TABLE psql_database.table1;
```

```
ATTACH TABLE psql_database.table1;
```

```
DESCRIBE TABLE psql_database.table1;
```

```
┌─name──┬─type─────────────┬─
│ id    │ Int32            │
│ name  │ Nullable(String) │
│ value │ Nullable(Int16)  │
└───────┴──────────────────┴─
```

The MaterializedPostgreSQL table engine

The MaterializedPostgreSQL table engine creates a ClickHouse table with an initial data dump of a PostgreSQL table and starts the replication process — applies changes made to the PostgreSQL table to the ADQM table in the background.

If you need to replicate more than one PostgreSQL table to ADQM, it is highly recommended to use the MaterializedPostgreSQL database engine instead of the table engine and use the `materialized_postgresql_tables_list` setting to specify the PostgreSQL tables to be replicated. It is much better in terms of CPU, fewer connections, and fewer replication slots in the PostgreSQL database.

> **CAUTION**
>
> Replication of TOAST values is not supported. The default value for the data type is used.

Requirements

The MaterializedPostgreSQL table engine requires the following conditions to be met:

- In the PostgreSQL configuration file, the `wal_level` parameter should be set to `logical`, and the `max_replication_slots` parameter value should be at least `2` (for more information on these parameters and how to configure them in ADPG, see the Logical replication article).
- A MaterializedPostgreSQL table should have the same primary key as the replication identity index (usually a primary key) of a PostgreSQL table.
- A MaterializedPostgreSQL table can be created in an Atomic database only.

Create a table

The basic syntax of a query to create a table based on the MaterializedPostgreSQL engine is:

```
CREATE TABLE <table_name> (<key_column_name> <key_column_type>, <column_name> <column_type>, ...)
ENGINE = MaterializedPostgreSQL('<postgres_host>:5432', '<postgres_database>', '<postgres_table>', '<postgres_user>',
'<user_password>')
PRIMARY KEY <key_column_name>;
```

Parameter descriptions are provided above.

Virtual columns

MaterializedPostgreSQL tables include the following virtual columns that you can use in `SELECT` queries:

- `_version` (UInt64) — transaction counter (LSN value in WAL);
- `_sign` (Int8) — deletion mark. Possible values:
  - `1` — row is not deleted;
  - `-1` — row is deleted.

Example

Configure ADPG

In ADPG, configure the following parameters required for logical replication, based on which MaterializedPostgreSQL-engine replication is implemented:

- set the `wal_level` parameter to `logical`;
- set the `max_replication_slots` parameter to a value no less than `2`.

Create a MaterializedPostgreSQL table

In ADQM, create a MaterializedPostgreSQL table:

```
CREATE TABLE psql_table_replica (id Int32, name String)
ENGINE = MaterializedPostgreSQL('<adpg_host>:5432', 'adpg_db2', 'table1', 'adqm_user', '<adqm_user_secret_password>')
PRIMARY KEY id;
```

Data from the source ADPG table is automatically inserted into this ADQM table:

```
SELECT * FROM psql_table_replica;
```

```
┌─id─┬─name─┐
│  1 │ one  │
│  2 │ two  │
└────┴──────┘
```

Test replication

Add new rows to the `adpg_db2.table1` table in ADPG:

```
INSERT INTO table1 (id, name) VALUES (11, 'eleven'), (22, 'twenty two');
```

In ADQM, ensure that these rows are added to the table:

```
SELECT * FROM psql_table_replica;
```

```
┌─id─┬─name─┐
│  1 │ one  │
│  2 │ two  │
└────┴──────┘
┌─id─┬─name───────┐
│ 11 │ eleven     │
│ 22 │ twenty two │
└────┴────────────┘
```

The MaterializedPostgreSQL database engine

The MaterializedPostgreSQL database engine allows you to create an ADQM database with tables replicating tables from a PostgreSQL database. Firstly, it creates a snapshot of the PostgreSQL database and loads necessary tables into the MaterializedPostgreSQL database (it can be any subset of tables from any subset of schemas in a PostgreSQL database). The MaterializedPostgreSQL database receives the initial data dump of PostgreSQL tables and starts the replication process — reads WAL and executes DML queries, applying data changes made to the PostgreSQL database.

The MaterializedPostgreSQL-based replication is implemented with the PostgreSQL logical replication protocol, which does not support replication of DDL queries, but can detect changes breaking replication (for example, changing column types, adding/removing columns). If such changes are found, corresponding tables stop receiving updates. In this case, use the `ATTACH TABLE`/`DETACH TABLE` queries to reload tables. If a DDL query does not break replication (for example, renaming a column), a table continues to receive updates (insertion is performed by a position, not by a column name).

> **IMPORTANT**
>
> - The MaterializedPostgreSQL database engine is currently experimental. To use it, enable the option before creating a database: `SET allow_experimental_database_materialized_postgresql = 1`.
> - Replication of TOAST values is not supported. The default value for the data type is used.

Requirements

The MaterializedPostgreSQL database engine requires the following conditions to be met:

- In the PostgreSQL configuration file, the `wal_level` parameter should be set to `logical`, and the `max_replication_slots` parameter value should be at least `2` (to get more information on these parameters and learn how to configure them in ADPG, refer to the Logical replication article in the ADPG documentation).
- Each replicated table should have a replica identity — a primary key (by default) or index defined as REPLICA IDENTITY.

Create a database

Below is the basic syntax of a query for creating a database based on the MaterializedPostgreSQL engine (parameter descriptions are provided above):

```
CREATE DATABASE <database_name>
ENGINE = MaterializedPostgreSQL('<postgres_host>:5432', '<postgres_database>', '<postgres_user>', '<user_password>')
[SETTINGS ...];
```

Settings

Using the `SETTINGS` clause, you can specify the following parameters for the MaterializedPostgreSQL database.

materialized_postgresql_tables_list	Comma-separated list of PostgreSQL database tables that will be replicated via the MaterializedPostgreSQL database engine. - In this list, you can specify tables from different schemas of a PostgreSQL database. To do this, specify each table name along with its schema name (separated by a dot), and also set the setting `materialized_postgresql_tables_list_with_schema = 1`. In this case, to query a table of the MaterializedPostgreSQL database in ADQM, you also need to specify the table name with the schema name. For example: `SELECT * FROM <database_name>.`<schema_name>.<table_name>`;` - If the MaterializedPostgreSQL database replicates tables from a single PostgreSQL schema (set via the `materialized_postgresql_schema` parameter or default), it is enough to specify table names (without schema names) in the `materialized_postgresql_tables_list` list. In this case, a table of the MaterializedPostgreSQL database can also be accessed by its name only. For example: `SELECT * FROM <database_name>.<table_name>;` - If the table list is not specified, all tables from a PostgreSQL schema (set as `materialized_postgresql_schema` or default) will be replicated.
materialized_postgresql_schema	PostgreSQL database schema whose tables will be replicated
materialized_postgresql_schema_list	Comma-separated list of PostgreSQL database schemas — full table sets of these schemas will be replicated. In case of replicating tables from multiple PostgreSQL schemas into the MaterializedPostgreSQL database, specify a schema name and a table name (separated by a dot) when accessing a table of the MaterializedPostgreSQL database in ADQM. For example: `SELECT * FROM <database_name>.`<schema_name>.<table_name>`;`
materialized_postgresql_max_block_size	Number of rows collected in memory before flushing data into a table of a PostgreSQL database
materialized_postgresql_replication_slot *materialized_postgresql_snapshot*	User-created replication slot and text string identifying a snapshot, from which an initial dump of PostgreSQL tables will be performed. Both these settings should be specified together. Use them only if there is a real need — for example, if it is required to define a permanent replication slot to avoid breaking PostgreSQL data replication in case of a primary server failure and switching to another server (to get more details and an example, see Failover of the logical replication slot in the ClickHouse documentation). Typically, you do not need to configure a replication slot yourself — a table creates and manages its own replication slot

Add/remove tables to/from replication dynamically

Once a MaterializedPostgreSQL database is created, new tables added to a PostgreSQL database are not automatically replicated — however, you can add them manually via the `ATTACH TABLE` query. Use also this query if you need to extend the `materialized_postgresql_tables_list` list specified on the MaterializedPostgreSQL database creation.

To remove specific tables from replication, use the `DETACH TABLE` query.

Example

Configure ADPG

The MaterializedPostgreSQL database engine requires the same settings for logical replication in ADPG as the MaterializedPostgreSQL table engine. In ADPG, check:

- the `wal_level` parameter is set to `logical`;
- the `max_replication_slots` parameter value is not less than `2`.

Create a MaterializedPostgreSQL database

In ADQM, enable the support of the MaterializedPostgreSQL database engine and create a database:

```
SET allow_experimental_database_materialized_postgresql=1;
```

```
CREATE DATABASE psql_db_replica
ENGINE = MaterializedPostgreSQL('<adpg_host>:5432', 'adpg_db2', 'adqm_user', '<adqm_user_secret_password>')
SETTINGS materialized_postgresql_tables_list = 'table1,table2';
```

View which tables the database contains:

```
SHOW TABLES FROM psql_db_replica;
```

```
┌─name───┐
│ table1 │
│ table2 │
└────────┘
```

Read one of these tables (for example, `table2`):

```
SELECT * FROM psql_db_replica.table2;
```

```
┌─id─┬─name──┐
│  3 │ three │
│  4 │ four  │
└────┴───────┘
```

Test replication

Add a new data row to the `adpg_db2.table2` table in ADPG:

```
INSERT INTO table2 (id, name) VALUES (30, 'thirty');
```

Make sure this record is sent to the replica table in ADQM:

```
SELECT * FROM psql_db_replica.table2;
```

```
┌─id─┬─name───┐
│  3 │ three  │
│  4 │ four   │
│ 30 │ thirty │
└────┴────────┘
```

Add a new table to replication

Use the `ATTACH TABLE` query to specify an additional table from the ADPG database to be replicated into the ADQM database (in other words, to modify the `materialized_postgresql_tables_list` list of the MaterializedPostgreSQL database):

```
ATTACH TABLE psql_db_replica.table3;
```

Now `table3` is also replicated to the `psql_db_replica` database in ADQM:

```
SHOW TABLES FROM psql_db_replica;
```

```
┌─name───┐
│ table1 │
│ table2 │
│ table3 │
└────────┘
```

```
SELECT * FROM psql_db_replica.table3;
```

```
┌─id─┬─name─┐
│  5 │ five │
│  6 │ six  │
└────┴──────┘
```

Dictionary data source of the POSTGRESQL type

You can set a PostgreSQL table as a data source for a dictionary in ClickHouse/ADQM. To do this, when creating a dictionary with the `CREATE DICTIONARY` query, specify the `POSTGRESQL` source type in the `SOURCE` clause and configure parameters for connecting to a PostgreSQL table as follows:

```
CREATE DICTIONARY <dictionary_name> (...)
...
SOURCE(POSTGRESQL(
    port 5432
    host '<postgres_host>'
    user '<postgres_user>'
    password '<postgres_user_password>'
    replica(host 'postgres_host_1' port 5432 priority 1)
    replica(host 'postgres_host_2' port 5432 priority 2)
    db '<postgres_database>'
    table '<postgres_table>'
    where '<filter_expr>'
    invalidate_query '<query_to_check_dictionary_status>'
    query '<custom_query>'))
...
```

Configuration of the POSTGRESQL dictionary data source includes the following fields:

- `port` — PostgreSQL port.
- `host` — PostgreSQL host.
- `user` — PostgreSQL user name used to connect to PostgreSQL from ClickHouse.
- `password` — PostgreSQL user password.
- `replica` — settings for connecting to a PostgreSQL replica.
  When defining a data source for a dictionary, you can configure general settings for connecting to PostgreSQL (`host`, `port`, `user`, and `password`) or list multiple replicas and configure each one separately — in the `replica` field, specify a host and port for connecting to a replica, and also set the replica priority, which ClickHouse will take into account when trying to connect to PostgreSQL (the lower the `priority` value, the higher the priority of the replica).
- `db` — PostgreSQL database name.
- `table` — name of a table in the PostgreSQL database.
- `where` — selection criteria (optional).
- `invalidate_query` — query to check the dictionary status (optional).
- `query` — custom query (optional).

A data source should be declared through one of the fields: `table` or `query`. The `table` and `where` fields cannot be used together with `query`.

Example

In ADQM, create a dictionary that reads data from the `adpg_db2.roles_dict` table in ADPG:

```
CREATE DICTIONARY roles_dict_psql (id Int32, role String DEFAULT 'no role assigned')
PRIMARY KEY id
SOURCE(POSTGRESQL(port 5432 host '<adpg_host_name>' user 'adqm_user' password '<adqm_user_secret_password>' db 'adpg_db2' table
'roles_dict'))
LIFETIME(MIN 10 MAX 20)
LAYOUT(FLAT());
```

Create the `users` table with users and identifiers of their assigned roles:

```
CREATE TABLE users (user_id Int32, name String, role_id Int32) ENGINE = MergeTree ORDER BY user_id;
```

```
INSERT INTO users VALUES (1, 'john', 10), (2, 'mary', 30), (3, 'andrew', 40), (4, 'harry', 70), (5, 'ann', 50);
```

```
--user_id---name-----role_id--
│       1 │ john   │      10 │
│       2 │ mary   │      30 │
│       3 │ andrew │      40 │
│       4 │ harry  │      70 │
│       5 │ ann    │      50 │
------------------------------
```

Run the following query to read usernames from the `users` table and find their role names in the `roles_dict_psql` dictionary by `role_id`:

```
SELECT  name AS user, dictGet('roles_dict_psql', 'role', toUInt64(role_id)) AS role FROM users;
```

The result:

```
┌─user───┬─role─────────────┐
│ john   │ admin            │
│ mary   │ author           │
│ andrew │ reviewer         │
│ harry  │ no role assigned │
│ ann    │ editor           │
└────────┴──────────────────┘
```

Integration between ADQM and S3

Elena Dvoryadkina

Contents

To Table of Contents

- Integration parameters
  - S3 engine and table function parameters
  - Additional settings
- S3 table engine
  - Create a table
  - Example
- Table functions
  - Example
- Use named collections
  - Parameter keys
  - Example
- Wildcards in a path to S3 files
  - Examples
- Partitioned data writes
- Virtual columns

For integration with S3, ClickHouse supports the S3 table engine and special table functions, which allow reading and writing S3 data from ClickHouse.

S3 files for test examples

In examples this article provides, a bucket in Yandex Object Storage named *test-bucket* is used as an S3 storage. For testing the integration of ADQM and S3, the following objects (files) are loaded into this bucket:

- *test_1.txt*:

```
"one",1
"two",2
"three",3
```

- *test_2.txt*:

```
"ten",10
"twenty",20
"thirty",30
```

To logically structure data and group files within the bucket, the *adqm* prefix (directory) is used. Thus, the files are accessible from ADQM at the following URLs:

- *https://storage.yandexcloud.net/test-bucket/adqm/test_1.txt*
- *https://storage.yandexcloud.net/test-bucket/adqm/test_2.txt*

Integration parameters

S3 engine and table function parameters

To create a table based on the S3 engine and call table functions for integration with S3 (the basic syntax of corresponding queries is provided in sections below), specify the following parameters.

<path>	Bucket URL with a path to a file. When using a table or function for reading data from S3, you can specify wildcards in the path to address multiple files
<access_key_id>, *<secret_access_key>*	Access key identifier and secret key for authenticating requests to an S3 storage. These parameters are optional — you can specify credentials in the configuration file or enable the use_environment_credentials option to retrieve them from environment variables. If a bucket is public, you can set the `NOSIGN` keyword instead of the access key parameters — in this case, requests will not be signed
<session_token>	Session token for a temporary access key, which can be optionally used with the specified `<access_key_id>` and `<secret_access_key>`
<format>	File format. To get a full list of formats ADQM/ClickHouse supports for input and output data, see the Formats for input and output data article of the ClickHouse documentation
<compression>	Compression method. Possible values: `none`; `gzip` or `gz`; `brotli` or `br`; `xz` or `LZMA`; `zstd` or `zst`. By default, the compression method is automatically detected by the file extension
<headers>	S3 request headers in the format `headers('key1'='value1', 'key2'='value2', …)`

TIP

For production environments, it is recommended to pass parameters using named collections.

Additional settings

Parameters at the query/session level

ADQM also supports the following additional parameters related to S3 integration that you can configure before executing a query or specify in a configuration file at the user level:

```sql
SELECT name, default FROM system.settings WHERE startsWith(name, 's3_');
```

```
    ┌─name────────────────────────────────────────────────┬─default────┐
 1. │ s3_strict_upload_part_size                           │ 0          │
 2. │ s3_min_upload_part_size                              │ 16777216   │
 3. │ s3_max_upload_part_size                              │ 5368709120 │
 4. │ s3_upload_part_size_multiply_factor                  │ 2          │
 5. │ s3_upload_part_size_multiply_parts_count_threshold   │ 500        │
 6. │ s3_max_part_number                                   │ 10000      │
 7. │ s3_max_single_operation_copy_size                    │ 33554432   │
 8. │ s3_max_inflight_parts_for_one_file                   │ 20         │
 9. │ s3_max_single_part_upload_size                       │ 33554432   │
10. │ s3_max_single_read_retries                           │ 4          │
11. │ s3_max_unexpected_write_error_retries                │ 4          │
12. │ s3_max_redirects                                     │ 10         │
13. │ s3_max_connections                                   │ 1024       │
14. │ s3_max_get_rps                                       │ 0          │
15. │ s3_max_get_burst                                     │ 0          │
16. │ s3_max_put_rps                                       │ 0          │
17. │ s3_max_put_burst                                     │ 0          │
18. │ s3_list_object_keys_size                             │ 1000       │
19. │ s3_use_adaptive_timeouts                             │ 1          │
20. │ s3_truncate_on_insert                                │ 0          │
21. │ s3_create_new_file_on_insert                         │ 0          │
22. │ s3_skip_empty_files                                  │ 0          │
23. │ s3_check_objects_after_upload                        │ 0          │
24. │ s3_allow_parallel_part_upload                        │ 1          │
25. │ s3_throw_on_zero_files_match                         │ 0          │
26. │ s3_ignore_file_doesnt_exist                          │ 0          │
27. │ s3_validate_request_settings                         │ 1          │
28. │ s3_disable_checksum                                  │ 0          │
29. │ s3_retry_attempts                                    │ 100        │
30. │ s3_request_timeout_ms                                │ 30000      │
31. │ s3_connect_timeout_ms                                │ 1000       │
    └──────────────────────────────────────────────────────┴────────────┘
```

See parameter descriptions in the Session Settings article of the ClickHouse documentation.

Endpoint-related settings

In the configuration file (*/etc/clickhouse-server/config.xml* or another XML file located in the */etc/clickhouse-server/config.d/* directory), you can configure parameters for a specific endpoint inside the `<s3>` tag — these parameters will be used in queries where the prefix of the specified URL exactly matches this endpoint.

Endpoint configuration in general terms:

```xml
<clickhouse>
    <s3>
        <endpoint_config>
            <endpoint>ENDPOINT_URL</endpoint>
            <access_key_id>ACCESS_KEY_ID</access_key_id>
            <secret_access_key>SECRET_ACCESS_KEY</secret_access_key>
            <!-- Other endpoint-related parameters -->
        </endpoint_config>
    </s3>
</clickhouse>
```

where:

- `endpoint_config` — configuration name.
- `ENDPOINT_URL` — S3 endpoint URL, including a bucket URL and path to a directory where data is stored (prefix). Required.
- `ACCESS_KEY_ID`, `SECRET_ACCESS_KEY` — credentials to access the endpoint. These parameters are optional — for example, you can skip them if a bucket is public or if the option to read credentials from environment variables is enabled (see the next section).

See the full list of settings for an S3 endpoint in the Endpoint-based Settings section of the ClickHouse documentation.

Retrieve S3 credentials from the environment

Instead of explicitly specifying credentials to access an S3 bucket in your queries or configuration files, you can enable the `use_environment_credentials` option in ADQM to obtain credentials from the `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY`, `AWS_SESSION_TOKEN` environment variables and Amazon EC2 metadata:

- at the S3 endpoint level:

```xml
<clickhouse>
    <s3>
        <endpoint_config>
            <use_environment_credentials>true</use_environment_credentials>
            <!-- Other endpoint-related parameters -->
        </endpoint_config>
    </s3>
</clickhouse>
```

- at the global level:

```xml
<clickhouse>
    <s3>
        <use_environment_credentials>true</use_environment_credentials>
    </s3>
</clickhouse>
```

S3 table engine

The S3 table engine allows you to execute `SELECT` and `INSERT` queries on data stored in S3.

Create a table

Below is the basic syntax of a query that creates an S3 table:

```sql
CREATE TABLE <table_name> (<column_name> <column_type>, ...)
ENGINE = S3('<path>' [, NOSIGN | '<access_key_id>', '<secret_access_key>' [, '<session_token>']]
            [, '<format>'] [, '<compression>'] [, <headers>])
[PARTITION BY <expr>]
[SETTINGS <setting_name> = <setting_value>, ...];
```

Parameters are described above.

The engine does not support `ALTER` and `SELECT…SAMPLE` queries, as well as indexes.

Example

Create an S3 table

1. In the clickhouse-client console client, create a table based on the S3 engine with a structure corresponding to data to be imported from S3:

```sql
CREATE TABLE s3_table (name String, value UInt32)
ENGINE=S3('https://storage.yandexcloud.net/test-bucket/adqm/test_1.txt',
          '<access_key_id>',
          '<secret_access_key>',
          'CSV');
```

where `<access_key_id>` and `<secret_access_key>` are credentials to access the *test-bucket* S3 bucket.

2. Select data from the table:

```sql
SELECT * FROM s3_table;
```

```
   ┌─name──┬─value─┐
1. │ one   │     1 │
2. │ two   │     2 │
3. │ three │     3 │
   └───────┴───────┘
```

Insert data into an S3 table

1. Enable the `s3_create_new_file_on_insert` setting to create a new file in S3 on each data insert in an ADQM table based on the S3 engine (for example, if an S3 table has been created for the *file_name.txt* file, names of new files will be generated according to the pattern — *file_name.1.txt*, *file_name.2.txt*, and so on):

```sql
SET s3_create_new_file_on_insert = 1;
```

2. Insert data into the table by repeating the `INSERT INTO` query twice:

```sql
INSERT INTO s3_table VALUES('four', 4);
```

```sql
INSERT INTO s3_table VALUES('five', 5);
```

3. Check the bucket contents — two new files (*test_1.1.txt* with the `"four",4` line and *test_1.2.txt* with `"five",5`) have been added to the *adqm* directory.
On subsequent reading the table, you can make sure that a sample includes data from all files:

```sql
SELECT * FROM s3_table;
```

```
┌─name──┬─value─┐
│ one   │     1 │
│ two   │     2 │
│ three │     3 │
└───────┴───────┘
┌─name─┬─value─┐
│ five │     5 │
└──────┴───────┘
┌─name─┬─value─┐
│ four │     4 │
└──────┴───────┘
```

4. Activate the `s3_truncate_on_insert` option and perform another data insertion into the table — this will overwrite the last data file:

```sql
SET s3_truncate_on_insert = 1;
```

```sql
INSERT INTO s3_table VALUES('six', 6);
```

5. Check the bucket contents. The list of files in the *adqm* directory has not changed, but the *test_1.2.txt* file has been overwritten — it now contains the `"six",6` line.
If the `s3_truncate_on_insert` setting is enabled, new data will replace the current contents of an existing file in any case, regardless of the `s3_create_new_file_on_insert` value.

Table functions

ADQM also provides two table functions for integration with S3:

- `s3` — creates a table for reading/writing data in S3.
Function syntax:

```sql
s3('<path>' [, NOSIGN | '<access_key_id>', '<secret_access_key>' [, '<session_token>']]
   [, '<format>'] [, '<structure>'] [, '<compression>'] [, <headers>])
```

where `<structure>` is a table structure in the format `column_name1 data_type1, column_name2 data_type2, …`. Other parameters are described above.

- `s3Cluster` — allows processing S3 files in parallel from multiple nodes of a specified cluster. On an initiator node, the function creates a connection to all nodes in the cluster, discloses asterisks in the S3 file path, and dispatches each file dynamically. On a worker node, it requests the next task from the initiator and processes it. This is repeated until all tasks are completed.
Function syntax:

```sql
s3Cluster('<cluster_name>',
          '<path>' [, NOSIGN | '<access_key_id>', '<secret_access_key>' [, '<session_token>']]
          [, '<format>'] , '<structure>' [, '<compression>'] [, <headers>])
```

where `<cluster_name>` is a name of a cluster used to build a set of addresses and connection parameters to remote and local servers. Other parameters are similar to parameters of the `s3` function, but specifying the table structure is required.

Example

Read data from S3

Run the following query to read data from the *test_2.txt* S3 file with the `s3` function:

```sql
SELECT * FROM s3('https://storage.yandexcloud.net/test-bucket/adqm/test_2.txt',
                 '<access_key_id>',
                 '<secret_access_key>',
                 'CSV',
                 'name String, value UInt32');
```

```
   ┌─name───┬─value─┐
1. │ ten    │    10 │
2. │ twenty │    20 │
3. │ thirty │    30 │
   └────────┴───────┘
```

Write data to S3

When inserting data through the `s3` function with the `s3_truncate_on_insert` setting enabled, existing data in a file referenced by the function will be replaced with new data.

For example, run the following query (the `s3_truncate_on_insert` value was set to `1` in the above example for the S3 table engine):

```sql
INSERT INTO FUNCTION s3('https://storage.yandexcloud.net/test-bucket/adqm/test_2.txt',
                        '<access_key_id>', '<secret_access_key>',
                        'CSV')
VALUES ('s3_func_test_string_1', 100);
```

Check that the *test_2.txt* file has been updated:

```sql
SELECT * FROM s3('https://storage.yandexcloud.net/test-bucket/adqm/test_2.txt',
                 '<access_key_id>',
                 '<secret_access_key>',
                 'CSV');
```

```
   ┌─c1────────────────────┬─c2──┐
1. │ s3_func_test_string_1 │ 100 │
   └───────────────────────┴─────┘
```

If the `s3_truncate_on_insert` parameter value is set to `0` and the `s3_create_new_file_on_insert` setting is enabled, a new file will be created when inserting data into S3 through the `s3` function. For example, if the function inserts data into the *file_name.txt* file, it will create a file named *file_name.1.txt* to save new data or overwrite the *file_name.1.txt* file if it already exists.

Update the settings related to data insertion and call the `s3` function to export data to the *test_2.txt* file:

```sql
SET s3_truncate_on_insert = 0, s3_create_new_file_on_insert = 1;
```

```sql
INSERT INTO FUNCTION s3('https://storage.yandexcloud.net/test-bucket/adqm/test_2.txt',
                        '<access_key_id>', '<secret_access_key>',
                        'CSV')
VALUES ('s3_func_test_string_2', 200);
```

As a result, the data will be saved to a new file named *test_2.1.txt*:

```sql
SELECT * FROM s3('https://storage.yandexcloud.net/test-bucket/adqm/test_2.1.txt',
                 '<access_key_id>',
                 '<secret_access_key>',
                 'CSV');
```

```
   ┌─c1────────────────────┬─c2──┐
1. │ s3_func_test_string_2 │ 200 │
   └───────────────────────┴─────┘
```

Use named collections

To avoid passing all S3 storage connection parameters each time you create an S3 table or call a table function, you can specify them once in a named collection as a list of key/value pairs. This approach also allows you to hide sensitive S3 integration credentials from users without administrator access rights.

Parameter keys

Main keys in an S3-related named collection correspond to names of parameters described above: `url`, `access_key_id`, `secret_access_key`, `session_token`, `format`, `compression` (or `compression_method`), `structure`, `use_environment_credentials`. Also, the following extra parameters are supported.

filename	File name that is appended to the URL
no_sign_request	Specifies whether to ignore credentials so that requests are not signed (for accessing public buckets)
expiration_window_seconds	Period for verifying the expiration of credentials
max_connections	Maximum number of connections per server
max_single_read_retries	Maximum number of attempts during a single S3 read
min_upload_part_size	Minimum object part size to upload during multipart upload to S3
upload_part_size_multiply_parts_count_threshold	Each time this number of object parts have been uploaded to S3, the value of `min_upload_part_size` is multiplied by `upload_part_size_multiply_factor`
upload_part_size_multiply_factor	Factor by which the `min_upload_part_size` value is multiplied each time a number of object parts equal to the value of `upload_part_size_multiply_parts_count_threshold` have been uploaded to S3 by a single write operation
max_single_part_upload_size	Maximum object size for singlepart upload to S3

Example

1. Create a named collection with parameters for connecting to the S3 bucket:

```sql
CREATE NAMED COLLECTION test_bucket_creds AS
url = 'https://storage.yandexcloud.net/test-bucket/adqm/',
access_key_id = '<access_key_id>',
secret_access_key = '<secret_access_key>';
```

2. Use the named collection to call the `s3` table function (a data file has not been defined in the named collection, so it should be specified separately in the function parameters):

```sql
SELECT * FROM s3(test_bucket_creds, filename = 'test_1.txt');
```

```
   ┌─c1────┬─c2─┐
1. │ one   │  1 │
2. │ two   │  2 │
3. │ three │  3 │
   └───────┴────┘
```

3. Create an S3 table using the named collection and read data from the table:

```sql
CREATE TABLE s3_table_nc (name String, value UInt32)
ENGINE=S3(test_bucket_creds, filename = 'test_1.txt');
```

```sql
SELECT * FROM s3_table_nc;
```

```
   ┌─name──┬─value─┐
1. │ one   │     1 │
2. │ two   │     2 │
3. │ three │     3 │
   └───────┴───────┘
```

Wildcards in a path to S3 files

If an ADQM table should accept data from a bunch of S3 files and be read-only, you can use the following wildcards to specify the path to multiple S3 files in the `path` parameter of the S3 table engine or the `s3`/`s3Cluster` table function:

- `*` — substitutes any number of any characters except `/`, including an empty string;
- `**` — substitutes any number of any characters, including `/` and an empty string (this pattern can be used for recursive directory traversal);
- `?` — substitutes any single character;
- `{first_string,second_string,third_one}` — substitutes any of strings `'first_string'`, `'second_string'`, `'third_one'` (you can also use numbers in this pattern — for example, `{1,3,5}`);
- `{n..m}` — substitutes any number in the `[n, m]` range. If the list of files contains a number range with leading zeros in file names, use the general pattern `{0n..0m}` or the separate pattern `{n..m}` (or the sign `?`) for each digit in a file name.

Wildcard characters can be used in multiple path components (for example, in a directory name and in a file name). Only existing files whose paths and names exactly match the pattern are processed. The list of files is determined during the `SELECT` (not `CREATE`) operation.

Examples

Import data from the *test_1.txt* and *test_2.txt* above files into ADQM in one of the following ways:

```sql
SELECT * FROM s3(test_bucket_creds, filename = 'test_{1,2}.txt');
```

```sql
SELECT * FROM s3(test_bucket_creds, filename = 'test_?.txt');
```

Read data from all files in the *adqm* directory:

```sql
SELECT * FROM s3(test_bucket_creds, filename = '*');
```

Partitioned data writes

If you specify the `PARTITION BY` clause when creating an S3 table, a separate file will be created for each partition key value in an S3 storage when you insert data into the table. For example, it may be useful to split data into separate files for further transfer to another system — since ClickHouse data sets are often very large, and network reliability is sometimes insufficient, it makes sense to transfer data sets in relatively small parts.

A partition key can also be specified for exporting data from ADQM to S3 via a table function: `INSERT INTO FUNCTION s3(…) PARTITION BY <expr> VALUES …`.

NOTE

- It is not recommended to use too granular data partition (for example, more granular than by month).
- Partitioning data does not make queries run faster (in contrast to the `ORDER BY` clause). For example, do not partition your data by user IDs or user names — list the ID or user name column first in the `ORDER BY` clause instead.
- Currently, selecting data directly from partitioned S3 tables is not supported, but you can retrieve data from individual partitions using the `s3` table function.

Below is an example of splitting data inserted into an S3 table by values of a column that has a low cardinality (a small number of unique values).

1. Create an S3 table, specifying the `value` column as a partition key and using the `{_partition_id}` parameter in the file name:

```sql
CREATE TABLE s3_table_partition (name String, value UInt32)
ENGINE=S3('https://storage.yandexcloud.net/test-bucket/adqm/test_partition_{_partition_id}.txt',
          '<access_key_id>',
          '<secret_access_key>',
          'CSV')
PARTITION BY value;
```

2. Insert data into the table:

```sql
INSERT INTO s3_table_partition
VALUES ('one_1', 1), ('two_1', 2), ('one_2', 1), ('two_2', 2), ('three_1', 3);
```

3. As a result, data is written to three files, where the `{_partition_id}` parameter has been replaced with the corresponding partition key value in each file's name: *test_partition_1.txt*, *test_partition_2.txt*, and *test_partition_3.txt*. The parameter can also be specified in a name of a directory (prefix).

4. Use the `s3` table function to read data from individual files, for example:

```sql
SELECT * FROM s3('https://storage.yandexcloud.net/test-bucket/adqm/test_partition_1.txt',
                 '<access_key_id>',
                 '<secret_access_key>',
                 'CSV');
```

```
   ┌─c1────┬─c2─┐
1. │ one_1 │  1 │
2. │ one_2 │  1 │
   └───────┴────┘
```

```sql
SELECT * FROM s3('https://storage.yandexcloud.net/test-bucket/adqm/test_partition_2.txt',
                 '<access_key_id>',
                 '<secret_access_key>',
                 'CSV');
```

```
   ┌─c1────┬─c2─┐
1. │ two_1 │  2 │
2. │ two_2 │  2 │
   └───────┴────┘
```

However, if you try to get data directly from the `s3_table_partition` table, an error message will be displayed:

```
Exception: Reading from a partitioned S3 storage is not implemented yet. (NOT_IMPLEMENTED)
```

Virtual columns

The following virtual columns are automatically included to tables based on the S3 engine and tables returned by the `s3` and `s3Cluster` table functions.

Column name	Data Type	Description
_path	LowCardinality(String)	File path
_file	LowCardinality(String)	File name
_size	Nullable(UInt64)	File size in bytes
_time	Nullable(DateTime)	Time of the last file change
_etag	LowCardinality(String)	File ETag

Use virtual columns in `SELECT` queries to get information about S3 files referenced by an ADQM table. For example, the following query returns a list of all *.txt* files in the *adqm* directory of the *test-bucket* bucket and calculates the number of rows in each file:

```sql
SELECT _file, count() AS count
FROM s3('https://storage.yandexcloud.net/test-bucket/adqm/*.txt', '<access_key_id>', '<secret_access_key>', 'CSV')
GROUP BY _file
ORDER BY _file;
```

```
   ┌─_file────────────────┬─count─┐
1. │ test_1.1.txt         │     1 │
2. │ test_1.2.txt         │     1 │
3. │ test_1.txt           │     3 │
4. │ test_2.1.txt         │     1 │
5. │ test_2.txt           │     1 │
6. │ test_partition_1.txt │     2 │
7. │ test_partition_2.txt │     2 │
8. │ test_partition_3.txt │     1 │
   └──────────────────────┴───────┘
```

Import and export to files

Elena Dvoryadkina

Contents

To Table of Contents

- Import from files
  - FROM INFILE
  - The File table engine
  - The file table function
  - Command-line redirection
- Export to files
  - INTO OUTFILE
  - The File table engine
  - The file table function
  - Command-line redirection

Import from files

ADQM supports the following functionality to import data into tables from files.

FROM INFILE

To load data into a table from a local file or multiple files at once (compressed files are also supported), you can use the `FROM INFILE` clause when running the `INSERT INTO` query in the clickhouse-client command-line client. For this, files should be located in the folder from which `clickhouse-client` is launched. The basic syntax is:

```
INSERT INTO <table_name> [(<column_list>)] FROM INFILE '<file_name.ext>' [COMPRESSION '<compr_type>'] [FORMAT <input_format>];
```

The `FROM INFILE` clause uses the following arguments.

<file_name.ext>	Name of the file (with the extension) from which data should be loaded into the ADQM table
<compr_type>	File compression method. Possible values are `none`, `gzip`, `deflate`, `br`, `xz`, `zstd`, `lz4`, `bz2`. By default, the compression method is detected by the extension of the file name
<input_format>	Input data format — see formats in the ***Input*** column of a table in the Formats for Input and Output Data article. By default, the format is automatically detected by the file extension. For examples on how to choose formats to import JSON data depending on its structure, see the JSON data type article

Example

1. Create two local test files (*input_1.csv* and *input_2.csv*) with data to be later inserted into ADQM tables:

   ```
   1,"aaa",100
   2,"bbb",200
   3,"ccc",300
   ```

   ```
   4,"ddd",400
   5,"eee",500
   6,"fff",600
   ```

   Start clickhouse-client from the folder where these files are located.

2. Create a table with the relevant structure:

   ```
   CREATE TABLE test_table (a Int32, b String, c Int32) ENGINE = MergeTree ORDER BY a;
   ```

3. Run the following query to import data from one of the files into the created table:

   ```
   INSERT INTO test_table FROM INFILE 'input_1.csv';
   ```

4. Check that data has loaded into the table:

   ```
   SELECT * FROM test_table;
   ```

   ```
   ┌─a─┬─b───┬───c─┐
   │ 1 │ aaa │ 100 │
   │ 2 │ bbb │ 200 │
   │ 3 │ ccc │ 300 │
   └───┴─────┴─────┘
   ```

5. To insert data from both files into the table at once, pass `input_*.csv` as the file name:

   ```
   INSERT INTO test_table FROM INFILE 'input_*.csv';
   ```

6. Check the import result:

   ```
   SELECT * FROM test_table;
   ```

   ```
   ┌─a─┬─b───┬───c─┐
   │ 1 │ aaa │ 100 │
   │ 2 │ bbb │ 200 │
   │ 3 │ ccc │ 300 │
   │ 4 │ ddd │ 400 │
   │ 5 │ eee │ 500 │
   │ 6 │ fff │ 600 │
   └───┴─────┴─────┘
   ┌─a─┬─b───┬───c─┐
   │ 1 │ aaa │ 100 │
   │ 2 │ bbb │ 200 │
   │ 3 │ ccc │ 300 │
   └───┴─────┴─────┘
   ```

The File table engine

You can load data into ADQM from a file via the File table engine, which uses a file to store data. Data modified directly within a file on a disk is automatically imported into the associated table of the File type when the `SELECT` query is executed. When creating a table based on this engine, pass one of the formats for input data as the engine's argument — see available formats in the ***Input*** column of the Formats for Input and Output Data table. Note that the File table engine does not allow you to explicitly specify the path to the file for storing data — it uses the directory specified via the ***Data path*** parameter on the configuration page of the ADQMDB service (the default is */var/lib/clickhouse/*).

Example

1. Create a `file_table` table using the File engine:

   ```
   CREATE TABLE file_table (name String, value UInt32) ENGINE=File(TabSeparated);
   ```

   ADQM will create the */var/lib/clickhouse/data/default/file_table* directory.

2. Manually create the */var/lib/clickhouse/data/default/file_table/data.TabSeparated* file and fill it with test data:

   ```
   $ sudo cat > /var/lib/clickhouse/data/default/file_table/data.TabSeparated
   a     1
   b     2
   ```

   If you want to export data from the table into this file in the future, assign the `clickhouse` owner to it with the following command:

   ```
   $ sudo chown clickhouse:clickhouse /var/lib/clickhouse/data/default/file_table/data.TabSeparated
   ```

3. Query data from `file_table`:

   ```
   SELECT * FROM test_table;
   ```

   ```
   ┌─name─┬─value─┐
   │ a    │     1 │
   │ b    │     2 │
   └──────┴───────┘
   ```

The file table function

The `file` table function also allows you to import data from a file into a table — see an example in the Table functions article.

Command-line redirection

You can import data into a table from a file directly in the clickhouse-client command line, for example:

```
$ clickhouse-client -q "INSERT INTO test_table FORMAT CSV" < input_2.csv
```

Export to files

You can export data from ADQM to files in the following ways.

INTO OUTFILE

To save the result of the `SELECT` query to a file, use the `INTO OUTFILE` clause when executing the query in the clickhouse-client command-line client. The basic syntax is:

```
SELECT <expr_list>
[FROM <table_name>]
...
INTO OUTFILE '<file_name.ext>' [AND STDOUT] [COMPRESSION '<compr_type>' [LEVEL <compr_level>]]
[FORMAT <output_format>];
```

The `INTO OUTFILE` clause uses the following arguments.

<file_name.ext>	Name of the file (with the extension) where the `SELECT` query result should be saved. The file will be created in the folder from which clickhouse-client is launched. If a file with the specified name already exists, the query will fail
<compr_type>	Compression method. Possible values are `none`, `gzip`, `deflate`, `br`, `xz`, `zstd`, `lz4`, `bz2`. By default, the compression method is detected by the file extension
<compr_level>	Compression level for the specified method. Possible values are positive integers from the following ranges: ▪ `[1, 12]` — for the `lz4` method; ▪ `[1, 22]` — for the `zstd` method; ▪ `[1, 9]` — for other compression methods.
<output_format>	Output data format — see formats supported for output in the Formats for Input and Output Data article of the ClickHouse documentation. By default, the format is detected by the file extension, or the `TabSeparated` format is used if the format could not be detected by the file extension. To get examples of formats for JSON data export, refer to the JSON data type article

If the `AND STDOUT` clause is used in a query, data saved to a file will also be displayed on standard output. If compression is applied, the plaintext will be displayed on standard output.

Example of the `SELECT … INTO OUTFILE` query:

```
SELECT * FROM test_table INTO OUTFILE 'output.csv';
```

The File table engine

Another way to send data from ADQM to a file is to use the File table engine. When creating a table of this type, specify one of the formats for data output as the File engine's argument — this format will be used for data export that is performed along with the `INSERT` queries. You can find available output formats in the Formats for Input and Output Data article.

Example

1. Create a `file_table_out` table based on the File engine:

   ```
   CREATE TABLE file_table_out (name String, value UInt32) ENGINE=File(TabSeparated);
   ```

   ADQM will create the */var/lib/clickhouse/data/default/file_table_out* directory, where `/var/lib/clickhouse/` is the directory for storing ADQM data, which is specified via the ***Data path*** parameter on the ADQMDB service's configuration page.

2. Fill the table with test data:

   ```
   INSERT INTO file_table_out VALUES('a', 1), ('b', 2), ('c', 3);
   ```

3. Make sure that the *data.TabSeparated* file is created in the */var/lib/clickhouse/data/default/file_table_out* directory, and this file contains data from the `file_table_out` table:

   ```
   a     1
   b     2
   c     3
   ```

The file table function

You can also use the `file` table function to export data from ADQM to a file — see an example in the Table functions article.

Command-line redirection

The following query creates the *redirected_output.csv* file in the folder from which clickhouse-client is launched and writes the result of the specified `SELECT` query into it (if such file already exists, data will be overwritten):

```
$ clickhouse-client --query "SELECT * from test_table" --format CSV > redirected_output.csv
```

Configuration parameters

Elena Dvoryadkina

Contents

To Table of Contents

ADQMDB
Chproxy
Zookeeper
Clickhousekeeper
Monitoring

This article describes the parameters that can be configured for ADQM services via ADCM. To read about the configuring process, refer to the relevant article: Configure services or Configure services.

NOTE

Some of the parameters become visible in the ADCM UI after the ***Show advanced*** flag being set.

ADQMDB

SSL Configuration

NOTE

For more information on SSL encryption, see the Encryption of database connections article.

Parameter	Description	Default value
Enable SSL	Enables HTTPS and TCP secure ports for SSL-encrypted connections	Disabled
HTTPS port	Port for secure communication via HTTPS	8443
TCP secure port	TCP port for secure communication with clients and interserver communication through the binary protocol	9440
Disable protocols	Protocols that are not allowed to be used	—
Server certificate file	Path to the server SSL certificate file in the PEM format	/etc/clickhouse-server/server.crt
Server private key file	Path to the file with the private key of the server SSL certificate	/etc/clickhouse-server/server.key
Certificate authority file	Path to the trusted CA certificate file used to verify client certificates if TLS authentication is enabled	—
Client certificate verification mode	Method of checking client SSL certificates signed by the CA certificate (the path to the CA certificate file should be specified in ***Certificate authority file***). Possible values: `none`, `relaxed`, `strict`, `once`	none
Client certificate file	Path to the client SSL certificate file in the PEM format (used for encrypted connections between ZooKeeper and ClickHouse)	—
Client private key file	Path to the file with the private key of the client SSL certificate (used for encrypted connections between ZooKeeper and ClickHouse)	—
Client certificate authority file	Path to the file with the CA certificate used to verify client certificates (used for encrypted connections between ZooKeeper and ClickHouse)	—

gRPC

NOTE

To access the configuration parameters, activate the ***Enable gRPC*** option.

Parameter	Description	Default value
gRPC port	Port for client connections via gRPC	9100
Enable SSL	Enables SSL for secure communication with clients via gRPC	Disabled
SSL certificate file	Path to the file with the SSL certificate (in the PEM format) for connections via gRPC. Used only if SSL is enabled	/etc/clickhouse-server/grpc_ssl_cert_file.crt
SSL private key file	Path to the file with the secret key of the SSL certificate for connections via gRPC. Used only if SSL is enabled	/etc/clickhouse-server/grpc_ssl_key_file.key
Client auth required	Indicates whether the server should request a client for a certificate. If enabled, ***Certificate authority file*** should be specified	Disabled
Certificate authority file	Path to the file with the trusted CA certificate (in the PEM format) used to verify client certificates. Used only if ***Client auth required*** is enabled	—
Compression algorithm	Compression algorithm needed to minimize traffic. Possible values: `none`, `deflate`, `gzip`, `stream_gzip`	deflate
Compression level	Level of traffic compression. Possible values: `none`, `low`, `medium`, `high`	medium
Send message size limits	Size limit for sent messages (in bytes). If `-1`, message size is unlimited	-1
Receive message size limits	Size limit for received messages (in bytes). If `-1`, message size is unlimited	-1
Verbose logs	Specifies whether detailed logs are saved	Disabled

LDAP

NOTE

- To access the configuration parameters, activate the ***Enable LDAP*** option.
- For details on how a remote LDAP server can be used to authenticate users, refer to the LDAP authentication article.
- The table below lists default values for an LDAP server added to the ***LDAP server*** section by default. For a new LDAP server, all fields in the configuration are initially empty, and the required fields are highlighted in red.

Parameter	Description	Default value
name	Name of the LDAP server connection configuration	default
ldap_hostname	LDAP server hostname or IP	localhost
ldap_port	LDAP server port. Default is `636` if ***enable_tls*** is set to `yes`, otherwise — `389`	636
bind_dn	Template used to construct the DN to bind to. The resulting DN will be constructed by replacing all `{user_name}` substrings of the template with the actual username each time the user tries to authenticate	uid={user_name},ou=users,dc=example,dc=com
base_dn	The distinguished name (DN) of the directory object from which to search for entries. Example: `OU=Peoples,DC=ad,DC=ranger-test`	—
search_filter	Template used to construct the user search filter for the LDAP search. The resulting filter will be created by replacing all `{user_name}`, `{bind_dn}`, and `{base_dn}` substrings of the template with the actual username, bind DN, and base DN during the LDAP search. Example: `(&(objectClass=user)(sAMAccountName={user_name}))`	—
enable_tls	Flag to trigger the use of secure connection to the LDAP server. The following values are available: - `no` — for the plain text protocol (ldap://), not recommended; - `yes` — for the SSL/TLS protocol (ldaps://), recommended; - `starttls` — for the StartTLS legacy protocol (the plain text protocol, upgraded to TLS).	no
tls_minimum_protocol_version	Minimum version of the SSL/TLS protocol. Accepted values are: `ssl2`, `ssl3`, `tls1.0`, `tls1.1`, `tls1.2`	—
tls_require_cert	SSL/TLS peer certificate verification behavior. Accepted values are: `never`, `allow`, `try`, `demand`	—
tls_cert_file	Path to the certificate file	—
tls_key_file	Path to the certificate key file	—
tls_ca_cert_file	Path to the CA certificate file	—
tls_ca_cert_dir	Path to the directory containing CA certificates	—
tls_cipher_suite	Allowed cipher suite	—
rbac_enable	Activates role-based access control	Disabled
rbac_default_role	Default role for all authenticated users	default
rbac_base_dn	Base path to start search for groups with a user	ou=Groups,dc=ad,dc=local
rbac_scope	Scope of the LDAP search. Possible values: `base`, `one_level`, `children`, `subtree`	—
rbac_search_filter	LDAP search filter to identify groups to select for mapping users. The resulting filter will be constructed by replacing all `{user_name}`, `{bind_dn}`, `{user_dn}`, and `{base_dn}` substrings of the template with the actual username, bind DN, user DN, and base DN during each LDAP search	(&amp;(objectClass=groupOfNames)(member={bind_dn}))
rbac_attribute	Attribute to look for in LDAP tree	cn
rbac_role_prefix	Attribute value prefix to look for in LDAP tree	—

Kafka engine

NOTE

To access the configuration parameters, activate the ***Kafka engine*** option.

Parameter	Description	Default value
Kafka Properties	In this field, specify parameters of the Kafka table engine. For example: `<debug>all</debug><auto_offset_reset>latest</auto_offset_reset><compression_type>snappy</compression_type>`. The field is required	—

Coordination system

Parameter	Description	Default value
Coordination system	Coordination system for data replication and distributed DDL queries execution. Possible values: `Zookeeper`, `Clickhouse_keeper (integrated)`, `Clickhouse_keeper (allocated)`	Zookeeper

Clickhouse Keeper (integrated)

NOTE

To access the configuration parameters, activate the ***Clickhouse Keepe (integrated)*** option.

Parameter	Description	Default value
Number of nodes in cluster	Number of nodes in a cluster to be used for ClickHouse Keeper	1
Enable IPv6	Allows you to disable IPv6 if your hosts do not support this protocol	Enabled
TCP port	Port for a client to connect	2129
Log storage path	Path to coordination logs	/var/lib/clickhouse-keeper/coordination/log
Snapshot storage path	Path to coordination snapshots	/var/lib/clickhouse-keeper/coordination/snapshots
Operation timeout ms	Timeout for a single client operation (in milliseconds)	10000
Session timeout ms	Timeout for a client session (in milliseconds)	30000
Snapshot distance	Specifies how often ClickHouse Keeper will create new snapshots (in the number of records in logs)	100000
Raft logs level	Coordination logging level (`trace`, `debug`, and so on)	error
Rotate log storage interval	Number of log records to store in a single file	10000
Port	Port where a server listens for RAFT connections	9444
root_path	Znode that is used as the root for znodes used by the ClickHouse server	/clickhouse
Distributed DDL path	In this ZK-path ClickHouse posts DDL tasks. Otherwise, make sure there are no intersections in the configuration with other clusters	/clickhouse/task_queue/ddl

Zookeeper

Parameter	Description	Default value
root_path	Znode that is used as the root for znodes used by the ClickHouse server	/clickhouse
Distributed DDL zookeeper path	In this ZK-path ClickHouse posts DDL tasks. If this CH-cluster is the only one who uses ZooKeeper — don't worry about this setting. Otherwise, make sure there are no intersections in the configuration with other clusters	/clickhouse/task_queue/ddl
SSL connection	Turns on secure connection to ZooKeeper	Disabled

External zookeeper

NOTE

To access the configuration parameters, activate ***External zookeeper***.

Parameter	Description	Default value
zk_hosts	Comma-separated locations of all servers in the ensemble, the ports on which they communicate. For example, `host1.mydomain.com:2181,host2.mydomain.com:2181,host3.mydomain.com:2181`. The field is required	—

Limits

Parameter	Description	Default value
Max connections	Maximum number of incoming connections	4096
Max concurrent queries	Maximum number of simultaneously processed requests	100
Max concurrent threads	Maximum number of query processing threads allowed to run all queries (excluding threads for retrieving data from remote servers). If the limit is reached, at least one thread will still be allocated to execute a query. If the parameter value is `0`, there are no restrictions	0
Max server memory usage	Limits total RAM usage by the ClickHouse server (in bytes). If the parameter value is `0`, the amount of RAM is not restricted	0
Max thread pool size	Limits the number of threads in the Global Thread pool that ClickHouse uses to process queries (if there are no idle threads to process a query, then a new thread is created in the pool)	10000

Network

Parameter	Description	Default value
Enable http	Enables a port for HTTP connection	Enabled
ClickHouse http port	Port for client or proxy connections	8123
Enable tcp	Enables a port for connections via the Native protocol (ClickHouse TCP protocol)	Enabled
ClickHouse tcp port	Port for TCP connections	9000
Listen host	Restriction on hosts that requests can come from. Consider possible absence of IPv6 on a target host and use e.g. `0.0.0.0` in this case	::

Metrics

Parameter	Description	Default value
Listen port	Port to listen for ADQMDB metrics in the Prometheus format	9363
Endpoint	HTTP endpoint to retrieve ADQMDB metrics	/metrics

Directories

Parameter	Description	Default value
Data path	Path to the directory containing data	/var/lib/clickhouse/
Log path	Path to the directory containing log files	/var/log/clickhouse-server/
Temp path	Path to temporary data for processing large queries	/var/lib/clickhouse/tmp/
User files path	Directory with user files. Used in the `file()` table function	/var/lib/clickhouse/user_files/
Access control path	Path to a folder where a ClickHouse server stores user and role configurations created by SQL commands	/var/lib/clickhouse/access/

Cluster configuration

NOTE

Specifics of using cluster configuration parameters are described in the Configure logical clusters in the ADCM interface article.

Parameter	Description	Default value
Replication factor	The number of shards multiplied by `replication factor` is the total number of nodes in the ADQM physical cluster. Cluster topology and macros are generated once and not altered by any further cluster reconfiguration. For more information, read Distributed Table Engine	1
Cluster Configuration	Allows you to add cluster configuration parameters	—
Default cluster topology	Cluster included in the `remote_servers` section in CH-node config. To access this parameter, activate the ***Show advanced*** option at the top of the service configuration page	{}

Intra-cluster communication security

Parameter	Description	Default value
Enable secure intra-cluster communication	Specifies whether secure protocols only (HTTPS, secure TCP) are used for data exchange between ClickHouse servers. If you activate this option, turn on ***Enable SSL*** in the ***SSL Configuration*** section	Disabled
Interserver protocol's port	Port for exchanging data between ClickHouse servers over the interserver protocol	9009
Interserver http name	Username for authentication between replicas	—
Interserver http password	Password for authentication between replicas	—
Shard secret	Cluster secret to validate distributed queries on shards	—

Kerberos configuration

Parameter	Description	Default value
security	Used Kerberos module	—
keytab	Path to the keytab file	—
reinit_frequency	Kerberos reinitialization frequency	3600
principal	Principal — unique name for a client that is allowed to authenticate with Kerberos	—

Log settings

NOTE

The Logging article provides general information about logs that ADQM maintains and writes to files or system tables.

Parameter	Description	Default value
Logging level	Verbosity level of messages written to a log file. Possible values: `trace`, `debug`, `information`, `notice`, `warning`, `error`, `critical`, `fatal`, `none`	trace
Log file size, MB	Log file size (in MB). Once a file reaches the specified size, ADQM archives and renames it, and creates a new log file in its place	1000
Log files count	Number of archived log files that ADQM stores	10
Database	Name of the system database the queries will be logged in	system
Flush interval, milliseconds	Interval for flushing data from the buffer in memory to a table	7500
TTL, days	Lifetime of logs (in days)	30
query_log	Information about executed queries (for example, query start time, duration of query processing, error messages). This is a commonly used log and it is turned on if the checkbox is selected. To disable it, set `log_queries = 0` for a session or query	Disabled
trace_log	Stack traces collected by the sampling query profiler. Useful for deep performance analysis. The log is turned on if the checkbox is selected	Disabled
query_views_log	Information about dependent views executed when running queries (for example, view type, duration of view execution). This is an advanced log and it is turned off by default (even if the checkbox is selected). To enable it, set `log_query_views = 1` for a session or query	Disabled
query_thread_log	Information about threads that execute queries (for example, thread name, thread start time, duration of query processing). This is an advanced log and it is turned off by default (even if the checkbox is selected). To enable it, set `log_query_threads = 1` for a session or query	Disabled
part_log	Information about events related to data parts of MergeTree tables (for example, adding or merging data). This is an advanced log. It is turned on if the checkbox is selected and it is not controlled at the session level	Disabled
crash_log	Information about stack traces for fatal errors. This is an advanced log. It is turned on if the checkbox is selected and it is not controlled at the session level	Disabled
session_log	Information about all successful and failed login and logout events. Logging is turned on if the checkbox is selected and it is not controlled at the session level	Disabled
metric_log	Indicates whether the history of metric values from the `system.metrics` and `system.events` tables is recorded in the `system.metric_log` table	Disabled
asynchronous_metric_log	Indicates whether the history of metric values from the `system.asynchronous_metrics` table is recorded in the `system.asynchronous_metric_log` table	Disabled
asynchronous_insert_log	Information about asynchronous inserts, including results of queries in fire-and-forget mode	Disabled

Default user and policy settings

Parameter	Description	Default value
default_profile_settings	Allows you to add default profile settings	`max_memory_usage : 10000000000` `use_uncompressed_cache : 0`
Access Management	SQL-driven access control and account management for a default user	Enabled
Default user password	Password for a default user	—
Default user IP	User can connect to ClickHouse server only from the specified IP address or a subnetwork. Examples: `192.168.0.0/16`, `2001:DB8::/32`	—
Default user host	User host can be specified as FQDN. For example, `mysite.com`	—
Default user host_regexp	You can use *pcre* regular expressions when specifying user hosts. For example, `.*\.mysite\.com`	—

User password options

Parameter	Description	Default value
Allow plaintext password	Allows or forbids using plaintext passwords for user profiles	Enabled
Allow NO_password	Allows or forbids creating users with no passwords	Enabled
Allow Implicit_NO_password	In combination with ***Allow NO_password***, forbids creating a user with no password unless IDENTIFIED WITH no_password is explicitly specified	Enabled
Length of password	Minimum length of a password. Default is `0` — password length is unlimited	0
Include Upper-case	Indicates whether a password should contain at least one uppercase character	Disabled
Include Lower-case	Indicates whether a password should contain at least one lowercase character	Disabled
Include Number	Indicates whether a password should contain at least one numeric character	Disabled
Include special symbols	Indicates whether a password should contain at least one special character	Disabled

Advanced configuration parameters

Parameter	Description	Default value
Advanced parameters (name/value)	Configuration parameters for ClickHouse's *advanced.xml* file	—
Advanced custom XML	ClickHouse server settings in the XML format. Parameters should be specified within the `<clickhouse>` tag and comply with the XML rules applied in the ClickHouse configuration files	`<clickhouse>` `</clickhouse>`

Additional local storage

NOTE

- To access the configuration parameters, activate the ***Enable additional local storage*** option.
- See the Configure data storages article for an example of how to set up a tiered data storage with additional block devices (SSD and HDD).

Parameter	Description	Default value
Disks	List of disks in the local file system that can be used for storing table data	—
Recursively create directories	Specifies whether directories for local disks should be automatically created if they do not exist	Enabled

S3 storage

NOTE

- To access the configuration parameters, activate the ***Enable S3 storage*** option.
- For details on configuring disks of different types to store ADQM table data (including descriptions of S3 disk configuration settings), see the Configure data storages article.

Parameter	Description	Default value
S3 disks	List of external S3 disks that can be used for storing table data	—

Encrypted storage

NOTE

- To access the configuration parameters, activate the ***Enable encrypted storage*** option.
- For details on configuring disks of different types to store ADQM table data (including descriptions of encrypted disk configuration settings), see the Configure data storages article.

Parameter	Description	Default value
Encrypted disks	List of encrypted disks to be created over disks of other types (local, S3, cache)	—
Recursively create directories	Specifies whether directories for encrypted disks should be automatically created if they do not exist	Enabled

Cache

NOTE

- To access the configuration parameters, activate the ***Enable cache*** option.
- For details on configuring disks of different types to store ADQM table data (including descriptions of cache disk configuration settings), see the Configure data storages article.

Parameter	Description	Default value
Cache disks	List of cache disks to be created over external disks that store data	—
Recursively create directories	Specifies whether directories for cache disks should be automatically created if they do not exist	Enabled

Storage policies

NOTE

For detailed information on configuring storage policies for ADQM tables, see the Configure data storages article.

Parameter	Description	Default value
Policies	List of storage policies that can be assigned to tables	—

Data compression

NOTE

- To access the configuration parameters, turn on the ***Data compression*** switch.
- For information on compression methods available in ADQM and options to use them, see the Data compression article.

Parameter	Description	Default value
Minimum part size	Minimum size of a data part (in bytes)	10000000000
Minimum part size ratio	Ratio of the minimum size of a data part to the table size	0.01
Method	Compression method. Possible values: `LZ4`, `LZ4HC`, `ZSTD`	LZ4
Level	Compression level. Possible values: - for the `LZ4HC` method — [`1`, `12`]; - for the `ZSTD` method — [`1`, `22`]; - the `LZ4` method has no compression levels — any value is ignored.	9

JDBC Bridge configuration

NOTE

For detailed information on ADQM integration with external data sources via the JDBC interface, see the JDBC Bridge article.

Parameter	Description	Default value
jdbc bridge log path	Path to the directory containing log files	/var/log/clickhouse-jdbc-bridge/
datasource.json	Data source for the JDBC Bridge	

```
[
    {
        "$schema": "./datasource.jschema",
        "postgres13": {
            "converter": {
                "mappings": [
                    {
                        "nativeType": "bool",
                        "toType": "String"
                    }
                ]
            },
            "driverUrls": [
                "drivers/postgresql/postgresql-jdbc.jar"
            ],
            "driverClassName": "org.postgresql.Driver",
            "jdbcUrl": "jdbc:postgresql://localhost/test",
            "username": "",
            "password": "",
            "initializationFailTimeout": 0,
            "minimumIdle": 0,
            "maximumPoolSize": 10
        }
    },
    {
        "$schema": "./datasource.jschema",
        "ch-server": {
            "driverUrls": [
                "drivers/clickhouse/clickhouse-jdbc.jar"
            ],
            "driverClassName": "ru.yandex.clickhouse.ClickHouseDriver",
            "jdbcUrl": "jdbc:clickhouse://ch-server:8123/system",
            "username": "default",
            "password": "",
            "initializationFailTimeout": 0,
            "minimumIdle": 0,
            "maximumPoolSize": 10
        }
    }
]
```

Parameter	Description	Default value
server.json	Server configuration for the JDBC Bridge	"{\n \"requestTimeout\": 5000,\n \"queryTimeout\": 60000,\n \"configScanPeriod\": 5000,\n \"repositories\": [\n {\n \"entity\": \"ru.yandex.clickhouse.jdbcbridge.core.NamedDataSource\",\n \"repository\": \"ru.yandex.clickhouse.jdbcbridge.impl.JsonFileRepository\"\n },\n {\n \"entity\": \"ru.yandex.clickhouse.jdbcbridge.core.NamedSchema\",\n \"repository\": \"ru.yandex.clickhouse.jdbcbridge.impl.JsonFileRepository\"\n },\n {\n \"entity\": \"ru.yandex.clickhouse.jdbcbridge.core.NamedQuery\",\n \"repository\": \"ru.yandex.clickhouse.jdbcbridge.impl.JsonFileRepository\"\n }\n],\n \"extensions\": [\n {\n \"class\": \"ru.yandex.clickhouse.jdbcbridge.impl.JdbcDataSource\"\n },\n {\n \"class\": \"ru.yandex.clickhouse.jdbcbridge.impl.ConfigDataSource\"\n },\n {\n \"class\": \"ru.yandex.clickhouse.jdbcbridge.impl.ScriptDataSource\"\n }\n]\n}\n"

Encryption

NOTE

The Encryption codecs article provides an example of how to set up and apply data encryption based on the AES-256 algorithm.

Parameter	Description	Default value
Credentials encrypt	Algorithm for encrypting sensitive data in Clickhouse configuration files. Possible values: - `AES-128`; - `AES-256`; - `none` — encryption is not applied, and sensitive data remains unencrypted.	none
aes_128_source	Source of encryption parameter values for the AES-128 algorithm. Possible values: - `from_env` — environment variable; - `from_zk` — ZooKeeper node; - `none` — values from the configuration.	none
aes_256_source	Source of encryption parameter values for the AES-256 algorithm. Possible values: - `from_env` — environment variable; - `from_zk` — ZooKeeper node; - `none` — values from the configuration.	none
aes_128_key_env	Environment variable that contains a key for the AES-128 encryption algorithm	—
aes_128_nonce_env	Environment variable that contains a nonce for the AES-128 encryption algorithm	—
aes_128_key_zk_path	Path to a key for the AES-128 encryption algorithm in ZooKeeper	—
aes_128_nonce_zk_path	Path to a nonce for the AES-128 encryption algorithm in ZooKeeper	—
aes_256_key_env	Environment variable that contains a key for the AES-256 encryption algorithm	—
aes_256_nonce_env	Environment variable that contains a nonce for the AES-256 encryption algorithm	—
aes_256_key_zk_path	Path to a key for the AES-256 encryption algorithm in ZooKeeper	—
aes_256_nonce_zk_path	Path to a nonce for the AES-256 encryption algorithm in ZooKeeper	—
AES-128 GCM-SIV key	Key for the AES-128 encryption algorithm as a 32-character hex-encoded string. For example, `00112233445566778899AABBCCDDEEFF`	—
AES-128 GCM-SIV nonce	Nonce for the AES-128 encryption algorithm as a 24-character hex-encoded string. For example, `ABCDEFABCDEFABCDEFABCDEF`	—
AES-256 GCM-SIV key	Key for the AES-256 encryption algorithm as a 64-character hex-encoded string. For example: `00112233445566778899AABBCCDDEEFF00112233445566778899AABBCCDDEEFF`	—
AES-256 GCM-SIV nonce	Nonce for the AES-256 encryption algorithm as a 24-character hex-encoded string. For example, `ABCDEFABCDEFABCDEFABCDEF`	—

CAUTION

ZooKeeper with enabled SSL cannot be used to provide the `encrypt_decrypt` functionality at this time.

Other

Parameter	Description	Default value
Time zone	Time zone used for conversions between the String and DateTime formats (for example, when values from DateTime fields are output to text format, or when strings are converted to DateTime values). If the parameter value is `System`, ADQM uses the `timezone` parameter value set in the ADQM server configuration or in the operation system settings at the moment of the ADQM server start	System
Default database	Default database	default
Default profile	Default settings profile. Settings profiles are located in the file specified in the `user_config` parameter	default
Auto Core Dump Service	Indicates whether an automatic core dump is enabled. To enable or disable an automatic core dump, use the corresponding action of the ADQMDB service	Disabled

Chproxy

Network configuration

Parameter	Description	Default value
Enable HTTP	Enables listening for HTTP requests	Enabled
HTTP listen port	Port to listen for HTTP requests	9090
HTTP allowed networks	List of networks from which HTTP access is allowed. Each item may contain IP address or IP subnet mask. The `127.0.0.1` value is added to configuration by default. Chproxy tries detecting the most obvious configuration errors such as `allowed_networks: ["0.0.0.0/0"]`	—
Enable HTTPS	Enables listening for HTTPS requests	Disabled
HTTPS listen port	Port to listen for HTTPS requests	8543
HTTPS allowed networks	List of networks from which HTTPS access is allowed. Each item may contain IP address or IP subnet mask. The `127.0.0.1` value is added to configuration by default. Chproxy tries detecting the most obvious configuration errors such as `allowed_networks: ["0.0.0.0/0"]`	—
Certificate file	Path to the server SSL certificate file in the PEM format	/etc/chproxy/server.crt
Private key file	Path to the file with the private key of the server SSL certificate	/etc/chproxy/server.key
Use HTTP to connect to ADQMDB	Specifies which protocol (HTTP or HTTPS) Chproxy should use to connect to ClickHouse	Enabled

Proxy users configuration

NOTE

The table below lists default values for a user added to the ***Clickhouse Proxy Users*** section by default. For a new user, all fields in the configuration are initially empty, and the required fields are highlighted in red.

Parameter	Description	Default value
deny_http	Specifies whether to block HTTP connections for the user	Disabled
user_name	ADQM user name or, if the `wildcarded_user` parameter is enabled, wildcarded user name that is a pattern like `<prefix>*` or `*<suffix>`	default
allow_cors	Specifies whether to allow CORS requests (required for tools like Tabix)	Disabled
cache_name	Response cache name (this setting enables caching responses that are not cached by default)	—
deny_https	Specifies whether to block HTTPS connections for the user	Disabled
max_queue_size	Maximum number of requests waiting for execution in the queue (`0` — requests are executed without waiting in the queue)	0
max_queue_time	Maximum duration a request may wait in the queue	10s
wildcarded_user	Indicates the configuration of a wildcarded user. If set, the `user_name` parameter should be set to a `<prefix>*` or `*<suffix>` user name pattern (for example, `dba_*` or `*-analyst`). In this case, any user with a name matching this pattern will log in to ADQM as a user specified in `clickhouse_target_user`	Disabled
allowed_networks	List of networks from which access is allowed (a list item can be an IP address or subnet mask)	—
max_execution_time	Maximum duration of a user query execution	120s
requests_per_minute	Maximum number of user requests per minute (`0` — the number is unlimited; a negative value will effectively block the user)	0
clickhouse_target_user	ADQM user whose credentials will be used to proxy requests from users matching the wildcarded user pattern (if the `wildcarded_user` setting is disabled, this parameter is not used)	default
max_concurrent_queries	Maximum number of concurrently running user queries	0
Users/password map	Passwords of users specified in the ***Proxy users configuration*** section	—

Connection pool configuration

Parameter	Description	Default value
Max idle connections	Maximum number of idle connections in the pool	100
Max idle connections per host	Maximum number of idle connections per host	2

ADQM cluster configuration

Parameter	Description	Default value
Interval	Interval for checking all cluster nodes for availability	5s
Timeout	Timeout of wait response from cluster nodes	5s

Zookeeper

Main

Parameter	Description	Default value
connect	ZooKeeper connection string used by other services or clusters. It is generated automatically	—
dataDir	Location where ZooKeeper stores the in-memory database snapshots and, unless specified otherwise, the transaction log of updates to the database. It is specified as the `dataDir` parameter in the *zoo.cfg* configuration file	/var/lib/zookeeper
admin.serverPort	AdminServer port. An embedded Jetty server that provides an HTTP interface to the four letter word commands	58080

zoo.cfg

Parameter	Description	Default value
clientPort	Port to listen for client connections, i.e. the port that clients attempt to connect to	2181
tickTime	Length of a single tick (in milliseconds) which is the basic time unit used by ZooKeeper to regulate heartbeats and timeouts	2000
initLimit	Amount of time (in ticks) to allow followers to connect and sync to a leader. Increase this value as needed, if the amount of data managed by ZooKeeper is large	5
syncLimit	Amount of time (in ticks) to allow followers to sync with ZooKeeper. If followers fall too far behind a leader, they will be dropped	2
maxClientCnxns	Limits the number of concurrent connections that a single client, identified by IP address, may make to a single member of the ZooKeeper ensemble	0
autopurge.snapRetainCount	Retains the specified number of the most recent snapshots and the corresponding transaction logs in dataDir and dataLogDir respectively, and deletes the rest. The minimum value is `3`	3
autopurge.purgeInterval	Time interval (in hours) for which the purge task has to be triggered. Set a positive integer to enable the auto purging	24
Add key,value	In this field, you can define additional parameters that are not displayed in ADCM UI, but are allowed in the zoo.cfg configuration file	—

SSL configuration

Parameter	Description	Default value
Enable SSL	Enables secure incoming connections in ZooKeeper	Disabled
sslQuorum	Enables encrypted quorum communication	Disabled
serverCnxnFactory	Specifies `ServerCnxnFactory` implementation. To use TLS-based server communication, set it to `NettyServerCnxnFactory`	org.apache.zookeeper.server.NettyServerCnxnFactory
ssl.quorum.keyStore.location	Fully-qualified path to the server keystore file (for example: /etc/zookeeper/ssl/keystore.jks)	/etc/zookeeper/server.jks
ssl.quorum.keyStore.password	Password for the keystore specified in `ssl.quorum.keyStore.location`	—
ssl.quorum.trustStore.location	Fully-qualified path to the server truststore file (for example: /etc/zookeeper/ssl/truststore.jks)	/etc/zookeeper/ca.jks
ssl.quorum.trustStore.password	Password for the truststore specified in `ssl.quorum.trustStore.location`	—
ssl.keyStore.location	Fully-qualified path to the server keystore file to be used for client TLS connections (for example: /etc/zookeeper/ssl/keystore.jks)	/etc/zookeeper/server.jks
ssl.keyStore.password	Password for the keystore specified in `ssl.keyStore.location`	—
ssl.trustStore.location	Fully-qualified path to the server truststore file to be used for client TLS connections (for example: /etc/zookeeper/ssl/truststore.jks)	/etc/zookeeper/ca.jks
ssl.trustStore.password	Password for the truststore specified in `ssl.trustStore.location`	—
ssl.protocol	Protocol to be used in client TLS negotiation	TLSv1.2
ssl.quorum.protocol	Protocol to be used in quorum TLS negotiation	TLSv1.2
secureClientPort	Port to be used for secure client connections	2281

Metrics

Parameter	Description	Default value
Listen port	Port to listen for Zookeeper service metrics in the Prometheus format	9020

zookeeper-env.sh

Parameter	Description	Default value
ZOO_LOG_DIR	Directory to store the logs	/var/log/zookeeper
ZOOPIDFILE	Directory to store the ZooKeeper process ID	/var/run/zookeeper/zookeeper_server.pid
SERVER_JVMFLAGS	Sets different JVM parameters (for example, parameters related to garbage collection)	-Xmx1024m
JAVA	Path to Java	$JAVA_HOME/bin/java
ZOO_LOG4J_PROP	Sets the log4j logging level and which log appenders to turn on. If you turn on the `CONSOLE` log appender, logs go to stdout. If you turn on `ROLLINGFILE`, zookeeper.log file is created, rotated, and expired	INFO, CONSOLE, ROLLINGFILE

Clickhousekeeper

General settings

Parameter	Description	Default value
Listen host	Restriction on hosts from which requests can come. Consider possible absence of IPv6 on a target host and use `0.0.0.0` in this case	::
Enable IPv6	Allows you to disable IPv6 if your hosts do not support this protocol	Enabled
TCP port	Port for a client to connect	2129
Log storage path	Path to coordination logs	/var/lib/clickhouse-keeper/coordination/log
Snapshot storage path	Path to coordination snapshots	/var/lib/clickhouse-keeper/coordination/snapshots
root_path	Znode that is used as a root for znodes used by the ClickHouse server	/clickhouse
Distributed DDL path	ZK-path to which ClickHouse posts DDL tasks. Otherwise, make sure there are no intersections in the configuration with other clusters	/clickhouse/task_queue/ddl

Coordination settings

Parameter	Description	Default value
Operation timeout ms	Timeout for a single client operation (in milliseconds)	10000
Session timeout ms	Timeout for a client session (in milliseconds)	30000
Snapshot distance	Specifies how often ClickHouse Keeper will create new snapshots (in the number of records in logs)	100000
Raft logs level	Coordination logging level (`trace`, `debug`, and so on)	error
Rotate log storage interval	Number of log records to store in a single file	10000

Raft settings

Parameter	Description	Default value
Port	Port where a server listens for RAFT connections	9444

Metrics

Parameter	Description	Default value
Listen port	Port to listen for Clickhousekeeper service metrics in the Prometheus format	9010
Endpoint	HTTP endpoint to retrieve metrics of the Clickhousekeeper service	/metrics

Logger options

Parameter	Description	Default value
Log level	Logging level (`trace`, `debug`, `information`, `warning`, or `error`)	Trace
Log path	Path to the directory containing log files	/var/log/clickhouse-keeper/
Log file size	Log file size (in MB). Once a file reaches the specified size, ADQM archives and renames it, and creates a new log file in its place	1000
Log file count	Number of archived log files that ADQM stores	5

Monitoring

Prometheus settings

Group	Parameter	Description	Default value
—	scrape_interval	Specifies how frequently to scrape targets	1m
—	scrape_timeout	Specifies how long to wait until a scrape request times out	10s
—	Password for grafana connection	Password of a Grafana user (`admprom_grafana`) to connect to Prometheus	—
—	Prometheus users to login/logout to Prometheus	User credentials for logging into the Prometheus web interface	—
Service parameters	config.file	Path to the Prometheus server configuration file	/etc/admprom/prometheus/prometheus.yml
	storage.tsdb.path	Path to the Prometheus server database	/var/lib/admprom/prometheus
	web.console.libraries	Path to console management libraries	/usr/share/admprom/prometheus/console_libraries
	web.console.templates	Path to Prometheus server console templates	/usr/share/admprom/prometheus/consoles
	web.config.file	Specifies which web configuration file to load. The file is written in YAML format	/etc/admprom/prometheus/prometheus-auth.yml
	storage.tsdb.retention.time	Specifies how long to retain samples in the storage. Supported units: `y`, `w`, `d`, `h`, `m`, `s`, `ms`	15d
	web.listen-address	Address to access the Prometheus web interface	0.0.0.0:11200

Grafana settings

Parameter	Description	Default value
Grafana administrator's password	Password of a Grafana administrator user	—
Grafana listen port	Port to access the Grafana web interface	11210

Pushgateway settings

Parameter	Description	Default value
listen_address	Address to access Pushgateway Web API	0.0.0.0:9091

Node Exporter settings

Parameter	Description	Default value
Listen port	Port to access ADQM host system metrics in the Prometheus format	11203
Metrics endpoint	Endpoint to which the node exporter exports system metrics in the Prometheus format	/metrics

SSL configuration

Parameter	Description	Default value
[Prometheus] → Enable SSL	Enables SSL encryption of incoming traffic for the Prometheus Server component	Disabled
[Prometheus] → Certificate file	Path to the server SSL certificate file for the Prometheus Server component	/etc/admprom/prometheus/server.crt
[Prometheus] → Private key file	Path to the file with the private key of the server SSL certificate for the Prometheus Server component	/etc/admprom/prometheus/server.key
[Prometheus] → Certificate authority file	Path to the file with the CA certificate to verify Prometheus targets	/etc/admprom/prometheus/ca.crt
[Grafana] → Enable SSL	Enables SSL encryption of incoming traffic for the Grafana component	Disabled
[Grafana] → Certificate file	Path to the server SSL certificate file for the Grafana component	/etc/admprom/grafana/server.crt
[Grafana] → Private key file	Path to the file with the private key of the server SSL certificate for the Grafana component	/etc/admprom/grafana/server.key
[Grafana] → Certificate authority file	Path to the file with the CA certificate to verify the Prometheus certificate	/etc/admprom/grafana/ca.crt
[Node-exporter] → Enable SSL	Enables SSL encryption of incoming traffic for the Node Exporter component	Disabled
[Node-exporter] → Certificate file	Path to the server SSL certificate file for the Node Exporter component	/etc/ssl/server.crt
[Node-exporter] → Private key file	Path to the file with the private key of the server SSL certificate for the Node Exporter component	/etc/ssl/server.key
[Pushgateway] → Enable SSL	Enables SSL encryption of incoming traffic for the Pushgateway component	Disabled
[Pushgateway] → Certificate file	Path to the server SSL certificate file for the Pushgateway component	/etc/admprom/pushgateway/server.crt
[Pushgateway] → Private key file	Path to the file with the private key of the server SSL certificate for the Pushgateway component	/etc/admprom/pushgateway/server.key

ADQM releases

Elena Dvoryadkina

Contents

To Table of Contents

25.8.7.3
25.3.2.39
24.8.8.17
24.3.11.7
24.3.2.23
23.8.2.7
23.3.2.37
22.8.12.45
22.8.6.71
22.8.4.7
22.3.7.28
22.3.2.2
21.8.13.6
21.8.8.29
21.3.14.1
21.3.12.2
21.3.11.5
21.3.4.25
20.8.13.15
20.8.11.17
20.8.4.11
20.3.20.6
20.8.2.3
20.3.13.130
20.3.10.75
19.14.11.16

25.8.7.3

25.8.7.3.1.b1

Date: 26.11.2025

New features

- Increased ClickHouse version to 25.8.7.3
- Extended the ***Advanced configuration parameters*** section of the ADQMDB service configuration with the ***Advanced custom XML*** field, an XML value of which is validated at the precheck stage and injected into the *config.xml* main configuration file
- Enabled ZooKeeper dynamic reconfiguration for expanding/shrinking an ensemble without the service restart
- Added options to enable SSL for components of the Monitoring service through the Manage SSL interface
- Added the ability to reload the configuration without restarting a service for the following services:
  - ADQMDB (including if the Clickhouse JDBC Bridge component is installed);
  - Clickhousekeeper;
  - Chproxy.

  The ***Reconfig and restart*** service action is replaced by ***Reconfig*** with the optional ***Restart*** checkbox in the action configuration window

Improvements

- Added the ability to delete mapped services in the `created` and `installed` states via the icon 🗑 on the Services page of the ADCM interface
- Added the top-level ***Coordination system*** parameter instead of the ***Engine*** group with a single ***System*** parameter on the configuration page of the ADQMDB service
- Implemented SSL support and corresponding settings for the Pushgateway component of the Monitoring service
- Improved handling of JDBC bridge packages during upgrade and rollback, including service stop, cleanup, and downgrade autotests
- Refactored configuration parameters of the Chproxy service:
  - added the ***Proxy users configuration*** and ***Connection pool configuration*** groups of parameters;
  - removed the ***Proxy wildcarded users*** group;
  - implemented the conversion and validation logic to be applied when updating the bundle version;
  - secured Chproxy user passwords using the secretmap data type.
- Implemented automatic deletion of obsolete data sources from the JDBC bridge configuration
- Added statuschecker removal during component shrink operations

25.3.2.39

25.3.2.39.1.b2

Date: 08.10.2025

Improvements

- Improved the Monitoring Clients service state handling by adjusting the service state before and after bundle switch
- Added validation and permission setting for the */var/lib/clickhouse* directory during installation
- Ensured that services in the maintenance mode remain in Prometheus targets only if the host is reachable
- Corrected behavior when all ADQMDB or ZooKeeper hosts are in the maintenance mode
- Ensured that upgrades fail correctly if any cluster host is in the maintenance mode

Bug fixes

- Fixed the `ADQM_arenadata_enterprise` attribute error occured during upgrade actions on the Community edition
- Fixed the installation error for allocated ClickHouse Keeper
- Fixed an error occured during the ***Add hosts*** action run for Chproxy with unreachable ADQMDB hosts in the maintenance mode
- Resolved failures of the ADQMDB's ***Reconfig and restart*** action occured when all shard hosts were in the maintenance mode

25.3.2.39.1.b1

Date: 19.08.2025

New features

- Introduced maintenance mode for ADQM
- Added support for Alt Linux 10.2
- Added support for Astra Linux 1.7.6 Voronezh
- Increased ClickHouse version to 25.3.2.39
- Upgraded clickhouse-copier — it now supports all schemas related to ADQMDB version 25.3.2.39
- Implemented the ***Add/Remove components*** action for all services of ADQM (instead of the ***Add Server(s)***, ***Remove Server(s)***, ***Add JDBC Bridge***, ***Remove JDBC Bridge***, ***Shrink***, and ***Expand*** actions)
- Added the jdbc-mysql-connector and jdbc-postgresql packages into the ADQM bundle
- Returned the ***session_log*** parameter to the ***Log settings*** section on the ***Configuration*** page of the ADQMDB service after fixes in ClickHouse upstream

Improvements

- FQDNs are now used during Zookeeper installation
- FQDNs are now used in the `remote_servers` section of */etc/clickhouse-server/config.xml*
- Password of the `default` user should now have a minimum length of 8 characters and should not contain special characters
- Added HTTPS support for Enterprise Tools

Misc/Internal

- Removed the old monitoring service based on Diamond

24.8.8.17

24.8.8.17.2.b1

Date: 24.03.2025

New features

- Added the Manage SSL cluster action
- Implemented secure connection to ADQMDB in Chproxy
- Increased ClickHouse version to 24.8.8.17
- Added more parameters to the ***Encryption codecs for tables*** section of the ADQMDB service configuration and renamed it to ***Encryption***
- Added encrypting passwords

Improvements

- Added the ***Metrics*** activatable group of parameters to the ADQMDB, Clickhousekeeper, and Zookeeper service configurations
- Removed autorestart options from the ***Reconfigure and restart*** action of the Monitoring service
- Enabled auto-activation of the ***Metrics*** parameter group for ADQM services when Prometheus is installed
- Implemented allocation of node exporter on all hosts after upgrade
- Added a pattern check to the Clickhousekeeper's ***Listen host*** field
- Password for the default user removed from all configs of `clickhouse-client`
- Cluster options from the ***Reliability control*** section are used to retry the task `Create Clickhouse znode`
- FQDN is used in Chproxy config
- Removed the `logo-container` HTML block for playground (`/play`) in ADQMDB

Bug fixes

- Fixed a template for the ***Proxy wildcarded users*** configuration of the Chproxy service
- Updated ADQM SSL defaults and permissions
- Fixed statuschecker
- Refactored an error message displayed on the ***Ansible [check]*** tab while checking Chproxy

Misc/Internal

- Increased the minimum version of ADQM for upgrade to 23.8.2.7.1.b1

24.3.11.7

24.3.11.7.1.b1

Date: 17.10.2024

New features

- Increased ClickHouse version to 24.3.11.7-lts

Bug fixes

- Abort of the cluster upgrade (see the ***Abort the upgrade*** action) now includes a step to remove ADQM packages if they have been updated

24.3.2.23

24.3.2.23.2.b1

Date: 02.08.2024

New features

- ADQM now supports Ubuntu OS
- Added the ***Custom JRE*** section to the cluster configuration where you can set a custom value to the `JAVA_HOME` variable
- Enabled custom JRE package installation on Astra Linux
- Added the ability to configure multiple LDAP servers
- Introduced rolling upgrade for ADQMDB since ADCM version 2.3
- Introduced rolling restart for ADQMDB since ADCM version 2.3
- Added the ***retention_time*** parameter to the ***Prometheus settings*** section of the Monitoring service configuration in ADCM

Improvements

- Moved Prometheus tasks to the cluster level

Bug fixes

- Fixed the ***Manage Kerberos*** action failure occured when there were special characters in a Kadmin password
- Fixed an issue with deleting the Monitoring service after adding an empty host

Misc/Internal

- Increased the minimum version of ADQM for upgrade to 23.3.2.37.1.b1
- Increased the minimum version of ADCM to 2.2.0
- Removed the check Baje

24.3.2.23.1.b1

Date: 25.04.2024

New features

- Increased ClickHouse version to 24.3.2.23-lts
- The minimum version of ADCM is now 2023.04.21.12
- Added the adqm-clickhouse-copier package for copying between clusters and resharding data (replacement for clickhouse-copier)
- Log output in ADCM is now available for non-starting services
- Implemented support for SSL between ZooKeeper and ADQMDB

Improvements

- Replaced old actions for Kerberos management (***Enable/Disable/Reconfig Kerberos***) with a new action — ***Manage Kerberos***
- Included LDAP options to specify `bind_dn` and `user_dn_detection`
- Added functionality to delete the Zookeeper service using the icon 🗑 on the Services page
- Implemented functionality to remove legacy monitoring
- Added the ***asynchronous_insert_log*** parameter to the ***Log settings*** section of the ADQMDB service configuration in ADCM
- Added statuschecker reinstallation for the Chproxy and Clickhousekeeper services

Bug fixes

- Fixed an error related to the `fqdn` attribute, which occurred while installing ADQM with allocated ClickHouse Keeper

Misc/Internal

- Implemented a check of the hosts in the cluster topology structure

23.8.2.7

23.8.2.7.1.b1

Date: 29.12.2023

New features

- Increased ClickHouse version to 23.8.2.7-lts
- Increased ZooKeeper version to 3.8.3
- Introduced the Prometheus-based monitoring service (the recommended way to install monitoring in ADQM) and deprecated monitoring cluster support for ADQM
- Introduced new Grafana dashboards with a new General dashboard
- Added the ***jdbc bridge log path*** setting to the ***JDBC Bridge configuration*** section of the ADQMDB service configuration in ADCM
- Implemented CA certificate support for inter-server communications. Added the ***Enable secure intra-cluster communication*** option (instead of ***Interserver protocol***) to the ***Intra-cluster communication security*** section of the ADQMDB service configuration settings
- ClickHouse errors are now displayed in ADQM jobs

Improvements

- Storage settings specified in ADCM now properly sync with *storage.xml*
- Implemented automatic creation of needed directories for local disk paths
- Added disk path validation to prevent directory conflicts
- Improved reliability in shard and replica management

Misc/Internal

- Included checks for undefined storage policies to the ***Reconfig and restart*** action of the ADQMDB service
- Implemented the ***Disable validation*** option for the ***Reconfig and restart*** action of the ADQMDB service to bypass configuration checks
- Creation of the cache over local disks is no longer supported

23.3.2.37

23.3.2.37.1.b1

Date: 21.07.2023

New features

- Increased ClickHouse version to 23.3.2.37-lts
- Implemented support for the gRPC network interface. Added the ***Enable gRPC*** section to the ***Configuration*** page of the ADQMDB service
- Extended data storage configuration options. Added the ability to:
  - store data on external S3 disks;
  - configure local cache over local and S3 disks;
  - create encrypted disks over local, S3, and caching disks.

  Added new sections to the configuration page of the ADQMDB service instead of the ***Storage Configuration*** parameter: ***Enable additional local storage***, ***Enable S3 storage***, ***Enable local cache***, ***Enable encrypted storage***, and ***Storage policies***.
- Implemented the ability to define requirements for the complexity of user passwords. Added the following parameters to the ***User password options*** section of the ADQMDB service's configuration page: ***Allow Implicit_NO_password***, ***Length of password***, ***Include Upper-case***, ***Include Lower-case***, ***Include Number***, and ***Include special symbols***
- Added the ***Encryption codecs for tables*** section to the configuration parameters of the ADQMDB service, where you can specify keys for encryption codecs
- Implemented support for RED OS in the Enterprise version of the ADQM bundle

Improvements

- Improved the security for ClickHouse directories and configuration files

Misc/Internal

- Added the check that a storage policy contains at least one volume
- Refactored the following sections of the ADQMDB service's configuration page in the ADCM interface:
  - ***Cluster configuration*** — removed the ***port***, ***user***, ***password*** replica parameters (the ***Intra-cluster communication security*** section parameters are used instead);
  - ***User password options*** — renamed the ***Access Management*** parameter to ***Allow plaintext password***.

22.8.12.45

22.8.12.45.1.b1

Date: 18.04.2023

New features

- Increased ClickHouse version to 22.8.12.45-lts
- Added support for Astra Linux to the Enterprise edition of ADQM
- Implemented the ability to configure Kerberos authentication in ADQM via the ADCM interface
- Implemented the ability to install ClickHouse Keeper as a separate service
- Added support for ClickHouse Keeper logging (a log file is saved to the */var/log/chkeeper* directory)
- Implemented support for SSL in the Chproxy service
- Added new parameters to the ***Configuration*** page of the ADQMDB service:
  - the ***Enable IPv6*** parameter in the ***Clickhouse Keeper (integrated)*** section (formerly ***Clickhouse Keeper***);
  - the ***Max server memory usage*** and ***Max thread pool size*** parameters in the ***Limits*** section;
  - the ***Interserver protocol*** and ***Interserver protocol's port*** parameters in the ***Intra-cluster communication security*** section;
  - the ***Logging level***, ***Log file size***, ***Log files count***, ***metric_log***, and ***asynchronous_metric_log*** parameters in the ***Log settings*** section;
  - the ***Time zone*** parameter in the ***Other*** section;
  - the ***User password options*** section;
  - the ***Data compression*** section;
  - the ***Kerberos configuration*** section.

Improvements

- Added the `relaxed` and `once` values for the `verificationMode` parameter and additional certificate checks depending on this parameter values
- Added new parameters to the client configuration file to work successfully with SSL/TCP enabled
- You can now pass an integer value to the ***Max concurrent threads*** parameter to specify the maximum number of query processing threads

Misc/Internal

- On the ***Configuration*** page of the ADQMDB service, the ***session_log*** parameter is removed from the ***Log settings*** section

22.8.6.71

22.8.6.71.1.b1

Date: 17.01.2023

New features

- Increased ClickHouse version to 22.8.6.71-lts
- Implemented the ***Enable auto core dump*** and ***Disable auto core dump*** actions for the ADQMDB service in the ADCM interface
- Added new parameters to the ***Configuration*** page of the ADQMDB service:
  - the ***Default user password*** parameter in the ***Default user and policy settings*** section;
  - the ***Log path*** parameter in the ***Directories*** section;
  - the ***Max concurrent threads*** parameter in the ***Limits*** section;
  - the ***Auto Core Dump Service*** parameter in the ***Other*** section;
  - the ***Intra-cluster communication security*** section.

Improvements

- Implemented a set of improvements to increase the security at the level of interaction between cluster hosts, SSL support, Chproxy and ClickHouse interaction

Misc/Internal

- Refactored the ***Configuration*** page of the ADQMDB service in the ADCM interface: improved descriptions of some parameters, deleted the ***external_dictionaries_path*** and ***external_dictionaries*** parameters from the ***Other*** section, and moved the ***Other*** section to the end of the page
- Extended a set of pre-checks before an ADQM cluster installation

22.8.4.7

22.8.4.7.1.b1

Date: 05.10.2022

New features

- Implemented the `concurrent_threads_soft_limit` parameter that allows you to limit the number of query processing threads
- Added the ***Disable SElinux*** and ***Disable Firewalld*** options to the cluster configuration page in the ADCM interface

Misc/Internal

- Excluded the Baje proxy service from ADQM
- Added Alt Linux version check before installing ADQM

22.3.7.28

22.3.7.28.1.b1

Date: 12.08.2022

New features

- Added the Chproxy service

Bug fixes

- Fixed an error occurred when upgrading ADQM on Alt Linux
- Fixed incorrect switching of a cluster status from `upgradable` to `installed` during the cluster upgrade

Misc/Internal

- Changed ADQM package names so that they differ from names of ClickHouse packages

22.3.2.2

22.3.2.2.1.b1

Date: 26.05.2022

New features

- Increased ClickHouse version to 22.3.2.2-lts
- Added support for Alt Linux 8.4 SP
- Added the ClickHouse Keeper component
- Expanded management options for built-in logging tools

Improvements

- Added checks of monitoring services installed in an Enterprise Tools cluster before importing monitoring settings into an ADQM cluster

21.8.13.6

21.8.13.6.1.b2

Date: 17.01.2023

Bug fixes

- Fixed: ADQM bundle upgrade could lead to loss of cluster configuration data if the ***Default cluster topology*** field was set to an incorrect cluster topology (for example, shards and/or replicas were unsorted or a shard/replica identifier contained a space)

21.8.13.6.1.b1

Date: 28.01.2022

New features

- Implemented the ***Reinstall status-checker*** action for a cluster in the `installed` state. It runs the *status-checker* deployment scripts for services as well as for Docker containers
- Added the ***Advanced configuration parameters*** section to the ***Configuration*** page of the ADQMDB service

Bug fixes

- Fixed the Baje proxy service installation error

21.8.8.29

21.8.8.29.1.b2

Date: 07.12.2021

Bug fixes

- Fixed an error with offline installation of the ZooKeeper package

21.8.8.29.1.b1

Date: 28.10.2021

New features

- Increased ClickHouse version to 21.8.8.29-lts
- Increased JDBC bridge version to 2.0.6
- Increased ODBC driver version to 1.1.10
- Added the ability to map roles from an LDAP directory to ADQM roles and assign these roles to authenticated users (see LDAP authentication)

21.3.14.1

21.3.14.1.1.b1

Date: 20.09.2021

New features

- Implemented the ***Expand*** and ***Shrink*** actions for the Monitoring Clients service
- Added support for Alt Linux 8.2

Misc/Internal

- Added retries to check JDBC bridge
- Changed the logic of adding the `log_queries` parameter to the `default` settings profile in the *users.xml* file

21.3.12.2

21.3.12.2.1.b2

Date: 23.07.2021

New features

- Added support for RHEL

21.3.12.2.1.b1

Date: 10.06.2021

Improvements

- Included checking of component versions to the ***Check*** action

Bug fixes

- Fixed an issue with importing the HTTP Mirror service of an Enterprise Tools cluster into an ADQM cluster

21.3.11.5

21.3.11.5.1.b1

Date: 26.05.2021

New features

- Increased ClickHouse version to 21.3.11.5

21.3.4.25

21.3.4.25.1.b1

Date: 29.04.2021

New features

- Increased ClickHouse version to 21.3.4.25-lts
- Implemented the ability to install ADQM without Internet access (see the Offline installation section for details)

Bug fixes

- Fixed a Zookeeper service error occurred when performing the ***Expand*** action
- Fixed an issue with skipping ADQMDB dashboards when installing monitoring
- Fixed an issue with loading more than one Avro-formatted message from Kafka into a ClickHouse table

20.8.13.15

20.8.13.15.1.b1

Date: 05.04.2021

New features

- Increased ClickHouse version to 20.8.13.15-lts
- Increased JDBC bridge version to 2.0
- Implemented the ability to use an LDAP server for user authentication

Improvements

- Added Zookeeper check after installation

Bug fixes

- Fixed an issue with installing an ADQM cluster when using the external ZooKeeper

20.8.11.17

20.8.11.17.1.b2

Date: 08.02.2021

Bug fixes

- Fixed an error in the *users.xml* file: the same IP address was duplicated in the `networks` list

20.8.11.17.1.b1

Date: 27.01.2021

New features

- Increased ClickHouse version to 20.8.11.17-lts
- Added parameters to configure a default user and a default settings profile

Misc/Internal

- Implemented creation of macros for multiple cluster topologies
- Deleted the ***Create user*** and ***Delete users*** buttons

20.8.4.11

20.8.4.11.1.b1

Date: 20.10.2020

New features

- Increased ClickHouse version to 20.8.4.11-lts

20.3.20.6

20.3.20.6.1.b1

Date: 20.10.2020

New features

- Increased ClickHouse version to 20.3.20.6-lts

20.8.2.3

20.8.2.3.1.b1

Date: 09.10.2020

New features

- Increased ClickHouse version to 20.8.2.3-stable
- Added the ability to specify Kafka configuration parameters

20.3.13.130

20.3.13.130.1.b1

Date: 27.08.2020

New features

- Increased ClickHouse version to 20.3.13.130-lts
- Added the following parameters to the ***Configuration*** page of the ADQMDB service in the ADCM interface: ***Cluster Configuration***, ***Storage Configuration***, ***User policy***, ***query_log***

Misc/Internal

- Updated the maximum version of Enterprise Tools (ET) to 2020022800
- Added the ability to specify a username and password for replicas in the `remote_servers` section of the *config.xml* configuration file

20.3.10.75

20.3.10.75.1.b2

Date: 06.08.2020

Bug fixes

- Fixed: incorrect minimum version for an upgrade

20.3.10.75

Date: 11.06.2020

New features

- Increased ClickHouse version to 20.3.10.75

Bug fixes

- Fixed Zookeeper service check error occurred when performing the ***Check*** action for a cluster

19.14.11.16

19.14.11.16.1.b3

Date: 11.08.2020

Misc/Internal

- Updated the maximum version of Enterprise Tools (ET) to 2020022800

19.14.11.16.1.b2

Date: 04.08.2020

Improvements

- Refactored the ADQM bundle

19.14.11.16

Date: 14.05.2020

New features

- Increased ClickHouse version to 19.14.11.16-lts
- Added support for Alt Linux
- Divided the ADQM bundle into the Community and Enterprise versions
- Updated EULA for the Community version
- Added a window to sign the license agreement after downloading the bundle
- Added the Clickhouse JDBC Bridge component to the ADQMDB service
- Implemented the ability to use an external ZooKeeper

ADQM known issues

Elena Dvoryadkina

To Table of Contents

Issue	From	Fixed	Workaround
Some queries cannot be terminated by the `KILL QUERY` command (for example, if a query contains a subquery that calculates a scalar value)	19.14.11.16	—	Restart the ClickHouse server (see details)