ADS

Arenadata Streaming

Arenadata Streaming (ADS) is a scalable, fault-tolerant solution for real-time streaming data processing, adapted for enterprise and built on Apache Kafka and Apache NiFi.

Version: CURRENT Language: EN

Contents

1. Get started
- Online installation
  - Install ADCM
  - Prepare hosts
  - Install ADS cluster
    - Create a cluster
    - Add services
    - Add hosts to a cluster
    - Add components
    - Configure services
    - Configure a cluster
    - Install a cluster
  - Install monitoring
    - Way 1. Monitoring service
    - Way 2. Monitoring cluster
      - Create a cluster
      - Add services
      - Add hosts to a cluster
      - Add components
      - Configure services
      - Install a cluster
      - Integrate with ADS cluster
- 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 ADS 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
    - Way 1. Monitoring service
    - Way 2. Monitoring cluster
      - Configure Enterprise Tools cluster
      - Integrate with ADS cluster

2. How to
- Kafka
  - Quick start
  - Access management
    - Authentication
      - SASL PLAINTEXT
      - Kerberos
        - MIT Kerberos
        - MS Active Directory
      - SSL channel security
    - Authorization
      - ADPS
        - Ranger Kafka plugin
        - Use Ranger in Kafka
      - ACL in Kafka
  - Usage examples
    - Write a Kafka Streams application
    - Use API
    - Cruise Control
  - Installation in the KRaft mode
  - Kafka migration to KRaft
  - Configure and use Kafka Tiered Storage
- Kafka Connect
  - Mirror Maker 2
- Kafka REST Proxy
  - Usage example
  - Work with kafka-rest-security
- ksqlDB
  - Usage example
- NiFi
  - UI overview
    - NiFi Server
    - NiFi Registry
  - Access management
    - Authentication
    - Authorization
      - User interface
      - ADPS
        - Ranger NiFi plugin
        - Use Ranger in NiFi
  - Basic operations
    - Work with processors
      - Configure processors and process groups
    - Work with attributes
    - Work with templates
    - Built-in monitoring
  - Use Expression Language
  - Encryption of configuration parameters
  - Usage examples
    - Use REST API
    - Build a dataflow
    - NiFi ADB Connector
      - Overview
      - Usage example
- Schema Registry
  - Usage example

3. Tutorials
- ADS cluster management
  - Cluster actions
  - Service actions
    - Kafka
    - Kafka Connect
    - Kafka REST Proxy
    - ksqlDB
    - NiFi
    - NiFi2
    - MiNiFi
    - Schema Registry
    - ZooKeeper
    - Monitoring Clients
    - Monitoring
- Access management
  - Authentication
    - Basic
    - Kerberos
      - MIT Kerberos
      - MS Active Directory
      - Samba
      - Custom kerberization settings
    - SASL PLAINTEXT
    - SSL channel security
  - Authorization
    - ADPS
      - Overview
      - Integration with ADS cluster
- Monitoring
  - Metrics

4. References
- ADS configuration parameters
- Kafka
  - Clients
  - API
- NiFi
  - API

5. Planning guide
- Hardware requirements
- Network requirements
- Software requirements
- Best practices
  - Kafka
  - NiFi

6. Concepts
- ADS overview
  - Use cases
- Architecture
  - Overview
  - Kafka
    - Storage concepts
    - Log compaction
    - Message delivery guarantees
    - Replication
    - KRaft overview
    - Tiered Storage overview
    - kTLS
    - Cruise Control overview
  - Kafka Connect
    - Overview
    - Debezium overview
  - Kafka REST Proxy
    - Overview
  - ksqlDB
    - Overview
  - MiNiFi
    - Overview
  - NiFi
    - Overview
  - Schema Registry
    - Overview
  - ZooKeeper
    - Service discovery
    - Leader election
- Data model
  - Kafka
    - Message format
  - NiFi
    - Objects
    - Processor types

7. Release notes
- ADS releases
- ADS supported services
- Version compatibility matrix

Get started with Arenadata Streaming

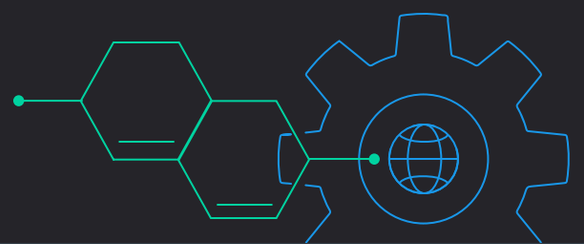
To Table of Contents

This tutorial guides you through the process of installing an Arenadata Streaming (ADS) 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 ADS 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 ADS. It can be applied to the hosts with access to Internet.

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

Step 1.

Install ADCM

Step 2.

Prepare hosts

Step 3.

Install ADS cluster

Step 4.

Install monitoring

Install AGCM

Contents

Step 1. Check prerequisites
- Software and business requirements
- Hardware requirements
- Install Docker
- Install Docker Compose
- Install Apache Kafka
- Check the Docker customer status
- Check the AGCM port availability

Step 2. Install AGCM
- Check the AGCM interface
- Set the AGCM URL
- Check the AGCM health

Step 3. Start AGCM

Step 4. Check installation

Step 5. Update AGCM

Step 1. Check prerequisites

Software and business requirements

Install the following software and business requirements on your server:

- Docker
- Docker Compose
- Apache Kafka
- Java
- Git
- Curl
- Netcat (`nc`)
- Telnet (`telnet`)
- Docker Hub credentials

Contact your Genesys Sales Representative if you do not have the required Docker Hub credentials. This allows you to download the containers you need to run Genesys Multicloud CX private edition services.

Make sure that you have the following information from your Genesys Sales Representative before you start installing AGCM:

- The Docker tag for the AGCM container image

Hardware requirements

Allocate the following CPU and memory resources for AGCM:

CPU	Memory
2 CPU	4 GB

Install Docker

Install Docker Engine on a supported supported platform. See Docker's documentation for more information.

Verify that Docker Engine is installed correctly by running the `hello-world` image. This command downloads a test image, runs it in a container, prints an informational message, and exits:

```
sudo docker run hello-world
```

Install Docker Compose on RHEL 7 Only

Install Docker Compose on RHEL 7 only:

```
sudo curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose
```

Install Kafka

Install Apache Kafka on a supported supported platform. See Kafka's documentation for more information.

Verify that Kafka is installed correctly by running the following command:

```
sudo systemctl status kafka
```

Install Java

Install Java on a supported supported platform. See Java's documentation for more information.

Verify that Java is installed correctly by running the following command:

```
sudo java -version
```

Install Git

Install Git on a supported supported platform. See Git's documentation for more information.

Verify that Git is installed correctly by running the following command:

```
sudo git --version
```

Install Curl

Install Curl on a supported supported platform. See Curl's documentation for more information.

Verify that Curl is installed correctly by running the following command:

```
sudo curl --version
```

Install Netcat

Install Netcat on a supported supported platform. See Netcat's documentation for more information.

Verify that Netcat is installed correctly by running the following command:

```
sudo nc -h
```

Install Telnet

Install Telnet on a supported supported platform. See Telnet's documentation for more information.

Verify that Telnet is installed correctly by running the following command:

```
sudo telnet --version
```

Docker Hub credentials

Contact your Genesys Sales Representative if you do not have the required Docker Hub credentials. This allows you to download the containers you need to run Genesys Multicloud CX private edition services.

Docker tag for the AGCM container image

Contact your Genesys Sales Representative before you start installing AGCM to get the Docker tag for the AGCM container image.

Disable SELinux

RHEL

To disable SELinux persistently, edit the `/etc/selinux/config` file and change the `SELINUX` line to `permissive`:

```
# 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 - No SELinux policy is loaded.
SELINUX=permissive
# SELINUXTYPE= can take one of these two values:
#     targeted - Targeted processes are protected,
#     mls - Multi Level Security protection.
SELINUXTYPE=targeted
```

To verify that SELinux is disabled, run the following command:

```
sudo sestatus
```

The command output should look like this example:

```
SELinux status:                 disabled
```

Ubuntu

To disable SELinux persistently, edit the `/etc/apparmor.d/usr.sbin.mysqld` file and add the following lines:

```
/usr/sbin/mysqld {
  ...
  /var/lib/mysql/ r,
  /var/lib/mysql/** rwk,
  ...
}
```

Then restart AppArmor:

```
sudo systemctl restart apparmor
```

To verify that SELinux is disabled, run the following command:

```
sudo aa-status
```

If SELinux is disabled, the command output should look like this example:

```
apparmor module is loaded.
0 profiles are loaded.
0 profiles are in enforce mode.
0 profiles are in complain mode.
0 processes have profiles defined.
0 processes are in enforce mode.
0 processes are in complain mode.
0 processes are unconfined but have a profile defined.
```

Allow access to the Docker CLI without root privileges

Allow access to the Docker CLI without root privileges. Add the `genesys` user to the `docker` group:

```
sudo usermod -aG docker genesys
```

Log out and log back in for the changes to take effect.

Step 2. Install AGCM

Download

Download the AGCM package from the Genesys Multicloud CX Private Edition page in the JFrog Artifactory software repository. Search for `agcm` and download the package for the release that you want to install.

Extract the AGCM installation files:

```
tar -xvf agcm-<version>.tar
```

Environment variables

Create a file named `agcm.env` and add the following environment variables:

Variable	Description
`AGCM_CONTAINER_REGISTRY`	The container registry that hosts the AGCM container image.
`AGCM_CONTAINER_REPOSITORY`	The container repository that hosts the AGCM container image.
`AGCM_CONTAINER_TAG`	The container tag for the AGCM container image.
`AGCM_CONTAINER_PULL_SECRET`	The name of the secret that contains the credentials for the container registry.

Example:

```
AGCM_CONTAINER_REGISTRY=pureengage-docker-staging.jfrog.io
AGCM_CONTAINER_REPOSITORY=genesys
AGCM_CONTAINER_TAG=agcm-1.0.0-1
AGCM_CONTAINER_PULL_SECRET=jfrog-secret
```

Environment variables

Variable	Description
`TZ`	The time zone of the host that the container runs on.
`LOG_LEVEL`	The level of logs that are produced by AGCM.
`LOG_FORMAT`	The format of the logs that are produced by AGCM.
`KAFKA_SECURITY_PROTOCOL`	The protocol that is used to communicate with Kafka. Possible values: - `PLAINTEXT` - Unencrypted traffic. - `SSL` - Encrypted traffic using SSL. - `SASL_PLAINTEXT` - Unencrypted traffic using SASL. - `SASL_SSL` - Encrypted traffic using SASL and SSL.
`KAFKA_SASL_MECHANISM`	The SASL mechanism that is used to communicate with Kafka. Possible values: - `PLAIN` - Simple username and password authentication. - `SCRAM-SHA-256` - Username and password authentication with SCRAM-SHA-256. - `SCRAM-SHA-512` - Username and password authentication with SCRAM-SHA-512.
`KAFKA_SASL_USERNAME`	The username that is used to authenticate with Kafka. This is required if you set `KAFKA_SECURITY_PROTOCOL` to `SASL_PLAINTEXT` or `SASL_SSL`.
`KAFKA_SASL_PASSWORD`	The password that is used to authenticate with Kafka. This is required if you set `KAFKA_SECURITY_PROTOCOL` to `SASL_PLAINTEXT` or `SASL_SSL`.
`KAFKA_SSL_CA_CERT`	The CA certificate that is used to verify the Kafka broker's certificate. This is required if you set `KAFKA_SECURITY_PROTOCOL` to `SSL` or `SASL_SSL`.
`KAFKA_SSL_CLIENT_CERT`	The client certificate that is used to authenticate with Kafka. This is required if you set `KAFKA_SECURITY_PROTOCOL` to `SSL` or `SASL_SSL`.
`KAFKA_SSL_CLIENT_KEY`	The client key that is used to authenticate with Kafka. This is required if you set `KAFKA_SECURITY_PROTOCOL` to `SSL` or `SASL_SSL`.
`KAFKA_SSL_CLIENT_KEY_PASSWORD`	The password that is used to decrypt the client key. This is required if you set `KAFKA_SECURITY_PROTOCOL` to `SSL` or `SASL_SSL`.
`KAFKA_SSL_CRL_CHECK`	Whether to check the certificate revocation list (CRL) of the Kafka broker. Possible values: - `true` - `false` (default value)
`KAFKA_SSL_CRL_CACHE_TTL`	The time-to-live (TTL) of the CRL cache, in milliseconds. The default value is `3600000` (1 hour).
`KAFKA_SSL_CRL_CACHE_SIZE`	The size of the CRL cache. The default value is `1000`.
`KAFKA_SSL_OCSP_CHECK`	Whether to check the online certificate status protocol (OCSP) of the Kafka broker. Possible values: - `true` - `false` (default value)
`KAFKA_SSL_OCSP_CACHE_TTL`	The time-to-live (TTL) of the OCSP cache, in milliseconds. The default value is `3600000` (1 hour).
`KAFKA_SSL_OCSP_CACHE_SIZE`	The size of the OCSP cache. The default value is `1000`.
`KAFKA_SSL_OCSP_FAIL_OPEN`	Whether to fail open if the OCSP check fails. Possible values: - `true` (default value) - `false`
`KAFKA_SSL_OCSP_TIMEOUT`	The timeout for OCSP checks, in milliseconds. The default value is `5000` (5 seconds).
`KAFKA_SSL_OCSP_CONNECT_TIMEOUT`	The connection timeout for OCSP checks, in milliseconds. The default value is `5000` (5 seconds).
`KAFKA_SSL_OCSP_READ_TIMEOUT`	The read timeout for OCSP checks, in milliseconds. The default value is `5000` (5 seconds).
`KAFKA_SSL_OCSP_RESPONDER_URL`	The URL of the OCSP responder.
`KAFKA_SSL_OCSP_RESPONDER_CERT`	The certificate of the OCSP responder.
`KAFKA_SSL_OCSP_NONCE`	Whether to use a nonce in OCSP requests. Possible values: - `true` (default value) - `false`
`KAFKA_SSL_OCSP_CERT_STATUS`	The certificate status that is required for OCSP checks. Possible values: - `GOOD` (default value) - `REVOKED` - `UNKNOWN`
`KAFKA_SSL_CLIENT_CERT_ALIAS`	The alias of the client certificate.
`KAFKA_SSL_CLIENT_KEY_ALIAS`	The alias of the client key.
`KAFKA_SSL_CLIENT_KEY_PASSWORD_ALIAS`	The alias of the password that is used to decrypt the client key.
`KAFKA_SSL_CLIENT_CERT_ALIAS_PEM`	The alias of the client certificate in PEM format.
`KAFKA_SSL_CLIENT_KEY_ALIAS_PEM`	The alias of the client key in PEM format.
`KAFKA_SSL_CLIENT_KEY_PASSWORD_ALIAS_PEM`	The alias of the password that is used to decrypt the client key in PEM format.
`KAFKA_SSL_CLIENT_CERT_ALIAS_PKCS12`	The alias of the client certificate in PKCS12 format.
`KAFKA_SSL_CLIENT_KEY_ALIAS_PKCS12`	The alias of the client key in PKCS12 format.
`KAFKA_SSL_CLIENT_KEY_PASSWORD_ALIAS_PKCS12`	The alias of the password that is used to decrypt the client key in PKCS12 format.
`KAFKA_SSL_CLIENT_CERT_ALIAS_JKS`	The alias of the client certificate in JKS format.
`KAFKA_SSL_CLIENT_KEY_ALIAS_JKS`	The alias of the client key in JKS format.
`KAFKA_SSL_CLIENT_KEY_PASSWORD_ALIAS_JKS`	The alias of the password that is used to decrypt the client key in JKS format.
`KAFKA_SSL_CLIENT_CERT_ALIAS_PKCS11`	The alias of the client certificate in PKCS11 format.
`KAFKA_SSL_CLIENT_KEY_ALIAS_PKCS11`	The alias of the client key in PKCS11 format.
`KAFKA_SSL_CLIENT_KEY_PASSWORD_ALIAS_PKCS11`	The alias of the password that is used to decrypt the client key in PKCS11 format.
`KAFKA_SSL_CLIENT_CERT_ALIAS_PKCS11_PEM`	The alias of the client certificate in PKCS11 PEM format.
`KAFKA_SSL_CLIENT_KEY_ALIAS_PKCS11_PEM`	The alias of the client key in PKCS11 PEM format.
`KAFKA_SSL_CLIENT_KEY_PASSWORD_ALIAS_PKCS11_PEM`	The alias of the password that is used to decrypt the client key in PKCS11 PEM format.
`KAFKA_SSL_CLIENT_CERT_ALIAS_PKCS11_PKCS12`	The alias of the client certificate in PKCS11 PKCS12 format.
`KAFKA_SSL_CLIENT_KEY_ALIAS_PKCS11_PKCS12`	The alias of the client key in PKCS11 PKCS12 format.
`KAFKA_SSL_CLIENT_KEY_PASSWORD_ALIAS_PKCS11_PKCS12`	The alias of the password that is used to decrypt the client key in PKCS11 PKCS12 format.
`KAFKA_SSL_CLIENT_CERT_ALIAS_PKCS11_JKS`	The alias of the client certificate in PKCS11 JKS format.
`KAFKA_SSL_CLIENT_KEY_ALIAS_PKCS11_JKS`	The alias of the client key in PKCS11 JKS format.
`KAFKA_SSL_CLIENT_KEY_PASSWORD_ALIAS_PKCS11_JKS`	The alias of the password that is used to decrypt the client key in PKCS11 JKS format.
`KAFKA_SSL_CLIENT_CERT_ALIAS_PKCS11_PKCS11`	The alias of the client certificate in PKCS11 PKCS11 format.
`KAFKA_SSL_CLIENT_KEY_ALIAS_PKCS11_PKCS11`	The alias of the client key in PKCS11 PKCS11 format.
`KAFKA_SSL_CLIENT_KEY_PASSWORD_ALIAS_PKCS11_PKCS11`	The alias of the password that is used to decrypt the client key in PKCS11 PKCS11 format.
`KAFKA_SSL_CLIENT_CERT_ALIAS_PKCS11_PKCS11_PEM`	The alias of the client certificate in PKCS11 PKCS11 PEM format.
`KAFKA_SSL_CLIENT_KEY_ALIAS_PKCS11_PKCS11_PEM`	The alias of the client key in PKCS11 PKCS11 PEM format.
`KAFKA_SSL_CLIENT_KEY_PASSWORD_ALIAS_PKCS11_PKCS11_PEM`	The alias of the password that is used to decrypt the client key in PKCS11 PKCS11 PEM format.
`KAFKA_SSL_CLIENT_CERT_ALIAS_PKCS11_PKCS11_PKCS12`	The alias of the client certificate in PKCS11 PKCS11 PKCS12 format.
`KAFKA_SSL_CLIENT_KEY_ALIAS_PKCS11_PKCS11_PKCS12`	The alias of the client key in PKCS11 PKCS11 PKCS12 format.
`KAFKA_SSL_CLIENT_KEY_PASSWORD_ALIAS_PKCS11_PKCS11_PKCS12`	The alias of the password that is used to decrypt the client key in PKCS11 PKCS11 PKCS12 format.
`KAFKA_SSL_CLIENT_CERT_ALIAS_PKCS11_PKCS11_JKS`	The alias of the client certificate in PKCS11 PKCS11 JKS format.
`KAFKA_SSL_CLIENT_KEY_ALIAS_PKCS11_PKCS11_JKS`	The alias of the client key in PKCS11 PKCS11 JKS format.
`KAFKA_SSL_CLIENT_KEY_PASSWORD_ALIAS_PKCS11_PKCS11_JKS`	The alias of the password that is used to decrypt the client key in PKCS11 PKCS11 JKS format.
`KAFKA_SSL_CLIENT_CERT_ALIAS_PKCS11_PKCS11_PKCS11`	The alias of the client certificate in PKCS11 PKCS11 PKCS11 format.
`KAFKA_SSL_CLIENT_KEY_ALIAS_PKCS11_PKCS11_PKCS11`	The alias of the client key in PKCS11 PKCS11 PKCS11 format.
`KAFKA_SSL_CLIENT_KEY_PASSWORD_ALIAS_PKCS11_PKCS11_PKCS11`	The alias of the password that is used to decrypt the client key in PKCS11 PKCS11 PKCS11 format.
`KAFKA_SSL_CLIENT_CERT_ALIAS_PKCS11_PKCS11_PKCS11_PEM`	The alias of the client certificate in PKCS11 PKCS11 PKCS11 PEM format.
`KAFKA_SSL_CLIENT_KEY_ALIAS_PKCS11_PKCS11_PKCS11_PEM`	The alias of the client key in PKCS11 PKCS11 PKCS11 PEM format.
`KAFKA_SSL_CLIENT_KEY_PASSWORD_ALIAS_PKCS11_PKCS11_PKCS11_PEM`	The alias of the password that is used to decrypt the client key in PKCS11 PKCS11 PKCS11 PEM format.
`KAFKA_SSL_CLIENT_CERT_ALIAS_PKCS11_PKCS11_PKCS11_PKCS12`	The alias of the client certificate in PKCS11 PKCS11 PKCS11 PKCS12 format.
`KAFKA_SSL_CLIENT_KEY_ALIAS_PKCS11_PKCS11_PKCS11_PKCS12`	The alias of the client key in PKCS11 PKCS11 PKCS11 PKCS12 format.
`KAFKA_SSL_CLIENT_KEY_PASSWORD_ALIAS_PKCS11_PKCS11_PKCS11_PKCS12`	The alias of the password that is used to decrypt the client key in PKCS11 PKCS11 PKCS11 PKCS12 format.
`KAFKA_SSL_CLIENT_CERT_ALIAS_PKCS11_PKCS11_PKCS11_JKS`	The alias of the client certificate in PKCS11 PKCS11 PKCS11 JKS format.
`KAFKA_SSL_CLIENT_KEY_ALIAS_PKCS11_PKCS11_PKCS11_JKS`	The alias of the client key in PKCS11 PKCS11 PKCS11 JKS format.
`KAFKA_SSL_CLIENT_KEY_PASSWORD_ALIAS_PKCS11_PKCS11_PKCS11_JKS`	The alias of the password that is used to decrypt the client key in PKCS11 PKCS11 PKCS11 JKS format.
`KAFKA_SSL_CLIENT_CERT_ALIAS_PKCS11_PKCS11_PKCS11_PKCS11`	The alias of the client certificate in PKCS11 PKCS11 PKCS11 PKCS11 format.
`KAFKA_SSL_CLIENT_KEY_ALIAS_PKCS11_PKCS11_PKCS11_PKCS11`	The alias of the client key in PKCS11 PKCS11 PKCS11 PKCS11 format.
`KAFKA_SSL_CLIENT_KEY_PASSWORD_ALIAS_PKCS11_PKCS11_PKCS11_PKCS11`	The alias of the password that is used to decrypt the client key in PKCS11 PKCS11 PKCS11 PKCS11 format.
`KAFKA_SSL_CLIENT_CERT_ALIAS_PKCS11_PKCS11_PKCS11_PKCS11_PEM`	The alias of the client certificate in PKCS11 PKCS11 PKCS11 PKCS11 PEM format.
`KAFKA_SSL_CLIENT_KEY_ALIAS_PKCS11_PKCS11_PKCS11_PKCS11_PEM`	The alias of the client key in PKCS11 PKCS11 PKCS11 PKCS11 PEM format.
`KAFKA_SSL_CLIENT_KEY_PASSWORD_ALIAS_PKCS11_PKCS11_PKCS11_PKCS11_PEM`	The alias of the password that is used to decrypt the client key in PKCS11 PKCS11 PKCS11 PKCS11 PEM format.
`KAFKA_SSL_CLIENT_CERT_ALIAS_PKCS11_PKCS11_PKCS11_PKCS11_PKCS12`	The alias of the client certificate in PKCS11 PKCS11 PKCS11 PKCS11 PKCS12 format.
`KAFKA_SSL_CLIENT_KEY_ALIAS_PKCS11_PKCS11_PKCS11_PKCS11_PKCS12`	The alias of the client key in PKCS11 PKCS11 PKCS11 PKCS11 PKCS12 format.
`KAFKA_SSL_CLIENT_KEY_PASSWORD_ALIAS_PKCS11_PKCS11_PKCS11_PKCS11_PKCS12`	The alias of the password that is used to decrypt the client key in PKCS11 PKCS11 PKCS11 PKCS11 PKCS12 format.
`KAFKA_SSL_CLIENT_CERT_ALIAS_PKCS11_PKCS11_PKCS11_PKCS11_JKS`	The alias of the client certificate in PKCS11 PKCS11 PKCS11 PKCS11 JKS format.
`KAFKA_SSL_CLIENT_KEY_ALIAS_PKCS11_PKCS11_PKCS11_PKCS11_JKS`	The alias of the client key in PKCS11 PKCS11 PKCS11 PKCS11 JKS format.
`KAFKA_SSL_CLIENT_KEY_PASSWORD_ALIAS_PKCS11_PKCS11_PKCS11_PKCS11_JKS`	The alias of the password that is used to decrypt the client key in PKCS11 PKCS11 PKCS11 PKCS11 JKS format.
`KAFKA_SSL_CLIENT_CERT_ALIAS_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11`	The alias of the client certificate in PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 format.
`KAFKA_SSL_CLIENT_KEY_ALIAS_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11`	The alias of the client key in PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 format.
`KAFKA_SSL_CLIENT_KEY_PASSWORD_ALIAS_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11`	The alias of the password that is used to decrypt the client key in PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 format.
`KAFKA_SSL_CLIENT_CERT_ALIAS_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PEM`	The alias of the client certificate in PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PEM format.
`KAFKA_SSL_CLIENT_KEY_ALIAS_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PEM`	The alias of the client key in PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PEM format.
`KAFKA_SSL_CLIENT_KEY_PASSWORD_ALIAS_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PEM`	The alias of the password that is used to decrypt the client key in PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PEM format.
`KAFKA_SSL_CLIENT_CERT_ALIAS_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS12`	The alias of the client certificate in PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS12 format.
`KAFKA_SSL_CLIENT_KEY_ALIAS_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS12`	The alias of the client key in PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS12 format.
`KAFKA_SSL_CLIENT_KEY_PASSWORD_ALIAS_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS12`	The alias of the password that is used to decrypt the client key in PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS12 format.
`KAFKA_SSL_CLIENT_CERT_ALIAS_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_JKS`	The alias of the client certificate in PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 JKS format.
`KAFKA_SSL_CLIENT_KEY_ALIAS_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_JKS`	The alias of the client key in PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 JKS format.
`KAFKA_SSL_CLIENT_KEY_PASSWORD_ALIAS_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_JKS`	The alias of the password that is used to decrypt the client key in PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 JKS format.
`KAFKA_SSL_CLIENT_CERT_ALIAS_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11`	The alias of the client certificate in PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 format.
`KAFKA_SSL_CLIENT_KEY_ALIAS_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11`	The alias of the client key in PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 format.
`KAFKA_SSL_CLIENT_KEY_PASSWORD_ALIAS_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11`	The alias of the password that is used to decrypt the client key in PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 format.
`KAFKA_SSL_CLIENT_CERT_ALIAS_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PEM`	The alias of the client certificate in PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PEM format.
`KAFKA_SSL_CLIENT_KEY_ALIAS_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PEM`	The alias of the client key in PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PEM format.
`KAFKA_SSL_CLIENT_KEY_PASSWORD_ALIAS_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PEM`	The alias of the password that is used to decrypt the client key in PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PEM format.
`KAFKA_SSL_CLIENT_CERT_ALIAS_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS12`	The alias of the client certificate in PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS12 format.
`KAFKA_SSL_CLIENT_KEY_ALIAS_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS12`	The alias of the client key in PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS12 format.
`KAFKA_SSL_CLIENT_KEY_PASSWORD_ALIAS_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS12`	The alias of the password that is used to decrypt the client key in PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS12 format.
`KAFKA_SSL_CLIENT_CERT_ALIAS_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_JKS`	The alias of the client certificate in PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 JKS format.
`KAFKA_SSL_CLIENT_KEY_ALIAS_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_JKS`	The alias of the client key in PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 JKS format.
`KAFKA_SSL_CLIENT_KEY_PASSWORD_ALIAS_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_JKS`	The alias of the password that is used to decrypt the client key in PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 JKS format.
`KAFKA_SSL_CLIENT_CERT_ALIAS_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11`	The alias of the client certificate in PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 format.
`KAFKA_SSL_CLIENT_KEY_ALIAS_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11`	The alias of the client key in PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 format.
`KAFKA_SSL_CLIENT_KEY_PASSWORD_ALIAS_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11`	The alias of the password that is used to decrypt the client key in PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 format.
`KAFKA_SSL_CLIENT_CERT_ALIAS_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PEM`	The alias of the client certificate in PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PEM format.
`KAFKA_SSL_CLIENT_KEY_ALIAS_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PEM`	The alias of the client key in PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PEM format.
`KAFKA_SSL_CLIENT_KEY_PASSWORD_ALIAS_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PEM`	The alias of the password that is used to decrypt the client key in PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PEM format.
`KAFKA_SSL_CLIENT_CERT_ALIAS_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS12`	The alias of the client certificate in PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS12 format.
`KAFKA_SSL_CLIENT_KEY_ALIAS_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS12`	The alias of the client key in PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS12 format.
`KAFKA_SSL_CLIENT_KEY_PASSWORD_ALIAS_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS12`	The alias of the password that is used to decrypt the client key in PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS12 format.
`KAFKA_SSL_CLIENT_CERT_ALIAS_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_JKS`	The alias of the client certificate in PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 JKS format.
`KAFKA_SSL_CLIENT_KEY_ALIAS_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_JKS`	The alias of the client key in PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 JKS format.
`KAFKA_SSL_CLIENT_KEY_PASSWORD_ALIAS_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_JKS`	The alias of the password that is used to decrypt the client key in PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 JKS format.
`KAFKA_SSL_CLIENT_CERT_ALIAS_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11`	The alias of the client certificate in PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 format.
`KAFKA_SSL_CLIENT_KEY_ALIAS_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11`	The alias of the client key in PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 format.
`KAFKA_SSL_CLIENT_KEY_PASSWORD_ALIAS_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11`	The alias of the password that is used to decrypt the client key in PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 format.
`KAFKA_SSL_CLIENT_CERT_ALIAS_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PEM`	The alias of the client certificate in PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PEM format.
`KAFKA_SSL_CLIENT_KEY_ALIAS_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PEM`	The alias of the client key in PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PEM format.
`KAFKA_SSL_CLIENT_KEY_PASSWORD_ALIAS_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PEM`	The alias of the password that is used to decrypt the client key in PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PEM format.
`KAFKA_SSL_CLIENT_CERT_ALIAS_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS12`	The alias of the client certificate in PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS12 format.
`KAFKA_SSL_CLIENT_KEY_ALIAS_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS12`	The alias of the client key in PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS12 format.
`KAFKA_SSL_CLIENT_KEY_PASSWORD_ALIAS_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS12`	The alias of the password that is used to decrypt the client key in PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS12 format.
`KAFKA_SSL_CLIENT_CERT_ALIAS_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_JKS`	The alias of the client certificate in PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 JKS format.
`KAFKA_SSL_CLIENT_KEY_ALIAS_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_JKS`	The alias of the client key in PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 JKS format.
`KAFKA_SSL_CLIENT_KEY_PASSWORD_ALIAS_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_JKS`	The alias of the password that is used to decrypt the client key in PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 JKS format.
`KAFKA_SSL_CLIENT_CERT_ALIAS_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11`	The alias of the client certificate in PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 format.
`KAFKA_SSL_CLIENT_KEY_ALIAS_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11`	The alias of the client key in PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 format.
`KAFKA_SSL_CLIENT_KEY_PASSWORD_ALIAS_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11`	The alias of the password that is used to decrypt the client key in PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 format.
`KAFKA_SSL_CLIENT_CERT_ALIAS_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PEM`	The alias of the client certificate in PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PEM format.
`KAFKA_SSL_CLIENT_KEY_ALIAS_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PKCS11_PEM`	The alias of the client key in PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKCS11 PKC

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

CAUTION

If you are going to use ADS or ADS Control with ADPS, you should specify FQDNs when creating ADS/ADS Control hosts. The maximum FQDN length is 48 symbols.

Create a cluster

Olga Semme, Daria Barysheva

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.

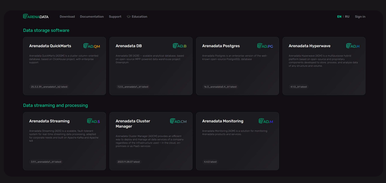
The steps for adding a cluster to ADCM are listed below.

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 Streaming***.



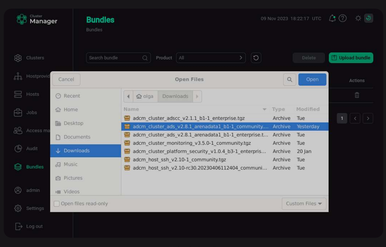
Switch to the bundle download page

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

Step 2. Upload a cluster bundle to ADCM

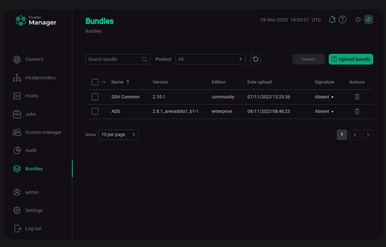
To upload a bundle to ADCM, follow the steps:

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



Upload a bundle

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



The result of successful uploading a bundle

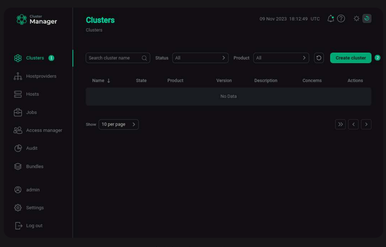
Step 3. Create a cluster based on the uploaded bundle

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.

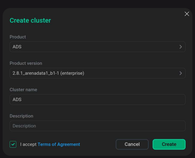
The steps for cluster creation are given below:

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



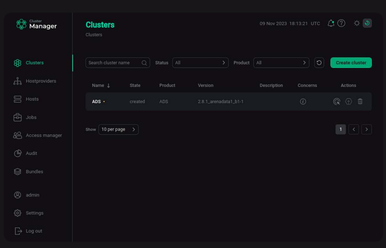
Switch to cluster creation

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 a cluster

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



The result of successful cluster creation

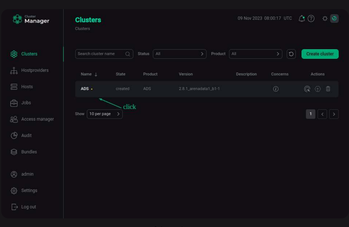
Add services

Olga Semme, Daria Barysheva

To Table of Contents

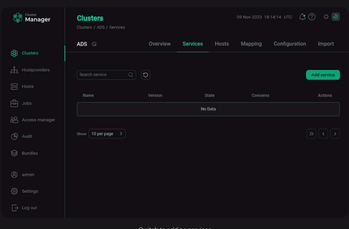
In ADCM a **service** means a software that performs some function. Examples of services for ADS clusters: NiFi, Kafka, ZooKeeper, etc. The steps for adding services to a cluster are listed below:

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



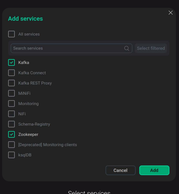
Select a cluster

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



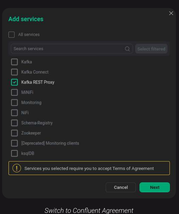
Switch to adding services

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



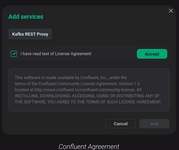
Select services

When adding ksqlDB, Schema-Registry, Kafka REST Proxy services, you should sign a Confluent license agreement. To do this, go to the agreement by clicking ***Next***.



Switch to Confluent Agreement

Next, read the text of the agreement and sign it by clicking ***Accept***.



Confluent Agreement

Click ***Add*** to add a service.

The brief description of available services is listed below.

Services that can be added to the ADS cluster

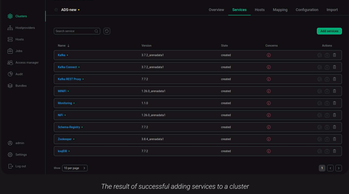
Service	Purpose
ZooKeeper	The centralized coordination service for distributed applications is used to store metadata about the sections (partitions) of topics and Apache Kafka brokers, as well as to select a broker as a Kafka controller. Informs each Kafka broker about the current state of the cluster
Kafka	An open-source distributed event streaming platform used for high-performance data pipelines, streaming analytics, data integration, and mission-critical applications
Kafka REST Proxy	Service provides a RESTful interface to an Apache Kafka cluster, making it easy to produce and consume messages, view the state of the cluster, and perform administrative actions without using the native Kafka protocol or clients
Schema-registry	The service performs the following functions: ■ Storing a version history of all Avro schemas based on a specified subject naming policy with support for several compatibility options. ■ The ability to change Avro schemas according to configured compatibility settings and extended support for these schema types. ■ Providing a RESTful interface for storing and retrieving Avro schemas. ■ Providing serializers that connect to Apache Kafka clients handling message map storage and retrieval.
ksqlDB	An open-source SQL streaming engine that provides real-time data processing using Apache Kafka
NiFi	A simple event (message) processing platform that provides real-time control of data flows from various sources using a graphical interface
NiFi2	NiFi-based service with version 2.x. The service is added in technology preview mode and is not intended for production use. Existing limitations for using NiFi2: ■ NiFi2 can only be installed as part of a separate cluster (NiFi2 + Zookeeper). ■ Before installing the service, you need to enable Java installation from the Arenadata repositories, using the ***arenadata_java*** parameter of the operating system repository to install Java 21 when setting up the cluster.
MiNiFi	A subproject of Apache NiFi — is a complementary data collection approach that supplements the core tenets of NiFi in dataflow management, focusing on the collection of data at the source of its creation
Monitoring	The service deploys the Prometheus server inside ADS to collect and store ADS cluster monitoring metrics. It also supports usage of the Grafana web application for visualization and analysis of information
Monitoring Clients	Agents sending information about hosts and services in monitoring. The service should be added after the monitoring cluster is installed and configured, if monitoring of the current cluster is planned
Kafka Connect	A tool for scalably and reliably streaming data between Kafka and other data systems

The minimal set of services recommended for ADS clusters is described below:

- NiFi
- Kafka
- ZooKeeper

These services make up the core of Streaming and are sufficient for fast and easy installation and management of Arenadata Streaming with Arenadata Cluster Manager. The full list of services depends on the requirements of a particular project. Not all services are required to be installed. For example, if you do not plan to use NiFi, then there is no need to add that service. Also, if a monitoring service is used (not based on Graphite), there is no need to install agents from Monitoring Clients. However, if you plan to use Kafka, the Kafka and ZooKeeper services are required. The same can be said about the NiFi service. In this case, the service can consist of mandatory and optional components. For example, a ksqlDB service consists of a required Server component and an optional Client component.

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



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.

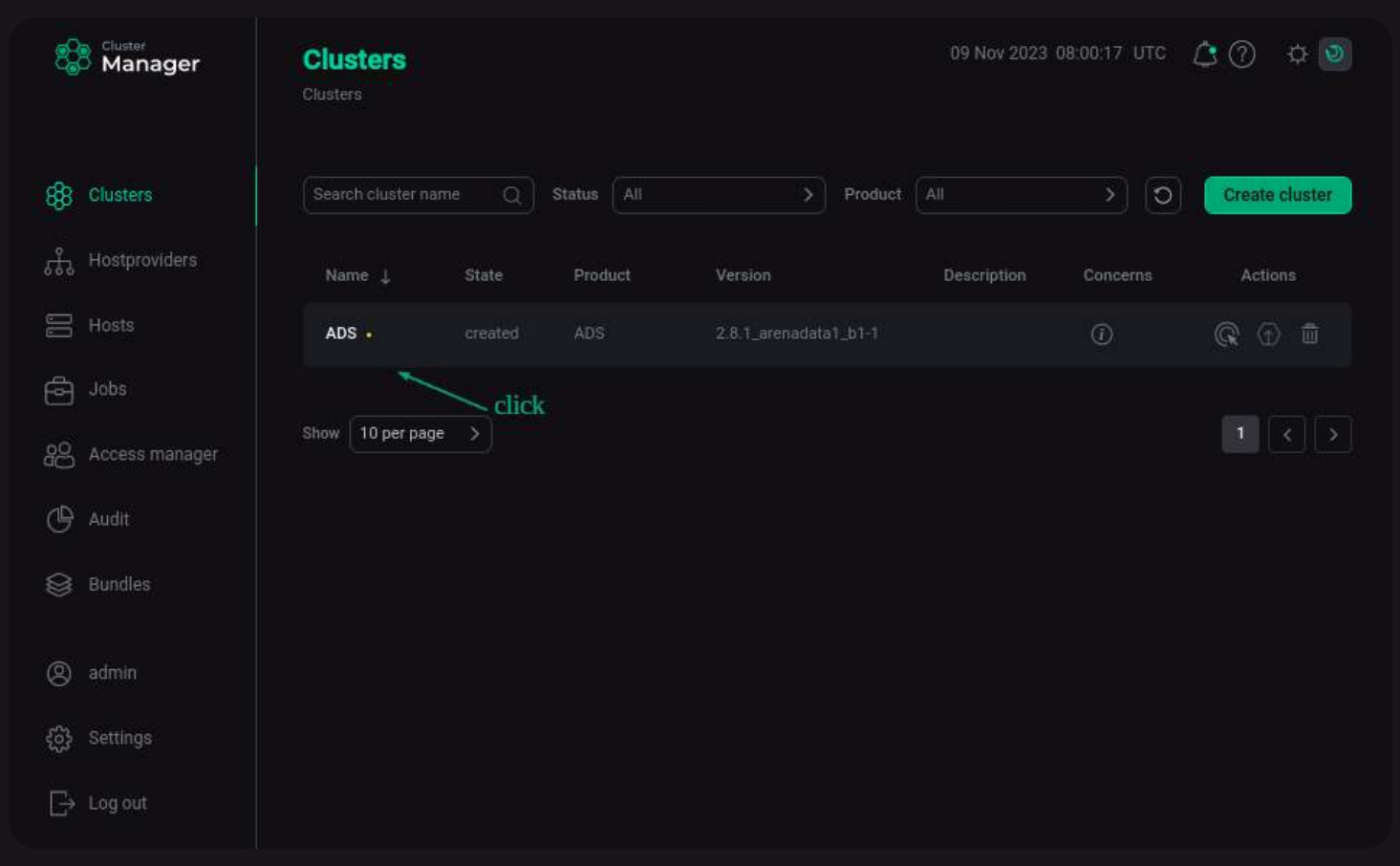
Add hosts to a cluster

Olga Semme, Daria Barysheva

To Table of Contents

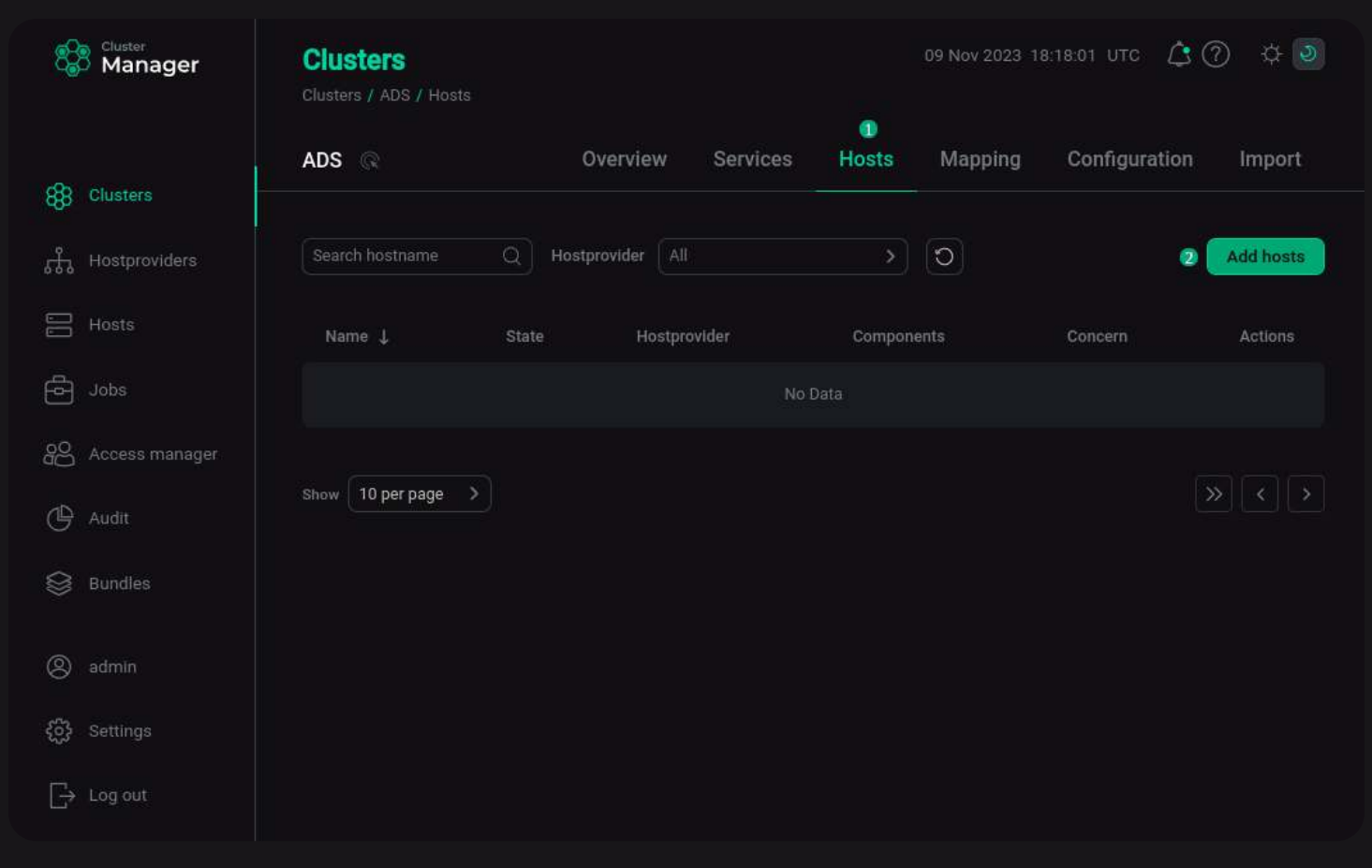
The steps for adding hosts to a cluster are listed below:

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



Select a cluster

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

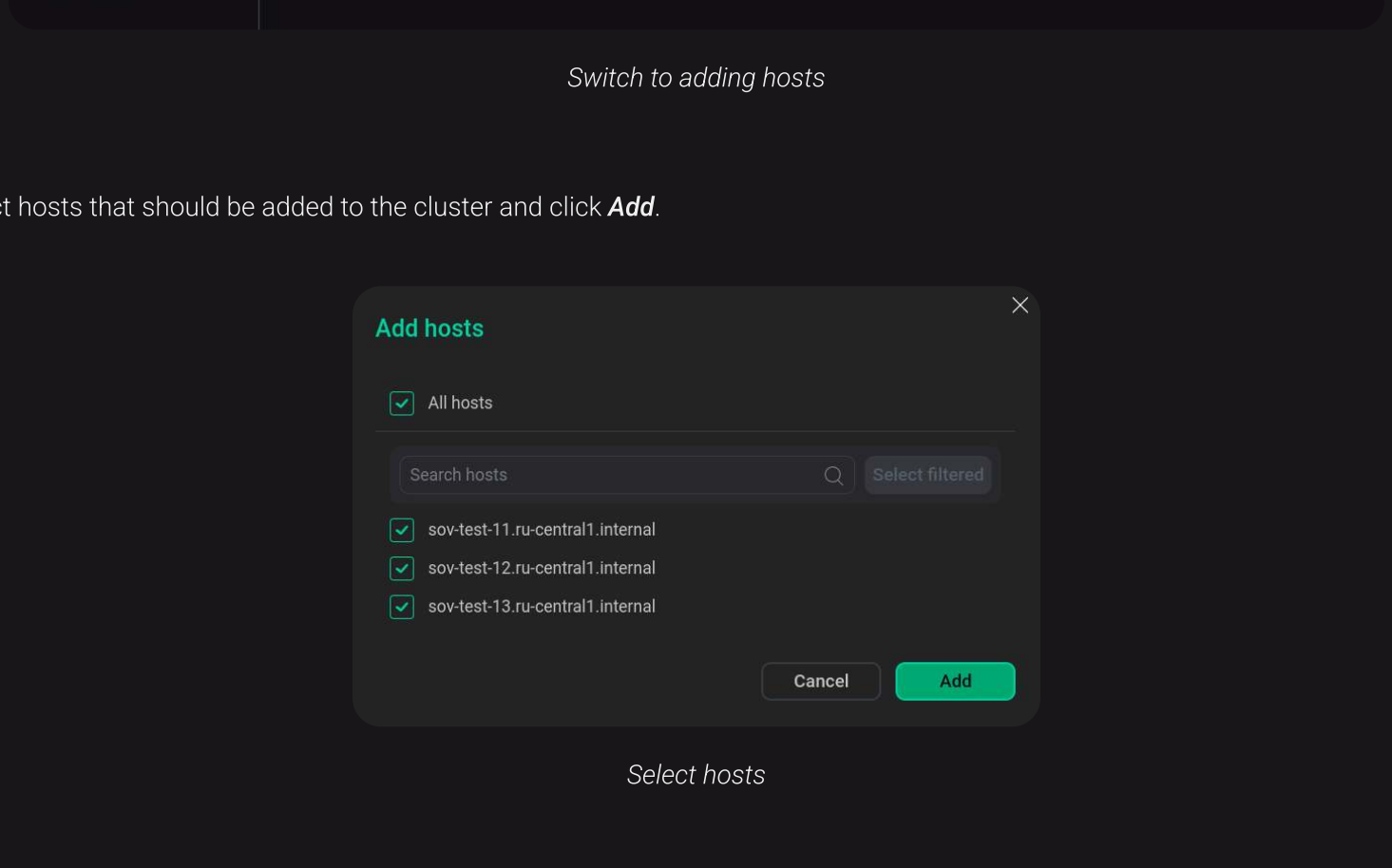


Switch to adding hosts

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

Select hosts

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



The result of successful adding hosts to a cluster

Add components

Olga Semreva, Daria Barysheva

To Table of Contents

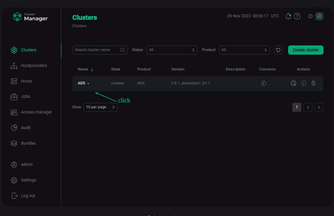
In ADCM a **component** means a part of a service that should be deployed on one or several cluster hosts. For example, NiFi service can consist of such components as NiFi Server, NiFi Registry.

IMPORTANT

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

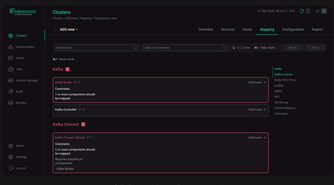
The steps for adding components to hosts are listed below:

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



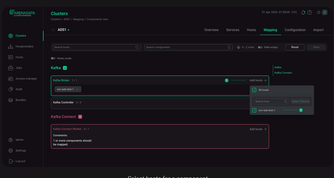
Select a cluster

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



Switch to mapping service components

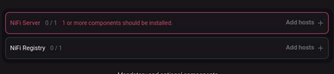
3. In the window that opens, map components to hosts. To assign hosts for each component, click ***Add hosts*** and select hosts in the pop-up window.



Select hosts for a component

Mandatory and optional 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.



Mandatory and optional components

CAUTION

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

The examples of distribution components in ADS clusters consisting of 3 and 5 hosts are described in the table below.

Examples of components distribution

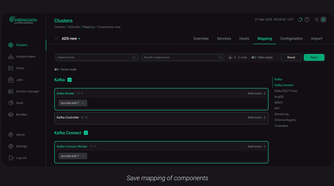
3 hosts

Host	Service	Component
Host 1	Zookeeper	Zookeeper server
	Kafka	Kafka Broker
	Kafka REST Proxy	Kafka REST Proxy
	Schema-Registry	Schema-Registry
Host 2	Zookeeper	Zookeeper Server
	ksqlDB	ksqlDB Client ksqlDB Server
	Kafka	Kafka Broker
	NiFi	NiFi Server
Host 3	Zookeeper	Zookeeper Server
	Kafka	Kafka Broker
	Kafka Connect	Kafka Connect Worker
	NiFi	NiFi Server NiFi Registry
	MiNiFi	MiNiFi C2 Server MiNiFi Agent

5 hosts

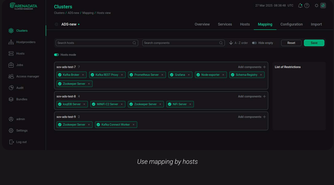
Host	Service	Component
Host 1	Zookeeper	Zookeeper Server
	Kafka	Kafka Broker
	Schema-Registry	Schema-Registry
Host 2	Zookeeper	Zookeeper Server
	Kafka	Kafka Broker
Host 3	Zookeeper	Zookeeper Server
	ksqlDB	ksqlDB Client ksqlDB Server
	Kafka REST Proxy	Kafka REST Proxy
Host 4	Kafka	Kafka Broker
	NiFi	NiFi Server NiFi Registry
Host 5	NiFi	NiFi Server
	MiNiFi	MiNiFi C2 Server MiNiFi Agent
	Kafka Connect	Kafka Connect Worker

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

Last updated: 26th November

ADS

Contents

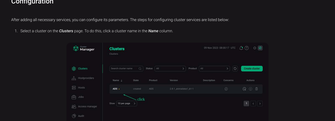
To Table of Contents

Configuration
Components
Configuration groups

Configuration

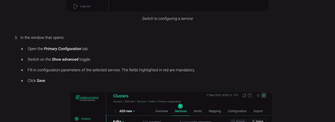
After adding all necessary services, you can configure its parameters. The steps for configuring cluster services are listed below:

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



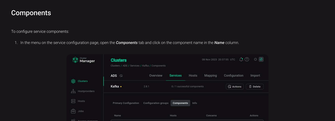
Select a cluster

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



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**.



Configure a service

NOTE

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

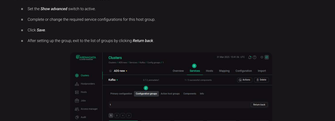
Components

To configure service components:

1. In the menu on the service configuration page, open the **Components** tab and click on the component name in the **Name** column.

Switch to setting up the service component

2. In the window that opens:
   - Set the **Show advanced** switch to active.
   - Fill in or change the configuration parameters of the selected component, if any. As a rule, these are the settings of the Log4j logging library. Templates for log4j_properties files are also available here.
   - Click **Save**.

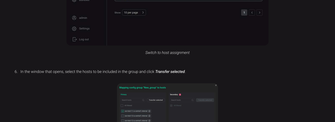


Configure a service component

Configuration groups

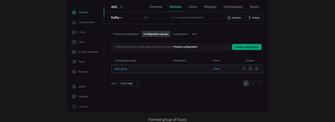
To configure one or a group of hosts:

1. On the service configuration page, open the **Configuration groups** tab.
2. On the page that opens, click **Create config group**.



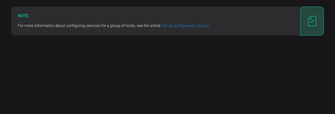
Switch to a host group creation

3. In the window that opens, enter the name of the group and click **Next**.



Create a host group

4. After creating a host group in the window that opens:
   - Set the **Show advanced** switch to active.
   - Complete or change the required service configurations for this host group.
   - Click **Save**.
   - After setting up the group, exit to the list of groups by clicking **Return back**.



Configure a host group

5. To assign hosts to a group, in the line corresponding to the created group, click on the ⚙ icon.

Switch to host assignment

6. In the window that opens, select the hosts to be included in the group and click *Transfer selected*.

Add hosts to a group

7. After completing the formation of the host group, click **Save**.

Save a group of hosts

As a result, the number of hosts in the line corresponding to the created group changes.

Formed group of hosts

NOTE

For more information about configuring services for a group of hosts, see the article Set up configuration groups.

Configure a cluster

Olga Semme, Daria Barysheva

Contents

To Table of Contents

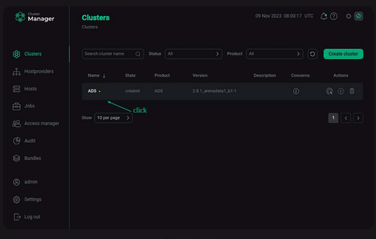
Primary configuration
Configuration groups
Ansible settings

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.



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.

Cluster configuration parameters are listed below:

- ***Reliability control*** — parameters of this section are intended to set the timeout of the DNF software package manager unlocking wait. Displayed if the ***Show advanced*** switch is activated. In Ubuntu, after the host reboot, automatic update starts, which locks DNF.

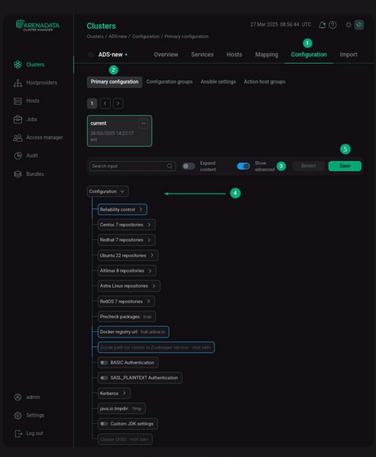
Name	Description	Default value
retries	The maximum number of retries	80
delay	The time delay between retries (in seconds)	30
timeout	Timeout for network ports checks (in seconds)	7200

- Paths to the repositories that are used during the installation process depending on the cluster operating system (CentOS, Red Hat, Ubuntu, ALT Linux 8, or Astra Linux). Only the marked repositories are registered on the cluster servers.

Repositories

Repository name	Parameter	Description	Default value
monitoring	desc	Repository description	Monitoring components url
	url	Arenadata repository path	Specified depending on OS
	enabled	Enables use of the repository	true
ADS	desc	Repository description	ADS repository
	url	Arenadata repository path	Specified depending on OS
	enabled	Enables use of the repository	true
prometheus_repo	desc	Repository description	Prometheus repository
	url	Arenadata repository path	Specified depending on OS
	enabled	Enables use of the repository	true
arenadata_java	desc	Repository description	Arenadata repository for OpenJDK
	url	Arenadata repository path	Specified depending on OS
	enabled	Enables use of the repository. If set to `true` before the ***Install***, ***Add/Remove components***, or ***Update*** action for a cluster or service, the latest version of Java 17 will be installed on hosts	false
ranger plugins	desc	Repository description	Arenadata Ranger plugins repository
	url	Arenadata repository path	Specified depending on OS
	enabled	Enables use of the repository	true

- ***Precheck packages*** — enables package availability check before installation.
- ***Docker registry url*** — specifies the location of Arenadata Docker Registry. Displayed if the ***Show advanced*** switch is activated.
- ***Znode path for cluster in Zookeeper service*** — Znode path for the ADS cluster in the Zookeeper service. Displayed if the ***Show advanced*** switch is activated.
- ***BASIC Authentication*** — configuration for BASIC authentication.
  For a description of enabling authentication, see the article Basic authentication.
- ***SASL_PLAINTEXT Authentication*** — configuration for SASL PLAINTEXT authentication.
  For a description of enabling authentication, see the article SASL PLAINTEXT.
- ***Kerberos*** — configuration for Kerberos authentication.
  The description of enabling authentication is given in the articles:
  - MIT Kerberos
  - Kerberos with MS Active Directory
  - Kerberos with Samba
- ***java.io.tmpdir*** — a Java system property that determines where the JVM writes temporary files.
- ***Custom JDK settings*** — a switch that indicates whether to use a custom Java path for ADS services. When the switch is activated, the ***JAVA_HOME*** field becomes available. Enter a work Java path in that field. The selected path should contain a *bin* folder with a Java binary file, for example */usr/lib/jvm/jre-openjdk/*. If the ***JAVA_HOME*** value is empty, the default system path is used */usr/lib/jvm/java-1.17.0-openjdk.amd64/* (for all operating systems except Centos, RedHat, and AstraLinux).
- ***Cluster UUID*** — displays the universally unique identifier of the cluster.



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

Olga Semina, Daria Borysheva

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.

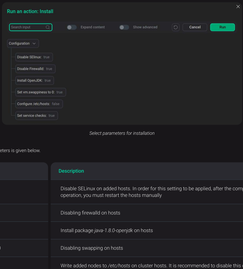
IMPORTANT

The installation must be performed on a clean host with no previous installations of related or similar software.

Whole cluster

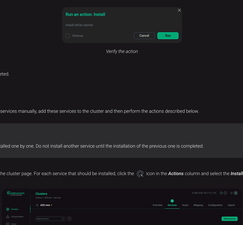
To install all cluster services within a single action, follow the steps:

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



Switch to the whole cluster installation

2. In the dialog box that opens, select the required options for installation.

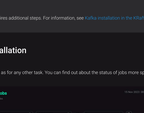


Select parameters for installation

The purpose of the parameters is given below.

Parameter	Description
Disable SELinux	Disable SELinux on added hosts. In order for this setting to be applied, after the completion of the **Install** operation, you must restart the hosts manually
Disable Firewalld	Disabling firewalld on hosts
Install OpenJDK	Install package *java-1.8.0-openjdk* on hosts
Set vm.swappiness to 0	Disabling swapping on hosts
Configure /etc/hosts	Write added nodes to */etc/hosts* on cluster hosts. It is recommended to disable this option if DNS is configured
Set service checks	Enables service checking when starting an action

3. Verify the action in the opened window.



Verify the action

4. Wait until the job is completed.

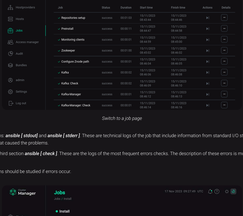
Single services

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.



Switch to installation of single services

2. In the dialog box that opens, select the required parameters for installation and click **Run**.
3. Verify the action in the opened window.

Verify the action

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

IMPORTANT

- When you install services manually, one by one, use the following order:
  - ZooKeeper;
  - Kafka manager, Kafka, NiFi (one by one);
  - other services.
- Installing a Kafka service in the KRaft mode requires additional steps. For information, see Kafka installation in the KRaft mode.

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.

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.

Switch to a job page

A job page contains two sections: ***ansible [stdout]*** and ***ansible [stderr]***. These are technical logs of the job that include information from standard I/O streams *stdout*/*stderr*. These logs can help to understand what caused the problems.

There can also be the optional third section **ansible [check]**. These are the logs of the most frequent errors checks. The description of these errors is more simple and specific, than in two previous technical logs.

The contents of all three sections should be studied if errors occur.

Job details

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`.

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`.

The state of cluster services after successful installation

Install monitoring

Olga Semme

To Table of Contents

Monitoring is an optional but strongly recommended ADS component that you can configure in one of the following ways:

- add the Monitoring service to the ADS cluster (available starting with 3.6.2.2.b1);
- create a monitoring cluster and integrate it with the ADS cluster (deprecated).

Monitoring service

Olga Ivanova

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 deploys the Prometheus server inside ADS to collect and store ADS cluster monitoring metrics. It also supports usage of the Grafana web application for visualization and analysis of information. The whole described steps required to add the service to an ADS cluster.

Overview

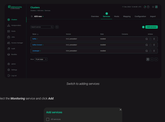
When you add the Monitoring service to an ADS cluster, Node Exporter is installed on all hosts. It is a monitoring agent that reads host system metrics collected by Prometheus. Prometheus also collects metrics from Kafka brokers, ZooKeeper nodes, Kafka MirrorMaker, Kafka Bridge, Kafka Exporter, and Cruise Control. These metrics are available in the Prometheus format. Some components specified in additional configuration settings. 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 ADS metrics. To do so, configure its ADS monitoring targets as described in the Metrics in the Prometheus format section. See then on the Viewing metrics page.
- You can also use the Federation mechanism to migrate all metrics from the Prometheus server deployed in ADS to your Prometheus.

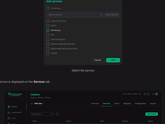
Step 1. Add the service

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



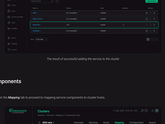
Switch to adding services

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



Select the service

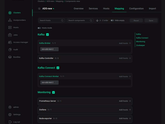
As a result, the added service is displayed on the **Services** list.



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.



Switch to mapping service components

Monitoring service components

Component	Description
Prometheus Server	Deploys a Prometheus server that serves as a: • proxy interface for all collections of metrics as a host, • storage for all metrics of a cluster, • generator of alerts based on collected metrics.
Grafana	Visualizes ADS metrics as graphs and charts organized into dashboards
Node exporter	Exports ADS host metrics to Prometheus

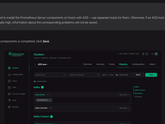
2. Assign a host to each component of the Monitoring service – click **Add hosts** and select the desired host in the pop-up window.

Select a host for a component

CAUTION

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

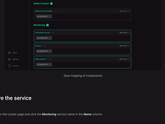
3. After the distribution of components is completed, click **Save**.



Save mapping of components

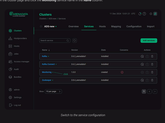
Step 3. Configure the service

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



Switch to the service configuration

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

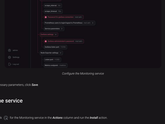


Configure the Monitoring service

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

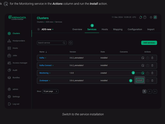
Step 4. Install the service

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



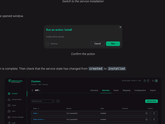
Switch to the service installation

2. Confirm the action in the opened window.



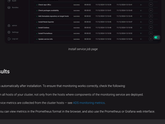
Confirm the action

3. Wait until the installation is complete. Then check that the service state has changed from **installed** to **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.



Install service job page

Step 5. View results

The Monitoring service starts automatically after installation. To ensure that monitoring works correctly, check the following:

- Metrics are collected from all hosts of your cluster, not only from the hosts where components of the monitoring service are deployed.
- Both systems and ADS service metrics are collected from the cluster hosts – see ADS monitoring metrics.

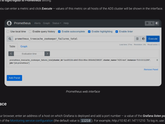
To check both these points, you can view metrics in the Prometheus format in the browser, and also use the Prometheus or Grafana web interface.

CAUTION

To enable Kafka monitoring, on the Kafka service Configuration page, set the **JMX Monitoring** switch to the active position, then restart the service.

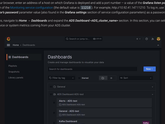
Metrics in the Prometheus format

1. In the address bar of the browser, enter the ADS cluster host address, and also specify the port and endpoint of access to the system metrics, which are defined on the **Services** > **Monitoring** > **Configuration** page in the **Prometheus settings** group of the Monitoring service configuration. For example:
An imperfect address for viewing ADS system metrics: **http://10.92.1.67:9101**. Here **10.92.1.67** – FQDN of the ADS host.
2. The page that opens will display monitoring metrics from the specified host of the ADS cluster in the Prometheus format.



Prometheus web interface

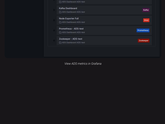
1. In the address bar of your browser, enter the IP address of the host where the Prometheus Server component of the Monitoring service is installed. Specify the port number, which is determined by the **web.listen-address** parameter in the **Prometheus settings** group of the Monitoring service configuration. The default port is **9090**. For example: **http://10.92.1.67:9090**. Here **10.92.1.67** – FQDN of the Prometheus Server host, and the parameter value that we also specified in the **Prometheus settings** section.
2. Click **Prometheus** (see the right figure in Prometheus Web).
3. In the **Expression** field, you can enter a metric and click **Execute** – values of this metric on all hosts of the ADS cluster will be shown in the interface.



Prometheus web interface

Grafana web interface

1. Enter the IP address of your previous server address of a server which Grafana is deployed and port number 3000. For example: **http://10.92.1.67:3000**. Here **10.92.1.67** – FQDN of the Grafana server, and **3000** – the default port number. If you changed the port in the **Grafana settings** section of service configuration parameters (in the **Grafana administrator's password** parameter value input field) or in the **Grafana settings** section of service configuration parameters, be sure to specify it.
2. In the window that opens, navigate to **Home** > **Dashboards** and expand the **ADS Dashboard - ADS_cluster_name** section. In this section, you can select one of available dashboards to view system metrics coming from your ADS cluster.



View ADS metrics in Grafana

Install monitoring

Daria Barysheva

To Table of Contents

Monitoring is installed optionally (yet strongly recommended). ADCM monitoring is based on three services: Diamond, Graphite, and Grafana.

For **online** installation, you need a separate monitoring cluster. The steps for installing monitoring in this case are listed below. You can perform them both before and after the ADS cluster installation, except the last step Integrate with ADS cluster, that can be executed only when the ADS cluster already exists.

- Create a cluster
- Add services
- Add hosts to a cluster
- Add components
- Configure services
- Install a cluster
- Integrate with ADS cluster

CAUTION

Starting with ADS 3.6.2.2.b1, it is not recommended to configure ADS monitoring using a monitoring cluster, as support for this functionality is currently suspended and will be completely discontinued in future ADS releases. The recommended way to monitor an ADS cluster is to use the Monitoring service.

Create a cluster

Daria Barysheva

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

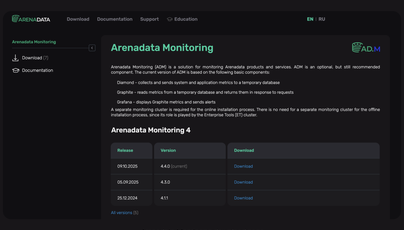
In ADCM a **cluster** means a set of services running on a set of hosts.

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.

Step 1. Download a cluster bundle

1. Go to https://network.arenadata.io/ and select ***Arenadata Monitoring***.
2. Download a bundle of the desired version.

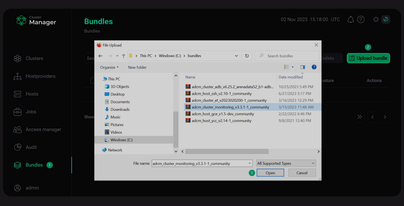


Arenadata Monitoring download page

Step 2. Upload a cluster bundle to ADCM

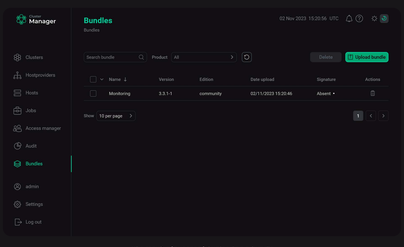
To upload a bundle to ADCM, follow the steps:

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



Upload a bundle

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



The result of successful uploading a bundle

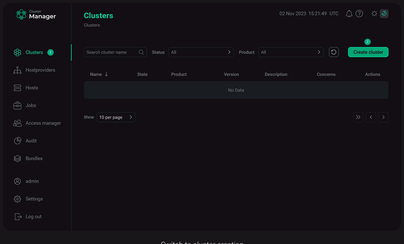
Step 3. Create a cluster based on the uploaded bundle

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.

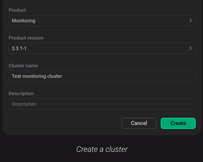
The steps for cluster creation are given below:

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



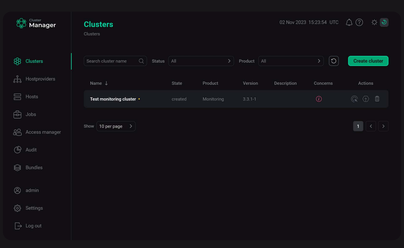
Switch to cluster creation

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 a cluster

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



The result of successful cluster creation

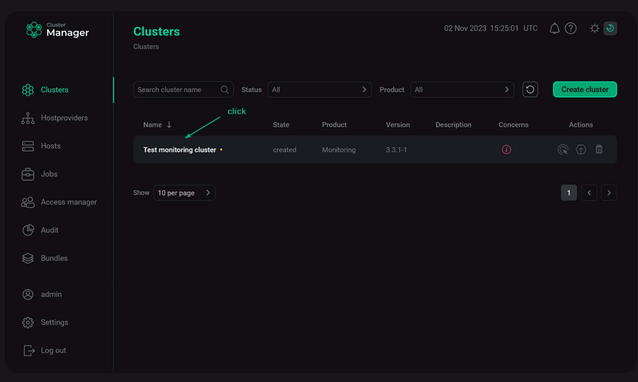
Add services

Daria Barysheva

To Table of Contents

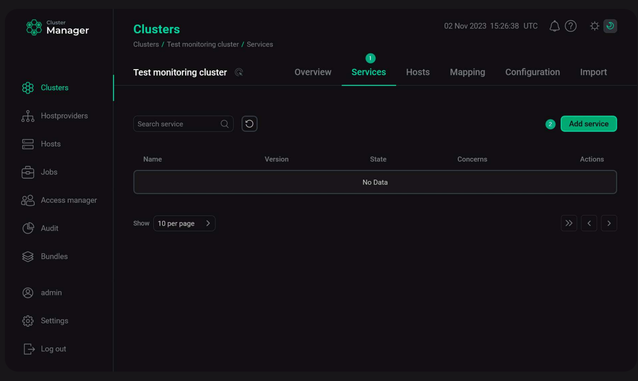
In ADCM a **service** means a software that performs some function. The steps for adding services to a cluster are listed below:

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



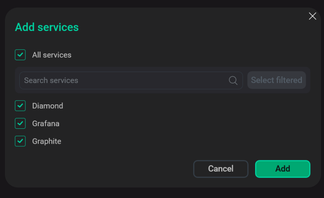
Select a cluster

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



Switch to adding services

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



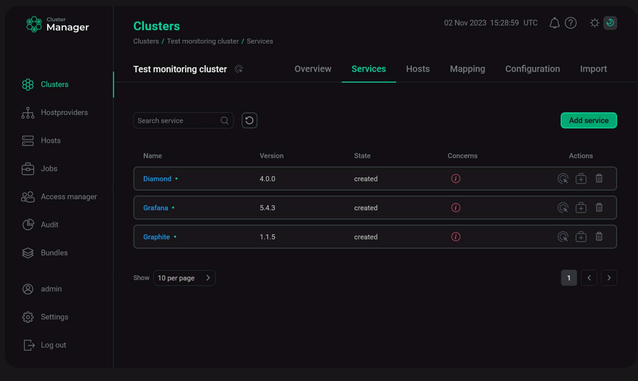
Select services

The brief description of available services is listed below.

Services that should be added to the monitoring cluster

Service	Purpose
Diamond	A Python service used for collecting system metrics from the hosts that are connected to the monitoring system. The collected metrics are processed by Carbon (the backend services of Graphite)
Graphite	Includes several components: ■ **Carbon** — a set of services that are used to process metrics collected by Diamond. ■ **Whisper DB** — a temporary file database where collected metrics are stored. ■ **Web service** — allows to get and visualize metrics from Whisper DB.
Grafana	A web service that allows users to create convenient sets of graphs, alerts, and other visualization tools for metrics received from Graphite. By default, uses the port `3000`

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



The result of successful adding services to a cluster

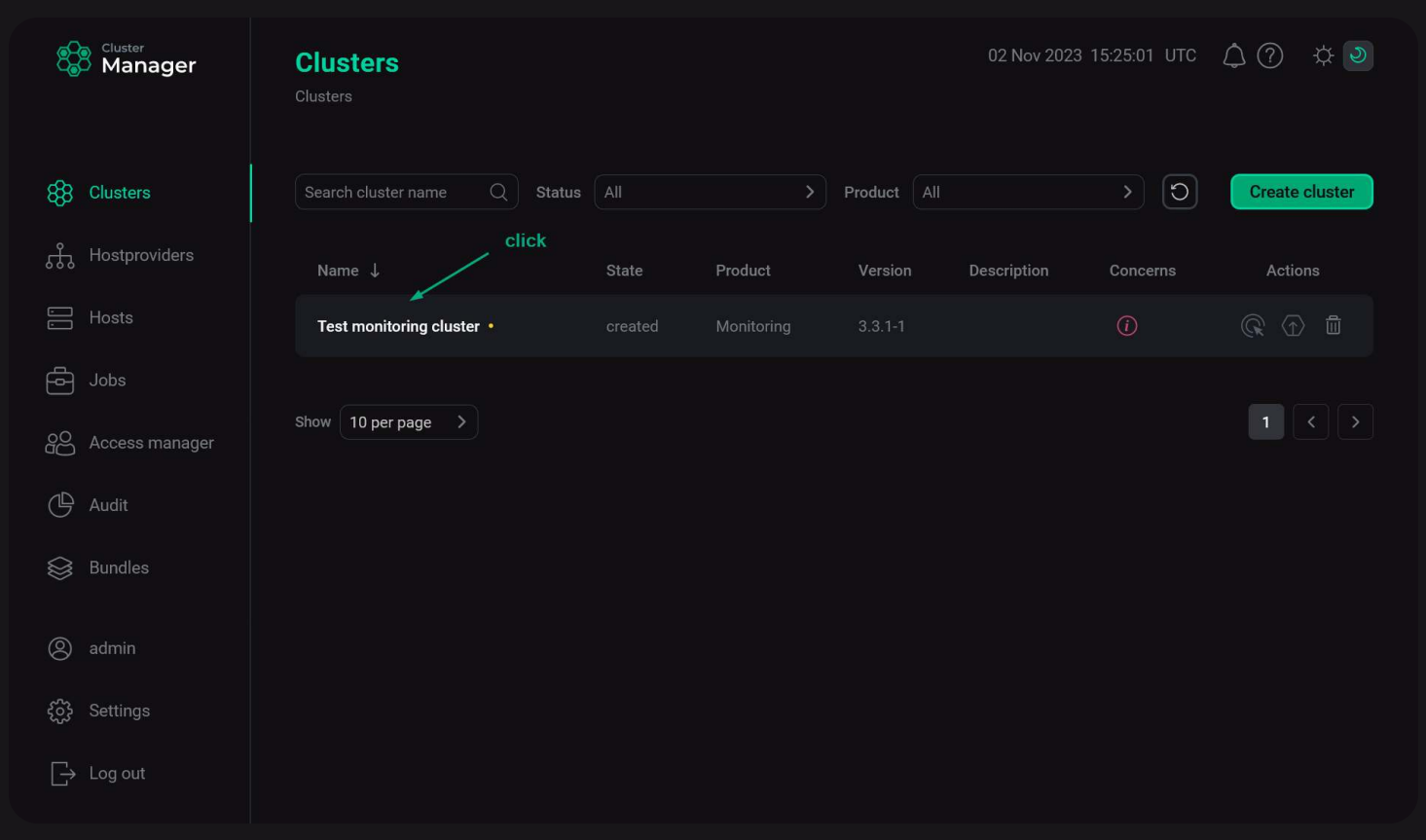
Add hosts to a cluster

Daria Barysheva

To Table of Contents

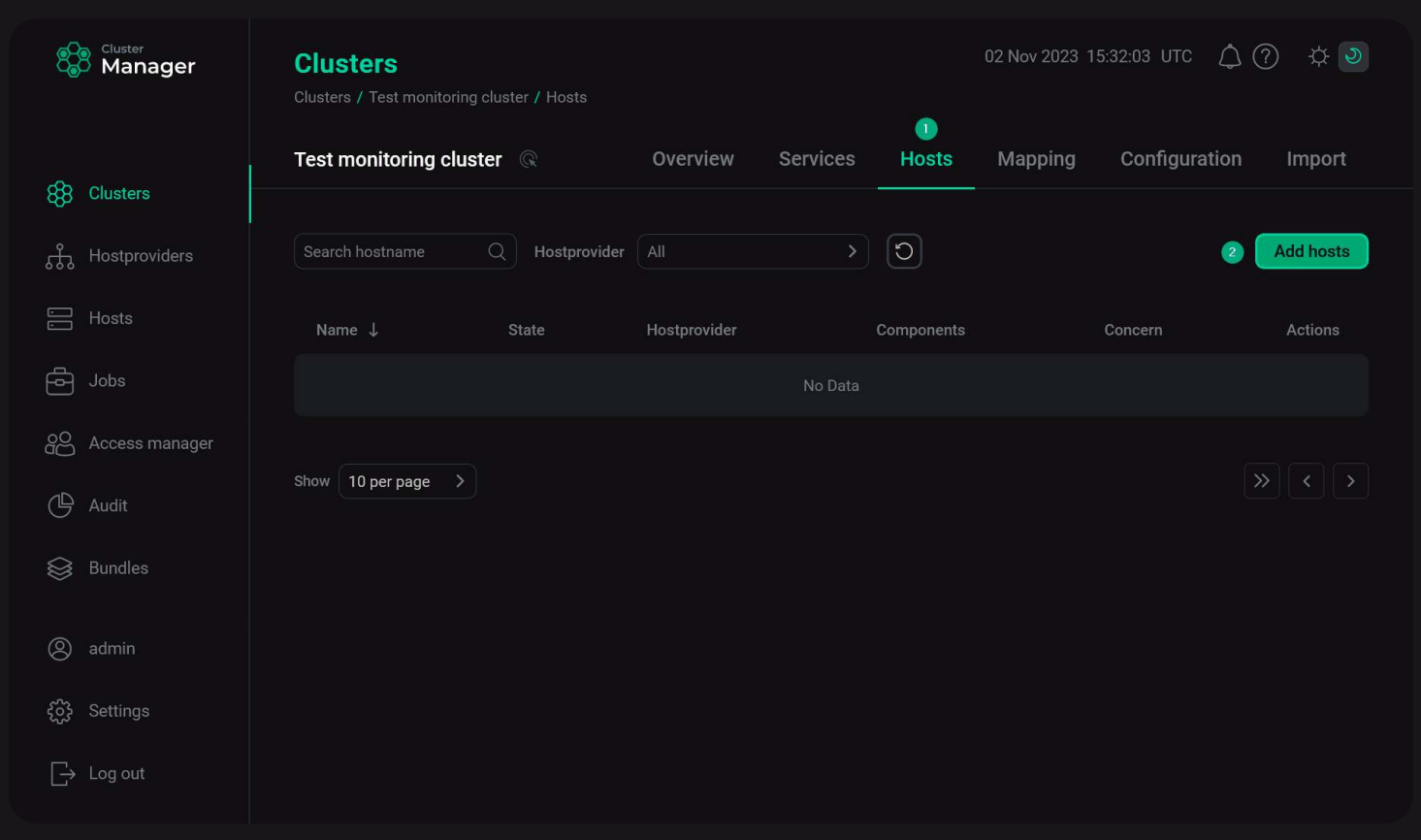
The steps for adding hosts to a cluster are listed below:

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



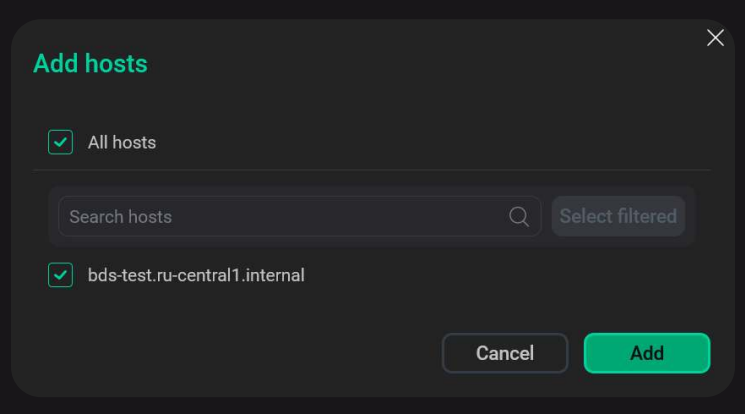
Select a cluster

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



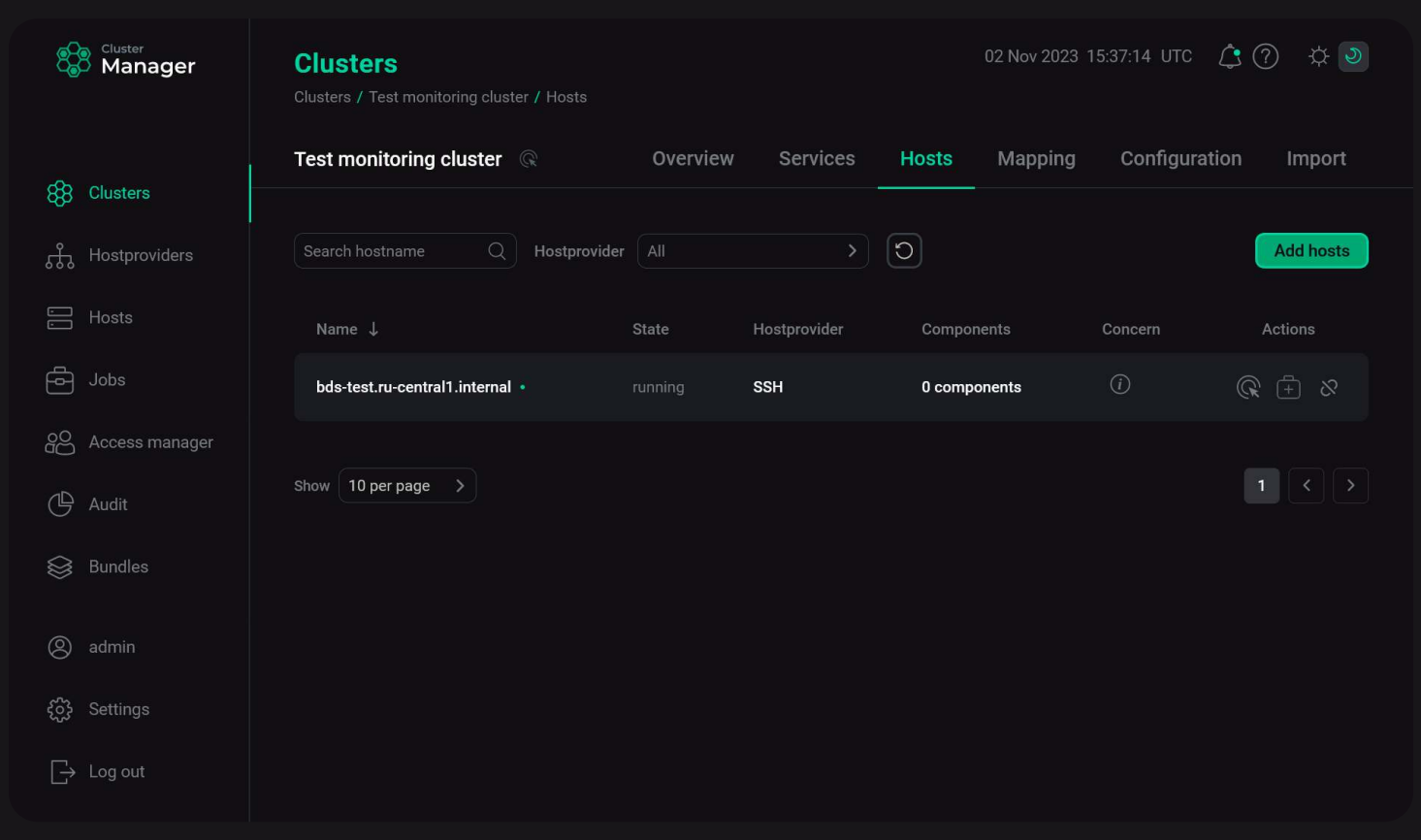
Switch to adding hosts

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



Select hosts

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



The result of successful adding hosts to a cluster

Add components

Daria Barysheva

To Table of Contents

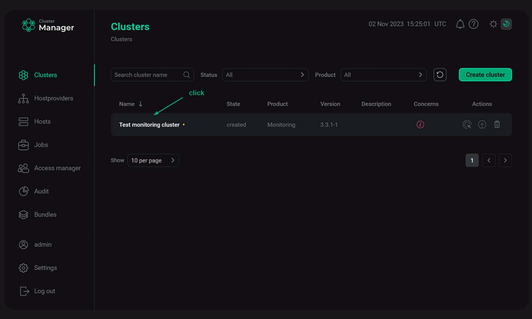
In ADCM a **component** means a part of a service that should be deployed on one or several cluster hosts.

IMPORTANT

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

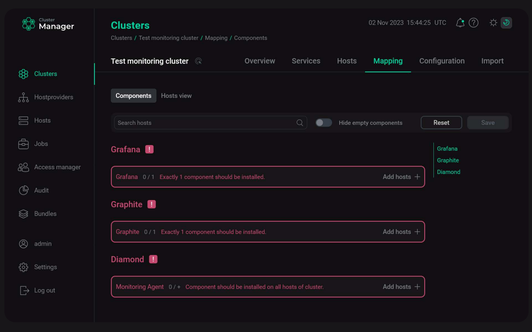
The steps for adding components to hosts are listed below:

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



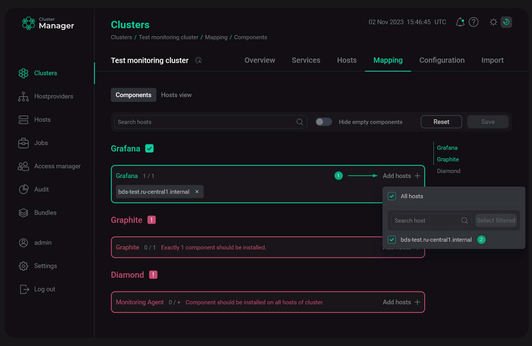
Select a cluster

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



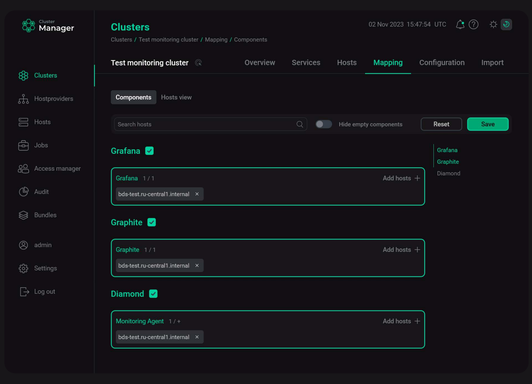
Switch to mapping service components

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.



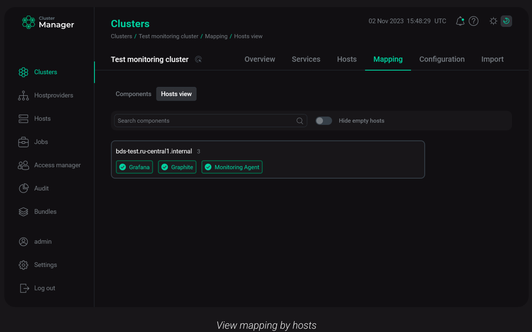
Select hosts for a component

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



Save mapping of components

5. To open an alternative mapping view (by hosts), select the ***Hosts view*** tab.



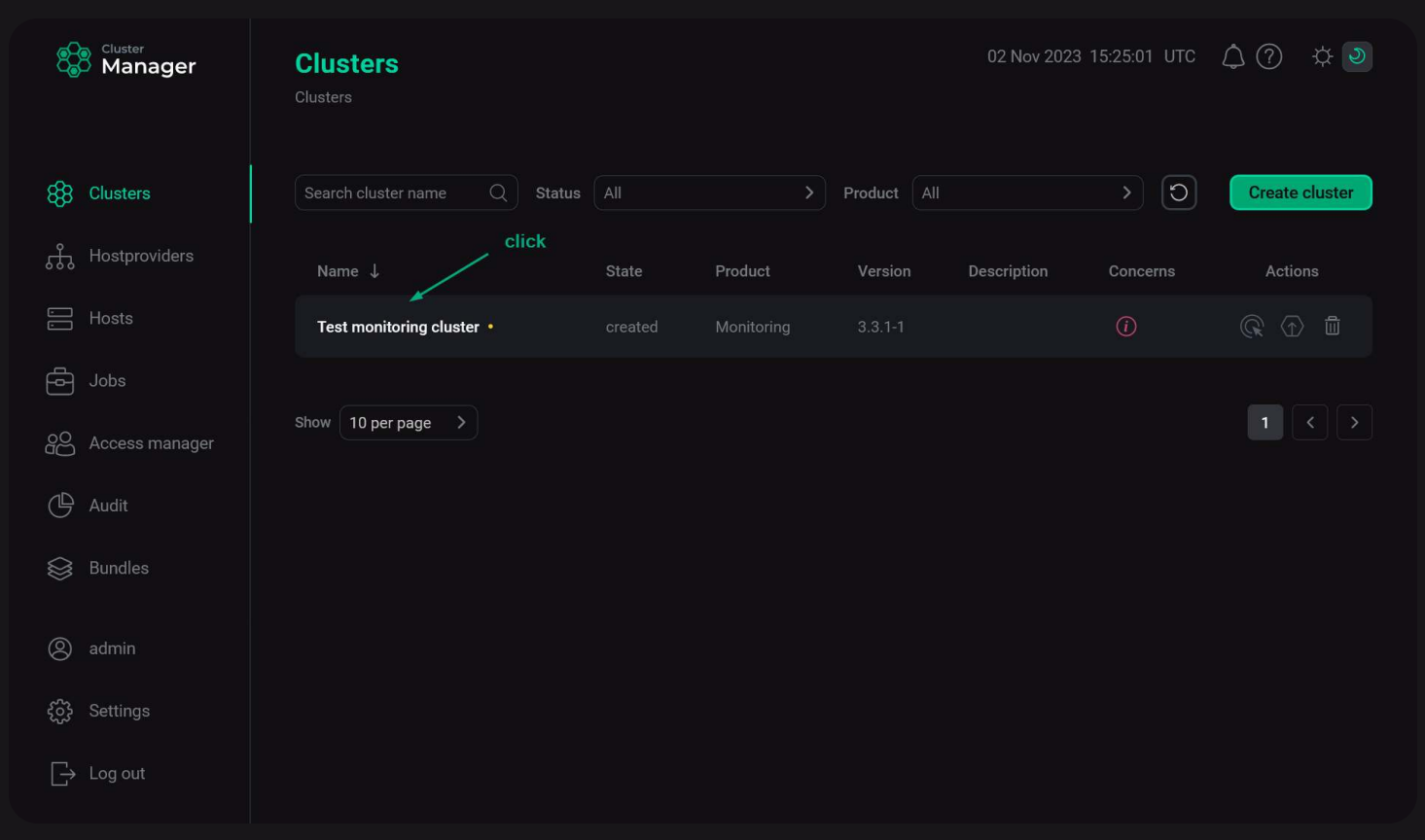
View mapping by hosts

Configure services

Daria Barysheva

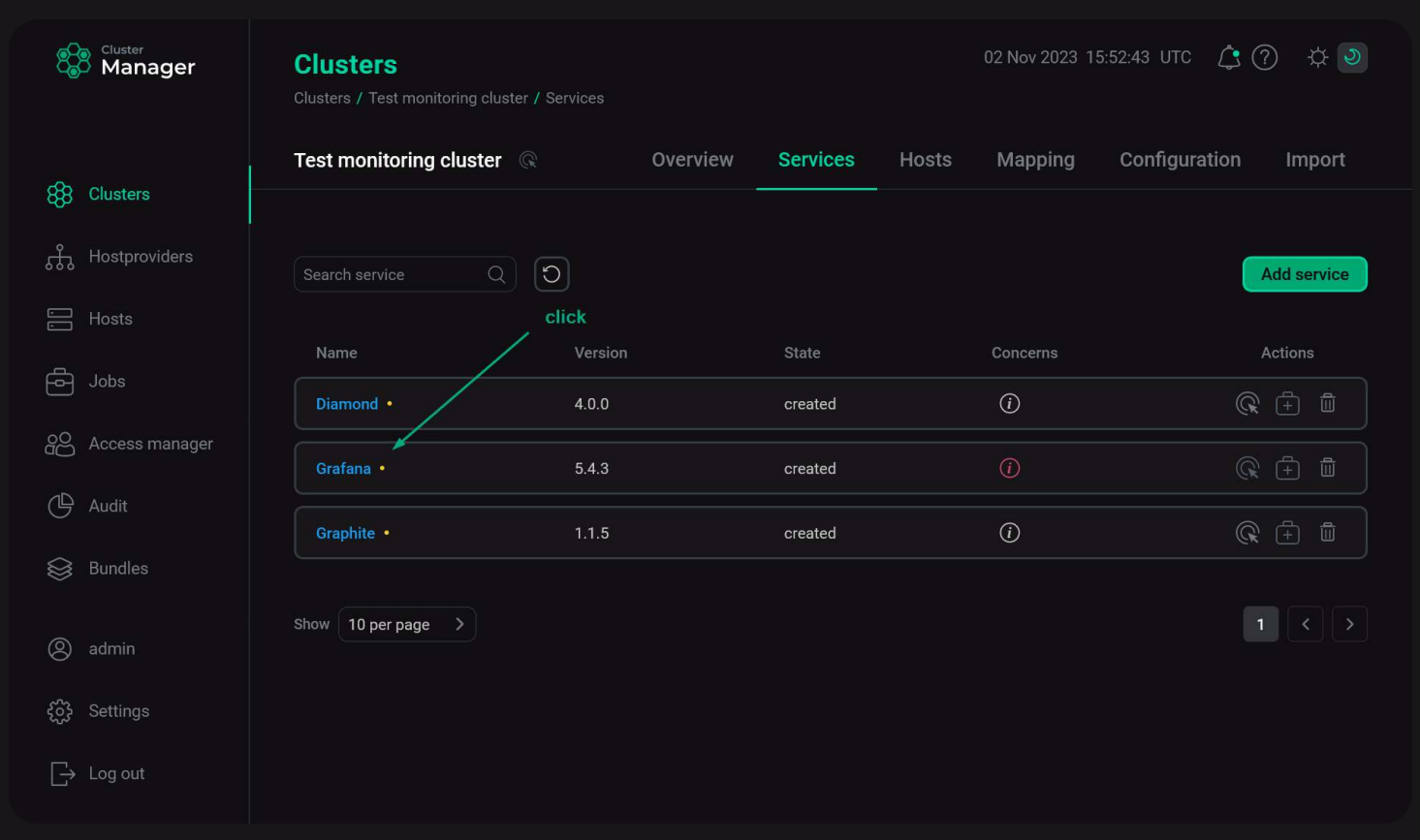
To Table of Contents

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



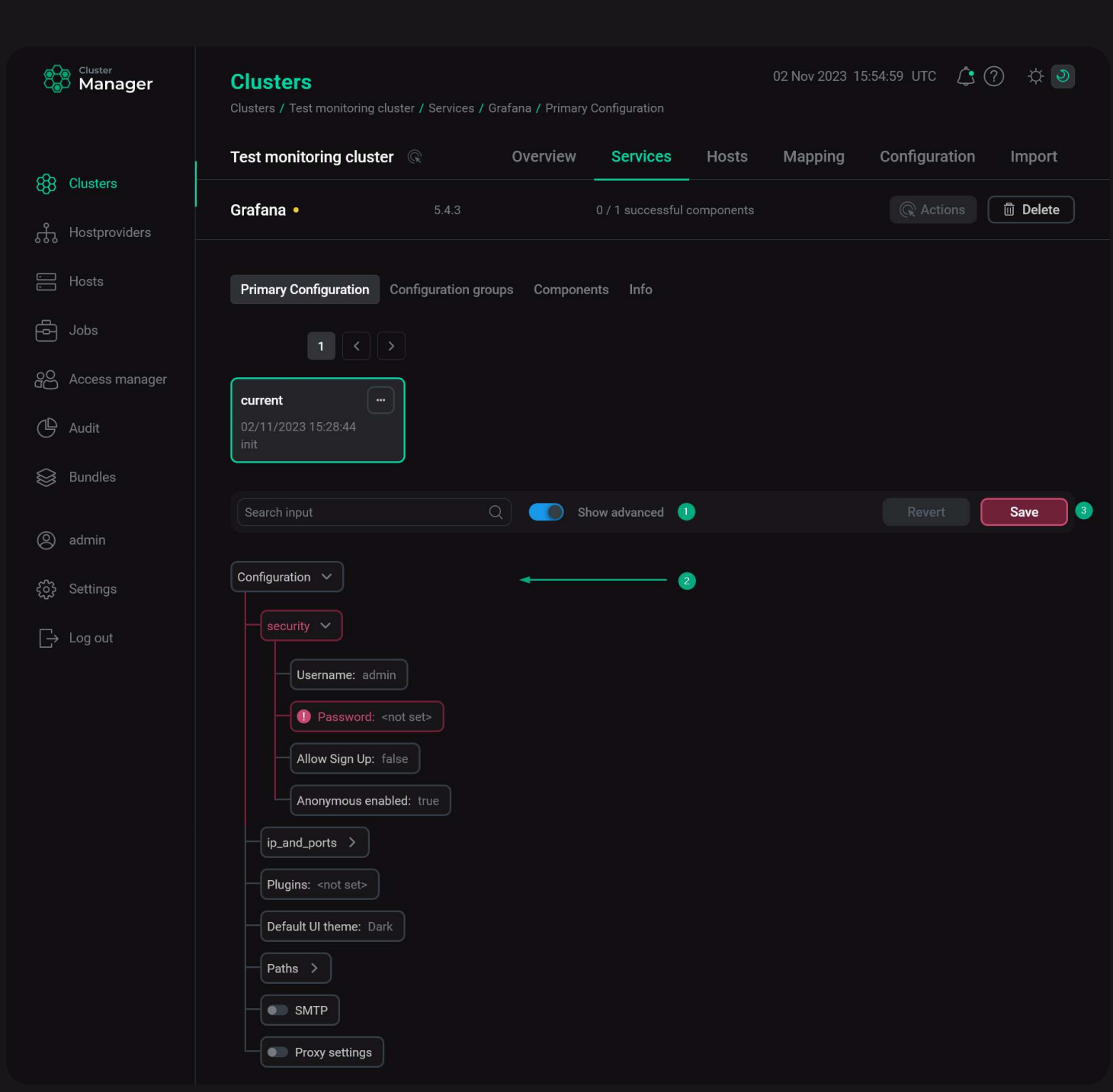
Select a cluster

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



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***.



Configure a service

Install a cluster

Daria Barysheva

Contents

To Table of Contents

Step 1. Run installation

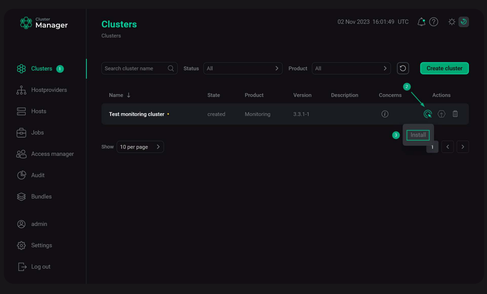
Step 2. View results

- Graphite
- Grafana

Step 1. Run installation

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

1. Apply the ***Install*** action to the cluster selected on the ***Clusters*** page by clicking the icon in the ***Actions*** column. Verify the requested action in the opened window. All services are installed consecutively.



Switch to monitoring cluster installation

2. Wait until the job is completed.

NOTE

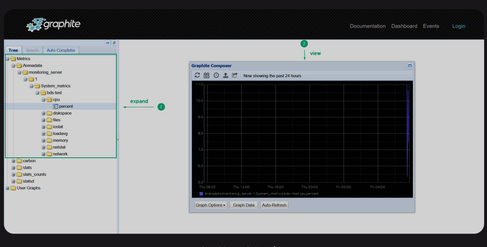
In case of errors, you can find logs on the ***Jobs*** page.

Step 2. View results

Graphite

To check availability of the Graphite web-interface, you can do:

1. In the browser bar, enter an address of the host where the monitoring cluster is deployed. To specify the port number, use the value of the ***Web-interface TCP port*** parameter from the ***ip_and_ports*** section defined when configuring the Graphite service during the monitoring cluster installation. The value `80` (used by default) can be omitted.
2. Expand a tree at the left side of the opened form. System metrics are displayed there, grouped by hosts. You can select some of these metrics to view their change dynamics in the form of a graph at the right side of the screen.

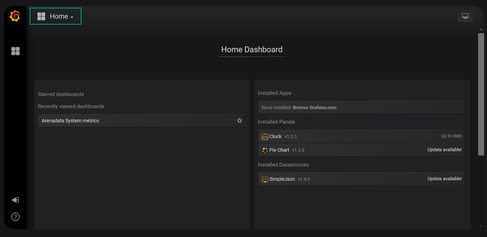


Graphite web-interface

Grafana

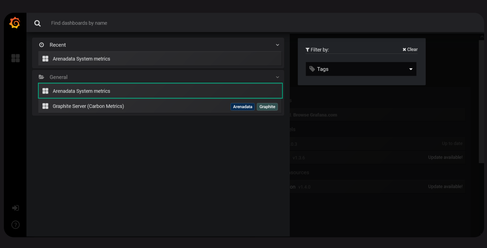
To check availability of the Grafana web-interface, you can do:

1. In the browser bar, enter an address of the host where the monitoring cluster is deployed. To specify the port number, use the value of the ***Port*** parameter from the ***ip_and_ports*** section defined when configuring the Grafana service during the monitoring cluster installation (by default, `3000`).
2. Select the ***Home*** menu item in the opened form.



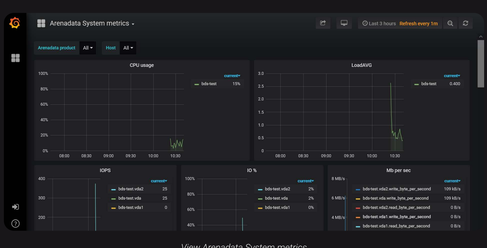
Grafana web-interface

3. Open one of available dashboards. For example, ***Arenadata System metrics***.



Switch to Arenadata System metrics

4. View available graphs.



View Arenadata System metrics

Integrate with ADS cluster

By Yan Chernikov, Lead Developer

Contents

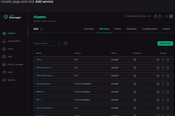
Step 1. Add a monitoring service
Step 2. Map monitoring components
Step 3. Import monitoring settings
Step 4. Install a monitoring service
Step 5. Verify results
- Graphite
- Grafana

In order to determine that a new cluster should be monitored, it is necessary to add a separate service Monitoring clients and integrate it with the previously created monitoring cluster. The steps for building integration between a monitoring cluster and ADS cluster are described below. All actions should be performed on the base of ADS cluster.

Step 1. Add a monitoring service

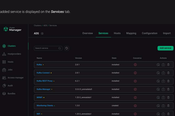
The steps for adding services to a cluster are listed below:

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



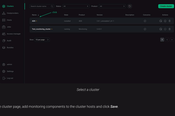
Select a cluster

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



Switch to adding services

3. Select the service Monitoring Clients in the opened dialog and click **Add**



Select services

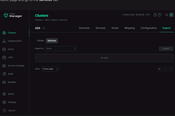
4. As a result, the successfully added service is displayed on the **Services** tab.

The result of successfully adding a service to the cluster

Step 2. Map monitoring components

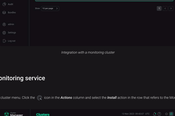
The steps for adding components to hosts are listed below:

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, add monitoring components to the cluster hosts and click **Save**



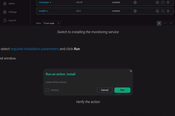
Map components to hosts

IMPORTANT

- Add the System Metric Agent on all hosts.
- JMX metric Collector can only be added to one host.

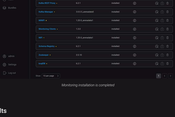
Step 3. Import monitoring settings

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



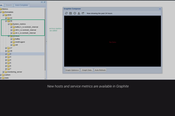
Select a cluster

2. Open the **Import** tab on the cluster page and go to the **Services** tab.



Switch to setting up import of services

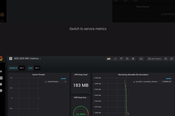
3. In the **Import to** drop-down list, select the Monitoring Clients service.
4. Select the **Graphite** and **Grafana** services of the previously created monitoring cluster.
5. Click **Import**



Integration with a monitoring cluster

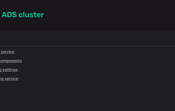
Step 4. Install a monitoring service

1. Open the **Services** tab on the cluster menu. Click the ⋮ icon in the **Actions** column and select the **Install** action in the row that refers to the Monitoring Clients.



Switch to installing the monitoring service

2. In the dialog box that opens, select required installation parameters and click **Run**.
3. Verify the action in the opened window.

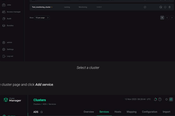


Verify the action

4. Wait until the job is completed. After successful installation, the monitoring server changes its state from `Created` → `Installed`

NOTE

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



Monitoring installation is completed

Step 5. Verify results

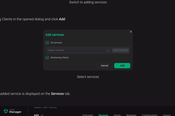
To ensure that installation process is completed successfully, check the following:

- System metrics are collected from hosts of your cluster. To do this, select the host where the monitoring cluster has been deployed earlier.
- Along with system metrics, service metrics are collected from the hosts of your cluster. For example, ADS cluster provides metrics for such services as Kafka, NiFi, MiNiFi.

You can check it both in the Graphite and Grafana web interface.

Graphite

1. In the browser bar, enter an address of the host where the monitoring cluster is deployed. To specify the port number, use the value of the **Web interface TCP port** parameter from the **ip_and_ports** section defined when configuring the Graphite service during the monitoring cluster installation (by default, **2004**).
2. Expand a tree at the left side of the opened form. System and service metrics are displayed there, grouped by hosts. You can select some of these metrics to view their change dynamics in the form of a graph on the right side of the screen.



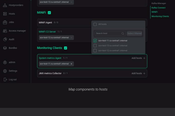
New hosts and service metrics are available in Graphite

Grafana

1. In the browser bar, enter an address of the host where the monitoring cluster is deployed. To specify the port number, use the value of the **Port** parameter from the **ip_and_ports** section defined when configuring the Grafana service during the monitoring cluster installation (by default, **3000**).
2. Select the **Home** menu item in the opened form.
3. Notice that new dashboards are available. Select one of them to view the service metrics that are collected from your cluster.

Switch to service metrics

4. View available graphs.



View service metrics

Offline installation

To Table of Contents

This topic describes **offline** installation of ADS. 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 ADS.

ADCM

Step 1.

Install ADCM

ADCM

Step 2.

Prepare hosts

ADCM

Step 3.

Install ET cluster

ADCM

Step 4.

Install ADS cluster

ADCM

Step 5.

Install monitoring

Install ADOM

Contents

Table of Contents

Step 1. Check prerequisites
- Software and hardware requirements
- Install Docker
- Disable SELinux
- Open Firewalld
- Disable the Firewall (Ubuntu 20 software and software)

Step 2. Install ADOM

Step 3. Start ADOM

Step 4. Check installation
- Check the Docker container status
- Check the ADOM web interface
- Get the ADOM URL

Step 5. Update ADOM

ADOM version supported: Software installing ADOM, we recommend to apply the latest patch.

This document is for the following versions:

- ADOM 6.4.x

Step 1. Check prerequisites

Software and hardware requirements

Before you begin to install and configure ADOM, ensure your system meets the following requirements:

- One of the following operating systems:
  - Ubuntu 20.04
  - CentOS 7
  - CentOS 8
  - Ubuntu 18.04
  - RHEL/CentOS 7.6-7.9.2009
  - RHEL/CentOS 8.0-8.3.2011
  - Ubuntu 20.04 LTS
- A minimum of 16 GB RAM and 100 GB HDD storage. The required storage is dependent on the number of devices and logs. Local disk is recommended. Logs can be stored on NFS or CIFS mounted volumes, but this may affect performance.
- The hostname must be resolved by the following conditions:

/etc/hosts

Hostname must be resolved in the **/etc/hosts**. The target value must be the same as the hostname defined in the customer machine/server.

/etc/resolv.conf

Setting up the DNS server in **/etc/resolv.conf** and resolve the target of the DNS server IP address and response to accurate DNS address that can be resolved by the customer machine/server. Please refer to the following example:

```
search fortinet.com
nameserver 8.8.8.8
```

To confirm that the hostname has been resolved by the following steps (Check the same name server IP address as the previous step):

```
$ sudo cat /etc/resolv.conf
```

To check that the hostname can be resolved, run the command **dig** (Check the same name server IP address as the previous step):

```
$ dig +short fortinet.com
```

Example: **CentOS_7.9.2009_1.0.0-1.el7.noarch.rpm** (CentOS 7.9.2009)

```
$ sudo yum-config-manager --add-repo=http://<IP address>/CentOS_7.9.2009_1.0.0-1.el7.noarch.rpm
$ sudo yum install adom
```

Example: **Ubuntu_18.04Server_1.0.0-1.deb** (Ubuntu 18.04 Server)

```
$ sudo apt-add-repository "deb [trusted=yes] http://<IP address>/Ubuntu_18.04Server_1.0.0-1.deb/ ./"
$ sudo apt-get update
$ sudo apt-get install adom
```

Example: **CentOS_7.9.2009_1.0.0-1.el7.noarch.rpm** (CentOS 7.9.2009)

```
$ sudo yum-config-manager --add-repo=http://<IP address>/CentOS_7.9.2009_1.0.0-1.el7.noarch.rpm
$ sudo yum install adom
```

Example: **Ubuntu_18.04Server_1.0.0-1.deb** (Ubuntu 18.04 Server)

```
$ sudo apt-add-repository "deb [trusted=yes] http://<IP address>/Ubuntu_18.04Server_1.0.0-1.deb/ ./"
$ sudo apt-get update
$ sudo apt-get install adom
```

/etc/yum.conf

Setting up the DNS server in **/etc/yum.conf** and resolve the target of the DNS server IP address and response to accurate DNS address that can be resolved by the customer machine/server. Please refer to the following example:

```
[main]
cachedir=/var/cache/yum/$basearch/$releasever
keepcache=0
debuglevel=2
logfile=/var/log/yum.log
exactarch=1
obsoletes=1
gpgcheck=1
plugins=1
installonly_limit=5
bugtracker_url=http://bugs.centos.org/set_project.php?project_id=23&ref=http://bugs.centos.org/bug_report_page.php?category=yum
distroverpkg=centos-release
nameserver=8.8.8.8
```

/etc/apt/sources.list

Setting up the DNS server in **/etc/apt/sources.list** and resolve the target of the DNS server IP address and response to accurate DNS address that can be resolved by the customer machine/server. Please refer to the following example:

```
# deb cdrom:[Ubuntu 18.04 LTS _Bionic Beaver_ - Release amd64 (20180426)]/ bionic main restricted
deb http://archive.ubuntu.com/ubuntu bionic main restricted
nameserver 8.8.8.8
```

Install Docker

/etc/hosts

Hostname must be resolved in the **/etc/hosts**. The target value must be the same as the hostname defined in the customer machine/server.

/etc/resolv.conf

Setting up the DNS server in **/etc/resolv.conf** and resolve the target of the DNS server IP address and response to accurate DNS address that can be resolved by the customer machine/server. Please refer to the following example:

```
search fortinet.com
nameserver 8.8.8.8
```

Many of the required dependencies of Docker components can be found in standard operating system repositories. Docker may have RPM or dependency packages that are not found in standard OS repositories.

CentOS 7/RHEL 7

1. Install the **EPEL** package repository which provides access to many dependent Linux packages:

```
$ sudo yum update -y
$ sudo yum install epel-release
```

2. Install the Docker CE package repository:

```
$ sudo yum install -y yum-utils device-mapper-persistent-data lvm2
$ sudo yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo
```

3. Install Docker CE:

```
$ sudo yum install -y docker-ce docker-ce-cli containerd.io
```

4. Start Docker CE and enable it to start at boot:

```
$ sudo systemctl start docker
$ sudo systemctl enable docker
```

CentOS 8/RHEL 8

1. Install the **EPEL** package repository which provides access to many dependent Linux packages:

```
$ sudo yum update -y
$ sudo yum install epel-release
```

2. Install the Docker CE package repository:

```
$ sudo yum install -y yum-utils device-mapper-persistent-data lvm2
$ sudo yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo
```

3. Install Docker CE:

```
$ sudo yum install -y docker-ce docker-ce-cli containerd.io
```

4. Start Docker CE and enable it to start at boot:

```
$ sudo systemctl start docker
$ sudo systemctl enable docker
```

Ubuntu 18.04 LTS

1. Install the **EPEL** package repository which provides access to many dependent Linux packages:

```
$ sudo apt update -y
$ sudo apt install -y apt-transport-https ca-certificates curl gnupg-agent software-properties-common
```

2. Install the Docker CE package repository:

```
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
$ sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
```

3. Install Docker CE:

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

4. Start Docker CE and enable it to start at boot:

```
$ sudo systemctl start docker
$ sudo systemctl enable docker
```

Ubuntu 20.04 LTS

1. Install the **EPEL** package repository which provides access to many dependent Linux packages:

```
$ sudo apt update -y
$ sudo apt install -y apt-transport-https ca-certificates curl gnupg-agent software-properties-common
```

2. Install the Docker CE package repository:

```
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
$ sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
```

3. Install Docker CE:

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

4. Start Docker CE and enable it to start at boot:

```
$ sudo systemctl start docker
$ sudo systemctl enable docker
```

All Linux PC

1. Install **Git**:

```
$ sudo apt-get install git
```

2. Install Docker Compose:

```
$ sudo curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
$ sudo chmod +x /usr/local/bin/docker-compose
$ sudo ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose
```

3. Add the Docker repository:

```
$ sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
```

4. Update the package database with Docker packages from the newly added repo:

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

5. Install Docker:

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

6. Make sure Docker is running and enabled to start at boot:

```
$ sudo systemctl status docker
$ sudo systemctl enable docker
```

7. Add your user to the docker group:

```
$ sudo usermod -aG docker $USER
```

8. Verify that Docker can download and run images:

```
$ sudo docker run hello-world
```

Disable SELinux

CentOS

SELinux is enabled in CentOS. You can check it in CentOS 7 or 8 by:

```
$ sestatus
```

To disable SELinux permanently, edit the **/etc/selinux/config** file and change the following line and then reboot your system:

```
SELINUX=disabled
```

Or, enter:

```
$ sudo -i /etc/selinux/config
```

You can check if the change has been applied successfully by entering:

```
$ sestatus
```

Ubuntu

SELinux is not enabled by default on the Ubuntu OS, but can be installed using the following command:

```
$ sudo apt-get install policycoreutils
```

You can then check if it is disabled and change it by editing the **/etc/selinux/config** file in the same way as in the CentOS example above.

Open Firewalld

Firewalld is enabled by default on the CentOS operating system. It is recommended to open the following ports:

```
$ sudo firewall-cmd --zone=public --add-port=8080/tcp --permanent
$ sudo firewall-cmd --zone=public --add-port=8443/tcp --permanent
$ sudo firewall-cmd --zone=public --add-port=514/udp --permanent
$ sudo firewall-cmd --zone=public --add-port=3306/tcp --permanent
$ sudo firewall-cmd --zone=public --add-port=9200/tcp --permanent
$ sudo firewall-cmd --zone=public --add-port=5432/tcp --permanent
$ sudo firewall-cmd --zone=public --add-port=161/udp --permanent
$ sudo firewall-cmd --zone=public --add-port=162/udp --permanent
$ sudo firewall-cmd --zone=public --add-port=443/tcp --permanent
$ sudo firewall-cmd --reload
```

Disable the Firewall (Ubuntu 20 software and software)

Ubuntu

Firewall is disabled in Ubuntu 20 by default.

To disable the Firewall permanently, run the **/etc/init.d/ufw stop** command. In the case where the Firewall is enabled, run the **ufw disable** command.

```
$ sudo -i /etc/init.d/ufw stop
```

You can check if the change has been applied successfully:

```
$ sudo ufw status
Status: inactive
```

All Linux PC

Add a rule to allow communication between Docker containers:

```
$ sudo iptables -I FORWARD -i docker0 -o docker0 -j ACCEPT
```

Allow access to the Docker CLI without root privileges

To run Docker commands as a non-root user, add your user to the **docker** user group with the following command:

```
$ sudo usermod -aG docker $USER
```

Log out and log back in for this to take effect.

Allow access to the Docker CLI without root privileges

To run Docker commands as a non-root user, add your user to the **docker** user group with the following command:

```
$ sudo usermod -aG docker $USER
```

Log out and log back in for this to take effect.

Step 2. Install ADOM

Many of the commands in this step require root privileges. Log in as root or switch to root with **sudo** for the following steps.

CentOS

Installing the yum-utils package (which provides the yum-config-manager utility) and then run the following command:

```
$ sudo yum-config-manager --add-repo=http://<IP address>/CentOS_7.9.2009_1.0.0-1.el7.noarch.rpm
```

Where **<IP address>** is the local server where the .rpm file is stored.

Install the ADOM package:

```
$ sudo yum install adom
```

Ubuntu

Installing the software-properties-common package:

```
$ sudo apt-get install software-properties-common
```

Add the ADOM repository:

```
$ sudo apt-add-repository "deb [trusted=yes] http://<IP address>/Ubuntu_18.04Server_1.0.0-1.deb/ ./"
```

Where **<IP address>** is the local server where the .deb file is stored.

Update the package database with Docker packages from the newly added repo:

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

Install ADOM:

```
$ sudo apt-get install adom
```

All Linux PC

1. Create a new Linux user account for the ADOM daemon:

```
$ sudo useradd -m adom
```

2. Add the ADOM binaries to the user's PATH variable:

```
$ sudo su - adom
$ echo 'export PATH=$PATH:/home/adom/.local/bin' >> ~/.bashrc
$ source ~/.bashrc
```

3. Download the ADOM image:

```
$ docker pull fortinet/adom:6.4.0
```

4. Create a new persistent container for the ADOM data:

```
$ docker create -v /data --name adom-data busybox
```

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

```
$ docker run -dt --volumes-from adom-data --name adom --restart unless-stopped -p 8080:8080 -p 8443:8443 -p 514:514/udp -p 3306:3306 -p 9200:9200 -p 5432:5432 -p 161:161/udp -p 162:162/udp -p 443:443 fortinet/adom:6.4.0
```

Where **<IP address>** is the local server where the .rpm file is stored.

Update the package database with Docker packages from the newly added repo:

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

Install ADOM:

```
$ sudo apt-get install adom
```

All Linux PC

1. Create a new Linux user account for the ADOM daemon:

```
$ sudo useradd -m adom
```

2. Add the ADOM binaries to the user's PATH variable:

```
$ sudo su - adom
$ echo 'export PATH=$PATH:/home/adom/.local/bin' >> ~/.bashrc
$ source ~/.bashrc
```

3. Download the ADOM image:

```
$ docker pull fortinet/adom:6.4.0
```

4. Create a new persistent container for the ADOM data:

```
$ docker create -v /data --name adom-data busybox
```

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

```
$ docker run -dt --volumes-from adom-data --name adom --restart unless-stopped -p 8080:8080 -p 8443:8443 -p 514:514/udp -p 3306:3306 -p 9200:9200 -p 5432:5432 -p 161:161/udp -p 162:162/udp -p 443:443 fortinet/adom:6.4.0
```

Step 3. Start ADOM

ADOM can be started and enabled to start at boot. As you start up and check ADOM, you can check the status of the ADOM daemon:

1. To start ADOM, run the following command:

```
$ sudo systemctl start adom
```

2. To check the status of ADOM, run the following command:

```
$ sudo systemctl status adom
```

3. To stop ADOM, run the following command:

```
$ sudo systemctl stop adom
```

4. To enable ADOM to start at boot, run the following command:

```
$ sudo systemctl enable adom
```

Step 4. Check installation

Check the Docker container status

To check and confirm that the container, **ADOM**, and the images are running and started, run the following command:

```
$ sudo docker ps
```

The example of the command execution result is described. Make sure that the **STATUS** column is healthy (**Up**).

```
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS                                                
```

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

For offline installation, the SSH hostprovider is recommended.

CAUTION

If you are going to use ADS or ADS Control with ADPS, you should specify FQDNs when creating ADS/ADS Control hosts. The maximum FQDN length is 48 symbols.

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.

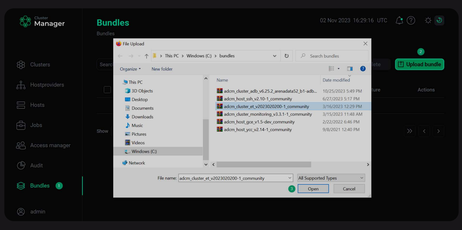
Step 1. Get the Enterprise version of a cluster bundle

Please get a cluster bundle from the Arenadata support team.

Step 2. Upload a cluster bundle to ADCM

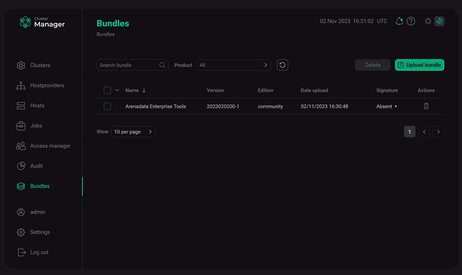
To upload a bundle to ADCM, follow the steps:

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



Upload a bundle

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



The result of successful uploading a bundle

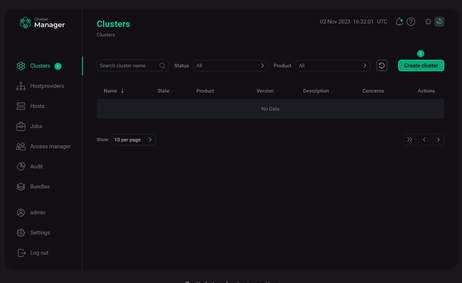
Step 3. Create a cluster based on the uploaded bundle

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.

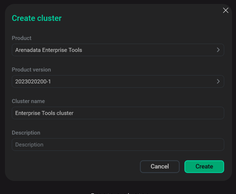
The steps for cluster creation are given below:

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



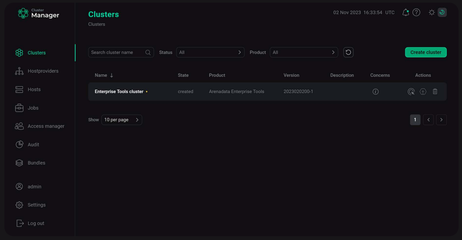
Switch to cluster creation

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 a cluster

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



The result of successful cluster creation

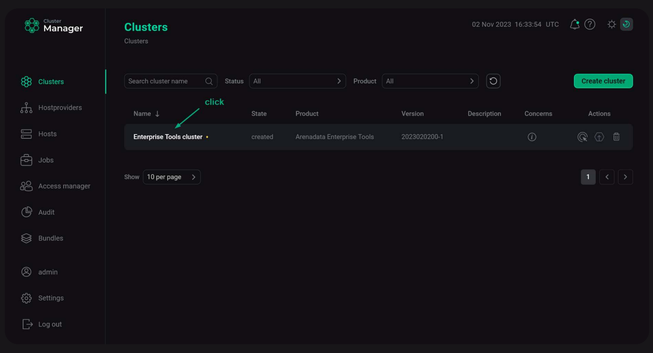
Add services

Daria Barysheva

To Table of Contents

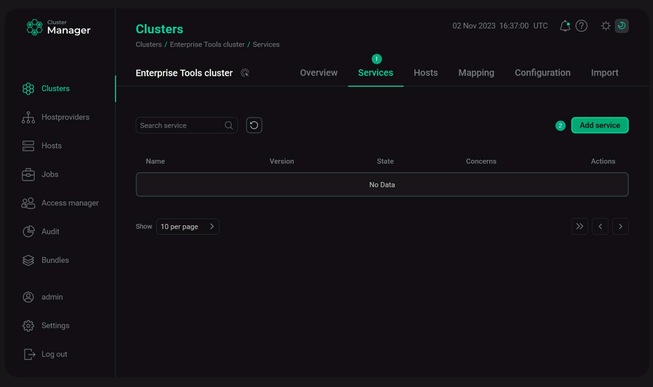
In ADCM a **service** means a software that performs some function. The steps for adding services to a cluster are listed below:

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



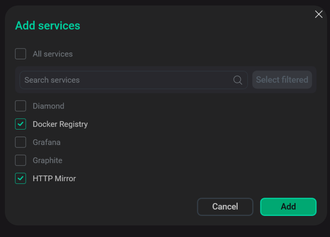
Select a cluster

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



Switch to adding services

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



Select services

The brief description of available services is listed below.

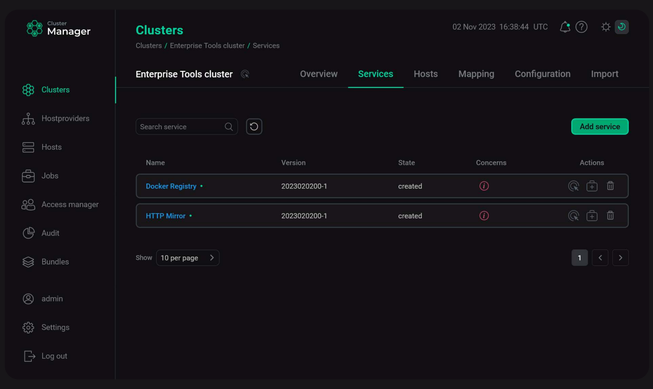
Services that should be added to the Enterprise Tools cluster

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.



The result of successful adding services to a cluster

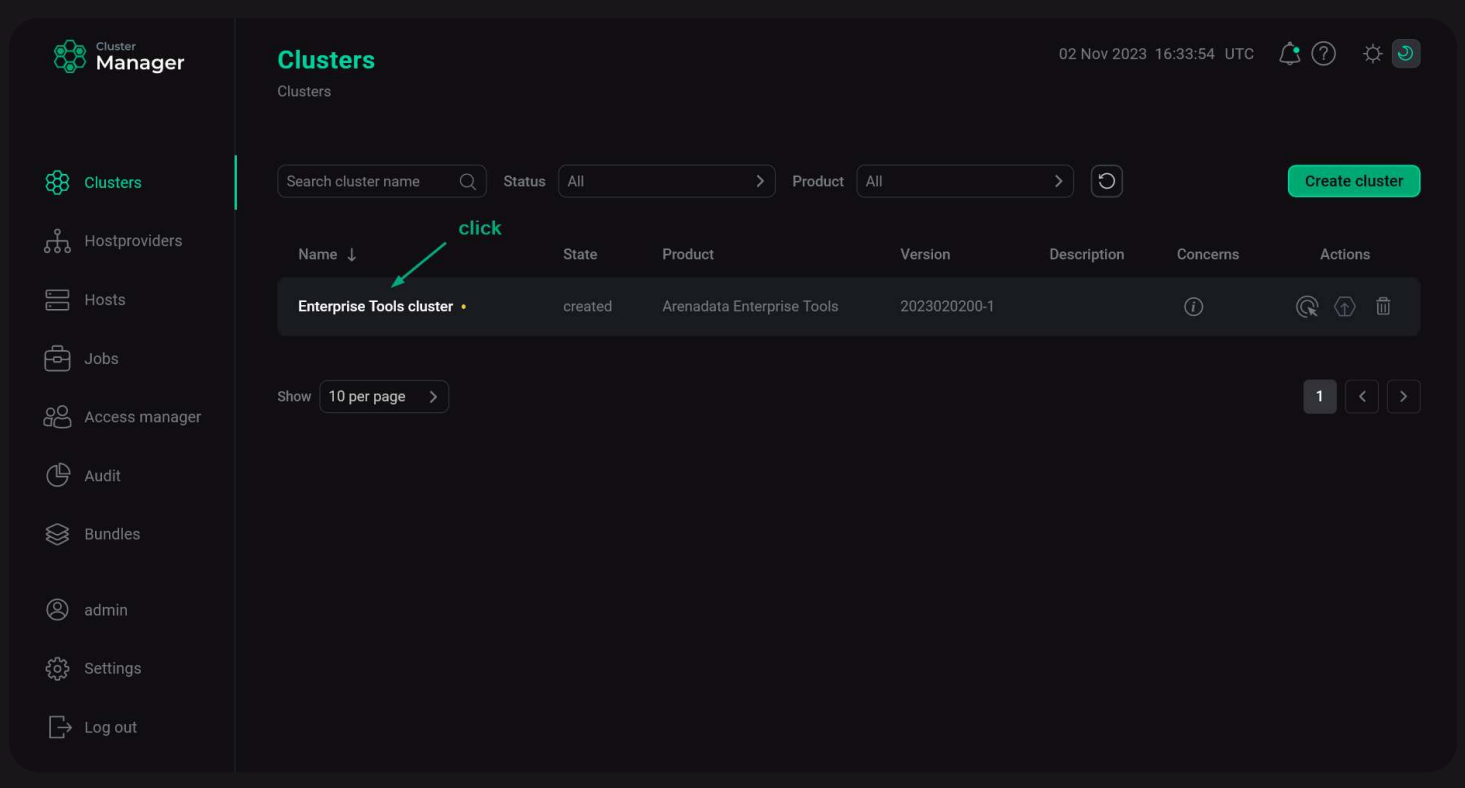
Add hosts to a cluster

Daria Barysheva

To Table of Contents

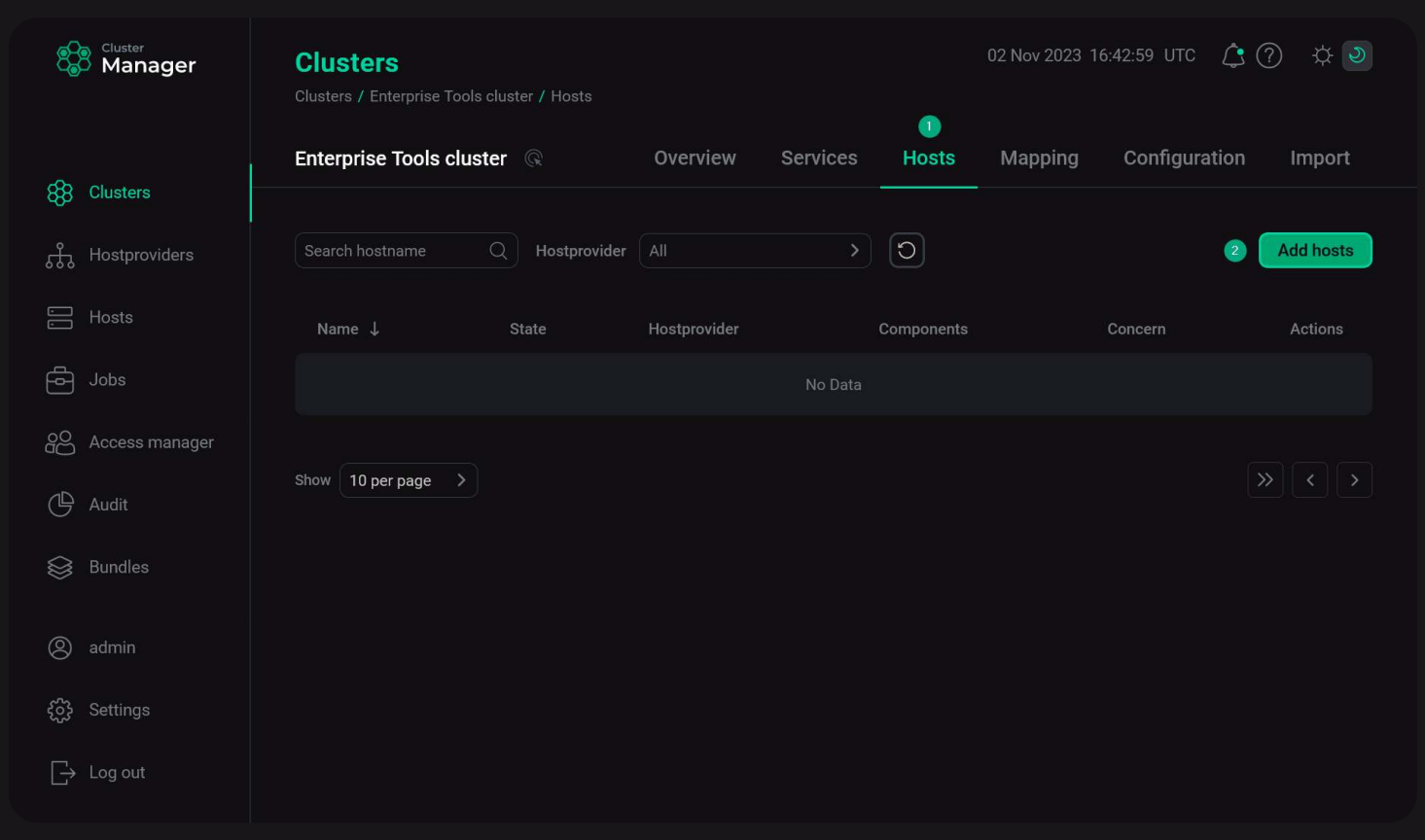
The steps for adding hosts to a cluster are listed below:

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



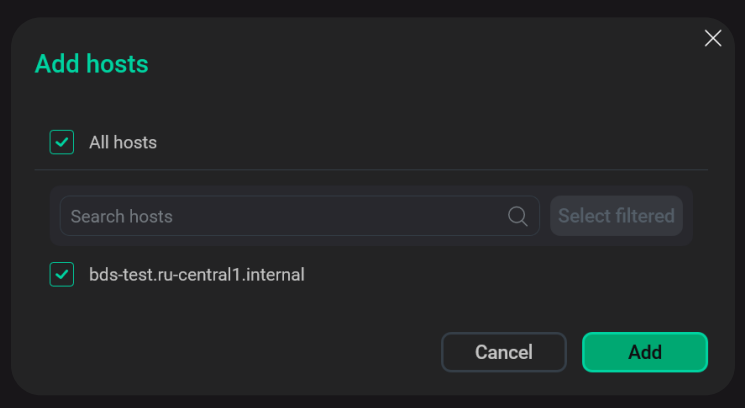
Select a cluster

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



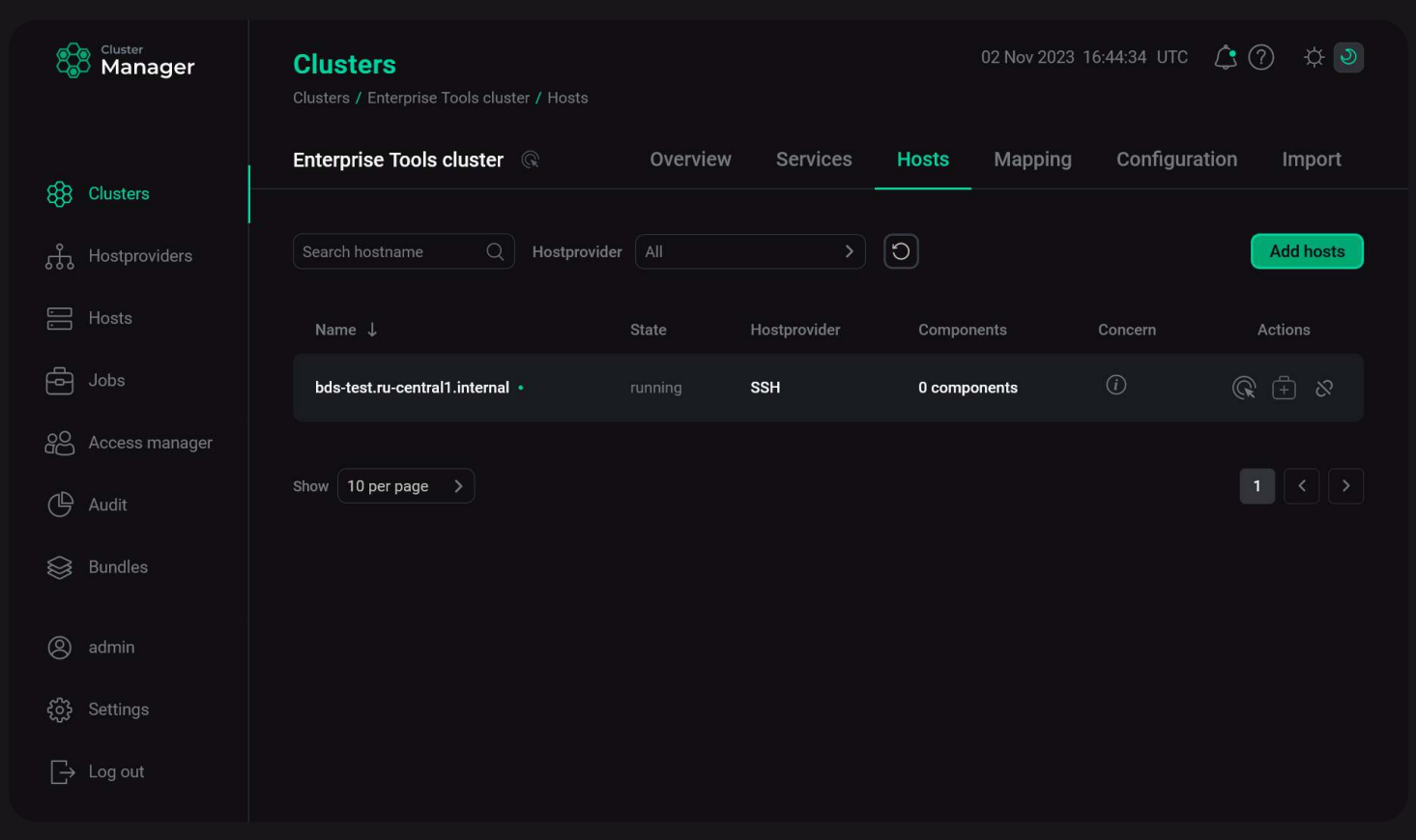
Switch to adding hosts

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



Select hosts

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



The result of successful adding hosts to a cluster

Add components

Daria Barysheva

To Table of Contents

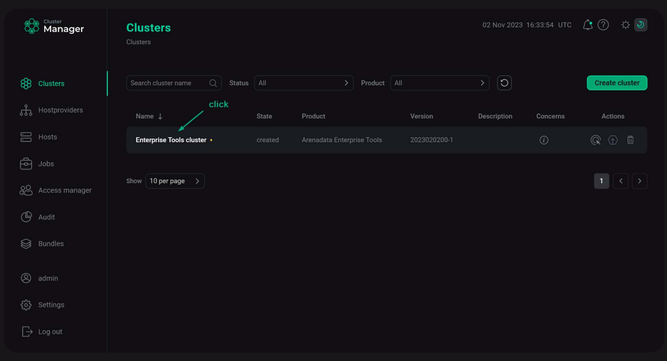
In ADCM a **component** means a part of a service that should be deployed on one or several cluster hosts.

IMPORTANT

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

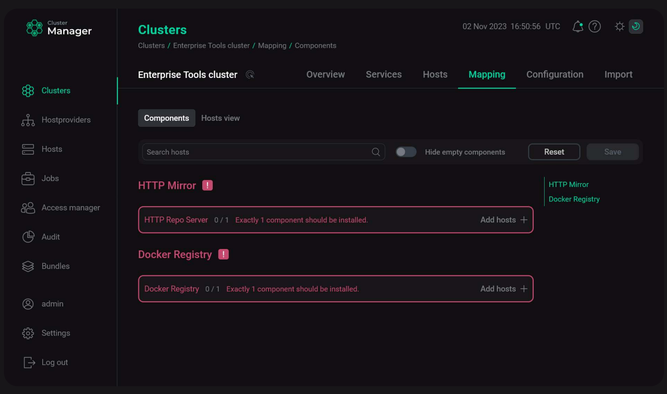
The steps for adding components to hosts are listed below:

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



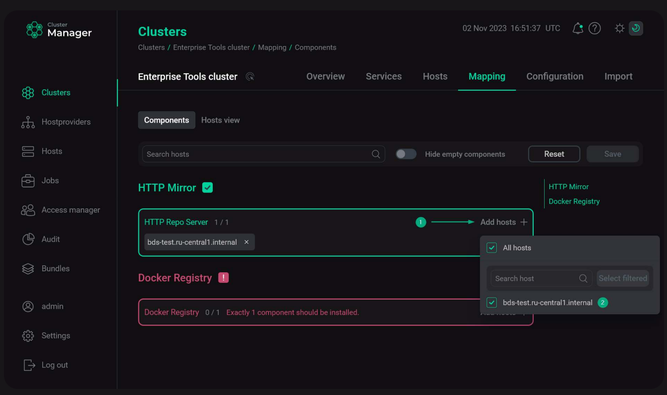
Select a cluster

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



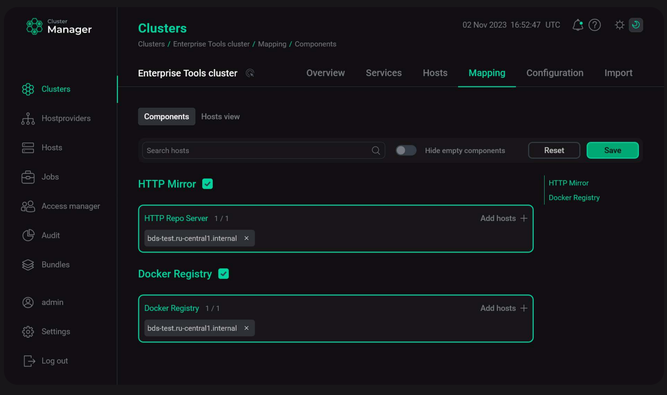
Switch to mapping service components

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

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



Save mapping of components

Install a cluster

Olga Semme, Daria Barysheva

Contents

To Table of Contents

Step 1. Run installation

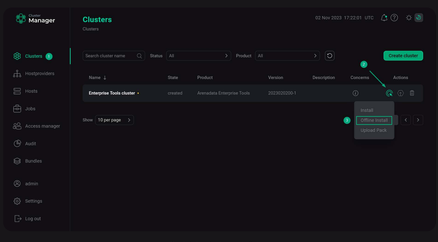
Step 2. Upload packages for offline installation

Step 3. Verify results

Step 1. Run installation

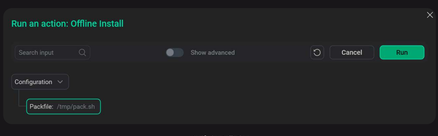
The last step of adding a new Enterprise Tools cluster is to install its services. To do this:

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



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***.

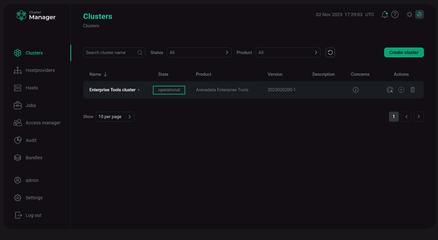


Verify installation

3. Wait until the installation process is completed.

NOTE

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

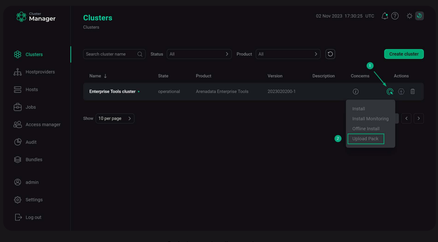


Cluster installation is completed

Step 2. Upload packages for offline installation

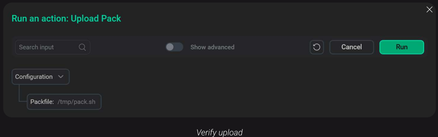
In addition to installing the cluster, you should process the shell script that is received from the Arenadata support team. It is necessary to enable offline installation of products in the future. To do this:

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



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***.



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 ADS and ADS Control installation.

```
total 0
drwxr-xr-x. 5 root root 45 May 23 16:01 .
drwxr-xr-x. 3 root root 28 May 23 16:08 ..
drwxr-xr-x. 3 root root 17 May 23 16:01 ADM
drwxr-xr-x. 3 root root 19 May 23 16:08 ADS
drwxr-xr-x. 3 root root 21 Nov 23 19:12 ADSCC
drwxr-xr-x. 3 root root 19 May 23 16:01 zookeeper
```

Create a cluster

Olga Semme, 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.

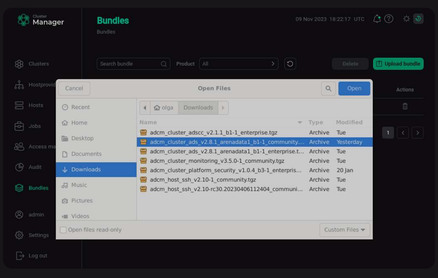
Step 1. Get the Enterprise version of a cluster bundle

Please get a cluster bundle from the Arenadata support team.

Step 2. Upload a cluster bundle to ADCM

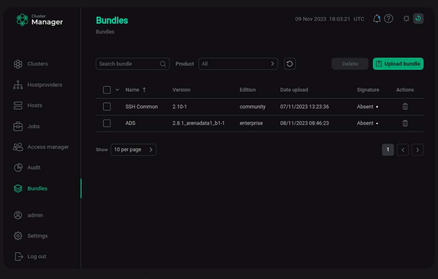
To upload a bundle to ADCM, follow the steps:

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



Upload a bundle

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



The result of successful uploading a bundle

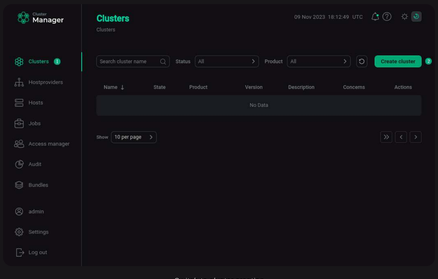
Step 3. Create a cluster based on the uploaded bundle

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.

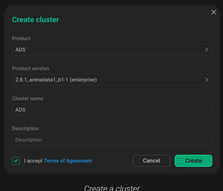
The steps for cluster creation are given below:

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



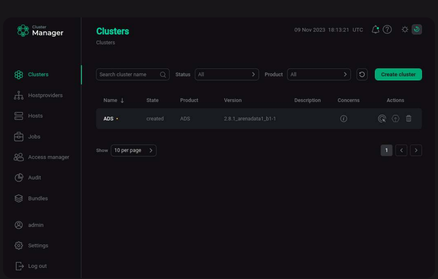
Switch to cluster creation

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 a cluster

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



The result of successful cluster creation

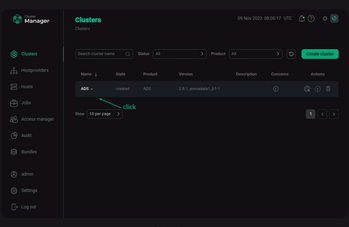
Add services

Olga Semme, Daria Barysheva

To Table of Contents

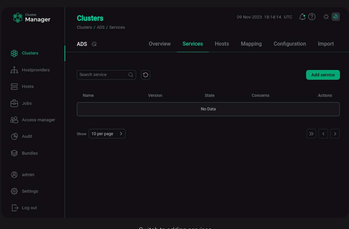
In ADCM a **service** means a software that performs some function. Examples of services for ADS clusters: NiFi, Kafka, ZooKeeper, etc. The steps for adding services to a cluster are listed below:

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



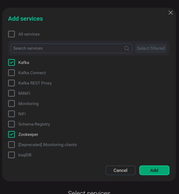
Select a cluster

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



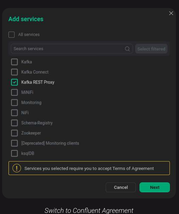
Switch to adding services

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



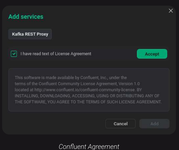
Select services

When adding ksqlDB, Schema-Registry, Kafka REST Proxy services, you should sign a Confluent license agreement. To do this, go to the agreement by clicking ***Next***.



Switch to Confluent Agreement

Next, read the text of the agreement and sign it by clicking ***Accept***.



Confluent Agreement

Click ***Add*** to add a service.

The brief description of available services is listed below.

Services that can be added to the ADS cluster

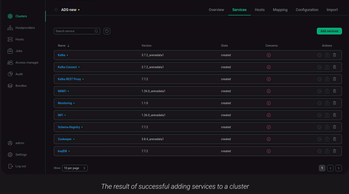
Service	Purpose
ZooKeeper	The centralized coordination service for distributed applications is used to store metadata about the sections (partitions) of topics and Apache Kafka brokers, as well as to select a broker as a Kafka controller. Informs each Kafka broker about the current state of the cluster
Kafka	An open-source distributed event streaming platform used for high-performance data pipelines, streaming analytics, data integration, and mission-critical applications
Kafka REST Proxy	Service provides a RESTful interface to an Apache Kafka cluster, making it easy to produce and consume messages, view the state of the cluster, and perform administrative actions without using the native Kafka protocol or clients
Schema-registry	The service performs the following functions: ■ Storing a version history of all Avro schemas based on a specified subject naming policy with support for several compatibility options. ■ The ability to change Avro schemas according to configured compatibility settings and extended support for these schema types. ■ Providing a RESTful interface for storing and retrieving Avro schemas. ■ Providing serializers that connect to Apache Kafka clients handling message map storage and retrieval.
ksqlDB	An open-source SQL streaming engine that provides real-time data processing using Apache Kafka
NiFi	A simple event (message) processing platform that provides real-time control of data flows from various sources using a graphical interface
NiFi2	NiFi-based service with version 2.x. The service is added in technology preview mode and is not intended for production use. Existing limitations for using NiFi2: ■ NiFi2 can only be installed as part of a separate cluster (NiFi2 + Zookeeper). ■ Before installing the service, you need to enable Java installation from the Arenadata repositories, using the ***arenadata_java*** parameter of the operating system repository to install Java 21 when setting up the cluster.
MiNiFi	A subproject of Apache NiFi — is a complementary data collection approach that supplements the core tenets of NiFi in dataflow management, focusing on the collection of data at the source of its creation
Monitoring	The service deploys the Prometheus server inside ADS to collect and store ADS cluster monitoring metrics. It also supports usage of the Grafana web application for visualization and analysis of information
Monitoring Clients	Agents sending information about hosts and services in monitoring. The service should be added after the monitoring cluster is installed and configured, if monitoring of the current cluster is planned
Kafka Connect	A tool for scalably and reliably streaming data between Kafka and other data systems

The minimal set of services recommended for ADS clusters is described below:

- NiFi
- Kafka
- ZooKeeper

These services make up the core of Streaming and are sufficient for fast and easy installation and management of Arenadata Streaming with Arenadata Cluster Manager. The full list of services depends on the requirements of a particular project. Not all services are required to be installed. For example, if you do not plan to use NiFi, then there is no need to add that service. Also, if a monitoring service is used (not based on Graphite), there is no need to install agents from Monitoring Clients. However, if you plan to use Kafka, the Kafka and ZooKeeper services are required. The same can be said about the NiFi service. In this case, the service can consist of mandatory and optional components. For example, a ksqlDB service consists of a required Server component and an optional Client component.

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



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.

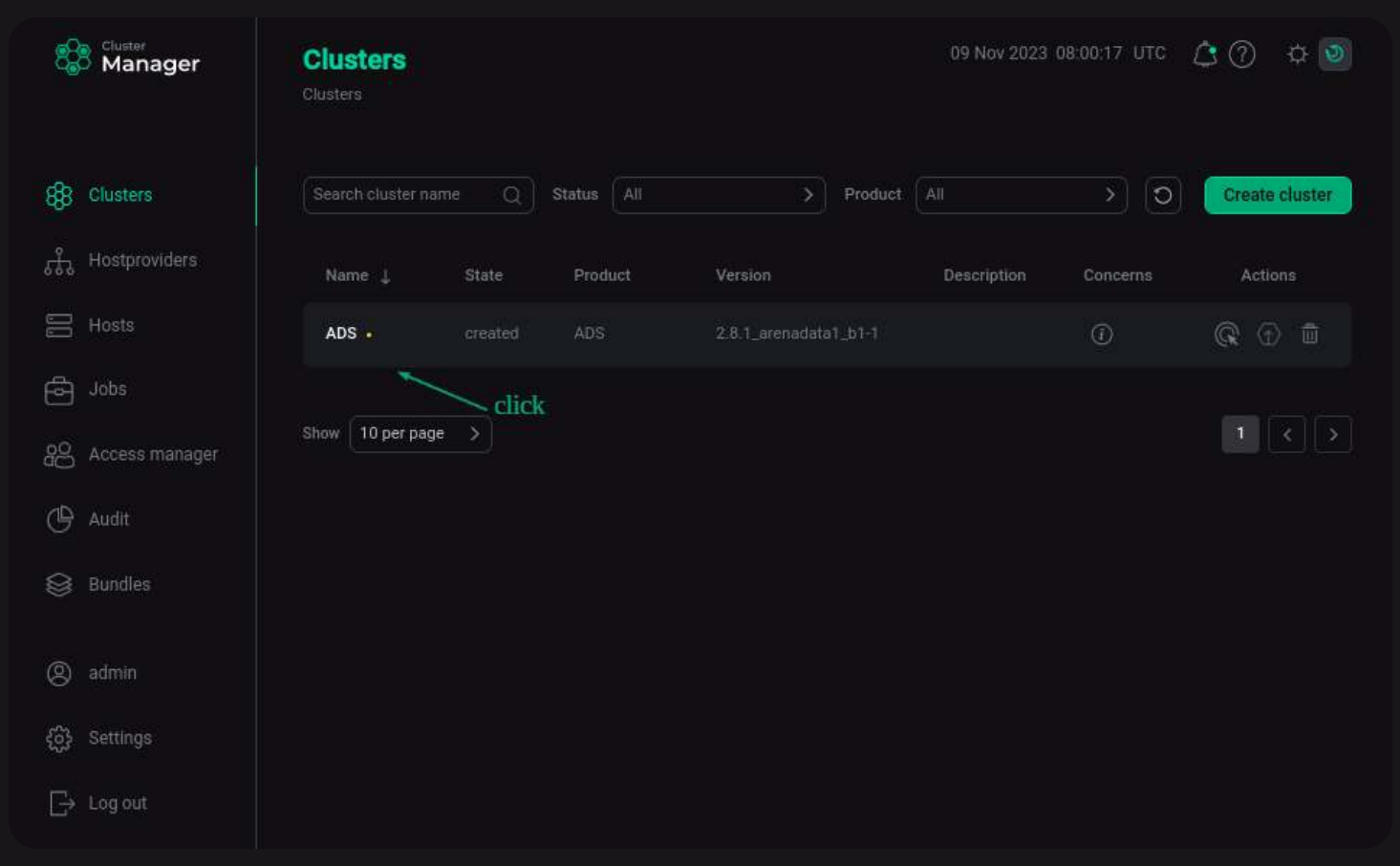
Add hosts to a cluster

Olga Semme, Daria Barysheva

To Table of Contents

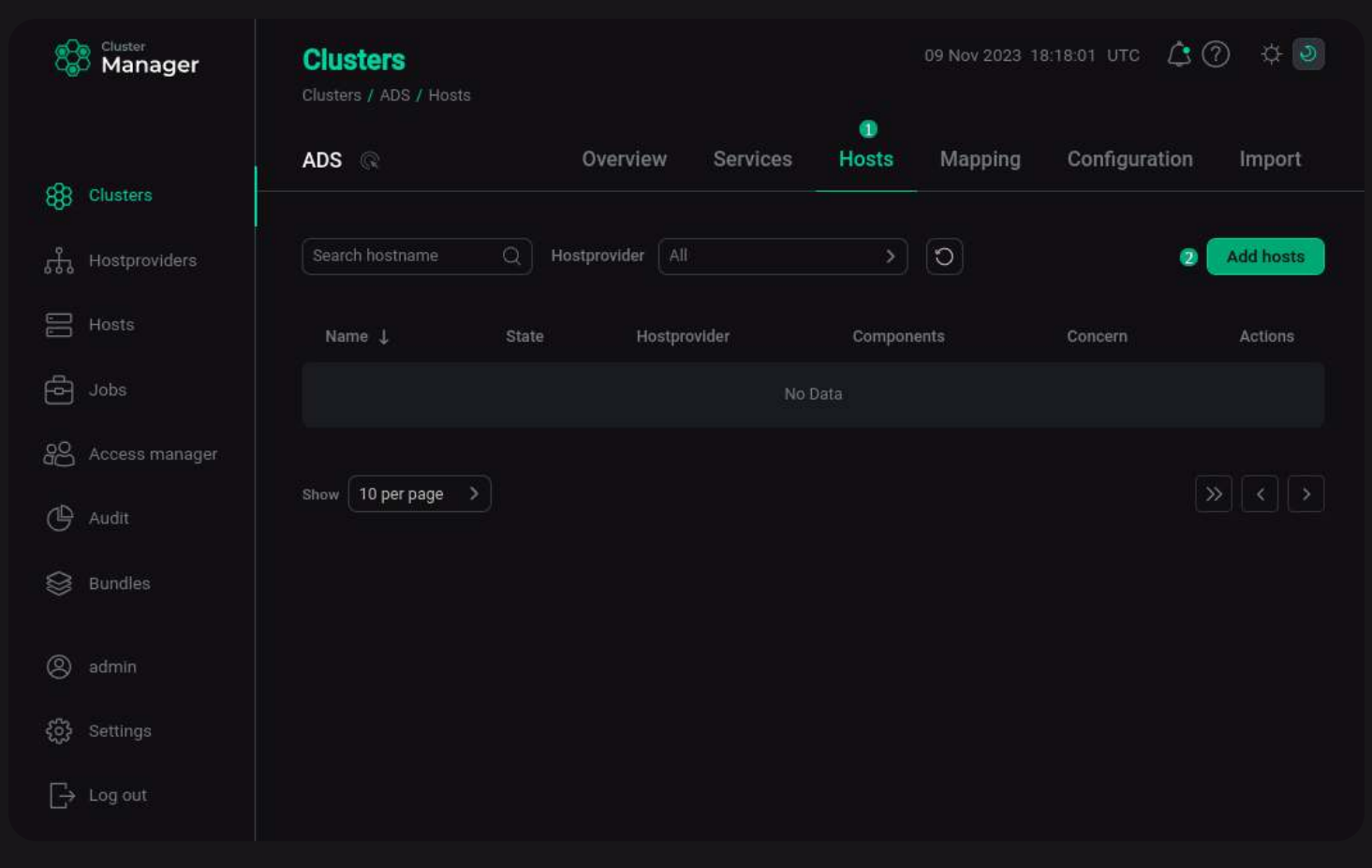
The steps for adding hosts to a cluster are listed below:

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



Select a cluster

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

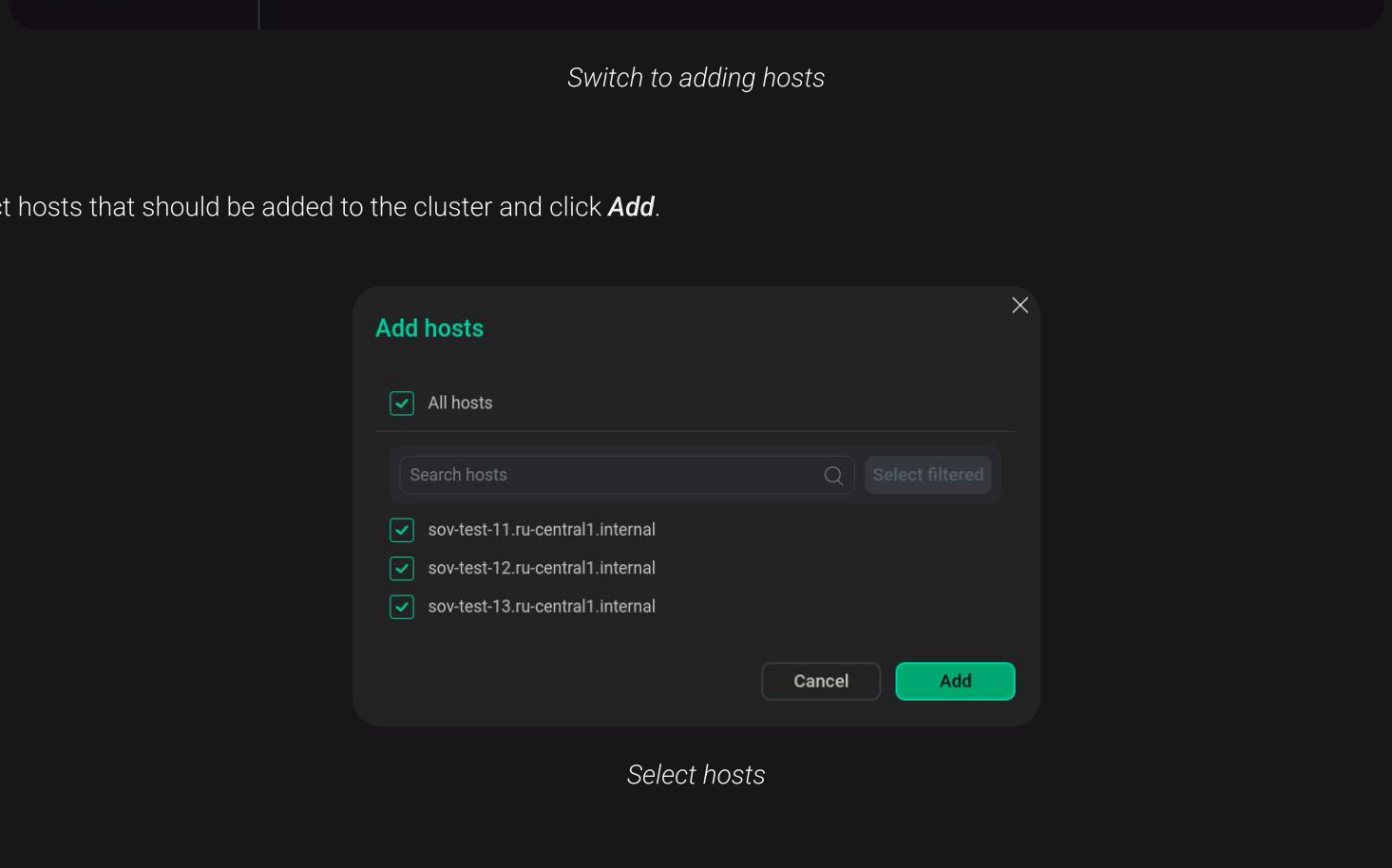


Switch to adding hosts

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

Select hosts

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



The result of successful adding hosts to a cluster

Add components

Olga Semreva, Daria Barysheva

To Table of Contents

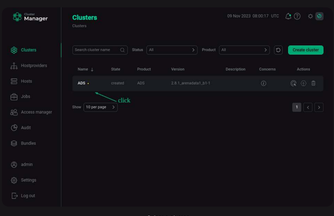
In ADCM a **component** means a part of a service that should be deployed on one or several cluster hosts. For example, NiFi service can consist of such components as NiFi Server, NiFi Registry.

IMPORTANT

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

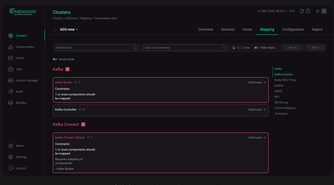
The steps for adding components to hosts are listed below:

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



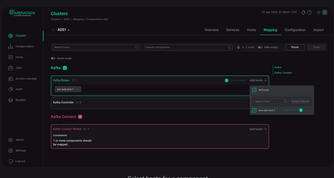
Select a cluster

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



Switch to mapping service components

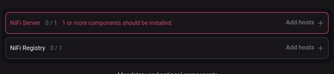
3. In the window that opens, map components to hosts. To assign hosts for each component, click ***Add hosts*** and select hosts in the pop-up window.



Select hosts for a component

Mandatory and optional 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.



Mandatory and optional components

CAUTION

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

The examples of distribution components in ADS clusters consisting of 3 and 5 hosts are described in the table below.

Examples of components distribution

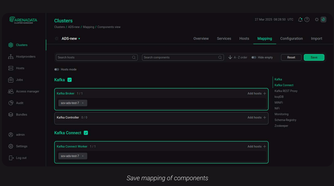
3 hosts

Host	Service	Component
Host 1	Zookeeper	Zookeeper server
	Kafka	Kafka Broker
	Kafka REST Proxy	Kafka REST Proxy
	Schema-Registry	Schema-Registry
Host 2	Zookeeper	Zookeeper Server
	ksqlDB	ksqlDB Client ksqlDB Server
	Kafka	Kafka Broker
	NiFi	NiFi Server
Host 3	Zookeeper	Zookeeper Server
	Kafka	Kafka Broker
	Kafka Connect	Kafka Connect Worker
	NiFi	NiFi Server NiFi Registry
	MiNiFi	MiNiFi C2 Server MiNiFi Agent

5 hosts

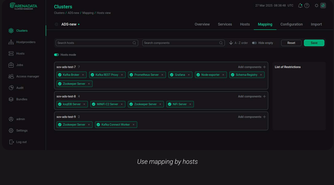
Host	Service	Component
Host 1	Zookeeper	Zookeeper Server
	Kafka	Kafka Broker
	Schema-Registry	Schema-Registry
Host 2	Zookeeper	Zookeeper Server
	Kafka	Kafka Broker
Host 3	Zookeeper	Zookeeper Server
	ksqlDB	ksqlDB Client ksqlDB Server
	Kafka REST Proxy	Kafka REST Proxy
Host 4	Kafka	Kafka Broker
	NiFi	NiFi Server NiFi Registry
Host 5	NiFi	NiFi Server
	MiNiFi	MiNiFi C2 Server MiNiFi Agent
	Kafka Connect	Kafka Connect Worker

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

Last updated: 26th November

ADS

Contents

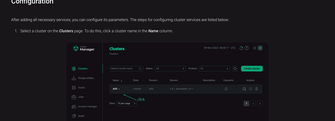
To Table of Contents

Configuration
Components
Configuration groups

Configuration

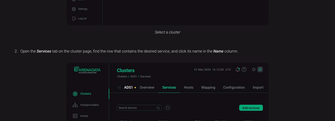
After adding all necessary services, you can configure its parameters. The steps for configuring cluster services are listed below:

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



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**.

Configure a service

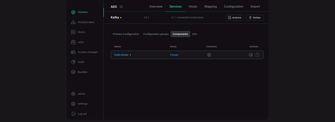
NOTE

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

Components

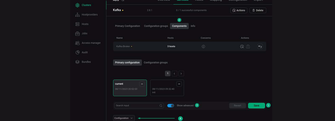
To configure service components:

1. In the menu on the service configuration page, open the **Components** tab and click on the component name in the **Name** column.



Switch to setting up the service component

2. In the window that opens:
   - Set the **Show advanced** switch to active.
   - Fill in or change the configuration parameters of the selected component, if any. As a rule, these are the settings of the Log4j logging library. Templates for log4j_properties files are also available here.
   - Click **Save**.

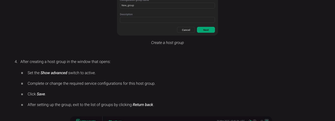


Configure a service component

Configuration groups

To configure one or a group of hosts:

1. On the service configuration page, open the **Configuration groups** tab.
2. On the page that opens, click **Create config group**.

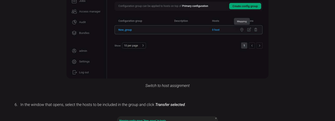


Switch to a host group creation

3. In the window that opens, enter the name of the group and click **Next**.

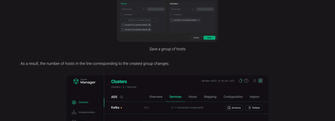
Create a host group

4. After creating a host group in the window that opens:
   - Set the **Show advanced** switch to active.
   - Complete or change the required service configurations for this host group.
   - Click **Save**.
   - After setting up the group, exit to the list of groups by clicking **Return back**.



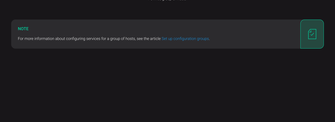
Configure a host group

5. To assign hosts to a group, in the line corresponding to the created group, click on the ⚙ icon.



Switch to host assignment

6. In the window that opens, select the hosts to be included in the group and click *Transfer selected*.



Add hosts to a group

7. After completing the formation of the host group, click *Save*.

Save a group of hosts

As a result, the number of hosts in the line corresponding to the created group changes.

Formed group of hosts

NOTE

For more information about configuring services for a group of hosts, see the article Set up configuration groups.

Configure a cluster

Olga Semme, Daria Barysheva

Contents

To Table of Contents

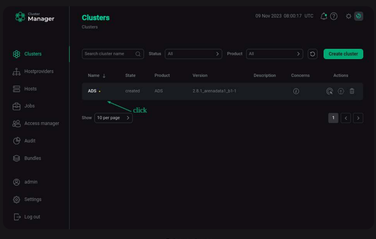
Primary configuration
Configuration groups
Ansible settings

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.



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.

Cluster configuration parameters are listed below:

- ***Reliability control*** — parameters of this section are intended to set the timeout of the DNF software package manager unlocking wait. Displayed if the ***Show advanced*** switch is activated. In Ubuntu, after the host reboot, automatic update starts, which locks DNF.

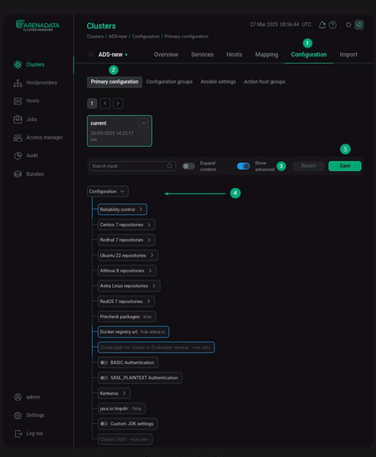
Name	Description	Default value
retries	The maximum number of retries	80
delay	The time delay between retries (in seconds)	30
timeout	Timeout for network ports checks (in seconds)	7200

- Paths to the repositories that are used during the installation process depending on the cluster operating system (CentOS, Red Hat, Ubuntu, ALT Linux 8, or Astra Linux). Only the marked repositories are registered on the cluster servers.

Repositories

Repository name	Parameter	Description	Default value
monitoring	desc	Repository description	Monitoring components url
	url	Arenadata repository path	Specified depending on OS
	enabled	Enables use of the repository	true
ADS	desc	Repository description	ADS repository
	url	Arenadata repository path	Specified depending on OS
	enabled	Enables use of the repository	true
prometheus_repo	desc	Repository description	Prometheus repository
	url	Arenadata repository path	Specified depending on OS
	enabled	Enables use of the repository	true
arenadata_java	desc	Repository description	Arenadata repository for OpenJDK
	url	Arenadata repository path	Specified depending on OS
	enabled	Enables use of the repository. If set to `true` before the ***Install***, ***Add/Remove components***, or ***Update*** action for a cluster or service, the latest version of Java 17 will be installed on hosts	false
ranger plugins	desc	Repository description	Arenadata Ranger plugins repository
	url	Arenadata repository path	Specified depending on OS
	enabled	Enables use of the repository	true

- ***Precheck packages*** — enables package availability check before installation.
- ***Docker registry url*** — specifies the location of Arenadata Docker Registry. Displayed if the ***Show advanced*** switch is activated.
- ***Znode path for cluster in Zookeeper service*** — Znode path for the ADS cluster in the Zookeeper service. Displayed if the ***Show advanced*** switch is activated.
- ***BASIC Authentication*** — configuration for BASIC authentication.
  For a description of enabling authentication, see the article Basic authentication.
- ***SASL_PLAINTEXT Authentication*** — configuration for SASL PLAINTEXT authentication.
  For a description of enabling authentication, see the article SASL PLAINTEXT.
- ***Kerberos*** — configuration for Kerberos authentication.
  The description of enabling authentication is given in the articles:
  - MIT Kerberos
  - Kerberos with MS Active Directory
  - Kerberos with Samba
- ***java.io.tmpdir*** — a Java system property that determines where the JVM writes temporary files.
- ***Custom JDK settings*** — a switch that indicates whether to use a custom Java path for ADS services. When the switch is activated, the ***JAVA_HOME*** field becomes available. Enter a work Java path in that field. The selected path should contain a *bin* folder with a Java binary file, for example */usr/lib/jvm/jre-openjdk/*. If the ***JAVA_HOME*** value is empty, the default system path is used */usr/lib/jvm/java-1.17.0-openjdk.amd64/* (for all operating systems except Centos, RedHat, and AstraLinux).
- ***Cluster UUID*** — displays the universally unique identifier of the cluster.



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

Import ET settings

Olga Semme, Daria Barysheva

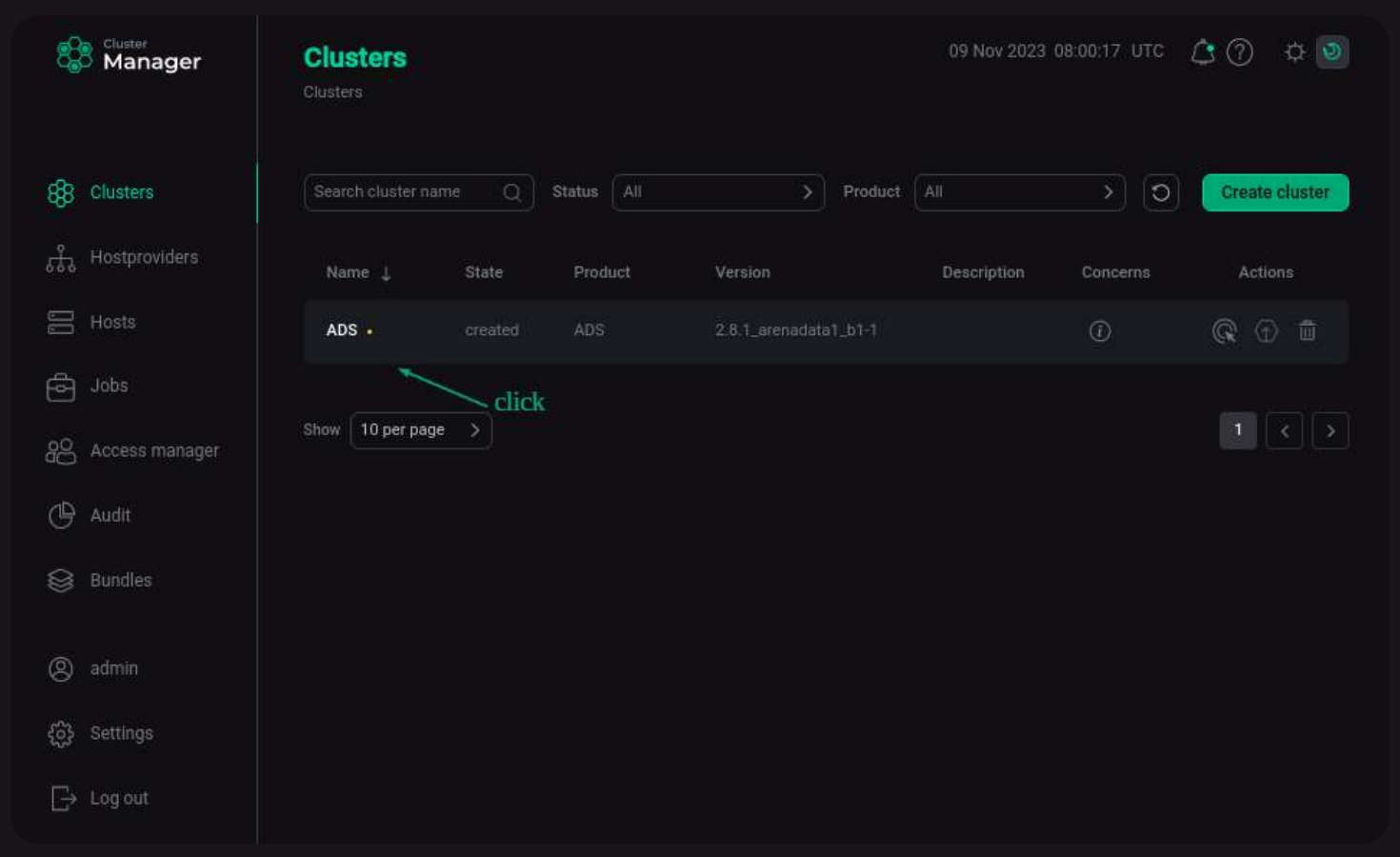
To Table of Contents

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.

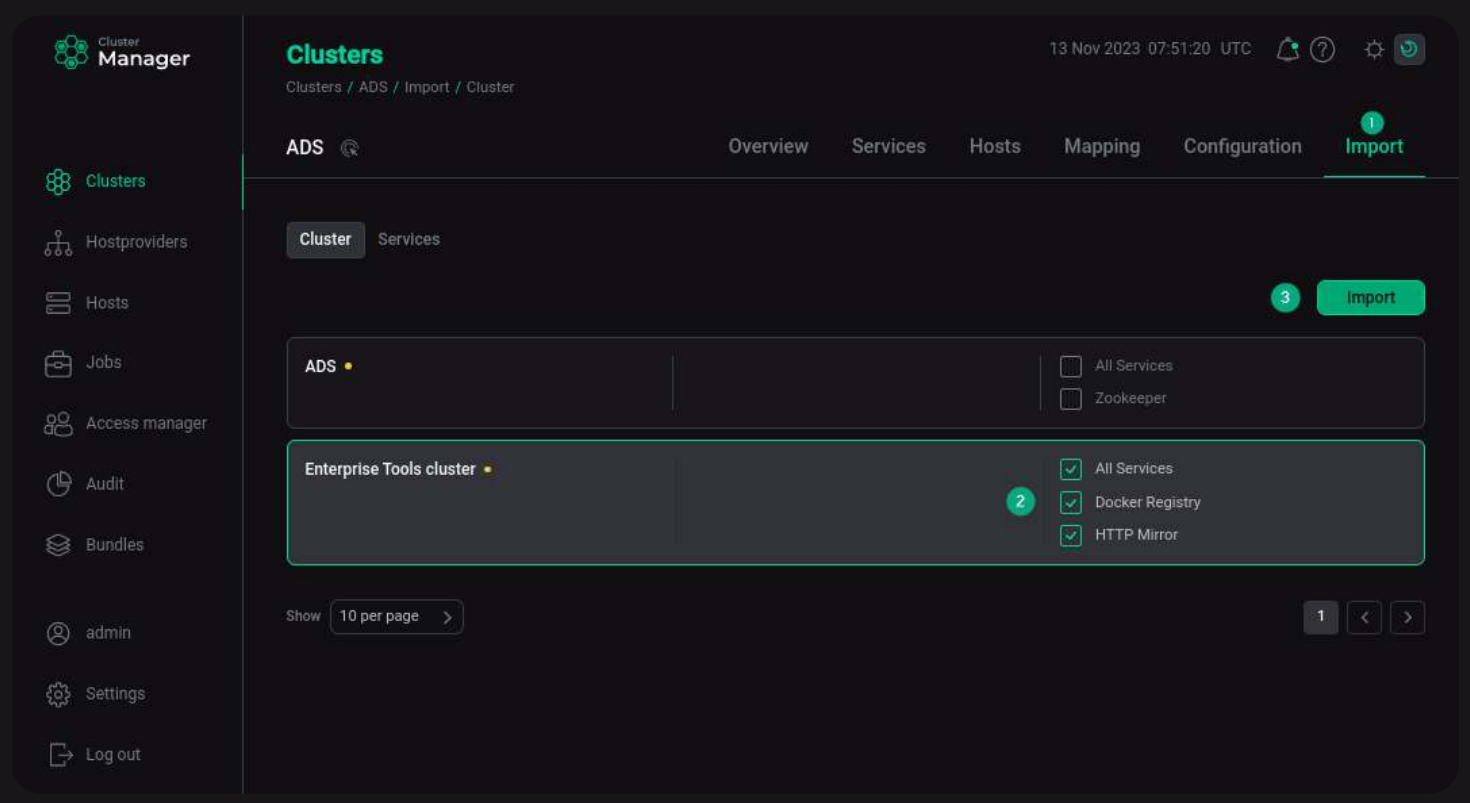


Select a cluster

2. Open the ***Import*** tab in the cluster menu.

3. Select ***Docker Registry*** and ***HTTP Mirror*** services of the previously created ET cluster.

4. Click ***Import***.



Integration with a 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

Olga Semina, Daria Borysheva

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.

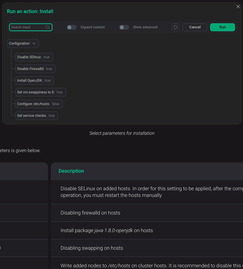
IMPORTANT

The installation must be performed on a clean host with no previous installations of related or similar software.

Whole cluster

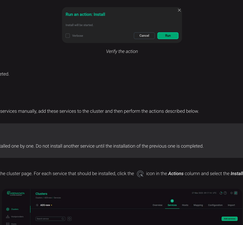
To install all cluster services within a single action, follow the steps:

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



Switch to the whole cluster installation

2. In the dialog box that opens, select the required options for installation.

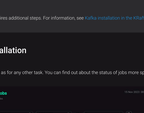


Select parameters for installation

The purpose of the parameters is given below.

Parameter	Description
Disable SELinux	Disable SELinux on added hosts. In order for this setting to be applied, after the completion of the **Install** operation, you must restart the hosts manually
Disable Firewalld	Disabling firewalld on hosts
Install OpenJDK	Install package *java-1.8.0-openjdk* on hosts
Set vm.swappiness to 0	Disabling swapping on hosts
Configure /etc/hosts	Write added nodes to */etc/hosts* on cluster hosts. It is recommended to disable this option if DNS is configured
Set service checks	Enables service checking when starting an action

3. Verify the action in the opened window.



Verify the action

4. Wait until the job is completed.

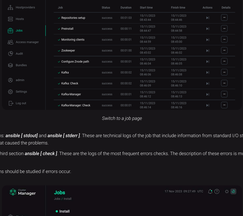
Single services

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.



Switch to installation of single services

2. In the dialog box that opens, select the required parameters for installation and click **Run**.
3. Verify the action in the opened window.

Verify the action

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

IMPORTANT

- When you install services manually, one by one, use the following order:
  - ZooKeeper;
  - Kafka manager, Kafka, NiFi (one by one);
  - other services.
- Installing a Kafka service in the KRaft mode requires additional steps. For information, see Kafka installation in the KRaft mode.

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.

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.

Switch to a job page

A job page contains two sections: ***ansible [stdout]*** and ***ansible [stderr]***. These are technical logs of the job that include information from standard I/O streams *stdout*/*stderr*. These logs can help to understand what caused the problems.

There can also be the optional third section **ansible [check]**. These are the logs of the most frequent errors checks. The description of these errors is more simple and specific, than in two previous technical logs.

The contents of all three sections should be studied if errors occur.

Job details

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`.

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`.

The state of cluster services after successful installation

Install monitoring

Olga Semme

To Table of Contents

Monitoring is an optional but strongly recommended ADS component that you can configure in one of the following ways:

- add the Monitoring service to the ADS cluster (available starting with 3.6.2.2.b1);
- add monitoring services to the Enterprise Tools cluster and integrate it with the ADS cluster (deprecated).

Monitoring service

Olga Ivanova

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 deploys the Prometheus server inside ADS to collect and store ADS cluster monitoring metrics. It also supports usage of the Grafana web application for visualization and analysis of information. The whole described steps required to add the service to an ADS cluster.

Overview

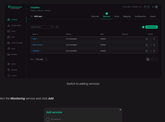
When you add the Monitoring service to an ADS cluster, Node Exporter is installed on all hosts. It is a monitoring agent that reads host system metrics collected by Prometheus. Prometheus also collects metrics from Kafka brokers, ZooKeeper nodes, Kafka MirrorMaker, Kafka Bridge, Kafka Exporter, and Cruise Control. These metrics are available in the Prometheus format. Some components specified in additional configuration settings. 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 ADS metrics. To do so, configure its ADS monitoring targets as described in the Metrics in the Prometheus format section. See then on the Viewing metrics page.
- You can also use the Federation mechanism to migrate all metrics from the Prometheus server deployed in ADS to your Prometheus.

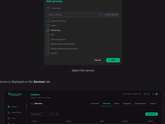
Step 1. Add the service

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



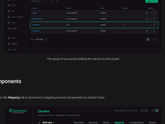
Switch to adding services

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



Select the service

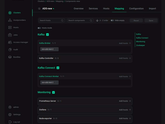
As a result, the added service is displayed on the **Services** list.



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.



Switch to mapping service components

Monitoring service components

Component	Description
Prometheus Server	Deploys a Prometheus server that serves as a: • proxy interface for all collections of metrics as a host, • storage for all metrics of a cluster, • generator of alerts based on collected metrics.
Grafana	Visualizes ADS metrics as graphs and charts organized into dashboards
Node exporter	Exports ADS host metrics to Prometheus

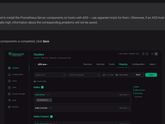
2. Assign a host to each component of the Monitoring service – click **Add hosts** and select the desired host in the pop-up window.

Select a host for a component

CAUTION

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

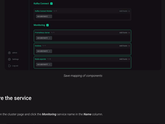
3. After the distribution of components is completed, click **Save**.



Save mapping of components

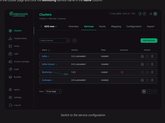
Step 3. Configure the service

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



Switch to the service configuration

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

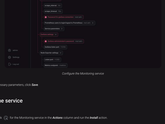


Configure the Monitoring service

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

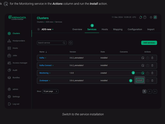
Step 4. Install the service

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



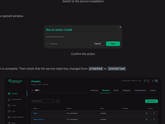
Switch to the service installation

2. Confirm the action in the opened window.



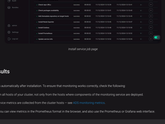
Confirm the action

3. Wait until the installation is complete. Then check that the service state has changed from **installed** to **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.



Install service job page

Step 5. View results

The Monitoring service starts automatically after installation. To ensure that monitoring works correctly, check the following:

- Metrics are collected from all hosts of your cluster, not only from the hosts where components of the monitoring service are deployed.
- Both systems and ADS service metrics are collected from the cluster hosts – see ADS monitoring metrics.

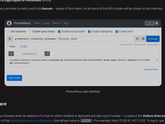
To check both these points, you can view metrics in the Prometheus format in the browser, and also use the Prometheus or Grafana web interface.

CAUTION

To enable Kafka monitoring, on the Kafka service Configuration page, set the **JMX Monitoring** switch to the active position, then restart the service.

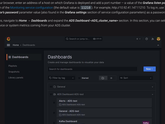
Metrics in the Prometheus format

1. In the address bar of the browser, enter the ADS cluster host address, and also specify the port and endpoint of access to the system metrics, which are defined on the **Services** > **Monitoring** > **Configuration** page in the **Prometheus settings** group of the Monitoring service configuration. For example:
An imperfect address for viewing ADS system metrics: **http://10.92.1.67:9101**. Here **10.92.1.67** – FQDN of the ADS host.
2. The page that opens will display monitoring metrics from the specified host of the ADS cluster in the Prometheus format.



Prometheus web interface

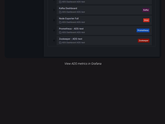
1. In the address bar of your browser, enter the IP address of the host where the Prometheus Server component of the Monitoring service is installed. Specify the port number, which is determined by the **web.listen-address** parameter in the **Prometheus settings** group of the Monitoring service configuration. The default port is **9090**. For example: **http://10.92.1.67:9090**. Here **10.92.1.67** – FQDN of the Prometheus Server host, and the parameter value that we also specified in the **Prometheus settings** section.
2. Click **Prometheus** (see the right figure in Prometheus Web).
3. In the **Expression** field, you can enter a metric and click **Execute** – values of this metric on all hosts of the ADS cluster will be shown in the interface.



Prometheus web interface

Grafana web interface

1. Enter the IP address of your previous server address of a server which Grafana is deployed and port number 3000. For example: **http://10.92.1.67:3000**. Here **10.92.1.67** – FQDN of the Grafana server, and **3000** – the default port number. If you changed the port in the **Grafana settings** section of service configuration parameters (in the **Grafana administrator's password** parameter value input field) or in the **Grafana settings** section of service configuration parameters, be sure to specify it.
2. In the window that opens, navigate to **Home** > **Dashboards** and expand the **ADS Dashboard - ADS_cluster_name** section. In this section, you can select one of available dashboards to view system metrics coming from your ADS cluster.



View ADS metrics in Grafana

Install monitoring

Daria Barysheva

To Table of Contents

Monitoring is installed optionally (yet strongly recommended). ADCM monitoring is based on three services: Diamond, Graphite, and Grafana.

For **offline** installation, there is no need for a separate monitoring cluster, as the Enterprise Tools (ET) cluster plays its role. The steps for installing monitoring in this case are listed below. You can perform them both before and after the ADS cluster installation, except the last step Integrate with ADS cluster, that can be executed only when the ADS cluster already exists.

- Configure Enterprise Tools cluster
- Integrate with ADS cluster

CAUTION

Starting with 3.6.2.2.b1, it is not recommended to configure ADS monitoring using a monitoring cluster, as support for this functionality is currently suspended and will be completely discontinued in future ADS releases. The recommended way to monitor an ADS cluster is to use the Monitoring service.

Configure Enterprise Tools cluster

© Axon Ada Mentis

Contents

To Table of Contents

Step 1. Add monitoring services to the ET cluster
Step 2. Map monitoring components
Step 3. Configure monitoring services
Step 4. Install monitoring services
Step 5. Verify results
- Graphite
- Grafana

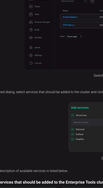
IMPORTANT

Before the subsequent monitoring configuration, make sure that the Enterprise Tools (ET) cluster is installed via ADCM. All the following actions are performed on its base.

Step 1. Add monitoring services to the ET cluster

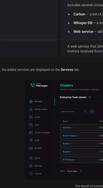
The steps for adding services to a cluster are listed below:

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



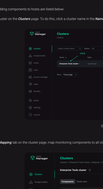
Select a cluster

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



Switch to adding services

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



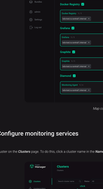
Select services

The brief description of available services is listed below.

Monitoring services that should be added to the Enterprise Tools cluster

Service	Purpose
Diamond	A Python service used for collecting system metrics from the hosts that are connected to the monitoring system. The collected metrics are processed by Carbon (the backend services of Graphite).
Graphite	Includes several components: • **Carbon** – a pair of services that are used to process metrics collected by Diamond; • **Whisper DB** – a temporary file database where collected metrics are stored; • **Web service** – allows to get and visualize metrics from Whisper DB.
Grafana	A web service that allows users to create convenient sets of charts, alerts, and other visualization tools for metrics received from Graphite. By default, uses the port **3000**.

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

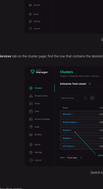


The result of successful adding services to a cluster

Step 2. Map monitoring components

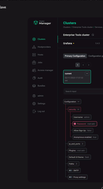
The steps for adding components to hosts are listed below:

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



Select a cluster

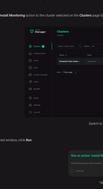
2. Open the **Mapping** tab on the cluster page, map monitoring components to all cluster hosts, and click **Save**.



Map components to hosts

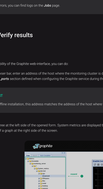
Step 3. Configure monitoring services

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



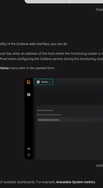
Select a cluster

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



Switch to configuring service

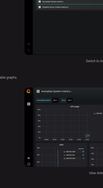
3. In the window that opens:
   - Open the **Primary Configuration** tab.
   - Switch on the **Show advanced** toggle.
   - Fill in configuration parameters of the selected service. The fields highlighted in red are mandatory.
   - Click **Save**.



Configure a service

Step 4. Install monitoring services

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



Switch to monitoring installation

2. In the opened window, click **Run**

Verify installation

3. Wait until the installation process is completed.

NOTE

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

Step 5. Verify results

Graphite

To check availability of the Graphite web interface, you can do:

1. In the browser bar, enter an address of the host where the monitoring cluster is deployed. To specify the port number, use the value of the **Web interface TCP port** parameter from the **gr_and_perrs** section defined when configuring the Graphite service during the monitoring cluster installation. The value **80** is used by default can be omitted.

IMPORTANT

In case of offline installation, this address matches the address of the host where the Enterprise Tools cluster is deployed.

2. Expand a tree at the left side of the opened form. System metrics are displayed there grouped by hosts. You can select some of these metrics to view their change dynamics in the form of a graph at the right side of the screen.

Graphite web interface

Grafana

To check availability of the Grafana web interface, you can do:

1. In the browser bar, enter an address of the host where the monitoring cluster is deployed. To specify the port number, use the value of the **Port** parameter from the **gr_and_perrs** section defined when configuring the Graphite service during the monitoring cluster installation. By default, **3000**.
2. Select the **Home** menu item in the opened form.

Grafana web interface

3. Open one of available dashboards. For example, **Anmadata System metrics**

Switch to Anmadata System metrics

4. View available graphs.

View Anmadata System metrics

Integrate with ADS cluster

By Tran Anh Tuan, Vinh Nguyễn

Contents

To Table of Contents

Step 1. Add a monitoring service

Step 2. Map monitoring components

Step 3. Import monitoring settings

Step 4. Install a monitoring service

Step 5. Verify results

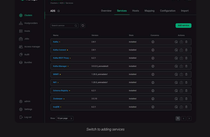
- Graphite
- Grafana

In order to determine that a new cluster should be monitored, it is necessary to add a separate service Monitoring Alarms and integrate it with the previously created ET cluster. The steps for building integration between an ET cluster and ADS cluster are described below. All actions should be performed on the base of ADS cluster.

Step 1. Add a monitoring service

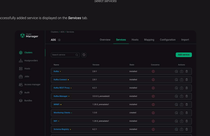
The steps for adding services to a cluster are listed below:

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



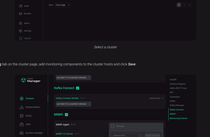
Select a cluster

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



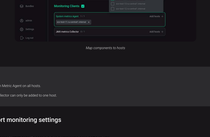
Switch to adding services

3. Select the service Monitoring Clients in the opened dialog and click **Add**



Select services

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

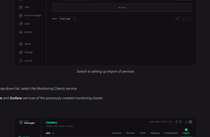


The result of successfully adding a service to the cluster

Step 2. Map monitoring components

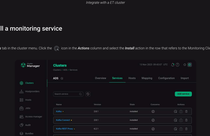
The steps for adding components to hosts are listed below:

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, add monitoring components to the cluster hosts and click **Save**

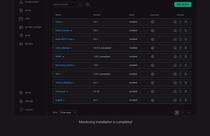


Map components to hosts

> **IMPORTANT!**
>
> - Add the System Metric Agent on all hosts.
> - JMX metric Collector can only be added to one host.

Step 3. Import monitoring settings

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

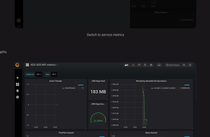


Select a cluster

2. Open the **Import** tab on the cluster page and go to the **Services** tab.

Switch to setting up import of services

3. In the **Import to** drop-down list, select the Monitoring Clients service.
4. Select the **Grafana** and **Grafana** services of the previously created monitoring cluster.
5. Click **Import**



Integrate with a ET cluster

Step 4. Install a monitoring service

1. Open the **Services** tab in the cluster menu. Click the ⚙ icon in the **Actions** column and select the **Install** action in the row that refers to the Monitoring Clients.

Switch to installing the monitoring service

2. In the dialog box that opens, select required installation parameters and click **Run**.
3. Verify the action in the opened window.

Verify the action

4. Wait until the job is completed. After successful installation, the monitoring service changes its state from `created` → `installed`.

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

Monitoring installation is completed

Step 5. Verify results

To ensure that installation process is completed successfully, check the following:

- System metrics are collected from the hosts of your cluster, not only from the host where the monitoring cluster has been deployed earlier.
- Along with system metrics, service metrics are collected from the hosts of your cluster. For example, ADS cluster provides metrics for such services as Kafka, Yarn, MAHA.

You can check it both in the Graphite and Grafana web interface.

Graphite

1. In the browser bar, enter an address of the host where the monitoring cluster is deployed. To specify the port number, use the value of the **Web interface TCP port** parameter from the **ip_and_ports** section defined when configuring the Graphite service during the monitoring cluster installation. The value `2880` is used by default) can be omitted.
2. Expand the menu at the left side of the opened form. System and service metrics are displayed there, grouped by hosts. You can select some of these metrics to view their change dynamics in the form of a graph at the right side of the screen.

New hosts and service metrics are available in Graphite

Grafana

1. In the browser bar, enter an address of the host where the monitoring cluster is deployed. To specify the port number, use the value of the **Port** parameter from the **ip_and_ports** section defined when configuring the Grafana service during the monitoring cluster installation. By default, `3000`.
2. Select the **Home** menu item in the opened form.
3. Notice that new dashboards are available. Select one of them to view the service metrics that are collected from your cluster.

Switch to service metrics

4. View available graphs.

View service metrics

Quick start with Kafka

Olga Semmel

Contents

To Table of Contents

Install and run a Kafka environment via ADCM

Connect to Kafka

Create a topic

Write a message to a topic

Read messages from a topic

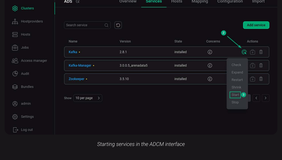
- Read from the beginning of the partition
- Read from the given offset

Terminate the Kafka environment

This article describes the first steps to get started with Kafka.

Install and run a Kafka environment via ADCM

1. To run a Kafka environment using ADCM in an AGS cluster, install the following services:
   - Kafka
   - ZooKeeper
2. After installation in the ADCM interface for all installed services, apply the ***Start*** action by clicking the icon in the ***Actions*** column one by one and wait for the launch to complete successfully.



Starting services in the ADCM interface

As a result, the Kafka environment is installed on the hosts, and the Kafka broker and ZooKeeper servers are running.

NOTE

Installation of services is described in the articles Online installation, Offline installation.

Connect to Kafka

When installing Kafka on hosts, a new directory */usr/lib/kafka/bin* is created. It contains *.sh* files, which are scripts for executing commands in the Kafka environment. To view the directory, run the following command:

```
$ ls -la /usr/lib/kafka/bin
```

Working with the Kafka service occurs by running the appropriate scripts on the command line. Typically, scripts are launched from the */usr/lib/kafka* folder, specifying the *bin/* part of the path in the command. To switch to the desired directory run the command:

```
$ cd /usr/lib/kafka
```

Create a topic

In order to write messages to Kafka, it is necessary to create a topic. This is done using the *kafka-topics.sh* script and the `--create` option from any cluster host that has a Kafka service installed.

To create a topic, run the following command:

```
$ bin/kafka-topics.sh --create --topic <topic_name> --bootstrap-server <hostname>:9092
```

where (in this command and below):

- `<topic_name>` – topic name;
- `<hostname>` – hostname where the topic is created;
- `9092` – HTTP port for accessing the service Kafka.

As a result of the script execution, the following message appears:

```
Created topic <topic_name>.
```

NOTE

The topic name can include alphanumeric characters (including ASCII), as well as the characters `.`, `_`, and `-`.

In order to view data about an existing topic, the *kafka-topics.sh* script and the `--describe` option are used.

To view data about topic, run the following command:

```
$ bin/kafka-topics.sh --describe --topic <topic_name> --bootstrap-server <hostname>:9092
```

As a result, the data about the requested topic appears in the form:

```
Topic: <topic_name>       TopicId: 5TGaa80QSF64a5bkMRtp-YA       PartitionCount: 1       ReplicationFactor: 1    Configs: unclean.
leader-election.enable=false
Topic: <topic_name>       Partition: 0    Leader: 1001    Replicas: 1001  Isr: 1001
```

It indicates that the created topic has the following parameters:

- number of partitions – `1`;
- replication factor – `1`;
- broker leader designation for this partition – `1001`;
- `unclean.leader.election.enable` parameter is set to false.

NOTE

Other actions that the *kafka-topics.sh* script performs can be seen in the list that appears after the command is executed:

```
$ bin/kafka-topics.sh
```

Write a message to a topic

Writing a message to a topic is done using the *kafka-console-producer.sh* script from any cluster host on which the Kafka service is installed.

TIP

Topic creation occurs automatically when using the *kafka-console-producer.sh* script, if you enable the ***auto.create.topics.enable*** parameter in the ***server.properties*** group when configuring the Kafka service.

To write a message to the `<topic_name>` topic:

1. Run the following command:

```
$ bin/kafka-console-producer.sh --topic <topic_name> --bootstrap-server <hostname>:9092
```

   This starts the message recording mode.

2. On the next line after entering the command enter the required messages, each message on a new line. Writing a message and moving to a new line is done with `Enter`. For example, several messages are entered in this way, each on its own line:

```
>Sunday
>Monday
>Tuesday
>Wednesday
>Thursday
>Friday
>Saturday
```

3. Exit the message recording mode, for this do the following:
   - after writing the last message go to the next line;
   - press `Ctrl+C`.

NOTE

To view other options that can be applied when executing the *kafka-console-producer.sh* script, run the command:

```
$ bin/kafka-console-producer.sh
```

Read messages from a topic

Reading messages from a topic is done using the *kafka-console-consumer.sh* script from any cluster host on which the Kafka service is installed.

Read from the beginning of the partition

To read messages from the `<topic_name>` topic starting from the very first message written to the partition run the following command with the `--from-beginning` option:

```
$ bin/kafka-console-consumer.sh --topic <topic_name> --from-beginning --bootstrap-server <hostname>:9092
```

This starts the message reading mode. Messages are read in the same order they are written:

```
Sunday
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
```

To exit from the message reading mode, press `Ctrl+C`.

TIP

If you write messages to the same topic using *kafka-console-producer.sh* from another terminal session, you can immediately read them in read mode.

Read from the given offset

To read messages from the `<topic_name>` topic starting at offset 4 (`Thursday`), use the options to specify the partition `--partition 0` and the offset `--offset 4`:

```
$ bin/kafka-console-consumer.sh --topic <topic_name> --partition 0 --offset 4 --bootstrap-server <hostname>:9092
```

The result contains messages written starting from the offset 4:

```
Thursday
Friday
Saturday
```

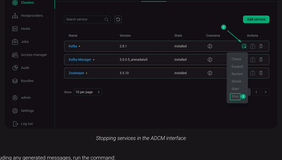
NOTE

To view other options that can be applied when executing the *kafka-console-consumer.sh* script, you need to run the command:

```
$ bin/kafka-console-consumer.sh
```

Terminate the Kafka environment

To stop services via the ADCM interface, apply the ***Stop*** action by clicking the icon in the ***Actions*** column one by one and wait for the action to complete successfully.



Stopping services in the ADCM interface

To delete local Kafka data including any generated messages, run the command:

```
$ rm -rf /tmp/kafka-logs /tmp/zookeeper
```

Use SASL PLAINTEXT in Kafka

Olga Semina

Contents

To Table of Contents

Check the installed SASL PLAINTEXT

Manage users with SASL PLAINTEXT authentication in Kafka

- Create a user
- Delete and change user data

User authentication in the Kafka environment

Work with .sh files (scripts) with participation of the created user

NOTE

Enabling SASL PLAINTEXT authentication is done in accordance with the article SASL PLAINTEXT.

Check the installed SASL PLAINTEXT

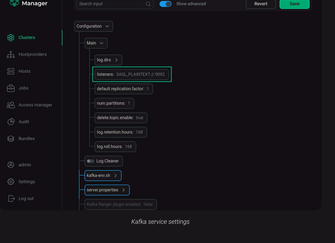
1. Check the security and authentication configuration of the Kafka brokers.
   On each host with a Kafka broker, issue the command:

```
$ vim /etc/kafka/conf/server.properties
```

   Verify that in the *server.properties* file for each Kafka broker, the lines defining the security protocol are changed to `SASL_PLAINTEXT`, strings defining the authentication mechanism – to `PLAIN`, and also passwords for all ADS cluster services are generated:

```
security.inter.broker.protocol=SASL_PLAINTEXT


sasl.mechanism.inter.broker.protocol=PLAIN
sasl.enabled.mechanisms=PLAIN

listener.name.sasl_plaintext.plain.sasl.jaas.config=org.apache.kafka.common$
  username=kafka \
  password="kafka-password" \
  user_kafka="kafka-password" \
  user_ksql-server="ksql-server-password" \
  user_kafka-rest="kafka-rest-password" \
  user_schema-registry="schema-registry-password" \
  user_kafka-connect="kafka-connect-password";
```

2. Verify that after installing SASL PLAINTEXT, the ***listeners*** parameter in the Kafka service settings has changed from `PLAINTEXT://:9092` to `SASL_PLAINTEXT://:9092`.



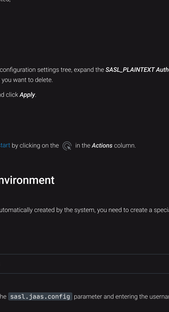
Kafka service settings

Manage users with SASL PLAINTEXT authentication in Kafka

Create a user

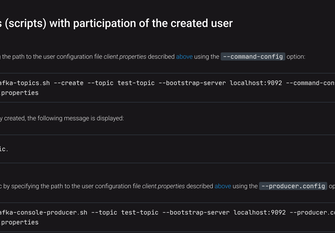
In order to create a user, do the following:

1. On the cluster configuration tab in the configuration settings tree, expand the ***SASL_PLAINTEXT Authentication*** node, then the ***SASL_PLAINTEXT Auth username.password map*** node and click ***+ Add property*** to create a new user:



Creating a new user

2. In the window that opens, enter the user name (field name), password (secret), confirm the password, and click ***Apply***.
   The username must contain lowercase letters, numbers, dots, underscores, and/or hyphens.



Name and password for the new user

3. Save the configuration by clicking ***Save***.
4. Restart the Kafka service. To do this, apply the action Restart by clicking on the ↻ in the ***Actions*** column.
5. Make sure the *server.properties* file for each Kafka broker has an entry for the new user:

```
listener.name.sasl_plaintext.plain.sasl.jaas.config=org.apache.kafka.common$
  username=kafka \
  password="kafka-password" \
  user_kafka="kafka-password" \
  user_ksql-server="ksql-server-password" \
  user_kafka-rest="kafka-rest-password" \
  user_schema-registry="schema-registry-password" \
  user_kafka-connect="kafka-connect-password" \
  user_new_user="password";
```

   where:

   - `user1` – the name entered when the user was created;
   - `password` – the given password for the user.

Delete and change user data

In order to delete a user, on the cluster configuration tab in the configuration settings tree, expand the ***SASL_PLAINTEXT Authentication*** node, then the ***SASL_PLAINTEXT Auth username.password map*** node and click 🗑 opposite the user you want to delete.

To change user data, select the desired user, enter new data and click ***Apply***.

After deleting a user or changing data:

1. Save the configuration by clicking ***Save***.
2. Restart the Kafka service. To do this, apply the action Restart by clicking on the ↻ in the ***Actions*** column.

User authentication in the Kafka environment

To authenticate each user, including the `kafka` user that is automatically created by the system, you need to create a special configuration file *client.properties*. For this you need to do:

1. Open the stub file by running the command:

```
$ sudo vim /etc/kafka/conf/client.properties
```

2. Make changes to the file by uncommenting the line with the `sasl.jaas.config` parameter and entering the username and password of the user as follows:

```
security.protocol=SASL_PLAINTEXT
sasl.mechanism=PLAIN
# Uncomment and set necessary username/password
sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required \
 username="user1" \
 password="password";
```

   where the username and password must be enclosed in quotation marks, as shown in the example.

Work with .sh files (scripts) with participation of the created user

1. Create a topic by specifying the path to the user configuration file *client.properties* described above using the `--command-config` option:

```
$ /usr/lib/kafka/bin/kafka-topics.sh --create --topic test-topic --bootstrap-server localhost:9092 --command-config
/etc/kafka/conf/client.properties
```

   If the topic was successfully created, the following message is displayed:

```
Created topic test-topic.
```

2. Write messages to the topic by specifying the path to the user configuration file *client.properties* described above using the `--producer.config` option:

```
$ /usr/lib/kafka/bin/kafka-console-producer.sh --topic test-topic --bootstrap-server localhost:9092 --producer.config
/etc/kafka/conf/client.properties
```

3. Read messages from a topic by specifying the path to the user configuration file *client.properties* described above using the `--consumer.config` option:

```
$ /usr/lib/kafka/bin/kafka-console-consumer.sh --topic test-topic --from-beginning  --bootstrap-server sov-ads-test-4.ru-central1.
internal:9092 --consumer.config /etc/kafka/conf/client.properties
```

   Verify the read messages are correct.

An attempt to run any script as a user that is not included in the *server.properties* file causes an authentication error, for example:

```
[2022-12-01 13:42:25,572] ERROR [AdminClient clientId=adminclient-1] Connection to node -1 (sov-ads-test-4.ru-central1.
internal/10.92.16.223:9092) failed authentication due to: Authentication failed: Invalid username or password (org.apache.kafka.
clients.NetworkClient)
[2022-12-01 13:42:25,574] WARN [AdminClient clientId=adminclient-1] Metadata update failed due to authentication error (org.
apache.kafka.clients.admin.internals.AdminMetadataManager)
org.apache.kafka.common.errors.SaslAuthenticationException: Authentication failed: Invalid username or password
Error while executing topic command : Authentication failed: Invalid username or password
[2022-12-01 13:42:25,579] ERROR org.apache.kafka.common.errors.SaslAuthenticationException: Authentication failed: Invalid
username or password
```

Use MIT Kerberos in Kafka

Olga Semina

Contents

To Table of Contents

Check installed Kerberos SASL

Set up accounts for authentication in Kafka

Create a JAAS file for a user

Create a configuration file .properties for the user

Connect to Kafka (create tickets) and work with .sh files (scripts) with the participation of different users

This article describes the first steps in Kafka with Kerberos SASL installed.

NOTE

Enabling Kerberos SASL authentication is done in accordance with the article MIT Kerberos.

Check installed Kerberos SASL

1. Check the settings set for Kerberos in the cluster configurations:
   Go to the cluster configuration, select ***Show advanced*** find the section with Kerberos configurations, expand it.

Installed ADS cluster configuration settings for Kerberos

2. Check the security and authentication configuration of the Kafka brokers.
   On each host with a Kafka broker, issue the command:

```
$ sudo vim /usr/lib/kafka/config/server.properties
```

   Ensure that in the *server.properties* file for each Kafka broker, the lines defining the *security protocol* are changed to `SASL_PLAINTEXT`, lines defining the authentication mechanism - to `GSSAPI`:

```
security.inter.broker.protocol=SASL_PLAINTEXT


sasl.mechanism.inter.broker.protocol=GSSAPI
sasl.enabled.mechanisms=GSSAPI
```

3. Verify that after installing Kerberos SASL in the Kafka service settings in the ***server.properties*** group, the ***listeners*** parameter has changed from `PLAINTEXT://:9092` to `SASL_PLAINTEXT://:9092`.
4. Check for the created principals for each service.
   On the host with the KDC server installed, enter the command:

```
$ sudo kadmin.local list_principals
```

   The output of the list of principals shows that there are principals for each host of each installed service, principals for admins and other technical principals for the created realm:

```
K/M@ADS-KAFKA.LOCAL
admin/admin@ADS-KAFKA.LOCAL
kadmin/admin@ADS-KAFKA.LOCAL
kadmin/changepw@ADS-KAFKA.LOCAL
kadmin/sov-ads-test-0.ru-central1.internal@ADS-KAFKA.LOCAL
kafka/sov-ads1-test.ru-central1.internal@ADS-KAFKA.LOCAL
kafka/sov-ads2-test.ru-central1.internal@ADS-KAFKA.LOCAL
kafka/sov-ads3-test.ru-central1.internal@ADS-KAFKA.LOCAL
kiprop/sov-ads-test-0.ru-central1.internal@ADS-KAFKA.LOCAL
krbtgt/ADS-KAFKA.LOCAL@ADS-KAFKA.LOCAL
zookeeper/sov-ads1-test.ru-central1.internal@ADS-KAFKA.LOCAL
zookeeper/sov-ads2-test.ru-central1.internal@ADS-KAFKA.LOCAL
zookeeper/sov-ads3-test.ru-central1.internal@ADS-KAFKA.LOCA
```

5. Check for the presence on hosts with installed services of files for storing passwords *.service.keytab*.
   On each host with installed services, issue the command:

```
$ ls -la /etc/security/keytabs/
```

   The file listing shows that *.service.keytab* files have been created for each service installed on the host:

```
total 16
drwxr-xr-x. 2 root      root       102 Aug  9 20:55 .
drwxr-xr-x. 7 root      root       4096 Aug  9 19:38 ..
-rw-------. 1 kafka     kafka      824 Aug  9 20:54 kafka.service.keytab
-rw-------. 1 zookeeper  zookeeper  858 Aug  9 20:55 zookeeper.service.keytab
```

Set up accounts for authentication in Kafka

For the producer and the consumer you can create the user principals.

To do this, enter the commands one by one using the username and password:

```
$ sudo kadmin.local -q "add_principal -pw PASSWORD reader@ADS-KAFKA.LOCAL"
$ sudo kadmin.local -q "add_principal -pw PASSWORD writer@ADS-KAFKA.LOCAL"
```

Get the result for each created account:

```
Principal "reader@ADS-KAFKA.LOCAL" created.
Principal "writer@ADS-KAFKA.LOCAL" created.
```

Create a JAAS file for a user

A JAAS file (Java Authentication and Authorization Service) must be created for all principals. It specifies how tickets for a particular principal will be used.

NOTE

- A description of the JAAS file and the options used are given in the article krb5.conf/krb5.kdc.
- Client connection data can also be specified in the `KafkaClient` section of the *kafka.jaas.conf* file when using JAAS file template for the Kafka service.

For broker principals, connection parameters are automatically created in the *kafka.jaas.conf* file after kerberization. To view the contents of the file, enter the following command:

```
$ sudo vim /usr/lib/kafka/config/kafka-jaas.conf
```

For the new `writer` and `reader` principals, you must create the JAAS file yourself.

1. Run command:

```
$ sudo vim /tmp/client.jaas
```

2. Write data to the file:

```
KafkaClient {
com.sun.security.auth.module.Krb5LoginModule required
useTicketCache=true;
};
```

   where `useTicketCache` is a parameter specifying whether a ticket for this user will be obtained from the ticket cache. If you set this parameter to `true`, you must create a user ticket before connecting to Kafka.

Create a configuration file .properties for the user

To create a configuration file .properties for the user, run the command:

```
$ sudo vim /tmp/client.properties
```

Fill the file with data:

```
security.protocol=SASL_PLAINTEXT
sasl.mechanism=GSSAPI
sasl.kerberos.service.name=kafka
```

Connect to Kafka (create tickets) and work with .sh files (scripts) with the participation of different users

1. Open terminal session 1 and connect to one of the Kafka brokers.
2. Create a ticket for user `writer`:

```
$ kinit -p writer@ADS-KAFKA.LOCAL
```

   Enter the password specified when creating the user.

3. Check ticket:

```
$ klist
```

```
Ticket cache: FILE:/tmp/krb5cc_1000
Default principal: writer@ADS-KAFKA.LOCAL

Valid starting     Expires            Service principal
08/10/2022 20:44:12  08/11/2022 20:44:12  krbtgt/ADS-KAFKA.LOCAL@ADS-KAFKA.LOCAL
```

NOTE

For a complete description of the *kinit* command functions and applicable options, see kinit.

4. 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"
```

5. Create a topic by specifying the path to the created *client.properties* file:

```
$ /usr/lib/kafka/bin/kafka-topics.sh --create --topic test-topic --bootstrap-server sov-ads-test-1.ru-central1.internal:9092,sov-ads-test-2.ru-central1.internal:9092,sov-ads-test-3.ru-central1.internal:9092 --command-config /tmp/client.properties
```

   Get a confirmation:

```
Created topic test-topic.
```

6. Write a message to the topic, specifying the path to the created *client.properties* file:

```
$ /usr/lib/kafka/bin/kafka-console-producer.sh --topic test-topic --bootstrap-server sov-ads-test-1.ru-central1.internal:9092,sov-ads-test-2.ru-central1.internal:9092,sov-ads-test-3.ru-central1.internal:9092 --producer.config /tmp/client.properties
```

```
>One
>Two
>Three
>Four
>Five
```

7. Open terminal session 2 and connect to one of the Kafka brokers.
8. Create a ticket for user `reader`:

```
$ kinit -p reader@ADS-KAFKA.LOCAL
```

   Enter the password specified when creating the user.

9. Check ticket:

```
$ klist
```

```
Ticket cache: FILE:/tmp/krb5cc_1000
Default principal: reader@ADS-KAFKA.LOCAL

Valid starting     Expires            Service principal
08/10/2022 21:30:47  08/11/2022 21:30:47  krbtgt/ADS-KAFKA.LOCAL@ADS-KAFKA.LOCAL
```

10. 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"
```

11. Read messages from a topic by specifying the path to the created *client.properties* file:

```
$ /usr/lib/kafka/bin/kafka-console-consumer.sh --topic test-topic --from-beginning  --bootstrap-server sov-ads-test-1.ru-central1.internal:9092,sov-ads-test-2.ru-central1.internal:9092,sov-ads-test-3.ru-central1.internal:9092 --consumer.config /tmp/client.properties
```

   Messages received:

```
One
Two
Three
Four
Five
```

   Verify that the received messages are correct.

Use Kerberos with MS Active Directory in Kafka

Olga Semina

Contents

To Table of Contents

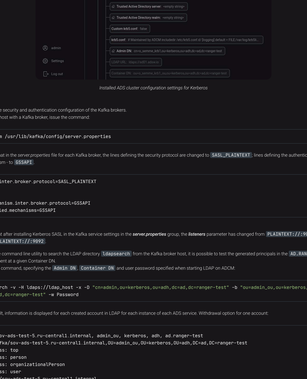
Verify installed Kerberos
Client connection to Kafka
Create a JAAS file for a user
Create a configuration file .properties for the user
Connect a user to Kafka (create tickets) and work with .sh files (scripts)

This article describes the first steps in Kafka after installing Kerberos with MS Active Directory.

- Enabling Kerberos with MS Active Directory authentication is done in accordance with the Kerberos with MS Active Directory article.
- The description given in this article also applies to working in Kafka after enabling the Kerberos with Samba authentication.

Verify installed Kerberos

1. Check the settings set for Kerberos in the cluster configurations.
   Go to the cluster configuration, select **Show advanced** find the section with Kerberos configurations, expand it.



Installed ADS cluster configuration settings for Kerberos

2. Check the security and authentication configuration of the Kafka brokers.
   On each host with a Kafka broker, issue the command

```
$ sudo vim /usr/lib/kafka/config/server.properties
```

   Ensure that in the server.properties file for each Kafka broker, the lines defining the security protocol are changed to `SASL_PLAINTEXT`; lines defining the authentication mechanism - to `GSSAPI`:

```
security.inter.broker.protocol=SASL_PLAINTEXT

sasl.mechanism.inter.broker.protocol=GSSAPI
sasl.enabled.mechanisms=GSSAPI
```

3. Verify that after installing Kerberos SASL in the Kafka service settings in the **server.properties** group, the **listeners** parameter has changed from `PLAINTEXT://9092` to `SASL_PLAINTEXT://9092`.
4. Using the command line utility to search the LDAP directory `ldapsearch` from the Kafka broker host, it is possible to test the generated principals in the `AD.RANGER-TEST` environment at a given Container DN.
   Enter the command, specifying the `Admin DN` **Container DN** and user password specified when starting LDAP on ADCM:

```
$ ldapsearch -v -H ldaps://ldap_host -x -D "cn=admin,ou=kerberos,ou=adh,dc=ad,dc=ranger-test" -b "ou=admin,ou,ou=kerberos,ou=adh,dc=ad,dc=ranger-test" -w Password
```

   As a result, information is displayed for each created account in LDAP for each instance of each ADS service. Withdrawal option for one account:

```
# kafka/sov-ads-test-5.ru-central1.internal, admin_ou, kerberos, adh, ad.ranger-test
dn: cn=kafka/sov-ads-test-5.ru-central1.internal,ou=admin_ou,ou=kerberos,ou=adh,dc=ad,dc=ranger-test
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: user
cn: kafka/sov-ads-test-5.ru-central1.internal
distinguishedName: cn=kafka/sov-ads-test-5.ru-central1.internal,ou=admin_ou,ou=kerberos,ou=adh,dc=ad,dc=ranger-test
instanceType: 4
whenCreated: 20220816180858.0Z
whenChanged: 20220816180942.0Z
uSNCreated: 14958764
uSNChanged: 14958784
name: kafka/sov-ads-test-5.ru-central1.internal
objectGUID:: $/7B9kZcE6aOmxyKU8JMQ==
userAccountControl: 66048
badPwdCount: 0
codePage: 0
countryCode: 0
badPasswordTime: 0
lastLogoff: 0
lastLogon: 133855441424209582
pwdLastSet: 133851176384507312
primaryGroupID: 513
objectSid: AQUAAAAAAAUVAAAAKcXXK3ndbwUK2br9Dv0QAA==
accountExpires: 0
logonCount: 50
sAMAccountName: $1%110-ESD0MALCF3DL
sAMAccountType: 805306368
userPrincipalName: kafka/sov-ads-test-5.ru-central1.internal@AD.RANGER-TEST
servicePrincipalName: kafka/sov-ads-test-5.ru-central1.internal
objectCategory: CN=Person,CN=Schema,CN=Configuration,DC=ad,DC=ranger-test
dSCorePropagationData: 16010101000000.0Z
lastLogonTimestamp: 133851176422839382
```

> **NOTE**
>
> For a complete description of the `ldapsearch` command-line utility features and applicable options, see ldapsearch. The syntax of the `ldapsearch` command may differ depending on the OS used.

5. Check for the presence on hosts with installed services of files for storing passwords *.service.keytab
   On each host with installed services, issue the command

```
$ ls -la /etc/security/keytabs/
```

   The file listing shows that *.service.keytab files have been created for each service installed on the host:

```
total 16
drwxr-xr-x  2 root      root      102 Aug  9 20:55
drwxr-xr-x  7 root      root      4096 Aug  9 19:38
-rw-------  1 kafka     kafka     836 Aug  9 20:54 kafka.service.keytab
-rw-------  1 zookeeper zookeeper  858 Aug  9 20:55 zookeeper.service.keytab
```

Client connection to Kafka

By default, every user in the Active Directory database that has a principal in a given realm has rights to connect to the Kafka cluster and perform actions on topics.

Create a JAAS file for a user

A JAAS file (Java Authentication and Authorization Service) must be created for all principals. It specifies how tickets for a particular principal will be used.

> **NOTE**
>
> - A description of the JAAS file and the options used are given in the article KrbLoginModule.
> - Client connection data can also be specified in the `KafkaClient` section of the kafka-jaas.conf file when using JAAS file template for the Kafka service.

For broker principals, connection parameters are automatically created in the kafka-jaas.conf file after kerberization. To view the contents of the file, enter the following command:

```
$ sudo vim /usr/lib/kafka/config/kafka-jaas.conf
```

For client principals, you need to create the JAAS file yourself:

1. Run command

```
$ sudo vim /tmp/client.jaas
```

2. Write data to the file:

```
KafkaClient {
com.sun.security.auth.module.Krb5LoginModule required
useTicketCache=true;
};
```

   where `useTicketCache` is a parameter specifying whether a ticket for this user will be obtained from the ticket cache. If you set this parameter to `true`, you must create a user ticket before connecting to Kafka.

Create a configuration file .properties for the user

To create a configuration file .properties for the user, run the command:

```
$ sudo vim /tmp/client.properties
```

Fill the file with data:

```
security.protocol=SASL_PLAINTEXT
sasl.mechanism=GSSAPI
sasl.kerberos.service.name=kafka
```

Connect a user to Kafka (create tickets) and work with .sh files (scripts)

1. Open a terminal session and connect to one of the Kafka brokers.
2. Create a ticket for a user by entering a password:

```
$ kinit -p admin-kafka@AD.RANGER-TEST

Password for admin-kafka@AD.RANGER-TEST:
```

> **NOTE**
>
> - In this example, `admin-kafka` is a user with an entry in the Active Directory database and a principal for the `AD.RANGER-TEST` realm.
> - For a complete description of the *kinit* command functions and applicable options, see kinit.

3. Check ticket:

```
$ klist
```

```
Ticket cache: FILE:/tmp/krb5cc_1000
Default principal: admin-kafka@AD.RANGER-TEST

Valid starting     Expires            Service principal
08/17/2022 14:18:31  08/20/2022 00:18:31  krbtgt/AD.RANGER-TEST@AD.RANGER-TEST
        renew until 08/24/2022 14:17:35
```

4. 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"
```

5. Create a topic by specifying the path to the created *client.properties* file:

```
$ /usr/lib/kafka/bin/kafka-topics.sh --create --topic test-topic --bootstrap-server sov-ads-test-1.ru-central1.internal:9092,sov-ads-test-2.ru-central1.internal:9092,sov-ads-test-3.ru-central1.internal:9092 --command-config /tmp/client.properties
```

   Get a confirmation:

```
Created topic test-topic.
```

6. Write a message to the topic, specifying the path to the created *client.properties* file:

```
$ /usr/lib/kafka/bin/kafka-console-producer.sh --topic test-topic --bootstrap-server sov-ads-test-1.ru-central1.internal:9092,sov-ads-test-2.ru-central1.internal:9092,sov-ads-test-3.ru-central1.internal:9092 --producer.config /tmp/client.properties
```

```
>One
>Two
>Three
>Four
>Five
```

7. Open terminal session 2 and connect to one of the Kafka brokers.
8. Create a ticket for user `reader`:

```
$ kinit -p reader@ADS-KAFKA.LOCAL
```

   Enter the password specified when creating the user.
9. Check ticket:

```
$ klist
```

```
Ticket cache: FILE:/tmp/krb5cc_1000
Default principal: reader@ADS-KAFKA.LOCAL

Valid starting     Expires            Service principal
08/18/2022 21:38:47  08/19/2022 21:38:47  krbtgt/ADS-KAFKA.LOCAL@ADS-KAFKA.LOCAL
```

10. 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"
```

11. Read messages from a topic by specifying the path to the created *client.properties* file:

```
$ /usr/lib/kafka/bin/kafka-console-consumer.sh --topic test-topic --from-beginning --bootstrap-server sov-ads-test-1.ru-central1.internal:9092,sov-ads-test-2.ru-central1.internal:9092,sov-ads-test-3.ru-central1.internal:9092 --consumer.config /tmp/client.properties
```

   Messages received:

```
One
Two
Three
Four
Five
```

   Verify that the received messages are correct.

SSL channel security in Kafka

Olga Semme

Contents

To Table of Contents

Check the installed SSL

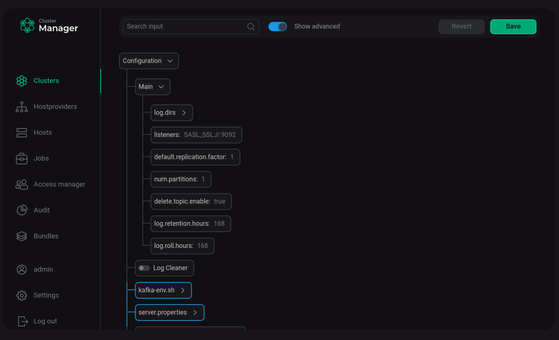
Connect to Kafka (create tickets) and work with .sh files (scripts) with participation of different users after installing SSL

NOTE

Enabling SSL channel security is carried out in accordance with the article SSL channel security.

Check the installed SSL

1. Verify that the ***listeners*** parameter in the Kafka service settings changed from `SASL_PLAINTEXT://:9092` to `SASL_SSL://:9092`.



Kafka service settings

2. Check for a change in the *server.properties* file on each Kafka server.
   Open file *server.properties*:

```
$ sudo vim /usr/lib/kafka/config/server.properties
```

3. Make sure that for each Kafka broker the line defining the security protocol is changed to `SASL_SSL` and there are lines defining the server RSA keystores and passwords for them:

```
security.inter.broker.protocol=SASL_SSL
...
...
...
ssl.keystore.location=/tmp/keystore.jks
ssl.keystore.password=bigdata
ssl.key.password=bigdata
ssl.keystore.type=JKS
ssl.truststore.location=/tmp/truststore.jks
ssl.truststore.password=bigdata
ssl.truststore.type=JKS
```

Connect to Kafka (create tickets) and work with .sh files (scripts) with participation of different users after installing SSL

1. To create a configuration file *.properties* for a user with SSL in mind, run the command:

```
$ sudo vim /tmp/client_ssl.properties
```

   Fill the file with data:

```
security.protocol=SASL_SSL
sasl.mechanism=GSSAPI
sasl.kerberos.service.name=kafka
```

2. Follow the steps to connect to Kafka as user `writer` from the article Use MIT Kerberos in Kafka:
   - Open terminal session 1 and connect to one of the Kafka brokers.
   - Create a ticket for user `writer`.
   - Check ticket.
   - Export the *client.jaas* file as a JVM option for the given user using the `KAFKA_OPTS` environment variable.
3. Create a topic by specifying the path to the created *client_ssl.properties* file:

```
$ /usr/lib/kafka/bin/kafka-topics.sh --create --topic test-topic1 --bootstrap-server sov-ads-test-1.ru-central1.internal:9092,sov-ads-test-2.ru-central1.internal:9092,sov-ads-test-3.ru-central1.internal:9092 --command-config /tmp/client_ssl.properties
```

NOTE

The *client.jaas* file does not need to be changed after installing SSL.

4. Write a message to a topic:

```
$ /usr/lib/kafka/bin/kafka-console-producer.sh --topic test-topic1 --bootstrap-server sov-ads-test-1.ru-central1.internal:9092,sov-ads-test-2.ru-central1.internal:9092,sov-ads-test-3.ru-central1.internal:9092 --producer.config /tmp/client_ssl.properties
```

5. Follow the steps to connect to Kafka as the user `reader` from the article Use MIT Kerberos in Kafka:
   - Open terminal session 2 and connect to one of the Kafka brokers.
   - Create a ticket for user `reader`.
   - Check ticket.
   - Export the generated *client.jaas* file as a JVM option for the given user using the `KAFKA_OPTS` environment variable.
6. Read messages from a topic by specifying the path to the created *client_ssl.properties* file:

```
$ /usr/lib/kafka/bin/kafka-console-consumer.sh --topic test-topic1 --from-beginning --bootstrap-server sov-ads-test-1.ru-central1.internal:9092,sov -ads-test-2.ru-central1.internal:9092,sov-ads-test-3.ru-central1.internal:9092 --consumer.config /tmp/client_ssl.properties
```

   Make sure messages are read correctly.

Ranger Kafka plugin

Olga Semme

Contents

To Table of Contents

Prerequisites

Activate the Ranger Kafka plugin

Check the activated Ranger Kafka Plugin

Prerequisites

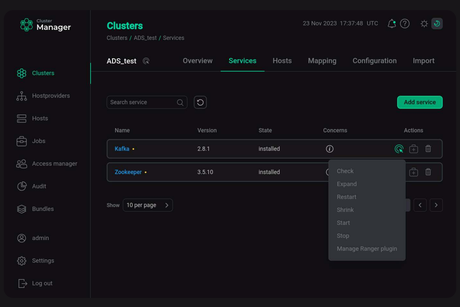
Prerequisites for authorization in Kafka using Ranger:

1. ADS cluster is installed with configured authentication using MIT Kerberos or MS Active Directory protocols.
2. ADPS cluster is installed.
3. ADS cluster integration is enabled on ADPS cluster.

Activate the Ranger Kafka plugin

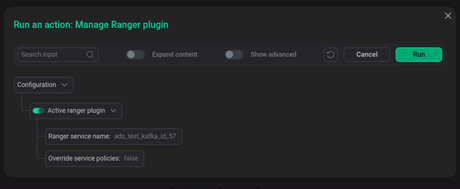
To activate the Ranger Kafka plugin, do the following:

1. Initiate activation of the Ranger Kafka plugin. To do this, apply the ***Manage Ranger plugin*** action by clicking in the ***Actions*** column of the Kafka service.



Activation of the Ranger Kafka plugin

In the ***Run an action: Manage Ranger plugin*** window that opens, activate the ***Active ranger plugin*** switch. If necessary, set the name of the Ranger service that will be added. If a service with the same name already exists, it can be recreated by activating the ***Override service policies*** parameter (`true` — the parameter is active, `false` — inactive), in which case the old service will be deleted and new policies will be generated for this service. If the ***Override service policies*** parameter is inactive, after activating the plugin, a connection to the existing Ranger service will be made.

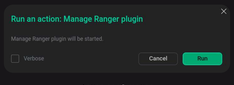


Enabling the Ranger Kafka plugin

NOTE

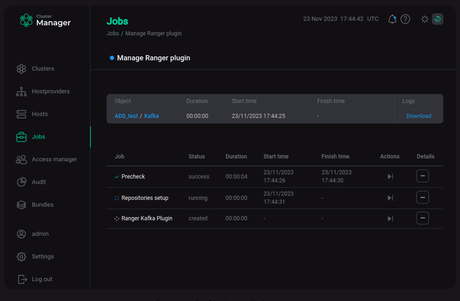
On the first plugin enabling, a service with policies will be created (if it doesn't exist yet) regardless of the ***Override service policies*** parameter.

2. Click ***Run*** and confirm the action in the window that opens.



Action confirmation

3. Wait until the activation process of the Ranger Kafka plugin is completed and the default policy is created on the Ranger side. Analyze and correct errors if they occur.

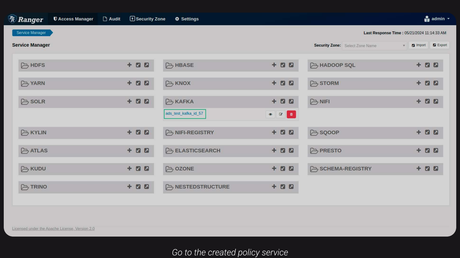


Ranger Kafka Plugin installation process

Check the activated Ranger Kafka Plugin

After successful activation, the authorization policy service for the Kafka service of the selected ADS cluster will appear in the ***Service manager*** interface.

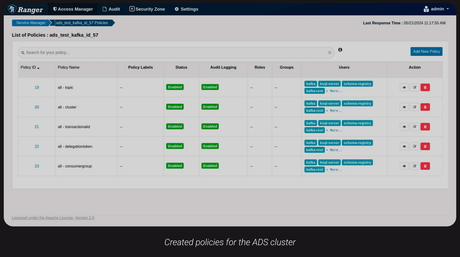
To view the created policies for the cluster, click the name of the service.



Go to the created policy service

This makes available the policies automatically created for the cluster servers.

To view, edit, or delete policies, use the buttons in the ***Action*** column.



Created policies for the ADS cluster

Use Ranger in Kafka

Olga Seleme

Contents

To Table of Contents

Set an authorization policy

How an authorized user works in Kafka

After activating Ranger Kafka plugin, creating a Ranger service, or connecting to an existing service (depending on the value of the Override service policies parameter), all actions in the Kafka service (creating topics, writing messages, reading messages, changing the configuration of topics, etc.) are possible only after setting up an authorization policy for the user.

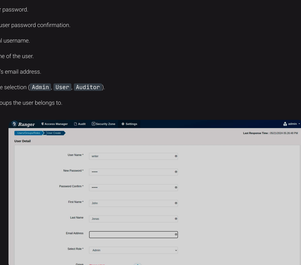
Set an authorization policy

NOTE

- Before configuring authorization for this user, you should create a principal in the MIT Kerberos database or an account in the MS Active Directory database.
- For more details on authorization in Kafka via Ranger, you can refer to Adding KAFKA Policies.
- For more details on working with Ranger, you can refer to the following articles:
  - Manage services in Ranger
  - Manage policies in Ranger

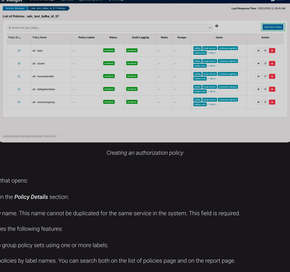
In order to add a new policy for a user to the created Kafka service, perform the following steps:

1. Create a user in Ranger. To do this, in the top menu of the Ranger interface, select ***Settings*** → ***Users/Groups/Roles*** and click ***Add new user***



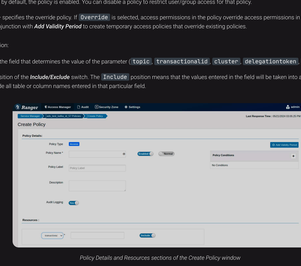
Creating a user in Ranger

2. In the window that opens, fill in the required data for the user:
   - ***User Name*** — username. This name cannot be duplicated in the system. The name must match the username in the KDC or LDAP database. For example, for a user with principal `writer@ADS-KAFKA.LOCAL`, the name in Ranger would be `writer`.
   - ***New Password*** — user password.
   - ***Password Confirm*** — user password confirmation.
   - ***First Name*** — personal username.
   - ***Last Name*** — last name of the user.
   - ***Email Address*** — user's email address.
   - ***Select Role*** — user role selection (`Admin`, `User`, `Auditor`).
   - ***Group*** — the group/groups the user belongs to.



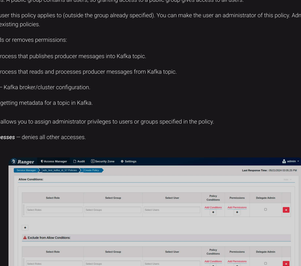
Creating a user in Ranger

3. Click ***Save***.
4. In the ***Service Manager*** window, click on the name of the Kafka service.



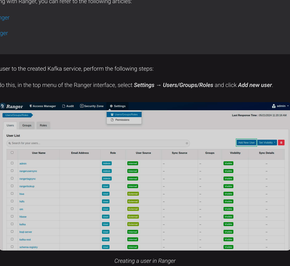
Go to the created policy service

5. In the ***List of Policies*** window that opens, click ***Add new policy*** to add a new policy for the user.



Creating an authorization policy

6. In the ***Create Policy*** window that opens:
   - fill in the required fields in the ***Policy Details*** section:
     - ***Policy Name*** – policy name. This name cannot be duplicated for the same service in the system. This field is required.
     - ***Policy Label*** – provides the following features:
       - Allows the user to group policy sets using one or more labels.
       - User can search policies by label names. You can search both on the list of policies page and on the report page.
       - Helps the user to export/import policies. If the user needs to export a specific set of policies, they can find the policy label and export the specific set of policies.
     - ***Description*** – the purpose of the policy. This field is optional.
     - ***Policy Conditions*** – by clicking **+** under ***Add conditions***, you can add additional conditions.
     - ***Add Validity Period*** – start and end time of the policy.
   - in the ***Policy Details*** section, set the desired switch positions:
     - ***Audit Logging*** – determines whether a specific policy will be audited.
     - ***Enable/Disabled*** – by default, the policy is enabled. You can disable a policy to restrict user/group access for that policy.
     - ***Override/Normal*** – specifies the override policy. If `Override` is selected, access permissions in the policy override access permissions in existing policies. This feature can be used in conjunction with ***Add Validity Period*** to create temporary access policies that override existing policies.
   - in the ***Resources*** section:
     - If necessary, fill in the field that determines the value of the parameter (`topic`, `transactionalid`, `cluster`, `delegationtoken`, `consumergroup`).
     - Set the desired position of the ***Include/Exclude*** switch. The `Include` position means that the values entered in the field will be taken into account. The `Exclude` position will exclude all table or column names entered in that particular field.



Policy Details and Resources sections of the Create Policy window

   - fill in the fields in the ***Allow Condition*** section:
     - ***Select Role*** – the role this policy applies to. A role is a set of permissions. Roles are an easier way to manage a set of permissions based on specific access criteria.
     - ***Select Group*** – the group this policy applies to. To promote a user to an administrator, select the ***Delegate Admin*** check box. Administrators can edit or delete a policy and create child policies. A public group contains all users, so granting access to a public group gives access to all users.
     - ***Select User*** – the user this policy applies to (outside the group already specified). You can make the user an administrator of this policy. Administrators can create child policies based on existing policies.
     - ***Permissions*** – adds or removes permissions:
       - `Publish` — process that publishes producer messages into Kafka topic.
       - `Consume` — process that reads and processes producer messages from Kafka topic.
       - `Configure` – Kafka broker/cluster configuration.
       - `Describe` – getting metadata for a topic in Kafka.
     - ***Delegate Admin*** — allows you to assign administrator privileges to users or groups specified in the policy.
     - ***Deny All Other Accesses*** — denies all other accesses.

The Allow Condition section of the Create Policy window

NOTE

The conditions are met in the order specified in the policy. The condition at the beginning of the list is applied first, then the second, third, and so on.

7. Click ***Add***.

How an authorized user works in Kafka

After activating Ranger and setting the authorization policy for the user, work in Kafka proceeds in accordance with the steps described in the articles:

- Use MIT Kerberos in Kafka – for a user that has a principal in a Kerberos SASL secured environment.
- SSL channel security in Kafka – for a user that has a principal in a Kerberos SASL secured environment with SSL channel security.
- Use Kerberos with MS Active Directory in Kafka – for a user that has an Active Directory account and a corresponding LDAP server SSL certificate.

Any action by an unauthorized user will cause an error. For example, an attempt to write messages to the `new-topic` topic for a user whose authorization policy does not specify the `Publish` permission will result in the following errors:

```
[2022-08-29 10:21:10,894] WARN [Producer clientId=console-producer] Error while fetching metadata with correlation id 3 : {new-topic=TOPIC_AUTHORIZATION_FAILED} (org.apache.kafka.clients.NetworkClient)
[2022-08-29 10:21:10,897] ERROR [Producer clientId=console-producer] Topic authorization failed for topics [new-topic] (org.apache.kafka.clients.Metadata)
[2022-08-29 10:21:10,898] ERROR Error when sending message to topic new-topic with key: null, value: 1 bytes with error: (org.apache.kafka.clients.producer.internals.ErrorLoggingCallback)
org.apache.kafka.common.errors.TopicAuthorizationException: Not authorized to access topics: [new-topic]
```

ACL in Kafka

A. Olga Jarmine

Contents

To Table of Contents

Enable ACL in Kafka

Prepare the authentication environment to create ACLs

- SASL PLAINTEXT
- Kerberos SASL

ACL configuration

- Assign rights and prohibitions to individual actions in Kafka for users
- Assign producer and consumer rights to a user
- View ACLs

How an authorized user works in Kafka

This article describes the basic steps for working with ACL authorization in Kafka.

Access Control List (ACL) — an access control list that defines who can access topics or groups in Kafka and what operations are allowed or denied to be performed by a user or group of users.

Kafka has **AclAuthorizer** — a built-in ACL based authorizer.

ACLs for Kafka are stored in ZooKeeper.

Enable ACL in Kafka

> **NOTE**
>
> Before configuring authorization for an AOS cluster, authentication must be enabled and configured, and users must be created using one of the following protocols:
>
> - SASL PLAINTEXT
> - GRT Kerberos
> - Kerberos with MS Active Directory

To enable and use AclAuthorizer, do the following:

1. On the host with the Kafka broker enter the command:

```
$ sudo vim /usr/lib/kafka/config/server.properties
```

2. Add new parameters to the file:

```
authorizer.class.name=kafka.security.authorizer.AclAuthorizer
allow.everyone.if.no.acl.found=true
```

where:

- `authorizer.class.name` — setting the authorizer class.
- `allow.everyone.if.no.acl.found` is a parameter that determines whether to enable authorization only for topics listed in the ACL.

3. After entering the data, restart the Kafka service on all hosts with the Kafka broker and check the status of the server. To do this, enter the commands one by one:

```
$ sudo systemctl restart kafka
$ sudo systemctl status kafka
```

As a result, information about the status of the Kafka service is displayed.

```
kafka.service - Apache Kafka
   Loaded: loaded (/usr/lib/systemd/system/kafka.service; enabled; vendor preset: disabled)
  Drop-In: /etc/systemd/system/kafka.service.d
           └─override.conf
   Active: active (running) since Wed 2022-10-12 12:48:08 UTC; 52s ago
  Process: 8782 ExecStop=/usr/lib/kafka/bin/kafka stop (code=exited, status=0/SUCCESS)
  Process: 8857 ExecStart=/usr/lib/kafka/bin/kafka start (code=exited, status=0/SUCCESS)
  Process: 8855 ExecStartPre=/bin/chown kafka:kafka /run/kafka (code=exited, status=0/SUCCESS)
  Process: 8851 ExecStartPre=/bin/mkdir -p /run/kafka (code=exited, status=0/SUCCESS)
Main PID: 8860 (java)
   Status: "Running..."
    Memory: 270.6M
    CGroup: /system.slice/kafka.service
            └─8860 java -Xmx1G -Xms1G -server -XX:+UseG1GC -XX:MaxGCPauseMillis=20 -XX:InitiatingHeapOcc...

Oct 12 12:48:49 sov-aos-test-1.ru-central1.internal systemd[1]: Starting Apache Kafka...
Oct 12 12:48:49 sov-aos-test-1.ru-central1.internal systemd[1]: Started Apache Kafka.
```

> **CAUTION**
>
> Restarting the Kafka service using the AOCM user interface tools may result in the set value of `authorizer.class.name` being removed.

Prepare the authentication environment to create ACLs

SASL PLAINTEXT

To create the user environment specified during authentication using the SASL PLAINTEXT protocol, do the following:

1. On the host with the Kafka broker, enter the command:

```
$ sudo vim /etc/kafka/conf/client.properties
```

2. In the file that opens, enable (uncomment) the authentication mechanism and fill in the data for the created user:

```
security.protocol=SASL_PLAINTEXT
sasl.mechanism=PLAIN
# Uncomment and set necessary username/password
#sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required\
#  username="username" \
#  password=password_for_username;
```

For more information about working with users in the SASL PLAINTEXT authentication environment, see the article Use SASL PLAINTEXT in Kafka.

Kerberos SASL

To create the environment for the user specified during authentication using the Kerberos SASL protocol, do the following:

1. Create a JAAS (Java Authentication and Authorization Service) file for the `user` principal that manages the ACLs. Run command:

```
$ sudo vim /tmp/client.jaas
```

2. Write data to file:

```
KafkaClient {
com.sun.security.auth.module.Krb5LoginModule required
useTicketCache=true;
};
```

3. Create a ticket for user `user`:

```
$ kinit -p user@AOS-KAFKA.LOCAL
```

4. Check ticket:

```
$ klist
```

```
Ticket cache: FILE:/tmp/krb5cc_1000
Default principal: user@AOS-KAFKA.LOCAL

Valid starting Expires Service principal
08/10/2022 20:44:12 08/11/2022 20:44:12 krbtgt/AOS-KAFKA.LOCAL@AOS-KAFKA.LOCAL
```

5. 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"
```

For more information about working with users in a Kerberos SASL authentication environment, see the article Use MIT Kerberos in Kafka.

ACL configuration

Working with the ACL-based authorizer occurs by running the kafka-acls.sh script on the command line from any cluster host on which the Kafka service is installed.

> **NOTE**
>
> To view all options for the *kafka-acls.sh* script, run the command:
>
> ```
> $ /usr/lib/kafka/bin/kafka-acls.sh
> ```

Assign rights and prohibitions to individual actions in Kafka for users

Assignment of rights or prohibitions to individual actions to the user occurs by running the *kafka-acls.sh* script with the `--operation` option, and specifying the operation to which access is given.

Permissions are assigned using the `--add` option.

The `--allow-principal` option enables action, and `--deny-principal` disables the action.

Removing a user from the ACL is done with the command option `--remove`.

The main operations for which user rights can be set are:

- `Describe` — view topic or group metadata.
- `DescribeConfigs` — view topic configuration.
- `Alter` — change a topic or a group.
- `IdempotentWrite` — write to a topic for idempotent and transactional APIs.
- `Read` — the right to read from a topic or from a group.
- `Delete` — delete a topic or a group.
- `Create` — create a topic or a group.
- `All` — all rights.
- `Write` — write to a topic or a group.
- `AlterConfigs` — change the topic configuration.

> **NOTE**
>
> For users authenticated using the Kerberos SASL or LDAP protocol, the ACLs use the username without the authentication realm.

In order to allow the `username` user to perform the `WRITE` operation on the `test-topic` topic, on the host with the Kafka broker, enter the command:

```
$ /usr/lib/kafka/bin/kafka-acls.sh --bootstrap-server localhost:9092 --command-config /etc/kafka/conf/client.properties --add --allow-principal User:username --operation WRITE --test-topic
```

In order to prevent the `username1` user from performing the `WRITE` operation on the `test-topic` topic, on the host with the Kafka broker, enter the command:

```
$ /usr/lib/kafka/bin/kafka-acls.sh --bootstrap-server localhost:9092 --command-config /etc/kafka/conf/client.properties --add --deny-principal User:username1 --operation WRITE --test-topic
```

As a result of each addition to the list, a message appears about the successful addition of rights or a ban and an ACL list with current user rights for the topic:

```
Adding ACLs for resource `ResourcePattern(resourceType=TOPIC, name=test-topic, patternType=LITERAL)`:
        (principal=User:username, host=*, operation=WRITE, permissionType=ALLOW)

Current ACLs for resource `ResourcePattern(resourceType=TOPIC, name=test-topic, patternType=LITERAL)`:
        (principal=User:username, host=*, operation=WRITE, permissionType=ALLOW)
        (principal=User:username, host=*, operation=CREATE, permissionType=ALLOW)
        (principal=User:username1, host=*, operation=CREATE, permissionType=DENY)
```

Assign producer and consumer rights to a user

It is possible to assign several producer rights to the user at once — create, write, view metadata (describe).

To assign `producer` rights to the `writter` user for the `test-topic` topic, enter the command on the host with the Kafka broker:

```
$ /usr/lib/kafka/bin/kafka-acls.sh --bootstrap-server localhost:9092 --command-config /etc/kafka/conf/client.properties --add --allow-principal User:writer --producer --topic test-topic
```

As a result, a message appears about the successful addition of `producer` rights and an ACL list with current user rights for the topic:

```
Adding ACLs for resource `ResourcePattern(resourceType=TOPIC, name=test-topic, patternType=LITERAL)`:
        (principal=User:writer, host=*, operation=CREATE, permissionType=ALLOW)
        (principal=User:writer, host=*, operation=WRITE, permissionType=ALLOW)
        (principal=User:writer, host=*, operation=DESCRIBE, permissionType=ALLOW)

Current ACLs for resource `ResourcePattern(resourceType=TOPIC, name=test-topic, patternType=LITERAL)`:
        (principal=User:writer, host=*, operation=WRITE, permissionType=ALLOW)
        (principal=User:writer, host=*, operation=CREATE, permissionType=ALLOW)
        (principal=User:writer, host=*, operation=DESCRIBE, permissionType=ALLOW)
        (principal=User:username, host=*, operation=DESCRIBE, permissionType=ALLOW)
```

It is also possible to assign several consumer rights at once — read, view metadata (describe).

To assign `consumer` rights to the `reader` user for the `test-topic` topic, enter the command on the host with the Kafka broker:

```
$ /usr/lib/kafka/bin/kafka-acls.sh --bootstrap-server localhost:9092 --command-config /etc/kafka/conf/client.properties --add --allow-principal User:reader --consumer --test-topic --group 1
```

> **NOTE**
>
> When assigning consumer rights, you must specify both the topic name (`topic`) and the name of the consumer group (`group`), even if there is only one consumer.

As a result, a message appears about the successful addition of `consumer` rights and an ACL list with current user rights for the `test-topic` topic, including those created earlier for the `writter` user:

```
Adding ACLs for resource `ResourcePattern(resourceType=TOPIC, name=test-topic, patternType=LITERAL)`:
        (principal=User:reader, host=*, operation=DESCRIBE, permissionType=ALLOW)
        (principal=User:reader, host=*, operation=READ, permissionType=ALLOW)

Current ACLs for resource `ResourcePattern(resourceType=TOPIC, name=test-topic, patternType=LITERAL)`:
        (principal=User:writer, host=*, operation=WRITE, permissionType=ALLOW)
        (principal=User:writer, host=*, operation=CREATE, permissionType=ALLOW)
        (principal=User:writer, host=*, operation=DESCRIBE, permissionType=ALLOW)
        (principal=User:username, host=*, operation=DESCRIBE, permissionType=ALLOW)
        (principal=User:reader, host=*, operation=DESCRIBE, permissionType=ALLOW)
        (principal=User:reader, host=*, operation=READ, permissionType=ALLOW)
```

It also displays data about the created rights for the consumer group:

```
Current ACLs for resource `ResourcePattern(resourceType=GROUP, name=1, patternType=LITERAL)`:
        (principal=User:reader, host=*, operation=DESCRIBE, permissionType=ALLOW)
        (principal=User:reader, host=*, operation=READ, permissionType=ALLOW)
```

View ACLs

Viewing ACLs is done with the `--list` option. In order to view the list of permissions for the `test-topic` topic, enter the command:

```
$ /usr/lib/kafka/bin/kafka-acls.sh --bootstrap-server localhost:9092 --command-config /etc/kafka/conf/client.properties --list --topic test-topic
```

The result is a listing of all assigned permissions for the `test-topic` topic:

```
Current ACLs for resource `ResourcePattern(resourceType=TOPIC, name=test-topic, patternType=LITERAL)`:
        (principal=User:writer, host=*, operation=WRITE, permissionType=ALLOW)
        (principal=User:writer, host=*, operation=CREATE, permissionType=ALLOW)
        (principal=User:writer, host=*, operation=DESCRIBE, permissionType=ALLOW)
        (principal=User:username, host=*, operation=DESCRIBE, permissionType=ALLOW)
        (principal=User:reader, host=*, operation=READ, permissionType=ALLOW)
```

To view the list of rights for all topics, use the same command without specifying the topic name.

How an authorized user works in Kafka

After setting up ACLs for users, work in Kafka proceeds according to the steps described in the articles:

- Use MIT Kerberos in Kafka — for a user that has a principal in a Kerberos SASL secured environment.
- SSL channel security in Kafka — for a user that has a principal in a Kerberos SASL secured environment with SSL channel security.
- Use Kerberos with MS Active Directory in Kafka — for a user that has an Active Directory account and a corresponding LDAP server SSL certificate.

Any action by an unauthorized user will cause an error. For example, attempting to write messages to the `test-topic` topic for a user who does not have `WRITE` permission in the ACL for that topic will result in the following errors:

```
[2022-10-12 13:31:57,301] WARN [Producer clientId=console-producer] Error while fetching metadata with correlation id 9 : {test-topic=TOPIC_AUTHORIZATION_FAILED} (org.apache.kafka.clients.NetworkClient)
[2022-10-12 13:31:57,302] ERROR [Producer clientId=console-producer] Topic authorization failed for topics [test-topic] (org.apache.kafka.clients.Metadata)
[2022-10-12 13:31:57,302] ERROR Error when sending message to topic test-topic with key: null, value: 1 bytes with error: (org.apache.kafka.clients.producer.internals.ErrorLoggingCallback)
org.apache.kafka.common.errors.TopicAuthorizationException: Not authorized to access topics: [test-topic]
```

Write a Kafka Streams application

By Paul Ashcroft

Contents

- Prepare to create an application
  - Install OpenJDK
  - Install the latest version of Apache Maven
  - Set environment variables
  - Verify Apache Maven installation
- Create a Kafka Streams project
  - Create a project structure
  - Project structure
  - Configure pom.xml
  - Ready-made Kafka Streams applications
- Run the Kafka Streams application
  - Pipe.java code
  - Run the application
  - Check the work of the created application

To Table of Contents

This article describes how to create a simple streaming application using Kafka Streams.

Kafka Streams is a client library for building applications and microservices, where the input and output data are stored in Kafka clusters.

To create a Kafka Streams application, it is necessary to use its API. It is useful to add the managing and packaging software projects.

In the examples below, the application is created in the Linux environment. It is not much different in the case of Windows.

Prepare to create an application

Note

In this section, all installation commands for [OpenJDK](#) and [Apache Maven](#) are given as reference information for Centos-7 OS. To install software packages on different platforms, please refer to the documentation of the corresponding operating system.

Install OpenJDK

Open a VM console as a sudo user/root and execute a command to install a Kafka Streams project using the OpenJDK library from Red Hat Apache Maven version 3.2 and higher requires JDK version 1.7 or higher to be installed.

Let's check the current version of Java:

```
$ java -version
```

Result:

```
-bash: java: command not found
```

To verify a successful Java installation, run the following command:

```
$ java -version
```

With a successful Java installation, the result looks like this:

```
openjdk version "1.8.0_181"
OpenJDK Runtime Environment (build 1.8.0_181-b13)
OpenJDK 64-Bit Server VM (build 25.181-b13, mixed mode)
```

Install the latest version of Apache Maven

It is recommended to use the latest version of Apache Maven to create new applications.

Check if Apache Maven is installed by running the command:

1. Using the `yum` command, download the latest Apache Maven to the `/opt` directory, selecting the **binary tar.gz** archive in the page of [downloading Apache Maven](#).

```
$ wget http://ftp.byfly.by.edu.by/pub/apache/maven/3.5.4/binary/apache-maven-3.5.4-bin.tar.gz -P /tmp
```

2. Next, unpack the downloaded archive to a prepared directory:

```
$ sudo tar xf /tmp/apache-maven-3.5.4-bin.tar.gz -C /opt
```

3. For easy navigation to Maven directories and updates, it is recommended to create a symbolic link that will point to the Maven installation directory using the command:

```
$ sudo ln -s /opt/apache-maven-3.5.4 /opt/maven
```

Set environment variables

To run the command to call the latest version of Apache Maven, create new applications:

1. Create a new file:

```
$ sudo vim /etc/profile.d/maven.sh
```

2. Fill it equal to with the new folders:

```
export JAVA_HOME=/usr/lib/jvm/jre-openjdk
export M2_HOME=/opt/maven
export MAVEN_HOME=/opt/maven
export PATH=${M2_HOME}/bin:${PATH}
```

3. Save and close the file.

4. Set environment to execute the script:

```
$ sudo chmod +x /etc/profile.d/maven.sh
```

5. Execute the script using the **source** command:

```
$ source /etc/profile.d/maven.sh
```

Verify Apache Maven installation

To verify the installation of Apache Maven, see the result looks like this:

```
$ mvn -version
```

Upon successful installation of Apache Maven, the result looks like this:

```
Apache Maven 3.5.4 (1edded0938998edf8bf061f1ceb3cfdeccf443fb7; 2018-06-17T21:33:14+03:00)
Maven home: /opt/maven
Java version: 1.8.0_181, vendor: Oracle Corporation, runtime: /usr/lib/jvm/jre-1.8.0-openjdk-1.8.0.181-3.b13.el7_5.x86_64/jre
Default locale: en_US, platform encoding: UTF-8
OS name: "linux", version: "3.10.0-862.14.4.el7.x86_64", arch: "amd64", family: "unix"
```

Create a Kafka Streams project

Create a project structure

Apache Maven uses [archetype](#) templates that allow you to create new projects. The [Streams quickstart Java](#) archetype is used to create a Kafka Streams application.

Before creating a project, you need to go to the directory in which the project will be created.

1. Go to `/opt/maven/bin`.

To create a Kafka Streams project structure in the Pipe directory, you must run the [`mvn archetype:generate`](#) command that creates a project from an archetype.

```
$ mvn archetype:generate \
    -DarchetypeGroupId=org.apache.kafka \
    -DarchetypeArtifactId=streams-quickstart-java \
    -DarchetypeVersion=2.0.1 \
    -DgroupId=streams.examples \
    -DartifactId=streaming \
    -Dversion=0.1 \
    -Dpackage=myapps
```

The generation options specified with **-D** are described below:

- **archetypeGroupId** — archetype group id in the repository where the used archetype is located;
- **archetypeArtifactId** — name of the archetype to use;
- **archetypeVersion** — required version of the archetype to use.

CAUTION

The archetype version specified with the **archetypeVersion** parameter must be the latest current software version of the AIDS cluster, as defined in [Supported Versions](#).

- **groupId** — group name of the software specified by the user. This is usually set to the name of the root directory;
- **artifactId** — directories of the contents of the artifact;
- **version** — version from which the project is created (default: 1.0-SNAPSHOT);
- **package** — name of the folder to build the project.

Upon successful creation of the project, the result looks like this:

```
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 14.417s
[INFO] Finished at: Wed Aug 22 11:16:51 UTC 2018
[INFO] Final Memory: 22M/160M
[INFO] ------------------------------------------------------------------------
```

Project structure

The project is automatically created by using the [`pom.xml`](#) build file and the examples:

The main structure of the created project is shown below:

```
$ tree streams-example
streams-quickstart
├── pom.xml
└── src
    ├── main
    │   ├── java
    │   │   ├── myapps
    │   │   │   ├── LineSplit.java
    │   │   │   └── Pipe.java
    │   │   └── streams.examples
    │   └── resources
    │       └── log4j.properties
```

Configure pom.xml

Configure the project using the [`pom.xml`](#) build file, [pom.xml](#) is located, in the AIDS. file containing project information and configuration information used by Maven to build the project.

To create simple Kafka Streams applications, the generated pom.xml is ready to use, and you can also specify other project dependencies, plugins, or targets that you can run, classifiers, distribution descriptors, developers, licenses, mailing lists, etc.

Below is the content of the pom.xml generated by Kafka Streams and the description of the tags used:

```
pom.xml

<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>streams.examples</groupId>
    <artifactId>streaming</artifactId>
    <version>0.1</version>
    <packaging>jar</packaging>

    <name>Kafka Streams Quickstart</name>

    <properties>
        <maven.compiler.source>1.7</maven.compiler.source>
        <maven.compiler.target>1.7</maven.compiler.target>
        <kafka.version>2.0.0</kafka.version>
        <slf4j.version>1.7.7</slf4j.version>
        <log4j.version>1.2.17</log4j.version>
    </properties>

    <repositories>
        <repository>
            <id>apache.snapshots</id>
            <name>Apache Development Snapshot Repository</name>
            <url>https://repository.apache.org/content/repositories/snapshots/</url>
            <releases>
                <enabled>false</enabled>
            </releases>
            <snapshots>
                <enabled>true</enabled>
            </snapshots>
        </repository>
    </repositories>

    <build>
        <plugins>
            <plugin>
                <artifactId>maven-assembly-plugin</artifactId>
                <configuration>
                    <descriptorRefs>
                        <descriptorRef>jar-with-dependencies</descriptorRef>
                    </descriptorRefs>
                    <archive>
                        <manifest>
                            <mainClass></mainClass>
                        </manifest>
                    </archive>
                </configuration>
                <executions>
                    <execution>
                        <id>make-assembly</id>
                        <phase>package</phase>
                        <goals>
                            <goal>single</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>
        </plugins>
    </build>

    <dependencies>
        <dependency>
            <groupId>org.apache.kafka</groupId>
            <artifactId>kafka-streams</artifactId>
            <version>${kafka.version}</version>
        </dependency>
        <dependency>
            <groupId>log4j</groupId>
            <artifactId>log4j</artifactId>
            <version>${log4j.version}</version>
            <scope>runtime</scope>
        </dependency>
    </dependencies>
</project>
```

Tags used in pom.xml:

- **groupId** — base tag containing all information about the project. The header contains information that Maven needs to understand the pom.xml file. The **modelVersion** tag defines a model version with which Maven must work. This parameter has not usually changed and should be set to the current version of the model;
- **groupId**, **artifactId**, **version** — [Maven project coordinates](#), which are determined when generating from the archetype;
- **name** — human-readable name of the project, used by Maven generated documentation;
- **properties** — specific settings such as file encoding and Java compiler source used. You can do without this block, then the default values will be used;
- **repositories** — block — contains the list of repositories;
- **build** — includes plugins from the archetype;
- **plugins** — block — declaration of all dependencies used in the project. Each found as required with a **dependency** request and specified unique identification using **groupId**, **artifactId** and **version**.

Ready-made Kafka Streams applications

When creating a project, the Streams quickstart archetype also creates boilerplate Stream Utilities examples, the generated code files for running Kafka Streams applications.

- [`LineSplit.java`](#) — a simple application that reads from a source **streams-plaintext-input** topic, splits each text line into words, and then writes to the sink **streams-linesplit-output** topic. The results can be observed using the **kafka-console-consumer** command.
- [`Pipe.java`](#) — an application that reads from a source **streams-plaintext-input** topic, splits each text line into words, and then writes to the sink **streams-linesplit-output** topic. The results can be observed using the **kafka-console-consumer** command.

The working code can be edited then compiled and tested and Maven can also be used to package your application in a JAR file.

Run the Kafka Streams application

Pipe.java code

The code below shows a Pipe application that consistently reads data using the high-level Streams DSL that reads messages from a source **streams-plaintext-input** topic and writes them unchanged to a destination **streams-pipe-output** topic:

```
Pipe

package myapps;

import org.apache.kafka.common.serialization.Serdes;
import org.apache.kafka.streams.KafkaStreams;
import org.apache.kafka.streams.StreamsBuilder;
import org.apache.kafka.streams.StreamsConfig;
import org.apache.kafka.streams.kstream.KStream;

import java.util.Properties;
import java.util.concurrent.CountDownLatch;

public class Pipe {

    public static void main(String[] args) throws Exception {
        Properties props = new Properties();
        props.put(StreamsConfig.APPLICATION_ID_CONFIG, "pipe-app");
        props.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092");
        props.put(StreamsConfig.DEFAULT_KEY_SERDE_CLASS_CONFIG, Serdes.String().getClass());
        props.put(StreamsConfig.DEFAULT_VALUE_SERDE_CLASS_CONFIG, Serdes.String().getClass());

        final StreamsBuilder builder = new StreamsBuilder();

        builder.stream("streams-plaintext-input").to("streams-pipe-output");

        final Topology topology = builder.build();
        final KafkaStreams streams = new KafkaStreams(topology, props);
        final CountDownLatch latch = new CountDownLatch(1);

        Runtime.getRuntime().addShutdownHook(new Thread("streams-shutdown-hook") {
            @Override
            public void run() {
                streams.close();
                latch.countDown();
            }
        });

        try {
            streams.start();
            latch.await();
        } catch (Throwable e) {
            System.exit(1);
        }
        System.exit(0);
    }
}
```

The Pipe.java code uses:

- [`props.put`](#) — a set of properties is passed, defined by the class [`StreamsConfig`](#):
  - **StreamsConfig.APPLICATION_ID_CONFIG** — an id required parameter to use in subscription to the Kafka cluster;
  - **StreamsConfig.BOOTSTRAP_SERVERS_CONFIG** — Streams: `localhost:9092` and **StreamsConfig.DEFAULT_VALUE_SERDE_CLASS_CONFIG**, **Serdes.String().getClass()** — [serialization](#) and deserialization of data used when the application topic using the class [`Serdes`](#) and [`Serdes.String()`](#);
  - **final StreamsBuilder builder = new StreamsBuilder()** — reference of the stream topology declared by the stream builder.
- **final StreamBuilder builder = new StreamsBuilder()** — creates an instance and uses the method with configuration settings [`StreamsBuilder`](#) and [`StreamsConfig`](#).
- **final Topology topology = builder.build()** — application termination logic defined by class [`KeyValueMapper`](#). You can terminate the program by pressing **Ctrl+C**.
- Displaying the states **Started**, **Stopped**, **started** and warnings descriptions using the **System.out.println** statement.

Run the application

The application, in the form of the file containing Maven and the file source, that is, `.pom/maven-file-streams-examples`.

Before running the application, you need to clean up the project and remove all files created by the previous build by running the command:

```
$ mvn clean package
```

Result:

```
[INFO] Building jar: /opt/maven-exec-0.1-SNAPSHOT/streams-examples/target/streams-examples-0.1.jar
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 02.476s
[INFO] Finished at: Wed Aug 22 11:47:28 UTC 2018
[INFO] Final Memory: 14M/160M
[INFO] ------------------------------------------------------------------------
```

Now to run the application, you need to compile the source code from the Pipe.java file using the command:

```
$ mvn compile exec:java -Dexec.mainClass="myapps.Pipe"
```

As a result of successful launch of the application, the message **Thread started** and a description of the topology are displayed, which has two connected nodes, a source node **KSTREAM-SOURCE-0000000000** and destination node **KSTREAM-SINK-0000000001**, and the corresponding topics attached to each node:

```
[INFO] --- exec-maven-plugin:1.6.0:exec (default-cli) @ streams-examples ---
[INFO]
[pipe-app-StreamThread-1] WARN org.apache.kafka.clients.consumer.ConsumerConfig - The configuration 'key.serializer' was supplied but isn't a known config.
[pipe-app-StreamThread-1] WARN org.apache.kafka.clients.consumer.ConsumerConfig - The configuration 'value.serializer' was supplied but isn't a known config.
[pipe-app-StreamThread-1] INFO org.apache.kafka.clients.producer.KafkaProducer - [Producer clientId=pipe-app-StreamThread-1-producer] Closing the Kafka producer with timeoutMillis = 9223372036854775807 ms.
Topology:
    Sub-topology: 0
    Source: KSTREAM-SOURCE-0000000000 (topics: [streams-plaintext-input])
      --> KSTREAM-SINK-0000000001
    Sink: KSTREAM-SINK-0000000001 (topics: streams-pipe-output)
      <-- KSTREAM-SOURCE-0000000000
```

The application is running. The program is terminated by pressing **Ctrl+C**.

Check the work of the created application

NOTE

For more information on the fundamentals of working with Kafka, see [quick-start-with-kafka](#).

To check the running application, you need to:

1. Create topics used in the application, or at least until a Kafka broker is enabled, run the following commands:

```
$ /opt/kafka/bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic streams-plaintext-input
```

2. Start the [`/opt/kafka/bin/kafka-topics.sh --create --topic streams-plaintext-input --bootstrap-server zoo-host-2:xx:xxxx2:2] command, [zoo-host-2:xx:xxxx2:2]

```
$ /opt/kafka/bin/kafka-topics.sh --create --topic streams-pipe-output --bootstrap-server zoo-host-2:xx:xxxx2:2
```

3. Create a producer for the source topic **streams-plaintext-input** using the command:

```
$ /opt/kafka/bin/kafka-topics.sh --create --topic streams-plaintext-input --bootstrap-server zoo-host-2:xx:xxxx2:2 --zookeeper localhost:2181
```

4. In another console, create a message consumer from the **streams-pipe-output** destination using the command:

```
$ /opt/kafka/bin/kafka-console-consumer.sh --topic streams-pipe-output --from-beginning --bootstrap-server zoo-host-2:xx:xxxx2:2 --zookeeper localhost:2181
```

5. Write messages in the input data format to the source topic **streams-plaintext-input** and read messages unchanged from the destination topic **streams-pipe-output**.

Kafka API usage example

Contents

- Create a Kafka application
- Create a project structure
  - File structure
  - Set up gulp.conf
- Create and run applications
- Producer API
- Consumer API
- Admin API

Prepare to create an application

Many of the example operations in this article refer to a Kafka command line tool. Download Kafka and unpack it to a convenient location.

Create a project structure

File structure

Create a directory structure for the application and add a basic `package.json` file to the root directory.

Set up gulp.conf

Add a `gulpfile.js` to the root directory and add the following code to it:

Create and run applications

Create a new file in the `src` directory called `producer.ts` and add the following code to it:

Create a new file in the `src` directory called `consumer.ts` and add the following code to it:

Producer API

The **Producer** class is instantiated by:

Many of the methods of the **Producer** class are asynchronous and return a promise. Some of the more commonly used methods are:

- **connect(metadataOptions?: MetadataOptions): Promise<void>** connects to the Kafka cluster.
- **send(record: ProducerRecord): Promise<RecordMetadata[]>** sends a message to the Kafka cluster. The **ProducerRecord** object has the following properties:
  - **topic: string** the topic to send the message to.
  - **messages: Array<Message>** an array of messages to send to the topic. The **Message** object has the following properties:
    - **key?: string | Buffer | null** the key of the message.
    - **value: string | Buffer | null** the value of the message.
    - **partition?: number** the partition to send the message to.
    - **headers?: IHeaders** the headers of the message.
  - **acks?: number** the number of acknowledgements required before the message is considered sent.
  - **timeout?: number** the timeout in milliseconds for the send operation.
  - **compression?: CompressionTypes** the compression type to use for the message.
- **sendBatch(batch: ProducerBatch): Promise<RecordMetadata[]>** sends a batch of messages to the Kafka cluster. The **ProducerBatch** object has the following properties:
  - **topicMessages: Array<TopicMessages>** an array of topic messages to send. The **TopicMessages** object has the following properties:
    - **topic: string** the topic to send the messages to.
    - **messages: Array<Message>** an array of messages to send to the topic.
  - **acks?: number** the number of acknowledgements required before the batch is considered sent.
  - **timeout?: number** the timeout in milliseconds for the send operation.
  - **compression?: CompressionTypes** the compression type to use for the batch.
- **transaction(): Promise<Transaction>** starts a transaction. The **Transaction** object has the following methods:
  - **send(record: ProducerRecord): Promise<RecordMetadata[]>** sends a message to the Kafka cluster.
  - **sendBatch(batch: ProducerBatch): Promise<RecordMetadata[]>** sends a batch of messages to the Kafka cluster.
  - **sendOffsets(offsets: { consumerGroupId: string; topics: Array<{ topic: string; partitions: Array<{ partition: number; offset: string }> }> }): Promise<void>** sends offsets to the transaction.
  - **commit(): Promise<void>** commits the transaction.
  - **abort(): Promise<void>** aborts the transaction.
- **events: ProducerEvents** an object containing the following event types:
  - **CONNECT** emitted when the producer connects to the Kafka cluster.
  - **DISCONNECT** emitted when the producer disconnects from the Kafka cluster.
  - **REQUEST** emitted when a request is sent to the Kafka cluster.
  - **REQUEST_TIMEOUT** emitted when a request times out.
  - **REQUEST_QUEUE_SIZE** emitted when the request queue size changes.
- **on(eventName: ProducerEvents, listener: (event: any) => void): void** adds a listener for the specified event.
- **removeListener(eventName: ProducerEvents, listener: (event: any) => void): void** removes a listener for the specified event.
- **removeAllListeners(eventName?: ProducerEvents): void** removes all listeners for the specified event.
- **logger(): Logger** returns the logger instance.
- **disconnect(): Promise<void>** disconnects from the Kafka cluster.

Run the application:

Consumer API

The **Consumer** class is instantiated by:

Many of the methods of the **Consumer** class are asynchronous and return a promise. Some of the more commonly used methods are:

- **connect(metadataOptions?: MetadataOptions): Promise<void>** connects to the Kafka cluster.
- **subscribe(topic: string | RegExp, fromBeginning?: boolean): Promise<void>** subscribes to a topic. The **topic** parameter can be a string or a regular expression. The **fromBeginning** parameter is optional and defaults to `false`.
- **run(config: { eachMessage: (payload: EachMessagePayload) => Promise<void>; autoCommit?: boolean; autoCommitInterval?: number; autoCommitThreshold?: number; }): Promise<void>** starts the consumer. The **eachMessage** function is called for each message received. The **autoCommit** parameter is optional and defaults to `true`. The **autoCommitInterval** parameter is optional and defaults to `5000`. The **autoCommitThreshold** parameter is optional and defaults to `100`.
- **stop(): Promise<void>** stops the consumer.
- **commitOffsets(offsets: { topic: string; partition: number; offset: string }[]): Promise<void>** commits offsets to the Kafka cluster.
- **seek(topicPartitionOffset: { topic: string; partition: number; offset: string }): void** seeks to the specified offset.
- **pause(topics: { topic: string; partitions?: number[] }[]): void** pauses the consumer for the specified topics and partitions.
- **resume(topics: { topic: string; partitions?: number[] }[]): void** resumes the consumer for the specified topics and partitions.
- **events: ConsumerEvents** an object containing the following event types:
  - **CONNECT** emitted when the consumer connects to the Kafka cluster.
  - **DISCONNECT** emitted when the consumer disconnects from the Kafka cluster.
  - **REQUEST** emitted when a request is sent to the Kafka cluster.
  - **REQUEST_TIMEOUT** emitted when a request times out.
  - **REQUEST_QUEUE_SIZE** emitted when the request queue size changes.
  - **GROUP_JOIN** emitted when the consumer joins a consumer group.
  - **FETCH_START** emitted when the consumer starts fetching messages.
  - **FETCH** emitted when the consumer fetches messages.
  - **START_BATCH_PROCESS** emitted when the consumer starts processing a batch of messages.
  - **END_BATCH_PROCESS** emitted when the consumer finishes processing a batch of messages.
  - **COMMIT_OFFSETS** emitted when the consumer commits offsets.
  - **CRASH** emitted when the consumer crashes.
- **on(eventName: ConsumerEvents, listener: (event: any) => void): void** adds a listener for the specified event.
- **removeListener(eventName: ConsumerEvents, listener: (event: any) => void): void** removes a listener for the specified event.
- **removeAllListeners(eventName?: ConsumerEvents): void** removes all listeners for the specified event.
- **logger(): Logger** returns the logger instance.
- **disconnect(): Promise<void>** disconnects from the Kafka cluster.

Run the application:

Admin API

The **Admin** class is instantiated by:

Many of the methods of the **Admin** class are asynchronous and return a promise. Some of the more commonly used methods are:

- **connect(): Promise<void>** connects to the Kafka cluster.
- **createTopics(topics: { topic: string; numPartitions?: number; replicationFactor?: number; replicaAssignment?: { partition: number; replicas: number[] }[]; configEntries?: { name: string; value: string }[] }[]): Promise<void>** creates topics.
- **deleteTopics(topics: string[]): Promise<void>** deletes topics.
- **listTopics(): Promise<string[]>** lists topics.
- **fetchTopicMetadata(topics: string[]): Promise<{ topics: { name: string; partitions: { partitionId: number; leader: number; replicas: number[]; isr: number[] }[] }[] }>** fetches topic metadata.
- **createPartitions(topics: { topic: string; count: number; assignments?: number[][] }[]): Promise<void>** creates partitions.
- **fetchOffsets(topics: { topic: string; partitions?: number[] }[]): Promise<{ topic: string; partitions: { partitionId: number; offset: string; high: string; low: string }[] }[]>** fetches offsets.
- **fetchTopicOffsets(topic: string): Promise<{ partitionId: number; offset: string; high: string; low: string }[]>** fetches topic offsets.
- **fetchTopicOffsetsByTimestamp(topic: string, timestamp: number): Promise<{ partitionId: number; offset: string; high: string; low: string }[]>** fetches topic offsets by timestamp.
- **setOffsets(topics: { topic: string; partitions: { partitionId: number; offset: string }[] }[]): Promise<void>** sets offsets.
- **resetOffsets(topics: { topic: string; partitions?: number[] }[]): Promise<void>** resets offsets.
- **describeCluster(): Promise<{ brokers: { nodeId: number; host: string; port: number; rack: string }[]; controller: number; clusterId: string }>** describes the cluster.
- **describeConfigs(resources: { type: number; name: string; configNames?: string[] }[]): Promise<{ resources: { type: number; name: string; configEntries: { name: string; value: string; isDefault: boolean; isSensitive: boolean; readOnly: boolean }[] }[] }>** describes configs.
- **alterConfigs(resources: { type: number; name: string; configEntries: { name: string; value: string }[] }[]): Promise<void>** alters configs.
- **deleteGroups(groupIds: string[]): Promise<void>** deletes groups.
- **listGroups(): Promise<{ groups: { groupId: string; protocolType: string }[] }>** lists groups.
- **describeGroups(groupIds: string[]): Promise<{ groups: { groupId: string; protocolType: string; state: string; members: { memberId: string; clientId: string; clientHost: string; memberMetadata: Buffer; memberAssignment: Buffer }[] }[] }>** describes groups.
- **deleteTopicRecords(topics: { topic: string; partitions: { partitionId: number; offset: string }[] }[]): Promise<void>** deletes topic records.
- **events: AdminEvents** an object containing the following event types:
  - **CONNECT** emitted when the admin client connects to the Kafka cluster.
  - **DISCONNECT** emitted when the admin client disconnects from the Kafka cluster.
  - **REQUEST** emitted when a request is sent to the Kafka cluster.
  - **REQUEST_TIMEOUT** emitted when a request times out.
  - **REQUEST_QUEUE_SIZE** emitted when the request queue size changes.
- **on(eventName: AdminEvents, listener: (event: any) => void): void** adds a listener for the specified event.
- **removeListener(eventName: AdminEvents, listener: (event: any) => void): void** removes a listener for the specified event.
- **removeAllListeners(eventName?: AdminEvents): void** removes all listeners for the specified event.
- **logger(): Logger** returns the logger instance.
- **disconnect(): Promise<void>** disconnects from the Kafka cluster.

Run the application:

Cruise Control usage example

Contents

- Prerequisites
- Configure Cruise Control
- Configure Cruise Control
- Start Cruise Control
- Use Cruise Control REST API
  - Cruise Control state
  - Rebalance
  - Proposals endpoint
  - Check the status of the request

Prerequisites

Cruise Control requires a running Kafka cluster to be deployed before it can be started.

Cruise Control supports Kafka 3.3.x and later.

Use the `KafkaRebalance` resource with Cruise Control to change the topic replication factor, optimize leadership, and rebalance the load across your Kafka cluster. By default, when you create a `KafkaRebalance` resource, it connects to the Cruise Control API and generates an optimization proposal for your Kafka cluster.

Configure Cruise Control

Configure Cruise Control by updating the `spec.cruiseControl` property in the `Kafka` resource. The `cruiseControl` property is optional, but should be included so that Cruise Control can be deployed.

If the `cruiseControl` property is omitted from the `Kafka` resource configuration, the Cluster Operator does not deploy Cruise Control and other components cannot be used, such as `KafkaRebalance` resources.

Configure the `cruiseControl` property in the `Kafka` resource. For example:

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    # ...
    config:
      # ...
      auto.create.topics.enable: "false" # (1)
  zookeeper:
    # ...
  cruiseControl:
    brokerCapacity: # (2)
      inboundNetwork: 10000KB/s
      outboundNetwork: 10000KB/s
    # ...
```

1. Kafka configuration options that set the capacity for brokers.
2. Broker capacity for inbound and outbound network bandwidth rates.

Configure the `KafkaRebalance` resource with the `goals` property to specify the goals used by the rebalance optimization proposal.

Start Cruise Control

Deploy the Kafka cluster with Cruise Control enabled to start the Cruise Control server.

After the Cruise Control server instance is deployed, the Topic Operator and User Operator enable the `KafkaRebalance` and `KafkaRebalanceNode` resources. Cruise Control can be started by applying a `KafkaRebalance` resource.

Use Cruise Control REST API

Interact with the Cruise Control REST API for generating optimization proposals using `curl` or `kubectl exec`.

Access the Cruise Control API using the `localhost:9090` address and port.

Cruise Control provides the following REST API endpoints:

- `/kafkacruisecontrol/state`
- `/kafkacruisecontrol/user_tasks`
- `/kafkacruisecontrol/rebalance`
- `/kafkacruisecontrol/proposals`

All of the Cruise Control REST API endpoints are listed in the Cruise Control wiki. Some of the endpoints are `POST` requests by default and have to be enabled in the Cruise Control configuration by setting `webserver.api.urlprefix` to `POST`. Other endpoints are `GET` requests.

`GET` requests are used for read-only requests to the Cruise Control server that do not change the state of the Kafka cluster. For example, a `GET` request can be used to obtain an optimization proposal that cannot be approved later.

`POST` requests are used for requests that change the state of the Kafka cluster. For example, a `POST` request is used to initiate a rebalance operation that will then be executed by the Cruise Control server.

Use the `curl` utility to send requests to the Cruise Control REST API within the Kafka broker container. For example, to get the configuration options of the Cruise Control server:

```
kubectl exec -it my-cluster-kafka-0 -- curl localhost:9090/kafkacruisecontrol/state
```

In this command, `curl` sends a request to the Cruise Control server state endpoint to get the server configuration. The command runs inside the Kafka broker container (`my-cluster-kafka-0`) and connects to the Cruise Control API server running on `localhost:9090`.

Cruise Control state

The state endpoint returns information about the Cruise Control server.

Use the `/kafkacruisecontrol/state` endpoint to get the state:

```
kubectl exec -it my-cluster-kafka-0 -- curl localhost:9090/kafkacruisecontrol/state
```

Cruise Control returns a response in JSON format:


```
{
    "MonitorState": {
        "state": "RUNNING",
        "numValidWindows": 1,
        "numMonitoredWindows": 1,
        "numFlawedWindows": 0,
        "monitoringCoveragePercentage": 0.0,
        "numValidPartitions": 0,
        "numTotalPartitions": 0,
        "numValidSnapshotWindows": 1,
        "numTotalSnapshotWindows": 1,
        "reasonOfLatestPauseOrResume": "Initialization."
    },
    "ExecutorState": {
        "state": "NO_TASK_IN_PROGRESS",
        "recentlyDemotedBrokers": [],
        "recentlyRemovedBrokers": []
    },
    "AnalyzerState": {
        "isProposalReady": false,
        "readyGoals": []
    },
    "version": 1
}
```


Use the `verbose` option to get more detailed information about the Cruise Control server:

```
kubectl exec -it my-cluster-kafka-0 -- curl localhost:9090/kafkacruisecontrol/state?verbose=true
```


```
{
    "MonitorState": {
        "state": "RUNNING",
        "numValidWindows": 1,
        "numMonitoredWindows": 1,
        "numFlawedWindows": 0,
        "monitoringCoveragePercentage": 0.0,
        "numValidPartitions": 0,
        "numTotalPartitions": 0,
        "numValidSnapshotWindows": 1,
        "numTotalSnapshotWindows": 1,
        "reasonOfLatestPauseOrResume": "Initialization.",
        "trainingMetricSamplerPartitionMetricSampleAggregatorState": "Training: Monitored 0/0 (0.0%) partitions in all windows. Ready: 0/0 (0.0%) partitions in all windows (0 windows).",
        "loadingMetricSamplerPartitionMetricSampleAggregatorState": "Loading: Monitored 0/0 (0.0%) partitions in all windows. Ready: 0/0 (0.0%) partitions in all windows (0 windows).",
        "bootstrapMetricSamplerPartitionMetricSampleAggregatorState": "Bootstrap: Monitored 0/0 (0.0%) partitions in all windows. Ready: 0/0 (0.0%) partitions in all windows (0 windows)."
    },
    "ExecutorState": {
        "state": "NO_TASK_IN_PROGRESS",
        "recentlyDemotedBrokers": [],
        "recentlyRemovedBrokers": [],
        "recentDemotionTasks": [],
        "recentRemovalTasks": [],
        "ongoingConcurrencyAdjustment": "false",
        "ongoingLeaderReplicaMovementTasks": [],
        "ongoingInterBrokerReplicaMovementTasks": [],
        "ongoingIntraBrokerReplicaMovementTasks": [],
        "numFinishedInterBrokerReplicaMovements": 0,
        "finishedInterBrokerDataMovement": 0,
        "numFinishedIntraBrokerReplicaMovements": 0,
        "finishedIntraBrokerDataMovement": 0,
        "numFinishedLeaderMovements": 0,
        "numCancelledInterBrokerReplicaMovements": 0,
        "numStoppedInterBrokerReplicaMovements": 0,
        "numCancelledIntraBrokerReplicaMovements": 0,
        "numStoppedIntraBrokerReplicaMovements": 0,
        "numCancelledLeaderMovements": 0,
        "numStoppedLeaderMovements": 0,
        "numDeadInterBrokerReplicaMovements": 0,
        "deadInterBrokerDataMovement": 0,
        "numDeadIntraBrokerReplicaMovements": 0,
        "deadIntraBrokerDataMovement": 0,
        "numDeadLeaderMovements": 0,
        "numPendingInterBrokerReplicaMovements": 0,
        "pendingInterBrokerDataMovement": 0,
        "numPendingIntraBrokerReplicaMovements": 0,
        "pendingIntraBrokerDataMovement": 0,
        "numPendingLeaderMovements": 0,
        "numTotalInterBrokerReplicaMovements": 0,
        "totalInterBrokerDataMovement": 0,
        "numTotalIntraBrokerReplicaMovements": 0,
        "totalIntraBrokerDataMovement": 0,
        "numTotalLeaderMovements": 0,
        "numTotalPartitionMovements": 0,
        "numTotalPartitionMovementsInExecution": 0,
        "numTotalPartitionMovementsInExecutionAndPending": 0,
        "numTotalPartitionMovementsInExecutionAndPendingAndCancelled": 0,
        "numTotalPartitionMovementsInExecutionAndPendingAndCancelledAndStopped": 0,
        "numTotalPartitionMovementsInExecutionAndPendingAndCancelledAndStoppedAndDead": 0,
        "numTotalPartitionMovementsInExecutionAndPendingAndCancelledAndStoppedAndDeadAndFinished": 0,
        "numTotalPartitionMovementsInExecutionAndPendingAndCancelledAndStoppedAndDeadAndFinishedAndAborted": 0,
        "numTotalPartitionMovementsInExecutionAndPendingAndCancelledAndStoppedAndDeadAndFinishedAndAbortedAndFailed": 0,
        "numTotalPartitionMovementsInExecutionAndPendingAndCancelledAndStoppedAndDeadAndFinishedAndAbortedAndFailedAndSkipped": 0,
        "numTotalPartitionMovementsInExecutionAndPendingAndCancelledAndStoppedAndDeadAndFinishedAndAbortedAndFailedAndSkippedAndUnknown": 0,
        "numTotalPartitionMovementsInExecutionAndPendingAndCancelledAndStoppedAndDeadAndFinishedAndAbortedAndFailedAndSkippedAndUnknownAndOther": 0,
        "numTotalPartitionMovementsInExecutionAndPendingAndCancelledAndStoppedAndDeadAndFinishedAndAbortedAndFailedAndSkippedAndUnknownAndOtherAndNone": 0,
        "numTotalPartitionMovementsInExecutionAndPendingAndCancelledAndStoppedAndDeadAndFinishedAndAbortedAndFailedAndSkippedAndUnknownAndOtherAndNoneAndNull": 0,
        "numTotalPartitionMovementsInExecutionAndPendingAndCancelledAndStoppedAndDeadAndFinishedAndAbortedAndFailedAndSkippedAndUnknownAndOtherAndNoneAndNullAndEmpty": 0,
        "numTotalPartitionMovementsInExecutionAndPendingAndCancelledAndStoppedAndDeadAndFinishedAndAbortedAndFailedAndSkippedAndUnknownAndOtherAndNoneAndNullAndEmptyAndBlank": 0,
        "numTotalPartitionMovementsInExecutionAndPendingAndCancelledAndStoppedAndDeadAndFinishedAndAbortedAndFailedAndSkippedAndUnknownAndOtherAndNoneAndNullAndEmptyAndBlankAndZero": 0,
        "numTotalPartitionMovementsInExecutionAndPendingAndCancelledAndStoppedAndDeadAndFinishedAndAbortedAndFailedAndSkippedAndUnknownAndOtherAndNoneAndNullAndEmptyAndBlankAndZeroAndOne": 0,
        "numTotalPartitionMovementsInExecutionAndPendingAndCancelledAndStoppedAndDeadAndFinishedAndAbortedAndFailedAndSkippedAndUnknownAndOtherAndNoneAndNullAndEmptyAndBlankAndZeroAndOneAndTwo": 0,
        "numTotalPartitionMovementsInExecutionAndPendingAndCancelledAndStoppedAndDeadAndFinishedAndAbortedAndFailedAndSkippedAndUnknownAndOtherAndNoneAndNullAndEmptyAndBlankAndZeroAndOneAndTwoAndThree": 0,
        "numTotalPartitionMovementsInExecutionAndPendingAndCancelledAndStoppedAndDeadAndFinishedAndAbortedAndFailedAndSkippedAndUnknownAndOtherAndNoneAndNullAndEmptyAndBlankAndZeroAndOneAndTwoAndThreeAndFour": 0,
        "numTotalPartitionMovementsInExecutionAndPendingAndCancelledAndStoppedAndDeadAndFinishedAndAbortedAndFailedAndSkippedAndUnknownAndOtherAndNoneAndNullAndEmptyAndBlankAndZeroAndOneAndTwoAndThreeAndFourAndFive": 0,
        "numTotalPartitionMovementsInExecutionAndPendingAndCancelledAndStoppedAndDeadAndFinishedAndAbortedAndFailedAndSkippedAndUnknownAndOtherAndNoneAndNullAndEmptyAndBlankAndZeroAndOneAndTwoAndThreeAndFourAndFiveAndSix": 0,
        "numTotalPartitionMovementsInExecutionAndPendingAndCancelledAndStoppedAndDeadAndFinishedAndAbortedAndFailedAndSkippedAndUnknownAndOtherAndNoneAndNullAndEmptyAndBlankAndZeroAndOneAndTwoAndThreeAndFourAndFiveAndSixAndSeven": 0,
        "numTotalPartitionMovementsInExecutionAndPendingAndCancelledAndStoppedAndDeadAndFinishedAndAbortedAndFailedAndSkippedAndUnknownAndOtherAndNoneAndNullAndEmptyAndBlankAndZeroAndOneAndTwoAndThreeAndFourAndFiveAndSixAndSevenAndEight": 0,
        "numTotalPartitionMovementsInExecutionAndPendingAndCancelledAndStoppedAndDeadAndFinishedAndAbortedAndFailedAndSkippedAndUnknownAndOtherAndNoneAndNullAndEmptyAndBlankAndZeroAndOneAndTwoAndThreeAndFourAndFiveAndSixAndSevenAndEightAndNine": 0,
        "numTotalPartitionMovementsInExecutionAndPendingAndCancelledAndStoppedAndDeadAndFinishedAndAbortedAndFailedAndSkippedAndUnknownAndOtherAndNoneAndNullAndEmptyAndBlankAndZeroAndOneAndTwoAndThreeAndFourAndFiveAndSixAndSevenAndEightAndNineAndTen": 0,
        "numTotalPartitionMovementsInExecutionAndPendingAndCancelledAndStoppedAndDeadAndFinishedAndAbortedAndFailedAndSkippedAndUnknownAndOtherAndNoneAndNullAndEmptyAndBlankAndZeroAndOneAndTwoAndThreeAndFourAndFiveAndSixAndSevenAndEightAndNineAndTenAndEleven": 0,
        "numTotalPartitionMovementsInExecutionAndPendingAndCancelledAndStoppedAndDeadAndFinishedAndAbortedAndFailedAndSkippedAndUnknownAndOtherAndNoneAndNullAndEmptyAndBlankAndZeroAndOneAndTwoAndThreeAndFourAndFiveAndSixAndSevenAndEightAndNineAndTenAndElevenAndTwelve": 0,
        "numTotalPartitionMovementsInExecutionAndPendingAndCancelledAndStoppedAndDeadAndFinishedAndAbortedAndFailedAndSkippedAndUnknownAndOtherAndNoneAndNullAndEmptyAndBlankAndZeroAndOneAndTwoAndThreeAndFourAndFiveAndSixAndSevenAndEightAndNineAndTenAndElevenAndTwelveAndThirteen": 0,
        "numTotalPartitionMovementsInExecutionAndPendingAndCancelledAndStoppedAndDeadAndFinishedAndAbortedAndFailedAndSkippedAndUnknownAndOtherAndNoneAndNullAndEmptyAndBlankAndZeroAndOneAndTwoAndThreeAndFourAndFiveAndSixAndSevenAndEightAndNineAndTenAndElevenAndTwelveAndThirteenAndFourteen": 0,
        "numTotalPartitionMovementsInExecutionAndPendingAndCancelledAndStoppedAndDeadAndFinishedAndAbortedAndFailedAndSkippedAndUnknownAndOtherAndNoneAndNullAndEmptyAndBlankAndZeroAndOneAndTwoAndThreeAndFourAndFiveAndSixAndSevenAndEightAndNineAndTenAndElevenAndTwelveAndThirteenAndFourteenAndFifteen": 0,
        "numTotalPartitionMovementsInExecutionAndPendingAndCancelledAndStoppedAndDeadAndFinishedAndAbortedAndFailedAndSkippedAndUnknownAndOtherAndNoneAndNullAndEmptyAndBlankAndZeroAndOneAndTwoAndThreeAndFourAndFiveAndSixAndSevenAndEightAndNineAndTenAndElevenAndTwelveAndThirteenAndFourteenAndFifteenAndSixteen": 0,
        "numTotalPartitionMovementsInExecutionAndPendingAndCancelledAndStoppedAndDeadAndFinishedAndAbortedAndFailedAndSkippedAndUnknownAndOtherAndNoneAndNullAndEmptyAndBlankAndZeroAndOneAndTwoAndThreeAndFourAndFiveAndSixAndSevenAndEightAndNineAndTenAndElevenAndTwelveAndThirteenAndFourteenAndFifteenAndSixteenAndSeventeen": 0,
        "numTotalPartitionMovementsInExecutionAndPendingAndCancelledAndStoppedAndDeadAndFinishedAndAbortedAndFailedAndSkippedAndUnknownAndOtherAndNoneAndNullAndEmptyAndBlankAndZeroAndOneAndTwoAndThreeAndFourAndFiveAndSixAndSevenAndEightAndNineAndTenAndElevenAndTwelveAndThirteenAndFourteenAndFifteenAndSixteenAndSeventeenAndEighteen": 0,
        "numTotalPartitionMovementsInExecutionAndPendingAndCancelledAndStoppedAndDeadAndFinishedAndAbortedAndFailedAndSkippedAndUnknownAndOtherAndNoneAndNullAndEmptyAndBlankAndZeroAndOneAndTwoAndThreeAndFourAndFiveAndSixAndSevenAndEightAndNineAndTenAndElevenAndTwelveAndThirteenAndFourteenAndFifteenAndSixteenAndSeventeenAndEighteenAndNineteen": 0,
        "numTotalPartitionMovementsInExecutionAndPendingAndCancelledAndStoppedAndDeadAndFinishedAndAbortedAndFailedAndSkippedAndUnknownAndOtherAndNoneAndNullAndEmptyAndBlankAndZeroAndOneAndTwoAndThreeAndFourAndFiveAndSixAndSevenAndEightAndNineAndTenAndElevenAndTwelveAndThirteenAndFourteenAndFifteenAndSixteenAndSeventeenAndEighteenAndNineteenAndTwenty": 0,
        "numTotalPartitionMovementsInExecutionAndPendingAndCancelledAndStoppedAndDeadAndFinishedAndAbortedAndFailedAndSkippedAndUnknownAndOtherAndNoneAndNullAndEmptyAndBlankAndZeroAndOneAndTwoAndThreeAndFourAndFiveAndSixAndSevenAndEightAndNineAndTenAndElevenAndTwelveAndThirteenAndFourteenAndFifteenAndSixteenAndSeventeenAndEighteenAndNineteenAndTwentyAndTwentyOne": 0,
        "numTotalPartitionMovementsInExecutionAndPendingAndCancelledAndStoppedAndDeadAndFinishedAndAbortedAndFailedAndSkippedAndUnknownAndOtherAndNoneAndNullAndEmptyAndBlankAndZeroAndOneAndTwoAndThreeAndFourAndFiveAndSixAndSevenAndEightAndNineAndTenAndElevenAndTwelveAndThirteenAndFourteenAndFifteenAndSixteenAndSeventeenAndEighteenAndNineteenAndTwentyAndTwentyOneAndTwentyTwo": 0,
        "numTotalPartitionMovementsInExecutionAndPendingAndCancelledAndStoppedAndDeadAndFinishedAndAbortedAndFailedAndSkippedAndUnknownAndOtherAndNoneAndNullAndEmptyAndBlankAndZeroAndOneAndTwoAndThreeAndFourAndFiveAndSixAndSevenAndEightAndNineAndTenAndElevenAndTwelveAndThirteenAndFourteenAndFifteenAndSixteenAndSeventeenAndEighteenAndNineteenAndTwentyAndTwentyOneAndTwentyTwoAndTwentyThree": 0,
        "numTotalPartitionMovementsInExecutionAndPendingAndCancelledAndStoppedAndDeadAndFinishedAndAbortedAndFailedAndSkippedAndUnknownAndOtherAndNoneAndNullAndEmptyAndBlankAndZeroAndOneAndTwoAndThreeAndFourAndFiveAndSixAndSevenAndEightAndNineAndTenAndElevenAndTwelveAndThirteenAndFourteenAndFifteenAndSixteenAndSeventeenAndEighteenAndNineteenAndTwentyAndTwentyOneAndTwentyTwoAndTwentyThreeAndTwentyFour": 0,
        "numTotalPartitionMovementsInExecutionAndPendingAndCancelledAndStoppedAndDeadAndFinishedAndAbortedAndFailedAndSkippedAndUnknownAndOtherAndNoneAndNullAndEmptyAndBlankAndZeroAndOneAndTwoAndThreeAndFourAndFiveAndSixAndSevenAndEightAndNineAndTenAndElevenAndTwelveAndThirteenAndFourteenAndFifteenAndSixteenAndSeventeenAndEighteenAndNineteenAndTwentyAndTwentyOneAndTwentyTwoAndTwentyThreeAndTwentyFourAndTwentyFive": 0,
        "numTotalPartitionMovementsInExecutionAndPendingAndCancelledAndStoppedAndDeadAndFinishedAndAbortedAndFailedAndSkippedAndUnknownAndOtherAndNoneAndNullAndEmptyAndBlankAndZeroAndOneAndTwoAndThreeAndFourAndFiveAndSixAndSevenAndEightAndNineAndTenAndElevenAndTwelveAndThirteenAndFourteenAndFifteenAndSixteenAndSeventeenAndEighteenAndNineteenAndTwentyAndTwentyOneAndTwentyTwoAndTwentyThreeAndTwentyFourAndTwentyFiveAndTwentySix": 0,
        "numTotalPartitionMovementsInExecutionAndPendingAndCancelledAndStoppedAndDeadAndFinishedAndAbortedAndFailedAndSkippedAndUnknownAndOtherAndNoneAndNullAndEmptyAndBlankAndZeroAndOneAndTwoAndThreeAndFourAndFiveAndSixAndSevenAndEightAndNineAndTenAndElevenAndTwelveAndThirteenAndFourteenAndFifteenAndSixteenAndSeventeenAndEighteenAndNineteenAndTwentyAndTwentyOneAndTwentyTwoAndTwentyThreeAndTwentyFourAndTwentyFiveAndTwentySixAndTwentySeven": 0,
        "numTotalPartitionMovementsInExecutionAndPendingAndCancelledAndStoppedAndDeadAndFinishedAndAbortedAndFailedAndSkippedAndUnknownAndOtherAndNoneAndNullAndEmptyAndBlankAndZeroAndOneAndTwoAndThreeAndFourAndFiveAndSixAndSevenAndEightAndNineAndTenAndElevenAndTwelveAndThirteenAndFourteenAndFifteenAndSixteenAndSeventeenAndEighteenAndNineteenAndTwentyAndTwentyOneAndTwentyTwoAndTwentyThreeAndTwentyFourAndTwentyFiveAndTwentySixAndTwentySevenAndTwentyEight": 0,
        "numTotalPartitionMovementsInExecutionAndPendingAndCancelledAndStoppedAndDeadAndFinishedAndAbortedAndFailedAndSkippedAndUnknownAndOtherAndNoneAndNullAndEmptyAndBlankAndZeroAndOneAndTwoAndThreeAndFourAndFiveAndSixAndSevenAndEightAndNineAndTenAndElevenAndTwelveAndThirteenAndFourteenAndFifteenAndSixteenAndSeventeenAndEighteenAndNineteenAndTwentyAndTwentyOneAndTwentyTwoAndTwentyThreeAndTwentyFourAndTwentyFiveAndTwentySixAndTwentySevenAndTwentyEightAndTwentyNine": 0,
        "numTotalPartitionMovementsInExecutionAndPendingAndCancelledAndStoppedAndDeadAndFinishedAndAbortedAndFailedAndSkippedAndUnknownAndOtherAndNoneAndNullAndEmptyAndBlankAndZeroAndOneAndTwoAndThreeAndFourAndFiveAndSixAndSevenAndEightAndNineAndTenAndElevenAndTwelveAndThirteenAndFourteenAndFifteenAndSixteenAndSeventeenAndEighteenAndNineteenAndTwentyAndTwentyOneAndTwentyTwoAndTwentyThreeAndTwentyFourAndTwentyFiveAndTwentySixAndTwentySevenAndTwentyEightAndTwentyNineAndThirty": 0,
        "numTotalPartitionMovementsInExecutionAndPendingAndCancelledAndStoppedAndDeadAndFinishedAndAbortedAndFailedAndSkippedAndUnknownAndOtherAndNoneAndNullAndEmptyAndBlankAndZeroAndOneAndTwoAndThreeAndFourAndFiveAndSixAndSevenAndEightAndNineAndTenAndElevenAndTwelveAndThirteenAndFourteenAndFifteenAndSixteenAndSeventeenAndEighteenAndNineteenAndTwentyAndTwentyOneAndTwentyTwoAndTwentyThreeAndTwentyFourAndTwentyFiveAndTwentySixAndTwentySevenAndTwentyEightAndTwentyNineAndThirtyAndThirtyOne": 0,
        "numTotalPartitionMovementsInExecutionAndPendingAndCancelledAndStoppedAndDeadAndFinishedAndAbortedAndFailedAndSkippedAndUnknownAndOtherAndNoneAndNullAndEmptyAndBlankAndZeroAndOneAndTwoAndThreeAndFourAndFiveAndSixAndSevenAndEightAndNineAndTenAndElevenAndTwelveAndThirteenAndFourteenAndFifteenAndSixteenAndSeventeenAndEighteenAndNineteenAndTwentyAndTwentyOneAndTwentyTwoAndTwentyThreeAndTwentyFourAndTwentyFiveAndTwentySixAndTwentySevenAndTwentyEightAndTwentyNineAndThirtyAndThirtyOneAndThirtyTwo": 0,
        "numTotalPartitionMovementsInExecutionAndPendingAndCancelledAndStoppedAndDeadAndFinishedAndAbortedAndFailedAndSkippedAndUnknownAndOtherAndNoneAndNullAndEmptyAndBlankAndZeroAndOneAndTwoAndThreeAndFourAndFiveAndSixAndSevenAndEightAndNineAndTenAndElevenAndTwelveAndThirteenAndFourteenAndFifteenAndSixteenAndSeventeenAndEighteenAndNineteenAndTwentyAndTwentyOneAndTwentyTwoAndTwentyThreeAndTwentyFourAndTwentyFiveAndTwentySixAndTwentySevenAndTwentyEightAndTwentyNineAndThirtyAndThirtyOneAndThirtyTwoAndThirtyThree": 0,
        "numTotalPartitionMovementsInExecutionAndPendingAndCancelledAndStoppedAndDeadAndFinishedAndAbortedAndFailedAndSkippedAndUnknownAndOtherAndNoneAndNullAndEmptyAndBlankAndZeroAndOneAndTwoAndThreeAndFourAndFiveAndSixAndSevenAndEightAndNineAndTenAndElevenAndTwelveAndThirteenAndFourteenAndFifteenAndSixteenAndSeventeenAndEighteenAndNineteenAndTwentyAndTwentyOneAndTwentyTwoAndTwentyThreeAndTwentyFourAndTwentyFiveAndTwentySixAndTwentySevenAndTwentyEightAndTwentyNineAndThirtyAndThirtyOneAndThirtyTwoAndThirtyThreeAndThirtyFour": 0,
        "numTotalPartitionMovementsInExecutionAndPendingAndCancelledAndStoppedAndDeadAndFinishedAndAbortedAndFailedAndSkippedAndUnknownAndOtherAndNoneAndNullAndEmptyAndBlankAndZeroAndOneAndTwoAndThreeAndFourAndFiveAndSixAndSevenAndEightAndNineAndTenAndElevenAndTwelveAndThirteenAndFourteenAndFifteenAndSixteenAndSeventeenAndEighteenAndNineteenAndTwentyAndTwentyOneAndTwentyTwoAndTwentyThreeAndTwentyFourAndTwentyFiveAndTwentySixAndTwentySevenAndTwentyEightAndTwentyNineAndThirtyAndThirtyOneAndThirtyTwoAndThirtyThreeAndThirtyFourAndThirtyFive": 0,
        "numTotalPartitionMovementsInExecutionAndPendingAndCancelledAndStoppedAndDeadAndFinishedAndAbortedAndFailedAndSkippedAndUnknownAndOtherAndNoneAndNullAndEmptyAndBlankAndZeroAndOneAndTwoAndThreeAndFourAndFiveAndSixAndSevenAndEightAndNineAndTenAndElevenAndTwelveAndThirteenAndFourteenAndFifteenAndSixteenAndSeventeenAndEighteenAndNineteenAndTwentyAndTwentyOneAndTwentyTwoAndTwentyThreeAndTwentyFourAndTwentyFiveAndTwentySixAndTwentySevenAndTwentyEightAndTwentyNineAndThirtyAndThirtyOneAndThirtyTwoAndThirtyThreeAndThirtyFourAndThirtyFiveAndThirtySix": 0,
        "numTotalPartitionMovementsInExecutionAndPendingAndCancelledAndStoppedAndDeadAndFinishedAndAbortedAndFailedAndSkippedAndUnknownAndOtherAndNoneAndNullAndEmptyAndBlankAndZeroAndOneAndTwoAndThreeAndFourAndFiveAndSixAndSevenAndEightAndNineAndTenAndElevenAndTwelveAndThirteenAndFourteenAndFifteenAndSixteenAndSeventeenAndEighteenAndNineteenAndTwentyAndTwentyOneAndTwentyTwoAndTwentyThreeAndTwentyFourAndTwentyFiveAndTwentySixAndTwentySevenAndTwentyEightAndTwentyNineAndThirtyAndThirtyOneAndThirtyTwoAndThirtyThreeAndThirtyFourAndThirtyFiveAndThirtySixAndThirtySeven": 0,
        "numTotalPartitionMovementsInExecutionAndPendingAndCancelledAndStoppedAndDeadAndFinishedAndAbortedAndFailedAndSkippedAndUnknownAndOtherAndNoneAndNullAndEmptyAndBlankAndZeroAndOneAndTwoAndThreeAndFourAndFiveAndSixAndSevenAndEightAndNineAndTenAndElevenAndTwelveAndThirteenAndFourteenAndFifteenAndSixteenAndSeventeenAndEighteenAndNineteenAndTwentyAndTwentyOneAndTwentyTwoAndTwentyThreeAndTwentyFourAndTwentyFiveAndTwentySixAndTwentySevenAndTwentyEightAndTwentyNineAndThirtyAndThirtyOneAndThirtyTwoAndThirtyThreeAndThirtyFourAndThirtyFiveAndThirtySixAndThirtySevenAndThirtyEight": 0,
        "numTotalPartitionMovementsInExecutionAndPendingAndCancelledAndStoppedAndDeadAndFinishedAndAbortedAndFailedAndSkippedAndUnknownAndOtherAndNoneAndNullAndEmptyAndBlankAndZeroAndOneAndTwoAndThreeAndFourAndFiveAndSixAndSevenAndEightAndNineAndTenAndElevenAndTwelveAndThirteenAndFourteenAndFifteenAndSixteenAndSeventeenAndEighteenAndNineteenAndTwentyAndTwentyOneAndTwentyTwoAndTwentyThreeAndTwentyFourAndTwentyFiveAndTwentySixAndTwentySevenAndTwentyEightAndTwentyNineAndThirtyAndThirtyOneAndThirtyTwoAndThirtyThreeAndThirtyFourAndThirtyFiveAndThirtySixAndThirtySevenAndThirtyEightAndThirtyNine": 0,
        "numTotalPartitionMovementsInExecutionAndPendingAndCancelledAndStoppedAndDeadAndFinishedAndAbortedAndFailedAndSkippedAndUnknownAndOtherAndNoneAndNullAndEmptyAndBlankAndZeroAndOneAndTwoAndThreeAndFourAndFiveAndSixAndSevenAndEightAndNineAndTenAndElevenAndTwelveAndThirteenAndFourteenAndFifteenAndSixteenAndSeventeenAndEighteenAndNineteenAndTwentyAndTwentyOneAndTwentyTwoAndTwentyThreeAndTwentyFourAndTwentyFiveAndTwentySixAndTwentySevenAndTwentyEightAndTwentyNineAndThirtyAndThirtyOneAndThirtyTwoAndThirtyThreeAndThirtyFourAndThirtyFiveAndThirtySixAndThirtySevenAndThirtyEightAndThirtyNineAndForty": 0,
        "numTotalPartitionMovementsInExecutionAndPendingAndCancelledAndStoppedAndDeadAndFinishedAndAbortedAndFailedAndSkippedAndUnknownAndOtherAndNoneAndNullAndEmptyAndBlankAndZeroAndOneAndTwoAndThreeAndFourAndFiveAndSixAndSevenAndEightAndNineAndTenAndElevenAndTwelveAndThirteenAndFourteenAndFifteenAndSixteenAndSeventeenAndEighteenAndNineteenAndTwentyAndTwentyOneAndTwentyTwoAndTwentyThreeAndTwentyFourAndTwentyFiveAndTwentySixAndTwentySevenAndTwentyEightAndTwentyNineAndThirtyAndThirtyOneAndThirtyTwoAndThirtyThreeAndThirtyFourAndThirtyFiveAndThirtySixAndThirtySevenAndThirtyEightAndThirtyNineAndFortyAndFortyOne": 0,
        "numTotalPartitionMovementsInExecutionAndPendingAndCancelledAndStoppedAndDeadAndFinishedAndAbortedAndFailedAndSkippedAndUnknownAndOtherAndNoneAndNullAndEmptyAndBlankAndZeroAndOneAndTwoAndThreeAndFourAndFiveAndSixAndSevenAndEightAndNineAndTenAndElevenAndTwelveAndThirteenAndFourteenAndFifteenAndSixteenAndSeventeenAndEighteenAndNineteenAndTwentyAndTwentyOneAndTwentyTwoAndTwentyThreeAndTwentyFourAndTwentyFiveAndTwentySixAndTwentySevenAndTwentyEightAndTwentyNineAndThirtyAndThirtyOneAndThirtyTwoAndThirtyThreeAndThirtyFourAndThirtyFiveAndThirtySixAndThirtySevenAndThirtyEightAndThirtyNineAndFortyAndFortyOneAndFortyTwo": 0,
        "numTotalPartitionMovementsInExecutionAndPendingAndCancelledAndStoppedAndDeadAndFinishedAndAbortedAndFailedAndSkippedAndUnknownAndOtherAndNoneAndNullAndEmptyAndBlankAndZeroAndOneAndTwoAndThreeAndFourAndFiveAndSixAndSevenAndEightAndNineAndTenAndElevenAndTwelveAndThirteenAndFourteenAndFifteenAndSixteenAndSeventeenAndEighteenAndNineteenAndTwentyAndTwentyOneAndTwentyTwoAndTwentyThreeAndTwentyFourAndTwentyFiveAndTwentySixAndTwentySevenAndTwentyEightAndTwentyNineAndThirtyAndThirtyOneAndThirtyTwoAndThirtyThreeAndThirtyFourAndThirtyFiveAndThirtySixAndThirtySevenAndThirtyEightAndThirtyNineAndFortyAndFortyOneAndFortyTwoAndFortyThree": 0,
        "numTotalPartitionMovementsInExecutionAndPendingAndCancelledAndStoppedAndDeadAndFinishedAndAbortedAndFailedAndSkippedAndUnknownAndOtherAndNoneAndNullAndEmptyAndBlankAndZeroAndOneAndTwoAndThreeAndFourAndFiveAndSixAndSevenAndEightAndNineAndTenAndElevenAndTwelveAndThirteenAndFourteenAndFifteenAndSixteenAndSeventeenAndEighteenAndNineteenAndTwentyAndTwentyOneAndTwentyTwoAndTwentyThreeAndTwentyFourAndTwentyFiveAndTwentySixAndTwentySevenAndTwentyEightAndTwentyNineAndThirtyAndThirtyOneAndThirtyTwoAndThirtyThreeAndThirtyFourAndThirtyFiveAndThirtySixAndThirtySevenAndThirtyEightAndThirtyNineAndFortyAndFortyOneAndFortyTwoAndFortyThreeAndFortyFour": 0,
        "numTotalPartitionMovementsInExecutionAndPendingAndCancelledAndStoppedAndDeadAndFinishedAndAbortedAndFailedAndSkippedAndUnknownAndOtherAndNoneAndNullAndEmptyAndBlankAndZeroAndOneAndTwoAndThreeAndFourAndFiveAndSixAndSevenAndEightAndNineAndTenAndElevenAndTwelveAndThirteenAndFourteenAndFifteenAndSixteenAndSeventeenAndEighteenAndNineteenAndTwentyAndTwentyOneAndTwentyTwoAndTwentyThreeAndTwentyFourAndTwentyFiveAndTwentySixAndTwentySevenAndTwentyEightAndTwentyNineAndThirtyAndThirtyOneAndThirtyTwoAndThirtyThreeAndThirtyFourAndThirtyFiveAndThirtySixAndThirtySevenAndThirtyEightAndThirtyNineAndFortyAndFortyOneAndFortyTwoAndFortyThreeAndFortyFourAndFortyFive": 0,
        "numTotalPartitionMovementsInExecutionAndPendingAndCancelledAndStoppedAndDeadAndFinishedAndAbortedAndFailedAndSkippedAndUnknownAndOtherAndNoneAndNullAndEmptyAndBlankAndZeroAndOneAndTwoAndThreeAndFourAndFiveAndSixAndSevenAndEightAndNineAndTenAndElevenAndTwelveAndThirteenAndFourteenAndFifteenAndSixteenAndSeventeenAndEighteenAndNineteenAndTwentyAndTwentyOneAndTwentyTwoAndTwentyThreeAndTwentyFourAndTwentyFiveAndTwentySixAndTwentySevenAndTwentyEightAndTwentyNineAndThirtyAndThirtyOneAndThirtyTwoAndThirtyThreeAndThirtyFourAndThirtyFiveAndThirtySixAndThirtySevenAndThirtyEightAndThirtyNineAndFortyAndFortyOneAndFortyTwoAndFortyThreeAndFortyFourAndFortyFiveAndFortySix": 0,
        "numTotalPartitionMovementsInExecutionAndPendingAndCancelledAndStoppedAndDeadAndFinishedAndAbortedAndFailedAndSkippedAndUnknownAndOtherAndNoneAndNullAndEmptyAndBlankAndZeroAndOneAndTwoAndThreeAndFourAndFiveAndSixAndSevenAndEightAndNineAndTenAndElevenAndTwelveAndThirteenAndFourteenAndFifteenAndSixteenAndSeventeenAndEighteenAndNineteenAndTw
```

Kafka installation in the KRaft mode

Paul Ionescu

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
- Configuration files
- Metadata storage

Overview

The Kafka service in the [RAD](#) model is installed as a regular service during [ADS cluster installation](#) in the ADCM interface using additional actions:

- [adding](#) an additional component **Kafka Controller** to create a controller quorum;
- setting the **Kafka quorum.voters** parameter to **localhost** during [configuration](#) of the Kafka service to create an environment for the KRaft mode (parameters, metadata storage, etc.).

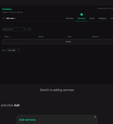
Below are the detailed steps to install the Kafka service with a description of additional actions.

CAUTION

- Installation of the Kafka service in KRaft mode is available starting from the [3.0.0.b2](#) ADS version.
- The KRaft protocol in Kafka is in technical preview and is not recommended for use in production environment. Available encourages you to explore this feature in non-production environments and provide feedback on your experience as appropriate.
- There is no migration from ZooKeeper to KRaft in Zookeeper.
- For an ADS cluster where you plan to install the Kafka service in KRaft mode, installing the ZooKeeper service is not required.

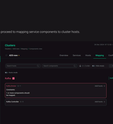
Step 1. Add the service

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



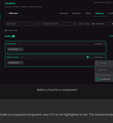
Switch to adding services

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



Select the service

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



The result of successful adding the service to the cluster

Step 2. Add components

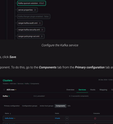
1. On the cluster page, open the **Mapping** tab to proceed to mapping service components to cluster hosts.

Switch to mapping service components

Kafka service components

Component	Description
Kafka Broker	The broker, which is responsible for storing partitions of topics to which clients connect to write or read messages
Kafka Controller	The broker, which is responsible for storing the cluster metadata topic. It is a participant in the controller quorum.

2. Assign a host to each component of the Kafka service – click **Add hosts** and select the desired host in the pop-up window.

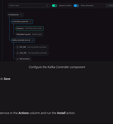


Select a host for a component

CAUTION

To install Kafka in KRaft mode, Kafka Controller is a required component, even if it is not highlighted in red. The recommended number of Kafka Controller instances per cluster is 3.

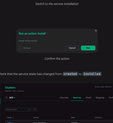
3. After the distribution of components is completed, click **Save**.



Save mapping of components

Step 3. Configure the service

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



Switch to the service configuration

2. Go to **Primary configuration** page that opens.
   - change the value of the **Kafka quorum.voters** parameter to **localhost**;
   - if necessary, fill in the remaining service configuration parameters – descriptions of the parameters are given in the **Kafka** section of the [ADS configuration parameters](#) article.

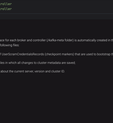
Configure the Kafka service

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

3. Switch to configure the Kafka Controller. To do this, go to the **Components** tab that is the **Primary configuration** tab and select the **Kafka Controller** component.

Switch to the Kafka Controller component configuration

4. In the window that opens, set the **Show advanced** switch to the active state and, if necessary, change the default values set for the configuration parameters of the Kafka Controller component – descriptions of the parameters are given under the heading "Kafka Controller component configuration parameters" in the **Kafka** section of the [ADS configuration parameters](#) article.

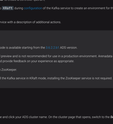


Configure the Kafka Controller component

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

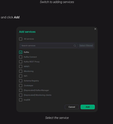
Step 4. Install the service

1. On the **Services** tab, click [] for the Kafka service in the **Actions** column and run the **Install** action.



Switch to the service installation

2. Confirm the action in the opened window.



Confirm the action

3. Wait until the installation is complete. Then check that the service state has changed from **created** → **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 **Job** list.

Install service job page

Step 5. View results

Configuration files

During Kafka installation in the KRaft mode, all the necessary parameters for working with [ADCM Ansible](#) are automatically installed via the ADCM interface in the configuration files. To view and control the service, files are created in the directories that were created during ADCM on the directory /etc/kafka.conf/, which display the parameters that were defined on the [Kafka Service page](#) in the corresponding section of the configuration parameters.

Kafka broker configuration file server.properties

```
node.id=1001
auto.create.topics.enable=false
listeners=PLAINTEXT://:9092
log.dirs=/data/kafka-logs
inter.broker.listener.name=test
num.partitions=3
delete.topic.enable=true
log.retention.hours=168
log.roll.hours=168
queued.max.requests=500
num.network.threads=3
num.io.threads=8
auto.leader.rebalance.enable=true
socket.send.buffer.bytes=102400
socket.receive.buffer.bytes=102400
transaction.state.log.replication.factor=1
transaction.state.log.min.isr=1
...
listener.security.protocol.map=PLAINTEXT:PLAINTEXT,CONTROLLER:PLAINTEXT

process.roles=controller
node.id=2000001
controller.listener.names=CONTROLLER
controller.quorum.voters=2000001@ads-ubu-test-1.ru-central1.internal:9093
log.dirs=/data/kafka-meta
controller.quorum.voters=2000001@ads-host-1.ru-central1.internal:9093
listener.security.protocol.map=CONTROLLER:PLAINTEXT

security.inter.broker.protocol=PLAINTEXT
```

Kafka Controller configuration file kafka-controller.properties

```
process.roles=controller
node.id=2000001
controller.listener.names=CONTROLLER
controller.quorum.voters=2000001@ads-host-1.ru-central1.internal:9093
log.dirs=/data/kafka-meta
controller.quorum.voters=2000001@ads-host-1.ru-central1.internal:9093
listener.security.protocol.map=CONTROLLER:PLAINTEXT

security.inter.broker.protocol=PLAINTEXT
```

File for specifying environment variables kafka-controller-env.sh

```
export LOG_DIR=/var/log/kafka-controller
export PID_DIR=/var/run/kafka-controller
```

Metadata storage

When [Kafka](#) starts in KRaft mode, a storage space for each broker and controller (Kafka name Kafka) is automatically created in the ADCM interface using the script /usr/lib/kafka/bin/kafka-storage.sh containing the following files:

- bootstrap.checkpoint – a file containing a list of LeaderAndIsrPartitionStateBundle (checkpoint metrics) that are used to bootstrap the cluster;
- „cluster_metadata/0" – a directory containing files in which all changes to cluster metadata are saved;
- meta.properties – a file containing information about the current server, version and cluster ID.

Kafka migration to KRaft

Olga Semina

Contents

To Table of Contents

Overview
Prepare for migration
- Check a cluster status
- Install Kafka controllers

Enable migration mode
Check data migration
Disable migration mode and set KRaft mode for Kafka

Overview

By default, in ADCM for an ADS cluster, metadata management of the Kafka service is performed via the ZooKeeper service, which provides quorum operation based on the ZAB consensus algorithm.

Also, Kafka service can be installed immediately in KRaft mode to use the Raft consensus algorithm.

For clusters where Kafka service is already used together with ZooKeeper, migration to KRaft mode can be performed.

CAUTION

- Kafka migration to KRaft mode is available starting from the 3.7.2.1.b1 ADS version.
- The KRaft protocol for Kafka is in technical preview and is not recommended for use in a production environment. Arenadata encourages you to explore this feature in non-production environments and provide feedback on your experience as appropriate.
- There is no possibility of reverse migration from KRaft to ZooKeeper.

The migration process is semi-automated and divided into steps:

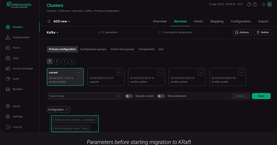
1. Prepare for migration
2. Enable migration mode
3. Check data migration
4. Disable migration mode and set KRaft mode for Kafka

Enabling and disabling migration mode is performed using the Kafka service actions in the ADCM interface, the start time of each of them is determined by the user.

Prepare for migration

Check a cluster status

Before enabling migration, check the values of the ***Kafka quorum solution*** and ***KRaft migration mode*** parameters on the Kafka service configuration page.



Parameters before starting migration to KRaft

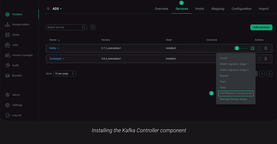
Below are instructions for actions with different parameter values.

Kafka quorum solution	KRaft migration mode	Directions for action
KRaft	false	The cluster is already in KRaft quorum mode. No further action is required
ZooKeeper	false	The cluster is in ZooKeeper quorum mode. The cluster is allowed to enable KRaft migration mode. Before enabling migration mode, install the Kafka Controller component
ZooKeeper	true	Migration mode is already enabled. Check data migration and proceed to disabling migration mode

Install Kafka controllers

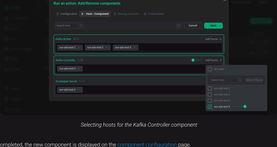
To switch to KRaft mode in a cluster, a new component must be installed for the Kafka service – Kafka Controller. The recommended number of ***Kafka Controller*** component instances per cluster is 3.

To add instances of a new Kafka service component in the ADCM interface, run the **Add/Remove components** action by clicking in the **Actions** column.



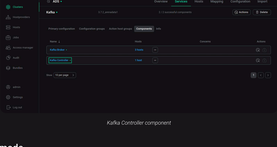
Installing the Kafka Controller component

Next, in the action confirmation form, select the parameter values and hosts for installing instances of the new component.



Selecting hosts for the Kafka Controller component

After the action is successfully completed, the new component is displayed on the component configuration page.



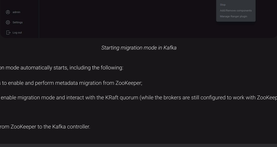
Kafka Controller component

Enable migration mode

To enable migration mode, ensure the following conditions are met:

- State of the Kafka service parameters:
  - ***Kafka quorum solution*** value is `ZooKeeper`.
  - ***KRaft migration mode*** value is `false`.
- The Kafka service has the ***Kafka Controller*** component installed as described above.

To start migration mode for the Kafka service in the ADCM interface, run the ***KRaft migration stage 1*** action by clicking in the **Actions** column.



Starting migration mode in Kafka

The process of enabling migration mode automatically starts, including the following:

- configuring Kafka controllers to enable and perform metadata migration from ZooKeeper;
- configuring Kafka brokers to enable migration mode and interact with the KRaft quorum (while the brokers are still configured to work with ZooKeeper);
- restarting Kafka brokers;
- migrating cluster metadata from ZooKeeper to the Kafka controller.

CAUTION

- After migration mode is enabled, Kafka controllers are fully functional and retrieve metadata from ZooKeeper, but Kafka brokers still use ZooKeeper as a quorum.
- Only proceed to disabling migration mode after verifying data migration, otherwise a cluster failure may occur.

Below is the state of the parameters after successfully enabling migration mode:

- ***Kafka quorum solution*** value is `ZooKeeper`.
- ***KRaft migration mode*** value is `true`.

Check data migration

The Kafka controller log will show the corresponding messages during the metadata migration.

All messages can be viewed by accessing the /var/log/kafka-controller/server.log file on the host where the ***Kafka Controller*** component is located.

Data migration is complete when the following entry appears in the log:

```
Completed migration of metadata from ZooKeeper to KRaft.
```

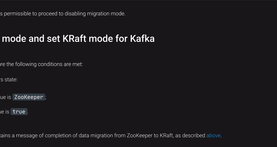
Only if this message appears, it is permissible to proceed to disabling migration mode.

Disable migration mode and set KRaft mode for Kafka

To disable migration mode, ensure the following conditions are met:

- The Kafka service parameters state:
  - ***Kafka quorum solution*** value is `ZooKeeper`.
  - ***KRaft migration mode*** value is `true`.
- The Kafka controller log contains a message of completion of data migration from ZooKeeper to KRaft, as described above.

To disable migration mode for the Kafka service, in the ADCM interface, run the ***KRaft migration stage 2*** action by clicking in the **Actions** column.



Disabling migration mode

The process of disabling migration mode automatically starts, including the following:

- configuring Kafka controllers to remove all ZooKeeper related properties and complete the migration;
- configuring Kafka brokers to point directly to the KRaft quorum instead of the ZooKeeper quorum, removing all ZooKeeper related properties, and completing the migration;
- restarting Kafka brokers and controllers.

Below is the state of the parameters after successfully disabling migration mode:

- ***Kafka quorum solution*** value is `KRaft`.
- ***KRaft migration mode*** value is `false`.

Configure and use Kafka Tiered Storage

Ivan Porto Carrero

Contents

To Table of Contents

Overview

Hadoop Distributed File System (HDFS)

- Prerequisites
- Step 1. Import ADH data
- Step 2. Configure and enable Tiered Storage
- Step 3. Check the results

Simple Storage Service (S3)

- Prerequisites
- Step 1. Configure and enable Tiered Storage
- Step 2. Check the results

Use Tiered Storage

Overview

The Tiered Storage option in ADS can be enabled in the configuration parameters of the Kafka service at a time for only one of two storage options: HDFS or S3.

Below are detailed steps to enable the Tiered Storage option.

Hadoop Distributed File System (HDFS)

For this example, the HDFS service of the Everyday Hadoop ADH cluster is used as storage.

Prerequisites

To enable the Tiered Storage option in Kafka, the following environment was used:

- The ADS cluster is installed according to the Online Installation guide. The minimum version of ADS is 3.0.1.2-b1.
- Kafka and Zookeeper services are installed in the ADS cluster.
- The ADH cluster is installed according to the Online Installation guide. The minimum version of ADH is 3.3.6.0-b1.
- HDFS, Core configuration, and the Gateway components are installed in the ADH cluster.
- In HDFS, you need to create a directory whose path and name matches that specified as the **storage.hdfs.root** configuration parameter and configure access rights:

```
$ sudo -u hdfs hdfs dfs -mkdir /kafka
$ sudo -u hdfs hdfs dfs -chown kafka:hadoop /kafka
```

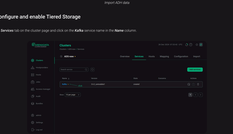
Step 1. Import ADH data

In the ADCM interface, open the *Clusters* page and click on the ADH cluster name. Then on the cluster page that opens, go to the *Import* tab, select *Cluster configuration* next to the ADH cluster name, and click *Import*.

Import ADH data

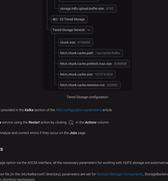
Step 2. Configure and enable Tiered Storage

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



Switch to the service configuration

2. In the *Primary configuration* window that opens:
   - Set the **HDFS Tiered Storage** switch to active and change the default settings as necessary.
   - If necessary, expand the *Tiered Storage General* group and change the default settings:



Tiered Storage configuration

Parameter descriptions are provided in the Kafka section of the ADS configuration parameters article.

3. Click *Save* and restart the Kafka service using the *Restart* action by clicking in the *Actions* column.
4. Wait until the service restarts. Analyze and correct errors if they occur on the *Jobs* page.

Step 3. Check the results

1. When you enable the Tiered Storage option on the ADCM interface, all the necessary parameters for working with HDFS storage are automatically set.
   - In the Kafka broker configuration file (in the /etc/kafka/conf directory), parameters are set for Remote Manager components, StorageAccident (defined automatically for the selected storage type), and the clustering mechanism.

     Below is an example of a configuration file modified by ADCM:

Kafka broker configuration file server.properties

```
# Managed by ADCM

node.id=1
reserved.broker.max.id=65000
auto.create.topics.enable=true
listeners=PLAINTEXT://:9092
log.dirs=/kafka/logs
default.replication.factor=1
min.insync.replicas=1
delete.topic.enable=true
log.retention.hours=168
log.roll.hours=168
queued.max.requests=500
num.io.threads=8
num.network.threads=3
auto.leader.rebalance.enable=true
controlled.shutdown.enable=true
offsets.topic.replication.factor=1
transaction.state.log.replication.factor=1
transaction.state.log.min.isr=1

zookeeper.connect=vos-adb-test-1.ru-central1.internal:2181/arenadata1/cluster/1a8
zookeeper.connection.timeout.ms=60000
zookeeper.session.timeout.ms=60000
zookeeper.sync.time.ms=2000
zookeeper.set.acl=false

log.cleaner.enable=true
log.cleanup.policy=delete
log.cleaner.threads=1
log.cleaner.min.compaction.lag.ms=0
log.cleaner.delete.retention.ms=86400000

security.inter.broker.protocol=PLAINTEXT


remote.log.metadata.manager.listener.name=PLAINTEXT

remote.log.storage.manager.class.name=io.arenadata.kafka.tieredstorage.RemoteStorageManager
remote.log.storage.manager.class.path=/usr/lib/kafka/libs/tiered-storage/*
remote.log.storage.system.enable=true
remote.log.storage.manager.impl.prefix=rsm.config.
rsm.config.fetch.chunk.cache.class=io.aiven.kafka.tieredstorage.fetch.cache.DiskChunkCache
rsm.config.fetch.chunk.cache.directory=/tmp
rsm.config.fetch.chunk.cache.prefetch.max.size=10000000
rsm.config.fetch.chunk.cache.size=1073741824
rsm.config.fetch.chunk.cache.retention.ms=60000
rsm.config.storage.backend.class.name=io.arenadata.kafka.tieredstorage.s3.S3Storage
rsm.config.storage.backend.class.path=/usr/lib/kafka/libs/tiered-storage/*:/usr/lib/hadoop/*:/usr/lib/hadoop/client/*
rsm.config.storage.s3.bucket.name=kafka
rsm.config.storage.aws.access.key.id=access_key
rsm.config.storage.aws.secret.access.key=secret_key
rsm.config.storage.s3.region=none
```


   - In the environment variables configuration file /etc/kafka/conf/kafka-env.sh set the value for the **HADOOP_CONF_DIR** parameter – the location where the ADH cluster configuration files are copied:

```
export HADOOP_CONF_DIR="/usr/lib/hadoop/config"
```

   - When the Tiered Storage option is enabled, the ADH configuration files are copied to the /usr/lib/kafka/config/ folder:
     - core-site.xml
     - hdfs-site.xml

Simple Storage Service (S3)

For this example, the MINIO storage is used as the S3 server.

Prerequisites

To enable the Tiered Storage option in Kafka, the following environment was used:

- The ADS cluster is installed according to the Online Installation guide. The minimum version of ADS is 3.0.1.2-b1.
- Kafka and Zookeeper services are installed in the ADS cluster.
- A bucket for storing data was created in the MINIO cloud storage.

NOTE

In the Tiered Storage implementation based on the S3 server, the transfer of records to the remote level is controlled using a special **__remote_log_metadata** topic, which by default has a replication factor value of **3**. This requires at least three Kafka brokers in the cluster for the topic to function properly. If the number of brokers in the cluster is less than three, change the replication factor for the **__remote_log_metadata** topic. This can be done, e.g., by changing the value in the **server.properties** group.

```
rlm.config.remote.log.metadata.topic.replication.factor=1
```

Step 1. Configure and enable Tiered Storage

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

Switch to the service configuration

2. In the *Primary configuration* window that opens:
   - The S3 server's remote title for this bucket is specified not in the ITSON format, but as a path (for example, for MINIO) the link to the bucket looks like this: http://s3clusternamp.ynh-ibcrawen-local:minio1, set the following parameters: *storage.s3.path.style.access.enabled* parameter value to *true*, *storage.aws.endpoint.url* parameter value to *http://s3clusternamp.ynh-ibcrawen-local:minio1*.

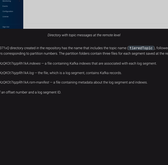
```
rsm.config.storage.s3.path.style.access.enabled=true
```

   To do this, expand the **server.properties** group and, using the **Add key/value** field, select *Add property* and enter the name of the parameter and its value.
   - Set the **S3 Tiered Storage** switch to active and change the parameter values for connecting to S3 storage.

NOTE

The value of the *storage.s3.region* parameter cannot be empty. If the S3 server does not provide such a parameter, set it to any value, for example, *none*.

   - If necessary, expand the *Tiered Storage General* group and change the default settings:



Tiered Storage configuration

Parameter descriptions are provided in the Kafka section of the ADS configuration parameters article.

3. Click *Save* and restart the Kafka service using the *Restart* action by clicking in the *Actions* column.
4. Wait until the service restarts. Analyze and correct errors if they occur on the *Jobs* page.

Step 2. Check the results

When you enable the Tiered Storage option via the ADCM interface, in the Kafka broker configuration file (in the /etc/kafka/conf/ directory), all the necessary parameters for working with S3 storage are automatically set: parameters of the Remote Manager components, StorageAccident (defined automatically for the selected storage type), and the clustering mechanism.

Below is an example of a configuration file modified by ADCM:

Kafka broker configuration file server.properties

```
# Managed by ADCM

node.id=1
reserved.broker.max.id=65000
auto.create.topics.enable=true
listeners=PLAINTEXT://:9092
log.dirs=/kafka/logs
default.replication.factor=1
min.insync.replicas=1
delete.topic.enable=true
log.retention.hours=168
log.roll.hours=168
queued.max.requests=500
num.io.threads=8
num.network.threads=3
auto.leader.rebalance.enable=true
controlled.shutdown.enable=true
offsets.topic.replication.factor=1
transaction.state.log.replication.factor=1
transaction.state.log.min.isr=1

zookeeper.connect=vos-adb-test-1.ru-central1.internal:2181/arenadata1/cluster/1b4
zookeeper.connection.timeout.ms=60000
zookeeper.session.timeout.ms=60000
zookeeper.sync.time.ms=2000
zookeeper.set.acl=false

log.cleaner.enable=true
log.cleanup.policy=delete
log.cleaner.threads=1
log.cleaner.min.compaction.lag.ms=0
log.cleaner.delete.retention.ms=86400000

security.inter.broker.protocol=PLAINTEXT


remote.log.metadata.manager.listener.name=PLAINTEXT

remote.log.storage.manager.class.name=io.aiven.kafka.tieredstorage.RemoteStorageManager
remote.log.storage.manager.class.path=/usr/lib/kafka/libs/tiered-storage/*
remote.log.storage.system.enable=true
rsm.config.fetch.chunk.cache.class=io.aiven.kafka.tieredstorage.fetch.cache.DiskChunkCache
rsm.config.fetch.chunk.cache.directory=/tmp
rsm.config.fetch.chunk.cache.prefetch.max.size=10000000
rsm.config.fetch.chunk.cache.size=1073741824
rsm.config.fetch.chunk.cache.retention.ms=60000
rsm.config.storage.backend.class.name=io.arenadata.kafka.tieredstorage.s3.S3Storage
rsm.config.storage.backend.class.path=/usr/lib/kafka/libs/tiered-storage/*:/usr/lib/hadoop/*:/usr/lib/hadoop/client/*
rsm.config.storage.s3.bucket.name=kafka
rsm.config.storage.aws.access.key.id=access_key
rsm.config.storage.aws.secret.access.key=secret_key
rsm.config.storage.s3.region=none
```


Use Tiered Storage

When creating a topic whose segments are to be moved to storage, you should enable the Tiered Storage option for this topic using the remote.storage.enable parameter. If the local.retention.ms parameter is not specified for the topic, the local data storage time corresponds to the time specified for retention.

Below is an example of a topic creation command (for partitions for hot storage and warm and another one):

```
$ /usr/lib/kafka/bin/kafka-topics.sh --create --topic tieredTopic --bootstrap-server adb-test-2.ru-central1.internal:9092 \
--config remote.storage.enable=true \
--config local.retention.ms=1000 \
--config retention.ms=604800000 \
--config segment.bytes=200
```

After manual rebalancing, if the local storage time has expired and the segment has reached the specified size, the file appears in the HDFS storage directory. These messages can be found in the server.

The figure below shows a directory with topic segments at the remote level in the MINIO storage bucket.

Directory with topic messages at the remote level

The kenotFqvo-ElgbaenTUwhgcbVYI1FUV directory created in the repository has the name that includes the topic name (**tieredTopic**), followed by a dash and the topic identifier. This directory, in turn, contains folders corresponding to partitions numbers. The partition folders contain three files for each segment stored at the remote level:

- 00000000000000000000-sdRv-QSQCHCQeOY1SqoWA.log.gz – a file containing a Kafka's internal data that are associated with each log segment.
- 00000000000000000000-sdRv-QSQCHCQeOY1SqoWA.log.gz.indexes – the file, which is a log segment, contains Kafka records.
- 00000000000000000000-sdRv-QSQCHCQeOY1SqoWA.log.gz.rsm-manifest – a file containing metadata about the log segment and indexes.

Such files have a name consisting of an offset number and a log segment ID.

Mirror Maker 2

Olga Semina

Contents

To Table of Contents

Overview of replication based on Kafka Connect
Setting
Topic replication check

Overview of replication based on Kafka Connect

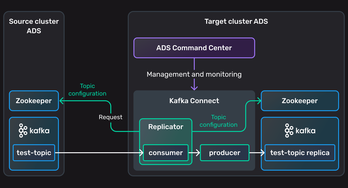
MirrorMaker 2.0 – mechanism for replicating data from a source cluster to a remote one, based on the Kafka Connect service platform.

Kafka Connect is available for installation in ADS starting from version 1.7.1.

With the **Active/Active** replication architecture, clusters receive data directly from data sources, and replicated data entered from a remote cluster.

With **Active/Standby** replication architecture target cluster is in passive mode (has no consumers and producers connected to it) and receives only replicated data.

The figure shows the sequence of the replication mechanism.



Replication scheme using Mirror Maker 2

To move data, a high-level pair **consumer - producer** is used. The consumer reads and processes the original topic, the producer places the data in the replica topic.

Replicator – a special component of the Kafka Connect service that accesses ZooKeeper data, copies the topic metadata, and creates the same topic in the target cluster.

MirrorMaker 2.0 as part of Kafka Connect creates special connectors to enable complex flows between ADS clusters:

- **MirrorSourceConnector** replicates topics from a source cluster to a target cluster.
- **MirrorCheckpointConnector** creates consumer offset checkpoints and synchronizes the offset with the `__consumer_offsets` service topic of the source cluster.
- **MirrorHeartbeatConnector** periodically checks connectivity between clusters by creating messages in a special `heartbeats` topic on the source cluster after a given period of time and reading them in the target cluster.

NOTE

This article describes how to replicate topics using the built-in Mirror Maker 2 tools in Kafka. Setting up Mirror Maker 2 using ADS Control is described in Mirror Maker 2 in ADS Control.

Setting

A topic is replicated to a remote cluster using the *connect-mirror-maker.sh* script from any host of the target ADS cluster.

The *connect-mirror-maker.sh* script creates and configures connectors based on the provided properties file *mm2.properties*.

To set up the *mm2.properties* configuration file, do the following:

1. Enter the command to open (create) the file:

```
$ sudo vim /usr/lib/kafka/bin/mm2.properties
```

2. Fill in the file with data:

```
clusters = <source_cluster_alias>, <target_cluster_alias>
<target_cluster_alias>.bootstrap.servers = hostname1:9092,hostname2:9092,hostname3:9092
<source_cluster_alias>.bootstrap.servers = hostname4:9092,hostname5:9092,hostname6:9092

topics
topics =<test_topic_name>
groups = *

<source_cluster_alias>->target_cluster_alias>.enabled=true

replication.factor=3

checkpoints.topic.replication.factor=3
heartbeats.topic.replication.factor=3
offset-syncs.topic.replication.factor=3

offset.storage.replication.factor=3
status.storage.replication.factor=3
config.storage.replication.factor=3
```

mm2.properties contents description

Parameters	Description
clusters = <source_cluster_alias>, <target_cluster_alias>	Names for each cluster to be used in the replication, where (here and below): ▪ `<source_cluster_alias>` – cluster that is the source of data; ▪ `<target_cluster_alias>` – cluster where the data should be moved (target).
<source_cluster_alias>.bootstrap.servers = hostname1:9092,hostname2:9092,hostname3:9092 <target_cluster_alias>.bootstrap.servers = hostname4:9092,hostname5:9092,hostname6:9092	Connection information for each cluster — comma-separated `host:port 9092` pairs
<source_cluster_alias>.→<target_cluster_alias>.enabled = true	Enabling and designating the direction of replication. In the example, replication is enabled from the `<source_cluster_alias>` cluster to the `<target_cluster_alias>` cluster
topics topics = <test_topic_name> groups = *	Name of the topics or groups intended for replication, where `<test_topic_name>` is the name of one or more topics, separated by commas
replication.factor=1	Replication ratio of newly created replica topics
heartbeats.topic.replication.factor=3 checkpoints.topic.replication.factor=3 offset-syncs.topic.replication.factor=3	Replication factor for MM2 service topics: ▪ `heartbeats` – heartbeat topic in each source cluster that is replicated to demonstrate connectivity through connectors. The messages in this topic contain information about the source cluster, target cluster, and the timestamp of the heartbeat. ▪ `source.checkpoints.internal` – topic in target cluster to record offsets for each consumer group in source cluster. ▪ `mm2-offset-syncs.<target_cluster_alias>.internal` – populates the MirrorSourceConnector and then uses the MirrorCheckpointConnector to change consumer group offsets.
offset.storage.replication.factor=3 status.storage.replication.factor=3 config.storage.replication.factor=3	Replication factor for creating internal cluster topics: ▪ `mm2-configs.<target_cluster_alias>.internal` ▪ `mm2-offsets.<target_cluster_alias>.internal` ▪ `mm2-status.<target_cluster_alias>.internal`
replication.policy.separator = _	If the value is left blank, the replica topic will have the same name as the original topic
sync.topic.acls.enabled = false	Enable monitoring of the source cluster for ACL changes
emit.heartbeats.interval.seconds = 5	Interval for heartbeat messages

Topic replication check

In order to run the *connect-mirror-maker.sh* script and create connectors, enter the following command on any broker of the target cluster:

```
$ /usr/lib/kafka/bin/connect-mirror-maker.sh mm2.properties
```

Next, connectors are launched that cyclically poll the source topic to read messages. This terminal session may not be terminated, then the messages can be read in real time.

During the operation of the connectors, internal cluster topics are created.

- on the side of the source cluster:
  - mm2-configs.<target_cluster_alias>.internal
  - mm2-offset-syncs.<target_cluster_alias>.internal
  - mm2-offsets.<target_cluster_alias>.internal
  - mm2-status.<target_cluster_alias>.internal
    where `<target_cluster_alias>` corresponds to the target cluster alias entered when *mm2.properties* was created.
- on the side of the target cluster:
  - mm2-configs.<source_cluster_alias>.internal
  - mm2-offset-syncs.<source_cluster_alias>.internal
  - mm2-offsets.<source_cluster_alias>.internal
  - mm2-status.<source_cluster_alias>.internal
    where `<source_cluster_alias>` corresponds to the source cluster alias entered when creating *mm2.properties*.

In order to check the presence of service topics, you must enter the command on any cluster broker (in a new terminal session) to display a list of topics:

```
$ /usr/lib/kafka/bin/kafka-topics.sh --list --bootstrap-server hostname:9092
```

The list of topics should contain the necessary topics.

The creation of a replicated source topic and the writing of messages to it occurs on the side of the source cluster.

To create an source topic, enter the command:

```
$ /usr/lib/kafka/bin/kafka-topics.sh --create --topic <test_topic_name> --bootstrap-server hostname:9092
```

where `<test_topic_name>` – the name of the source topic, as entered when creating *mm2.properties*.

To write messages to the original topic, enter the command:

```
$ /usr/lib/kafka/bin/kafka-console-producer.sh --topic <test_topic_name> --bootstrap-server hostname:9092
```

After writing messages to the source topic in the list of topics on the target cluster, a replica topic appears with the name in the format `<source_cluster_alias>.<test_topic_name>`, where:

- `<source_cluster_alias>` is the corresponding source cluster alias entered when creating *mm2.properties*.
- `<test_topic_name>` is the name of the original topic, entered when creating *mm2.properties*.

In order to read messages from the replica topic on the target cluster, enter the command:

```
$ /usr/lib/kafka/bin/kafka-console-consumer.sh --topic <source_cluster_alias>.<test_topic_name> --from-beginning --bootstrap-server hostname:9092
```

Read messages in the target cluster completely repeat those written in the source cluster.

Kafka REST Proxy usage example

Olga Semme

Contents

To Table of Contents

Write messages to a topic

Read messages from a topic

Get data on topics

- List of topics
- Topic configuration parameters
- Partitions

The article describes an example of using the Kafka REST Proxy service — working with Kafka topics over the RESTful interface using the command line tool cURL.

Write messages to a topic

To write to a topic, use the request form POST /topics/{string:topic_name}.

The query to write the `{"name":"Helen"}` key/value pair in JSON format to the `my_topic` topic looks like this:

```
$ curl -X POST -H "Content-Type: application/vnd.kafka.json.v2+json" \
      --data '{"records":[{"value":{"name":"Helen"}}]}' http://hostname:8082/topics/my_topic
```

where:

- `X POST` — defines an HTTP request method that is used to send data to the server.
- `H "Content-Type: application/vnd.kafka.json.v2+json"` — defines a custom header that describes the content-type of the request body.
- `data '{"records":[{"value":{"name":"Helen"}}]}'` — data that describes the request body. The body of the request contains the information needed to create or modify the object.
- `http://hostname:8082/topics/my_topic` — endpoint.
- `hostname` — hostname where the Kafka REST Proxy service is installed.

NOTE

The queries below have components similar to the query for writing messages to a topic.

Query result on successful message recording:

```
{"offsets":[{"partition":0,"offset":0,"error_code":null,"error":null}],"key_schema_id":null,"value_schema_id":null}
```

CAUTION

When writing messages to a topic, keep the following in mind:

- To automatically create a topic, you must enable the ***auto.create.topics.enable*** parameter in the ***server.properties*** group when configuring the Kafka service.
- The topic name can include alphanumeric characters (including ASCII), as well as the characters `.`, `_`, and `-`. Any other characters will generate an error.

Read messages from a topic

To read a message from a topic, do the following:

1. Create a consumer. To do this, use the request form POST /consumers/{string:group_name}.
   The request to create a `my_consumer` consumer in the `my_group` consumer group to read JSON-formatted data, starting from the beginning of the topic, looks like this:

```
$ curl -X POST -H "Content-Type: application/vnd.kafka.v2+json" \
      --data '{"name": "my_consumer", "format": "json", "auto.offset.reset": "earliest"}' \
      http://hostname:8082/consumers/my_group
```

   Query result upon successful consumer creation:

```
{"instance_id":"my_consumer","base_uri":"http://hostname:8082/consumers/my_group/instances/my_consumer"}
```

2. Subscribe to the topic. To do this, use the request form POST / consumers/{string:group_name}/instances/{string:instance}/subscription.
   To subscribe to a topic, you must use the base URL `base_uri` obtained from the request to create a consumer.
   The request to subscribe to the `my_topic` topic for the `my_consumer` consumer in the `my_group` group looks like this:

```
$ curl -X POST -H "Content-Type: application/vnd.kafka.v2+json" --data '{"topics":["my_topic"]}' \
 http://hostname:8082/consumers/my_group/instances/my_consumer/subscription
```

3. Read messages from a topic. To do this, use the request form GET /consumers/{string:group_name}/instances/{string:instance}/records.
   The request for the `my_consumer` consumer from the `my_group` group to read messages from the topic to which the subscription was connected looks like this:

```
$ curl -X GET -H "Accept: application/vnd.kafka.json.v2+json" \
      http://hostname:8082/consumers/my_group/instances/my_consumer/records
```

   where `H "Accept: application/vnd.kafka.json.v2+json"` — defines a custom header indicating content type that the client can accept.

   Query result on successful reading of messages:

```
[{"topic":"my_topic2","key":null,"value":{"name":"Helen"},"partition":0,"offset":0}]
```

   Read a `{"name":"Helen"}` key/value pair in JSON format that was written to the topic.

4. You can close a consumer using the request form DELETE / consumers/{string:group_name}/instances/{string:instance}.
   The request to close the `my_consumer` consumer from the `my_group` group looks like this:

```
$ curl -X DELETE -H "Content-Type: application/vnd.kafka.v2+json" \
      http://hostname:8082/consumers/my_group/instances/my_consumer
```

   The consumer has been deleted and can no longer read messages.

Get data on topics

List of topics

To get a list of topics, use the request form GET /topics.

A request to get a list of ADS cluster topics with a Kafka REST Proxy service installed on the `hostname` host looks like this:

```
$ curl -X GET http://hostname:8082/topics/
```

The query result displays all topics that exist in the cluster:

```
["my_topic","_schemas"]
```

Topic configuration parameters

To get data on the configuration parameters of a topic, use the request form GET /topics/{string: topic_name}.

The request to get the parameters of the `my_topic` topic looks like this:

```
$ curl -X GET http://hostname:8082/topics/my_topic
```

The query result displays all parameters of the requested topic:

```
{"name":"my_topic","configs":{"compression.type":"producer","leader.replication.throttled.replicas":"","message.downconversion.
enable":"true","min.insync.replicas":"1","segment.jitter.ms":"0","cleanup.policy":"delete","flush.
ms":"9223372036854775807","follower.replication.throttled.replicas":"","segment.bytes":"1073741824","retention.
ms":"604800000","flush.messages":"9223372036854775807","message.format.version":"2.8-IV1","max.compaction.lag.
ms":"9223372036854775807","file.delete.delay.ms":"60000","max.message.bytes":"1048588","min.compaction.lag.ms":"0","message.
timestamp.type":"CreateTime","preallocate":"false","min.cleanable.dirty.ratio":"0.5","index.interval.bytes":"4096","unclean.leader.
election.enable":"false","retention.bytes":"-1","delete.retention.ms":"86400000","segment.ms":"604800000","message.timestamp.
difference.max.ms":"9223372036854775807","segment.index.bytes":"10485760"},"partitions":[{"partition":0,"leader":1001,"replicas":
[{"broker":1001,"leader":true,"in_sync":true}]}]}
```

Partitions

To get information about topic partitions, use the form GET /topics/{string:topic_name}/partitions.

The request to get information about the partitions of the `my_topic` topic looks like this:

```
$ curl -X GET http://hostname:8082/topics/my_topic/partitions
```

The query result displays information about topic partitions:

```
[{"partition":0,"leader":1001,"replicas":[{"broker":1001,"leader":true,"in_sync":true}]}]
```

Work with kafka-rest-security

Olga Semme

Contents

To Table of Contents

Overview
Prerequisites
Enable the kafka-rest-security module
Example of working with an authenticated user

Overview

By default, a Kafka REST Proxy user can only authenticate to Kafka with one predefined principal created when Kerberos is enabled.

The `kafka-rest-security` module provides the ability to authenticate in Kafka with different principals when connecting to Kafka via Kafka REST Proxy.

The request is filtered through the `kafka-rest-security` module to obtain the username and match it against a list of predefined users for Kafka authentication.

If the Kafka REST Proxy username matches one of the Kafka users, a new security context is created using that principal and used for authentication to Kafka.

Prerequisites

To enable the `kafka-rest-security` module, ensure the following conditions are met:

- The ADS cluster has authentication enabled using the Kerberos protocol with any of the authentication systems:
  - MIT KDC
  - Active Directory
  - FreeIPA
  - Samba
- A user has been created in the authentication system with a principal and a keytab file for authentication in Kafka.
- The ADS cluster has authentication enabled for the Kafka REST Proxy service, for example, via the basic authentication or using a *jaas* file, a user has been created and a role has been assigned to it.

Enable the kafka-rest-security module

To enable the module on the configuration page of the Kafka REST Proxy service, do the following:

1. Set the ***Show advanced*** switch to active.
2. Open the ***JAAS template*** template, where you specify user data in the ***KafkaClient*** section:
   - ***userNames*** — an option for recording the username to map the Kerberos principal with the kafka-rest principal. The username must match the name entered when creating a new user during basic authentication. Multiple usernames can be specified, separated by a space.
   - ***keyTab*** — an option to specify the Kerberos principal's keytab file.
   - ***principal*** — an option to specify the Kerberos principal.

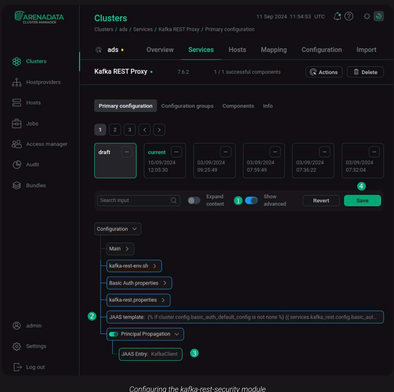
   For each user, the entire section must be specified. For example, below are two entries for a user, the first of which matches the Kerberos principal parameters for the Kafka REST Proxy user `username`, and the second of which does not:

```
KafkaClient {
    com.sun.security.auth.module.Krb5LoginModule required
    useKeyTab=true
    storeKey=true
    useTicketCache=false
    serviceName="kafka"
    userNames="username"
    keyTab="/etc/security/keytabs/kafka-rest.service.keytab"
    principal="kafka-rest/<hostname>@<realm>";

    com.sun.security.auth.module.Krb5LoginModule required
    useKeyTab=true
    storeKey=true
    useTicketCache=false
    serviceName="kafka"
    keyTab="/etc/security/keytabs/kafka.service.keytab"
    principal="kafka/<hostname>@<realm>";
};
```

NOTE

- For more information about the ***JAAS template*** template, see Configure a custom jaas.conf.
- The following are out-of-the-box authentication methods that can be configured in a JAAS file and can be used with the `kafka-rest-security` module for mapping Kafka REST Proxy principals (usernames) to Kafka principals:
  - JDBCLoginModule
  - PropertyFileLoginModule
  - DataSourceLoginModule
  - LdapLoginModule
- mTLS authentication can be used in conjunction with the above methods. This authentication method cannot be used alone because it does not provide REST principal information.

3. Set the ***Principal Propagation*** switch to active. In the ***JAAS Entry*** field, specify the section of the ***JAAS template*** user file, which contains the data of new users.
4. Save the settings by clicking ***Save***.



Configuring the kafka-rest-security module

After changing the settings using the ADCM interface, restart the Kafka REST Proxy service. To do this, apply the Restart action by clicking in the ***Actions*** column.

Make sure that the following parameters appear in the */etc/kafka-rest/kafka-rest.properties* configuration file:

```
kafka.rest.resource.extension.class=io.arenadata.kafkarest.security.KafkaRestSecurityResourceExtension
kafka.rest.auth.impersonation.enabled=true
kafka.rest.auth.impersonation.jaas.entry=KafkaClient
```

where:

- `kafka.rest.resource.extension.class` — a list of classes that will be used to implement user resources;
- `kafka.rest.auth.impersonation.enabled` — enables user name recognition in Kafka REST Proxy;
- `kafka.rest.auth.impersonation.jaas.entry` — specifies the JAAS file section for name resolution.

The mTLS authentication method is enabled by adding a custom property parameter in the *kafka-rest.properties* file (via the ***Add key, value*** field on the configuration page of the Kafka REST Proxy service):

- ***Enter field name*** — `ssl.client.authentication`
- ***Enter field value*** — `REQUIRED`

CAUTION

For Kafka REST, you should configure SSL before enabling mTLS authentication.

mTLS needs to be combined with one of the JAAS login modules for use with `kafka-rest-security`.

List of ready-made modules that can be configured in the JAAS template:

- org.eclipse.jetty.security.jaas.spi.JDBCLoginModule
- org.eclipse.jetty.security.jaas.spi.PropertyFileLoginModule
- org.eclipse.jetty.security.jaas.spi.DataSourceLoginModule
- org.eclipse.jetty.security.jaas.ldap.LdapLoginModule

Example of working with an authenticated user

Below is a request to write the key/value pair `{"foo":"bar"}` to the `test` topic with user authentication, where `username:password` is the username and password entered when creating a new user during basic authentication:

```
$ curl -u username:password -X POST -H "Content-Type: application/vnd.kafka.json.v2+json" --data '{"records":[{"value":{"foo":"bar"}}]}' "http://hostname:8082/topics/test"
```

NOTE

More information about working with Kafka REST Proxy using cURL is described in the Kafka REST Proxy usage example article.

KsqlDB usage example

By Ilgiz Gilmullin

Contents

Table Of Contents

Write events to a Kafka topic

Start ksqlDB

Create a stream from a Kafka topic

Stream data management

- View data
- Count events
- Select by name

Write from ksqlDB to a Kafka topic

The article describes an example of managing event data on a site (for example Wikipedia) using the ksqlDB service

Write events to a Kafka topic

To write to Kafka the last 10 events from the site, you can use a Python script that:

1. Sets the configuration options for the Kafka producer.
2. Sets the configuration parameters to work with the Wikipedia API.
3. Creates a Kafka producer.
4. Gets the latest changes from the Wikipedia API (data about the last 10 edited articles).
5. Writes the events to a Kafka topic in the JSON format.
6. Closes the Kafka producer

> **NOTE**
>
> To work with this Python script, you need first install the packages:
>
> - requests
> - kafka-python

To write events to a Kafka topic, you need to:

1. Create a script file

```
$ sudo vim /tmp/events.py
```

2. Fill events.py with the script text below, replacing individual values:

- `bootstrap_servers` - hostname to connect to the Kafka server;
- `topic_name` – name of the topic to write events.

events.py

```
import requests
from kafka import KafkaProducer
import json

# Kafka producer configuration
bootstrap_servers = 'ksw-test-s.ru-central1.internal:9092'
topic_name = 'wikipedia-topic'

# Wikipedia API configuration
api_url = 'https://en.wikipedia.org/w/api.php'
params = {
    'action': 'query',
    'format': 'json',
    'list': 'recentchanges',
    'rcprop': 'user|title|timestamp',
    'rclimit': '10'
}

# Producer initialization
producer = KafkaProducer(bootstrap_servers=bootstrap_servers,
                         value_serializer=lambda v: json.dumps(v).encode('utf-8'))

# Fetch recent changes from Wikipedia and push messages to Kafka
response = requests.get(api_url, params=params)
data = response.json()
recent_changes = data['query']['recentchanges']

for change in recent_changes:
    event = {
        'user_id': change['user'],
        'page_title': change['title'],
        'event_type': 'edit',
        'timestamp': change['timestamp']
    }
    producer.send(topic_name, value=event)
    print("Produced event:", event)

# Close the Kafka producer
producer.close()
```

3. Give permission to execute the script:

```
$ sudo chmod +x /tmp/events.py
```

4. Run the script:

```
$ python /tmp/events.py
```

As a result of the correct execution of the script, events are displayed in the JSON format.

Example of recorded events from the site

```
{'Produced event:', {'page_title': u'User:190.205.61.245', 'user_id': u'ST47ProxyBot', 'event_type': 'edit', 'timestamp': u'2023-05-31T15:17:46Z'}}
{'Produced event:', {'page_title': u'User:178.221.183.209', 'user_id': u'ST47ProxyBot', 'event_type': 'edit', 'timestamp': u'2023-05-31T15:17:46Z'}}
{'Produced event:', {'page_title': u'Carlos Alcaraz career statistics', 'user_id': u'Dobmeliaan', 'event_type': 'edit', 'timestamp': u'2023-05-31T15:17:47Z'}}
{'Produced event:', {'page_title': u'User:179.6.164.242', 'user_id': u'ST47ProxyBot', 'event_type': 'edit', 'timestamp': u'2023-05-31T15:17:47Z'}}
{'Produced event:', {'page_title': u'User:45.250.251.198', 'user_id': u'ST47ProxyBot', 'event_type': 'edit', 'timestamp': u'2023-05-31T15:17:47Z'}}
{'Produced event:', {'page_title': u'User:ial 8.47 7b', 'user_id': u'ST47ProxyBot', 'event_type': 'edit', 'timestamp': u'2023-05-31T15:17:47Z'}}
{'Produced event:', {'page_title': u'Vera Farmiga on screen and stage', 'user_id': u'ChaitanyaJu', 'event_type': 'edit', 'timestamp': u'2023-05-31T15:17:44Z'}}
{'Produced event:', {'page_title': u'Personal health record', 'user_id': u'John of Reading', 'event_type': 'edit', 'timestamp': u'2023-05-31T15:17:44Z'}}
{'Produced event:', {'page_title': u'User:189.204.188.128', 'user_id': u'ST47ProxyBot', 'event_type': 'edit', 'timestamp': u'2023-05-31T15:17:44Z'}}
{'Produced event:', {'page_title': u'User:TechTajm', 'user_id': u'TechTajm', 'event_type': 'edit', 'timestamp': u'2023-05-31T15:17:44Z'}}
```

To view data written to a Kafka topic, you can read data from a Kafka topic using the command line

Start ksqlDB

After installing ksqlDB, the service is started using the command:

```
$ sudo ksql http://<hostname>:8088
```

where `<hostname>` is the name of the host where the ksqlDB service is installed.

Server started:

```
                  ===========================================
                  =       _              _ ____  ____       =
                  =      | | _____  __ _| |  _ \| __ )      =
                  =      | |/ / __|/ _` | | | | |  _ \      =
                  =      |   <\__ \ (_| | | |_| | |_) |     =
                  =      |_|\_\___/\__, |_|____/|____/      =
                  =                   |_|                   =
                  =  Event Streaming Database purpose-built =
                  =        for stream processing apps       =
                  ===========================================

Copyright 2017-2021 Confluent Inc.

CLI v7.2.1, Server v6.2.1 located at http://ksw-test-s.ru-central1.internal:8088
Server Status: RUNNING

Having trouble? Type 'help' (case-insensitive) for a rundown of how things work!

ksql>
```

The ksqlDB functions are invoked in the `ksql>` line.

Create a stream from a Kafka topic

To manipulate data written to a Kafka topic using ksqlDB, you need to create a stream.
A stream is a partitioned, immutable, append-only collection that represents a set of facts about events or records.
The request for creating a `wikipedia_events` stream for the Kafka topic `wikipedia-topic` is as follows:

```
CREATE STREAM wikipedia_events (\
    user_id VARCHAR,\
    page_title VARCHAR,\
    event_type VARCHAR,\
    timestamp VARCHAR)\
WITH (\
    kafka_topic='wikipedia-topic',\
    value_format='json',\
    key_format='kinds');
```

where are defined:

- names of the stream columns that match the keys of the object described in the topic and the SQL data types that will be placed in columns;
- stream properties.

Stream is created:

```
 Message
----------------
 Stream created
----------------
```

Stream data management

With various queries you can retrieve data from the created stream.

> **NOTE**
>
> In order for a Kafka topic to be read from the beginning, and not from the last offset, you can set a parameter on the work line that determines from which offset you want to read topics:
>
> ```
> SET 'auto.offset.reset' = 'earliest';
> ```
>
> Without this setting, to work out the examples below, after running the ksqlDB function, it is necessary to overwrite the data in the topic used in the thread (for example, re-execute the script described by above).

View data

You can view the entire contents of the `wikipedia_events` stream via a query using the `SELECT` operator without specifying selection conditions:

```
SELECT * FROM wikipedia_events EMIT CHANGES;
```

where `EMIT CHANGES` indicates that any changes that occur should be emitted.

The query result is shown below:

```
+-----------------+-----------------+-----------------+-----------------+
| USER_ID         | PAGE_TITLE      | EVENT_TYPE      | TIMESTAMP       |
+-----------------+-----------------+-----------------+-----------------+
| JM_Amanda       | Idola Quadros   | [edit           | [2023-05-29T18:56:22Z |
| Deprosiadrian   | Everything It Took (Rates of Stylelit | [edit | [2023-05-29T18:56:20Z |
|                 | the album)      |                 |                 |
| 139.137.128.5   | Victor Koko     | [edit           | [2023-05-29T18:56:20Z |
| Dbvoiceactor    | Vivorre Research| [edit           | [2023-05-29T18:56:42Z |
| Drunkatsalamihere | IMS 98        | [edit           | [2023-05-29T18:56:42Z |
| 192.249.209.156 | [Category:1966 establishments in | [edit | [2023-05-29T18:56:15Z |
|                 | South Africa    |                 |                 |
| 192.249.209.156 | 100 Air Navigation School, SAAF | [edit | [2023-05-29T18:56:15Z |
| Idola2 or Roman Nisar | Idola Chola or Roman Nisar | [edit | [2023-05-29T18:56:15Z |
| Edits           | [2023 Cannes Film Festival | [edit           | [2023-05-29T18:56:14Z |
| Super Dromeosaurus | [Bulgrenes, Talesman | [edit           | [2023-05-29T18:56:14Z |
+-----------------+-----------------+-----------------+-----------------+
```

If you run the script described by above again for the same topic, you can see that more data is coming into the stream.

Count events

To display the number of revisions of articles from a selection recorded in a topic, you can use the operators `SELECT` and `COUNT`, saving the results in a new `edit_count` column.
For example, to find the number of records in the Kafka topic by changing the number of returned changes (`"rclimit"`) in the script to 100 or more, and re-run the script.
The query to count the number of revisions of the last 10 articles in the `wikipedia-events` stream and store the results in a new `edit_count` column looks like this:

```
SELECT page_title, COUNT(*) AS edit_count
FROM wikipedia_events
GROUP BY page_title
EMIT CHANGES
LIMIT 70;
```

where the `GROUP BY` function is applied.

The query result is shown below:

```
+-----------------------------------------+-----------------------------------------+
| PAGE_TITLE                              | EDIT_COUNT                              |
+-----------------------------------------+-----------------------------------------+
| Category:Redirects from long names      | 1                                       |
| Science-fiction: The Germanic Years     | 1                                       |
| Category:Articles with short descr      | 1                                       |
| ription  from 2016 Through 2050         |                                         |
| Ben Anderson (journalist)               | 1                                       |
| User talk:175.142.45.166                | 1                                       |
| Wikipedia:Articles for creation/Redirec | 1                                       |
| ists and categories                     |                                         |
| 2023 North Indian Ocean cyclone season  | 1                                       |
| Category:Accepted AFC submissions       | 1                                       |
| Category:AFC submissions by date/No Data | 1                                      |
| ine 2023                                |                                         |
| Category:Redirect-Class AFC articles    | 1                                       |
| Category:Unprintworthy redirects        | 1                                       |
| List: Reached                           |                                         |
+-----------------------------------------+-----------------------------------------+
```

Select by name

To select articles which title contains `Category:`, use the `SELECT` operator and the `WHERE` function:

```
SELECT page_title,
timestamp
FROM wikipedia_events
WHERE page_title LIKE 'Category:%'
EMIT CHANGES;
```

The query result is shown below:

```
+-----------------------------------------+-----------------------------------------+
| PAGE_TITLE                              | TIMESTAMP                               |
+-----------------------------------------+-----------------------------------------+
| Category:Redirects from long names      | [2023-06-01T07:21:57Z                   |
| Category:Accepted AFC submissions       | [2023-06-01T07:21:57Z                   |
| Category:AFC submissions by date/No June 2023 | [2023-06-01T07:21:57Z             |
| Category:Redirect-Class AFC articles    | [2023-06-01T07:21:56Z                   |
| Category:Unprintworthy redirects        | [2023-06-01T07:21:56Z                   |
+-----------------------------------------+-----------------------------------------+
```

Write from ksqlDB to a Kafka topic

To write the latest selection to a new Kafka topic `new_topic` in the JSON format, you can create a new stream `wikipedia_new`:

```
CREATE STREAM wikipedia_new
  WITH ( kafka_topic='new_topic', value_format='json')
  AS SELECT
  page_title,
  timestamp
  FROM wikipedia_events
  WHERE page_title LIKE 'Category:%'
  EMIT CHANGES;
```

Stream is created:

```
 Message
----------------------------------------------
 Created query with ID CSAS_WIKIPEDIA_NEW_191
----------------------------------------------
```

In the `ksql>` line, it is also possible to run a command to read the messages written to the `new_topic` topic:

```
print new_topic;
```

The query result is shown below:

```
key format: ¼{¿{¿/ – no data processed
Value format: JSON or KAFKA_STRING
rowtime: 2023/06/01 07:21:59.522 Z, key: <null>, value: {"PAGE_TITLE":"Category:Redirects from long names","TIMESTAMP":"2023-06-01T07:21:57Z"}, partition: 0
rowtime: 2023/06/01 07:21:59.522 Z, key: <null>, value: {"PAGE_TITLE":"Category:Accepted AFC submissions","TIMESTAMP":"2023-06-01T07:21:57Z"}, partition: 0
rowtime: 2023/06/01 07:21:59.522 Z, key: <null>, value: {"PAGE_TITLE":"Category:AFC submissions by date/No June 2023","TIMESTAMP":"2023-06-01T07:21:57Z"}, partition: 0
rowtime: 2023/06/01 07:21:59.522 Z, key: <null>, value: {"PAGE_TITLE":"Category:Redirect-Class AFC articles","TIMESTAMP":"2023-06-01T07:21:56Z"}, partition: 0
rowtime: 2023/06/01 07:21:59.522 Z, key: <null>, value: {"PAGE_TITLE":"Category:Unprintworthy redirects","TIMESTAMP":"2023-06-01T07:21:56Z"}, partition: 0
```

The topic contains messages in accordance with the selection made.

NiFi Server UI overview

Contents

To Table of Contents

- Connect to NiFi Server
- Main window
  - Toolbar
  - Status bar
  - Action palette
  - Navigation palette
- Search
- Global menu

Connect to NiFi Server

After starting the NiFi server via the startNiFi.bat file (or, on Linux the NiFi Server component is located in the startNiFi.sh file of the browser. As Bonsai users specify the value of the **nifi.ui.port** parameter from the **bin** section defined as the [Bonsai](#) stage of the NiFi service when creating the node cluster.

Another way to connect to the NiFi Server interface is to follow one of the two instances in the **URL** of the NiFi server menu.

Go to NiFi Server interface

The NiFi user interface provides mechanisms for creating automated data flows, as well as visualizing, editing, monitoring, and administering those data flows. Almost all components of the NiFi UI screenshot.

Main window

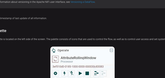
This is the window that the user sees after logging into the NiFi user interface. By default, login to the NiFi server user interface is provided without authentication.

NiFi UI main window

Primarily the NiFi user interface is presented as a blank canvas on which you can create your data flows.

Toolbar

The toolbar is located at the top left of the screen. To create data flows and can be transferred to the canvas.



Component toolbar

The toolbar contains the following elements (icons):

- **Processor**. Responsible for data ingress, egress, routing, and manipulation. There are many different types of processors.
- **Input Port**. Provides a mechanism for transferring data into a process group.
- **Output Port**. Provides a mechanism for transferring data out of a process group.
- **Process Group**. Used to logically group a set of components to make the flow of data easier to understand and maintain.
- **Remote Process Group**. Used to transfer data to a remote group. However, the remote process group refers to the entire NiFi instance.
- **Funnel**. Used to combine data from many connections into one connection.
- **Template**. Can be created from selections of other components and then also flows. These templates provide larger building blocks to quickly create a complex flow.
- **Label**. Used to attach remarks to parts of a data flow.

Status bar

The status bar contains the component counter.

Status bar

The status bar contains elements (icons) for each parameter and for the numerical value of the parameter for the user server. The status bar displays the following points:

- Total component counters:
  - The number of nodes in the cluster and the number of nodes currently connected;
  - The number of threads currently active;
  - The amount of data that currently exists in the stream;
  - The number of remote process groups currently being;
  - The number of remote process groups not sending data;
  - The number of active processors;
  - The number of processors stopped;
  - The number of invalid processors;
  - The number of disabled processors.
- Counters of processor status of components:
  - The number of components with errors;
  - The number of locally modified process groups;
  - The number of stale process groups;
  - The number of locally modified and stale process groups;
  - The number of process groups with synchronization error.

NOTE

For more information about administering of the Apache NiFi user interface, see [Administration](#).

- The connection of the cluster update of all information.

Action palette

The action palette is located on the left side of the screen. The palette consists of icons that are used to control the flow, as well as to control user access and set system properties.

Action palette

To work with a flow you need to click on the Global Menu for the menu and designation of the flow are displayed on the operator's options. The Global is available for management. The palette contains the following elements (icons):

- **Summary**. Provides the configuration of the created flow.
- **Counters**. Controls the flow (commission in the scheme).
- **Bulletin Board**. Search of errors in the process.
- **Data Provenance**. Information about the flow of data in the NiFi system.
- **Flow Configuration History**. Shows the history of configuration changes for the flows.
- **Controller Settings**. Allows users to view/modify the controller, including management controller services, reporting tasks, registry clients, settings providers, and nodes in the cluster.
- **Parameter Contexts**. Allows parameter contexts globally, without the UUID instance.
- **Cluster**. Allows cluster parameters.

Navigation palette

The navigation palette is used to move around the canvas and to control the level.

Navigation palette

Search

On the right side of the canvas is the search and the global menu.

Search and global menu

The search has allows you to:

- Find components by name, type, identifier, configuration properties and their values;
- Limit the search to the search results based on current selection and by Name and Keywords.

After clicking on the search icon a drop-down is opened the search parameter.

Search

Global menu

The global menu contains options that allow you to control processing components on the canvas.

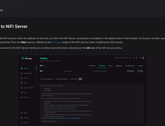
Global menu

The global menu includes the following tabs:

- **Summary** – provides a summary of all configured data flows and their statistics.

Summary of all dataflow elements

- **Counters** – shows counters and how to monitoring progression.

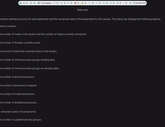


Counters

- **Bulletin Board** – bulletins of errors that have occurred. Filtering based on components, message, etc. is possible.

Bulletin Board with errors

- **Data Provenance** – detailed information for each piece of data that NiFi analyzes.



Data Provenance

Information about the Provenance Event is available by clicking the **i** icon.

Provenance Event

- **Controller Settings** – allows users to view/modify the controller, including management controller services, reporting tasks, registry clients, settings providers, and nodes in the cluster.

Controller configuration

- **Parameter Contexts** – allows parameter contexts globally, without the UUID instance.

Parameter contexts

- **Cluster** – allows cluster parameters.

Cluster settings

- **Flow configuration history** – shows the history of configuration changes for the flows.

Config change history for all flows

- **Media Status History** – shows the status process of all cluster nodes.

Status history of all cluster nodes

- **Parameter** – shows created parameters.

Created NiFi flow templates

- **Help** – opens the NiFi documentation.

Apache NiFi documentation

- **About** – shows brief information about Apache NiFi.

Apache NiFi user interaction

NOTE

You have entered to about the Apache NiFi user interface, please refer to [User Guide](#).

NiFi Registry UI overview

Olga Semme

Contents

To Table of Contents

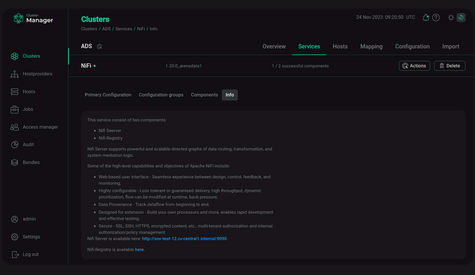
Connect to NiFi Registry

Main window

Connect to NiFi Registry

After installing the NiFi service, enter the address of the host, on which the NiFi Registry component is installed, in the address bar of the browser. As the port number, specify the value of the ***Nifi Registry UI port*** parameter from the ***Main*** section, defined at the configure stage of the NiFi service when installing the ADS cluster.

Another way to connect to the NiFi Registry interface is to follow the link provided on the ***Info*** tab of the NiFi service menu.

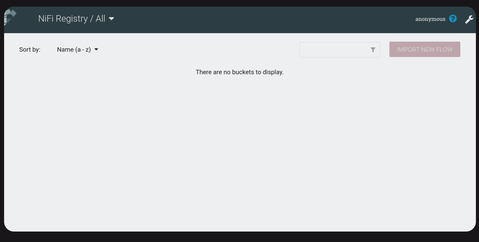


Go to the NiFi Registry interface

The NiFi Registry user interface provides a place to store and manage shared resources in one or more NiFi and/or MiNiFi instances.

Main window

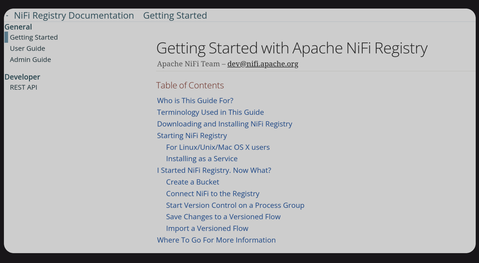
After entering the NiFi Registry user interface, the user sees the main window. By default, logging into the NiFi Registry user interface is available without authentication.



NiFi Registry UI main window

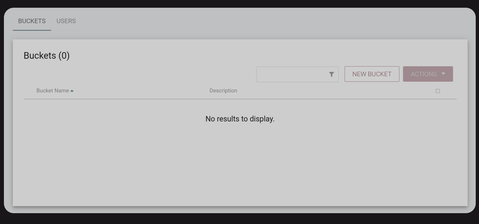
The main window of NiFi Registry displays the following items:

- anonymous — the name of the current user. It is displayed in the upper right corner. If no authentication is configured, the name `anonymous` is displayed.
- — help icon displayed in the upper right corner. Provides access to the NiFi Registry documentation.



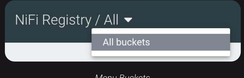
NiFi Registry documentation

- — configuration icon, displayed in the upper right corner. Opens the UI NiFi Registry settings window.



UI NiFi Registry settings

- "Buckets" menu is located on the top left side of the screen. It allows the user to display flows depending on which bucket they are in.

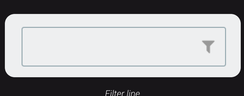


Menu Buckets

- Flows sorting area is located on the left side of the screen under the "Buckets" menu. It allows you to sort flows by name, type, and update time.

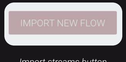
NiFi Registry flow sorting area

- Filter line is located on the right side of the screen. It allows you to set a filter to display groups of flows united by a common designation.



Filter line

- ***Import new flow*** button is located on the right side of the screen. It allows you to import the NiFi flow.



Import streams button

NOTE

For more information about the Apache NiFi Registry user interface, please refer to NiFi Registry User Interface.

NiFi service authentication

by Admin

Contents

Prerequisites for enabling authentication
Set configuration options
- Required parameters for configuring SSL
- Required parameters for setting up LDAP

Verify that NiFi Server authentication is enabled
Verify that NiFi's Registry authentication is enabled
Disable authentication
Change NiFi Initial Admin data

To Table of Contents

NOTE

- Enabling authentication is done using static NiFi configuration after installing the configured system of the NiFi service, including UAP settings.
- Authentication through the NiFi Server and NiFi Registry services are replaced with encrypted values in the configuration files. For more information about data encryption, see the article Configuration of NiFi settings and values encryption.

The required configuration parameters to enable authentication are given in series.
Basic UAP concepts used in NiFi service authentication parameters:

- **Active Directory** — a database and a set of services that connect users to the necessary network resources.
- **LDAP server** — a hierarchical database in Active Directory and in most directory, used for centralized storage of accounts.
- **Lightweight directory access protocol (LDAP)** — an application layer protocol for accessing the Active Directory directory service.
- **Distinguished name (DN)** — an element in Active Directory. The file must be unique within the tree. The file describes the name of the attributes in the tree configuration specific to the domains of servers of **Idaho distinguished names (DN)**, distinguished by moving up the tree in the direction of the root entry. DNs are written from left to right:

```
Example DN for user: user1 in the service
DN options, DC=ad, OU=major tree
Example from DN: user1 in the article
DN name from 1 example.com, OU=OU, CN=ad1, CN=ad1, CN=ad1
```

Entry fields are arranged in the following format:

- CN — common name of the user or host
- OU – an 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 — domain component (eg, domain) the top of the UAP tree, which uses DNS to determine its hierarchy.
- O — an organization name.
- L — locality name.
- ST — state or province name.
- C — country.

Prerequisites for enabling authentication

CAUTION

Enabling SSL and LDAP Auth for the NiFi service is available starting with ADS 1.5.0.1.

1. Verify that the prerequisites for running the **Manage SSL** action on an ADS cluster are met in the guide ADS. Essential security.
2. Verify that services DNS resolution and time synchronization in the NTP service of the cluster.
3. LDAP server address (UNP:389, URL: LDAP) as the LDAP for synchronization source is the former **Tinkoff LTI Search Agent**.
4. One of user must be an owner of the Active Directory domain and the rights to search users and groups and the password.
5. Base DN is used for the users in AD (and DN if necessary).
6. Base DN is search for groups in AD (and DN if necessary).

Set configuration options

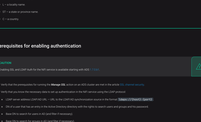
The following describes the parameters that need to be set in the tabs on the configuration page of the UAP service.

NOTE

This article only lists the settings you need to set to successfully enable authentication. Detailed information on all configuration parameters can be found in the article.

Required parameters for configuring SSL

To configure SSL, fill in the **authorizers.xml** configuration parameters.



Configuration parameters of the authorizers.xml file

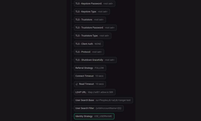
The required parameters are described below.

authorizers.xml

Parameter	Description	Example
ZK NiFi 1 cluster list	List of all cluster specifications to read the UAP file. These are the Zookeeper connection strings for the NiFi cluster. Specify the full Zookeeper chains of the NiFi service components, but also the chrome of the NiFi cluster. Specify the full Zookeeper chains of the NiFi service components, but also the chrome of the NiFi cluster. Specify the full Zookeeper chains of the NiFi service components, but also the chrome of the NiFi cluster. Specify the full Zookeeper chains of the NiFi service components, but also the chrome of the NiFi cluster.	ZK node list:2181 (Zookeeper client for 1 connection, timeout 15000 (15 000), 1 ACL, 1 ACL, 1 ACL)
NiFi initial Admin password	The account initial Admin password which will be used to login to NiFi API Service (**UAP NiFi**).	gpadmin
NiFi initial Admin password	The password with which this user is registered in Active Directory is used.	gpadmin

Required parameters for setting up LDAP

To enter LDAP parameters, you must enable the **LDAP Login Identity Provider** and **LDAP UserGroupProvider** sections.

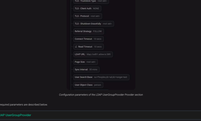


Configuration parameters of the LDAP Login Identity Provider section

The required parameters are described below.

LDAP Login Identity Provider

Parameter	Description	Example
Authentication Strategy	Use the connection to the LDAP server (e.g. **LDAPS**). Required to search users.	LDAPS
Manager DN	DN of a user that has an entry in the Active Directory with right to search users and groups. Will be used to bind to LDAP server to search for users.	cn=admin,ou=ad,dc=major,dc=ru
Manager Password	Password of the manager that is used to bind to the LDAP server to search for users.	The password with which this user is registered in Active Directory is used
Referral Strategy	Strategy for handling referrals.	FOLLOW
Connect Timeout	Duration of connect timeout.	10 sec
Read Timeout	Duration of read timeout.	10 sec
LDAP URL	Space-separated list of URLs of the LDAP servers (e.g. **ldaps://<Hostname>:<port>**).	ldaps://ad01.ad.local:636
User Search Base	Base DN for searching for users (e.g. **ou=people,o=nifi**). Required to search users.	ou=People,dc=major,dc=ru
User Search Filter	Filter for searching for users against the User Search Base (e.g. **cn=<Manager DN>**). You can specify here a restriction to specific users.	cn=cn=cn=cn=user1
Identity Strategy	Strategy to identify users. Possible values are: **USE_DN** and **USE_USERNAME**.	USE_USERNAME
Authentication Expiration	Duration of how long the user authentication is valid for. If the user never logs out, they will be required to re-login after the configured time period.	12 hours



Configuration parameters of the LDAP UserGroupProvider section

The required parameters are described below.

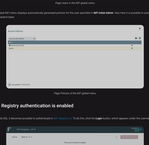
LDAP UserGroupProvider

Parameter	Description	Example
Authentication Strategy	Use the connection to the LDAP server (e.g. **LDAPS**).	LDAPS
Manager DN	DN of a user that has an entry in the Active Directory with right to search users and groups. Will be used to bind to LDAP server to search for users and groups.	cn=admin,ou=ad,dc=major,dc=ru
Manager Password	Password of the manager that is used to bind to the LDAP server to search for users	The password with which this user is registered in Active Directory is used
Referral Strategy	Strategy for handling referrals.	FOLLOW
Connect Timeout	Duration of connect timeout.	10 sec
Read Timeout	Duration of read timeout.	10 sec
LDAP URL	Space-separated list of URLs of the LDAP servers (e.g. **ldaps://<Hostname>:<port>**).	ldaps://ad01.ad.local:636
Sync Interval	Duration of time between syncing users and groups. Minimum allowable value is **10 secs**.	30 mins
User Search Base	Base DN for searching for users (e.g. **ou=people,o=nifi**). Required to search users.	ou=People,dc=major,dc=ru
User Object Class	Object class for identifying users (e.g. **person**). Required if searching users.	person
User Search Scope	Search scope for searching users.	ONE_LEVEL
User Identity Attribute	Attribute to use to determine the name (e.g. **cn**) — its format, if not set, the entire DN is used.	cn=cn=cn=cn=user1
Group Search Base	Base DN for searching for groups (e.g. **ou=groups,o=nifi**). Required to search groups.	ou=Groups,dc=major,dc=ru
Group Object Class	Object class for identifying groups (e.g. **groupOfNames**). Required if searching groups.	group
Group Search Scope	Search scope for user group.	ONE_LEVEL
Group Member Attribute	Attribute to use to define group membership (e.g. **member**).	member

After changing the parameters, click **Save**.

Verify that NiFi Server authentication is enabled

After successfully enabling SSL, login to NiFi Server 1 is done through the authorization window.

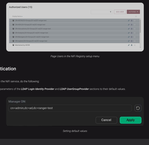


Login window

For initial authentication, you must log in via LAN administrator (enter the username and password specified in **NiFi Initial Admin**). After successful authentication, the logged in user will be displayed in the upper right corner of the interface.

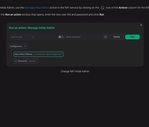
User after authentication

The global menu displays two sections **Users** and **Policies**.



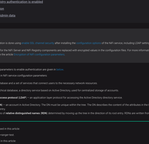
Global Menu

The **Users** page of the global UAP menu displays a list of cluster nodes, users, and LDAP/Active Directory groups that are in the specified **User Search Base** and **Group Search Base**.



Page Users in the NiFi global menu

The **Policies** page of the global NiFi menu displays automatically generated policies for the user specified in **NiFi Initial Admin**. Also here it is possible to assign or access policy for a user or group of the current cluster.



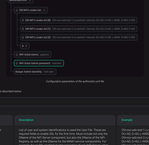
Page Policies in the NiFi global menu

Verify that NiFi Registry authentication is enabled

After successfully enabling SSL, login to NiFi Registry 1 is done. To do this, click the login button, which appears under the username.

Go to login window

In the authorization window for primary authentication, you must log in via a NiFi administrator under the username and password specified in **NiFi Initial Admin**.

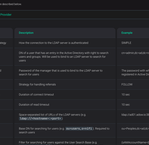


Login window

After successful authentication, the logged in user is displayed in the upper right corner of the interface.

User after authentication

After clicking on **Users** and selecting the **Users** tab in the settings menu, you can see a list of cluster nodes, users and LDAP/Active Directory groups located in the specified **User Search Base** and **Group Search Base**. Also here you can add and delete NiFi Registry users and groups.

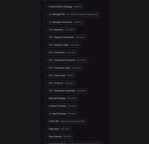


Page Users in the NiFi Registry setup menu

Disable authentication

To disable authentication in the NiFi service do the following:

1. Set the configuration parameters of the **LDAP Login Identity Provider** and **LDAP UserGroupProvider** sections to their default values.



Setting default values

CAUTION

If you want to reset the NiFi service configuration parameters (**LDAP Login Identity Provider** and **LDAP UserGroupProvider**) to available starting from version 1.5.0.1, you must first disable authentication for the NiFi service. To do this, uncheck the box next to the **Authorizers.xml** and **ldap.xml** fields in the **Manage SSL** window and click **Apply**. Now from such a reset where NiFi service is installed.

2. Disable SSL. To do this, apply the steps described in the guide by clicking on the **SSL** and the **Actions** options.

Change NiFi Initial Admin data

To change data about NiFi Initial Admin, use the Manage Users action in the NiFi service by clicking on the **Actions** section for the NiFi service.

After entering the section in the **Set an admin** window that opens, enter the new user ID and password and click **Set**.

Change NiFi Initial Admin

Authorization in the NiFi user interface

Olga Semma

Contents

To Table of Contents

NiFi Server
- Configure access policies via the global NiFi menu
- Configure access policies at the processor level

NiFi Registry

Authorization in the NiFi service using interface tools is carried out after authentication of a user who has an account in Active Directory.

It is possible to assign access policies to Active Directory users located in the *User Search Base* specified in the parameter configuration step of the NiFi service.

NOTE

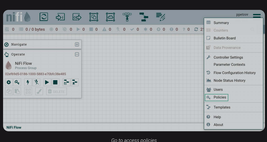
All sensitive values for the NiFi Server and NiFi Registry components are replaced with encrypted values in the configuration files. For more information about data encryption, see the article Encryption of NiFi configuration parameters.

NiFi Server

Configure access policies via the global NiFi menu

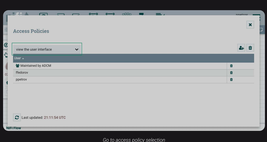
To assign an access policy via the NiFi Server interface global menu, do the following:

1. Go to the **Policies** page in the global menu.



Go to access policies

2. Open the list of access policies.



Go to access policy selection

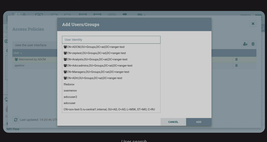
3. Select an access policy from the list and click the access policy assignment icon.

Available access policies

NOTE

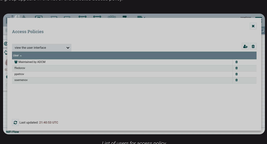
For detailed information about access policies in the NiFi Server interface, please refer to Access Policies NiFi.

4. Find a user or a group to assign the policy using ***User Indentity*** string.



User search

5. Select a user and click **ADD**
   As a result, a new user or a group appears in the list of the selected access policy.

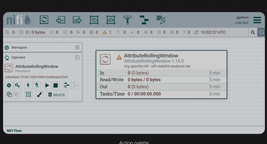


List of users for access policy

Configure access policies at the processor level

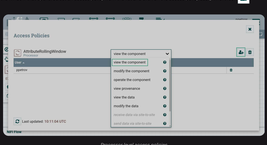
After configuring authentication, the icon appears on the Action palette, with which access policies are assigned at the level of the selected processor or groups. To assign policies, do the following:

1. Select the desired processor or a group and highlight it. Then the name of the selected processor appears on the palette of operations.



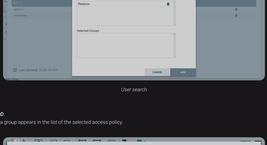
Action palette

2. Click on the icon. In the window that opens, select an access policy from the list and click on the access policy assignment icon.



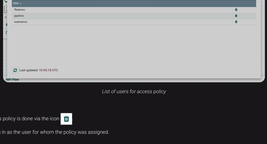
Processor-level access policies

3. Find a user or a group to assign the policy using ***User Indentity*** string.



User search

4. Select a user and click **ADD**
   As a result, a new user or a group appears in the list of the selected access policy.



List of users for access policy

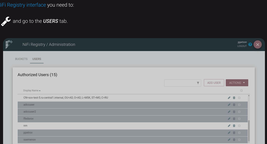
Deleting a user from the access policy is done via the icon.

To check the created policy, log in as the user for whom the policy was assigned.

NiFi Registry

To assign an access policy in NiFi Registry interface you need to:

1. Click on the settings icon and go to the **USERS** tab.



List of users for access policy

2. Select a user or a group and click on the edit access policy icon.
3. Select the required permissions and click **CLOSE**

Select required permissions

NOTE

For detailed information about access policies in the NiFi Registry interface, refer to the article Access Policies NiFi Registry.

To check the created policy, log in as the user for whom the policy was assigned.

Ranger NiFi plugin

Olga Semme

Contents

To Table of Contents

Prerequisites

Establish trust over SSL

Activate the Ranger NiFi plugin

Check the activated Ranger NiFi plugin

Prerequisites

Prerequisites for logging in to NiFi with Ranger:

1. ADS cluster is installed with authentication configured for a user with an Active Directory account.
2. ADPS cluster is installed.
3. SSL (using SSL certificates or Active Directory certificates) is enabled and configured on ADPS cluster.
4. To disable case conversion for user and group names in NiFi, the following parameters of the ***LDAP sync source for User synchronizer*** group for the ADPS cluster Ranger service are set to `none`:
   - ***ranger.usersync.ldap.groupname.caseconversion***
   - ***ranger.usersync.ldap.username.caseconversion***

   For more information about configuring user synchronization in Ranger, please refer to the Configure Ranger User Sync article.
5. ADS cluster integration is enabled on ADPS cluster.

NOTE

- This article describes authorization with self-signed SSL certificates.
- Minimum requirements to enable authorization in Ranger for NiFi service:
  - ADPS version 1.0.4.b3.
  - ADS version 1.7.1.b1 (for NiFi Server) and ADS version 1.7.2.b1 (for NiFi Registry).

Establish trust over SSL

Enabling the Ranger plugin requires each cluster to trust the other's SSL certificates.

For this you need:

1. Import the certificate *.crt for each host in the cluster into the *truststore* of each host in another cluster.
   On each host where certificates are imported, run:

```
$ keytool -import -file /tmp/sov-test-1.ru-central1.internal.crt -keystore /tmp/truststore.jks -storepass bigdata -noprompt
```

2. On all hosts, re-import *truststore* into the operating system's trust store:

```
$ keytool -importkeystore -srckeystore /tmp/truststore.jks -destkeystore /etc/pki/java/cacerts -deststorepass changeit -srcstorepass bigdata -noprompt
```

3. On each host, add certificates to *ca-bundle.pem* with the following commands:

```
$ chmod 777 /etc/pki/ca-trust/extracted/pem/ca-bundle.pem
$ echo "######################" >> /etc/pki/ca-trust/extracted/pem/ca-bundle.pem
$ cat /tmp/sov-test-1.ru-central1.internal.crt >> /etc/pki/ca-trust/extracted/pem/ca-bundle.pem
$ chmod 444 /etc/pki/ca-trust/extracted/pem/ca-bundle.pem
```

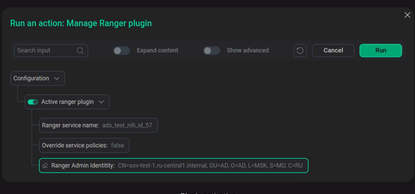
NOTE

If your ADPS cluster uses Active Directory certificates to enable SSL, you should import the Active Directory root and client certificates into the *truststore.jks* of the ADS cluster hosts.

Activate the Ranger NiFi plugin

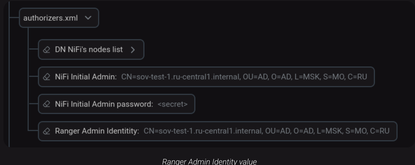
To activate the Ranger NiFi plugin, do the following:

1. Initiate activation of the Ranger NiFi plugin. To do this, apply the action Manage Ranger plugin by clicking in the ***Actions*** column of the NiFi service.
2. In the ***Run an action: Manage Ranger plugin*** window that opens, activate the ***Active ranger plugin*** switch. Wherein:
   - If necessary, set the name of the Ranger service that will be added. If a service with the same name already exists, it can be recreated by activating the ***Override service policies*** parameter (`true` – the parameter is active, `false` – inactive), in this case, the old service will be deleted, and policies will be regenerated for the new service. If the ***Override service policies*** parameter is inactive, after activating the plugin, a connection to the existing Ranger service will be made.
   - Specify ***Ranger Admin Identity*** – DN of the ADPS host certificate that was generated when configuring SSL using self-signed certificates and imported to the host (for example, `CN=sov-test-1.ru-central1.internal, OU=AD, O=AD, L=MSK, S=MO, C=RUADS`). This DN Ranger will be used to communicate with NiFi.



Plugin activation

3. Wait until the activation process of the Ranger NiFi plugin is completed and the default policy is created on the Ranger side. Analyze and correct errors if they occur.
4. Verify that the ***Ranger Admin Identity*** property in the NiFi configuration is populated with the DN value.

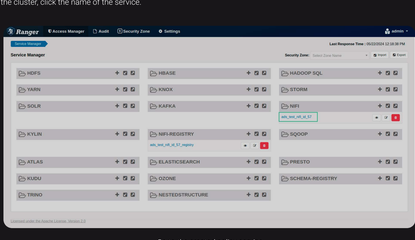


Ranger Admin Identity value

Check the activated Ranger NiFi plugin

After authorization in Ranger web interface, authorization policy services for the NiFi service components of the selected ADS cluster appear in the ***Service manager*** window.

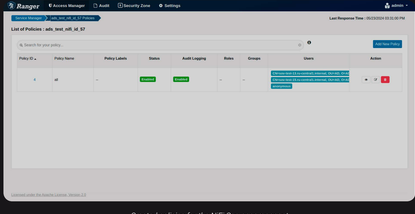
To view the created policies for the cluster, click the name of the service.



Go to the created policy service

This makes available the policies automatically created for the cluster servers.

To view, edit, or delete policies, use the buttons in the ***Action*** column.



Created policies for the NiFi Server component

Use Ranger in NiFi

Apr 2023 | 20 min

Contents

Set an authorization policy in NiFi Server
- Authorize NiFi Initial Admin in NiFi Server
- Node user authorization

Work of an authorized user in NiFi

Configure an authorization policy in NiFi Registry
- Authorize NiFi Initial Admin in NiFi Registry

To Table of Contents

After activating Ranger NiFi plugin, creating a Ranger service, or connecting to an existing service (depending on the value of the Ranger service policies parameter), authorization in NiFi Server and NiFi Registry is moved for all users, including the NiFi administrator specified in **NiFi Initial Admin** when setting Authorization in NiFi service.

After trying to log in with NiFi Initial Admin, a message appears that there is no configured policy for the user:

User authorization error

NOTE

- This article describes authorization using SSL-enabled HDFS with self-signed SSL certificates. However, users in Ranger must be created manually. To automatically synchronize with Active Directory users, you should configure LDAP on HDFS.
- For more information on authorization in NiFi using Ranger, you can refer to the article LDAP Ranger.
- All sensitive values for the NiFi and NiFi Registry components are replaced with encrypted values in the configuration files. For more information about encryption, see the article Encrypt configuration passwords in NiFi.
- For more details on working with Ranger, you can refer to the following articles:
  - Manage services in Ranger
  - Manage policies in Ranger

Set an authorization policy in NiFi Server

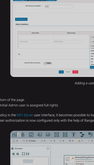
Authorize NiFi Initial Admin in NiFi Server

To authorize NiFi Initial Admin, do the following:

1. Create a user in Ranger. To do this, in the top menu of the Ranger interface, select **Settings** → **Users/Groups/Roles** and click **Add new user**

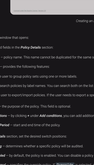
Creating a user in Ranger

2. In the window that opens, fill in the required data for the user:
   - **User Name** – username.
     The name must match the username in the Active Directory database. The value can be:
     - full DN of the user (starting with `CN=`) if the **Identity Strategy** configured parameter of the **LDAP Login Identity Provider** group is `USE_DN`
     - only login (name) of the user when setting **Identity Strategy** – `USE_USERNAME`
   - **New Password** – user password.
   - **Password Confirm** – user password confirmation.
   - **First Name** – personal username.
   - **Last Name** – last name of the user.
   - **Email Address** – user's email address.
   - **Select Role** – user role selection (`ROLE_USER`, `ROLE_ADMIN`).
   - **Group** – the group groups the user belongs to.



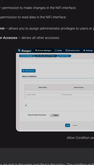
Creating a user in Ranger

3. Click **Save**.
4. In the **Service Manager** window, click the name of the NiFi service.



Go to the created policy service

5. In the **List of Policies** window that opens, click the edit icon of the created **all** policy.



Go to editing the authorization policy

6. In the policy editing window that opens, in the **Allow Conditions** → **Select User** table, select the NiFi Initial Admin user created by above.

Adding a user to an access policy

7. Click **Save** at the bottom of the page.
   As a result, the NiFi Initial Admin user is assigned full rights.

After setting the access policy in the user interface, it becomes possible to log in as NiFi Initial Admin. At the same time, **Users** and **Policies** tabs are missing in the global menu of the interface – authorization is now configured only with the help of Ranger.

After logging with NiFi Initial Admin

New user authorization

It is possible to assign access policies to Active Directory users located in the User Search Base specified in the parameter configuration step of the NiFi service.

For this you need:

1. Create a new user as described by above.
2. In the **List of Policies** window that opens, click **Add new policy** to add a new policy for the user.

Creating an authorization policy

3. In the **Create Policy** window that opens:
   - fill in the required fields in the **Policy Details** section:
     - **Policy Name** – policy name. This name cannot be duplicated for the same service in the system. This field is required.
     - **Policy Label** – provides the following features:
       - Allows the user to group policy with using one or more labels.
       - User can search policies by label names. You can search on the list of policies page and on the report page.
       - Helps the user to export/import policies. If the user needs to export a specific set of policies, they can find the policy label and export the specific set of policies.
     - **Description** – the purpose of the policy. This field is optional.
     - **Policy Conditions** – by clicking + point **Add conditions**, you can add additional conditions.
     - **Add Validity Period** – start and end time of the policy.
   - in the **Policy Details** section, set the desired switch positions:
     - **Audit Logging** – determines whether a specific policy will be audited.
     - **Enable Disabled** – by default, the policy is enabled. You can disable a policy to restrict user/group access for that policy.
     - **Override Parent** – determines whether the current policy (`override`) or inherited access permissions in the policy override access permissions in existing policies. This feature can be used in conjunction with **Add Validity Period** to create temporary access policies that override existing policies.

Create Policy window

4. In the **Resources** section, if necessary, fill in the field that determines the value of the `NIFI Resource Identifier` parameter.

Create Policy window

5. fill in the fields in the **Allow Condition** section:
   - **Select Role** – for role this policy applies to. A role is a set of permissions. Roles are an easier way to manage a set of permissions based on specific access criteria.
   - **Select Group** – the group this policy applies to. To promote a user to an administrator, select the **Delegate Admin** check box. Administrators can edit or delete a policy and create child policies. A audit group contains all users, so granting access to a public group gives access to all users.
   - **Select User** – the user this policy applies to (outside the group already specified). You can make the user an administrator of this policy. Administrators can create child policies based on existing policies.
   - **Permissions** – adds or removes permissions:
     - `Write` – permission to make changes in the NiFi interface
     - `Read` – permission to read data in the NiFi interface
   - **Delegate Admin** – allows you to assign administrator privileges to users or groups specified in the policy
   - **Deny All Other Accesses** – denies all other accesses.

Allow Condition section of the Create Policy window

NOTE

The conditions are met in the order specified in the policy. The condition at the beginning of the list is applied first, then the second, third, and so on.

6. Click **Add**

Work of an authorized user in NiFi

After activating Ranger and setting up an authorization policy for the user, work in the NiFi server interface occurs after authorization in accordance with the assigned access policy. For example, for a user assigned only the `Read` permission, the toolbar is inactive, and the user can only view (service elements without changing) settings:

Authorized user without rights to change parameters

Configure an authorization policy in NiFi Registry

Authorize NiFi Initial Admin in NiFi Registry

To authorize NiFi Initial Admin in NiFi Registry, do the following:

1. In the **Service Manager** window, click the name of the NiFi Registry service.

Go to the created policy service

2. In the **List of Policies** window that opens, click the edit icon of the created **all** policy.

Go to editing the authorization policy

3. In the policy editing window that opens, in the **Allow Conditions** → **Select User** table, select the NiFi Initial Admin user created by above.

Adding a user to an access policy

4. Click **Save** at the bottom of the page.
   As a result, the NiFi Initial Admin user is assigned full rights.

After setting the access policy in the NiFi Registry user interface, it becomes possible to log in as NiFi Initial Admin. At the same time, there is no **Users** tab in the settings menu – user authorization is now configured only with the help of Ranger.

Authorizing a new user and creating a policy in the NiFi registry policy service is similar to NiFi Server, while users can give the following permissions in the **Allow Conditions** → **Permissions** table:

- `Write` – permission to make changes to thread groups in the NiFi Registry interface.
- `Read` – permission to read stream data in the NiFi Registry interface.
- `Delete` – permission to delete threads.

After activating Ranger and setting up an authorization policy for the user, work in the NiFi Registry interface occurs after authorization in accordance with the assigned access policy.

Work with processors

Table of contents

Contents

To Table of Contents

Separate processor
- Create a processor
- Processor management
- Connect and disconnect processors

Process group
- Create a process group
- Process group management
- Connect and disconnect a process group

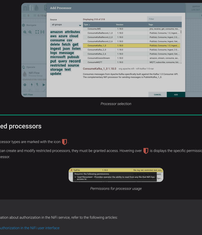
Work with processors is carried out in [API Server Interface](#).

Separate processor

Create a processor

The processor is created in the following sequence:

1. Move the icon from the [toolbox](#) to the canvas.
2. Select the desired processor in the dialog box that opens. When choosing, you can use a tag from the tag cloud.


Add Processor
Processor
Version
Created by NiFi On-Demand
All types
Tags
amazon attributes aws azure cloud compress content csv dbcp dynamodb ease egress ingest json kafka kerberos microsoft parquet put record s3 scripted sql storage text update
Name
ConsumeAMQP 1.19.0
ConsumeAzureEventHub 1.19.0
ConsumeEWS 1.19.0
ConsumeGCPubSub 1.19.0
ConsumeGCPubSubLite 1.19.0
ConsumeGrpc 1.19.0
ConsumeIMAP 1.19.0
ConsumeJMS 1.19.0
ConsumeKafka 1.19.0
ConsumeKafkaRecord_0_10 1.19.0
ConsumeKafkaRecord_0_11 1.19.0
ConsumeKafkaRecord_1_0 1.19.0
ConsumeKafkaRecord_2_0 1.19.0
ConsumeKafkaRecord_2_6 1.19.0
ConsumeMQTT 1.19.0
ConsumeMongo 1.19.0
ConsumeMongoDB 1.19.0
ConsumeNetworkFlowData 1.19.0
ConsumeNetworkFlowData 1.19.0
ConsumeNetworkFlowData

Configure processors and process groups

Contents

To Table of Contents

Configure a processor
- SETTINGS tab
- SCHEDULING tab
- PROPERTIES tab
- RELATIONSHIP tab
- COMMENTS tab

Configure process groups
- GENERAL tab
- CONTROLLER SERVICES tab

Configure connections
- DETAILS tab
- SETTINGS tab

Configure a processor

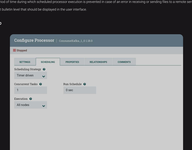
To configure a processor, right-click the processor and select the **Configure** option from the context menu. The configuration dialog opens with four different tabs, each of which is described below. After configuring a processor, it is possible to apply the changes by clicking **APPLY** or cancel all changes by clicking **CANCEL**.

CAUTION

- After the processor is started, the context menu displayed for the processor contains the **View configuration** option instead of **Configure**. The processor configuration cannot be changed while the processor is running. Before tuning the processor again, you must stop the processor and wait for all of its active tasks to complete.
- Inputting certain characters is not supported and is automatically filtered out as you type. The following characters and any unpaired Unicode surrogate codes are not preserved in any configuration:

```
[0x0], [0x1], [0x2], [0x3], [0x4], [0x5], [0x6], [0x7], [0x8], [0xE], [0xF], [0x10], [0x11], [0x12], [0x13], [0x14], [0x15], [0x16], [0x17], [0x18], [0x19], [0x1A], [0x1B], [0x1C], [0x1D], [0x1E], [0x1F], [0x7F]
```

SETTINGS tab



SETTINGS tab

Name – processor name. The default processor name is the same as the processor type. Next to the processor name is an **Enabled** flag indicating whether the processor is enabled (**✓**) or the unique processor identifier.

Type – the processor type.

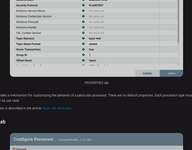
Bundle – the NAR bundle.

Yield Duration – the period of time to which a response from the remote service, when the processor uses "yield", which will prevent the processor from being scheduled to run for some period of time.

Penalty Duration – the period of time during which scheduled processor execution is prevented in case of an error or receiving or sending files to a remote service.

Bulletin Level – the lowest bulletin level that should be displayed in the user interface.

SCHEDULING tab



SCHEDULING tab

Scheduling Strategy – the scheduling strategy. Possible options for scheduling components:

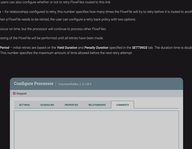
- **Timer driver** – the processor will run at regular intervals. The processor startup interval is determined by the Startup Schedule parameter.
- **CRON driver** – starting using CRON expressions. According to cron, the processor is scheduled to run periodically, similar to time driven scheduling mode. However, CRON-managed mode provides much more flexibility at the cost of increased configuration complexity. The CRON-managed scheduling value is a string of six required fields and one optional field, each separated by a space.

Concurrent Tasks – determines how many threads will use the processor. In other words, this determines how many FlowFiles should be processed by this processor at the same time.

Run Schedule – determines how often the processor should be scheduled to run. Valid values for this field depend on the selected processing strategy.

Execution – determines which node(s) the processor will be scheduled to execute on.

PROPERTIES tab



PROPERTIES tab

The **PROPERTIES** tab provides a mechanism for customizing the behavior of a particular processor. There are no default properties. Each processor type must determine which properties make sense for its use case.

Setting processor properties is described in the article [Work with attributes](#).

RELATIONSHIP tab

RELATIONSHIP tab

Automatically Terminate / Retry Relationships

- **Automatically Terminate** – for a processor to be considered valid and able to run, each relationship defined by the processor must either be connected to a child component or automatically terminated. If a relationship is terminated automatically, any FlowFile routed to that relationship will be removed from the flow and its processing will be considered complete. Any association that is already associated with a child component cannot be automatically terminated. The relation must first be removed from any connection that uses it. Also, for any relationship that is selected for auto-termination, the auto-completes state will be retained (disabled) if the relationship is added to the connection.
- **Automatically Retry** – users can also configure whether or not to retry FlowFiles routed to this link.

Number of Retry Attempts – for relationships configured to retry, this number specifies how many times the FlowFile will try to retry before it is routed to another location.

Retry Back Off Policy – when a FlowFile needs to be retried, the user can configure a retry back off policy. The options are:

- **Penalize** – retries will occur on time, but the processor will continue to process other FlowFiles.
- **Yield** – the whole thread will yield, so that the processor will not execute any more tasks.

Retry Maximum Back Off Period – initial retries are based on the **Yield Duration** and **Penalty Duration** specified in the **SETTINGS** tab. The duration time is doubled many times for each retry attempt. This number specifies the maximum amount of time allowed between the two retry attempts.

COMMENTS tab

COMMENTS tab

The tab simply provides an area for users to include any comments that are appropriate for that component.

NOTE

For more information on configuring NiFi processors, see [Configuring a processor](#).

Configure process groups

GENERAL tab

GENERAL tab

Process Group Name – the process group's name. The name appears in the title bar of the process group when the group is added to the canvas, as well as in the breadcrumbs at the bottom of the user interface. For the root process group (i.e., the highest level group), this is also the name that appears in the upper-left corner of the browser tab. Note that this information is visible to any other NiFi instance that remotely connects to the instance (using remote process groups, also known as Site-to-Site).

Process Group Parameter Context – the parameter context that is used to provide parameters to flow components. From this drop-down list, the user can select which configuration context should be used by the process group, and an optional description. A new context is created by selecting the **Create new...** option.

Process Group Comments – process group comments. This provides a mechanism for adding any useful information about the process group.

Process Group FlowFile Concurrency – used to control how data is passed to the process group.

These options are available:

- **Unbounded** – the input ports in a process group will receive data as fast as possible. Unbounded but backpressure does not prevent them from doing so.
- **Single FlowFile Per Node** – the input ports will only receive data from one node. Once the FlowFile enters a process group, no additional FlowFiles will be added until all FlowFiles have left the process group (either by being purged/auto-terminated or exited via an exit port).
- **Single Batch Per Node** – the input port will behave the same as a **Single FlowFile Per Node** mode, but when receiving a FlowFile, the input ports will continue to receive all data until all queues feeding the input ports, until not be emptied. At this point, they will no longer be considered done in the process group until all data has finished processing and leaves the process group.

Process Group Outbound Policy – while the FlowFile Concurrency dictates how data should be delivered to the process group, the outbound policy controls the flow of data out of the process group. There are two options available:

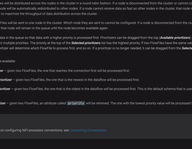
- **Stream When Available** – data arriving at the output port is immediately streamed from the process group, provided no backpressure is applied.
- **Batch Output** – the output ports will not transmit data from the process group until all of the process group has been queued at the output port (i.e. no data has left the process group until processing of data is completed). A downstream FlowFile Concurrency is based on the same restriction, but applies to output port. A port of these values is queued to output port. Both of these restrictions are considered the same in terms of the completion of FlowFile processing.

Default Settings for Connections – the default FlowFile expiration.

Default Back Pressure Object Threshold – the default backpressure object threshold.

Default Back Pressure Data Size Threshold – the default backpressure data size threshold.

CONTROLLER SERVICES tab



CONTROLLER SERVICES tab

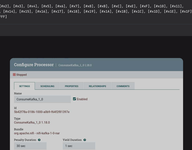
The tab displays controller services for data flows.

NOTE

For more information on configuring NiFi processors groups, refer to [Configuring a Process Group](#).

Configure connections

DETAILS tab



DETAILS tab

The **DETAILS** tab provides information about the source and target components, including the name of the component, the type of the component, and the process group the component is in.

For Relationships – provides the ability to select which relationships should be included in this connection. At least one relationship must be selected. If only one relationship is available, it is selected automatically.

If multiple connections are created with the same relation, any FlowFile routed for that relation will automatically be "cloned" and a copy of it will be sent to each of those connections.

SETTINGS tab

SETTINGS tab

Name – connection name. Can be left blank.

FlowFile Expiration – the constant that data that cannot be processed in a timely manner can be automatically removed from the flow. This is useful, for example, when the amount of data is constantly changing and it is critical that it can be sent to the remote host.

Back Pressure – two configuration items for backpressure:

- **Object Threshold** – number of FlowFiles that can be queued before being applied. The default value is `10,000`.
- **Size Threshold** – the maximum amount of data to exist that must be queued before backpressure is applied. The default value is `1 GB`.

Load Balance Strategy – to distribute data in a stream across nodes in a cluster. NiFi offers the following load balancing strategies:

- **Do not load balance** – do not load balance FlowFile across nodes in the cluster. This is the default value.
- **Partition by attribute** – determines which node to send the FlowFile to, based on the value of the user-specified FlowFile attribute. All FlowFiles with the same attribute value will be sent to the same node in the cluster. If the destination node is disconnected from the cluster or cannot communicate, no data is sent to the same to the node. The data will queue up, potentially causing backpressure to be applied. If the node is restarted or removed, the FlowFiles will be distributed to the remaining nodes. If the node is disconnected, consistent hashing is applied to cause redistributing of data.
- **Round robin** – FlowFiles will be distributed across the nodes in the cluster in a round-robin fashion. If a node is disconnected from the cluster or cannot communicate with a node, data will be queued up and will be sent to that node when it is able to receive data. This may result in some nodes receiving more or less FlowFiles than other nodes in the cluster. That node may also be skipped for one or more iterations to maximize the throughput of data distribution across the cluster.
- **Single node** – all FlowFiles will be sent to one node in the cluster. Which node they are sent to cannot be configured. If a node is disconnected from the cluster or cannot contact the node, data queued for that node will remain in the queue until the node becomes available again.

Prioritizers – provides data in the queue in the order with a higher priority is processed first. Prioritizers can be dragged from the top (**Available prioritizers**) to the bottom (**Selected prioritizers**). You can select multiple prioritizers. The priority at the top of the **Selected prioritizers** list has the highest priority. If two FlowFiles have the same value according to this prioritizer, the second prioritizer will determine which FlowFile is process first, and so on. If a prioritizer is no longer needed, it can be dragged from the **Selected prioritizers** list to the **Available prioritizers** list.

The following prioritizers are available:

- **FirstInFirstOutPrioritizer** – given two FlowFiles, the one that reaches the connection first will be processed first.
- **NewestFlowFileFirstPrioritizer** – given two FlowFiles, the one that is the newest in the dataflow will be processed first.
- **OldestFlowFileFirstPrioritizer** – given two FlowFiles, the one that is the oldest in the dataflow will be processed first. This is the default scheme that is used if no priorities are selected.
- **PriorityAttributePrioritizer** – given two FlowFiles, an attribute called `priority` will be retrieved. The one with the lowest priority value will be processed first.

NOTE

For more information on configuring NiFi processor connections, see [Understanding Components](#).

Work with attributes

Contents

- FlowFile attributes
- Processor properties
- Parameters
  - Create a parameter context
  - Create parameters in the parameter context
  - Work with the parameter contexts in the NiFi service interface
- Other Parameters components
- Variables
  - Create a variable
  - Use variables in processor properties via Expression Language
  - Check dependent components
- Parameters using Expression Language

To Table of Contents

FlowFile attributes

As we learnt (or at least mentioned) in attributes, each FlowFile has two sets (pairs, sorted and duplicate, don't know about the last?) that are created by some of the processors we have used so far: [UpdateAttribute](#) and [RouteOnAttribute](#).

FlowFile attributes

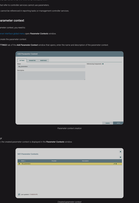
FlowFile attributes

As a reminder, UUID is a universally unique identifier, [Apache Commons](#) has one that we've used, and Java also has one.

Processor properties

Setting [properties](#) is the most important and essential configuration for processors.

Some processors have [dynamic properties](#), which allow you to add your own properties (starting with the value provided for the property's name). The [Value](#) of each property can be [literal text](#) or can reference [Expression Language](#) and [variables](#).



Scheduling properties

Processor properties

Processors with one set name for a property, like the Routing tree of routing processors, will have a validation.

- Valid, a [green tick](#) if there's any properties saved.
- Invalid, a [red stop](#) if there's no properties saved.
- Validating, a [grey hourglass](#) if [Expression Language](#) is used and [variables](#) are referenced, to check that they exist, or validity of any of Expression Language.

Invalid processor

Validating processor

Properties have the following properties over variables:

- Support the [Expression Language](#).
- Can reference [variables](#).
- Can be [sensitive](#) or not sensitive.
- Can be [required](#) or not required.

Properties have the following properties over variables:

- Can be [created](#) in the [Parameter Context](#) window.
- Can be [edited](#) in the [Parameter Context](#) window.
- Can be [set](#) to [Empty string](#) or [Unset](#) parameter values when the value is removed.

Parameters

Parameters, known internally and sometimes as in-application variables, [Parameters](#) are [variables](#) in the **Parameter Context** window that can be [referenced](#) in [Expression Language](#) and [variables](#) in processor properties. Parameters are [sensitive](#) or not sensitive and [required](#) or not required.

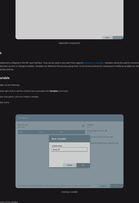
Parameters have the following properties over variables:

- Cannot [reference](#) other variables.
- Can be [referenced](#) in [Expression Language](#).
- Can be [referenced](#) in [variables](#).
- Can be [referenced](#) in [Expression Language](#) in [variables](#).

Create a parameter context

To create parameter contexts we need to:

1. Open the [Configure](#) window for the [Process Group](#) we're working in.
2. Click the [Parameter Context](#) drop down menu.
3. Click [Create new parameter context](#).



Create parameter context

4. Add a [name](#).
5. Add a [description](#) (optional).
6. Add [sensitive](#) and [required](#) parameters.
7. Add the [Value](#) for each [parameter](#) created in the above steps then click [Apply](#).

Create parameter context

Create parameters in the parameter context

To create parameters within our contexts:

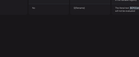
1. Open the [Configure](#) window for the [Process Group](#) we're working in.
2. Click the [Parameter Context](#) drop down menu.
3. Click [Edit Parameter Context](#).

Edit parameter context

4. Click the [Add Parameter](#) button.
5. Add a [name](#).
6. Add a [description](#) (optional).
7. Add a [Value](#).
8. Set [Sensitive value](#) or [Empty string](#) or [Unset](#) parameter values when the value is removed.
9. Set [Required](#).
10. Click [Apply](#).

Add parameter

11. Click [CLOSE](#) to return to the edit parameter context window for [Update Parameter Context](#) (parameters).

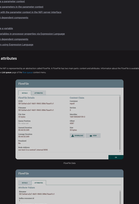


Edit parameter context

Work with the parameter context in the NiFi server interface

To create parameter contexts we need to:

1. Go to the top right corner and click the drop down menu for [Configuration](#) (or [Global](#)).
2. Click [NiFi Flow Configuration](#) (this will be greyed out if [CONTROL](#) is [1]) or [Process Group Parameter Context](#) (parameters, when the [Process Group](#) is selected in the [canvas]).
3. Click [APPLY](#).



NiFi Flow Configuration

Process Group Configuration

In order to apply the created parameters from the edit parameter context to a work processor, do the following:

1. Open the [Configure](#) window for the work processor we want to add the parameter to.
2. Click the [PROPERTIES](#) tab, select the required property and click the operation pencil to edit the property value, then click [EDIT](#). Here we've entered one parameter, `#{filename}`, and two parameters and literal text, `#{path}/#{filename}`.
3. Click [APPLY](#).

Processor properties in the Edit Property box

In order to see the [parameters](#) from the [Edit Parameter Context](#) for a work processor, apply the following:

1. Open the [Configure](#) window for the work processor we want to add the parameter to.
2. Click the [PROPERTIES](#) tab, select the required property and click the operation pencil to edit the property value, then click [EDIT](#). Here we've entered one parameter, `#{filename}`, and two parameters and literal text, `#{path}/#{filename}`.
3. Click [APPLY](#).

Processor properties in the Edit Property box

When clicking on the [?] parameter you are taken to the Edit Parameter Context.

Edit Parameter Context from Edit Property box

Check dependent components

Go to **Advanced Client** and **Processor Controller** to check for the relationships by **Referencing Components** to display the names of components that are dependent.

Dependent components

Variables

[Variables](#) are created and edited in the NiFi UI's [canvas](#). They can be used in any field that supports [Expression Language](#). Variables can be used to reference [parameters](#) in the [Parameter Context](#) window, [properties](#) in [processors](#), and [Expression Language](#) in [variables](#) and [parameters](#). Variables are [sensitive](#) or not sensitive and [required](#) or not required.

Variables have the following properties over variables:

- Cannot [reference](#) other variables.
- Can be [referenced](#) in [Expression Language](#).
- Can be [referenced](#) in [variables](#).
- Can be [referenced](#) in [Expression Language](#) in [variables](#).

Create a variable

To create variables, do the following:

1. Access the right-click on and select [Add Variable](#) (or [Variables](#) in the [Operate](#) palette).
2. Add a [Variable Name](#) and [Value](#).
3. Click [Variable Scope](#).
4. Click [OK](#).

Adding a variable

5. Click the drop down variable.

Variable operations

6. Click [OK].
7. Click [Variables](#) (below edit) and [APPLY](#) to create an instruction window for updating or creating variable values.

Variables

Use variables in processor properties via Expression Language

In order to use variables in processor properties, do the following:

1. Open the [Configure](#) window for the work processor we want to add the variable to, click [Settings](#).
2. Click the [PROPERTIES](#) tab, select the required property and click the operation pencil to edit the property value, then click [EDIT](#). Here we've entered one parameter, `#{filename}`, and two parameters and literal text, `#{path}/#{filename}`.
3. Click [APPLY](#).

Apply the created property

Expression Language functions

[Expression Language](#) functions are [literal text](#) or [variables](#) that are [referenced](#) in [Expression Language](#). They can be used in [properties](#) and [variables](#) using the [#{}](#) and [${}](#) syntax. Functions are [separated](#) with [:](#) and [()](#) and can be [chained](#) together. Functions can be [literal text](#) or [variables](#) and [parameters](#) and [literal text](#).

- Functions can be [literal text](#) or [variables](#) and [parameters](#) and [literal text](#) inside [quotes](#) with [single quotes](#) `'literal'` or [double quotes](#) `"literal"`.
- Functions can be [literal text](#) or [variables](#) and [parameters](#) and [literal text](#) inside [quotes](#) with [single quotes](#) `'literal'` or [double quotes](#) `"literal"` and [escaped](#) with [\](#) `\'literal\'` or `\"literal\"`.
- Functions can be [literal text](#) or [variables](#) and [parameters](#) and [literal text](#) inside [quotes](#) with [single quotes](#) `'literal'` or [double quotes](#) `"literal"` and [escaped](#) with [\](#) `\'literal\'` or `\"literal\"` and [trimmed](#) with [trim()](#) `trim()`.

Some [Expression Language](#) functions are:

- [toUpper()](#) `toUpper()` converts [literal text](#) or [variables](#) and [parameters](#) and [literal text](#) to [upper case](#).
- [toLower()](#) `toLower()` converts [literal text](#) or [variables](#) and [parameters](#) and [literal text](#) to [lower case](#).
- [trim()](#) `trim()` removes [whitespace](#) from [literal text](#) or [variables](#) and [parameters](#) and [literal text](#).

[Expression Language](#) functions are [separated](#) with [:](#) and [()](#) and can be [chained](#) together.

Some [Expression Language](#) functions are:

- [replace()](#) `replace()` replaces [literal text](#) or [variables](#) and [parameters](#) and [literal text](#) with [literal text](#) or [variables](#) and [parameters](#) and [literal text](#).
- [replaceAll()](#) `replaceAll()` replaces [literal text](#) or [variables](#) and [parameters](#) and [literal text](#) with [literal text](#) or [variables](#) and [parameters](#) and [literal text](#) using [regular expressions](#).

[Expression Language](#) functions are [separated](#) with [:](#) and [()](#) and can be [chained](#) together.

Some [Expression Language](#) functions are:

- [substring()](#) `substring()` returns [literal text](#) or [variables](#) and [parameters](#) and [literal text](#) [substrings](#) between [numbers](#) [separated](#) with [,](#).
- [substringAfter()](#) `substringAfter()` returns [literal text](#) or [variables](#) and [parameters](#) and [literal text](#) [substrings](#) after [literal text](#) or [variables](#) and [parameters](#) and [literal text](#).
- [substringBefore()](#) `substringBefore()` returns [literal text](#) or [variables](#) and [parameters](#) and [literal text](#) [substrings](#) before [literal text](#) or [variables](#) and [parameters](#) and [literal text](#).

[Expression Language](#) functions are [separated](#) with [:](#) and [()](#) and can be [chained](#) together.

Some [Expression Language](#) functions are:

- [isEmpty()](#) `isEmpty()` returns [true](#) or [false](#) if [literal text](#) or [variables](#) and [parameters](#) and [literal text](#) is [empty](#).
- [isBlank()](#) `isBlank()` returns [true](#) or [false](#) if [literal text](#) or [variables](#) and [parameters](#) and [literal text](#) is [blank](#).
- [contains()](#) `contains()` returns [true](#) or [false](#) if [literal text](#) or [variables](#) and [parameters](#) and [literal text](#) [contains](#) [literal text](#) or [variables](#) and [parameters](#) and [literal text](#).

[Expression Language](#) functions are [separated](#) with [:](#) and [()](#) and can be [chained](#) together.

Some [Expression Language](#) functions are:

- [equals()](#) `equals()` returns [true](#) or [false](#) if [literal text](#) or [variables](#) and [parameters](#) and [literal text](#) [equals](#) [literal text](#) or [variables](#) and [parameters](#) and [literal text](#).
- [equalsIgnoreCase()](#) `equalsIgnoreCase()` returns [true](#) or [false](#) if [literal text](#) or [variables](#) and [parameters](#) and [literal text](#) [equals](#) [literal text](#) or [variables](#) and [parameters](#) and [literal text](#) [ignoring case](#).

[Expression Language](#) functions are [separated](#) with [:](#) and [()](#) and can be [chained](#) together.

Some [Expression Language](#) functions are:

- [startsWith()](#) `startsWith()` returns [true](#) or [false](#) if [literal text](#) or [variables](#) and [parameters](#) and [literal text](#) [starts with](#) [literal text](#) or [variables](#) and [parameters](#) and [literal text](#).
- [endsWith()](#) `endsWith()` returns [true](#) or [false](#) if [literal text](#) or [variables](#) and [parameters](#) and [literal text](#) [ends with](#) [literal text](#) or [variables](#) and [parameters](#) and [literal text](#).

[Expression Language](#) functions are [separated](#) with [:](#) and [()](#) and can be [chained](#) together.

Some [Expression Language](#) functions are:

- [length()](#) `length()` returns [literal text](#) or [variables](#) and [parameters](#) and [literal text](#) [length](#).
- [indexOf()](#) `indexOf()` returns [literal text](#) or [variables](#) and [parameters](#) and [literal text](#) [index](#) of [literal text](#) or [variables](#) and [parameters](#) and [literal text](#).
- [lastIndexOf()](#) `lastIndexOf()` returns [literal text](#) or [variables](#) and [parameters](#) and [literal text](#) [last index](#) of [literal text](#) or [variables](#) and [parameters](#) and [literal text](#).

[Expression Language](#) functions are [separated](#) with [:](#) and [()](#) and can be [chained](#) together.

Some [Expression Language](#) functions are:

- [toString()](#) `toString()` converts [literal text](#) or [variables](#) and [parameters](#) and [literal text](#) to [string](#).
- [toNumber()](#) `toNumber()` converts [literal text](#) or [variables](#) and [parameters](#) and [literal text](#) to [number](#).
- [toDecimal()](#) `toDecimal()` converts [literal text](#) or [variables](#) and [parameters](#) and [literal text](#) to [decimal](#).

[Expression Language](#) functions are [separated](#) with [:](#) and [()](#) and can be [chained](#) together.

Some [Expression Language](#) functions are:

- [plus()](#) `plus()` [adds](#) [literal text](#) or [variables](#) and [parameters](#) and [literal text](#).
- [minus()](#) `minus()` [subtracts](#) [literal text](#) or [variables](#) and [parameters](#) and [literal text](#).
- [multiply()](#) `multiply()` [multiplies](#) [literal text](#) or [variables](#) and [parameters](#) and [literal text](#).
- [divide()](#) `divide()` [divides](#) [literal text](#) or [variables](#) and [parameters](#) and [literal text](#).
- [mod()](#) `mod()` [modulus](#) [literal text](#) or [variables](#) and [parameters](#) and [literal text](#).

[Expression Language](#) functions are [separated](#) with [:](#) and [()](#) and can be [chained](#) together

Work with templates

Olga Semina

Contents

To Table of Contents

Manage templates
- Create a template
- Export the created template as an XML file
- Import template

Use a template

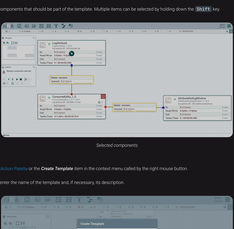
Templates in the NiFi Server interface are used when it is necessary to create several typical identical sections in a large flow scheme. Using a template, it is possible to group basic components created with toolbar.

Manage templates

Templates are managed in the **Templates** window, which opens when you click the menu item of the same name in the global menu of the NiFi Server interface window. This window contains all the templates that can be used in this instance of NiFi.

Possible actions on templates in the **Templates** window:

- filter templates by name or description;
- export a template as an XML file;
- delete a template.



The Templates window in the global menu

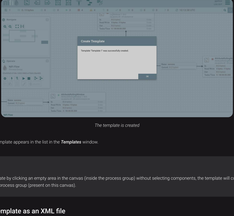
There are two ways to add a template to the list of templates:

- create a template in the current instance of NiFi,
- import a template as an XML file exported in another NiFi instance.

Create a template

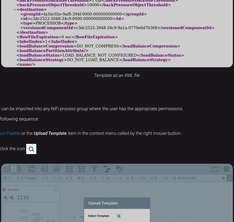
The template is created in the following sequence:

1. On the canvas, select the components that should be part of the template. Multiple items can be selected by holding down the SHIFT key.



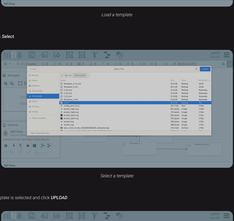
Selected components

2. Click the icon in the Action Palette or the *Create Template* item in the context menu called by the right mouse button.
3. In the window that opens, enter the name of the template and, if necessary, its description.



Create a template

4. Click **CREATE** and get a message about the successful creation of the template.



The template is created

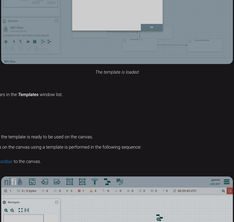
After being created, the template appears in the list in the **Templates** window.

NOTE

If you start creating a template by clicking an empty area in the canvas (inside the process group) without selecting components, the template will contain all the components of the current process group (present on this canvas).

Export the created template as an XML file

After the template appears in the list of the **Templates** window, it is possible to save this template as an XML file. To do this, in the line with the selected template, click the icon .
After loading the <template_name>.xml file, you can view its XML representation.

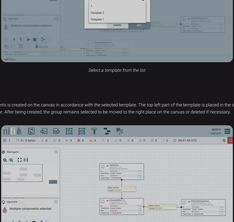


Template as an XML file

Import template

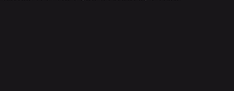
A template saved as an XML file can be imported into any NiFi process group where the user has the appropriate permissions.
The template is imported in the following sequence:

1. Click the icon in Action Palette or the *Upload Template* item in the context menu called by the right mouse button.
2. In the window that opens, click the icon .



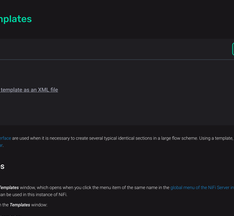
Load a template

3. Select a template and click **Select**



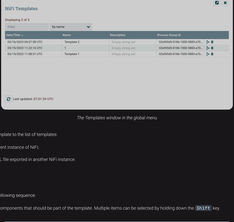
Select a template

4. Make sure the correct template is selected and click **UPLOAD**.



Load the template

5. Receive a message about the successful loading of the template.



The template is loaded

After import, the template appears in the **Templates** window list.

Use a template

After being created or imported, the template is ready to be used on the canvas.
Creating a group of components on the canvas using a template is performed in the following sequence:

1. Move icon from the toolbar to the canvas.

Create a group of components using a template

2. In the window that opens, select a template from the list.

Select a template from the list

3. Click **ADD**

As a result, a group of components is created on the canvas in accordance with the selected template. The top left part of the template is placed in the same area where the template icon was moved from the toolbar. After being created, the group remains selected to be moved to the right place on the canvas or deleted if necessary.

Group of components created from template

Components created using a template are used in the flow in the same way as components created directly from the toolbar.

Built-in monitoring in the NiFi interface

Data Ideas Sp. z o.o.

Contents

To Table of Contents

Status bar

Summary of the system components

- Component operation data
  - Work with component operation data
  - System diagnostics data

Event counters

Bulletins

Processor

Process group

Remote process group

Queue

This article describes how the NiFi UI presents information about a flow and its components to monitor its health and state.

Status bar

General system status information is provided on NiFi Server interface status bar.

Summary of the system components

A summary of data on the operation of all system components is presented in the **Summary** window, which opens when you click the menu item of the same name in the global menu of the NiFi Server interface.

The summary provides information about all core components created with Apache in a table format.

Window Summary of the global menu

Component operation data

Information about the selected component can be obtained by clicking on the corresponding icon:

- Provides basic information about the component. The data of the window repeats the data of the **Configure** window, which opens when you click the menu item in the menu called by the right mouse button.

- Provides basic data on the operation of each cluster host. This icon is only available for a multi-host cluster.

- Displays state statistics for the selected component.

The left side provides the following information.

- **Type** – the component type.
- **Id** – the identifier of the component for which statistics are displayed.
- **Group Id** – the identifier of the process group in which the component resides.
- **Name** – the name of the component for which statistics are displayed.
- **Start** – the start time shown on the chart.
- **End** – the last time shown on the chart.
- **Min/Max/Mean** – the minimum, maximum, and average (arithmetic mean or average) values. These values are based only on the selected time range if any time range is selected. Each metric is displayed in the corresponding row.

On the right side are:

- Dropdown list of different types of metrics displayed in the chart below.
- Enlarged bar graph for easier reading. In the lower right corner of this graph is a small handle that can be dragged to resize the graph.
- Lower graph, which provides the ability to select the time range to be displayed on the upper graph.

Work with component operation data

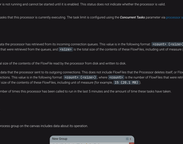
The **Summary** window uses the following elements to navigate and simplify the use of data:

- The filter at the top of the table allows users to filter the components in the table by entering all or part of some criteria such as processor type or processor name.
- allows you to update the displayed information without closing the dialog box and opening it again. The information on the page is not updated automatically.
- opens the **Summary** page and takes the user directly to the component on the NiFi canvas. This can change the process group the user is currently in.
- opens the entire **Summary** dialog to open on a new browser tab or window. In the window that opens, only data called using the tile is available for viewing.

Summary window opened on a new tab

System diagnostics data

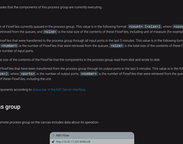
To get information about the use of system resources, click on the **system diagnostics** line in the lower right corner of the **Summary** window. The window that opens displays summary information about the system.



System diagnostics

Event counters

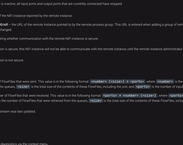
Event counters for each component are presented in the **Counters** window, which opens when you click the menu item of the same name in the global menu of the NiFi Server interface.



Event counters

Bulletins

Event bulletins are presented in the **Bulletin Board** window, which opens when you click the menu item of the same name in the global menu of the NiFi Server interface.



Bulletins

Processor

The graphical representation of the processor on the canvas includes data about the processor and its operation.

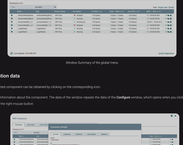
Processor

The information elements are described below using the example of the presented processor.

- **ConsumeKafka_2_0** – user-defined processor name. By default, the processor name is the same as the processor type. The processor name can be changed in processor settings on the **SETTINGS** tab.
- **ConsumeKafka_2_1.0.0.0** – processor type. The process is designed to receive messages from Apache Kafka created for the Kafka 2.0 ConsumeK API.
- **org.apache.nifi : nifi-kafka-2-0-nar** – processor launch name.
- indicates that a bulletin has been created for this processor as a result of an event. This bulletin level can be changed as processor settings on the **SETTINGS** tab. When the event occurred, hovering the mouse over the icon displays a message containing the processor as well as the bulletin level. If this NiFi instance is grouped, it will also show the node that issued the bulletin. Bulletins automatically expire after five minutes.
- Status indicator showing the current state of the processor. The following indicators are possible:
  - – the processor is currently running.
  - – the processor is valid and invalid but not running.
  - – the processor is enabled but currently invalid and cannot be started. Hovering over this icon will display a tooltip indicating why the processor is invalid.
  - – the processor is not running and cannot be started and is disabled. This status does not indicate whether the processor is valid.
- – the number of tasks that this processor is currently executing. The task limit is configured using the **Concurrent Tasks** parameter on processor settings on the **SCHEDULING** tab.
- 5 minute statistics:
  - **In** – the amount of data the processor has retrieved from its incoming connection queues. The value is in the following format: `nnnnnn (nnnnnn bytes)`, where `nnnnnn` – number of FlowFiles that were retrieved from the queues, and `nnnnnn` – the total size of the contents of these FlowFiles, including any set of measures (for example, **(0 GB / 0 MB)**).
  - **Read/Write** – the total size of the contents of the FlowFile read by the processor from disk and written to disk.
  - **Out** – the amount of data that the processor sent to its outgoing connections. This value is in the FlowFile, the size of the processor details itself, or FlowFiles that are redirected to open connection connections. The value is in the following format: `nnnnnn (nnnnnn bytes)`, where `nnnnnn` is the number of FlowFiles that were removed from the queues, and `nnnnnn` is the total size of the contents of these FlowFiles, including any set of measures (for example, **(0 GB / 0 MB)**).
  - **Tasks/Time** – the number of times this processor has been called to run in the last 5 minutes and the amount of time these tasks have taken.

Process group

The graphical display of a process group on the canvas includes data about its operation.



Process group

The information elements are described below using the example of the presented process group.

- **New group** – name defined by the user during process group creation. The process group name can be changed via process group settings on the **GENERAL** tab.
- indicates that a bulletin has been issued in the child process group components.
- – the number of tasks that the components of the process group are currently executing.
- 5 minute statistics:
  - **Queued** – the number of FlowFiles currently queued in the process group. This value is in the following format: `nnnnnn (nnnnnn bytes)`, where `nnnnnn` is the number of FlowFiles in the queue, and `nnnnnn` – the total size of the contents of these FlowFiles, including sets of measures (for example, **(0 GB / 0 MB)**).
  - **In** – the number of FlowFiles that were transferred to the process group through all input ports in the last 5 minutes. This value is in the following format: `nnnnnn (nnnnnn bytes)`, where `nnnnnn` is the number of FlowFiles that were transferred to the queues, and `nnnnnn` is the total size of the contents of these FlowFiles, including the set of measures (for example, **(0 GB / 0 MB)**).
  - **Read/Write** – the total size of the contents of the FlowFile that the components in the process group read from disk and wrote to disk.
  - **Out** – the number of FlowFiles that have been transferred from the process group through its output ports in the last 5 minutes. This value is in the following format: `nnnnnn (nnnnnn bytes)`, where `nnnnnn` is the number of FlowFiles in the queue, and `nnnnnn` is the number of FlowFiles that were removed from the queues, and `nnnnnn` is the total size of the contents of these FlowFiles, including the set of measures (for example, **(0 GB / 0 MB)**).
- Status counters for components according to status bar in the NiFi Server interface.

Remote process group

The graphical display of a remote process group on the canvas includes data about its operation.

Remote process group

The information elements are described below using the remote process group as an example.

- Transfer status indicating whether data transfer is currently enabled between the current and remote NiFi instances. The following indicators are possible:
  - – transmission is active, any of the input ports or output ports is currently set to transmit.
  - – transmission is inactive, all input ports and output ports that are currently connected but stopped.
- **NiFi Host** – the name of the NiFi instance required by the remote instance.
- **http://10.57.227.59:9090/nifi** – the URL of the remote instance pointed to by the remote process group. The URL is entered when adding a group of remote processes to the canvas of the managed.
- security indicator indicating whether communication with the remote NiFi instance is secure:
  - – communication is secure. The NiFi instance will not be able to communicate with the remote instance until the remote instance administrator grants access.
  - – communication is not secure.
- 5 minute statistics:
  - **Sent** – the number of FlowFiles that have been sent. The value is in the following format: `nnnnnn (nnnnnn bytes)`, where `nnnnnn` is the number of FlowFiles that were removed from the queues, `nnnnnn` – the total size of the contents of these FlowFiles, including the set, and `nnnnnn` is the number of input ports.
  - **Received** – the number of FlowFiles that were received. The value is in the following format: `nnnnnn (nnnnnn bytes)`, where `nnnnnn` is the number of output ports, `nnnnnn` – the total size of the contents of these FlowFiles, including the set, `nnnnnn` is the number of FlowFiles that were removed from the queues.
- Time when the remote stream was last updated.

Queue

It is possible to view queue diagnostics via the context menu.

Queue context menu

View status history – displays status statistics for the queue.

Queue status statistics

List queue – will return 100 FlowFiles in the active queue according to the configured priority.

List FlowFiles

NOTE

For more information about monitoring in the Apache NiFi user interface, see Monitoring & Overflow.

Use Expression Language

By Stephane

Contents

To Table of Contents

- Expression Language structure
- Expression Language hierarchy
- Expression Language editor
- Data types for Expression Language
- Use Expression Language when working with processors
  - Variable reference example
  - Flowfile attributes reference example
  - Example of a parameter value in a parameter context

The NiFi Expression Language allows you to manipulate values of FlowFile attributes when setting properties of processors.

Expression Language can also be used

- when using a reference to the variables,
- as the value of a parameter created in the parameter context.

Expression Language structure

The NiFi Expression Language always starts with a start delimiter `${` and ends with an end delimiter `}`. Between the start and end delimiters is the text of the expression itself. In its simplest form, an expression can only consist of an attribute name. For example, `${topic.name}` will return the value of the topic name attribute.

To manipulate the value of an attribute, variables, or parameter, the Expression Language uses functions.

You can replace part of an attribute's value by calling the replace function `${topic.name:replace('a','2')}` in this case, we refer to the topic name attribute, and then we replace the letter a with the number 2.

When a function call is made on an attribute, the attribute is the subject on which the function operates. It is possible to combine multiple function calls, where the return value of the first function becomes the subject of the second function, and its return value becomes the subject of the third function, and so on. There is no limit to the number of functions that can be linked together.

Function call structure

A function call consists of 5 elements:

- Function call separator `:`
- The name of the function, in this case `replace`
- Opening parenthesis `(`
- Function arguments. The required arguments depend on which function is being called. The example above uses the arguments:
  - `'a'` – the text to be replaced,
  - `'2'` – replacement text,
- The closing parenthesis `)` indicates the end of a function call.

Any FlowFile attribute can be referenced using an expression language. However, if the attribute name contains a "special character", the attribute name must be escaped by enclosing it in quotation marks. The following characters are considered special characters: `~ ! @ # $ % ^ & * ( ) - + ` = { } [ ] | \ : ; " ' < , > . ? /` space, `\n`, `\r`, `\t`.

In addition, a number is considered a "special character" if it is the first character of an attribute name. If any of these special characters are present in an attribute, it is enclosed in single or double quotes.

There are also some functions that assume the existence of a subject. These functions are called by a simple function call at the beginning of an expression, such as `${hostname()}`. Attempting to evaluate a function with a subject with a result in an error.

Expression Language hierarchy

When using an Expression Language to refer to a property by name, there is a specific hierarchy in which NiFi will look for the value.

The system hierarchy in NiFi looks like this:

1. Finding a FlowFile for an attribute/key
2. Finding Process Group Variables for an attribute/key
3. Finding a file in the File Registry for an attribute/key
4. Finding NiFi JVM Properties for an attribute/key
5. Finding system environment variables for an attribute/key

NiFi will search and return the first occurrence of the matching property. If no matching property is found, `null` is returned.

Expression Language editor

Setting the value of processor properties is more convenient when using the Expression Language editor. As soon as the expression starts with `${`, the editor starts to highlight the parentheses and curly braces so that the user can easily tell which open brace is curly brace matches with close brace or curly brace.

Bracket selection

The editor also provides context-sensitive help, showing a list of all functions that can be used at the current cursor position. To activate this feature, press `Ctrl+Space`.

Context sensitive help

The user can also type part of a function name and then press `Ctrl+Space` to see all available functions that start with the same prefix.

- replace
- replaceAll
- replaceEmpty
- replaceFirst
- replaceNull

Quick function selection

If you continue typing, you can narrow down the displayed functions. You can also select one of the functions from the list by double-clicking it or using the arrow keys to highlight the desired function and press `Enter`.

Data types for Expression Language

Each function called value, argument value, and function return value has a specific data type. The type of the function subject **Subject Type**, the type of arguments (**Arguments**, if any), or the return function (**Return Type**), are specified in the description of each of functions.
The Expression Language supports four different data types:

- **String** – a sequence of characters that can be numbers, letters, spaces, and special characters.
- **Number** – an integer, part of one or more digits (from 0 to 9). When converted to numbers from the Date data types, they are represented as the number of milliseconds since midnight GMT on January 1, 1970.
- **Decimal** – a number that can support decimal and larger values with minimal loss of precision. More precisely, it is IEEE 754 double-precision floating point.
- **Date** – an object that contains the date and time. Using the `Date Manipulation` and `Type Coercion` functions, this data type can be converted to/from strings and numbers.
- **Boolean** – a boolean value that can be either `true` or `false`.

After evaluating the Expression Language functions, all attributes are stored as type **String**.

Use Expression Language when working with processors

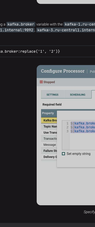
Expression Language is widely used in the NiFi Server interface to set processor properties. Not all processor properties support Expression Language. Information about whether the Expression Language is supported for the processor property is indicated in the pop-up hints for entering values under processor settings on the **PROPERTIES** tab.

Also, when you click on the ? next to the property name, you can see which scope for the Expression Language is supported:

- **NONE** – Expression Language is not supported for this property.
- **VARIABLE_REGISTRY** – Expression Language is available for this property at the variables level. In this case, the search for a variable is performed hierarchically:
  - Variables are defined at the process group level, and then, recursively, at the higher process group level up to the root process group.
  - Variables defined in custom properties files via the `nifi.variable.registry.properties` property or the nifi.properties file
  - Environment variables defined at the JVM level via system properties.
- **FLOWFILE_ATTRIBUTES** – attributes of each individual FlowFile, as well as variables can be used for Expression Language.

Variable reference example

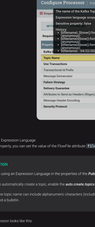
For the **PublishKafka** processor, only properties can be used when setting the **Kafka Broken** property.



Scope for Expression Language

By creating a `kafka.broker` variable with the `kafka:ip-can(n1).internal:9092` value, this value can be manipulated by setting the `kafka:ip-can(n1).internal:9092` `kafka:ip-can(n1).internal:9092` values and so on described by ipconf. An expression to replace part of a variable's value looks like this:

```
${kafka.broker:replace('1','2')}
```



Specifying multiple values with replacement of part of the text

> **NOTE**
>
> Replacement of symbols can be used with a large number of Kafka brokers with hostnames that differ by a few symbols. When creating new processors, using one or more character substitutions, you can set the value of Kafka brokers without having to write the broker name manually, enter each Kafka broker in a parameter context or as a variable.

Flowfile attribute example

Further, setting the following **Topic Name** property for the **PublishKafka** processor, you can see that this property allows the use of Flowfile attributes.

Scope for Expression Language

Scope for Expression Language
For this property, you can set the value of the FlowFile attribute `filename` and append the current time value to it using the now function with the format function.

> **CAUTION**
>
> When using an Expression Language in the properties of the **PublishKafka** processor, consider the following:
>
> - To automatically create a topic, enable the **auto.create.topics.enable** parameter in the **server.properties** group when configuring the Kafka service.
> - The topic name must include alphanumeric characters (including ASCII), as well as the characters `'_'`, `'.'` and `'-'`. Any other characters will generate an error and a bulletin.

The expression looks like this:

```
${filename}_${now():format('YYYY.MM.dd.HH')}
```

File name using the current time as the topic name

Then, if the value of the `filename` attribute in the given FlowFile is `dataflow`, the **PublishKafka** processor publishes the data in the topic `dataflow_12.20.16`, where `12.20.16` is the substitution time of the topic in the format "YY.mm.dd" at the timezone "GMT+0".

> **NOTE**
>
> The file name with the current time as the topic name can be used when it is necessary to publish data from a FlowFile created from a frequently overwritten local file to Kafka topics.

Example of a parameter value in a parameter context

The **Topic Name** property of the **PublishKafka** processor can be configured via a parameter context. If you create the `Topic` parameter with the value `${filename}` and the `time` parameter with the value `${now():format('YYYY.MM.dd')}` in the parameter context, you can specify these parameters as a property.

The expression looks like this:

```
#{topic}_#{time}
```

File name using the current time as the topic name

Then, if the value of the `filename` attribute in the given FlowFile is `dataflow`, the **PublishKafka** processor publishes the data in the topic `dataflow_12.20.16`, as in the previous examples.

Encryption of NiFi configuration parameters

Olga Semme

Contents

To Table of Contents

Main information about encryption

Encrypt new data when using the service

Change a password

Encrypted sensitive data in NiFi configuration files

Main information about encryption

Once the NiFi service is installed, all sensitive plaintext values for the NiFi Server and NiFi Registry components are replaced with encrypted values in the following configuration files:

- *nifi.properties*
- *authorizers.xml*
- *login-identity-providers.xml*
- *nifi-registry.properties*

Encryption is performed automatically by ADCM using the encrypt-config tool, which reads plaintext-sensitive configuration values from files, queries the **root key**, and encrypts each value (replaces plain values with a protected value in the same file).

The default encryption algorithm used is 128/256-bit AES/GCM.

The root key is generated from the password specified in the ***Nifi config encryption password*** configuration parameter and written to the *bootstrap.conf* file by the encryption command to save and provide security scheme configuration when needed.

The *bootstrap.conf* file should only be granted read permissions to the user who can execute it (user `nifi`, user in the `nifi` group, or `root`).

Encrypt new data when using the service

Encryption for the NiFi service is supported for the following operations:

- Enable authentication and authorization using UI or Ranger plugin — in this case, all new confidential data is written to configuration files in the encrypted form.
- Enable the Expand action — this uses the existing password to encrypt sensitive data in configuration files on the new host (or multiple hosts).

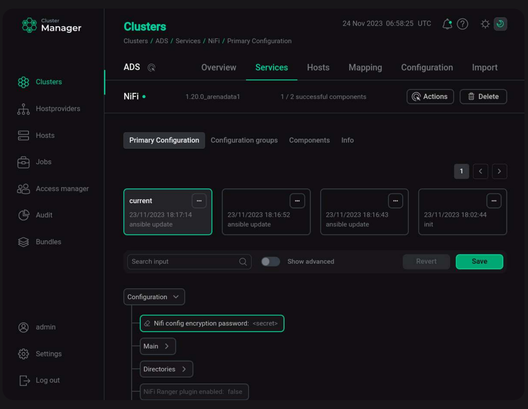
Change a password

If you need to change a password, do the following:

1. On the configuration page of the NiFi service, for the ***Nifi config encryption password*** parameter enter a new password and save the configuration by clicking ***Save***.

CAUTION

The password must be at least 12 characters long and can contain letters, numbers, and punctuation.



Changing the encryption password for NiFi configuration parameters

2. Restart the NiFi service. To do this, apply the action Restart by clicking on the in the ***Actions*** column.

Encrypted sensitive data in NiFi configuration files

The table below shows which parameters in NiFi configuration files are encrypted.

CAUTION

The parameter table is for informational purposes only. It is not recommended to make changes to configuration files.

Component	Configuration file	Parameter	Default value	Encrypted
NiFi Server	nifi.properties	nifi.sensitive.props.key	mysensetivekey	+
		nifi.security.keystorePasswd	–	+
		nifi.security.keyPasswd	–	+
		nifi.security.truststorePasswd	–	+
	nifi_cli.properties	keystorePasswd	–	–
		keyPasswd	–	–
		truststorePasswd	–	–
	login-identity-providers.xml	Manager Password	–	+
		TLS - Keystore Password	–	+
		TLS - Truststore Password	–	+
	authorizers.xml	Manager Password	–	+
		TLS - Keystore Password	–	+
		TLS - Truststore Password	–	+
NiFi Registry	nifi-registry.properties	nifi.registry.security.keystorePasswd	–	+
		nifi.registry.security.keyPasswd	–	+
		nifi.security.truststorePasswd	–	+
		nifi.registry.db.password	nifireg	+

NiFi REST API usage example

Table of Contents

Contents

- [NiFi documentation for REST API](#)
- [Create a processor using NiFi REST API](#)
- [Create a connection between processors using NiFi REST API](#)
- [Run processors using NiFi REST API](#)
- [Get parameters, attributes of Processor, Input, Process, Output using NiFi REST API](#)
  - [Create a Process Group](#)
  - [Create a Processor](#)
  - [Get Processor parameters and attributes](#)
- [Stop and terminate processors using NiFi REST API](#)
  - [Stop the processor](#)
  - [Terminate the processor](#)
- [User authorization using NiFi REST API](#)
  - [Create a User](#)
  - [Create a Policy for a User](#)
- [Flow Traffic content](#)
  - [Create a Provenance Report](#)
  - [Get Provenance Report](#)
  - [Delete Provenance Report](#)

NiFi documentation for REST API

- Apache NiFi documentation for REST API
- Apache NiFi documentation for REST API (Latest)

Create a processor using NiFi REST API

- Create a Process Group
- Create a Processor

Create a Process Group

- Apache NiFi documentation for `/process-groups/{id}/process-groups` operation
- Apache NiFi documentation for `ProcessGroupEntity` object

To create a Process Group, we need to send a `POST` request to `/process-groups/{id}/process-groups` endpoint. The `{id}` is the parent Process Group ID. If you want to create a Process Group in the root canvas, you can use `root` as the `{id}`.

The request body should be a `ProcessGroupEntity` object. The minimum required fields are `component.name` and `revision.version`.


```
{
  "component": {
    "name": "Process Group Name"
  },
  "revision": {
    "version": 0
  }
}
```


Create a Processor

- Apache NiFi documentation for `/process-groups/{id}/processors` operation
- Apache NiFi documentation for `ProcessorEntity` object

To create a Processor, we need to send a `POST` request to `/process-groups/{id}/processors` endpoint. The `{id}` is the Process Group ID where the processor will be created.

The request body should be a `ProcessorEntity` object. The minimum required fields are `component.type` and `revision.version`.


```
{
  "component": {
    "type": "org.apache.nifi.processors.standard.GenerateFlowFile",
    "bundle": {
      "group": "org.apache.nifi",
      "artifact": "nifi-standard-nar",
      "version": "1.16.3"
    }
  },
  "revision": {
    "version": 0
  }
}
```


Create a connection between processors using NiFi REST API

- Apache NiFi documentation for `/process-groups/{id}/connections` operation
- Apache NiFi documentation for `ConnectionEntity` object

To create a connection between processors, we need to send a `POST` request to `/process-groups/{id}/connections` endpoint. The `{id}` is the Process Group ID where the connection will be created.

The request body should be a `ConnectionEntity` object. The minimum required fields are `component.source.id`, `component.source.type`, `component.destination.id`, `component.destination.type`, `component.selectedRelationships` and `revision.version`.


```
{
  "component": {
    "source": {
      "id": "source-processor-id",
      "type": "PROCESSOR"
    },
    "destination": {
      "id": "destination-processor-id",
      "type": "PROCESSOR"
    },
    "selectedRelationships": ["success"]
  },
  "revision": {
    "version": 0
  }
}
```


Run processors using NiFi REST API

- Apache NiFi documentation for `/process-groups/{id}` operation
- Apache NiFi documentation for `ProcessGroupEntity` object

To run processors, we need to send a `PUT` request to `/process-groups/{id}` endpoint. The `{id}` is the Process Group ID where the processors are located.

The request body should be a `ProcessGroupEntity` object. The minimum required fields are `id` and `state`. `state` can be `RUNNING` or `STOPPED`.


```
{
  "id": "process-group-id",
  "state": "RUNNING"
}
```


Get parameters, attributes of Process, Input, Process, Output using NiFi REST API

Create a Process Group

- Apache NiFi documentation for `/process-groups/{id}/process-groups` operation
- Apache NiFi documentation for `ProcessGroupEntity` object

To create a Process Group, we need to send a `POST` request to `/process-groups/{id}/process-groups` endpoint. The `{id}` is the parent Process Group ID. If you want to create a Process Group in the root canvas, you can use `root` as the `{id}`.

The request body should be a `ProcessGroupEntity` object. The minimum required fields are `component.name` and `revision.version`.


```
{
  "component": {
    "name": "Process Group Name"
  },
  "revision": {
    "version": 0
  }
}
```


Create a Processor

- Apache NiFi documentation for `/process-groups/{id}/processors` operation
- Apache NiFi documentation for `ProcessorEntity` object

To create a Processor, we need to send a `POST` request to `/process-groups/{id}/processors` endpoint. The `{id}` is the Process Group ID where the processor will be created.

The request body should be a `ProcessorEntity` object. The minimum required fields are `component.type` and `revision.version`.


```
{
  "component": {
    "type": "org.apache.nifi.processors.standard.GenerateFlowFile",
    "bundle": {
      "group": "org.apache.nifi",
      "artifact": "nifi-standard-nar",
      "version": "1.16.3"
    }
  },
  "revision": {
    "version": 0
  }
}
```


Get Processor parameters and attributes

- Apache NiFi documentation for `/processors/{id}` operation
- Apache NiFi documentation for `ProcessorEntity` object

To get the processor parameters and attributes, we need to send a `GET` request to `/processors/{id}` endpoint. The `{id}` is the Processor ID.

The response body will be a `ProcessorEntity` object.


```
{
  "component": {
    "id": "processor-id",
    "name": "Processor Name",
    "type": "org.apache.nifi.processors.standard.GenerateFlowFile",
    "bundle": {
      "group": "org.apache.nifi",
      "artifact": "nifi-standard-nar",
      "version": "1.16.3"
    },
    "config": {
      "properties": {
        "File Size": "1 KB",
        "Batch Size": "1",
        "Data Format": "Text",
        "Unique FlowFiles": "false",
        "Custom Text": "Hello, World!"
      }
    }
  }
}
```


Stop and terminate processors using NiFi REST API

Stop the processor

- Apache NiFi documentation for `/process-groups/{id}` operation
- Apache NiFi documentation for `ProcessGroupEntity` object

To stop the processor, we need to send a `PUT` request to `/process-groups/{id}` endpoint. The `{id}` is the Process Group ID where the processor is located.

The request body should be a `ProcessGroupEntity` object. The minimum required fields are `id` and `state`. `state` can be `STOPPED`.


```
{
  "id": "process-group-id",
  "state": "STOPPED"
}
```


Terminate the processor

- Apache NiFi documentation for `/processors/{id}/terminate` operation

To terminate the processor, we need to send a `DELETE` request to `/processors/{id}/terminate` endpoint. The `{id}` is the Processor ID.

User authorization using NiFi REST API

- Apache NiFi documentation for `/tenants/users` operation
- Apache NiFi documentation for `UserEntity` object

Create a User

To create a User, we need to send a `POST` request to `/tenants/users` endpoint.

The request body should be a `UserEntity` object. The minimum required fields are `component.identity` and `revision.version`.


```
{
  "component": {
    "identity": "user-identity"
  },
  "revision": {
    "version": 0
  }
}
```


Create a Policy for a User

- Apache NiFi documentation for `/policies` operation
- Apache NiFi documentation for `AccessPolicyEntity` object

To create a Policy for a User, we need to send a `POST` request to `/policies` endpoint.

The request body should be an `AccessPolicyEntity` object. The minimum required fields are `component.resource`, `component.action`, `component.users` and `revision.version`.


```
{
  "component": {
    "resource": "/process-groups/process-group-id",
    "action": "read",
    "users": [
      {
        "id": "user-id"
      }
    ]
  },
  "revision": {
    "version": 0
  }
}
```


Flow Traffic content

To view FlowFile content, we need to Create a Provenance Report first and then Get Provenance Report.

Create a Provenance Report

To create a Provenance Report, we need to send a `POST` request to `/provenance` endpoint.

The request body should be a `ProvenanceEntity` object. The minimum required fields are `request.maxResults`.


```
{
  "request": {
    "maxResults": 1000
  }
}
```


Get Provenance Report

To get the Provenance Report, we need to send a `GET` request to `/provenance/{id}` endpoint. The `{id}` is the Provenance Report ID.

Delete Provenance Report

To delete the Provenance Report, we need to send a `DELETE` request to `/provenance/{id}` endpoint. The `{id}` is the Provenance Report ID.

Build a dataflow

Contents

[To Table of Contents]

[Prepare to create a dataflow]
[Configure permanent parameters]
[Write to a dataflow from a file]
[Replace a tool in a dataflow]
[Publish dataflow content to a Kafka topic]
[Get data from a Kafka topic]
[Split the data into multiple dataflows]
[Write dataflow content to a file]

This article describes an example of building a dataflow in the [Talend Cloud](#) that implements the following tasks:

- Write a dataflow from a file and a dataflow to a file.
- Use the Replace tool as an example. It contains descriptions of the processes that will be used to build the dataflow within this article.

[Prepare to create a dataflow]

Before creating a dataflow, the following steps must be taken:

1. If it is necessary to restrict access to the data, administrators configure the Safe and SFTP services in accordance with the articles:
   - [SFTP service]
   - [Safe service and Safe connections]
   - [Tag-based access]
2. Make sure that the Runtime and Data Preparation environments are provisioned. For more information, see [Talend Cloud Management Console Administrator Guide].
3. Make sure that the Runtime and Data Preparation environments are provisioned. For more information, see [Talend Cloud Management Console Administrator Guide].
4. Make sure that the Runtime and Data Preparation environments are provisioned. For more information, see [Talend Cloud Management Console Administrator Guide].

[Configure permanent parameters]

Permanent flow items can be configured before creating the dataflow.

1. Select the **Write Dictionary** connection component in the Flow and select **Add Layout** format. Assume that all connections to a dataflow are built.
2. Select **Parquet** format as the dataflow input format and select **Add Layout** format.
3. Select **Write Dictionary** connection component in the Flow and select **Add Layout** format.

Flow items configured in Parameter Context

[Write to a dataflow from a file]

Writing to a dataflow from a file involves the following sequence:

1. Select **Connections**. To create data connections you can use the connections that can be found by using the **Add** key. This task uses the processor **Add Key**. The processor is [Add Key]. The Add Key processor is [Add Key]. The Add Key processor is [Add Key].

Creating a Add Processor

2. Configure **Add** processor options.
   - Set the **Add Key** to the **Add Dictionary** parameter. Add the key that will be used to find the dataflow, the value is obtained from the **Add Key** tool.
   - Set the **Add Layout** to the **Add Layout** parameter. Add the key that will be used to find the dataflow, the value is obtained from the **Add Layout** tool.
   - Add

NiFi ADB Connector overview

Olga Semme

Contents

To Table of Contents

Overview

NiFi ADB Connector architecture

- PutGreenplumRecord
- StandartGpfdistService
- GreenplumDBCPConnectionPool

Supported data types

Overview

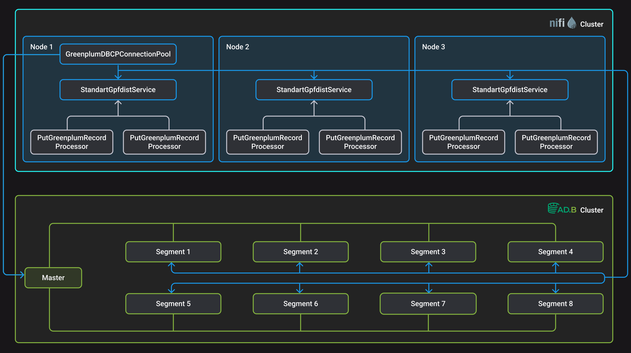
NiFi ADB Connector provides high-speed parallel writing of data from NiFi to Arenadata DB (ADB).

Starting with ADS 3.9.0.1.b1, in the NiFi user interface, Arenadata development components for creating NiFi ADB Connector are available — PutGreenplumRecord processor and StandartGpfdistService controller service.

NiFi ADB Connector architecture

PutGreenplumRecord processors collect data and pass it to the **StandartGpfdistService** controller service, which, using the built-in **gpfdist** protocol, connects to ADB via the **GreenplumDBCPConnectionPool** database connection pool service.

Greenplum Parallel File Server (gpfdist) is a Greenplum utility for reading and writing data from files located on remote servers. It is installed on all hosts of the ADB cluster and provides parallel loading of data, distributing it between segments evenly or according to the specified data distribution key.



NiFi ADB Connector architecture

PutGreenplumRecord

PutGreenplumRecord — NiFi processor that processes input records through a gpfdist receiver created by StandartGpfdistService.

The processor parameters are listed below.

Parameter	Required	Description	Default value
Gpfdist Service	true	Link to the running service of the StandartGpfdistService controller	—
Record Reader	true	Link to start one of the controller services: CSVReader, AvroReader, or another, depending on the source of input data	—
Schema Name	false	Name of the schema where the data will be loaded	null
Table Name	true	Name of the table where the data will be loaded	—
Table Columns	true	Columns of the table where the data will be loaded	—

StandartGpfdistService

StandartGpfdistService — controller service for writing data to ADB segments, using the mechanism of a readable external table with the gpfdist protocol version = 1.

The parameters of the controller service are given below.

Parameter	Required	Description	Default value
Listening Port	true	Port to listen for incoming gpfdist requests	—
Database Connection Pooling Service	true	Reference to the configured DBCPConnectionPool controller service	—
Minimum Gpffist Server Threads	false	Minimum number of threads used to run gpfdist server	1
Maximum Gpffist Server Threads	false	Maximum number of threads used to run gpfdist server	4
The Maximum Gpffist Server Threads Idle Timeout	false	Maximum idle time of gpfdist server threads in milliseconds	60000
Write buffer size in bytes	false	Size of the write byte buffer for serialized writes	1Mb
Maximum Record Processor Threads	false	Maximum number of threads used to process records	8
Maximum Gpfdist Request Processor Threads	false	Maximum number of threads used to process a gpfdist request	8

GreenplumDBCPConnectionPool

GreenplumDBCPConnectionPool — ADB connection service. Provides a database connection pool service.

Supported data types

Mapping ADB data types to NiFi record field types is given below.

ADB data type	NiFi record field data type	Comment
BIT	BOOLEAN	—
BOOLEAN	BOOLEAN	—
SMALLINT	SHORT, INT	Value size should be less than 2 bytes
INTEGER	INT	—
BIGINT	BIGINT, LONG	—
REAL	FLOAT	—
DOUBLE	DOUBLE	—
NUMERIC(p, s)	DECIMAL	—
CHAR(n)	STRING	—
VARCHAR(n)	STRING	—
ENUM	STRING	—
BYTEA	BYTE[], STRING	For the STRING type, a value should be written in hexadecimal format, for example: `\xdb78`
DATE	DATE	—
TIME(n)	TIME	—
TIMESTAMP(n)	TIMESTAMP	—
TIMESTAMPTZ(n)	TIMESTAMP	—
MONEY	DECIMAL, STRING, DOUBLE, FLOAT	—
UUID	STRING	—
JSONB	STRING	—
HSTORE	STRING, MAP(STRING, STRING)	For the STRING type, a value should be in the format `key=value, key1=value1`
ARRAY	STRING, STRING[]	For the STRING type, a value should be in the format `{value, value1}`

NiFi ADB Connector usage example

by Paul Parkhomenko

Contents

To Table of Contents

Overview
Prerequisites
- ADS
- ADPG
- ADB

Connect to PostgreSQL server (ADPG)
Connect to Greenplum (ADB)
Start a data flow

Overview

To illustrate the operation of NiFi ADB Connector, the article shows the implementation of loading data from PostgreSQL database tables into ADB tables (based on Greenplum).
The creation of NiFi ADB Connector is performed in NiFi user interface and is available starting from ADS 1.0.0 1.b1.

Prerequisites

The following describes the environment used to create the NiFi ADB Connector.

ADS

- The ADS cluster is installed according to the Online installation guide. The minimum ADS version is 1.0.0 1.b1.
- The NiFi and Zookeeper services are installed on the ADS cluster.
- Hosts where NiFi is installed have a binary JAR file of a PostgreSQL JDBC driver that allows programs to connect to a PostgreSQL database using standard Java code. The path to the file is used when configuring DBCPConnectionPool services to connect to PostgreSQL (ADPG) and Greenplum (ADB) servers.
  The command below downloads the required version of JAR file (in this example, version 42.7.6 is used) to the required directory:

```
$ wget https://jdbc.postgresql.org/download/postgresql-42.7.5.jar
```

- When configuring the DBCPConnectionPool service, a link in the format `jdbc:postgresql://<hostname>:5432/<database>` is used, where:
  - `<hostname>` is the host on which the ADPG component (for connecting to the PostgreSQL server) or the ADB Master component (for connecting to the Greenplum server) is installed.
  - `<database>` is the name of the database in which the table being used is created.

ADPG

- The ADPG cluster is installed according to the Online installation guide.
- The following preparatory actions were performed on the installed PostgreSQL server (the ADPG cluster):
  - A user named `ny_user`, with `SUPERUSER` privileges and a password, was created in the `postgres` database.
  - A `my_table` table was created in the `postgres` database and several rows with data were added to it.
  - The pg_hba.conf file is configured to provide user access from the host on which the ADS cluster NiFi service of the ADS cluster is installed. For this purpose, an entry about the host address and user is added to the `PG_HBA` field on the configuration parameters page of the ADPG service:

```
host    postgres    ny_user     10.92.43.204/32     trust
```


NOTE

- For information on working with PostgreSQL tables, see Examples of working with tables using psql.
- For information on creating a user (role) and assigning privileges in a PostgreSQL database, see database roles and privileges.

ADB

- The ADB cluster is installed according to the Online installation guide.
- The ADB service is installed in the ADB cluster.
- A user with the `new_user` name, `SUPERUSER` privileges and password has been created in the `adb` database.
- A `dg_table` table has been created in the `adb` database to record data.
- The pg_hba.conf file is configured to provide user access from the host on which the ADS cluster NiFi service is installed. To do this, an entry about the host address and user has been added to the `PG_HBA` field on the configuration parameters page of the ADB service:

```
host    adb     new_user        10.92.44.128/24     trust
```

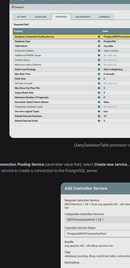
- On the interconnect network to which the cluster hosts are connected, you should specify a jumbo frame `MTU=9000` so that the packets generated by ADB `gp_max_packet_size` + overhead can fit into these frames. For more information about ADB cluster network requirements, see network requirements.

NOTE

For information on working with ADB tables, see Tables overview.

Connect to PostgreSQL server (ADPG)

1. Create a QueryDatabaseTable processor and open its configuration. This processor executes SQL queries to retrieve data from a PostgreSQL table. Fill in the parameters related to the PostgreSQL table used.



QueryDatabaseTable processor configuration

2. Go to the *Database Connection Pooling Service* parameter value field, select **Create new service...** in the pop-up list and in the window that opens, create an instance of the DBCPConnectionPool service to create a connection to the PostgreSQL server.

Creating an instance of the DBCPConnectionPool service

3. After saving the created instance, click the edit button in the next **NiFi Flow Configuration** → **Controller Services** window, open the service configuration and specify parameters related to the PostgreSQL database.

PostgreSQLDBCPConnectionPool service configuration

Creating service instances can be done before creating the processor. This can be used to connect multiple processors to a single service. To create a service:

1. Right-click in the empty Flow field and select **Configure** in the context menu that opens.
2. In the **NiFi Flow Configuration** window, go to the **Controller Services** tab and click +.
3. Select the desired service from the list and create an instance of the service in the window that opens.

Connect to Greenplum (ADB)

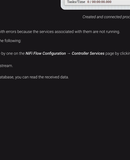
1. To connect to the Greenplum table, create a PutGreenplumRecord processor, open its configuration, and specify parameters related to the Greenplum table.

PutGreenplumRecord processor configuration

2. Go to the *Sybase Service* parameter value field, select **Create new service...** in the pop-up list, and create an instance of the StandardSybaseService service in the window that opens.

Creating an instance of the StandardSybaseService service

3. After saving the created instance, click the edit button in the next **NiFi Flow Configuration** → **Controller Services** window, open the service configuration and specify necessary parameters.



StandardSybaseService service configuration

4. Go to the *Database Connection Pooling Service* parameter value field, select **Create new service...** in the pop-up list and in the opened window, create an instance of the DBCPConnectionPool service to create a connection to Greenplum.
5. After saving the created instance, click the edit button and in the opened **NiFi Flow Configuration** → **Controller Services** window, open the service configuration and specify parameters related to the ADB database.

GreenplumDBCPConnectionPool service configuration

6. Close the **NiFi Flow Configuration** → **Controller Services** window and go back to the **PutGreenplumRecord** processor configuration. In the **Record Reader** parameter value field, create an instance of the AvroReader service to read records from PostgreSQL in Avro format with built-in schema.

If you are using multiple **PutGreenplumRecord** processors to transfer data from different sources to a single ADB database, for the *Sybase Service* value, select the same **StandardSybaseService** you created.

After all services are created, they are displayed on the **NiFi Flow Configuration** → **Controller Services** page:

Created services

The **StandardSybaseService** service fails until its associated **GreenplumDBCPConnectionPool** service starts.

Start a data flow

Create and configure a connection between processors.

Created and connected processors

Processors are displayed with errors because the services associated with them are not running.
To start the flow, perform the following:

1. Start the services one by one on the **NiFi Flow Configuration** → **Controller Services** page by clicking on the icon.
2. Run the created data stream.

Using queues to the ADB database, you can read the received data.

Schema Registry usage example

Contents

Prepare to work
- Install Python3
- Install docker-ce

Get data

Create and register a schema
- Create a schema
- Register a schema

Write data to a Kafka topic using a schema

Read messages according to the schema

Schema evolution

This article describes a usage example of the Schema Registry service – extracting data written to Kafka using the Avro format and reading data using the generated schema. Also it contains useful comments and links.

- **Serialization** is translation from a human-readable format into the machine representation for machine processing.
- **Deserialization** is the reverse process of serialization.
- **Avro** is one of data schema serialization formats.
- **Subject** is an unit of work in schema can develop.

Some of the examples in this article are based on the examples of Confluent Platform that are used in the article.

Also in this article I assume that Apache Kafka and Zookeeper are not presented on each of the hosts and Confluent Platform and schema Registry services will be on the first node.

Prepare to work

NOTE

In this section, all commands are given as alternatives (for Version 7.0). To install software packages, in other operating systems, please refer to the documentation of the software.

Install Python3

Before starting Install Python, you should install:

- Software-properties-common

```
$ sudo yum install yum-utils build-essential python-tools yum-plugin-ovl -y --skip-broken
```

- Development Tool package

```
$ sudo yum groupinstall -y "Development Tools" --skip-broken
```

- Kernel for working with SSL

```
$ sudo yum install -y zlib-devel openssl-devel sqlite-devel bzip2-devel xz-devel libffi-devel readline-devel -y --skip-broken
```

- Kernel for working with compressed JSON and Vim3 files

```
$ sudo install -y vim-devel python-devel2to3-tree
```

- **GNOME** config Accessing

```
$ sudo yum install wget
```

You can also download Python3 using the command:

```
$ wget https://www.python.org/ftp/python/3.9.10/Python-3.9.10.tgz
```

Unpack the downloaded archive:

```
$ tar xvfz Python-3.9.10.tgz
```

Then run the following commands in the directory with the unpacked files:

```
$ ./configure
$ make altinstall
```

The last command must be executed. Otherwise, you can damage the operating system.

Install libraries

You can install the `confluent-kafka` library using the command:

```
$ pip3 install confluent_kafka
```

You can install the `requests` library the command:

```
$ pip3 install requests
```

You can install the rest of the libraries using the command:

```
$ pip3 install fastavro
$ pip3 install avro-python3
$ pip3 install jsonschema
```

NOTE

When running Python3 scripts, there is a possibility of the **No module named 'avschema.client'** error, which defines that you should install the required library with the command:

```
$ pip3 install avschema.client
```

Installing the `fastavro` library, the version used `1.4.9` (pip3).

If you get the **ImportError: cannot import name 'Sequence'** error after installing on the libraries, do the following:

1. Open the Library file `/usr/local/lib/python3.9/collections/__init__.py`
2. Change lines:

```
from _collections_abc import *
from _collections_abc import __all__
```

to:

```
try:
    from _collections_abc import *
    from _collections_abc import __all__
except ImportError:
    from future_builtins import *
    from _collections_abc import __all__
```

Get data

This example uses `Wikipedia` as stream data for work.

In order to get the data from which the Avro schema will be created, use the Python3 script that executes:

1. Receiving `Wikipedia`.
2. Receiving **RANDOM**.
3. Sending data to the web-socket (127.0.0.1).
4. Connecting the client to socket (SSH client).

To receive data from Wikipedia, run the script:

```
$ nano ~/wp_thirdparty.py
```

Fill working with the script that reads:

wp_thirdparty.py

```
import requests
import json

sse_client = None
wikipedia_url = 'https://stream.wikimedia.org/v2/stream/recentchange'

def on_message(message_data):
    global sse_client
    with open('wikipedia_data.txt', 'a') as output_file:
        if message_data:
            data = json.loads(message_data)
            output_file.write(str(data) + '\n')
            print(data)
        else:
            sse_client = EventSource(wikipedia_url, on_message=on_message)
            sse_client.connect()

print('Starting to receive Wikipedia data...')
on_message(None)
```

3. Save and exit to execute the script.

Run the script:

```
$ sudo chmod +x ~/wp_thirdparty.py
```

4. Run the script:

```
$ python3 ~/wp_thirdparty.py
```

The result is data format. The result of one event is shown:

```
{'$schema': '/mediawiki/recentchange/1.0.0', 'meta': {'uri': 'https://de.wikivoyage.org/wiki/Wikivoyage', 'request_id': '7a0f0ed1-1ed1-4ed1-8ed1-ed1ed1ed1ed1', 'id': '1ed1ed1ed1ed1ed1ed1ed1ed1', 'dt': '2022-03-14T08:08:08Z', 'domain': 'de.wikivoyage.org', 'stream': 'mediawiki.recentchange', 'topic': 'eqiad.mediawiki.recentchange', 'partition': 0, 'offset': 1ed1ed1ed1}, 'id': 1ed1ed1ed1, 'type': 'edit', 'namespace': 0, 'title': 'Wikivoyage', 'comment': '/* Wikivoyage */', 'timestamp': 1ed1ed1ed1, 'user': 'Wikiwal', 'bot': False, 'minor': False, 'patrolled': True, 'length': {'old': 1ed1ed1ed1, 'new': 1ed1ed1ed1}, 'revision': {'old': 1ed1ed1ed1, 'new': 1ed1ed1ed1}, 'server_url': 'https://de.wikivoyage.org', 'server_name': 'de.wikivoyage.org', 'server_script_path': '/w', 'wiki': 'dewikivoyage', 'parsedcomment': '<span dir="auto"><span class="autocomment">Wikivoyage</span></span>'}
```

5. Save and exit to execute the script.

6. Run the script:

```
$ sudo chmod +x ~/wp_thirdparty.py
```

7. Run the script:

```
$ python3 ~/wp_thirdparty.py
```

Create and register a schema

Create a schema

Based on the received data, you need to create a schema in one of the supported Schema Registry formats. Avro format is the best.

Also in this example, the schema is created using only a few of the **fields** used in the event and the rest can be added as a new version of the schema.

In order to create a schema in the Schema Registry, you must first create a file with the schema in the Avro format. To do this, run the command:

```
$ sudo nano ~/wp_thirdparty.avsc
```

1. Fill working avsc with the created schema as shown below:

wp_thirdparty.avsc

```
{"schema": "{\"type\": \"record\", \"name\": \"value_wikistream\", \"namespace\": \"wikimedia\", \"fields\": [{\"name\": \"type\", \"type\": [\"string\", \"null\"]}, {\"name\": \"namespace\", \"type\": [\"int\", \"null\"]}, {\"name\": \"title\", \"type\": [\"string\", \"null\"]}, {\"name\": \"comment\", \"type\": [\"string\", \"null\"]}, {\"name\": \"timestamp\", \"type\": [\"int\", \"null\"]}, {\"name\": \"user\", \"type\": [\"string\", \"null\"]}, {\"name\": \"bot\", \"type\": [\"boolean\", \"null\"]}, {\"name\": \"server_url\", \"type\": [\"string\", \"null\"]}, {\"name\": \"server_name\", \"type\": [\"string\", \"null\"]}, {\"name\": \"wiki\", \"type\": [\"string\", \"null\"]}]}"}
```

2. Save and exit to execute the file script.

3. Run the script:

```
$ sudo chmod +x ~/wp_thirdparty.py
```

4. Run the script:

```
$ python3 ~/wp_thirdparty.py
```

Register a schema

In order to register a schema in the Schema Registry, use the Python3 script that executes:

1. Set the schema for entering.
2. Specify the schema format name (**TOPIC**) suffix is created in the case name of the schema, schema is created in values, not for keys.
3. Specify the schema registry URL.
4. Create a **SchemaRegistryClient** object with the specified **SCHEMA_REGISTRY_URL** parameter.
5. Define a function for receiving data with the schema ID from the Schema Registry service for the corresponding scope and returning the ID of the new schema.
6. Define a function to receive the ID of the registered schema from the Schema Registry service for the corresponding scope and returning the ID of the new schema.
7. Launch the control application functions.
8. Display the first registered schema.

To register a schema in the **SCHEMAREGISTRY** network, execute the following:

1. Create a script file.

```
$ sudo nano ~/sr_thirdparty_register.py
```

2. Fill working sr_thirdparty_register.py with the script that reads:

sr_thirdparty_register.py

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-
from confluent_kafka.schema_registry import SchemaRegistryClient
from confluent_kafka.schema_registry.avro import AvroSerializer

from uuid import uuid4, UUID4

SCHEMA_REGISTRY_URL = 'http://localhost:8081'
SCHEMA_REGISTRY_SCOPE = f'{TOPIC_NAME}-value'

def get_schema_from_schema_registry(SCHEMA_REGISTRY_URL, SCHEMA_REGISTRY_SCOPE):
    sr = SchemaRegistryClient({'url': SCHEMA_REGISTRY_URL})
    latest_version = sr.get_latest_version(SCHEMA_REGISTRY_SCOPE)
    return latest_version.schema.schema_str

def register_schema(SCHEMA_REGISTRY_URL, SCHEMA_REGISTRY_SCOPE, schema):
    sr = SchemaRegistryClient({'url': SCHEMA_REGISTRY_URL})
    schema = AvroSerializer(schema)
    schema_id = sr.register_schema(SCHEMA_REGISTRY_SCOPE, schema)
    return schema_id

def update_schema(SCHEMA_REGISTRY_URL, SCHEMA_REGISTRY_SCOPE, schema):
    sr = SchemaRegistryClient({'url': SCHEMA_REGISTRY_URL})
    schema = AvroSerializer(schema)
    schema_id = sr.update_schema(SCHEMA_REGISTRY_SCOPE, schema)
    return schema_id

sr, latest_version = get_schema_from_schema_registry(SCHEMA_REGISTRY_URL, SCHEMA_REGISTRY_SCOPE)
print('Latest schema version for {} is {}'.format(SCHEMA_REGISTRY_SCOPE, latest_version.schema_id))
```

3. Save and exit to execute the script.

4. Run the script:

```
$ sudo chmod +x ~/sr_thirdparty_register.py
```

5. Fill working sr_thirdparty_register.py with the script that reads:

```
$ python3 ~/sr_thirdparty_register.py
```

6. Run the script:

```
$ python3 ~/sr_thirdparty_register.py
```

As a result, the identifier of the newly recorded schema and the schema itself, registered under this identifier, are displayed.

NOTE

Each time you modify a script to execute the schema, first of the schema version should be different from the schema version associated with the subject. If the schema version is the same as the schema version associated with the subject, then the schema will not be added to the Schema Registry.

Write data to a Kafka topic using a schema

NOTE

Before creating a topic in **KEYSERVER**, you must ensure that the **auto.create.topics.enable** parameter in the **server.properties** program section (`/opt/kafka`) has the value **false**.

In order to write data to a Kafka topic using a schema, you must:

1. Define the **KEYSERVER_TOPIC** threshold that is shown when the message is received. If the message is not delivered, an error is logged. If the message is delivered, the log displays the name of the topic and the partition number.
2. Define the **KEYSERVER_SCHEMA** schema, which is received from the Schema Registry service.
   - Create the **KEYSERVER** string for data registered schema.
   - Configure the **KEYSERVER_SCHEMA** parameter using the **KEYSERVER_SCHEMA** schema.
   - **KEYSERVER_TOPIC_ID** – the threshold that is being sent to producer, we deliver a message (not to producer).
   - Create the **KEYSERVER_PRODUCER** producer using the **KEYSERVER_SCHEMA** schema and the **KEYSERVER_TOPIC_ID** parameter.
3. Handle streaming data (you can find it in the file I'm getting and may also, described by `data`).

To define the function of type `bytes` and update the schema in the Schema Registry service, run the following: opening a new file that is stored there.

1. Launch the **KEYSERVER** framework.

To perform the operation, the **KEYSERVER** file is stored in the format, do the following:

1. Create a script file:

```
$ sudo nano ~/wp_thirdparty_producer.py
```

2. Fill working wp_thirdparty_producer.py with the script that reads:

wp_thirdparty.py

```
#!/usr/bin/python3
import fastavro
import json

from confluent_kafka import SerializingProducer
from confluent_kafka.schema_registry import SchemaRegistryClient
from confluent_kafka.schema_registry.avro import AvroSerializer

from uuid import uuid4, UUID4

SCHEMA_REGISTRY_URL = 'http://localhost:8081'
SCHEMA_REGISTRY_SCOPE = f'{TOPIC_NAME}-value'

def get_schema_from_schema_registry(SCHEMA_REGISTRY_URL, SCHEMA_REGISTRY_SCOPE):
    sr = SchemaRegistryClient({'url': SCHEMA_REGISTRY_URL})
    latest_version = sr.get_latest_version(SCHEMA_REGISTRY_SCOPE)
    return sr, latest_version

def delivery_report(err, msg):
    """
    Reports the success or failure of a message delivery.
    Args:
        err (KafkaError): The error that occurred on None on success.
        msg (Message): The message that was produced or failed.
    """
    if err is not None:
        print("Delivery failed for User record {}: {}".format(msg.key(), err))
        return
    print('User record {} successfully produced to {} [{}] at offset {}'.format(msg.key(), msg.topic(), msg.partition(), msg.offset()))

def write_protobuf(data, key, topic, schema_registry_client) -> None:
    """
    Write protobuf data to Kafka.
    Args:
        data (str): Data to be written to Kafka.
        key (str): Key to be written to Kafka.
        topic (str): Topic to write to.
        schema_registry_client (SchemaRegistryClient): Schema Registry client instance.
    """
    value_serializer = AvroSerializer(schema_registry_client = sr,
                                      schema_str = get_schema_from_schema_registry(SCHEMA_REGISTRY_URL, SCHEMA_REGISTRY_SCOPE)[1].schema.schema_str,
                                      to_dict = None)

    producer = SerializingProducer({
        'bootstrap.servers': 'localhost:9092',
        'key.serializer': StringSerializer('utf_8'),
        'value.serializer': value_serializer,
        'delivery.timeout.ms': 120000, # set it to 2 mins
        'enable.idempotence': 'true'
    })

    try:
        producer.produce(topic=topic,
                         key=key,
                         value=data,
                         on_delivery=delivery_report)
    except ValueError:
        print("Invalid input, discarding record...")
        pass

    print("\nFlushing records...")
    producer.flush()

def get_schema_from_schema_registry(SCHEMA_REGISTRY_URL, SCHEMA_REGISTRY_SCOPE):
    sr = SchemaRegistryClient({'url': SCHEMA_REGISTRY_URL})
    latest_version = sr.get_latest_version(SCHEMA_REGISTRY_SCOPE)
    return sr, latest_version

def register_schema(SCHEMA_REGISTRY_URL, SCHEMA_REGISTRY_SCOPE, schema):
    sr = SchemaRegistryClient({'url': SCHEMA_REGISTRY_URL})
    schema = AvroSerializer(schema)
    schema_id = sr.register_schema(SCHEMA_REGISTRY_SCOPE, schema)
    return schema_id

def update_schema(SCHEMA_REGISTRY_URL, SCHEMA_REGISTRY_SCOPE, schema):
    sr = SchemaRegistryClient({'url': SCHEMA_REGISTRY_URL})
    schema = AvroSerializer(schema)
    schema_id = sr.update_schema(SCHEMA_REGISTRY_SCOPE, schema)
    return schema_id

sr, latest_version = get_schema_from_schema_registry(SCHEMA_REGISTRY_URL, SCHEMA_REGISTRY_SCOPE)
print('Latest schema version for {} is {}'.format(SCHEMA_REGISTRY_SCOPE, latest_version.schema_id))
```

3. Save and exit to execute the file script.

4. Run the file script:

```
$ sudo chmod +x ~/wp_thirdparty_producer.py
```

5. Run the script:

```
$ python3 ~/wp_thirdparty_producer.py
```

As a result of the changes made and the schema registered in the Schema Registry service, the data is stored in the **KEYSERVER** system under the subject of schema 1.

Example: `Data successfully produced to Topic wikimedia Partition [0] at offset [1]` (Partition [0] at offset [1]).

Read messages according to the schema

In order to read the received data from a Kafka topic using a consumer, use the Python3 script that executes:

1. Setting **AUTO_OFFSET_RESET** to `earliest` – moving data using the Avro format schema.
   - Set consumer messages.
   - Read messages.
   - Determine of the format of messages received.
2. Launch the **AUTO_OFFSET_RESET** framework.

To read consumer data from a **KEYSERVER** topic, do the following:

1. Create a script file:

```
$ sudo nano ~/avro_thirdparty_consumer.py
```

2. Fill working avro_thirdparty_consumer.py with the script that reads:

avro_thirdparty_consumer.py

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-
from confluent_kafka.avro import AvroConsumer
from confluent_kafka.avro.serializer import SerializerError

conf = {'bootstrap.servers': 'localhost:9092',
        'group.id': 'test',
        'schema.registry.url': 'http://localhost:8081'}

c = AvroConsumer(conf)
c.subscribe(['test'])

while True:
    try:
        msg = c.poll(10)
    except SerializerError as e:
        print("Message deserialization failed for {}: {}".format(msg, e))
        break

    if msg is None:
        continue

    if msg.error():
        print("AvroConsumer error: {}".format(msg.error()))
        continue

    print(msg.value())

c.close()
```

3. Save and exit to execute the file script.

4. Run the script:

```
$ sudo chmod +x ~/avro_thirdparty_consumer.py
```

5. Run the script:

```
$ python3 ~/avro_thirdparty_consumer.py
```

As a result, the stream of received events is printed to the print column in the form of a record in JSON.

Format: `{'key': 'value', 'timestamp': 1}` (category:1234,structure_id:bot,id:1ed1ed1ed1,`{'key': 'value', 'timestamp': 1}`).

Schema evolution

To demonstrate schema evolution, you can do the follow.

1. Add the new schema by executing the following file script: `~/sr_thirdparty_register.py`.

{"type": "record", "name": "value_wikistream", "namespace": "wikimedia", "fields": [{"name": "type", "type": ["string", "null"]}, {"name": "namespace", "type": ["int", "null"]}, {"name": "title", "type": ["string", "null"]}, {"name": "comment", "type": ["string", "null"]}, {"name": "timestamp", "type": ["int", "null"]}, {"name": "user", "type": ["string", "null"]}, {"name": "bot", "type": ["boolean", "null"]}, {"name": "server_url", "type": ["string", "null"]}, {"name": "server_name", "type": ["string", "null"]}, {"name": "wiki", "type": ["string", "null"]}]}

The outputting script with the change data has the:

{"type": "record", "name": "value_wikistream", "namespace": "wikimedia", "fields": [{"name": "type", "type": ["string", "null"]}, {"name": "namespace", "type": ["int", "null"]}, {"name": "title", "type": ["string", "null"]}, {"name": "comment", "type": ["string", "null"]}, {"name": "timestamp", "type": ["int", "null"]}, {"name": "user", "type": ["string", "null"]}, {"name": "bot", "type": ["boolean", "null"]}, {"name": "server_url", "type": ["string", "null"]}, {"name": "server_name", "type": ["string", "null"]}, {"name": "wiki", "type": ["string", "null"]}, {"name": "minor", "type": ["boolean", "null"]}, {"name": "patrolled", "type": ["boolean", "null"]}, {"name": "length", "type": ["int", "null"]}, {"name": "revision", "type": ["int", "null"]}]}

2. Run the file script to verify work:

```
$ sudo nano ~/sr_thirdparty_register.py
```

3. Run the script:

```
$ python3 ~/sr_thirdparty_register.py
```

NOTE

Using the schema data documentation can also be done using the `doc` field that supports documentation of Avro messages using the confluent Schema Registry service.

Use the `default` field to set a default value. This ensures that old data can be read by newer versions of the schema. Also, the default value is used when the field is missing in the input Avro data or when the input data contains a value of `null`. You can also use `null` as the default value for the field, but you must change the field type to a combination of the original type and `null`.

Schema evolution

To demonstrate schema evolution, you can do the follow.

1. Add the new schema by executing the following file script: `~/sr_thirdparty_register.py`.

{"type": "record", "name": "value_wikistream", "namespace": "wikimedia", "fields": [{"name": "type", "type": ["string", "null"]}, {"name": "namespace", "type": ["int", "null"]}, {"name": "title", "type": ["string", "null"]}, {"name": "comment", "type": ["string", "null"]}, {"name": "timestamp", "type": ["int", "null"]}, {"name": "user", "type": ["string", "null"]}, {"name": "bot", "type": ["boolean", "null"]}, {"name": "server_url", "type": ["string", "null"]}, {"name": "server_name", "type": ["string", "null"]}, {"name": "wiki", "type": ["string", "null"]}, {"name": "minor", "type": ["boolean", "null"]}, {"name": "patrolled", "type": ["boolean", "null"]}, {"name": "length", "type": ["int", "null"]}, {"name": "revision", "type": ["int", "null"]}]}

2. Run the file script to verify work:

```
$ sudo nano ~/sr_thirdparty_register.py
```

3. Run the script:

```
$ python3 ~/sr_thirdparty_register.py
```

- Set consumer
- Read messages
- Determine of the format of messages received

4. Launch the **AUTO_OFFSET_RESET** framework.

To read consumer data from a **KEYSERVER** topic, do the following:

1. Create a script file:

```
$ sudo nano ~/avro_thirdparty_consumer.py
```

2. Fill working avro_thirdparty_consumer.py with the script that reads:

```
$ python3 ~/avro_thirdparty_consumer.py
```

As a result, the stream of received events is printed to the print column in the form of a record in JSON.

Format: `{'key': 'value', 'timestamp': 1}` (category:1234,structure_id:bot,id:1ed1ed1ed1,`{'key': 'value', 'timestamp': 1}`).

Schema evolution

To demonstrate schema evolution, you can do the follow.

1. Add the new schema by executing the following file script: `~/sr_thirdparty_register.py`.

{"type": "record", "name": "value_wikistream", "namespace": "wikimedia", "fields": [{"name": "type", "type": ["string", "null"]}, {"name": "namespace", "type": ["int", "null"]}, {"name": "title", "type": ["string", "null"]}, {"name": "comment", "type": ["string", "null"]}, {"name": "timestamp", "type": ["int", "null"]}, {"name": "user", "type": ["string", "null"]}, {"name": "bot", "type": ["boolean", "null"]}, {"name": "server_url", "type": ["string", "null"]}, {"name": "server_name", "type": ["string", "null"]}, {"name": "wiki", "type": ["string", "null"]}, {"name": "minor", "type": ["boolean", "null"]}, {"name": "patrolled", "type": ["boolean", "null"]}, {"name": "length", "type": ["int", "null"]}, {"name": "revision", "type": ["int", "null"]}]}

2. Run the file script to verify work:

```
$ sudo nano ~/sr_thirdparty_register.py
```

3. Run the script:

```
$ python3 ~/sr_thirdparty_register.py
```

Cluster actions

Olga Samsina

Contents

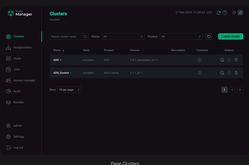
To Table of Contents

Manage a cluster

Cluster update

This article describes the main actions available in the ADCM interface for working with an ADS cluster.

Actions with the cluster are performed on the **Clusters** page:

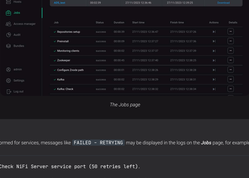


Page Clusters

The **Clusters** page contains a table with the following columns:

- **Name** – the name specified when creating the cluster.
- **State** – current state of the cluster (`created`, `installed` or `upgrade from X.X.X`).
- **Product** – product name.
- **Version** – version of the product bundle that was used to install the cluster.
- **Description** – description specified when creating the cluster.
- **Concerns**. The icon ⓘ serves as an indicator that the cluster is configured correctly. If critical errors are detected in the cluster configuration, the icon ⓘ is displayed in the column — when you hover over it, a window is displayed with a description of the error and a link that you can follow to perform the required configuration.
- **Actions**. The column shows icons for cluster management:
  - ⚙ – opens a list of actions for working with the cluster.
  - ↑ – indicates whether a new version of the bundle is available, and allows you to run cluster update to the new version.
  - 🗑 – deletes information about the cluster from ADCM (does not delete the cluster or make any changes to the hosts belonging to the cluster).

You can use the icons described above to perform the corresponding actions. Once an activity is started, ADCM displays its execution progress and result on the **Jobs** page. From this page you can go to the page of an individual task (by clicking on the task name) to see the internal stages of its execution and analyze errors if they occur.

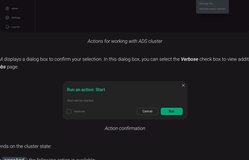


The Jobs page

> **NOTE**
>
> When actions are being performed for services, messages like `FAILED - RETRYING` may be displayed in the logs on the **Jobs** page, for example:
>
> ```
> FAILED - RETRYING: Check Nifi Server service port (50 retries left).
> FAILED - RETRYING: Check that the topic data is absent (40 retries left).
> ```
>
> This is because some processes may take a while to start. If the next retry for the current task succeeds and the action continues to execute and completes generally without errors, messages like `FAILED - RETRYING` are not an indication of a failure.
>
> If all retries have been repeated and none of them succeeded, the action is terminated and is considered unsuccessful.

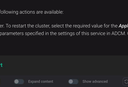
Manage a cluster

Click the icon ⚙ to open a drop-down list with available actions and select an action to run it:



Actions for working with ADS cluster

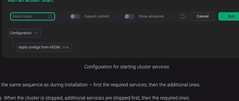
When you select an action, ADCM displays a dialog box to confirm your selection. In this dialog box, you can select the **Verbose** check box to view additional information about the execution of the action on the **Jobs** page.



Action confirmation

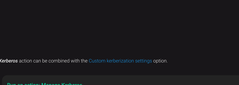
The set of available actions depends on the cluster state:

- For new cluster (cluster state `created`), the following action is available:
  - **Install** – installs all cluster services.
  - **Validate configuration** – checks for duplicate ports on each host before installing the services.
- After cluster installation (cluster state `installed`) the following actions are available:
  - **Restart** – stops and then starts all services of the cluster. To restart the cluster, select the required value for the ***Apply configs from ADCM*** option. If the value `true` is selected, then the service is launched taking into account all the parameters specified in the settings of this service in ADCM. Otherwise, the service settings in ADCM are ignored. Next, confirm the action by clicking **Run**.



Configuration for restarting cluster services

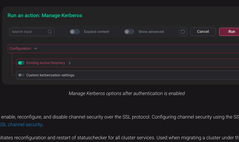
  - ***Start*** – starts the cluster services. To start the cluster, select the required value for the ***Apply configs from ADCM*** option and confirm the action by clicking **Run**.



Configuration for starting cluster services

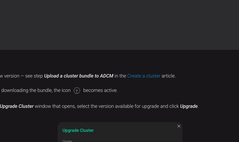
    The services are launched in the same sequence as during installation — first the required services, then the additional ones.
  - **Stop** – stops cluster services. When the cluster is stopped, additional services are stopped first, then the required ones.
  - ***Manage Kerberos*** – allows you to enable, reconfigure, and disable Kerberos authentication.
    Once you select an action, the window that appears will offer several options for running the action:
    - Existing MIT KDC
    - Existing Active Directory
    - Existing FreeIPA
    - Existing Samba

    Every option of the **Manage Kerberos** action can be combined with the Custom kerberization settings option.



Manage Kerberos options

After enabling authentication for one of the protocols, when you select the ***Manage Kerberos*** action, in the window that opens you will be asked to edit the already enabled option and the ***Custom kerberization settings*** option.



Manage Kerberos options after authentication is enabled

  - ***Manage SSL*** – allows you to enable, reconfigure, and disable channel security over the SSL protocol. Configuring channel security using the SSL protocol is carried out in accordance with the article SSL channel security.
  - ***Reinstall status-checker*** – initiates reconfiguration and restart of statuschecker for all cluster services. Used when migrating a cluster under the control of a new ADCM server.
  - ***Validate configuration*** – checks for duplicate ports on each host.
- For a cluster prepared for upgrade (the cluster state is `upgrade from X.X.X`), an available action is:
  - **Upgrade** – upgrades the cluster version.

Cluster update

> **IMPORTANT**
>
> - *Prior to upgrading, verify that the configurations on the hosts align with the ADCM configurations. If there were manual edits on the hosts, bring them into a consistent state yourself.*
> - When upgrading an ADS cluster, the product upgrade order should be as follows:
>
> 1. ADCM
> 2. ADPS
> 3. ADS

To update the cluster version:

1. Download the bundle ADS new version – see step ***Upload a cluster bundle to ADCM*** in the Create a cluster article.
2. Open the **Clusters** page. After downloading the bundle, the icon ↑ becomes active.
3. Click on the icon ↑. In the **Upgrade Cluster** window that opens, select the version available for upgrade and click **Upgrade**.

Start upgrade

4. In the dialog box that opens, click ***Run***.

Confirmation of preparation for update

As a result, the status in the **State** column changes to `upgrade from XXX` (depending on the cluster version) – the cluster is ready for upgrade.

5. Click the icon ⚙ in the **Actions** column and run the **Upgrade** action:

Performing an update

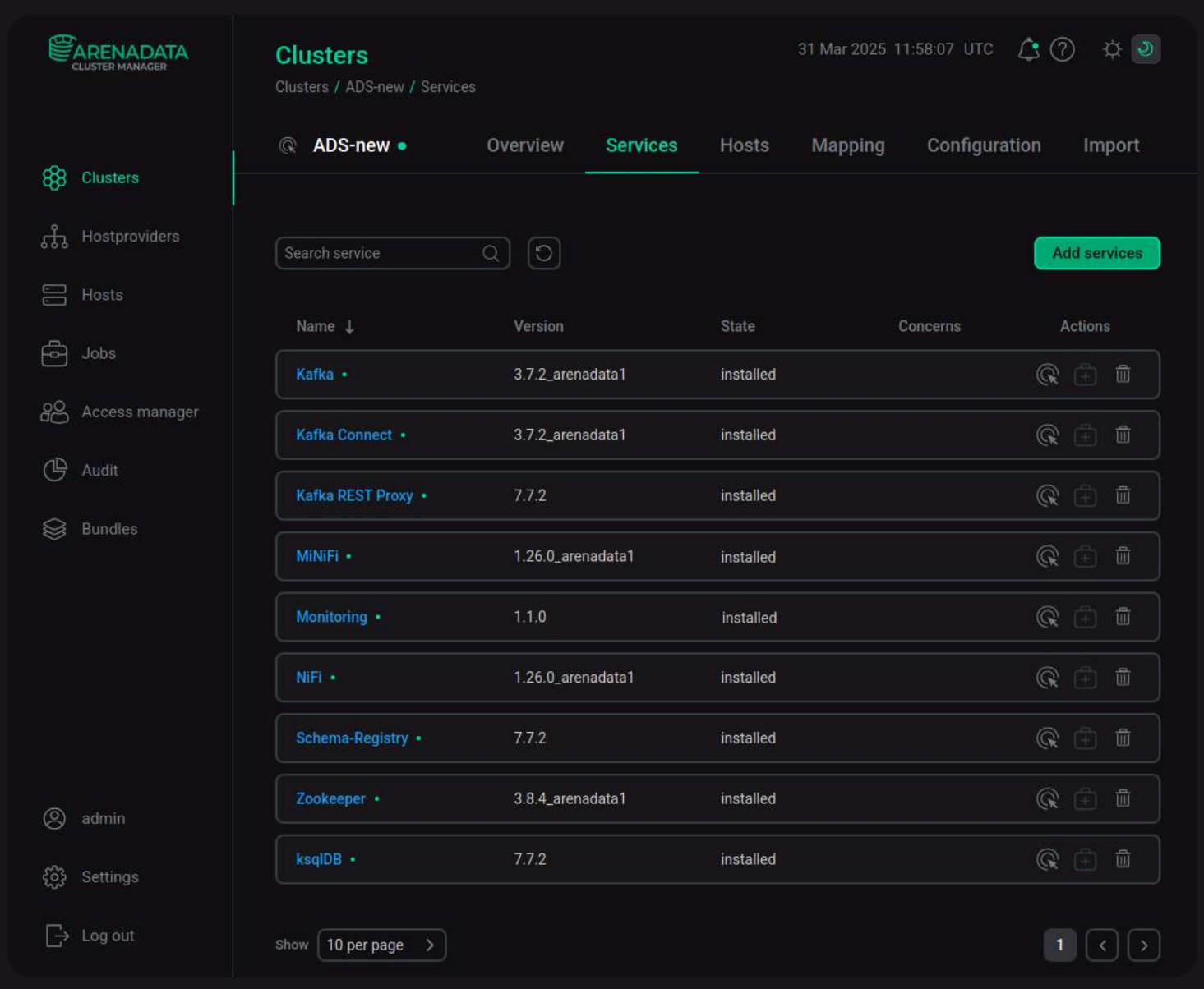
In the dialog box that opens, click ***Run*** to confirm the update. Once the upgrade process is complete, ADCM will change the cluster status to `installed`.

Service actions

Olga Semme

To Table of Contents

The ADCM user interface allows you to perform actions to manage services of ADS. To switch to the actions, click on the cluster name on the ***Clusters*** page and open the ***Services*** tab.



ADS cluster services in the ADCM interface

The ***Services*** tab contains a table with the following columns:

- ***Name*** — service name.
- ***Version*** — service version.
- ***State*** — current state of the service (for example, `created` or `installed`).
- ***Concerns***. The icon ⓘ serves as an indicator of the correct configuration of service components. If critical errors are detected in the service configuration (for example, required service components are not installed on the cluster hosts), the icon ⓘ is displayed in the column — when you hover over it, a window is displayed with a description of the error and a link that you can follow to perform the required configuration.
- ***Actions***. When you click on the icon , a list with actions to work with the service opens.
  Once the action is started, ADCM displays the process and the result of its execution on the ***Jobs*** page. Also on the ***Jobs*** page you can analyze errors if they occur.

  The set of actions depends on the type of service.

  Available actions for each service are described in the articles:

  - Kafka actions
  - Kafka Connect actions
  - Kafka REST Proxy actions
  - ksqlDB actions
  - NiFi actions
  - NiFi2 actions
  - MiNiFi actions
  - Schema-Registry actions
  - ZooKeeper actions
  - Monitoring Clients actions
  - Monitoring actions

The ***Actions*** column also separately shows icons for the following actions:

- — transfer the service to maintenance mode. This functionality is not currently implemented in this cluster.
- — delete a service from the cluster.

If the service deletion fails, on the ***Jobs*** page, you can analyze the causes of the error. Some services are interdependent. For example, before deleting some services, you need to delete the Kafka service.

Kafka actions

Olga Semme

Contents

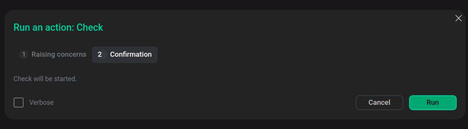
To Table of Contents

- Service actions
  - Check
  - KRaft migration stage
  - Restart
  - Start
  - Stop
  - Add/Remove components
  - Manage Ranger plugin
  - Validate configuration
- Component actions
  - Restart
  - Start
  - Stop

This article describes the main actions available for working with the service in ADCM.

To go to actions in the service, click the cluster name on the ***Clusters*** page, open the ***Services*** tab, click in the line corresponding to Kafka service, and choose the desired action.

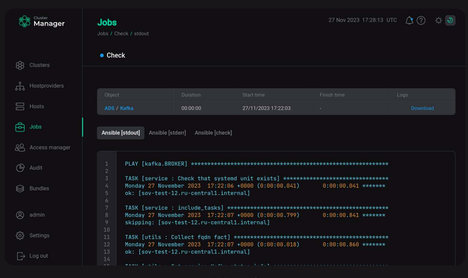
When you select an action, ADCM displays a confirmation form. In this form, on the ***Confirmation*** tab, you can select the ***Verbose*** check box to view additional information about the action execution on the ***Jobs*** page.



Action confirmation form

To view the results of an action, go to ***Jobs*** and click the name of the corresponding action or on its component.

Next, the page with the results of the action becomes available for viewing, where you can analyze errors if they occur.



Action result

To initiate action in the components of the service, from the ***Services*** tab, navigate to the service configuration parameters page by clicking on the service name. Next, from the ***Primary configuration*** tab, go to the service components configuration page.

To select an action, click in the row corresponding to the component.

Service actions

Check

In Kafka, the ***Check*** action starts the creation of test topics and checks their availability on each of the hosts with Kafka Broker.

KRaft migration stage

The ***KRaft migration stage 1*** and ***KRaft migration stage 2*** actions are used when migrating Kafka to KRaft mode.

Restart

The ***Restart*** action stops and then starts the service.

To restart the service:

1. Select the required value for the ***Apply configs from ADCM*** option. If it is set to `true`, all service configurations defined in ADCM settings will be applied during the service restart. Otherwise, the service restarts without applying configurations from ADCM.
2. Select the value of the ***Rolling restart*** option. If it is set to `true`, then rolling restart will be applied to running Java applications when the action is launched.

Start

The ***Start*** action starts the service. To start the service, select the required value for the ***Apply configs from ADCM*** option.

Stop

The ***Stop*** action stops the service.

Add/Remove components

The ***Add/Remove components*** action allows you to change the number of components installed for the service.

After selecting the action, do:

1. Select parameter values for installation.
2. Select hosts to install instances of the new component or remove hosts previously associated with the service component.

Selecting hosts for a service component

Manage Ranger plugin

The ***Manage Ranger plugin*** action triggers the Kafka Ranger plugin activation to enable Kafka service authorization.

Validate configuration

The ***Validate configuration*** action checks for duplicate ports on each host for the service.

Component actions

Restart

The ***Restart*** action stops and then starts the component. After selecting the action, specify the required value for the ***Apply configs from ADCM*** option. If it is set to `true`, all service configurations defined in ADCM settings will be applied during the component restart. Otherwise, the component restarts without applying configurations from ADCM.

Start

The ***Start*** action starts the component. After selecting the action, specify the required value for the ***Apply configs from ADCM*** option.

Stop

The ***Stop*** action stops the component.

Kafka Connect actions

Olga Semme

Contents

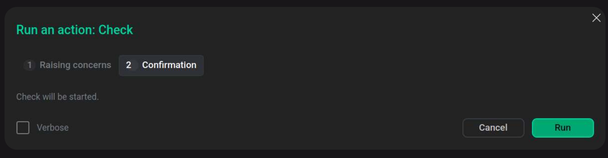
To Table of Contents

Check
Restart
Start
Stop
Add/Remove components
Validate configuration

This article describes the main actions available for working with the service in ADCM.

To go to actions in the service, click the cluster name on the ***Clusters*** page, open the ***Services*** tab, click in the line corresponding to Kafka Connect service, and choose the desired action.

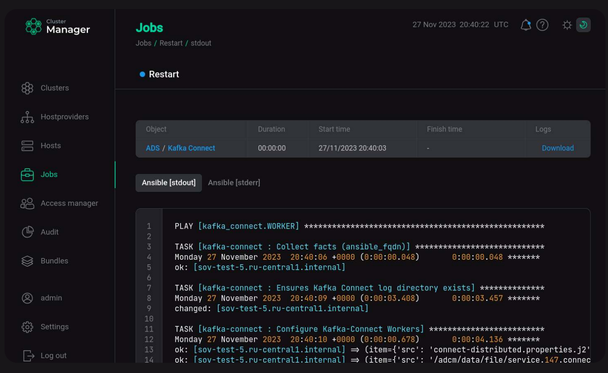
When you select an action, ADCM displays a confirmation form. In this form, on the ***Confirmation*** tab, you can select the ***Verbose*** check box to view additional information about the action execution on the ***Jobs*** page.



Action confirmation form

To view the results of an action, go to ***Jobs*** and click the name of the corresponding action or on its component.

Next, the page with the results of the action becomes available for viewing, where you can analyze errors if they occur.



Action result

Check

The ***Check*** action checks the health of the service.

In Kafka Connect, ***Check*** checks the service availability on the port and the service status.

Restart

The ***Restart*** action stops and then starts the service. To restart the service, select the required value for the ***Apply configs from ADCM*** option. If it is set to `true`, all service configurations defined in ADCM settings will be applied during the service restart. Otherwise, the service restarts without applying configurations from ADCM.

Start

The ***Start*** action starts the service. To start the service, select the required value for the ***Apply configs from ADCM*** option.

Stop

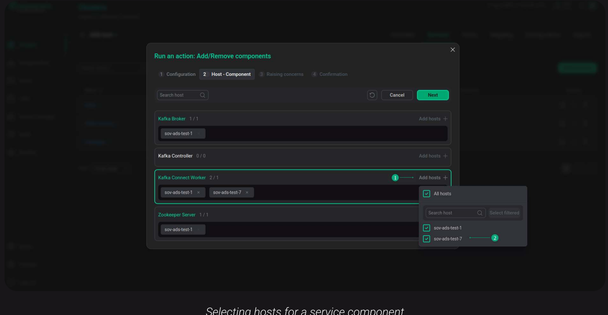
The ***Stop*** action stops the service.

Add/Remove components

The ***Add/Remove components*** action allows you to change the number of components installed for the service.

After selecting the action, do:

1. Select parameter values for installation.
2. Select hosts to install instances of the new component or remove hosts previously associated with the service component.



Selecting hosts for a service component

Validate configuration

The ***Validate configuration*** action checks for duplicate ports on each host for the service.

Kafka REST Proxy actions

Olga Semme

Contents

To Table of Contents

Check

Restart

Start

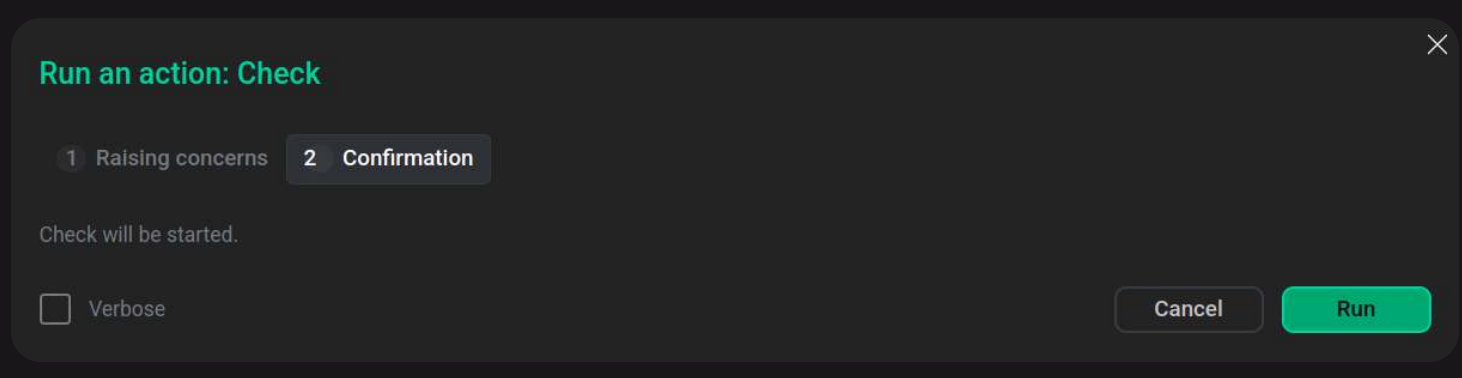
Stop

Validate configuration

This article describes the main actions available for working with the service in ADCM.

To go to actions in the service, click the cluster name on the ***Clusters*** page, open the ***Services*** tab, click in the line corresponding to Kafka REST Proxy service, and choose the desired action.

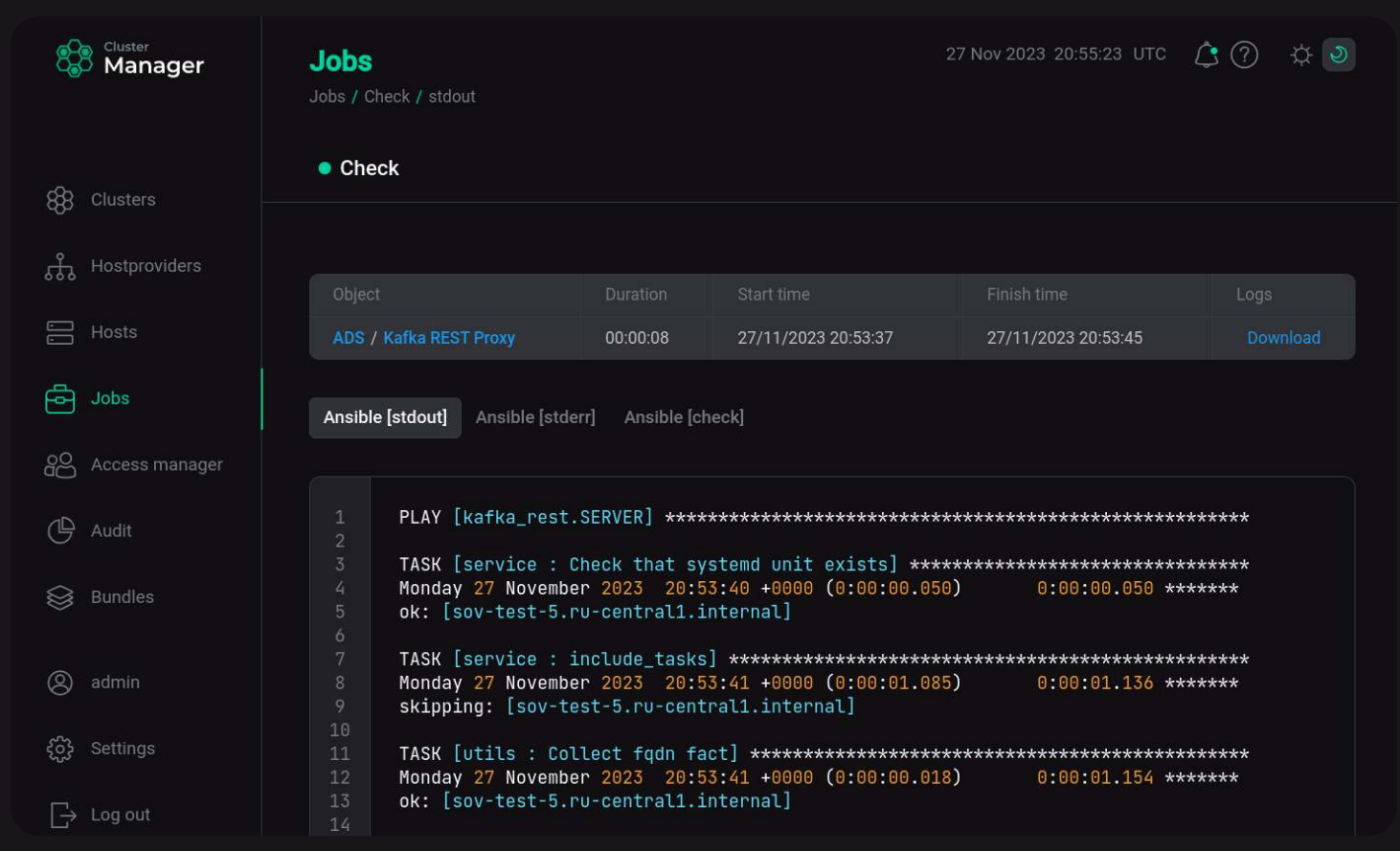
When you select an action, ADCM displays a confirmation form. In this form, on the ***Confirmation*** tab, you can select the ***Verbose*** check box to view additional information about the action execution on the ***Jobs*** page.



Action confirmation form

To view the results of an action, go to ***Jobs*** and click the name of the corresponding action or on its component.

Next, the page with the results of the action becomes available for viewing, where you can analyze errors if they occur.



Action result

Check

The ***Check*** action checks the health of the service.

In Kafka REST Proxy, ***Check*** checks the service availability on the port and the service status.

Restart

The ***Restart*** action stops and then starts the service. To restart the service, select the required value for the ***Apply configs from ADCM*** option. If it is set to `true`, all service configurations defined in ADCM settings will be applied during the service restart. Otherwise, the service restarts without applying configurations from ADCM.

Start

The ***Start*** action starts the service. To start the service, select the required value for the ***Apply configs from ADCM*** option.

Stop

The ***Stop*** action stops the service.

Validate configuration

The ***Validate configuration*** action checks for duplicate ports on each host for the service.

ksqlDB actions

Olga Semme

Contents

To Table of Contents

Check

Restart

Start

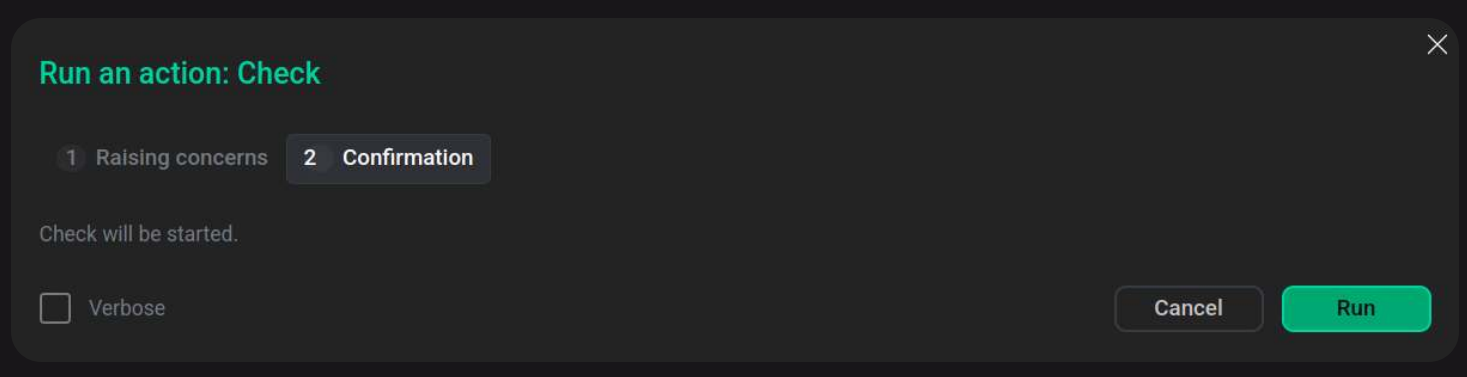
Stop

Validate configuration

This article describes the main actions available for working with the service in ADCM.

To go to actions in the service, click the cluster name on the ***Clusters*** page, open the ***Services*** tab, click in the line corresponding to ksqlDB service, and choose the desired action.

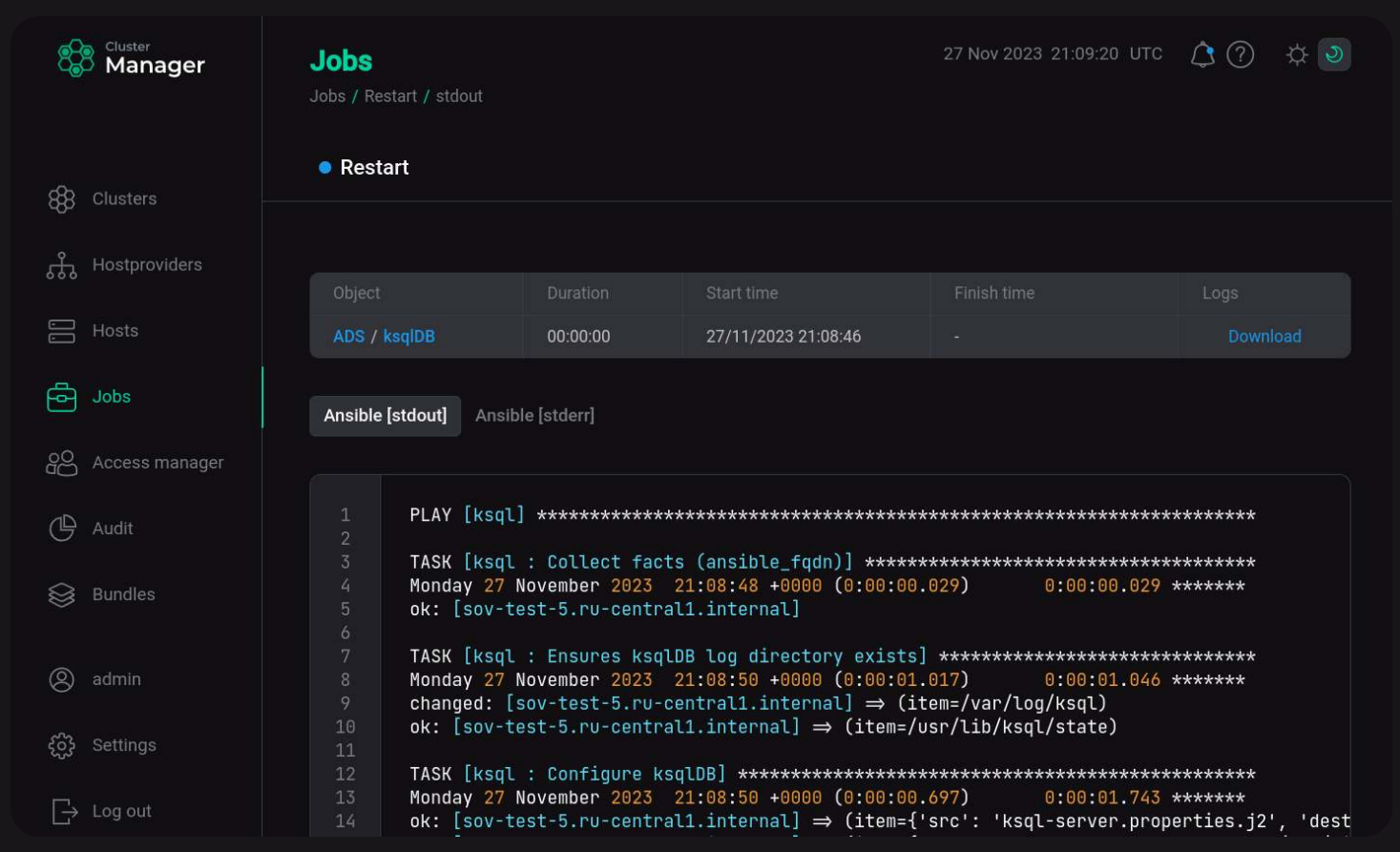
When you select an action, ADCM displays a confirmation form. In this form, on the ***Confirmation*** tab, you can select the ***Verbose*** check box to view additional information about the action execution on the ***Jobs*** page.



Action confirmation form

To view the results of an action, go to ***Jobs*** and click the name of the corresponding action or on its component.

Next, the page with the results of the action becomes available for viewing, where you can analyze errors if they occur.



Action result

Check

The ***Check*** action checks the health of the service.

In ksqlDB, ***Check*** checks the service availability on the port and the service status.

Restart

The ***Restart*** action stops and then starts the service. To restart the service, select the required value for the ***Apply configs from ADCM*** option. If it is set to `true`, all service configurations defined in ADCM settings will be applied during the service restart. Otherwise, the service restarts without applying configurations from ADCM.

Start

The ***Start*** action starts the service. To start the service, select the required value for the ***Apply configs from ADCM*** option.

Stop

The ***Stop*** action stops the service.

Validate configuration

The ***Validate configuration*** action checks for duplicate ports on each host for the service.

NiFi actions

Olga Semme

Contents

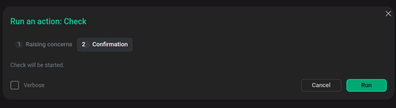
To Table of Contents

- Service actions
  - Check
  - Kafka integration
  - Restart
  - Start
  - Stop
  - Add/Remove components
  - Manage Initial Admin
  - Manage Ranger plugin
  - Validate configuration
- Component actions
  - Restart
  - Start
  - Stop

This article describes the main actions available for working with the service in ADCM.

To go to actions in the service, click the cluster name on the ***Clusters*** page, open the ***Services*** tab, click in the line corresponding to NiFi service, and choose the desired action.

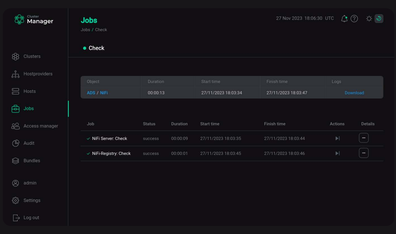
When you select an action, ADCM displays a confirmation form. In this form, on the ***Confirmation*** tab, you can select the ***Verbose*** check box to view additional information about the action execution on the ***Jobs*** page:



Action confirmation form

To view the results of an action, go to ***Jobs*** and click the name of the corresponding action or on its component.

Next, the page with the results of the action becomes available for viewing, where you can analyze errors if they occur.



Action result

To initiate action in the components of the service, from the ***Services*** tab, navigate to the service configuration parameters page by clicking on the service name. Next, from the ***Primary configuration*** tab, go to the service components configuration page.

To select an action, click in the row corresponding to the component.

NOTE

When actions are being performed for the ***NiFi*** service, messages like `FAILED - RETRYING` may be displayed in the logs on the ***Jobs*** page, for example:

```
FAILED - RETRYING: Check NiFi Server service port (50 retries left).
...
FAILED - RETRYING: Get Nifi-Registry info from Nifi Server (50 retries left).
```

This is because NiFi and NiFi Registry servers may take some time to launch due to the nature of their architecture. If the next retry for the current task succeeds and the action continues to execute and completes generally without errors, messages like `FAILED - RETRYING` are not an indication of a failure.

If all retries have been repeated and none of them succeeded, the action is terminated and is considered unsuccessful.

Service actions

Check

In NiFi2, the ***Check*** action checks the service availability on the port and the service status.

Kafka integration

The ***Kafka integration*** action performs the pre-configuration of the integration between NiFi (NiFi2) and Kafka/Schema Registry. After the action is executed, a parameters context and a controller service instance are automatically created, reflecting the current cluster topology and security settings.

Restart

The ***Restart*** action stops and then starts the service.

To restart the service, select the required value for the ***Apply configs from ADCM*** option. If it is set to `true`, all service configurations defined in ADCM settings will be applied during the service restart. Otherwise, the service restarts without applying configurations from ADCM.

Start

The ***Start*** action starts the service. To start the service, select the required value for the ***Apply configs from ADCM*** option.

Stop

The ***Stop*** action stops the service.

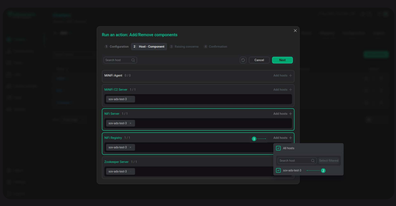
Add/Remove components

The ***Add/Remove components*** action allows you to change the number of components installed for the service.

After selecting the action, do:

1. Select parameter values for installation.
   Fill in the fields:
   - ***List of DNames to add*** — list of NiFi user IDs and systems to add to NiFi Server/Registry.
   - ***List of DNames to remove*** — list of NiFi user IDs and systems to remove from NiFi Server/Registry.

   To specify a user, click the ***+ 1*** symbol and enter the user name, for example `CN=nifi_node_hostname, OU=Arenadata, O=Arenadata, L=Moscow, ST=Moscow, C=RU`.
2. Select hosts to install instances of the new component or remove hosts previously associated with the service component.



Selecting hosts for a service component

Manage Initial Admin

The ***Manage Initial Admin*** action is used when the ***NiFi Initial Admin*** parameter needs to be changed after configuring NiFi service authentication.

Performing an action triggers:

- Change the ***NiFi Initial Admin*** setting in the ***authorizers.xml*** config settings section of the NiFi service.
- Change NiFi admin login details in NiFi Server UI and NiFi Registry UI.

Manage Ranger plugin

The ***Manage Ranger plugin*** action triggers the Ranger NiFi plugin activation to enable NiFi service authorization.

Validate configuration

The ***Validate configuration*** action checks for duplicate ports on each host for the service.

Component actions

Restart

The ***Restart*** action stops and then starts the component.

Start

The ***Start*** action starts the component.

Stop

The ***Stop*** action stops the component.

NiFi2 actions

Olga Semme

Contents

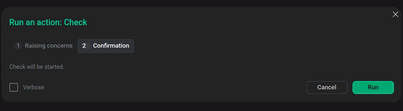
To Table of Contents

- Service actions
  - Check
  - Kafka integration
  - Restart
  - Start
  - Stop
  - Add/Remove components
  - Manage Initial Admin
  - Validate configuration
- Component actions
  - Restart
  - Start
  - Stop

This article describes the main actions available for working with the service in ADCM.

To go to actions in the service, click the cluster name on the ***Clusters*** page, open the ***Services*** tab, click in the line corresponding to NiFi2 service, and choose the desired action.

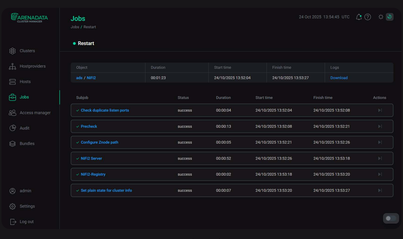
When you select an action, ADCM displays a confirmation form. In this form, on the ***Confirmation*** tab, you can select the ***Verbose*** check box to view additional information about the action execution on the ***Jobs*** page.



Action confirmation form

To view the results of an action, go to ***Jobs*** and click the name of the corresponding action or on its component.

Next, the page with the results of the action becomes available for viewing, where you can analyze errors if they occur.



Action result

To initiate action in the components of the service, from the ***Services*** tab, navigate to the service configuration parameters page by clicking on the service name. Next, from the ***Primary configuration*** tab, go to the service components configuration page.

To select an action, click in the row corresponding to the component.

NOTE

When actions are being performed for the ***NiFi*** service, messages like `FAILED - RETRYING` may be displayed in the logs on the ***Jobs*** page, for example:

```
FAILED - RETRYING: Check NiFi Server service port (50 retries left).
...
FAILED - RETRYING: Get Nifi-Registry info from Nifi Server (50 retries left).
```

This is because NiFi and NiFi Registry servers may take some time to launch due to the nature of their architecture. If the next retry for the current task succeeds and the action continues to execute and completes generally without errors, messages like `FAILED - RETRYING` are not an indication of a failure.

If all retries have been repeated and none of them succeeded, the action is terminated and is considered unsuccessful.

Service actions

Check

In NiFi2, the ***Check*** action checks the service availability on the port and the service status.

Kafka integration

The ***Kafka integration*** action performs the pre-configuration of the integration between NiFi (NiFi2) and Kafka/Schema Registry. After the action is executed, a parameters context and a controller service instance are automatically created, reflecting the current cluster topology and security settings.

Restart

The ***Restart*** action stops and then starts the service.

To restart the service, select the required value for the ***Apply configs from ADCM*** option. If it is set to `true`, all service configurations defined in ADCM settings will be applied during the service restart. Otherwise, the service restarts without applying configurations from ADCM.

Start

The ***Start*** action starts the service. To start the service, select the required value for the ***Apply configs from ADCM*** option.

Stop

The ***Stop*** action stops the service.

Add/Remove components

The ***Add/Remove components*** action allows you to change the number of components installed for the service.

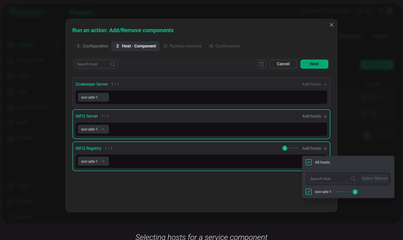
After selecting the action, do:

1. Select parameter values for installation.
   Fill in the fields:

   - ***List of DNames to add*** — list of NiFi user IDs and systems to add to NiFi Server/Registry.
   - ***List of DNames to remove*** — list of NiFi user IDs and systems to remove from NiFi Server/Registry.

   To specify a user, click the ***+ 1*** symbol and enter the user name, for example `CN=nifi_node_hostname, OU=Arenadata, O=Arenadata, L=Moscow, ST=Moscow, C=RU`.

2. Select hosts to install instances of the new component or remove hosts previously associated with the service component.



Selecting hosts for a service component

Manage Initial Admin

The ***Manage Initial Admin*** action is used when the ***NiFi Initial Admin*** parameter needs to be changed after configuring NiFi2 service authentication.

Performing an action triggers:

- Change the ***NiFi Initial Admin*** setting in the ***authorizers.xml*** config settings section of the NiFi2 service.
- Change NiFi2 admin login details in NiFi Server UI and NiFi Registry UI.

Validate configuration

The ***Validate configuration*** action checks for duplicate ports on each host for the service.

Component actions

Restart

The ***Restart*** action stops and then starts the component.

Start

The ***Start*** action starts the component.

Stop

The ***Stop*** action stops the component.

MiNiFi actions

Olga Semme

Contents

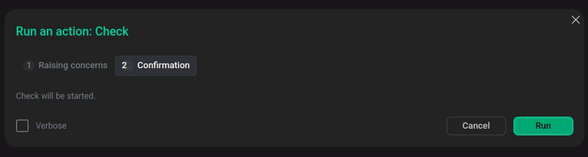
To Table of Contents

Check
Restart
Start
Stop
Add/Remove components
Validate configuration

This article describes the main actions available for working with the service in ADCM.

To go to actions in the service, click the cluster name on the ***Clusters*** page, open the ***Services*** tab, click in the line corresponding to MiNiFi service, and choose the desired action.

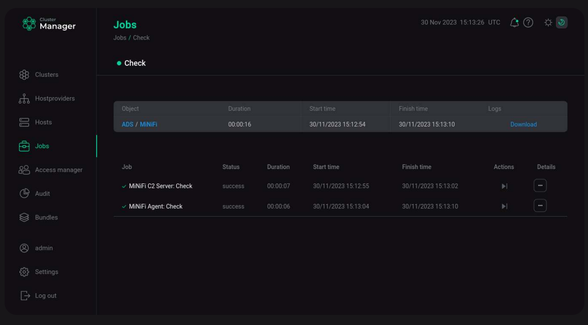
When you select an action, ADCM displays a confirmation form. In this form, on the ***Confirmation*** tab, you can select the ***Verbose*** check box to view additional information about the action execution on the ***Jobs*** page.



Action confirmation form

To view the results of an action, go to ***Jobs*** and click the name of the corresponding action or on its component.

Next, the page with the results of the action becomes available for viewing, where you can analyze errors if they occur.



Action result

Check

The ***Check*** action checks the health of the service.

In MiNiFi, ***Check*** checks the service availability on the port and the service status.

Restart

The ***Restart*** action stops and then starts the service. To restart the service, select the required value for the ***Apply configs from ADCM*** option. If it is set to `true`, all service configurations defined in ADCM settings will be applied during the service restart. Otherwise, the service restarts without applying configurations from ADCM.

Start

The ***Start*** action starts the service. To start the service, select the required value for the ***Apply configs from ADCM*** option.

Stop

The ***Stop*** action stops the service.

Add/Remove components

The ***Add/Remove components*** action allows you to change the number of components installed for the service.

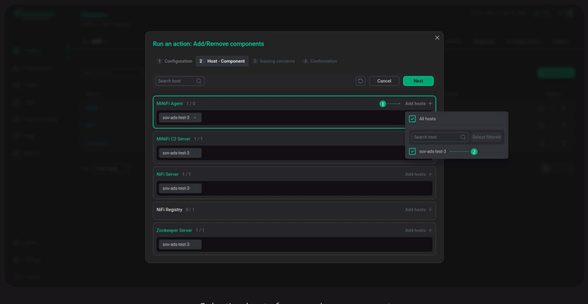
After selecting the action, do:

1. Select parameter values for installation.
   Fill in the fields:

   - ***List of DNames to add*** — list of MiNiFi user IDs and systems to add to NiFi Server/Registry.
   - ***List of DNames to remove*** — list of MiNiFi user IDs and systems to remove from NiFi Server/Registry.

   To specify a user, click the ***+ 1*** symbol and enter the user name, for example `CN=nifi_node_hostname, OU=Arenadata, O=Arenadata, L=Moscow, ST=Moscow, C=RU`.

2. Select hosts to install instances of the new component or remove hosts previously associated with the service component.



Selecting hosts for a service component

Validate configuration

The ***Validate configuration*** action checks for duplicate ports on each host for the service.

Schema-Registry actions

Olga Semme

Contents

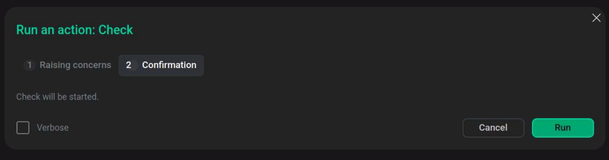
To Table of Contents

Check
Restart
Start
Stop
Add/Remove components
Validate configuration

This article describes the main actions available for working with the service in ADCM.

To go to actions in the service, click the cluster name on the ***Clusters*** page, open the ***Services*** tab, click in the line corresponding to Schema-Registry service, and choose the desired action.

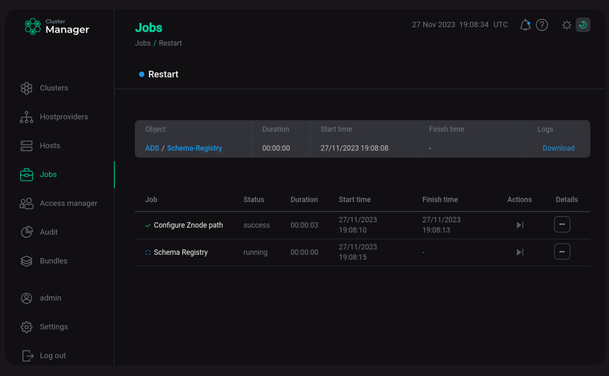
When you select an action, ADCM displays a confirmation form. In this form, on the ***Confirmation*** tab, you can select the ***Verbose*** check box to view additional information about the action execution on the ***Jobs*** page.



Action confirmation form

To view the results of an action, go to ***Jobs*** and click the name of the corresponding action or on its component.

Next, the page with the results of the action becomes available for viewing, where you can analyze errors if they occur.



Action result

Check

The ***Check*** action checks the health of the service.

In Schema-Registry, ***Check*** action starts the creation of a test topic, writes messages to it, and checks the availability of messages on each of the hosts with Kafka Broker.

Restart

The ***Restart*** action stops and then starts the service. To restart the service, select the required value for the ***Apply configs from ADCM*** option. If it is set to `true`, all service configurations defined in ADCM settings will be applied during the service restart. Otherwise, the service restarts without applying configurations from ADCM.

Start

The ***Start*** action starts the service. To start the service, select the required value for the ***Apply configs from ADCM*** option.

Stop

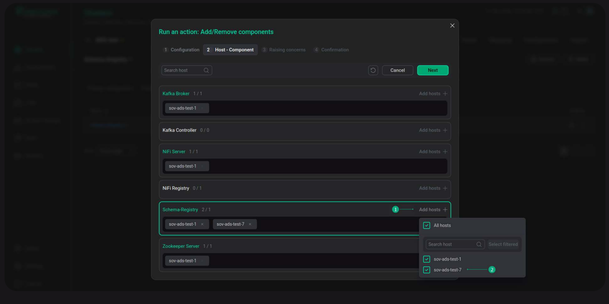
The ***Stop*** action stops the service.

Add/Remove components

The ***Add/Remove components*** action allows you to change the number of components installed for the service.

After selecting the action, do:

1. Select parameter values for installation.
2. Select hosts to install instances of the new component or remove hosts previously associated with the service component.



Selecting hosts for a service component

Validate configuration

The ***Validate configuration*** action checks for duplicate ports on each host for the service.

ZooKeeper actions

Olga Semme

Contents

To Table of Contents

Check

Restart

Start

Stop

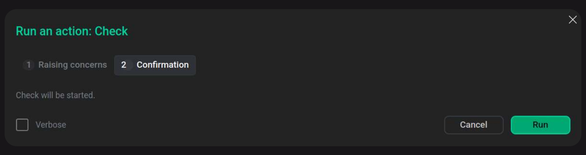
Add/Remove components

Validate configuration

This article describes the main actions available for working with the service in ADCM.

To go to actions in the service, click the cluster name on the ***Clusters*** page, open the ***Services*** tab, click in the line corresponding to ZooKeeper service, and choose the desired action.

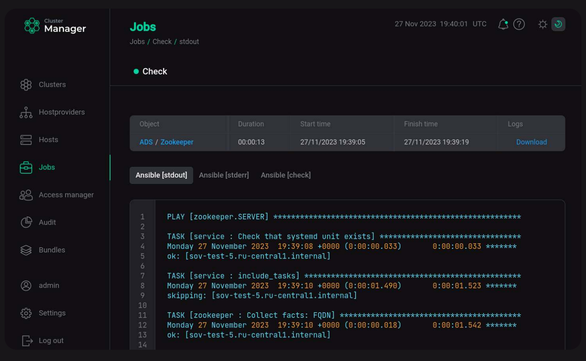
When you select an action, ADCM displays a confirmation form. In this form, on the ***Confirmation*** tab, you can select the ***Verbose*** check box to view additional information about the action execution on the ***Jobs*** page.



Action confirmation form

To view the results of an action, go to ***Jobs*** and click the name of the corresponding action or on its component.

Next, the page with the results of the action becomes available for viewing, where you can analyze errors if they occur.



Action result

Check

The ***Check*** action checks the health of the service.

In ZooKeeper, ***Check*** connects to the ZooKeeper quorum, creates a test znode in it, and checks that the created znode is available to each of the quorum hosts.

Restart

The ***Restart*** action stops and then starts the service. To restart the service, select the required value for the ***Apply configs from ADCM*** option. If it is set to `true`, all service configurations defined in ADCM settings will be applied during the service restart. Otherwise, the service restarts without applying configurations from ADCM.

Start

The ***Start*** action starts the service. To start the service, select the required value for the ***Apply configs from ADCM*** option.

Stop

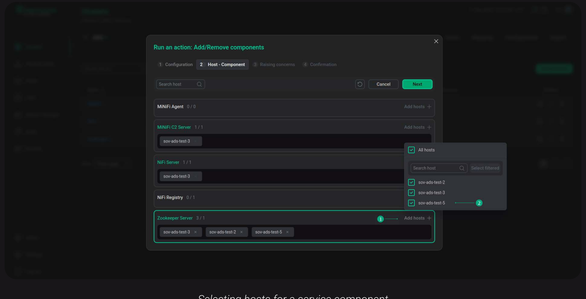
The ***Stop*** action stops the service.

Add/Remove components

The ***Add/Remove components*** action allows you to change the number of components installed for the service.

After selecting the action, do:

1. Select parameter values for installation.
2. Select hosts to install instances of the new component or remove hosts previously associated with the service component.



Selecting hosts for a service component

NOTE

The number of ZooKeeper instances must be odd.

Validate configuration

The ***Validate configuration*** action checks for duplicate ports on each host for the service.

Monitoring Clients actions

Olga Semme

Contents

To Table of Contents

Check

Restart

Start

Stop

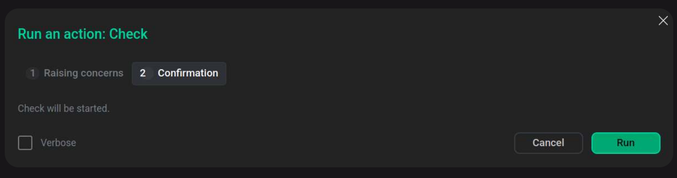
Add/Remove components

Reinstall

This article describes the main actions available for working with the service in ADCM.

To go to actions in the service, click the cluster name on the ***Clusters*** page, open the ***Services*** tab, click in the line corresponding to Monitoring Clients service, and choose the desired action.

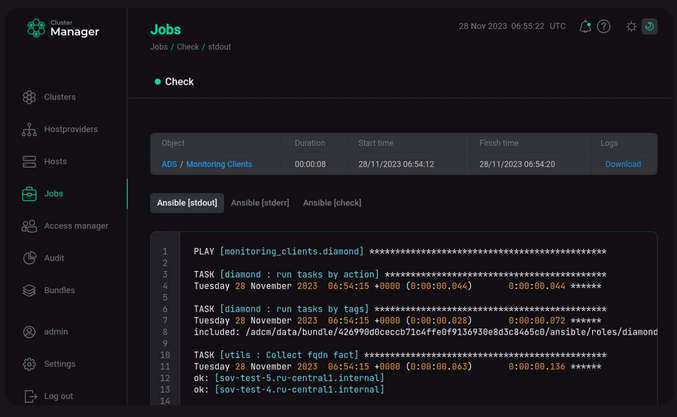
When you select an action, ADCM displays a confirmation form. In this form, on the ***Confirmation*** tab, you can select the ***Verbose*** check box to view additional information about the action execution on the ***Jobs*** page.



Action confirmation form

To view the results of an action, go to ***Jobs*** and click the name of the corresponding action or on its component.

Next, the page with the results of the action becomes available for viewing, where you can analyze errors if they occur.



Action result

Check

The ***Check*** action checks the health of the service.

In Monitoring Clients, ***Check*** checks the statuses of all service components.

Restart

The ***Restart*** action stops and then starts the service. To restart the service, select the required value for the ***Apply configs from ADCM*** option. If it is set to `true`, all service configurations defined in ADCM settings will be applied during the service restart. Otherwise, the service restarts without applying configurations from ADCM.

Start

The ***Start*** action starts the service. To start the service, select the required value for the ***Apply configs from ADCM*** option.

Stop

The ***Stop*** action stops the service.

Add/Remove components

The ***Add/Remove components*** action allows you to change the number of components installed for the service.

After selecting the action, do:

1. Select parameter values for installation.
2. Select hosts to install instances of the new component or remove hosts previously associated with the service component.

NOTE

The System metrics Agent must be installed on each host.

Reinstall

The ***Reinstall*** action reinstalls Monitoring Clients. After starting the action, set the parameters and click ***Run***.

Monitoring actions

Olga Semme

Contents

To Table of Contents

Add/Remove components

Check

Reconfigure and restart

Reinstall

Start

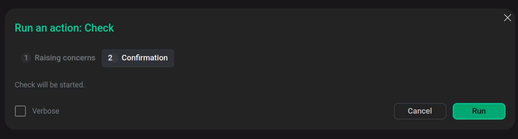
Stop

Validate configuration

This article describes the main actions available for working with the service in ADCM.

To go to actions in the service, click the cluster name on the ***Clusters*** page, open the ***Services*** tab, click in the line corresponding to the Monitoring service, and choose the desired action.

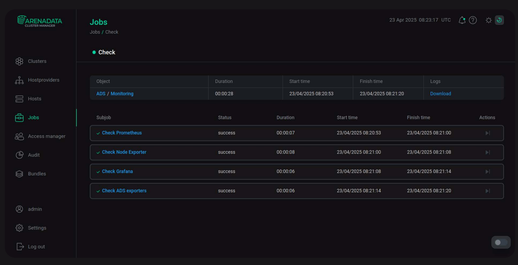
When you select an action, ADCM displays a confirmation form. In this form, on the ***Confirmation*** tab, you can select the ***Verbose*** check box to view additional information about the action execution on the ***Jobs*** page.



Action confirmation form

To view the results of an action, go to ***Jobs*** and click the name of the corresponding action or on its component.

Next, the page with the results of the action becomes available for viewing, where you can analyze errors if they occur.

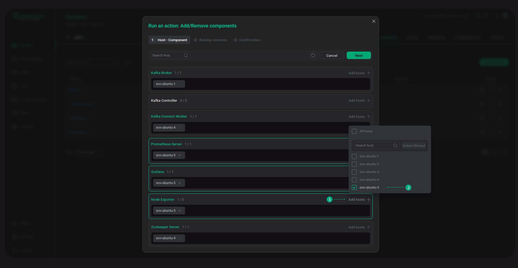


Action result

Add/Remove components

The ***Add/Remove components*** action allows you to change the number of components installed for the service.

After selecting the action, select hosts to install instances of the new component or remove hosts previously associated with the service component.



Selecting hosts for a service component

NOTE

The ***Prometheus Server*** and ***Grafana*** components can only be installed one instance per cluster.

Check

The ***Check*** action checks the health of the service.

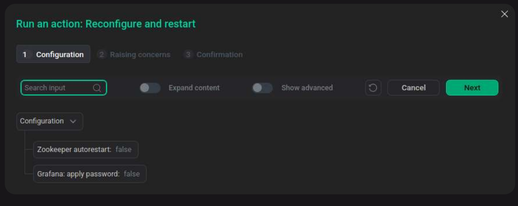
In Monitoring, the ***Check*** checks the statuses of all service components.

Reconfigure and restart

The ***Reconfigure and restart*** action allows you to reconfigure the service if configuration parameters have been changed and to stop and start the service.

After selecting the action, select parameter values if necessary:

- ***Zookeeper autorestart*** — set to `true` if you want to reconfigure and restart the ZooKeeper service.
- ***Grafana: apply password*** — set to `true` if you want to reset the password for the Grafana component.



Configuring restart parameters

Reinstall

The ***Reinstall*** action reinstalls Monitoring.

Start

The ***Start*** action starts the service.

Stop

The ***Stop*** action stops the service.

Validate configuration

The ***Validate configuration*** action checks for duplicate ports on each host for the service.

Basic authentication

Olga Semme

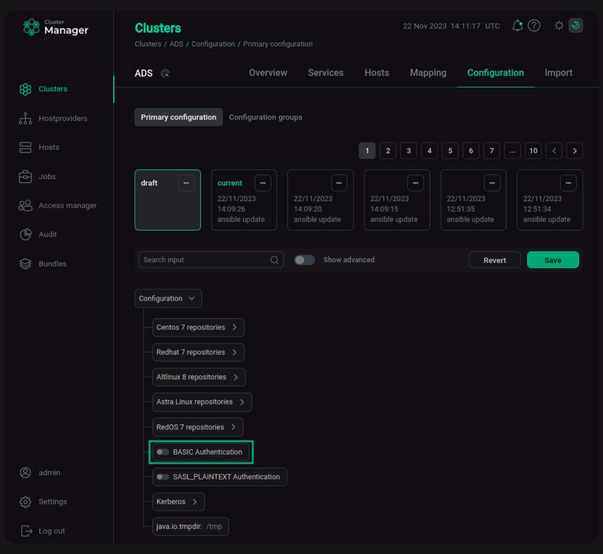
To Table of Contents

This article describes how to enable basic authentication for services:

- Schema Registry
- Kafka REST Proxy
- ksqlDB

To enable Basic authentication with ADCM, do the following:

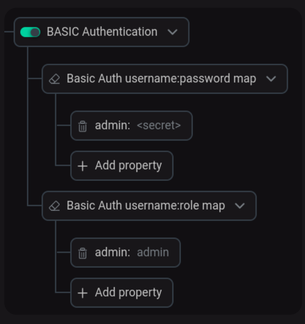
1. On the cluster configuration tab in the configuration settings tree, enable the ***BASIC Authentication*** switch.



Enabling BASIC Authentication on the ADS cluster

The nodes that open show the default values for the `admin` user:

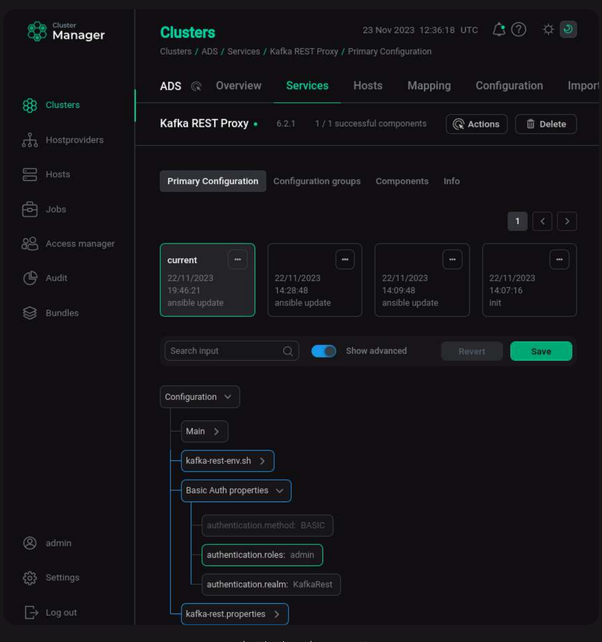
- in the node ***Basic Auth username:password map:*** password for the user (the default password for the `admin` user is `admin`);
- in the node ***Basic Auth username:role map:*** the role assigned to the user (the default role for the `admin` user is `admin`).



BASIC Authentication configuration

User roles

The roles that can be assigned to users are defined on the service configuration page using the ***authentication.roles*** parameter in the ***Basic Auth properties*** group.



authentication.roles parameter

Multiple roles can be specified separated by commas. To be authorized to work in the service, the authenticated user must belong to at least one of these roles.

2. Save the configuration by clicking ***Save***.
3. Restart all services for which authentication has been configured. To do this, apply the action Restart by clicking on the in the ***Actions*** column.

After successfully applying the ***Basic Auth*** action, the connection to the services is made after the specified user is authenticated.

MIT Kerberos

By Ajay Vohora

ADS

Contents

Configure KDC

Set up configuration files

Configure a custom jass.conf

Start MIT Kerberos KDC on ADS cluster in ADCM interface

To Table of Contents

Kerberos - network authentication protocol that provides a mechanism for mutual authentication of a client and server before establishing a connection between them.

Unix - authentication and data security framework used in network programming.

Main concepts used in Kerberos:

- **Principal** - unique name for a client that is allowed to authenticate with Kerberos.
- **Ticket** - temporary data used by the client for authentication on the server where the required service is located.
- **Realm** - network used by Kerberos, consisting of KDC servers and multiple clients.
- **Key distribution center (KDC)** - service that issues Kerberos tickets.

Configure KDC

NOTE

- KDC can be installed on any host that has access to a host with ADCM installed (FQDN connection).
- KDC can also be installed on the same host as ADCM.

To install the KDC, run the following command on the selected host:

```
$ sudo yum install -y krb5-server
```

The result is a message:

```
Complete!
```

Set up configuration files

NOTE

- Examples of commands for working with configuration files are given using the vim text editor, which must be preinstalled on this host.
- It is possible to use other text editors.

1. The configuration file for each KDC host is /etc/krb5.conf. It displays the name of the realm in which authentication is performed and the path to the configuration files. Enter the command to open the file:

```
$ sudo vim /var/kerberos/krb5kdc/kdc.conf
```

Edit the opened file as follows:

```
[kdcdefaults]
 kdc_ports = 88
 kdc_tcp_ports = 88
 default_realm = ADS-KAFKA.LOCAL

[realms]
 ADS-KAFKA.LOCAL = {
  kdc_ports = 750,88
  admin_keytab = FILE:/var/kerberos/krb5kdc/kadm5.keytab
  database_name = /var/kerberos/krb5kdc/principal
  acl_file = /var/kerberos/krb5kdc/kadm5.acl
  key_stash_file = /var/kerberos/krb5kdc/stash
  max_life = 10h 0m 0s
  max_renewable_life = 7d 0m 0s
  master_key_type = aes256-cts
  supported_enctypes = aes256-cts:normal aes128-cts:normal arcfour-hmac:normal camellia256-cts:normal camellia128-cts:normal
  default_principal_flags = +preauth
 }

[domain_realm]
```

This file contains the sections described below:

[kdcdefaults]

This section specifies the default values for the logs to be used in the `[realms]` section does not contain a value for that log.

Tag	Tag description
kdc_ports	UDP port for connecting to KDC
kdc_tcp_ports	TCP port for connecting to KDC
default_realm	Area designated by the user for authentication

[realms]

Every subsection in the section is the name of a Kerberos realm. The meaning of the subsection tags determines the KDC settings for that particular realm:

Tag	Tag description
acl_file	Location of the file. File that contains rules to determine which principals are granted which permissions in the Kerberos database
admin_keytab	Location of the keytab file that the kadmin and kadmind daemons require to authenticate with each other
database_name	Location of principal and its associated database
key_stash_file	Location of the file where the master key is stored. Before starting the KDC daemon, the machine must be configured to have this file, which is created with the kdb5_util command
kdc_ports	Specifies the default list of UDP ports for requests by Kerberos clients
kdc_tcp_ports	Specifies the default list of TCP ports for requests by Kerberos clients. An empty list means that Kerberos does not listen over TCP
max_life	Maximum time period for which a ticket can be valid
max_renewable_life	Maximum time period during which a valid ticket can be renewed
master_key_type	Primary key encryption type for the database key
supported_enctypes	Default list of key/salt combinations supported by the realm. Any realm that you intend to support must have at least one of these combinations stored in its database

NOTE

A complete description of the tag values for this file can be found at [krb.conf].

2. Next, you need to edit the kadm5.acl file to provide administrative access to the Kerberos domain.

Enter the command to open the file:

```
$ sudo vim /var/kerberos/krb5kdc/kadm5.acl
```

Edit the opened file as follows:

```
*/admin@ADS-KAFKA.LOCAL *
```

The example above is configured so that `/admin` principal in the `ADS-KAFKA.LOCAL` realm has all administrative privileges except key extraction.

NOTE

A complete description of the tag values for sections of this file can be found at [krb.acl].

3. Next, you need to edit the Kerberos domain configuration file krb5.conf.

Enter the command to open the file:

```
$ sudo vim /etc/krb5.conf
```

Edit the opened file as follows:

```
[logging]
 default = FILE:/var/log/krb5libs.log
 kdc = FILE:/var/log/krb5kdc.log
 admin_server = FILE:/var/log/kadmind.log

[libdefaults]
 default_realm = ADS-KAFKA.LOCAL
 dns_lookup_realm = false
 dns_lookup_kdc = false
 ticket_lifetime = 24h
 renew_lifetime = 7d
 forwardable = true
 rdns = false
 pkinit_anchors = FILE:/etc/pki/tls/certs/ca-bundle.crt
 default_ccache_name = KEYRING:persistent:%{uid}
 udp_preference_limit = 1
 default_tgs_enctypes = aes256-cts
 default_tkt_enctypes = aes256-cts
}

[realms]
 ADS-KAFKA.LOCAL = {
  admin_server = ads-kdc-host1.ru-central1.internal
  kdc = ads-kdc-host1.ru-central1.internal
 }

[domain_realm]
```

This file contains the sections described below:

[libdefaults]

This section describes the parameters used by the Kerberos 5 library.

Tag	Tag description
default_realm	Area designated by the user for authentication
dns_lookup_realm	Identifies the host on which the administration server is running. Typically, this is the master KDC. Clients can query DNS to find the name of the KDC. Administrators can query the domain_realm section of krb.conf
kdc	Name or address of the host running the KDC for the realm. An optional port number can be included, separated from the hostname by a colon

NOTE

A complete description of the tag values for the areas of this file can be found at [krb.conf].

4. Create a database of Kerberos principals for realm `ADS-KAFKA.LOCAL` using the kdb5_util utility.

Enter command:

```
$ sudo /usr/sbin/kdb5_util create -s -r ADS-KAFKA.LOCAL -P Password
```

As a result a message is displayed about the created database for the `ADS-KAFKA.LOCAL` area:

```
Loading random data
Initializing database '/var/kerberos/krb5kdc/principal' for realm 'ADS-KAFKA.LOCAL',
master key name 'K/M@ADS-KAFKA.LOCAL'
```

NOTE

A complete description of the functions of the kdb5_util utility and applicable options is available at [krb5_util].

5. Create user `admin` using command line interface kadmin.local.

Enter command:

```
$ sudo /usr/sbin/kadmin.local -q "addprinc admin/admin"
```

As a result a message is displayed:

```
Authenticating as principal root/admin@ADS-KAFKA.LOCAL with password.
Principal "admin/admin@ADS-KAFKA.LOCAL" created.
```

NOTE

For a complete description of the kadmin.local command line interface functions and applicable options, see [kadmin.local].

6. Restart Kerberos and KDC services.

Enter the command to service krb5:

```
$ sudo systemctl restart krb5kdc
$ sudo systemctl status krb5kdc
```

Check the status of running services by running the commands:

```
$ sudo systemctl status krb5kdc
$ sudo systemctl status kadmin
```

Configure a custom jass.conf

Before starting **Manager Kerberos**, the user file jass.conf can be configured to specify user ID for connecting other service clients to the current service (parts in regular files), the `/opt/clientconfig` parameters are stored.

The table shows the path of the location of service for a custom jass.conf file can be configured and the file storage paths for each service.

Path to the jass.conf file for services

Service	Path to jass.conf
Kafka	/etc/kafka/conf/kafka-jaas.conf
Kafka Connect	/etc/kafka-connect-jaas_config.conf
Kafka REST Proxy	/etc/kafka-rest-jaas_config.conf
Schema Registry	/etc/schema-registry-jaas_config.conf
KsqlDB	/etc/ksqldb-jaas_config.conf
Kafka REST Proxy	/etc/kafka-rest-jaas_config.conf
ZeC	/etc/zkCli.conf/zk-jaas.conf

The user can put the file /configuration.yml to specify **zkcli template** on the configuration page of the service. The configuration can be performed both for an individual service and for configuration groups defined for the service components.

Configuring jass.conf:

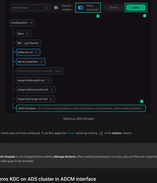
1. Turn on the **Show advanced** switch.
2. Select the Kerberos, enter changes and click the **Apply**.
To configure user data when enabling authentication via Kerberos protocol, only those **zkcli template** template blocks that describe the configuration for the ZooKeeper service are changed.
3. Below is an example of a zkcli file template for the Kafka service and a brief description of the blocks in the file:

zkcli file for Kafka

```
{% if cluster.config.kerberos_client_and_cluster_config.kerberos_client_enable.kerberos_ht
tp_enable %}
{% if ssl.security.auth.enable.kafka_enable_required
 useKeyTab=true
 storeKey=true
 keyTab="/etc/security/keytabs/zkcli.keytab_ht"
 principal="zkcli/cluster.config.kerberos_client.realm_ht"
{% else %}
Client {
 com.sun.security.auth.module.Krb5LoginModule required
 useKeyTab=true
 storeKey=true
 keyTab="/etc/security/keytabs/zkcli.keytab_ht"
 principal="zkcli/cluster.config.kerberos_client.realm_ht";
{% endif %}
{% if ssl.security.auth.enable.kafka_enable_required
 useKeyTab=true
 storeKey=true
 keyTab="/etc/security/keytabs/zkcli.keytab_ht"
 principal="zkcli/cluster.config.kerberos_client.realm_ht"
 serviceName="zookeeper";
{% else %}
zookeeper {
 com.sun.security.auth.module.Krb5LoginModule required
 useKeyTab=true
 storeKey=true
 keyTab="/etc/security/keytabs/zkcli.keytab_ht"
 principal="zkcli/cluster.config.kerberos_client.realm_ht"
 serviceName="zookeeper";
};
{% endif %}
{% if cluster.config.kerberos_client_and_cluster_config.sasl_enable.sasl_plain_users %}
KafkaClient {
 com.sun.security.auth.module.Krb5LoginModule required
 useKeyTab=true
 storeKey=true
 keyTab="/etc/security/keytabs/zkcli.keytab_ht"
 principal="zkcli/cluster.config.kerberos_client.realm_ht"
{% for user, pass in cluster.config.kerberos_client_and_cluster_config.sasl_plain_users.ite
ms() %}
 user_{{ user }}="{{ pass }}"
{% endfor %}
};
{% endif %}
```

- `{% if cond` - describes configuration parameters for connecting other services to the cluster, if condition1 is fulfilled.
- `{% elif cond2 %}` - describes configuration parameters for connecting external applications (producers and consumers) to Kafka brokers.
- `{% endif %}` - describes configuration parameters for communication between Kafka brokers.

4. Save the configuration by clicking **Save**.


Service level
Cluster level
Show advanced
Apply
Save
Configuration
Main
Log Cleaner
Kafka server
Kerberos
Server properties
Kerberos
Kerberos
Kerberos

Kerberos with MS Active Directory

Olga Semme

Contents

To Table of Contents

Prepare to run Active Directory on an ADS cluster

Run Active Directory on ADS cluster in the ADCM interface

Basic concepts used in Kerberos LDAP:

- **Active Directory** — database and set of services that connect users to the necessary network resources.
- **Lightweight directory access protocol** (**LDAP**) — application layer protocol for accessing the Active Directory directory service, used in cluster kerberization to establish user principals.
- **LDAPS** — LDAP activation using SSL/TLS protocol to create secure traffic. Creating passwords for ADS users with accounts in Active Directory during cluster kerberization is only possible using LDAPS.
- **LDAP server** — hierarchical database, a directory service based on Active Directory, used for centralized storage of accounts. The credentials include the principals against which the user is validated in Kerberos.
- **Key distribution center** (**KDC**) — access control system component responsible for servicing user requests for access to resources by providing access tickets and session keys. It uses Active Directory as the account database.
- **Distinguished name** (**DN**) is an account in Active Directory. The DN must be unique within the tree. The DN describes the content of the attributes in the tree (navigation path) for accessing a particular entry.
  A DN consists of a series of **relative distinguished names** (**RDN**) determined by moving up the tree in the direction of its root entry. RDNs are written from left to right.

An example DN for a user used in this article:

cn=admin-kafka,ou=kerberos,ou=adh,dc=ad,dc=ranger-test.

The following RDNs are assigned to this entry:

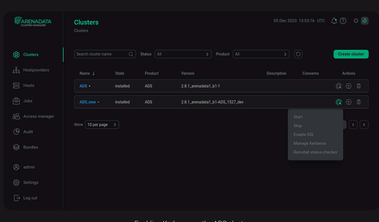
- cn — common name.
- ou — organizational unit, a container in an Active Directory domain that can contain users, groups, and computers. An organizational unit can have multiple ou within it.
- dc — parts of the domain name (domain component), represent the top of the LDAP tree, which uses DNS to determine its namespace. The notation for an Active Directory domain with a DNS name `AD.RANGER-TEST` is `dc=ad,dc =ranger-test`. The name `AD.RANGER-TEST` is also assigned to the realm that is being authenticated.

Prepare to run Active Directory on an ADS cluster

1. Set up an Active Directory server (LDAP server).
2. Create a user account to connect the cluster to 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, `admin-kafka@AD.RANGER-TEST`, and generate a *keytab* file with these principal and keys.

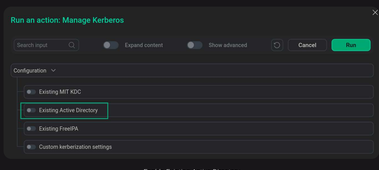
Run Active Directory on ADS cluster in the ADCM interface

1. Make sure you know the necessary data to start Active Directory:
   - **KDC hosts** — address of the host where Key Distribution Center is installed, for AD kerberization this address is the same as the address of the LDAP server;
   - **Realm** — designation of realm for authentication in Kerberos;
   - **Kadmin server** — server for installing the Kerberos V5 administration system, for kerberization on AD this address is the same as the address of the LDAP server;
   - **Kadmin principal** — designation of the user principal created to connect the cluster to Active Directory;
   - **Kadmin password** — password for the user principal created to connect the cluster to Active Directory;
   - **Admin DN** — account in Active Directory for the user used to connect the cluster to Active Directory;
   - **LDAP URL** — LDAP server address;
   - **Container DN** — distinguished name ou, which will be the default location for cluster objects created during kerberization below the user described in Admin DN;
   - **TLS CA certificate Path** — path to the generated CA certificate for the Admin DN user (you can leave it blank, then ADCM will contact the LDAP server);
   - **TLS CA certificate (optional)** — the number of the created CA certificate for the user Admin DN (it is allowed not to specify, then ADCM will contact the LDAP server)
2. Initiate enabling MS Active Directory for the selected cluster. To do this, apply the ***Manage Kerberos*** action by clicking on the in the ***Actions*** column.



Enabling Kerberos on the ADS cluster

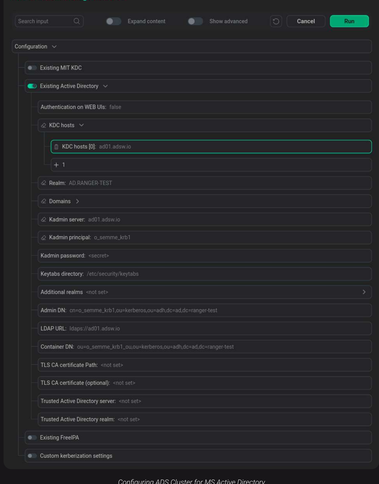
3. Enable Existing Active Directory. To do this, you need to enable the eponymous switch in the window that opens.



Enable Existing Active Directory

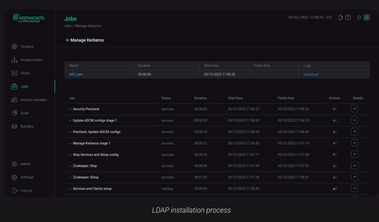
> **NOTE**
>
> Enabling ***Existing Active Directory*** can be combined with the Custom kerberization settings option.

4. Set the configuration parameters of the ADS cluster for MS Active Directory according to the values set for the user in the LDAP directory and click ***Run***.



Configuring ADS Cluster for MS Active Directory

5. Wait for the Kerberos LDAP enablement to complete. Analyze and correct errors if they occur.



LDAP installation process

> **NOTE**
>
> LDAP authentication after running ***Manage Kerberos*** on the Kafka service is done according to Use Kerberos with MS Active Directory in Kafka.

Kerberos with Samba

By Olga Samsonova

Contents

To Table of Contents

Overview
Start Kerberos Samba on an ADS cluster in the ADCM interface
Samba Kerberos parameters

Overview

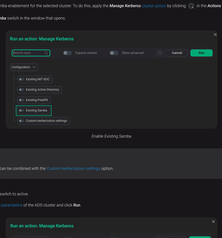
Samba acts as an Active Directory domain controller, so setting up Kerberos authentication with Samba is very similar to setting up authentication with MIT Active Directory.

CentOS 7 specifics

On CentOS 7, the Samba server no longer generates keytab files automatically, since the standard Samba package for CentOS 7 does not support creating principals on a remote controller. Because of this specificity, you need to prepare the keytab files manually before enabling Kerberos Samba in ADS. For this you can change the variables in the script.conf and use it. On other systems, the generation is automatic.

Start Kerberos Samba on an ADS cluster in the ADCM interface

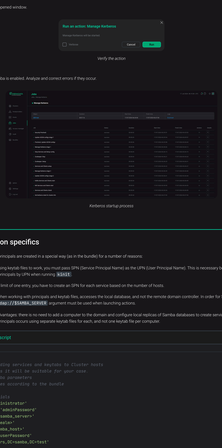
1. Initiate the Kerberos Samba enablement for the selected cluster. To do this, apply the **Manage Kerberos** cluster action by clicking ▷ in the **Actions** column.
2. Turn on the **Existing Samba** switch in the window that opens.



Enable Existing Samba

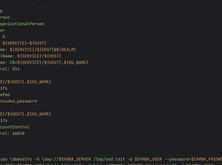
> **NOTE**
>
> Enabling **Existing Samba** can be combined with the Custom kerberization settings option.

3. Set the **Show advanced** switch to active.
4. Set the Samba Kerberos parameters of the ADS cluster and click **Run**.



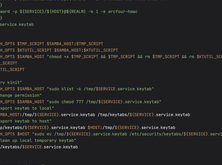
Configuring an ADS cluster for Kerberos Samba

5. Verify the action in the opened window.



Verify the action

6. Wait until Kerberos Samba is enabled. Analyze and correct errors if they occur.



Kerberos startup process

Implementation specifics

In the presented script, principals are created in a special way (as in the bundle) for a number of reasons:

- For automation using keytab files to work, you must clear SPN (Service Principal Name) as the UPN (User Principal Name). This is necessary because Samba does not allow searching for principals by UPN when running `kinit`.
- Since there is a UPN limit of one entry, you have to create an SPN for each service based on the number of hosts.
- By default, Samba, when working with principals and keytab files, accesses the local database, and not the remote domain controller. In order for Samba to access the controller, the `-U <adm>@<KRBREALM.SERVER>` argument must be used when launching actions.

This approach has its advantages: there is no need to edit a computer to the domain and configure local replicas of Samba databases to create service users and principals. Also, authentication of principals occurs using separate keytab files for each, and not one keytab file per computer.

Keytab generation script

```
#!/bin/bash
# Script for adding services and keytabs to Cluster hosts
# Edit: edit.sh if all is suitable for your case.
# Edit your Samba parameters
# Select services according to the bundle

# Samba credentials
SAMBA_USER='Administrator'
SAMBA_PASSWORD='adminPassword'
SAMBA_SERVER='samba.server'
REALM='samba.realm'
SAMBA_HOST='samba.hostname'
USER_PASSWORD='userPassword'
DC_NAME='DC=test,DC=samba,DC=test'

declare -a HOSTS

SSH_OPTS="-o UserKnownHostsFile=/dev/null -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no"

HOSTS=(
    "list_your_hosts here"
)
encoded_password=$(echo -n "\"${USER_PASSWORD}\"" | iconv -f UTF-8 -t UTF-16LE | base64)

#ADD services
SERVICES=(HTTP nifi zookeeper kafka ksql-server kafka-rest schema-registry)

mkdir /tmp/keytabs

for HOST in ${HOSTS[@]}; do
    echo "Add host $HOST"
    ssh $SSH_OPTS $HOST "sudo hostnamectl --static set-hostname $HOST"
    echo "Create keytab dir"
    ssh $SSH_OPTS $HOST "sudo mkdir -p /etc/security/keytabs"
    for SERVICE in ${SERVICES[@]}; do
        echo "Create service user"
        ssh $SSH_OPTS $SAMBA_HOST "sudo useradd $SERVICE"
        echo "Create service principal"
        echo "Create and talif"
        ssh $SSH_OPTS $SAMBA_HOST "sat -vtOP -r /tmp/add.talif
DN: CN=${SERVICE},${HOST},${DN_NAME}
changetype: add
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: user
accountExpires: 0
sAMAccountName: ${SERVICE}-${HOST}
userPrincipalName: ${SERVICE}/${HOST}@${REALM}
servicePrincipalName: ${SERVICE}/${HOST}
distinguishedName: CN=${SERVICE}/${HOST},${DN_NAME}
userAccountControl: 514

DN: CN=${SERVICE}/${HOST},${DN_NAME}
changetype: modify
replace: unicodePwd
unicodePwd:: $encoded_password

DN: CN=${SERVICE}/${HOST},${DN_NAME}
changetype: modify
replace: userAccountControl
userAccountControl: 66048

EOF"
        echo "sudo ldbmodify -H ldap://$SAMBA_SERVER /tmp/add.talif -U $SAMBA_USER --password=$SAMBA_PASSWORD"
        ssh $SSH_OPTS $SAMBA_HOST "sudo ldbmodify -H ldap://$SAMBA_SERVER /tmp/add.talif -U $SAMBA_USER --password=$SAMBA_PASSWORD"

        echo "Delete keytab"
        ssh $SSH_OPTS $SAMBA_HOST "sudo rm -f /tmp/$SERVICE.service.keytab"

        echo "Extract keytab"
        TMP_SCRIPT="/tmp/extract_keytab_${SERVICE}_${HOST}.sh"
        KTUTIL_SCRIPT="/tmp/ktutil_commands_${SERVICE}_${HOST}.txt"
        cat <<EOT > $TMP_SCRIPT
#!/bin/bash
cd /tmp
sudo ktutil < $KTUTIL_SCRIPT
EOT

        cat <<EOT > $KTUTIL_SCRIPT
add_entry -password -p ${SERVICE}/${HOST}@${REALM} -k 1 -e aes256-cts-hmac-sha1-96
${USER_PASSWORD}
add_entry -password -p ${SERVICE}/${HOST}@${REALM} -k 1 -e aes128-cts-hmac-sha1-96
${USER_PASSWORD}
add_entry -password -p ${SERVICE}/${HOST}@${REALM} -k 1 -e arcfour-hmac
${USER_PASSWORD}
wkt ${SERVICE}.service.keytab
quit
EOT

        ssh $SSH_OPTS $HOST "sudo rm -f /tmp/${KTUTIL_SCRIPT##*/}"
        scp $SSH_OPTS $KTUTIL_SCRIPT $SAMBA_HOST:$KTUTIL_SCRIPT
        ssh $SSH_OPTS $SAMBA_HOST "chmod +x $TMP_SCRIPT && $TMP_SCRIPT && rm $TMP_SCRIPT && rm $KTUTIL_SCRIPT"
        rm $TMP_SCRIPT
        rm $KTUTIL_SCRIPT

        echo "Try klist"
        ssh $SSH_OPTS $SAMBA_HOST "sudo klist -k /tmp/$SERVICE.service.keytab"
        echo "Change permission"
        ssh $SSH_OPTS $SAMBA_HOST "sudo chmod 777 /tmp/$SERVICE.service.keytab"
        echo "Export keytab to local"
        scp $SSH_OPTS $SAMBA_HOST:/tmp/$SERVICE.service.keytab /tmp/keytabs/$SERVICE.service.keytab
        echo "Export keytab to host"
        scp /tmp/keytabs/$SERVICE.service.keytab $HOST:/tmp/$SERVICE.service.keytab
        ssh $SSH_OPTS $HOST "sudo mv /tmp/$SERVICE.service.keytab /etc/security/keytabs/$SERVICE.service.keytab"
        echo "Change on local temporary keytab"
        rm /tmp/keytabs/$SERVICE.service.keytab
    done

    echo "Change owner and group"

    ssh $SSH_OPTS $HOST sudo chown nifi:nifi /etc/security/keytabs/nifi*
    ssh $SSH_OPTS $HOST sudo chown kafka:kafka /etc/security/keytabs/kafka*
    ssh $SSH_OPTS $HOST sudo chown kafka:kafka /etc/security/keytabs/ksql*
    ssh $SSH_OPTS $HOST sudo chown ksql:kafka /etc/security/keytabs/ksql-server*
    ssh $SSH_OPTS $HOST "sudo chown kafka-rest:kafka /etc/security/keytabs/kafka-rest*
    ssh $SSH_OPTS $HOST sudo chown schema-registry:kafka /etc/security/keytabs/schema-registry*

    echo "Change permissions"

    ssh $SSH_OPTS $HOST "sudo chmod -R 777 /etc/security/keytabs/*

done
```

To enable Kerberos Samba on hosts running CentOS 7, where default weak encryption algorithms are used and the Kerberos environment has not been previously configured, enabling authentication should be performed in several steps:

1. Before running the script, make sure that the Domain Name Service (DNS) of the local computer (host) from which the script is launched, as well as the ADS cluster hosts, is configured to connect via SRV to the Samba server.
2. The first time you enable **Manage Kerberos**, disable the **Custom kerberization settings** option, which sets the value `True` only for the **Set up Kerberos utils** and **Configure Kerberos on hosts** parameters. This will install all utilities and create all necessary configuration files.
3. On the Primary configuration tab of the cluster in the **Kerberos** parameter group, set the **Custom krb5.conf** parameters `True`. Next, open the custom krb5.conf template (located under **Custom krb5.conf**) and edit the `[libdefaults]` section as in the krb5.conf example below and save the configuration.
4. The second time you enable **Manage Kerberos**, set the **Custom kerberization settings** option to `True` for all parameters except the **Set up principals and keytabs** parameter.

Example of custom krb5.conf

```
[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 = /etc/pki/tls/certs/ca-bundle.crt
 default_realm = SAMBA.TEST
 permitted_enctypes = aes256-cts-hmac-sha1-96 aes128-cts-hmac-sha1-96 arcfour-hmac rc4-hmac arcfour-hmac-md5
 default_tkt_enctypes = aes256-cts-hmac-sha1-96 aes128-cts-hmac-sha1-96 rc4-hmac
 default_tgs_enctypes = aes256-cts-hmac-sha1-96 aes128-cts-hmac-sha1-96 rc4-hmac
 allow_weak_crypto = true

[realms]
 SAMBA.TEST = {
  admin_server = dc1.samba.test
  kdc = dc1.samba.test
 }
```

Samba Kerberos parameters

Parameter	Description
Authentication on WEB UIs	Enables Kerberos authentication on Web UIs
KDC hosts	One or more domain controller hosts
Realm	Kerberos realm
Domains	Domains associated with hosts
Kadmin server	Host where `kadmin` is running
Kadmin principal	Principal name used to connect via `kadmin`, for example `admin@MY-CENTRAL1.INTERNAL`
Kadmin password	A principal password used to connect via `kadmin`
Keytabs directory	Directory of the keytab file that contains one or several principals along with their keys
Additional realms	Additional Kerberos realms
LDAP URL	LDAP URL consists of `ldap://` or `ldaps://`, hostname or IP address, and port of the LDAP server
Container DN	Container distinguished name
Trusted Active Directory server	Trusted DC server
Trusted Active Directory realm	Realm for cross-realm trust

> **NOTE**
>
> Samba-based Kerberos authentication after running **Manage Kerberos** in the Kafka service is done according to Use Kerberos with MS Active Directory in Kafka.

Custom kerberization settings

Olga Semme

Contents

To Table of Contents

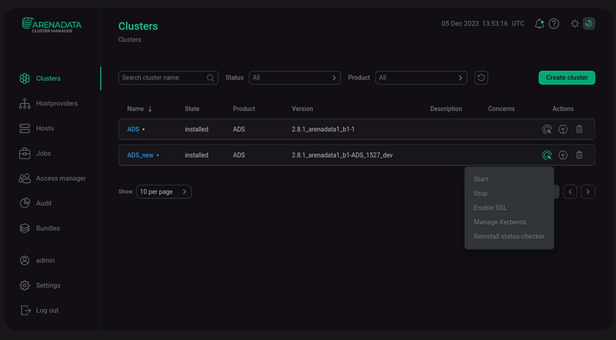
Custom keytab recommendations

The ***Custom kerberization settings*** option allows the user to select kerberization steps, such as creating principals and keytab files.

Each protocol available when selecting the ***Manage Kerberos*** cluster action (MIT Kerberos, MS Active Directory, FreeIPA) can be combined with the ***Custom kerberization settings*** option .

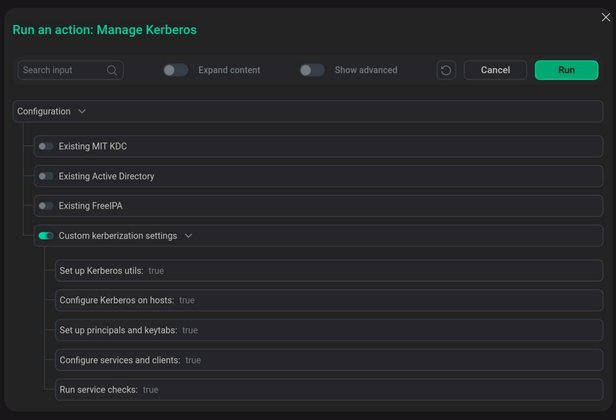
To set the option, follow the steps:

1. Initiate Kerberos enablement for the selected cluster. To do this, apply the ***Manage Kerberos*** action by clicking on the in the ***Actions*** column.



Enabling Kerberos on an ADS cluster

2. Turn on the ***Custom kerberization settings*** switch in the window that opens, expand the settings tree, set the necessary parameters, and click ***Run***.



Custom kerberization settings parameters

Custom kerberization settings parameter description

Parameter	Description	Default value
Set up Kerberos utils	Enables installation or removal of Kerberos clients and utils. Affects the ***Expand*** and ***Install*** actions	True
Configure Kerberos on hosts	Enables cluster configuration, including *krb5.conf*, *ldap.conf*	True
Set up principals and keytabs	Enables creation, recreation, or removal of principals and keytabs. Passwords for principals are generated randomly before keytab creation. Affects the ***Expand*** and ***Install*** actions. ADCM bundle will set up owner and permissions for keytabs only if this checkbox is selected in the cluster configuration. In case of absence of admin permissions, a customer should provide the prepared keytabs with correctly set owner and permissions (see Custom keytab recommendations)	True
Configure services and clients	This parameter does not affect ADS operation. Custom settings are made using JAAS file template	True
Run service checks	Enables service check runs	True

Custom keytab recommendations

Below is the table with recommendations for owners, groups, and permissions for keytabs.

Keytab recommendations

Component short name	Keytab owner	Keytab group	Permissions
kafka	kafka	kafka	600
kafka-rest	kafka-rest	kafka	600
ksql-server	ksql	kafka	600
nifi	nifi	nifi	600
nifi-registry	nifi	nifi	600
schema-registry	schema-registry	kafka	600
kafka-connect	kafka-connect	kafka	600
zookeeper	zookeeper	zookeeper	600
HTTP	nifi	nifi	640

SASL PLAINTEXT

Olga Semme

To Table of Contents

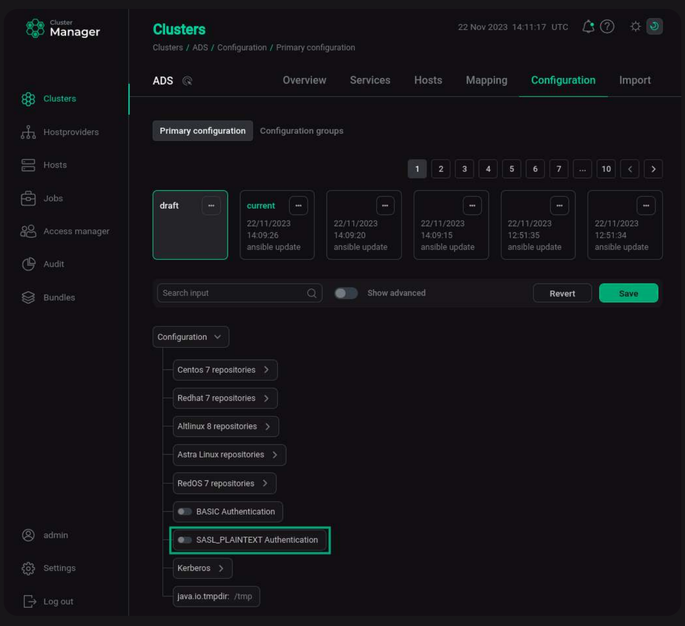
SASL — authentication and data security framework used in Internet protocols.

PLAIN — simple mechanism for passing passwords in a clear text.

PLAINTEXT — authenticator that is configured to support the PLAIN authentication mechanism.

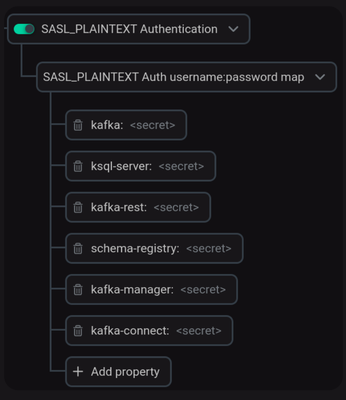
To enable authentication SASL PLAINTEXT with ADCM, do the following:

1. On the cluster configuration tab in the configuration settings tree, enable the ***SASL_PLAINTEXT Authentication*** switch and expand the ***SASL_PLAINTEXT Auth username:password map*** node.



Enable SASL_PLAINTEXT Authentication on the ADS cluster

2. The nodes that open contain the default encrypted user passwords for each service designed to work with Kafka.



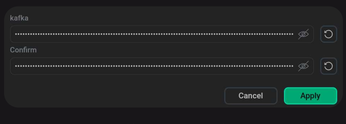
Configuration nodes with users and passwords SASL_PLAINTEXT Authentication

When ***SASL_PLAINTEXT Authentication*** is enabled for the first time, passwords for users of each service are automatically generated:

```
kafka: kafka-password
ksql-server: ksql-server-password
kafka-rest: kafka-rest-password
schema-registry: schema-registry-password
kafka-connect: kafka-connect-password
```

It is recommended to change the default passwords.

3. To change the password, click on ***<secret>*** next to the username, enter the new password, confirm the password and click ***Apply***.



Change Password

4. Save the configuration by clicking ***Save***.

5. Restart all services for which authentication has been configured. To do this, apply the action Restart by clicking on the in the ***Actions*** column.

NOTE

User experience in the Kafka service after enabling SASL PLAINTEXT authentication is described in the article Use SASL PLAINTEXT in Kafka.

SSL channel security

© Olga Baranova

Contents

To Table of Contents

Prerequisites for enabling SSL
Create an SSL certificate for Certification Authority
- Certificate generation sequence
- Create an SSL certificate for Certification Authority using a script

Start SSL on an ADS cluster using ADCM
Disable SSL on an ADS cluster using ADCM

Prerequisites for enabling SSL

- Every cluster host must have the keystore.jks key and certificate store (unique for each cluster host), know its path and password.
- Every cluster host must have the truststore.jks key and certificate store (common for all cluster hosts), the path to it and its password must be known.
- The list of DNs of all cluster hosts (the `CN=ads-host1.example.com, OU=AD, O=AD, L=MSK, ST=MS, C=RU` generated when creating certificates SSL must be known.
- The name of the user who has an entry in the ADS Access Directory that will act as the NiFi administrator and his password must be known.

Create an SSL certificate for Certification Authority

Certificate generation sequence

1. Generate an RSA key pair for each host.
2. Create a keystore.jks keystore on each host.
3. Create an SSL certificate for each host.
4. Collect all certificates in one place.
5. Copy all certificates for each host to the keystore.jks store.
6. Sign the certificates with a key that belongs to one of the trusted root certificates included in the Java certificate store.
7. Create a truststore.jks store containing the required certificates.
8. Import truststore.jks into Java CA store.
9. Create and import an OpenSSL certificate for Nginx.

CAUTION

To correctly enable authentication in the NiFi service when creating certificates, specify and fill in the `-ext` (subjectAltNames) field using the `-ext` option (for example, for self-signed certificates created below, the filled field looks like this: `-ext san=dns:$HOST`).

Create an SSL certificate for Certification Authority using a script

Use the special script generate.sh, when generating certificates for each host and importing certificates into the Java keystore, you can use the special script generate.sh. Using this script, it is possible to create new certificates and overwrite existing ones.

It is launched on a host that has access rights to cluster hosts via FQDN and internal IP.

Below there is the sequence of creating the certificate:

1. Enter the command to create the script file:

```
$ sudo vim /tmp/ssl/generate.sh
```

2. Replace for the following script lines (only for them!) the following values with values that apply to your case:

- `NUMHOSTS=` – in this line specify the values (letters or numbers), common for all cluster hosts. The values `[1:2:3]` are given as an example for a list of hosts:
  - `ssa-ads-test-1.ru-central1.internal`,
  - `ssa-ads-test-2.ru-central1.internal`,
  - `ssa-ads-test-3.ru-central1.internal`.
- `HOSTS=` – specify the common part of the FQDN of the host in this line, replacing `$i` with the value (letters or numbers) specified in the `NUMHOSTS` parameter. The value `ssa-ads-test-$i.ru-central1.internal` is given as an example.
- `bigdata` – password for stores, recommended to change.

3. Fill in generate.sh with the script text given below, taking into account individual parameters.

generate.sh

```
#!/bin/bash

# Script that is used to generate and import self-signed certificates to Java keystore
# FQDN NUMHOSTS and HOSTS so it will be suitable for your case.

declare -a NUMHOSTS
declare -a HOSTS

SSH_OPTS="-o UserKnownHostsFile=/dev/null -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no"
NUMHOSTS=(1 2 3)
HOSTS=($(for i in "${NUMHOSTS[@]}"; do echo "ssa-ads-test-$i.ru-central1.internal"; done))


for HOST in ${HOSTS}; do
        echo "update-ca-trust"
        sudo $SSH_OPTS $HOST "sudo update-ca-trust extract"
done

echo Generate keystore.jks on each host
for HOST in ${HOSTS}; do
        echo "remove old alias"
        sudo $SSH_OPTS $HOST "keytool -delete -alias $HOST -keystore /tmp/keystore.jks -storepass bigdata";

        echo "Generating keypair"
        sudo $SSH_OPTS $HOST "keytool                   -genkey \
                                                        -noprompt \
                                                        -keyalg RSA \
                                                        -ext san=dns:$HOST \
                                                        -alias $HOST \
                                                        -dname \"CN=$HOST, OU=AD, O=AD, L=MSK, S=MS, C=RU\" \
                                                        -keystore /tmp/keystore.jks \
                                                        -storepass bigdata \
                                                        -keypass bigdata \
                                                        -validity 360 \
                                                        -keysize 2048";

done

echo
echo Export certificates
for HOST in ${HOSTS}; do
        sudo $SSH_OPTS $HOST "keytool                   -export \
                                                        -file /tmp/$HOST.crt \
                                                        -keystore /tmp/keystore.jks \
                                                        -storepass bigdata \
                                                        -alias $HOST \
                                                        -rfc";

done

echo
echo Collect all certificates
for HOST in ${HOSTS}; do
        sudo $SSH_OPTS $HOST:/tmp/$HOST.crt /tmp/

#       scp $SSH_OPTS $HOST:/tmp/x.crt /tmp/
done

echo
echo Transfer certificates on hosts
for HOST in ${HOSTS}; do
        for CERT in ${HOSTS}; do
                scp $SSH_OPTS /tmp/$CERT.crt $HOST:/tmp/
        done

#       scp $SSH_OPTS /tmp/x.crt $HOST:/tmp/
done

echo
echo Import certificates on each host
for HOST in ${HOSTS}; do

        sudo $SSH_OPTS $HOST "for CERT in $(echo ${HOSTS[*]}); do

                echo \"remove old alias before import\"
                keytool -delete -noprompt -alias \$CERT -keystore /tmp/truststore.jks -storepass bigdata

                echo \"Import Cert\"
                keytool                 -import \
                                        -noprompt \
                                        -alias \$CERT \
                                        -file /tmp/\$CERT.crt \
                                        -keystore /tmp/truststore.jks \
                                        -storepass bigdata;
        sudo bash -c \"cat /tmp/\$CERT.crt >> /etc/pki/tls/certs/ca-bundle.crt\";
        done";

done

echo
echo Import truststore to Java CA store
for HOST in ${HOSTS}; do
        sudo $SSH_OPTS $HOST "sudo keytool              -importkeystore \
                                                        -noprompt \
                                                        -srckeystore /tmp/truststore.jks \
                                                        -destkeystore /etc/pki/java/cacerts \
                                                        -deststorepass changeit \
                                                        -srcstorepass bigdata"


done

echo
echo Create directories for certs
for HOST in ${HOSTS}; do
        sudo $SSH_OPTS $HOST "[ -d /etc/ssl ] || sudo mkdir /etc/ssl;
                              [ -d /etc/ssl/certs ] || sudo mkdir /etc/ssl/certs"

done

echo
echo Create and import cert for Nginx
for HOST in ${HOSTS}; do
        sudo $SSH_OPTS $HOST "sudo openssl req -new -newkey rsa:4096 -days 365 -nodes -x509 -subj
\"/C=RU/ST=Russia/L=MSK/O=ADS/CN=$HOST\" -keyout /etc/ssl/host_cert.key -out /etc/ssl/certs/host_cert.cert"
        sudo $SSH_OPTS $HOST "sudo bash -c \"cat /etc/ssl/certs/host_cert.cert >> /etc/pki/tls/certs/ca-bundle.crt\""
done
```

4. Allow script execution:

```
$ sudo chmod +x /tmp/ssl/generate.sh
```

5. Run the script:

```
$ /tmp/ssl/generate.sh
```

6. Monitor script execution.
7. Check each host for storages and files:

- /etc/ssl/host_cert.key
- /tmp/truststore.jks
- /tmp/keystore.jks

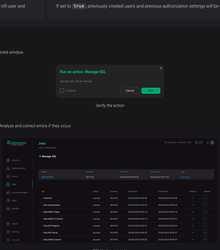
NOTE

The NiFi service requires additional configurations in accordance with the article NiFi service authentication.

Start SSL on an ADS cluster using ADCM

To enable SSL, do the following:

1. Apply the cluster action **Manage SSL** by clicking ⚙ in the **Actions** column of the ADS cluster.
2. In the window that opens, activate the **Enable SSL** switch, expand the configuration tree, and enter the required parameter values.



Enabling SSL on the ADS cluster

Parameters for enabling SSL

Parameter	Description
Keystore path	Path to the keystore.jks storage on hosts
Keystore password	Password for the keystore.jks store, default is `bigdata`
Truststore path	Path to the truststore.jks storage on hosts
Truststore password	Password for the truststore.jks store, default is `bigdata`
Force rewrite services SSL configuration	If set to `true`, the configuration for all services will be overwritten. Use if you entered the wrong password or must store path when you first enabled SSL.
DANGEROUS! Clean nifi user and authorizers	If set to `true`, previously created users and previous authorization settings will be deleted

3. Click **Run**.
4. Verify the action in the opened window.

Verify the action

5. Wait until SSL is enabled. Analyze and correct errors if they occur.

SSL startup process

NOTE

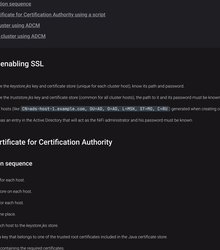
Work in services after enabling SSL is carried out in accordance with the articles:

- SSL channel security in Kafka
- NiFi service authentication

Disable SSL on an ADS cluster using ADCM

To disable SSL, do the following:

1. Apply the cluster action **Manage SSL** by clicking ⚙ in the **Actions** column of the ADS cluster.
2. In the window that opens, deactivate the **Enable SSL** switch.



Disabling SSL on an ADS cluster

3. Click **Run**.
4. Verify the action in the opened window.

Verify the action

5. Wait for the disable SSL to complete. Analyze and correct errors if they occur.

Arenadata Platform Security overview

Olga Semme

To Table of Contents

Arenadata Platform Security (ADPS) is a security framework that is based on Apache Ranger and Apache Knox. You can use ADPS to configure authorization and policy-based access control for ADS services.

Before ADPS installation, ensure the following requirements are met:

- Arenadata Streaming cluster of version 1.6.2 or higher is installed and configured according to the Get started with Arenadata Streaming guide.
- All ADPS prerequisites are met:
  - Hardware requirements
  - Network requirements
  - Software requirements
- ADPS services should not use ADS cluster hosts. It is recommended to install ADPS on a separate virtual machine or a dedicated server.
- If an external DBMS is used to store metadata (PostgreSQL/MySQL), it should be started and available. The Ranger installer creates two new users (default names: `rangeradmin` and `rangerlogger`) and two new databases (default names: `ranger` and `ranger_audit`).
- One of the following authentication types should be configured:
  - SASL PLAINTEXT
  - MIT Kerberos
  - MS Active Directory

NOTE

For more information on ADPS installation, see Get started with Arenadata Platform Security.

After installing ADPS and integration with ADS, you can start working with Ranger plugins available for ADS services.

Activation of the Ranger plugin and use of Ranger in ADS services are described in the articles:

- for the Kafka service:
  - Ranger Kafka plugin
  - Use Ranger in Kafka
- for NiFi service:
  - Ranger NiFi plugin
  - Use Ranger in NiFi

Integration with ADS cluster

Olga Semme

Contents

To Table of Contents

Integrate ADPS cluster with ADS cluster

Access to the Ranger web interface

After verifying the user's authentication, it is necessary to determine the access rights. User access rights to resources are determined by authorization.

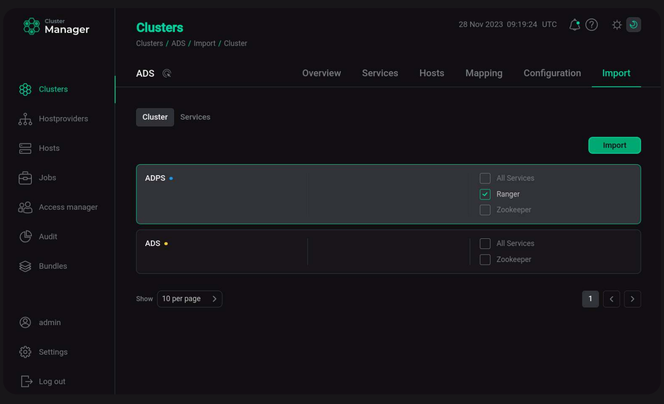
Ranger features:

- Create services for the Kafka ADS service and add a user access policy to the services.
- Create services based on tags — allows you to control access to resources of multiple components without creating separate services and policies in each component.
- Use Ranger Tab Sync to synchronize the Ranger tag store with an external metadata service such as Apache Atlas.

Integrate ADPS cluster with ADS cluster

In order to integrate the ADPS cluster with the ADS cluster, perform the following steps:

1. Go to the ***Import*** tab of the ADS cluster.
2. Select the ***Ranger*** service of the previously created ADPS cluster.
3. Click ***Import***.

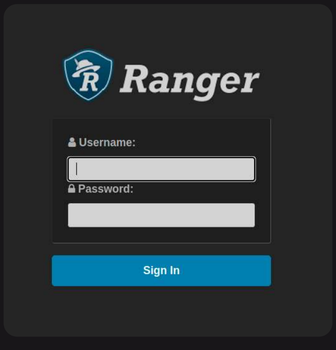


Enable Import Ranger ADPS

Access to the Ranger web interface

In order to gain access to the Ranger web interface, perform the following steps:

1. Enter in the browser the designation of the host on which ADPS is installed; as the port number, use the value specified in the ***ranger.service.http.port*** parameter (default `6080`) of the Ranger service.
2. Enter your administrator credentials and click ***Sign in*** to log in as a Ranger administrator.

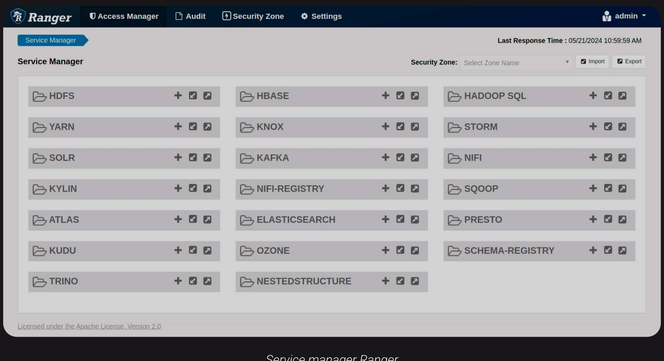


Log in to Ranger

NOTE

Passwords for accessing the Ranger web interface are set before installing the ADPS cluster services in the ***Credentials*** Ranger service configuration section.

As a result, the ***Service manager*** interface window opens.



Service manager Ranger

ADS monitoring metrics

By Peter

Contents

Metrics scopes

View monitoring metrics

Graphite

Grafana

This article describes the metrics for monitoring an ADS cluster. For information on installing monitoring, refer to the articles:

- Initial installation
- Offline installation

Metric types

Two groups of metrics are available for an ADS cluster: system metrics and service metrics.

All [logpacks](#) system metrics are shown in the **System_metrics** section, and service metrics in the **services** section.

System metrics show the general characteristics of the servers available to hosts or clusters. System metrics are shown in the table below.

System metrics

Metrics group	Description
cpu	CPU utilization
diskspace	Disk metrics
files	File statistics
iostat	Input/output operation performance
loadavg	System load averages
memory	Memory usage
netstat	Network connection statistics
network	Network interface performance
uptime	Uptime statistics

Service metrics show the characteristics of the services available to hosts or an ADS cluster.

The metrics for each service are shown in the tables below.

Kafka

Metrics group	Metrics name	Description
ReplicaManager	AtMinIsrPartitionCount	Number of partitions corresponding to minimal ISR (Shrink, Fast Copy)
	FailedIsrUpdatesPerSec	Number of failed In-Sync Replicas (ISR) updates per second This metric has the following attributes: • count • fifteen-minute rate • five-minute rate • mean rate • one-minute rate
	IsrExpandsPerSec	In-Sync Replicas (ISR) expand rate This metric has the following attributes: • count • fifteen-minute rate • five-minute rate • mean rate • one-minute rate
	IsrShrinksPerSec	In-Sync Replicas (ISR) shrink rate This metric has the following attributes: • count • fifteen-minute rate • five-minute rate • mean rate • one-minute rate
	LeaderCount	Number of partition leaders
	OfflineLogDirCount	Number of offline replicas
	PartitionCount	Number of partitions
	ReassignedPartitions	Number of reassigned partitions
	UnderMinIsrPartitionCount	Number of partitions where number of In-Sync Replicas (ISR) is less than minisr
	UnderReplicatedPartitions	Number of partitions replicated partitions (ISR < current replicas)
Server Group metrics have the following attributes: • type • name	BytesInPerSec	Number of new bytes per second
	BytesOutPerSec	Number of bytes sent per second
	BytesRejectedPerSec	Rejected data transfer rate (bytes per second)
	FailedFetchRequestsPerSec	Number of get requests that failed per second
	FailedProduceRequestsPerSec	Number of requests per second of the message producer that failed
	FetchMessageConversionsPerSec	Number of fetch message conversions per second
	FetchMessageTotalTimeMsPerSec	Execution second when checking a message that is read (Fetch message)
	InvalidMessageCrcRecordsPerSec	Number of errors while calculating a message's CRC in the received offset or damaged data
	MessagesInPerSec	Total number of incoming messages
	NoKeyCompactedTopicRecordsPerSec	Number of messages validation errors due to no key specified for a compressed topic per second
	ProduceMessageConversionsPerSec	Number of message conversions per second for the broker
	ReassignmentBytesInPerSec	Number of bytes of incoming redirect traffic per second
	ReassignmentBytesOutPerSec	Number of bytes of outgoing redirect traffic per second
	ReplicationBytesInPerSec	Number of bytes of data transfer from other brokers per second
	ReplicationBytesOutPerSec	Number of bytes transferred for other brokers per second
	TotalFetchRequestsPerSec	Number of get requests per second
	TotalProduceRequestsPerSec	Number of requests to produce messages per second
controller	ActiveControllerCount	Number of active controllers
	ControllerStats	Controller status
	GlobalPartitionCount	Total number of partitions
	GlobalTopicCount	Total number of topics
	OfflinePartitionsCount	Number of partitions that do not have an active leader and are therefore not available or accessible
	PreferredReplicaImbalanceCount	Number of imbalances in the preferred replica
	ReplicasIneligibleToElectCount	Number of non-compliant replicas waiting to be deleted
	ReplicasToDeleteCount	Number of replicas waiting to be deleted
	TopicsIneligibleToDeleteCount	Number of non-compliant topics waiting to be deleted
	TopicsToDeleteCount	Number of topics waiting to be deleted
controllerStats Group metrics have the following attributes: • name • type	AutoLeaderBalanceRateAndTimeMs	Frequency and duration of automatic leader rebalancing
	ControlledShutdownRateAndTimeMs	Frequency and duration of controlled stops
	ControllerChangeRateAndTimeMs	Frequency and duration of controller changes
	ControllerShutdownRateAndTimeMs	Frequency and duration of controller key-key-controller stops
	IsrChangeRateAndTimeMs	Websites frequency and duration of controller delay for compromised cluster
	LeaderAndReplicaRequestRateAndTimeMs	Frequency and duration of receiving a leader response and a request to the topic Replica. DNS
	LeaderElectionRateAndTimeMs	Frequency and duration of leader election
	LogDirChangeRateAndTimeMs	Frequency and duration (CPU latency) of Logfic changes
	ManualLeaderBalanceRateAndTimeMs	Frequency and duration (CPU latency) of manual leadership balancing
	PartitionReassignmentRateAndTimeMs	Frequency and duration (CPU latency) of partition reshaping
	TopicChangeRateAndTimeMs	Frequency and duration (CPU latency) of creating new topics
	TopicDeletionRateAndTimeMs	Frequency and duration (CPU latency) of deleting topics
	UncleanLeaderElectionsPerSec UncleanLeaderElectionRateAndTimeMs	Number of unclean leader elections per second Frequency and duration (CPU latency) of unclean leader elections
	UncleanLeaderElectionsPerSec	Number of unclean leader elections per second
	UpdateParentZkNodeRateAndTimeMs	Frequency and duration (CPU latency) of Nature updates
memory	HeapMemoryCommitted	Usage of dynamic (heap) memory — committed (in MB)
	HeapMemoryUsage_init	Usage of dynamic (heap) memory — initialization (in MB)
	HeapMemoryUsage_max	Usage of dynamic (heap) memory — maximum (in MB)
	HeapMemoryUsage_used	Used dynamic (heap) memory (in MB)
	NonHeapMemoryUsage_committed	Use of non-heap memory — committed (in MB)
	NonHeapMemoryUsage_init	Non-heap memory usage — initialization (in MB)
	NonHeapMemoryUsage_max	Use of non-heap memory — maximum (in MB)
	NonHeapMemoryUsage_used	Used out-dynamic (heap) memory (in MB)
	OpenFileDescriptorCount	Number of objects in the Zookeeper's usability
network	NetworkProcessorAvgIdlePercent	The average percentage of time that network processor threads are idle. Values range from 0% (all resources in use) to 1% (all resources available)
	RequestsPerSec	Requests per second
	RequestQueueSize	Response queue size
session Group metrics have the following attributes: • name • fifteen-minute rate • five-minute rate • mean rate • one-minute rate	ZookeeperAuthFailuresPerSec	Number of unsuccessful (due to authorization errors) attempts to connect to the cluster per second
	ZookeeperDisconnectsPerSec	Number of Zookeeper client disconnects from the cluster per second. This metric will be green (zero) when Kafka does not see if a computer is running
	ZookeeperExpiresPerSec	Number of Zookeeper client disconnects from the cluster per second. This metric will be green (zero) when Kafka does not see if a computer is running
	ZookeeperReadOnlyConnectsPerSec	Number of successful Zookeeper client connections to the cluster per second
	ZookeeperSaslAuthenticationsPerSec	Number of client authentication successes per second
	ZookeeperSyncConnectsPerSec	Number of successful Zookeeper client connections to the cluster per second

etcd3

Metrics group	Metrics name	Description
MinIOHighExcellence.Instance	activeThreads	Number of active threads
	fsfsyncDur	Number of sparks read
	fsWalSync	Number of sparks read
	queueEventsSize	Queue content size
	queueEventTurn	Number of bytes in the queue
MinIOHighExcellence.SystemInspectorio	availableProcessors	Available processors
	diskStatistics	Disk usage (in %)
	heapUtilization	Heap usage (in %)
	loadAverage	Average load (in %)
	osPhysi	Memory heap used (in MB)
	osSystemCpuLoad	Disk space usage (in MB)

ssh

Metrics group	Metrics name	Description
heap	committed	Usage of heap memory — committed (in MB)
	init	Usage of heap memory — initialization (in MB)
	max	Usage of heap memory — maximum (in MB)
	usage	Usage of heap memory (in MB)
	used	Used heap memory (in MB)
non-heap	committed	Use of non-heap memory — committed (in MB)
	init	Non-heap memory usage — initialization (in MB)
	max	Use of non-heap memory — maximum (in MB)
	usage	Use of non-heap memory (in MB)
	used	Used non-heap memory (in MB)
pools -> Code Cache	committed	Code Cache memory area — committed (in MB)
	init	Code Cache memory area — initialization (in MB)
	max	Code Cache memory area — maximum (in MB)
	usage	Usage of the Code Cache memory area (in MB)
	used	Used Code Cache memory area (in MB)
pools -> Compressed Class Space	committed	Compressed Class Space — committed (in MB)
	init	Memory area of Compressed Class Space — initialization (in MB)
	max	Compressed Class Space — maximum (in MB)
	usage	Use area of memory Compressed Class Space (in MB)
	used	Used memory area Compressed Class Space (in MB)
pools -> G1 Eden Space	committed	Memory area of memory G1 Eden Space — committed (in MB)
	init	Memory area of memory G1 Eden Space — initialization (in MB)
	max	Memory area of memory G1 Eden Space — maximum (in MB)
	usage	Use area of memory G1 Eden Space (in MB)
	used	Used area of memory G1 Eden Space (in MB)
	usedAfterGc	Used after GC start area of memory G1 Eden Space (in MB)
pools -> G1 Old Gen	committed	Memory area G1 Old Gen — assignment (committed) (in MB)
	init	Memory area G1 Old Gen — initialization (in MB)
	max	G1 Old Gen memory area — maximum (in MB)
	usage	Usage of the G1 Old Gen memory area (in MB)
	used	Used area of memory G1 Old Gen (in MB)
	usedAfterGc	Used after GC start area of memory G1 Old Gen (in MB)
pools -> G1 Survivor Space	committed	Memory area G1 Survivor Space — assignment (committed) (in MB)
	init	Memory area G1 Survivor Space — initialization (in MB)
	max	G1 Survivor Space memory area — maximum (in MB)
	usage	Usage of the memory area G1 Survivor Space (in MB)
	used	Used area of memory G1 Survivor Space (in MB)
	usedAfterGc	Used after GC start Used after gc area of memory G1 Survivor Space (in MB)
zeek -> Metaspace	committed	Memory area Metaspace — committed (in MB)
	init	Memory area Metaspace — initialization (in MB)
	max	Memory area Metaspace — maximum (in MB)
	usage	Usage of the Metaspace memory area (in MB)
	used	Used Metaspace memory area (in MB)
total	committed	Assigned to use total memory (in MB)
	init	Initialization of total memory (in MB)
	max	Maximum total memory (in MB)
	usage	Usage of total memory (in MB)
	used	Used total memory (in MB)
-	activeThreads	Number of active threads
-	daemonThreads	Number of threads in the queue
-	fsSyncReadLastMinCalls	Number of sparks read in the last 5 minutes
-	fsSyncReadLastMinLatency	Number of sparks open in the last 5 minutes
-	fsSyncWriteLastMinLa tency	Number of sparks written in the last 5 minutes
-	FloatFileSizeUsed	Number of FloatTabs in the queue
-	FloatTransferLastMi nuteCalls	Number of FloatTabs received in the last 5 minutes
-	FloatTransferLastMinLa tency	Number of FloatTabs sent in the last 5 minutes
-	TotalTransferLastMinNumB ytes	Total byte duration (in nanoseconds)

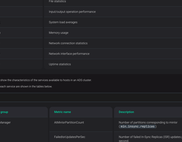
View monitoring metrics

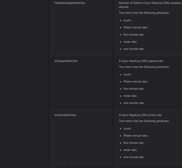
You can use the web interfaces [Graphite](#) and [Grafana](#) to view monitoring metrics.

Graphite

Graphite stores numeric time series data and displays graphs of the data on demand. To view the metrics, enter the address of the host with the monitoring cluster into the browser. For example, `http://10.0.0.1:8080`. The Graphite web interface will open in the browser. To view the metrics, click **Graphite** in the upper left corner. The **Graphite Web Interface** TM will open in configuring Graphite, see the screenshot described by a color. Example, `http://10.0.0.1:8080`.

In the left part of the window that opens, expand the **Metrics** → **Annotates** → **ADS** node. The list displays two groups of metrics: **System_metrics** and **services**.

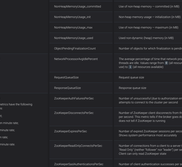




Monitoring metrics in Graphite

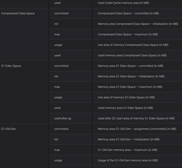
Grafana

Grafana allows you to query, data and visualize metrics stored in Graphite. To open the Grafana web interface, enter the host address of the monitoring cluster into the browser bar. If you don't specify a port number, Grafana opens on port 3000. For example, `http://10.0.0.1:3000`. The Grafana login page will open. Enter the login and password. The default login and password are: admin / admin. Click **Log in**. You can change the password later.



Grafana web interface

A page Zoom will open.



Metrics in the Grafana visualization interface

Kafka clients

Olga Semmе

To Table of Contents

Clients for working with Apache Kafka are available in many programming languages.

ADS provides the ability to install client libraries for various languages that provide both low-level access to Kafka and higher-level stream processing.

Kafka clients allow you to create distributed applications and microservices that read, write, and process event streams in parallel, at scale, and in a fault-tolerant manner, even in the event of network problems or machine failures.

Kafka comes with some of these clients, which are complemented by dozens of clients provided by the Kafka community.

NOTE

Information about Kafka clients is provided as a reference. Featured clients are independent open source projects.

Below is a list of currently known and supported Kafka clients.

Implementation language	Client	Description
C/C++	librdkafka	Robust, high-performance C/C++ library with full protocol support, built on top of many other language bindings, including Haskell, Node.js, OCaml, PHP, Python, Ruby, C#/.NET
	libkafka	Native C++ library with protocol support for metadata, production, sampling, and offset
	libkafka-asio	Kafka client library for C++ headers only using Boost Asio
	csi-kafka	C++ 11 asynchronous producer/consumer library for Apache Kafka based on Boost Asio
	libasynckafkaclient	Single-threaded asynchronous library based on C++
	kafka-cpp	C++ client for Apache Kafka. A library that allows you to create messages for a distributed Kafka publish/subscribe messaging service
Python	confluent-kafka-python Docs	High performance Python client based on librdkafka with full protocol support
	kafka-python	Pure Python implementation with full protocol support; consumer and producer implementations included; GZIP, LZ4 and Snappy compression supported
	pykafka	Python driver with full protocol support, Pythonic API, and managed balanced consumer implementation
	pykafka	Allows you to create messages for the distributed Kafka publish/subscribe messaging service
	pykafka	A programmer-friendly Kafka client for Python. It includes implementations of Kafka producers and consumers in Python, which are additionally supported by an extension built on top of librdkafka
	pykafkap	Urban Airship Python client –simple Kafka producer client for Python
	brod	Allows you to create messages for the Kafka publish/subscribe distributed messaging service (producer, simple consumer, ZK-consumer)
Go (AKA golang)	sarama	Kafka client library Go licensed by MIT; clean implementation with full protocol supported; consumer and producer implementations included; GZIP and Snappy compression supported
	confluent-kafka-go Docs	Kafka Confluent client for Golang includes the librdkafka library, providing full Kafka protocol support with high performance and reliability
	franz-go Docs	Pure Go implementation provides full protocol support, high performance, and support for all client-facing KIPs
Erlang	brod	Erlang Kafka client library. Full consumer protocol support, very efficient manufacturer implementation
	ekaf	Minimalist high-performance Kafka client in Erlang
	erlkafka kafka-erlang	Kafka client written in Erlang
PowerShell	KNet Docs	PowerShell client with producer, consumer, admin support. Additionally, you can run all available tools from PowerShell. For a complete list of available commands for using KNet PowerShell, see KNet: PowerShell Module
.NET	KNet	KNet is a comprehensive .NET package for Apache Kafka providing all the features: producer, consumer, admin, streams, connect, backends (ZooKeeper and Kafka)
	confluent-kafka-dotnet	Full featured .NET client for Apache Kafka based on librdkafka
	kafka-net	Pure C# client with full protocol support. Includes consumer, producer, lower-level components, and gzip support (no fastload)
	kafka-net	This .NET client implementation for Kafka using C# covers most of the core functionality, including a simple producer and consumer
	CSharpClient-for-Kafka	.Net implementation of the Apache Kafka protocol that provides basic functionality through the producer/consumer classes. The project also offers a balanced consumer implementation. The project is a fork of the ExactTarget Kafka-net client
	kafka4net	C# client, asynchronous, compression supported, transparently tracking changes to the leading partition
	kafka-sharp	High performance .NET Kafka driver
	kafunk	Full featured native F# client running on .NET Core
Clojure	kafka-fast	Fast Kafka API for JVM languages implemented in Clojure
	clj-kafka	Wrapper for Java API to interact with Kafka
	kinsky Docs	Kafka client library for clojure
	clj-kafka kafka-clj	Provides a producer and consumer that support the underlying sampling API as well as a controlled sequencing interface
Ruby	Karafka	Multi-threaded efficient Kafka processing framework on Ruby and Rails. Based on librdkafka
	WaterDrop	Standalone library, part of Karafka, for creating Kafka messages
	Racecar	Simple framework for writing Kafka consumers in Ruby that integrates well with Rails, based on ruby-kafka
	DeliveryBoy	Easy way to publish messages to Kafka from Ruby applications, based on ruby-kafka
	EM-Kafka	EventMachine Driver for Kafka
	ruby-kafka	Ruby client library for Apache Kafka
Node.js	node-rdkafka	High-performance NodeJS client for Apache Kafka that wraps a native library librdkafka
	Kafka-node	NodeJS client with ZooKeeper integration
	node-kafka-0.8-plus	JavaScript implementation for NodeJS Server with Vagrant and Docker support
	kafka-java-bridge Docs	Nodejs wrapper for the JAVA Kafka API
	Prozess Docs	Low-level protocol support in Node.js
	franz-kafka	Alternative node client from Tagged
Proxy (HTTP REST, etc)	Dory	Producer daemon for Apache Kafka that simplifies the work of clients sending messages to Kafka, freeing them from the complexity of directly interacting with the Kafka cluster
	dropwizard-kafka-http	This project creates an Apache Kafka HTTP endpoint for creating and consuming messages. It is built on the Dropwizard platform which makes Java REST services operator friendly and easy to deploy
	Kafka HTTP	Rest endpoint provides access to Scala's native high-level consumer and producer API
	Kafka REST Proxy Docs	Provides a RESTful interface to a Kafka cluster
	Kafka-Pixy	Dual API (gRPC and REST) proxy server for Kafka with automatic consumer group control. It is designed to hide the complexity of the Kafka client protocol and provide a simple API that is trivial to implement in any language
	Kastle	Efficient Kafka REST Proxy for message producers
	KastleX	Full featured HTTP REST Proxy for Kafka in Elixir with JWT-based authentication
Perl	Kafka Docs	Pure Perl implementation that supports producer and consumer, ZooKeeper integration, GZIP and Snappy compression
stdin/stdout	kcat	Generic producer and consumer without JVM for Kafka
PHP	phpkafka	PHP extension built on librdkafka
	PHP Kafka client	Stable, ready to go and fast Kafka client for PHP based on librdkafka
	log4php-kafka Kafka PHP	PHP library with consumer (simple and based on ZooKeeper), producer and compression support
Rust	Kafka Rust Client Docs	Pure Rust implementation with support for Metadata, produce, fetch and offset queries. Supports Gzip and Snappy compression
Alternative Java	Krackle	Kafka client designed to minimize the number of objects created and therefore reduce garbage collection overhead
Storm	storm-kafka-0.8-plus	Kafka client that is a part of Apache Storm
Scala DSL	scala-kafka	DSL for Scala developers to create and consume messages with Kafka
Swift	Franz	Kafka client for iOS and macOS that includes swift producer implementation, and simple and high-Level consumer

Kafka API

Olga Semme

Contents

To Table of Contents

Overview
Set dependencies
Producer API
Consumer API
Streams API
Connect API
Admin API

Overview

Kafka includes five core APIs:

- Producer API
- Consumer API
- Streams API
- Connect API
- Admin API

APIs in Kafka are used to implement client applications that consume and transmit data to Kafka, manage Kafka brokers and topics, and perform stream processing.

Clients for Java applications are maintained as part of the main Kafka project, others are available as independent open source projects.

NOTE

For a list of non-Java clients, see the Kafka clients article.

Set dependencies

To create an API in Kafka, Apache Maven is used. It is a tool based on the concept of the POM (Project Object Model) that manages the build of the project, loading the appropriate versions of packages and libraries. To download the required package, you need to specify the dependency in the file pom.xml. Based on the given dependency, Maven will download and add the JAR file (Java archive) to the Java path. Java will then be able to find and use the classes defined in the JAR file.

The GAV parameters (groupId, artifactId, version) are used to specify the dependency:

- `groupId` — ID of the object's producer. Often the scheme adopted in the designation of Java packages is used.
- `artifactId` — object identifier. This is usually the name of the module or application being created.
- `version` — the version of the described object.

To use the API, the Maven dependencies in the *pom.xml* file are set as follows:

```
<dependency>
        <groupId>org.apache.kafka</groupId>
        <artifactId>kafka-clients</artifactId>
        <version>3.5.0</version>
</dependency>
```

Producer API

The Producer API allows applications to send streams of data to topics in a Kafka cluster.

Consumer API

The Consumer API allows applications to read data streams from topics in a Kafka cluster.

Streams API

The Streams API allows you to convert data streams from input topics to output topics.

NOTE

- For more information about Kafka Streams, see the KAFKA STREAMS article.
- For information on creating a Kafka Streams application using an ADS cluster, see the Write a Kafka Streams application article.

Connect API

The Connect API allows you to implement connectors that continuously fetch data from some source data system in Kafka or pass data from Kafka to some sink data system. Many Connect users do not need to use this API directly, but they can use pre-built connectors without having to write any code.

NOTE

- For more information about Kafka Connect, see the KAFKA CONNECT article.
- For information on working with Kafka Connect using the ADS Control interface tools, see the articles:
  - ADS Control UI overview
  - Mirror Maker 2 in ADS Control

Admin API

The Admin API supports the management and validation of topics, brokers, ACLs, and other Kafka objects.

NOTE

For a complete list of available APIs, see the kafka 2.8.1 API article.

NiFi REST API

Olga Semme

To Table of Contents

REST API allows using REST requests without using NiFi user interface to perform various operations, for example:

- work with attributes and parameters of the data flow;
- monitoring of the stream and data flow elements;
- configuration of the flow elements.

A description of the API groups provided in NiFi is given below.

API Group	Purpose
Access User	Authentication and token endpoints
Connections	Manage and retrieve a connection configuration, for example: - setting the priority of the queue; - update destination connection.
Controller	Manage controller configuration, for example: - get the current configuration; - manage cluster; - create reporting tasks.
Controller Services	Manage controller services, update links to controller services
Counters	Manage event counters: - receive event counters; - reset event counters.
Data Transfer	Manage transactions, for example: - manage the lifetime of the transaction; - cancel the specified transaction; - send data; - receive data.
Flow	Manage flow configuration, for example: - receive data about the flow (for example, a list of created bulletins); - obtain data on the status of the flow components; - get the history of requests.
FlowFile Queues	Manage and retrieve dataflow queue configurations, for example: - view the contents of the queue; - download the contents of a flow file; - clean the queue.
Funnels	Manage funnels
Input Ports	Manage and retrieve input port configurations, for example: - create an input port; - control remote access to port.
Labels	Label management, for example: - create a label; - set a label style.
Output Ports	Manage and retrieve output port configurations, for example: - create an output port; - control remote access to port.
Parameter Contexts	Create, delete, manage, and retrieve settings context configurations
Parameter Providers	Manage and retrieve parameter provider configurations
Process Groups	Manage and retrieve process group configurations, for example: - create components; - create an instance of the template; - download template.
Processors	Manage and retrieve processor configurations, for example: - create and remove processors; - set the properties of the processor, including the schedule of the processor.
Policies	Manage and retrieve access policy data
Provenance	Manage data provenance: - request for provenance; - search for the provenance of data.
Provenance Events	Manage event provenance: - download the contents of the event; - replay content from an event of a given provenance.
Remote Process Groups	Manage and retrieve remote group configurations, for example: - create a remote group; - enable data transfer.
Reporting Tasks	Manage reporting tasks
Resources	Get available resources that support access/authorization policies
Site to Site	Get data for a "site to site" link: - obtain information about the NiFi instance required for communication (available ports); - obtain information about available peer network members and the status of their NiFi instance.
Snippets	Manage schema fragment: - move a fragment; - delete a fragment.
System Diagnostics	Obtain system diagnostics
Templates	Manage templates, for example: - download a template; - delete a template.
Tenants	Manage user groups (tenants): - add users; - create user groups.
Versions	Manage version process groups

NOTE

For a complete list of REST API available on NiFi, including requests and endpoints, see the /nifi-api article.

Hardware requirements

Olga Semma

Contents

To Table of Contents

ZooKeeper, Kafka Manager, Schema Registry
Kafka Broker
NiFi
ksqlDB
REST Proxy
Arenadata Enterprise Tools
Arenadata Monitoring

IMPORTANT

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

It is preferable to use physical servers, but it is necessary to analyze the power consumption of the equipment you are considering.

Arenadata Streaming works equally effectively both on bare metal and in the cloud.

When providing network availability between servers, the following scenarios are also possible:

- heterogeneous IT infrastructure;
- Multi-cloudic;
- distributed installation on different infrastructures.

The sequence of preparation of physical and virtual hosts is given in the article Prepare hosts.

Examples of requirements for different services depending on the load pattern are listed below.

ZooKeeper, Kafka Manager, Schema Registry

Minimal

Storage	CPU	RAM	Network
400 GB SSD for OS 500 GB SSD for Data	8 cores	32 GB	1 x 10 GbE

Proof of concept

Number of nodes	Storage	CPU	RAM	Network
3 or more	1 x 500 GB HDD for OS 1 x 2.4 TB SSD for Transaction Logs 1 x 1 TB HDD for Data	8 cores	32 GB	1 x 10 GbE

Commodity

Number of nodes	Storage	CPU	RAM	Network
3 or more	2 x 500 GB HDD RAID1 for OS 2 x 2.4 TB SSD RAID1 for Transaction Logs 2 x 1 TB HDD RAID1 for Data	8 cores	32 GB	2 x 10 GbE

Balanced

Number of nodes	Storage	CPU	RAM	Network
3 or more	2 x 500 GB HDD RAID1 for OS 2 x 2.4 TB SSD RAID1 for Transaction Logs 2 x 1 TB HDD RAID1 for Data	8 cores	32 GB	2 x 10 GbE

Compute-intensive

Number of nodes	Storage	CPU	RAM	Network
3 or more	2 x 500 GB HDD RAID1 for OS 2 x 2.4 TB SSD RAID1 for Transaction Logs 2 x 1 TB HDD RAID1 for Data	8 cores	32 GB	2 x 10 GbE

IMPORTANT

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

Kafka Broker

Proof of concept

Number of nodes	Storage	CPU	RAM	Network
3 or more	1 x 500 GB HDD for OS 2 x 500 GB SSD RAID1 for Data	8 cores	32 GB	2 x 10 GbE

Commodity

Number of nodes	Storage	CPU	RAM	Network
3 or more	2 x 500 GB HDD RAID1 for OS 4 x 500 GB SAS12G SSD RAID10 (PCIe Gen3 x4+) for Data	8 cores	32 GB	2 x 25 GbE

Balanced

Number of nodes	Storage	CPU	RAM	Network
3 or more	2 x 500 GB HDD RAID1 for OS 8 x 500 GB SAS12G SSD RAID10 (PCIe Gen3 x8) for Data	16 cores	64 GB	2 x 25 GbE

Compute-intensive

Number of nodes	Storage	CPU	RAM	Network
6 or more	2 x 500 GB HDD RAID1 for OS 8 x 500 GB SAS12G SSD RAID10 (PCIe Gen3 x8) for Data	24 cores	96 GB	2 x 25 GbE

NiFi

Minimal

Storage	CPU	RAM	Network
400 GB SSD for OS 500 GB SSD for Data	8 cores	32 GB	1 x 10 GbE

Proof of concept

Number of nodes	Storage	CPU	RAM	Network
2 or more	1 x 500 GB HDD for OS 1 x 500 GB SSD for Data	8 cores	32 GB	1 x 10 GbE

Commodity

Number of nodes	Storage	CPU	RAM	Network
2 or more	2 x 500 GB SSD RAID1 for OS 4 x 500 GB SAS12G SSD RAID10 (PCIe Gen3 x4+) for Data	12 cores	64 GB	2 x 25 GbE

Balanced

Number of nodes	Storage	CPU	RAM	Network
2 or more	2 x 500 GB SSD RAID1 for OS 8 x 500 GB SAS12G SSD RAID10 (PCIe Gen3 x8) for Data	24 cores	96 GB	2 x 25 GbE

Compute-intensive

Number of nodes	Storage	CPU	RAM	Network
4 or more	2 x 500 GB SSD RAID1 for OS 16 x 500 GB SAS12G SSD RAID10 (2 controllers PCIe Gen3 x8) for Data	48 cores	192 GB	2 x 25 GbE

ksqlDB

Load pattern	Number of nodes	Storage	CPU	RAM
All	2 or more	SSD (volume depends on the number of requests and aggregations)	4 cores	32 GB

REST Proxy

Load pattern	Number of nodes	Storage	CPU	RAM
All	2 or more	HDD (no speed requirement)	16 cores or more (work with producer/ consumer)	1 GB 64 MB per producer 16 MB per consumer

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.

Network requirements

Olga Semme

To Table of Contents

1. One or more dedicated peer-to-peer Interconnect networks are required for internal cluster communication.
2. All cluster servers must be connected to Interconnect networks.
3. All servers must have a fully qualified domain name (FQDN) and the ability to communicate with each other at required speeds.
4. The IP address of each server must be known.
5. Ping access to any servers — internet control message protocol (ICMP) — must be provided.
6. In the case of installation with Internet access, access is required to the Arenadata Cluster Manager deployed on a dedicated server. Installation of Arenadata Cluster Manager is described in the article Install ADCM.
7. In the case of installation without Internet access (offline), access is required to the Arenadata Enterprise Tools deployed on a dedicated server. Installation of Enterprise Tools is described in the section Install Enterprise Tools cluster.
8. The ports below must be open on the servers where you install the cluster.

NOTE

For ADS, each port refers to a server on which a specific service is installed.

ADS

Service	Port	Purpose
Common	22	Port for connection to SSH provider
	2015	TCP port to send metrics to the monitoring server
	2016	UDP port to send metrics to the monitoring server
ZooKeeper	2181	Service access port
	2888 3888	Server-to-server communication ports for quorum components
Kafka	9092	Service access HTTP port
	9093	Service access HTTPS port
	9999	JMX port to access service metrics
Kafka REST Proxy	8082	Service access port
	9998	JMX-port to access service metrics
Schema-Registry	8081	Service access port
	9997	JMX-port to access service metrics
ksqlDB	8088	Access port to the ksqlDB Server component
NiFi	9090	Service UI access HTTP port
	9091	Service UI access HTTPS port
	11443	Communication port between clustered NiFi servers
	18080	Access port to the NiFi Registry component
NiFi2	9090	Service UI access HTTP port
	9091	Service UI access HTTPS port
	11443	Communication port between clustered NiFi servers
	18080	Access port to the NiFi2-Registry component

Monitoring

Port	Purpose
3000	HTTP port used by Grafana. The default value can be changed when installing the monitoring cluster on the configuration page of the Grafana service in the ***ip_and_ports*** section
80	TCP port used by Graphite. The default value can be changed 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

ADPS

All usable ports for an ADPS cluster are listed in Network requirements.

Software requirements

Olga Semme

Contents

To Table of Contents

Operating system
Preinstalled software
Install Java from the Arenadata repository
Installation features
SELinux
Firewall
Supported browsers for NiFi

Operating system

IMPORTANT

- The life cycle of Centos 7 ended on June 30, 2024.
- Ubuntu 22.04 is being replaced by Ubuntu 24.04 in releases after April 1, 2026.
- Alt Linux 8.4 SP is being replaced by Alt Linux 10 SP in releases after January 1, 2026.

1. ADS supports only x86_64 architecture and the following operating systems:
   - Centos 7.9;
   - RHEL 7.9;
   - Ubuntu 22.04;
   - Alt Linux 8.4 SP — available only for the Enterprise version;
   - Alt Linux 10 SP — available only for the Enterprise version;
   - Astra Linux 1.7 "Orel" SE with Axiom JDK — available only for the Enterprise version;
   - Astra Linux 1.7 "Voronezh" SE with Axiom JDK — available only for the Enterprise version;
   - RED OS 7.3 Certified edition — available only for the Enterprise version.
2. You need to prepare all dependent repositories together with the chosen operating system.
3. All machines in the cluster must have access to the official repositories (or create a local mirror):
   - CentOS Extras
   - CentOS Updates
   - CentOS Base
4. In the case of online installation, you must provide access to the repositories listed on the ***Configuration*** tab in the ADS cluster menu in the ADCM UI. The ***Configuration*** tab is described in Configure a cluster.
5. In the case of offline installation, you must provide access to the host where the Enterprise Tools cluster is deployed. Installing an Enterprise Tools cluster is described in the section Install Enterprise Tools cluster.
6. To support kTLS (supported since ADS 3.9.0.1 b1), you need an operating system kernel version 5.9 or higher. kTLS mode is enabled by default. However, if your kernel version is lower than required, you will be rolled back to the default mode.
7. To use Astra Linux "Voronezh", the ADCM service user must have the integrity label `63`: `sudo pdpl-user -i 63 adcm`.

Preinstalled software

1. NTP must be preconfigured on servers.
2. Access must be given to the `root` user or another user with `sudo` rights.
3. Java 17 must be pre-installed on the hosts for all services, and Java 21 for the NiFi2 service.

Centos, RedHat, and AstraLinux operating systems do not have Java 17 in the system packages.

For hosts with any of these operating systems, use one of the following methods to install Java 17 before upgrading/installing the cluster:

- Manually install Java 17 on hosts and configure the JAVA_HOME variable.
- Install Java 17 from the Arenadata repository using the method described below.

CAUTION

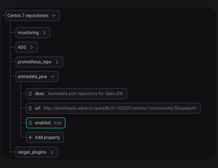
Before installing the NiFi2 service, Java 21 must be installed from the Arenadata repository **for all operating systems**. Manual installation of Java is not suitable.

Install Java from the Arenadata repository

Starting with ADS 3.7.2.1 b1, on the cluster configuration page, there is an option to install Java 17 and Java 21 from the Arenadata repository.

If you enable the option before running the ***Install***, ***Add/Remove components***, or ***Update*** action for a cluster or service, the latest version of Java 17 will be installed on cluster hosts during the action.

To enable the option to install Java 17 and Java 21 from the Arenadata repository, expand the group with the repositories of the selected OS (for example, ***Centos 7 repositories***), select the ***arenadata_java*** repository, and expand its parameters. Set the ***enabled*** parameter to `true`.



Configuring Java installation from Arenadata repository for Centos 7

Once this option is enabled, Java 17 and Java 21 will be installed from the Arenadata repository regardless of what is specified in the JAVA_HOME variable (***Custom JDK settings*** group).

CAUTION

Arenadata is not an official JDK provider, and this feature is provided to ensure a fully automatic installation on systems that do not have the necessary dependencies. In production environments, we recommend using a JDK from official providers.

Installation features

1. The installation process modifies files that should not be controlled by the configuration management system (if available):
   - system file */etc/hosts*;
   - system file */etc/selinux/config*;
   - system file */etc/sysctl.conf*.
2. The installation process creates files in */usr/lib/systemd/system/* that 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).
4. It is recommended to switch off any antivirus software before ADS installation and during the ADS operation process. In case of using antivirus, please contact your antivirus software vendor to find out the settings necessary to allow ADS.

SELinux

SELinux must be turned off. The full guide about this operation you can find in Install ADCM.

Firewall

Firewall must be stopped. The full guide about this operation you can find in Install ADCM.

Supported browsers for NiFi

1. The work of the NiFi service is supported by browsers:
   - Chrome
   - FireFox
   - Edge
   - Safari
2. UI is supported in the current stable release of that browser and the preceding one. For instance, if the current stable release is 45.X then the officially supported versions will be 45.X and 44.X.
3. For Safari, which releases major versions much less frequently, supported versions will be the two latest releases.
4. The supported browser versions are driven by the capabilities the UI employs and the dependencies it uses. UI features are developed and tested against the supported browsers.

Kafka configuration best practices

By AD.S

Contents

To Table of Contents

Configuration best practices for Kafka brokers
- Configure topic replication
- Configure topic partitions
- Retention policies configure
- Set the graceful shutdown
- Set the leadership balance

Configuration best practices for working with producers and consumers
- Configure the producer to receive confirmation of a message record
- Check the consumer position
- Assign consumer groups to topics

Secure environment

Hardware
- Equipment
- Memory
- Processor
- Data storage

Network

Software
- Main guidelines
- File handles
- Virtual memory

NOTE

For information about the basic principles of working in Kafka, you can refer to quick start with Kafka article.

Configuration best practices for Kafka brokers

After installing ADS, the necessary configuration files in Kafka brokers are automatically set in the configuration file /usr/lib/kafka/config/server.properties. The ADS configuration parameters article describes the parameters that can be changed when configuring the Kafka service.

If the required parameter is missing, to add it, use the **Add key/value** line in the **server.properties** group list, where you need to write the key and value in the appropriate fields.

Another way is to add the appropriate line in the /usr/lib/kafka/config/server.properties file.

Some parameters can be set directly in the command line for running scripts using options. This setting overrides the defaults specified in /usr/lib/kafka/config/server.properties.

Recommendations for configuring some parameters for Kafka brokers are given below.

Configure topic replication

Topic replication is determined by the following parameters:

- **default.replication.factor** - sets the replication factor for automatically created topics.
A high replication factor is essential for designing a stable system. With a replication factor of **3** or more, Kafka replicates the log for partitions of each topic across multiple servers. When the server goes down, it can maintain automatic failover to these replicas, messages remain available. For efficient operation of the system, it is recommended to set the replication factor to at least **3**.

This parameter can be changed in the ADCM interface.

It is possible to set the replication factor for each topic separately.

Example of setting the replication factor for a single topic

It is necessary to use the **--replication-factor** option when creating a topic.

```
$ /usr/lib/kafka/bin/kafka-topics.sh --create --topic new-topic --replication-factor 3 --bootstrap-server hostname:9092
```

When requesting information for the created topic, you can see that the topic has 3 replicas:

```
$ /usr/lib/kafka/bin/kafka-topics.sh --describe --topic new-topic --bootstrap-server hostname:9092
```

```
Topic: new-topic        TopicId: 0xRhOBTIQuvtyvRJK3dLlw PartitionCount: 1       ReplicationFactor: 3    Configs: unclean.
Leader.election.enable=false
        Topic: new-topic        Partition: 0    Leader: 1002    Replicas: 1002,1001,1003        Isr: 1002,1001,1003
```

If necessary, it is possible to change the replication factor for a single partition:

Example of changing the replication factor

To change, create custom reassignment.json in JSON file:

```
$ sudo vim increase-replication-factor.json
```

JSON file content:

```
{"version":1,
 "partitions":[{"topic":"new-topic","partition":0,"replicas":[1002,1001,1003]}]}
```

where the replication factor of the partition **0** of the **new-topic** topic is extended to the brokers **1002**,**1001**,**1003**.

Run the remapping process using the generated JSON file with the **--execute** option:

```
$ /usr/lib/kafka/bin/kafka-reassign-partitions.sh --bootstrap-server localhost:9092 --reassignment-json-file increase-
replication-factor.json --execute
```

- **min.insync.replicas** - the minimum number of in-sync replicas required to receive producer requests with **request-required.acks** set, described below.
When the producer sets **request-required.acks** to **all**(or **-1**), **min.insync.replicas** specifies the minimum number of replicas that must acknowledge a write for the write to be considered successful. If this minimum cannot be met, then the producer will raise an error - **NotEnoughReplicas** or **NotEnoughReplicasAfterAppend**.

This parameter can be set in the ADCM interface.

When used together, **min.insync.replicas** and **request-required.acks** provide higher reliability guarantees:

Success scenario:
- topic with replication factor **3**
- **min.insync.replicas** with value **2**
- **request.required.acks** with value **all** (or **-1**)

This combination of options ensures that the producer will raise an error if most of the replicas do not receive a write.

It is possible to set **min.insync.replicas** for each topic separately.

Example of setting a parameter for a single topic

When creating a topic, use the **--config** option with the parameter **min.insync.replicas**:

```
$ /usr/lib/kafka/bin/kafka-topics.sh --create --topic new-topic1 --config min.insync.replicas=3 --bootstrap-server
hostname:9092
```

Configure topic partitions

The **num.partitions** parameter specifies the default number of partitions for each topic created.

Splitting topics into partitions leads to better data balancing and promotes consumer parallelism. For data with keys, you should avoid changing the number of partitions in a topic.

This setting can be changed in the ADCM interface.

It is possible to set the number of partitions for each topic separately.

Example of setting the number of partitions for a single topic

When creating a topic, use the **--partitions** option:

```
$ /usr/lib/kafka/bin/kafka-topics.sh --create --topic topic1 --partitions 3 --bootstrap-server hostname:9092
```

Retention policies configure

To optimize disk space, set the correct policy for storing messages in a topic.

The **controlled.shutdown.enable** parameter is set to **true** by default.

Topic with replication factor **3**.

- **log.cleanup.policy** - defines the policy for deleting log segments. Optional value is valid for all custom topics. Can be changed in the ADCM interface. Possible values:
  - **delete** - deletes old topic segments when their retention time or size limit has been reached.
  - **compact** - enables log compression, which keeps the last value for each key. You can also specify both policies in a comma-separated list (e.g. **delete**,**compact**). In this case, the old segments will be discarded according to the retention time and size configuration, and the remaining segments will be compressed. The **compact** policy makes sense for topics where the last value for each key in the topic is important.
- **log.retention.ms** - specifies the maximum time that messages in a topic will be retained before old topic segments are deleted to make room if the retention policy is set to **delete**. Parameter is set in milliseconds. The parameter can be set in the ADCM interface.
- **log.retention.bytes** - specifies the maximum partition size after which old log segments will be deleted to make room if the storage policy is set to **delete**. There is no size limit by default, only time limit. Since this limit is applied at the partition level, multiply it by the number of partitions to calculate the topic retention in bytes. The parameter can be set in the ADCM interface.

It is possible to set these parameters for each topic separately.

Example of setting a retention policy for a single topic

When creating a topic, use the **--config** option with parameters **cleanup.policy**, **retention.ms**, **retention.bytes**:

```
$ /usr/lib/kafka/bin/kafka-topics.sh --create --topic new-topic1 --config cleanup.policy=delete --config retention.ms=10000 --
config retention.bytes=120000 --bootstrap-server localhost:9092
```

Set the graceful shutdown

Graceful shutdown is important to system reliability.

The **controlled.shutdown.enable** parameter is set to **true** enables a controlled shutdown of the server. The parameter can be set in the ADCM interface.

When a server goes down or is intentionally taken down for maintenance or a configuration change, the Kafka cluster will automatically detect any outage or broker failure and select new leaders for the partitions on that machine.

When the server gracefully stops, it uses two optimizations:
1. Synchronizing all logs to disk to avoid having to rebuild the logs on restart (i.e. checking the checksum for all messages at the end of the log). Log recovery takes time, so it speeds up otherwise restarts.
2. Move all partitions for which the server is the master to replicas before shutting down. This will speed up the transfer of leadership and minimize the downtime of each partition to a few milliseconds.

Note that a controlled shutdown will only succeed if all partitions hosted on the broker have replicas (i.e. **default.replication.factor** is set to **2** or greater).

Set the leadership balance

Whenever a broker stops or crashes, control of that broker's partitions is transferred to other replicas. After the broker is restarted, it will only be a slave for all its partitions, that is, it will not be used for client reads and writes. To avoid this imbalance, Kafka has the concept of preferred replicas. If the replica list for a partition is 1, 5, 9, then node 1 is preferred as the leader over node 5 or 9 because it is first in the replica list.

The **auto.leader.rebalance.enable** parameter set to **true** enables automatic leader rebalancing (returning leadership to the preferred replica) in the background at regular intervals. The parameter can be set in the ADCM interface.

If you set this parameter to **false**, you should manually restore the leadership of the restored replicas after a server reboot. To do this, you need to run the command:

```
$ /usr/lib/kafka/bin/kafka-leader-election.sh --bootstrap-server localhost:9092 --election-type preferred --all-topic-partitions
```

Configuration best practices for working with producers and consumers

Configure the producer to receive confirmation of a message record

The **request.required.acks** parameter specifies when a production request is considered complete. In particular, how many other brokers should log the data and confirm it to the leader. Possible values:

- **0** - the producer never waits for confirmation from the broker. This option provides the least latency, but guarantees the least reliability (some data will be lost if the server crashes).
- **1** - the producer is acknowledged after the master has received data. This is the default value.
- **-1** - the producer receives confirmation after all synchronized replicas have received data. This option provides the best stability.

Using the **min.insync.replicas** options described by above and **request.required.acks** together allows for higher reliability guarantees.

Parameter setting example

When setting messages to a topic, use the **request-required-acks** option:

```
$ bin/kafka-console-producer.sh --topic new-topic --request-required-acks -1 --bootstrap-server hostname:9092
```

Check the consumer position

Kafka stores consumer offsets in a special topic so it is possible to see the position of the topic consumers.

Example of viewing the position of the topic consumers

Command to see information for a consumer group:

```
$ /usr/lib/kafka/bin/kafka-consumer-groups.sh --bootstrap-server localhost:9092 --describe --group new-group
```

The result shows the consumer's position in the **new_group** group and how far behind it is from the end of the **new-topic** topic log.

```
GROUP           TOPIC           PARTITION  CURRENT-OFFSET  LOG-END-OFFSET  LAG             CONSUMER-ID
HOST            CLIENT-ID
new-group       new-topic       2          2               3               1               consumer-new-group-1-e7833a5a-9111-
4e0c-a028-6ae33b17f7e /10.51.10.30  consumer-new-group-1
```

Assign consumer groups to topics

Every message is sent to each consumer group that subscribes to a topic or partition, but within a group it is sent to only one consumer. Thus, all consumer groups that subscribe to a topic receive messages, but only one consumer in a consumer group receives a message from the topic.

If you need to send a message to multiple consumers, you should assign them to different consumer groups.

Example of assigning a group to a consumer

When connecting to a topic, use the **--group** option:

```
$ /usr/lib/kafka/bin/kafka-console-consumer.sh --topic new-topic --from-beginning --group new-group --bootstrap-server
hostname:9092
```

Secure environment

For information about configuring security in a Kafka environment, you can go to **Basic Operations** → **Kafka** → **Access management**.

Hardware

Equipment

- If possible, use separate servers or racks for Kafka brokers.
- To apply the optimal replication factor, have at least 3 Kafka brokers in the system.
- Provide Ink (high Availability) by meeting the requirements described in the hardware requirements article in accordance with the planned load.

Memory

- Kafka can run on a large file system to store and cache messages and use little heap space.
- Kafka may need enough memory to buffer active reads and writes from.

Processor

- Enabling SSL channel encryption may increase CPU requirements (exact data depends on CPU type and JVM implementation).
- When choosing a processor between faster processors or more cores, the higher number of cores should be selected. The extra parallelism that multiple cores offer is preferable to a higher clock speed.

Data storage

- It is not recommended to use a JBOD disk array because tiered storage requires a single mount point.
- It is recommended to use multiple drives to maximize throughput.
- It is not recommended to share the same drives used for Kafka data with application logs or other OS file system activity.
- If you are setting up multiple data directories, the broker places the new partition on the path with the fewest partitions saved. Each partition will reside entirely in one of the data directories. If data is not balanced between partitions, this may lead to imbalance between disks.
- A RAID disk array can potentially do a better job of load balancing across disks because it balances the load at a lower level.
- A RAID 10 disk array is recommended as the best option for most cases. It provides improved read and write performance, data protection (ability to withstand disk failures), and fast recovery.
- Network Attached Storage (NAS) is not recommended. NAS is often slower, shows higher latency with wider average latency variance, and is a single point of failure.

Network

For network connection requirements, see Network requirements article. Low latency ensures that nodes can easily communicate, while high throughput helps move and restore shards. Modern data center networks (1 GbE, 10 GbE) are sufficient for the vast majority of clusters.

Software

Main guidelines

- For software requirements, see Software requirements article.
- It is recommended to use NTP to set the system's to run Kafka.
- Use the latest released version of Java to ensure that known security issues are resolved.
- The recommended GC setup (tested on a large deployment with JDK 1.8 u5) is as follows:

```
-Xmx6g -Xms6g -XX:MetaspaceSize=96m -XX:+UseG1GC -XX:MaxGCPauseMillis=20
-XX:InitiatingHeapOccupancyPercent=35 -XX:G1HeapRegionSize=16M
-XX:MinMetaspaceFreeRatio=50 -XX:MaxMetaspaceFreeRatio=80
```

- The guidelines below may be helpful to improve compatibility between Kafka and your operating system. For the best application of recommendations, you should refer to the documentation for a specific distribution.

File handles

Kafka has the potential to require a relatively large number of available file handles at any given time.

Many modern Linux distributions come with only 1024 file descriptors allowed per process. This is not enough for Kafka.

You need to increase the number of file handles to at least 100,000, and possibly much more.

To change the open file limit, you need to edit /etc/security/limits.conf a file that sets resource limits for users logged in through PAM(Privileged Access Management) (PAM). To do this, for all user groups, increase the value of the **nofile** parameter - the maximum number of open files for types **hard** and **soft** , by adding the following lines to /etc/security/limits.conf:

```
* hard  nofile  100000
* soft  nofile  100000
```

Virtual memory

You may need to change the virtual memory size limit **vm.max_map_count** to allow the required number of mmap functions to be processed.

To calculate the current number of mmaps, you need to know the number of .index files in the Kafka data directory. .index files are the most memory-mapped files.

You can count .index files with this command:

```
$ find . -name '*.index' | wc -l
```

Set **vm.max_map_count** in the /etc/sysctl.conf file for the session. This will set the current number of memory-mapped files. The minimum **mmap** limit value (**vm.max_map_count**) is the number of open **splits** files.

Set **vm.max_map_count** significantly higher than the number of .index files to account for broker segment growth.

```
$ sysctl -w vm.max_map_count=262144
$ echo "vm.max_map_count=262144" >> /etc/sysctl.conf
$ sysctl -p
```

NIFi configuration best practices

Olga Semme

Contents

To Table of Contents

Increase the number of file handles

Increase the number of forked processes

Increase the number of available TCP socket ports

Set the duration of the TIMED_WAIT state for sockets

Disable swap option

Increase the hardware resources

Typical settings for Linux may not be suitable for I/O intensive applications such as NiFi. For the areas of the operating system used, the requirements of the distribution may vary.

The guidelines below may be helpful in improving compatibility between NiFi and your operating system.

For the best application of the recommendations, you should refer to the documentation for a specific distribution.

Increase the number of file handles

NiFi could potentially have a very large number of open file handles at any given time.

File handles are integer identifiers that define data structures. The Linux kernel refers to these structures as file structures because they describe open files. The file structure index is a file handle.

To change the open file limit, you need to edit */etc/security/limits.conf*, a file that sets resource limits for users logged in through PAMPrivileged Access Management (PAM). To do this, for all user groups, increase the value of the `nofile` parameter — the maximum number of open files for types `hard` and `soft`, by adding the following lines to */etc/security/limits.conf*:

```
*  hard  nofile  50000
*  soft  nofile  50000
```

It is necessary to take into account the size of the operating system kernel memory when increasing the limit.

Increase the number of forked processes

NiFi can be configured to create a significant number of flows. To increase the allowed number, you also need to edit the */etc/security/limits.conf* file. To do this, for all user groups, increase the value of the `nproc` parameter — the maximum number of processes for the `hard` and `soft` types, by adding the following lines to */etc/security/limits.conf*:

```
*  hard  nproc  10000
*  soft  nproc  10000
```

And also, if available, you need to edit the */etc/security/limits.d/90-nproc.conf* or */etc/security/limits.d/20-nproc.conf* file, which overrides the `nproc` value specified in the */etc /security/limits.conf* file. In this file, it is necessary for all user groups to increase the value of the `nproc` parameter — the maximum number of processes for the `soft` type, by adding the line:

```
*  soft  nproc  10000
```

Increase the number of available TCP socket ports

The number of available TCP socket ports is especially important if your thread will be setting up and dropping a large number of sockets in a short period of time.

To set the maximum number of TCP sockets, run the command:

```
$ sudo sysctl -w net.ipv4.ip_local_port_range="10000 65000"
```

Set the duration of the TIMED_WAIT state for sockets

TIME_WAIT is the state of the server socket in which it will time out in order to collect randomly delayed packets on the network. The timeout duration can be changed to allow you to quickly configure and disable new sockets.

The following commands are used to change the timeout duration:

- for distributions with kernel 2.6:

```
$ sudo sysctl -w net.ipv4.netfilter.ip_conntrack_tcp_timeout_time_wait="1"
```

- for distributions with kernel 3.0:

```
$ sudo sysctl -w net.netfilter.nf_conntrack_tcp_timeout_time_wait="1"
```

Disable swap option

The swap setting affects the kernel's preference to move memory pages from applications to the swap page rather than releasing memory from the cache.

To improve the efficiency of NiFi, it is recommended to set the `swappiness` parameter to zero. This ensures that if memory is limited, the page cache will be reduced in an attempt to reclaim memory before the application's pages are moved to swap space.

To disable the swap option, you need to edit */etc/sysctl.conf*, the file that controls kernel options.

You need to add the following line:

```
vm.swappiness = 0
```

For partitions handling different NiFi repositories, disable the `atime` parameter — a field to write the timestamp. This can cause an unexpected bandwidth spike. To do this, in the */etc/fstab* file (which is used to configure mount options for various block devices, disk partitions, and remote file systems), add the `noatime` parameter for the partitions of interest, which completely disables the recording of file access time.

Increase the hardware resources

For efficient operation of NiFi, it is recommended:

- If possible, use separate servers or racks for NiFi instances.
- Provide HA (High Availability) in accordance with the requirements described in the Hardware requirements article.

Use cases

Olga Semme

Contents

To Table of Contents

Messaging

Website activity tracking

Metrics

Log aggregation

Stream processing

Event sourcing

Commit log

The article describes the use cases for ADS.

Messaging

Message brokers are used for the optimal organization of the messages exchange. A message broker is an intermediary computer program module that translates a message from the formal messaging protocol of the sender to the formal messaging protocol of the receiver. Kafka has several advantages over most message brokers:

- High **throughput** allows you to process a large amount of data in a short period of time.
- Organized built-in **partitioning** allows to control the storage time and amount of data due to the fact that the messages are divided into segments by partitions and are not stored in one large file.
- **Replication** of partitions provides an automatic failover opportunity to replicas of partitions when a server in the cluster fails, so messages remain available even in case of failures.
- **Fault-tolerance** ensures that the messaging system will keep messages in the absence of active subscribers until the consumer subscribes to the message delivery queue.

Services that complement and optimize the work of Kafka

Service	Interoperability with Kafka	Advantages of use
NiFi	NiFi as producer — takes data from sources directly to a central NiFi instance that delivers data to the appropriate Kafka topic	■ Data transfer to Kafka without special software. ■ Ability to visually monitor and control the data pipeline.
	NiFi as consumer — takes data from Kafka and transfers it to another system	■ Data transfer from Kafka to some file systems (HFDS, HBase) without special software. ■ Ability to visually monitor and control the data pipeline.
	Dynamic self-adjusting data flow — combining the power of NiFi, Kafka, and an analytic platform: 1. NiFi sends data to Kafka. 2. Data becomes available for analytical platforms. 3. Results are written back to another Kafka topic. 4. Data goes to NiFi.	■ Ability to further use the data received by NiFi without special software. ■ Ability to visually monitor and control the data pipeline.
ZooKeeper	Informs each Kafka broker about the current state of the cluster	Automatic updating of metadata in the Kafka client when connecting to any broker
Confluent Schema Registry	Implements a mechanism for working with different Kafka data schemas	■ Definition of the fields required to describe the event, the type of each field. ■ Documentation of events and values of each of its fields in an accessible form. ■ Preventing consumers from receiving corrupted data. ■ Working with different data formats from different information sources.

Website activity tracking

Kafka is a great tool for tracking website activity. When a new user registers on the website, their activity can be tracked as follows:

1. The user presses a button in the web page interface.
2. The web application creates a message with the metadata of this button.
3. Messages with metadata for this button are collected in a data packet and sent to Kafka, while creating **commit logs**.
4. On the next user action with this button, a message about the action is added to the commit logs, and the offset of the message in the queue increases.

Further from Kafka this data can be collected for analytics. This data shows real-time website usage.

Metrics

Kafka can be used to quickly collect metrics of various applications and operational data:

- technological processes;
- audit statistics and data collection;
- system activity;
- aggregation of statistics of running applications and infrastructure;
- trackers of consumption of data streams by users in real time.

Kafka has several advantages when used in monitoring:

- Ability to connect new producers to send metrics, and then use the monitoring data for several different systems.
- Ability to perform real-time analysis on a large data set, while collecting metrics.
- The consumer application may use a small amount of code to process data.
- Ability to allocate data to different modules in accordance with their purpose.

Log aggregation

Log aggregation is one possible use case for ADS. Kafka is the best tool for log aggregation for distributed environments with different architectures. Kafka collects physical log files from servers and places them in a central location (perhaps a file server or HDFS) for processing. Compared to other systems, Kafka has a number of advantages:

- Kafka abstracts away the details of files and gives a clearer abstraction of log data or events as a message stream. This provides lower latency processing and makes it easier to support multiple data sources and distributed data consumption.
- Kafka offers the same high performance as other systems but better reliability guarantees through replication and much lower end-to-end latency.

Stream processing

Stream processing is used to improve messaging performance. Benefits of stream processing using Kafka over batch pipelines:

- Message streams are consumed in real time.
- Transformation, filtering, aggregation or join operations are applied to messages in order to publish the processed messages in another thread.
- Stream pipelines reduce the load on the data source, because you do not need to execute full queries, but you can extract data from the log files of the DBMS or other systems.
- Messages can be kept as long as needed.

Tools and services for stream processing

Tool or Service	Role in Stream Processing	Advantages of use
Kafka Streams	Client library for developing streaming Big Data applications that work with data stored in Kafka topics	Powerful and flexible API with all the benefits of Kafka (scalability, reliability, minimum latency, analytic query mechanisms) allows the developer to write code in local mode (outside the cluster)
NiFi	Uses the concept of a stream as a sequence of operations: transfer, transformation and enrichment of data over a sequence of individual events	A stream is NOT treated as a large batch operation that requires an initial load of all data before processing can begin. For example, a SQL database with millions of rows is treated as millions of individual rows that need to be processed
ksqlDB	Platform for streaming Big Data processing in Kafka using structured SQL queries	Data structures in Kafka SQL are software units that are able to store and process a lot of data, linked by ksqlDB logic

Event sourcing

Kafka is the best option to support applications built with the **event sourcing** pattern for event processing. Advantages of using Kafka in event sourcing:

- Support for storing very large log data.
- The ability to work with delayed events due to the queue and offset mechanisms in Kafka.

Commit log

Kafka can serve as an external commit log for a distributed system. Advantages of using Kafka:

- The ability to replicate data between nodes.
- Possibility of data recovery (re-synchronization mechanism for failed nodes).
- Log compression feature in Kafka.

ADS architecture

Olga Semme

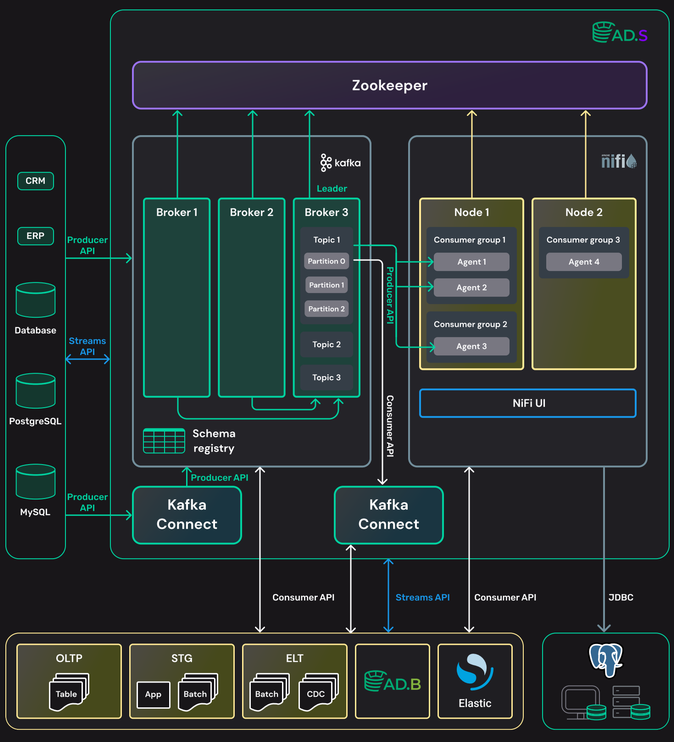
Contents

To Table of Contents

- Overview
  - Kafka
  - ZooKeeper
  - NiFi
  - API

Overview

For each use case, the architecture of ADS is determined by the consumer, depending on the tasks. The high-level architecture view of one of the options of ADS is shown below.



ADS architecture

Kafka

- Every named **topic** in Kafka consists of one or more **partitions** distributed among **brokers** within one ADS cluster.
- When a new message is added to a topic, it is written to one of that topic's partitions. Messages with the same keys are always written to the same partition, thereby guaranteeing the sequence or order of writing and reading.
- To ensure data integrity, each partition in Kafka can be replicated multiple times. This ensures that there are multiple copies of the message stored on different brokers.
- Every partition has a **leader** — a broker that works with clients. It is the leader that works with the **producers** and generally gives messages to the **consumers**. Messages are always sent to the leader and, in general, are read from the leader.
- For scalably and reliably streaming data between Kafka and other data systems Kafka Connect is used. Kafka Connect can ingest entire databases or collect metrics from all your application servers into Kafka topics, and can also deliver data from Kafka topics, for example, into data analysis and retrieval systems (ADLS) or database management system (ADB).
- Confluent Schema Registry ensures schemas compatibility between Kafka producer and consumer.

ZooKeeper

- ZooKeeper decides which of the brokers is the controller (i.e. the broker responsible for choosing the partition leaders). It also determines the state of the partition leaders and their replicas.
- ZooKeeper also acts as a consistent metadata storage and distributed log service. In the event of a broker crash, it is in ZooKeeper that the controller will write information about new partition leaders.

NiFi

- To work with stream files, NiFi uses a lot of handlers (processors) — separate code fragments to perform a specific operation with stream files.
- Every Kafka consumer is part of a **Consumer Group** that is configured by the ConsumeKafka processor in NiFi with a specific Kafka broker and group name.
- Every Consumer Group has a unique name and is registered by brokers in the Kafka cluster. Data from the same topic can be read by multiple Consumer Groups at the same time.
- When several consumers read data from Kafka and they are members of the same group, each of them receives messages from different topic partitions, thus distributing the load. NiFi in this case acts as a Kafka consumer and implements all the data processing logic.
- It is also possible to transfer data from Kafka to another system using NiFi.
- The NiFi web user interface allows development, management and monitoring in a single interface.

API

- **Producer API** allows an application to publish a message stream to one or more topics of the Kafka platform.
- **Consumer API** allows an application to subscribe to one or more topics and process the post streams that belong to them.
- **Streams API** allows an application to act as a **stream processor**. It consumes an input data stream from one or more topics and creates an output data stream for one or more topics as well. Thus Streams API effectively converts input streams into output streams.

Storage concepts in Kafka

Olga Semina

AD.S

Contents

To Table of Contents

Main components of Kafka

Basic settings when working with Kafka

Sequence of writing and reading messages

Replication and distribution of partitions in the broker

- Replication
- Replica synchronization
- Brokers

Consumer groups

- Point to point
- Publish-subscribe

Main components of Kafka

The components that form the basis of the Kafka storage concepts:

- **Message** – a record of an event that has occurred, including the state of the object, the value of a physical quantity, or any other parameter that needs to be tracked, stored, or transferred to another system.
- **Producer** – a client application that publishes (writes) messages to Kafka.
- **Consumer** – a client application that subscribes (reads and processes) messages.
- **Topic** – a category in which messages are published in Kafka. In a topic messages are written to the commit log.
- **Commit log** – an ordered structure of messages that is write-only. Commit log messages are not editable or deletable. Large logs are divided into partitions. If the log consists of one partition, then it is a partition.
- **Partition** – an ordered immutable sequence of records that is constantly added to the topic log.
- **Broker** – a server where partitions are stored.
- **Segment** – a part of the partition stored in a separate file on disk.
- **Consumer group** – multiple consumers grouped together to receive messages from one or more topics. Partitions of one topic are automatically evenly distributed among consumers in the same group. Each partition is available to only one consumer from the group.

Basic settings when working with Kafka

When writing messages, the producer specifies:

- topic to which the data will be written;
- partition where messages with the same key will be written (if necessary for structuring the data in the topic);
- retention period of messages in Kafka;
- necessary compression;
- required level of confirmation about writing a message to the partition and its replicas;
- partition replication factor.

The consumer for reading messages specifies:

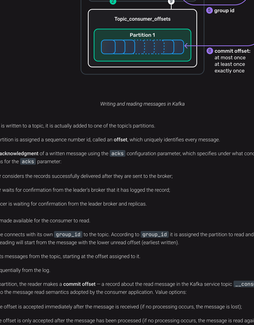
- number of unique **consumer groups**;
- `group_id` for each of the consumer groups, which determines where the group will read messages from;
- maximum time the client has to wait for a response to a request;
- required level of message delivery confirmation (semantics).

The message has the following parameters:

- **data**;
- **key** to determine whether messages are written to the same partition if necessary (for example, client identifier);
- **timestamp** and others.

Sequence of writing and reading messages

The figure below shows the sequence of message processing within a partition when writing and reading.



Writing and reading messages in Kafka

1. When a new message is written to a topic, it is actually added to one of the topic's partitions.
2. Every message in a partition is assigned a sequence number id, called an **offset**, which uniquely identifies every message.
3. The producer obtains **acknowledgment** of a written message using the `acks` configuration parameter, which specifies under what condition the message is considered written to the partition. Options for the `acks` parameter:
   - `0` – the producer considers the records successfully delivered after they are sent to the broker;
   - `1` – the producer waits for confirmation from the leader's broker that it has logged the record;
   - `all` – the producer is waiting for confirmation from the leader broker and replicas.
4. This message is then made available for the consumer to read.
5. The consumer instance connects with its own `group_id` to the topic. According to `group_id` it is assigned the partition to read and the offset in that partition, if any. If they are not specified, the reading will start from the message with the lower unread offset (earliest written).
6. The consumer requests messages from the topic, starting at the offset assigned to it.
7. Messages are read sequentially from the log.
8. When reading from a partition, the reader makes a **commit offset** – a record about the read message in the Kafka service topic `__consumer_offsets`. The offset commit is performed according to the message read semantics adopted by the consumer application. Value options:
   - at most once – the offset is accepted immediately after the message is received (if no processing occurs, the message is lost);
   - at least once – the offset is only accepted after the message has been processed (if no processing occurs, the message is read again);
   - exactly once – all messages are delivered once.
9. The message with the next offset becomes available for reading.

The metadata of each message is its offset or position in the log. The offset to start reading the message can be set by the consumer:

- linear advancement of the offset when reading the record (by default);
- read the record in any order if necessary (revert to an older offset or jump to the most recent actual record).

Replication and distribution of partitions in the broker

Replication

To make messages fault-tolerant and highly available, each topic with its partitions can be **replicated**. This means to create a copy and put it on the server (write it to disk).

Replication factor – a parameter that determines how many copies of each partition will be distributed on brokers. The recommended value for the replication factor (default value) is `3`.

Each partition replica is written on a separate broker. This means that if the broker on which one of the replicas is located fails, there will be two more copies of this partition left. This ensures that data is always available, even during broker maintenance.

Brokers break each partition into **segments**. Each segment is stored in a separate data file on a disk connected to the broker. By default, each segment contains either 1 GB of data or a week worth of data, whichever limit is reached first. When the segment limit is reached, the broker closes the file and creates a new file to record new messages. **Active segment** is the file that is currently being written to. **Closed segment** is the file in which messages are stored and new messages are not written.

Replica synchronization

The replication format is called **InSync**. This means that of all the replicas, one is the **leader**, and the others are **followers**. The producer can control the synchronous/asynchronous replication with the **`acks`/acknowledgment** option.

Writing and reading messages is carried out through the leader replica of the partition. The follower that contains all the messages held by the leader is called an **in-sync replica**.

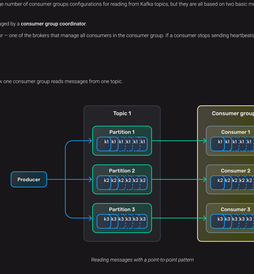
Brokers

To ensure fault tolerance in a Kafka cluster, partition leaders are automatically distributed among different brokers. Each broker can be a leader for some partitions and a follower for others.

If a broker acting as a leader for a partition goes down, any broker that is up to date or synchronized for that partition can take over the leader role.

The figure below shows how partitions can be distributed with a given cluster configuration:

- number of brokers – 3;
- number of topics – 2;
- number of partitions in `Topic 1` – 2;
- number of partitions in `Topic 2` – 1;
- replication factor – 3.



Distribution of partitions between Kafka brokers

Consumer groups

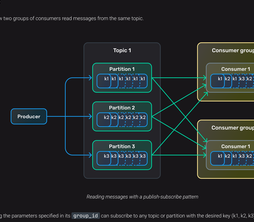
It is possible to create a large number of consumer groups configurations for reading from Kafka topics, but they are all based on two basic messaging schemes – **point-to-point** and **publish-subscribe**.

Consumer groups are managed by a **consumer group coordinator**.

Consumer group coordinator – one of the brokers that manage all consumers in the consumer group. If a consumer stops sending heartbeats, the coordinator initiates consumer rebalancing.

Point to point

The figure below shows how one consumer group reads messages from one topic.



Reading messages with a point-to-point pattern

This is an ideal load distribution option.

The consumer group has a `group_id` that contains information about which topic to connect to and where to start reading from, and Kafka evenly distributes all partitions among the consumers of the same group.

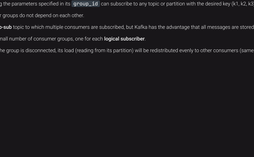
Kafka gives access to a partition to only one consumer (from a group of consumers) and each of them reads data from one partition that has the same key (k1, k2, k3).

In this configuration, the Kafka broker only passes messages sequentially to the consumer when it requests them. This requires a minimum of additional settings.

When one consumer from the group is disconnected, its load (reading from its partition) will be redistributed evenly to other consumers.

Publish-subscribe

The figure below shows how two groups of consumers read messages from the same topic.



Reading messages with a publish-subscribe pattern

Every consumer group using the parameters specified in its `group_id` can subscribe to any topic or partition with the desired key (k1, k2, k3).

At the same time, consumer groups do not depend on each other.

Kafka acts like a normal **pub-sub** topic to which multiple consumers are subscribed, but Kafka has the advantage that all messages are stored and can be processed multiple times.

Most often, topics have a small number of consumer groups, one for each **logical subscriber**.

When one consumer from the group is disconnected, its load (reading from its partition) will be redistributed evenly to other consumers (same as in the **point-to-point** reading scheme).

Log compaction in Kafka

by Maja Kuzmić

Contents

To Table of Contents

Log cleanup policy
Delete
Compact
- Compact policy overview
- Start compaction
- Compaction algorithm
- Deleting using compaction
- Compaction guarantees and benefits

Delete and compact
Configure log cleanup
- Configure a cluster
- Configure a topic

This article describes the existing log cleaner (compaction) policies in Kafka, the parameters used when configuring the policies, and how to configure the policies.

Log cleanup policy

log.cleanup.policy – the main parameter that determines the strategy for deleting old records.

The following are possible log purge policy options:

- delete – deleting records due to reaching one of the specified parameters or two parameters at the same time:
  - messages storage period
  - volume of stored data (in bytes).
- compact — retaining only the latest values for each key.
- delete and compact — retaining only the latest values for each key, taking into account the parameters of storage periods or volume of information.

The choice of log cleanup policy depends on the project requirements.

The figure below shows how the message queue differs after applying different log cleanup policies.

Log cleanup policies

The following describes the storage algorithms depending on the selected policy for a message at offset 5:

- with the `delete` policy, the message is deleted because the storage time is longer than the specified value;
- with the `compact` policy, the message is stored in the queue, since it contains the last value for the `key2` key;
- with the `delete and compact` policy, the message is deleted due to expiration, even though the value for the `key2` key was not overwritten.

All log cleanup policies are based on working with segment partitions.

All log cleanup policies and the parameters used are discussed in detail below.

Delete

The `delete` policy is suitable for logs where only events data is required and messages are not overwritten.

When you select a deletion policy, messages are deleted according to the configured settings:

- log.retention.hours — storage time for a message, after which it will be deleted. This parameter is set by default for all brokers and created topics (value `168`). This value can also be changed for each topic individually via the retention.ms parameter.
- log.retention.bytes — log size in bytes, after reaching which the log will be deleted (default value is `-1` – no limit). This value can also be set for each topic individually via the retention.bytes parameter.

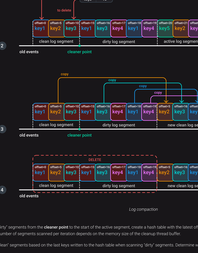
It is also possible to set both parameters at once to simultaneously control the message lifetime and log volume.

To control how often messages are deleted, use the `log.cleanup.interval.ms` parameter. This parameter determines to determine how often the log cleaner checks for logs to be deleted.

The message deletion algorithm distinguishes between the following types of log partition segments:

- **Log segment** – segment where messages are stored and new messages are not written.
- **Active log segment** – segment that is currently being written to. The active segment does not participate in the message deletion algorithm.

The diagram for deleting Kafka log messages with a retention time limit is shown in the figure below.



Deletion policy

Messages are deleted as part of segments in the following sequence:

1. Determining the message with the maximum timestamp (the newest one available) within a single segment.
2. Determining the difference between the maximum timestamp (`Tmax`) and the current time (`Tcurrent`), comparing the resulting value with the value specified in `log.retention.hours` `retention_ms` or topics).
3. If the condition `Tcurrent - Tmax > log.retention` is met for a segment, then the entire segment is deleted.

Thus, the total lifetime of a message may be greater than the value specified in `log.retention.hours` and depends on the segment size value set via segment.bytes.

Compact

Compact policy overview

The **compact** policy is typically used for the logs containing messages in key/value format. For example, databases that store frequently overwritten values for the same keys.

This policy is the default for the `__consumer_offsets` Kafka service topic.

The log compaction algorithm distinguishes between the following types of log partition segments:

- **Clean log segment** – segment in which there are no duplicate keys, i.e. the segment contains only one value for each key – the last one at the time of the previous compaction. In "clean" segments there may be gaps in the offset numbering due to previous compaction.
- **Dirty log segment** – each segment that was written after the last "clean" segment. They may contain duplicate keys, as well as keys present in "clean" segments. "Dirty" segments "stop" at the **cleaner point**.
- **Active log segment** – segment that is currently being written to. The active segment does not participate in the log compaction algorithm.

Start compaction

The moment the compaction starts is regulated by the broker parameters:

- log.cleaner.min.cleanable.ratio – ratio of the volume of "dirty" segments to the total volume of the log at which compaction starts (the default value is `0.5`). This value can also be set for each topic individually via the min.cleanable.dirty.ratio parameter.
- log.cleaner.min.compaction.lag.ms – the minimum time during which a message will remain in the log uncompacted (default is `0` – no limit). Compaction will not start until the specified time has elapsed, even if the value of the `min.cleanable.dirty.ratio` parameter has reached the specified level. This value can also be set for each topic individually via the min.compaction.lag.ms parameter.
- log.cleaner.max.compaction.lag.ms – the maximum time during which a message cannot be compacted in the log (the default value is `9223372036854775807` for all created topics). Compaction will begin after the specified time has elapsed, even if the value of the `min.cleanable.dirty.ratio` parameter has not reached the specified level. This value can also be set for each topic individually via the max.compaction.lag.ms parameter.

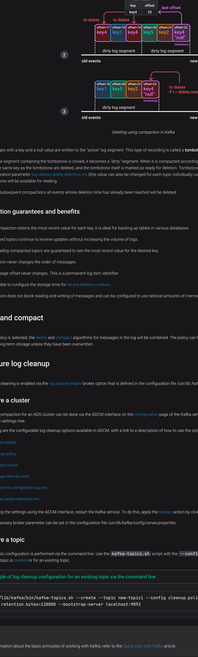
Compaction algorithm

The basis of the compaction algorithm is the creation of a **hash table**, which is the result of scanning "dirty" segments. The hash table records the last offsets found for each key. Next, using this table, the cleanup thread copies the entire log to a buffer, removes offsets that are not in the hash table, and rewrite the remaining offsets into a new segment. This preserves the original offset of each event (maybe with gaps in the offsets).

The size of the cleanup thread buffer and its fill factor are determined by the broker configuration parameters:

- log.cleaner.io.buffer.size
- log.cleaner.io.buffer.load.factor

The Kafka log compaction sequence is shown in the figure below.



Log compaction

1. Scan "dirty" segments from the **cleaner point** to the start of the active segment, create a hash table with the latest offsets, and determine which messages will be deleted. The exact number of segments scanned per iteration depends on the memory size of the cleanup thread buffer.
2. Scan "clean" segments based on the last keys in the hash table when scanning "dirty" segments. Determine which messages will be deleted.
3. Copy retained offsets in both "dirty" and "clean" segments to new log segments. Depending on the segment size value set segment.bytes, messages can be copied into one or more new segments.
4. Removing old segments and setting a **new cleaner point** at the last cleaned offset that marks the start of the first "dirty" segment, which will become the "active" segment after filling. The next stage of log compaction will start from this segment.

Deleting using compaction

After writing a message with a key and a null value, the message will be removed from the log.

The sequence for deleting a log entry using compaction is shown in the figure below.

Deleting using compaction in Kafka

1. Messages with a key and a null value are written to the "active" log segment. This type of recording is called a **tombstone**.
2. After the segment containing the tombstone is closed, it becomes a "dirty" segment. When it is compacted according to the algorithm described by above, all previous messages with the same key as the tombstone are deleted, and the tombstone itself is marked as ready for deletion. Tombstone is assigned a deletion time determined by the broker configuration parameter log.cleaner.delete.retention.ms (this value can also be changed for each topic individually using the parameter delete.retention.ms). During this time, the tombstone will be available for reading.
3. During subsequent compactions all events whose deletion time has already been reached will be deleted.

Compaction guarantees and benefits

- Since compaction retains the most recent value for each key, it is ideal for backing up tables in various databases.
- Compacted topics continue to receive updates without increasing the volume of logs.
- Users reading compacted topics are guaranteed to see the most recent value for the desired key.
- Compaction never changes the order of messages.
- The message offset never changes. This is a permanent log item identifier.
- It is possible to configure the storage time for record deletion markers.
- Compaction does not block reading and writing of messages and can be configured to use rational amounts of memory resources so as not to impact producers and consumers.

Delete and compact

When this policy is selected, the delete and compact algorithms for messages in the log will be combined. The policy can be applied to logs with messages in key/value format that do not require long term storage unless they have been deleted.

Configure log cleanup

In Kafka, log cleaning is enabled via the log cleaner enable broker option that is defined in the configuration file /usr/lib /kafka/config/server.properties

Configure a cluster

Configuring compaction for an ADS cluster can be done via the ADCM interface on the configuration page of the Kafka service. To do this, open the **server.properties** node in the configuration settings tree.

The following are the configurable log cleanup options available in ADCM, with a link to a description of how to use the option:

- log.cleaner.enable
- log.cleanup.policy
- log.retention.hours
- log.cleanup.interval.mins
- log.cleaner.min.compaction.lag.ms
- log.cleaner.delete.retention.ms

After changing the settings using the ADCM interface, restart the Kafka service. To do this, apply the Restart action by clicking ⋮ in the **Actions** column.

Also, the necessary broker parameter can be set in the configuration file /usr/lib/kafka/config/server.properties.

Configure a topic

Individual topic configuration is performed via the command line. Use the `kafka-topics.sh` script with the `--config` option. This option overrides the configuration parameters when a new topic is created or for an existing topic.

An example of log cleanup configuration for an existing topic via the command line

```
$ /usr/lib/kafka/bin/kafka-topics.sh --create --topic new-topic1 --config cleanup.policy=delete --config retention.ms=10000 --config retention.bytes=120000 --bootstrap-server localhost:9092
```

NOTE

For information about the basic principles of working with Kafka, refer to the Quick start with Kafka article.

Message delivery guarantees in Kafka

Olga Semiree

Contents

To Table of Contents

At most once
At least once
Exactly once
- Idempotent producer
- Transactions
- Exactly once semantics

This article describes message delivery guarantees when using the delivery semantics that exist in Kafka.

The main message delivery problems that arise when working with Kafka:

- loss of messages due to various failures (producer, consumer, messaging system, network);
- duplication of messages.

Kafka components that participate in [message writing and sequence reading] can be configured to provide the following delivery semantics:

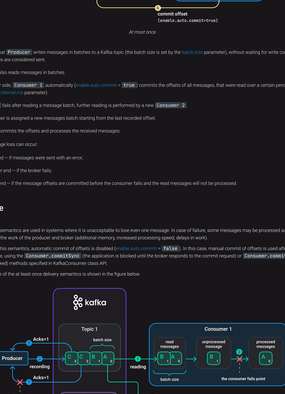
- *At most once* — messages will be processed once or not processed at all (lost).
- *At least once* — messages will be processed at least once.
- *Exactly once* — every message will be processed once and only once.

When using different delivery semantics, system performance may be affected. A stronger guarantee may reduce performance, while a weaker guarantee will increase performance.

At most once

At most once delivery semantics is usually used in systems where it is preferable to lose rather than deliver a message twice. Has the lowest "overhead" (additional memory, increased processing speed, delays in operation).

The operating principle of the at most once delivery semantics is shown in the figure below.



At most once

1. Let us assume that `Producer` writes messages in batches to a Kafka topic (the batch size is set by the `batch.size` parameter), without waiting for write confirmation (`acks = 0`). All messages are considered sent.
2. `Consumer 1` also reads messages in batches.
3. On the consumer side, `Consumer 1` automatically (`enable.auto.commit = true`) commits the offsets of all messages, that were read over a certain period of time (defined by the `auto.commit.interval.ms` parameter).
4. If `Consumer 1` fails after reading a message batch, further reading is performed by a new `Consumer 2`.
5. The new consumer is assigned a new messages batch starting from the last recorded offset.
6. `Consumer 2` commits the offsets and processes the received messages.

In this scheme, message loss can occur:

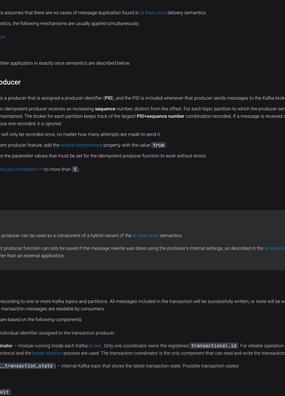
- on the producer end — if messages were sent with an error;
- on the Kafka broker end — if the broker fails;
- on the consumer end — if the message offsets are committed before the consumer fails and the read messages will not be processed.

At least once

At least once delivery semantics are used in systems where it is unacceptable to lose even one message. In case of failure, some messages may be processed additionally. It has possible "overhead" in the work of the producer and broker (additional memory, increased processing speed, delays in work).

As a rule, when using this semantics, automatic commit of offsets is disabled (`enable.auto.commit = false`). In this case, manual commit of offsets is used after processing messages, for example, using the `Consumer.commitSync()` (the application is blocked until the broker responds to the commit request) or `Consumer.commitAsync()` (the application is not blocked) methods specified in KafkaConsumer class API.

The operating principle of the at least once delivery semantics is shown in the figure below.



At least once

Description of semantics and possible problems on the end of the producer or broker:

1. Let us assume that `Producer` writes a message `C` with offset `2` to the Kafka topic and does not receive confirmation of the write from the partition leader (`acks = 1`).
2. `Producer` retries sending the message until the values of `retries` and `delivery.timeout.ms` parameters expire.
3. If the write is successful, `Producer` receives confirmation from the broker.

In this scheme, duplication of `C` messages can occur if a broker or producer fail after writing a message to a topic. This results in two identical messages written at different offsets. If the producer writes messages in batches and tries to write again, multiple messages may be duplicated during a single write iteration.

Description of semantics and possible problems on the consumer end:

1. `Consumer 1` reads recorded messages `A` and `B` from Kafka.
2. Let us assume that `Consumer 1` fails after processing `A` and before processing `B`, and they are in the same batches. Then, depending on the logic of the consumer application, the commit of offset `A` may not be executed.
3. Since offsets `A` and `B` are not committed, the new consumer `Consumer 2` reads the same messages.
4. Upon successful message processing, `Consumer 2` commits offsets `A` and `B`.

In this scheme, duplicate processing of `A` messages can occur if a consumer fails and the offset commit fails, after processing the message.

Exactly once

Exactly once semantics assumes that there are no cases of message duplication found in at least once delivery semantics.

For exactly once semantics, the following mechanisms are usually applied simultaneously:

- Idempotent producer
- Transactions

The mechanisms and their application in exactly once semantics are described below.

Idempotent producer

Idempotent producer is a producer that is assigned a producer identifier (**PID**), and the PID is included whenever that producer sends messages to the Kafka broker.

Every message from an idempotent producer receives an increasing **sequence** number, distinct from the offset. For each topic partition to which the producer sends messages, a separate sequence is maintained. The broker for each partition keeps track of the largest **PID+sequence number** combination recorded. If a message is received with a lower sequence number than the previous one recorded, it is ignored.

This way the message will only be recorded once, no matter how many attempts are made to send it.

To enable the idempotent producer feature, add the `enable.idempotence` property with the value `true`.

The following describes the parameter values that must be set for the idempotent producer function to work without errors:

- `max.in.flight.requests.per.connection` — no more than `5`;
- `retries` > `0`;
- `acks` = `all`.

> **NOTE**
>
> - An idempotent producer can be used as a component of a hybrid variant of the at least once semantics.
> - The idempotent producer function can only be saved if the message rewrite was done using the producer's internal settings, as described in the at least once semantics, rather than an external application.

Transactions

Transaction – atomic recording to one or more Kafka topics and partitions. All messages included in the transaction will be successfully written, or none will be written. If a transaction is aborted, none of the transaction messages are readable by consumers.

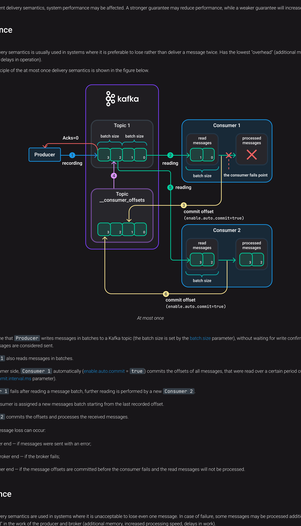
Transactions in Kafka are based on the following components:

- transactional id – individual identifier assigned to the transaction producer.
- **Transaction coordinator** – module running inside each Kafka broker. Only one coordinator owns the registered `transactional.id`. For reliable operation of the coordinator, the Kafka replication protocol and the Kafka [election] process are used. The transaction coordinator is the only component that can read and write the transaction log.
- **Transaction log** – `__transaction_state` – internal Kafka topic that stores the latest transaction state. Possible transaction states:
  - `ongoing`,
  - `prepare_commit`,
  - `committed`.
  Each transaction coordinator manages some partitions in the transaction log and is a leader for them.
- **Epoch** – piece of metadata that stores the number of connections of each `transactional.id`.
- **Zombie instance** – producer instance that started recording messages and failed without completing the recording.

Configuring and working within the transaction occurs using the API class `KafkaProducer` and other APIs related to the operation of transactions.

Exactly once semantics

The main operating principle of exactly once semantics in Kafka is shown in the figure below.



Exactly once

Transaction recording sequence

1. The transaction coordinator is searched using the FindCoordinatorRequest request. The producer connects to any known broker in the cluster to find out the location of its coordinator.
2. Registration of the producer with the transaction coordinator is performed using a synchronous InitPidRequest request. In this case, the producer indicates its `transactional.id`.
   This request performs the following functions:
   - Return the same PID for one `transactional.id` to future instances of the producer.
   - Increment and return the PID epoch so that any previous producers' "zombie instance" is blocked and cannot continue its transaction.
   - Remove any transaction left unfinished by a previous instance of the producer.
   - Create an entry `init` in the `__transaction_state` log for the given `transactional.id` if this is the first transaction for the producer.
3. Starting a transaction is done by calling the `producer.beginTransaction` method. Once a transaction is started, all recorded messages will be part of the transaction. The `__transaction_state` log for this `transactional.id` changes the transaction state to `ongoing`.
4. Cycle **consume** → **processing** → **produce** messages:
   - Register a new partition as part of a transaction using the AddPartitionsToTxnRequest request.
   - Write messages (including PID, epoch, and sequence number) to the user's partition using the `producer.send` method.
   - Send offsets of read messages within a transaction with `groupId` identifier to the transaction coordinator using the `producer.sendOffsetsToTransaction` method to register the addition of this partition to the transaction log.
   - Register offsets in the `__consumer_offsets` topic using the TxnOffsetCommitRequest request from producer to consumer coordinator (including PID and epoch). Recording offsets in the `__consumer_offsets` topic also becomes part of this transaction. The consumer coordinator verifies that the producer is not a "zombie instance". Recorded offsets are not visible to consumers until the transaction is confirmed.
5. Confirming a transaction using the `commitTransaction` method, which performs the following functions:
   - Write the `prepare_commit` message to the transaction log.
   - Write `COMMIT` markers to user topics, including the `__consumer_offsets` topic.
   - Write the `committed` message to the transaction log.
   - Open access for consumers to messages recorded in a topic.

Thus, the offsets of all messages read in one transaction were written only after they were processed.

Read transactional messages

The reading of transactional messages by the consumer is controlled by the `isolation.level` parameter.

- If the parameter value is set to `read_committed`, the consumer buffers messages that have a PID until it reads a message with a `COMMIT` marker. After this, the messages will be delivered. Messages are read only up to the last stable offset, which is less than the offset of the first open transaction. `read_committed` consumers will not be able to read topics in their entirety while there are pending transactions.
- If the parameter value is set to `read_uncommitted`, all messages are visible to consumers, even if they were part of an aborted transaction.

> Guarantees of exactly once semantics can be provided if the application in which the consumer reads transactional messages from a Kafka topic then passes them (after processing) to another Kafka topic (for example, Kafka Streams applications). This is because the consumer offset is stored as a message in the topic, so offsets when reading messages can be written in the same transaction as the user topic messages. If the transaction is aborted, the last offset returns to the old value, and the created messages in the user topics will not be visible to consumers.

Possible "overhead" of exactly once semantics:

- in the work of the producer and broker – the more messages in the transaction, the less resource loss (additional memory, increased processing speed, delays in work);
- in the work of the consumer – reading a large transaction causes blocking of the read partition for consumers with the `read_committed` setting and loss of resources (memory for buffering, increased processing speed, delays in operation).

Replication in Kafka

Olga Semme

Contents

To Table of Contents

- Replication factor in Kafka
- Replica synchronization
- Recommendations for configuring replication settings
  - Replication factor
  - Number of topic partitions
  - Synchronized replicas
- Additional features
- Configure replication
  - Configure a cluster
  - Configure a topic

This article describes the principles of replication in Kafka and how to configure it.

NOTE

- Main storage concepts and core components of Kafka are described in the Storage concepts in Kafka article.
- The principles and algorithms of interaction between Kafka and ZooKeeper are described in the Leader election in ZooKeeper article.

Replication is a mechanism that creates and distributes exact copies of each topic partition on brokers in Kafka.

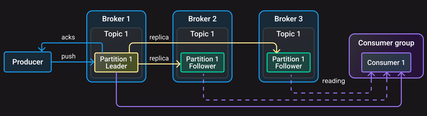
Replication ensures that messages are available in the event of failures or maintenance.

Replication factor in Kafka

Replication factor is a parameter that determines the number of copies (**replicas**) of each partition.

Each partition replica is recorded on a separate **follower** broker. There cannot be more replicas of each partition than there are brokers.

The figure below shows how replicas of one partition are distributed on brokers, messages are written by the producer, and messages are read by the consumer.



Replicas distributed on brokers

Message writing operations are performed through the **leader** broker of the partition. Message reading operations can be performed either through the leader (default) or through followers (since Kafka 2.4).

Leader election for each partition is carried out using ZooKeeper in accordance with the consensus algorithm protocol ZAB (ZooKeeper Atomic BroadCast).

Partition leaders are evenly distributed among brokers. Each broker can be a leader for one partition and a follower for another.

The replication factor in Kafka for a cluster is determined by the default.replication.factor parameter. All topics created in the cluster will be created with the default replication factor specified for the cluster.

It can also be installed individually for a topic:

- when creating a topic via the command line;
- when creating a topic via Kafka Streams applications (available starting from Kafka 2.4);
- when creating a remote topic via the Mirror Maker 2 mechanism.

If the topic configuration specifies a replication factor value greater than the number of brokers in the cluster, the topic will not be created.

The replication factor for the created topic cannot be overridden. If the replication factor value is changed for the entire cluster, only new topics will be created with the new replication factor.

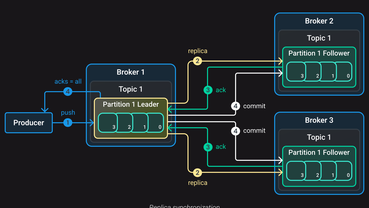
Replica synchronization

Replica synchronization is relevant when the replication factor value is greater than `1` and the `all` value is assigned to the acks producer parameter. This combination of parameters is a step towards exactly once message delivery guarantee.

The interaction between leader and followers is based on messaging model used in ZooKeeper.

Follower logs tend to be a copy of the leader's log, having the same offsets and messages in the same order. A replica that contains all messages written to the leader's log is called a **synchronized replica (In-Sync Replica, ISR)**.

The figure below shows how replica synchronization is achieved.



Replica synchronization

The main parameter that controls replica synchronization is replica.lag.time.max.ms, the time period after which the leader accepts decision that the replica wrote (if a write acknowledgment occurred) or did not write (a write acknowledgment did not occur) a message.

The success criterion for writing a message as a whole is determined by the min.insync.replicas parameter – the minimum number of replicas that must confirm the write (can also be set and changed for each topic individually using the min.insync.replicas) parameter. Only when this parameter is reached, the leader transmits data about the recorded partition (metadata) and about the created ISRs to the controller.

After receiving the message, the leader offers a recording of the message to the replicas and waits for confirmation from them. Having received confirmation from the required number of replicas (the `min.insync.replicas` parameter), the leader sends a signal to commit messages to the replicas and a confirmation to the producer. The message becomes available only after that.

Recommendations for configuring replication settings

Replication factor

There is a general recommendation to set the replication factor to `3`.

In fact, the exact value of the parameter depends on the requirements and resources of your system. When choosing a value, pay attention to the following:

- High replication factor provides better system stability.
- High replication factor can cause higher latency experienced by producers as data needs to be replicated to all replica brokers before an acknowledgment is returned.
- High replication factor requires large disk space due to the large number of copies of data on disks for high availability (HA).

Number of topic partitions

When setting the replication factor, you should remember that the operation of the system is also affected by the num.partitions parameter – the number of partitions of all topics in the cluster (can be configured via the `--partitions` option for an individual topic). The replication factor and the number of partitions are parameters that work in conjunction, and the performance and durability of the system depend on their settings.

When choosing the value of the `num.partitions` parameters, it is worth paying attention to what capabilities and limitations a large number of partitions causes:

- Provides the ability to combine more consumers into a group for scaling.
- Allows you to use existing cluster brokers more efficiently.
- Increases the load on ZooKeeper – leader election, support for a list of synchronized replicas.
- Increases the number of files created and opened on brokers.

Synchronized replicas

The `min.insync.replicas` parameter in combination with the acks producer parameter, which is set to `all`, affects the exactly once delivery guarantee of messages.

It is recommended to set the `min.insync.replicas` parameter less than the replication factor, otherwise even if one broker fails, the message record will not be confirmed due to insufficient synchronized replicas.

Standard recommended scenario for HA guarantees:

- default.replication.factor = `3`
- min.insync.replicas = `2`
- acks = `all`

Additional features

Starting with Apache Kafka 2.4, you can configure consumers to read from nearby synchronized replicas.

To select the closest synchronized replicas, use the RackAwareReplicaSelector plugin. For it to work the following parameters must be configured:

- broker.rack – indication of the broker's affiliation with a specific rack. Each broker must be assigned a rack. For the plugin to work effectively, it is recommended to configure the same number of brokers per rack – this way the replicas will be evenly distributed among the brokers.
- client.rack – setting up a rack ID for connecting consumers.

The `RackAwareReplicaSelector` plugin attempts to match the consumer's `client.rack` to the available `broker.racks`. It then selects a replica that has the same rack ID as the consumer.

Configure replication

Configure a cluster

After adding and installing the ***Kafka*** service as part of ADS cluster, you can configure the replication factor and the number of partitions on the configuration page of the ***Kafka*** service via ADCM. To do this, enable the ***Show advanced*** switch, expand the ***server.properties*** node in the configuration settings tree, and enter a new value for the parameters.

To change the default value for the min.insync.replicas parameter and other broker parameters that are not available in the ADCM interface, use the ***Add key,value*** field: select ***Add property*** and enter the parameter name and its value.

After changing the settings using the ADCM interface, restart the ***Kafka*** service. To do this, apply the Restart action by clicking in the ***Actions*** column.

After new values and parameters are filled in, all changes are reflected in the configuration file */usr/lib/kafka/config/server.properties*.

Configure a topic

Configuration of the replication factor for an individual topic is performed via the command line. Use the *kafka-topics.sh* script via the `--replication-factor` option when a new topic is created.

The value of the `min.insync.replicas` parameter can be changed via the `--config` option, which overrides the configuration parameters for an existing topic or sets the configuration parameters when a new topic is created.

An example of replication factor and min.insync.replicas parameters configuration for an existing topic via the command line

```
$ /usr/lib/kafka/bin/kafka-topics.sh --create --topic new-topic --replication-factor 3 --config min.insync.replicas=2 --
bootstrap-server localhost:9092
```

NOTE

For information about the basic principles of working with Kafka, refer to the Quick start with Kafka article.

KRaft overview

Olga Semme

Contents

To Table of Contents

- KRaft benefits
- Controller quorum
- Raft algorithm
  - Overview
  - Vote
  - Message replication
  - KRaft
- Metadata management
  - Write metadata
  - Metadata replication
- KRaft in ADS

Kafka Raft (KRaft) is a protocol based on the Raft consensus algorithm that manages Kafka cluster metadata. KRaft organizes the storage of information about the cluster configuration, ensures synchronization of Kafka brokers during data replication without using leader election and metadata storage mechanisms provided by ZooKeeper.

NOTE

The ability to enable KRaft mode has been available since Kafka version 3.6.2.

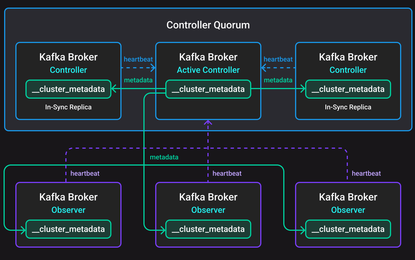
KRaft benefits

The main characteristic of KRaft is the absence of an intermediary service, which entails the following advantages:

- Fast and efficient metadata storage and distribution – updates are sent directly to the controller quorum, eliminating an additional transmission node (ZooKeeper).
- Brokers and controllers can store metadata locally in their cache and access it without using a connection to an external service.
- Reducing delays when broker stops their work.
- It is expected that the KRaft protocol will significantly increase the limit on the number of partitions for one node and an entire cluster.

Controller quorum

The main concept of KRaft is based on the creation of a **controller quorum** – a set of specialized brokers that are responsible for storing and timely updating cluster metadata.



Controller quorum in KRaft

For controller assignment in Kafka the process.roles parameter with the `controller` value is used.

The controller quorum only serves the internal `__cluster_metadata` topic with Kafka metadata. The topic contains one partition, which records all information about the current state of Kafka brokers, topics, and partitions. The leader of the partition – **active controller** – is selected based on the Raft consensus algorithm by voting for the **leader epoch**. The active controller is responsible for writing metadata changes to the `__cluster_metadata` topic.

Controllers that are not the leader store an exact copy of the topic with metadata as in-sync replicas (ISR) and have the right to vote for the new leader epoch. As a result of the elections, each of these voters can become a leader.

Brokers whose process.roles parameter is set to `broker` cannot vote for the leader epoch, they are **observers**. Observers work like regular Kafka brokers, but they store a copy of the `__cluster_metadata` topic, each time rewriting the cluster metadata state into their own cache whenever the topic changes.

The active controller also monitors the state of all controllers and brokers using timeouts and heartbeat messages sent by brokers. When a broker stops, the leader redistributes partitions to other brokers and updates information about them in the metadata topic.

To coordinate replicas, the **high water mark (HWM)** marker is used — the largest offset in the log that was written to all ISRs. Followers cache its value and update it whenever the leader's log changes, thus maintaining a consistent state of the cluster. When a leader changes, all follower partitions truncate their logs to the HWM recorded by the new leader, and all messages above the HWM become unreadable. After the log is truncated, the follower can continue to replicate any new messages from the leader. Once all new messages are replicated to a majority of nodes, the HWM is increased.

In addition to the `process.roles` parameter, the following parameters must be defined for Kafka cluster brokers configured to work in KRaft mode:

- controller.quorum.voters.roles – addresses of nodes participating in the controller quorum.
- node.id – a node ID, must be a unique integer and is assigned during installation/extension. Cannot be the same as the ID of any other broker or controller in the cluster.
- controller.listener.names – a comma-separated list of listener names used by the controller. When communicating with a controller quorum, a broker in KRaft mode will always use the first listener in this list.
- metadata.log.dir – a directory for storing cluster metadata. The directory name must not be the same as any broker directory; the default is */kafka-meta*.

Raft algorithm

Overview

The Raft and ZAB (used in ZooKeeper) consensus algorithms have the following conceptual similarities:

- the only leader responsible for data processing at any time;
- separate processes for leader elections and log replication;
- general focus on ensuring data consistency in distributed systems.

The Raft architecture has a simpler implementation, in which the following roles are allocated for the operation of nodes:

- **Leader** of the cluster, which processes client requests and controls data replication.
- **Candidate** trying to become a leader.
- **Follower** recording messages and processing requests from the leader and candidates.

Vote

The leader is chosen randomly on the first run, and the remaining nodes start as followers.

The leader monitors the health of followers using timeouts and periodic heartbeats, and also sends its heartbeats with or without messages (if any).

If a follower does not receive heartbeats from the leader during the timeout, it becomes a leader candidate. Each candidate sends a request for leadership to the others and accepts the same requests from the others. The candidate who receives the most votes becomes the leader. The leader is selected for a period, the called **term**. The term is identified as **epoch** – a number that i incremented with each new leader.

Message replication

The leader sends a request to all followers to add a new record and considers the transaction successful only after receiving confirmation of the message record from the majority of nodes.

Messages are always added sequentially.

If an external client connects to the cluster through a follower, the request is still forwarded to the leader.

KRaft

Additionally, the KRaft protocol introduces a method for storing metadata based on the Kafka log replication protocol, where controller quorum is used. The basic principle is that each cluster broker strives to store a replica of the metadata topic and does not require a connection to ZooKeeper to obtain the state of the cluster (topic partition leaders, new brokers, etc.), and therefore does not require latency and a significant amount of memory, since the process of storing metadata in the log is part of the operation of the Kafka algorithms.

Metadata management

Write metadata

Each new broker or controller that joins the cluster requests a copy of the internal topic __cluster_metadata and creates a replica of it, simultaneously caching the topic data, i.e. saves data about the complete state of the cluster.

If the broker starts after a failure, it reads the most recent changes available in its metadata cache. After that, it queries the active controller for events that have occurred since the last update in `__cluster_metadata` and in response receives the latest changes in the form of snapshots of the cluster state.

Metadata replication

For Kafka running in KRaft mode, HWM is tracked and taken into account by the leader for the total number of metadata topic replicas (not just for ISR replicas, as in regular Kafka).

The HWM cannot be increased until all messages written by the leader have been stored by a majority of the replicas. If there is a change of leader, the epoch increases. To reconcile their logs with the leader, followers send a request containing the last offset and the last epoch during which messages were recorded.

Based on such requests, the leader monitors the performance of brokers and in response sends information about the new epoch and the last offset. Next, the replicas catch up with the leader or cut off their log to the HWM transmitted by the leader. After a majority of brokers confirm the last offset entry, the HWM is incremented. After the log is truncated, the follower can continue to replicate any new messages from the leader.

KRaft in ADS

Starting from ADS version 3.6.2.2.b1, installation of the Kafka service in KRaft mode is available via the ADCM interface. This method automatically sets the necessary parameters for brokers.

CAUTION

- The KRaft protocol for Kafka is in technical preview and is not recommended for use in a production environment. Arenadata encourages you to explore this feature in non-production environments and provide feedback on your experience as appropriate.
- For clusters where Kafka service is used together with ZooKeeper, migration to KRaft mode can be performed.
- There is no possibility to migrate from KRaft to ZooKeeper.
- For ADS cluster where installation of Kafka service in KRaft mode is planned, ZooKeeper service installation is not required.

Kafka Tiered Storage overview

Olga Semme

Contents

To Table of Contents

Benefits of Tiered Storage

Tiered Storage architecture

Tiered Storage limitations

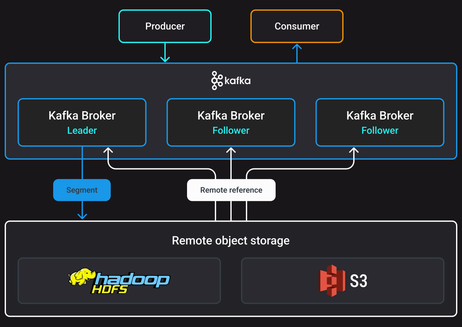
Kafka Tiered Storage in ADS

Tiered Storage configuration

Kafka Tiered Storage is an option that enables a tiered way of the data storage in a Kafka cluster, available since Kafka version 3.6.0.

After the option activation in Kafka, two levels are set for storage:

- **Local level** — local disks of Kafka brokers, storing log segments in the same way as regular Kafka brokers.
- **Remote level** — file storage systems (like HDFS or S3) on which closed log segments are stored.



Kafka Tiered Storage

The partition leader, which uses threads managed via special Remote Manager components, copies all closed segments to the storage (the period after which closed segments are sent to the external storage is determined using the remote.log.manager.task.interval.ms broker parameter). Next, the leader saves and publishes a link containing indexes and metadata of the **remote segment** — a closed segment stored in the storage (leader epoch, producer, offset, and data about the storage of the segment in the storage). Closed segments are removed from the broker when the local retention time expires (local.retention.ms) or when the maximum local storage size is reached (local.retention.bytes).

Followers, seeking to become ISR, replicate data written to the leader, while caching links to remote segments.

Writing and deleting messages are idempotent processes and are performed as transactions.

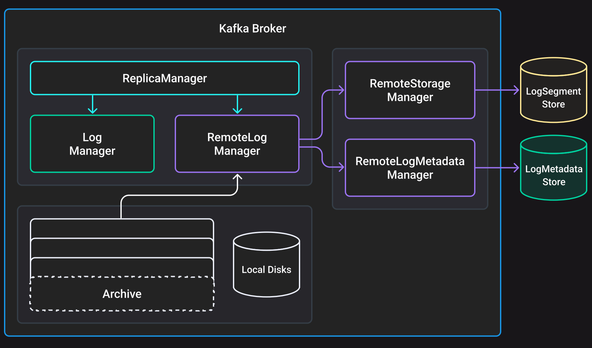
The interaction of producers and consumers with Kafka brokers does not change after enabling the Tiered Storage option. Messages are written to active segments located at the local level, and for each consumer request to read a message from the remote level, a separate thread is created, which is served by Remote Manager components, as well as threads for writing segments to storage.

Benefits of Tiered Storage

- Possibility of long-term data storage in external storages.
- Reducing the amount of data on Kafka brokers, and therefore the amount of data that needs to be copied during recovery and rebalancing.
- Simplification of the operation of large Kafka clusters with long-term data storage.

Tiered Storage architecture

After activating the Tiered Storage option, the components responsible for managing remote segments are launched on the Kafka broker.



Tiered Storage architecture

RemoteLogManager is an internal component that runs on each Kafka broker. Through it, the broker interacts with remote segments. RemoteLogManager does not have a public API.

The ReplicaManager persistent component that manages partition replicas calls RemoteLogManager to manage the partitions.

And RemoteLogManager, in turn, passes the command to copy segments to storage or delete them to the RemoteStorageManager component and maintains the corresponding remote segment metadata for RemoteLogMetadataManager.

RemoteStorageManager is an interface that provides the life cycle of remote log segments and indexes.

Remote Log Metadata Manager is an interface that provides the life cycle of remote log segment metadata with strictly consistent semantics. There is a default implementation that uses the internal topic `__remote_log_metadata`. Users can plug in their own implementation if they intend to use a different system to store remote segment metadata.

Tiered Storage limitations

- Once enabled for a specific topic, the option cannot be disabled.
- The option cannot be used for topics in which the cleanup policy is set to compact.
- For an existing topic, it is not possible to migrate segments back to local brokers, increasing the local data storage time.
- Cannot be used with a disk array (JBOD) as a local broker.

Kafka Tiered Storage in ADS

The Tiered Storage option in ADS is based on the Aiven implementation, which adds the following capabilities to the built-in Kafka RemoteStorageManager:

- Segment compression using the **chunking** mechanism — dividing the original segments into parts and compressing them before sending them to storage.
- Segment encryption with key rotation support.
- Optimization of the number of accesses to remote storage.
- Abstraction of remote storage in the form of a StorageBackend — a container of byte arrays based on key/value messages.

Additionally, ADS implements support for HDFS as remote storage, for which StorageBackend has been developed based on the Aiven solution, providing:

- Support for configuration XML files.
- Support for authentication via Kerberos.
- HDFS client metrics export.

Tiered Storage configuration

The Tiered Storage option is configured in the */etc/kafka/conf/server.properties* configuration file and consists of several blocks of parameters:

- Enabling the Tiered Storage option using broker remote.log.storage.system.enable or topic remote.storage.enable parameters.
- Configuring Remote Manager components.
- Setting up the storage backend.
- Configuring the chunking cache — if supported in the storage backend.

NOTE

Configuring Tiered Storage in ADS using the ADCM interface is described in the Configure and use Kafka Tiered Storage article.

kTLS overview

Olga Semme

Contents

To Table of Contents

Kafka with TLS

Kafka and kTLS

Kernel TLS (kTLS) — implementation of the TLS protocol based on the transfer of some operations, including encryption and decryption, to the Linux kernel level, which reduces the load on the processor and improves performance.

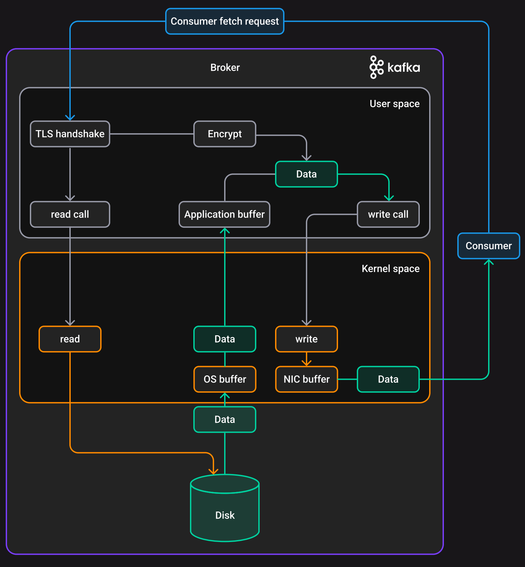
Kafka supports kTLS starting with ADS 3.9.0.1.b1.

The following describes how a Kafka consumer and broker interact when using traditional TLS and when using kTLS.

Key concepts used to describe:

- **Transport layer security** (**TLS**) is a cryptographic protocol that provides secure data transfer over a network when exchanging data between two applications. It is an improved version of the secure sockets layer (SSL) protocol.
- **Kernel space** is a memory area reserved for the operating system kernel, its extensions, and device drivers.
- **User space** is the address space of the operating system's virtual memory intended for user programs, separated from the kernel space to ensure system security and stability.
- **TLS handshake** is the interaction between a client and a server when connecting. During the handshake, the TLS version used, the ciphers used, the keys are exchanged and the signatures of the SSL certification center are verified, and session keys are generated for using symmetric encryption.
- **Network interface controller** (**NIC**) is a hardware device for interacting with a network.
- **NIC buffer** is a memory area used by the network card for temporary storage of data during reception and transmission.
- **Socket** is an abstraction in Linux for exchanging data between processes or devices. In the Linux kernel, sockets are implemented as data structures that allow the system to handle network operations without directly accessing the hardware details.
- **File descriptor** is a natural number (identifier) assigned to a specific input/output stream.

Kafka with TLS



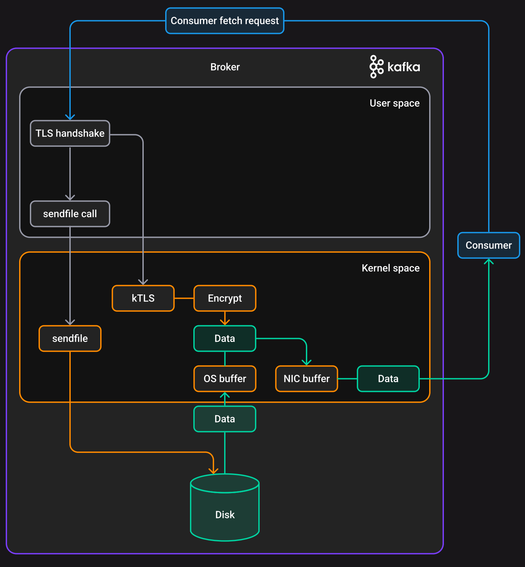
Kafka and traditional TLS

When a connection is established between a client (consumer) and a Kafka broker after a TLS handshake, the user space application makes a `read` system call. The file descriptor, pointer to the application buffer, and other data to be read are then passed to the kernel. The data is copied through the kernel buffer to the application buffer, encrypted (in the user space), and, using the `write` system call, written to the network card buffer (in the kernel space) file descriptor, data buffer, and number of bytes to write, and sent to the consumer.

Disadvantages:

- Each data packet is copied twice between the kernel and user space, causing delays due to additional operations and switches between the kernel and user space.
- High CPU load due to encryption being performed in software mode.

Kafka and kTLS



Kafka and kTLS

When a connection is established between a client (consumer) and a Kafka broker, after a TLS handshake (which is still done in user space), the application makes a `sendfile` system call, which copies the data between two file descriptors, i.e. the data is copied directly from the kernel cache to the NIC buffer.

To perform data encryption in the kernel space, a special TLS socket file descriptor (KTLS) is created, where information about session keys, identifiers, and data encryption is extracted from the TLS library. Data is encrypted directly in the kernel before being sent to the network.

Encrypted data is decrypted in the kernel before being passed to user space, if so requested by the application.

Advantages:

- Enforce the **zero-copy** principle — the `sendfile` system call copies data between one file descriptor and another one immediately within the kernel, instead of a combination of `read` and `write` calls that perform data transfer to and from user space. This reduces the load on the processor and improves performance and efficiency, which is especially important when transferring large amounts of data.
- Reduced CPU load by offloading encryption/decryption operations to the kernel.
- Improved throughput and reduced latency when transmitting data over secure channels.
- Transparent integration with existing TLS libraries (e.g. OpenSSL) via the kernel API.

IMPORTANT

To support kTLS, you need an operating system kernel version 5.9 or higher. kTLS mode is enabled by default. However, if your kernel version is lower than required, you will be rolled back to the default mode.

Cruise Control overview

Olga Semmo

Contents

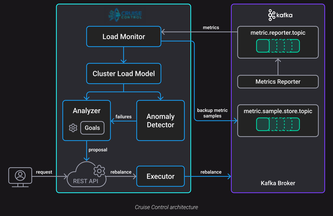
To Table of Contents

- Architecture
- Load Monitor
- Analyzer
- Anomaly Detector
- Executor

Architecture

Cruise Control is a tool designed for managing high-load Kafka clusters or clusters with a large number of hosts, allowing for the assessment of broker states as well as performing load balancing.

Working with Cruise Control in ADS is available starting with 3.9.0.1.b1.



Cruise Control architecture

The **Metrics Reporter** interface collects metrics from the Kafka broker via JMX and stores them in a topic defined by the `metric.reporter.topic` parameter.

Load Monitor collects metrics from the topic and creates **Cluster Load Model**, which **Analyzer** uses to generate a **proposal** for optimization based on the **goals** set by the user, while **Anomaly Detector** identifies anomalies that can be corrected during the cluster's self-healing.

Cluster Load Model is a software model reflecting the current distribution of cluster replicas with granularity of load data for various resources.

The user can obtain a generated proposal by making a `GET` request to the Cruise Control REST API, and using a `POST` request to initiate the **rebalance** of the Kafka cluster. Before launching the rebalance, the user can edit the goals and/or configure the cluster's self-healing if necessary.

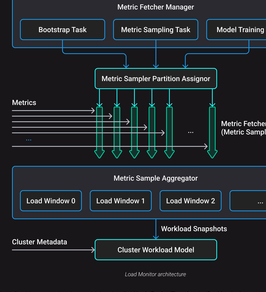
Executor is responsible for implementing the optimization proposals received from the analyzer.

The load metrics processed by the monitor are also sent to the connected storage, where they are saved for cluster recovery. In ADS, these are special topics in the Kafka cluster (defined by the `m.metric.sample.store.topic` parameters). When Cruise Control is restarted, it will load metrics from this storage to populate the load monitor.

Below are the detailed descriptions of the Cruise Control components.

Load Monitor

To create Cluster Load Model within Load Monitor, an up-to-date **Metrics Sampler** is used in conjunction with the cluster's metadata.



Load Monitor architecture

Metrics for all partitions of the Kafka cluster are assigned to special threads called **Metric Fetcher Threads** using **Partition Assignor**, which provides:

- the same number of partition metrics for all threads;
- assignment of partition metrics of one topic to one thread.

Threads are part of various **Sampling Tasks**.

There are three types of sampling tasks:

- Metric Sampling Task
- Bootstrap Task
- Linear Model Training Task

Tasks are coordinated using the **Metric Fetcher Manager** module. It creates a certain number of threads to execute each task.

All metric samplers are organized using the **Metric Sample Aggregator** module. Each metric sampler is placed in **load window** according to the timestamp of the sample.

Load Window is the aggregated information about metrics for a specified time interval (time window). Such an interval (window) is defined by the parameter `partition.metrics.window.ms` `broker.metrics.window.ms` for Kafka broker metrics).

The number of supported windows is determined by the `num.partition.metrics.windows` parameter `num.broker.metrics.window` for Kafka broker metrics).

For example, if the time window is 1 hour and the user has configured storage for 168 windows, Cruise Control will store hourly load information for 7 days.

The number of samples in a single load window is determined by the `metric.sampling.interval.ms` parameter.

For example, if a user specifies a time window of 1 hour and a sampling interval of 5 minutes, each window should contain 12 metric samples.

Using the parameter `min.samples.per.partition.metrics.window`, the user can define the minimum number of samples in each window at which these samples will be included in Cluster Load Model.

When requested, Metric Sample Aggregator returns all aggregated data of the load window — **workload snapshot** in the Load Monitor as part of Cluster Load Model.

Analyzer

Analyzer generates proposals based on user-defined optimization goals and Cluster Load Model obtained from Load Monitor.

The proposal is an optimization plan for rebalancing the Kafka cluster. It describes a series of actions aimed at improving resource utilization in the cluster and balancing the load among brokers, such as redistributing partitions and leader elections.

Cruise Control allows you to set **hard** and **soft** goals.

A hard goal is a goal that must be achieved. Such goals should be placed in the `hard.goals` list.

A soft goal is a goal that may not be achieved if it allows all hard goals to be reached.

There is a set of goals that the user can add and remove from the `goals` list to get the expected balancing result. Users can create their own goals and add them to Cruise Control.

Below is a description of all existing goals.

Goals description

Goal	Description
BrokerSetAwareGoal	BrokerSet — a subset of brokers in the cluster. This goal limits the movement of replicas within the BrokerSet
RackAwareGoal	The goal ensures that all replicas of each partition are assigned with regard to the rack
RackAwareDistributionGoal	The goal allows placing multiple replicas of a partition in one rack, providing that the replicas of each partition are perfectly evenly distributed across the racks
ReplicaCapacityGoal	When this goal is set, the system will try to ensure that the number of replicas of all brokers in the cluster is lower than the specified value
DiskCapacityGoal	The goal guarantees that the use of disk resources will be below the specified threshold. This goal will be achieved if the `rebalance_disk` parameter in the balancing request is set to `true`
NetworkInboundCapacityGoal	The goal ensures that the volume of the incoming network used by each broker is below the established threshold. The volume is set using the `network.inbound.capacity.threshold` parameter in *cruisecontrol.properties*
NetworkOutboundCapacityGoal	The goal ensures that the volume of the outgoing network used by each broker is below the established threshold. The volume is set using the parameter `network.outbound.capacity.threshold` in *cruisecontrol.properties*
CpuCapacityGoal	The goal ensures that CPU load on each broker is below the set threshold; it is set using the `cpu.capacity.threshold` parameter in *cruisecontrol.properties*
ReplicaDistributionGoal	When this goal is set, the system will try to ensure that all brokers in the cluster have the same number of replicas
PotentialNwOutGoal	The goal ensures that the potential output capacity of the network (when all replicas become leaders) on each broker does not exceed the outgoing bandwidth of the broker's network
DiskUsageDistributionGoal	When this goal is set, the system will try to ensure that the usage spread among all disks within a broker is kept within a certain range. This goal will be achieved if the `rebalance_disk` parameter in the rebalance request is set to `true`
NetworkInboundUsageDistributionGoal	When this goal is set, the system will try to ensure that the distribution of incoming network traffic among all brokers is even
NetworkOutboundUsageDistributionGoal	When this goal is set, the system will try to ensure that the distribution of outgoing network traffic among all brokers is even
CpuUsageDistributionGoal	When setting this goal, the system will try to ensure that the CPU utilization variance among all brokers is uniform
TopicReplicaDistributionGoal	A goal that ensures the even distribution of replicas of a single topic across the entire cluster
LeaderReplicaDistributionGoal	The goal that ensures an even distribution of leader replicas across all brokers in the cluster
LeaderBytesInDistributionGoal	When this goal is set, the system will try to ensure that the incoming network traffic to the leaders of the partitions in the cluster is balanced
KafkaAssignerDiskUsageDistributionGoal	The goal that ensures the assignment of all replicas of each partition taking into account the rack
KafkaAssignerEvenRackAwareGoal	When setting this goal, the system will attempt to ensure an equal number of replicas on all brokers in the cluster
PreferredLeaderElectionGoal	When this goal is set, the system will try to make the first response in the list of responses the leading response for all partitions of the topic

Anomaly Detector

Anomaly Detector detects various anomalies and, after self-healing is enabled (using the `self.healing.enabled` parameter), initiates an attempt to automatically fix certain types of failures. The goals used for self-healing are specified in the `self.healing.goals` list.

The anomalies are described below.

Anomaly description

Anomaly	Description
Broker failures	A failure of a non-empty broker or exit from the cluster. This leads to the emergence of offline replicas/under-replicated partitions. In this case, Cruise Control sends a notification, and if self-healing is enabled for this type of anomaly, Cruise Control initiates an operation to relocate all offline replicas to other functioning brokers in the cluster. Since this can also occur during normal cluster failures, the anomaly detector provides a configurable grace period before before the notification is triggered and the cluster is restored. The anomaly detector provides a customizable grace period before the notification is triggered and the cluster is restored.
Goal violations	The optimization goal is violated. In this case, Cruise Control will send a notification, and if self-healing is enabled for this type of anomaly, Cruise Control will proactively try to remedy the goal violation by automatically analyzing the load and implementing optimization proposals
Disk failure	The failure of one of the non-empty disks. Note that this is only related to the Kafka broker operating on JBOD. In this case, Cruise Control will send a notification, and if self-healing is enabled for this type of anomaly, Cruise Control will initiate the operation to move all standalone replicas to other operational brokers in the cluster
Metric anomaly	One of the metrics collected by Cruise Control detects an anomaly (for example, a sharp increase in log cleanup time metric). In this case, Cruise Control sends a notification. Currently, there is no standardized self-recovery operation for metric anomalies, as different metric anomalies require different remediation methods. A user can define their own anomaly and remediation operation by implementing their own MetricAnomaly and MetricAnomalyFinder
Topic anomaly	One or several topics in the cluster violate user-defined properties (for example, some disk partitions are too large). In this case, Cruise Control will send a notification. Currently, there is no standardized self-recovery operation for topics' anomalies, as different anomalies in topics require different correction methods. The user can define their own anomaly and correction operation by implementing their own TopicAnomaly and TopicAnomalyFinder methods

Executor

Executor is responsible for executing optimization proposals received from the analyzer. The executor can be interrupted while executing proposals. The executor ensures that the execution is resource-aware and does not overload any broker. Some parameters can control partition movements during rebalancing, such as `num.concurrent.partition.movements.per.broker` or `max.num.cluster.partition.movements`.

NOTE

For information on the installation, configuration, and use of Cruise Control in ADS, refer to the Cruise Control usage example article.

Kafka Connect overview

Adam Bellemare

To Table of Contents

Contents

- Kafka Connect features
- Kafka Connect architecture
  - Worker
  - Connector
  - Task
  - Converter
  - Transform
  - Handle errors
- Kafka Connect in ADS
  - Connection
  - Kafka
  - Schema Registry
  - ksqlDB
  - Configure Kafka Connect

Kafka Connect features

Kafka Connect is a tool for scalable and reliable data streaming between Apache Kafka and other data systems. Kafka Connect works by creating special connectors that move data to a Kafka or out-since.

Kafka Connect features

The main benefits of Kafka Connect are listed below:

- Data centralization, ensuring that events are ingested into the REST interface
- Simple scalability, changing the number of processes, increasing/ decreasing connections to existing platforms, managing the number of running tasks
- Automatic control of data offsets during connector operations
- Wide selection of existing connector plugins, as well as the ability to create your own to suit to the needs of your system
- Integration of streaming data from platforms

Kafka Connect architecture

The Kafka Connect architecture is described below for the case when data transfer from an external system (source) to Kafka is required. To perform data copying, the source connector is launched in the Kafka Connect cluster with the Kafka Connect worker components and the tasks created by the connector.

Kafka Connect architecture

A Kafka Connect cluster consists of several workers.

To create a connector instance, you can send a request to the REST API of any of the workers. The created connector connects to the data storage, creates the required (specified in the connector configuration) number of tasks and distributes them for execution between the Kafka Connect worker components and the tasks created by the connector.

Kafka Connect can also be launched in the standalone mode, when all the connectors and tasks are executed in one process. In this case, the connector configuration files are used instead of the REST API requests.

The data created by the tasks can be converted to the data passed through Kafka Connect using the single message transform module. The required data transformations are specified in the connector configuration.

As data passes through Kafka Connect, it is serialized using a converter before being written to Kafka. The converter type can be specified for the entire Kafka Connect service as well as for each specific connector.

Worker

Worker is a container that launches connectors and tasks. Each worker is a Java process running inside the JVM.

A worker can function in two modes:

- **Standalone mode**, when a single Kafka Connect operating mode when only one worker is in the cluster, all data about connectors is stored inside Kafka Connect. Offsets mode has limited functionality and also lacks fault tolerance.
- **Distributed mode**, when Kafka Connect operating mode in which several workers are used. To combine several workers into a cluster, use the group.id parameter. In this mode, the REST API of each worker is used to create, configure, and manage connectors. Requests can be sent to any member of the cluster.

In distributed mode, workers use Kafka topics to store data passed through Kafka Connect using the single message transform module. The required data transformations are specified in the connector configuration.

- offset.storage.topic — the name of the topic that stores offsets for data already processed by connectors. This is described in more detail below in the description of the read architecture
- config.storage.topic — the name of the topic that stores information about the states of connectors. To write data to the Kafka Archiver, workers use the Kafka API.
- Management of connectors and tasks.
- Ensuring scalability and automatic fault tolerance for Kafka Connect.
  - when adding workers, tasks are redistributed among all workers.
  - when the number of workers is reduced (managed or in case of failure), tasks are redistributed between the workers.

The Kafka Connect connect.config for the distributed mode are set in the connect-distributed.properties configuration file.

The figure below shows an example of data copying in Kafka Connect and the interaction of the worker with Kafka.

Kafka Connect workers

Connector

A **connector instance** is a logical job that copies messages from an input stream in an output stream, with a Kafka topic (or multiple topics) being the data source or ongoing destination. Depending on what Kafka is in the process, the type of connector is determined:

- **Source connector** — a connector that receives data from the API of the data source system, retrieves data and data schemas and passes it to Kafka Connect in order to create entries in Kafka topics and table updates.
- **Sink connector** — a connector that extracts data from the topics of records in the target system using its API. Such connectors can deliver data from Kafka topics to systems such as Elasticsearch or Hadoop.

The figure below shows the process of creating a connector instance.

Creating a connector instance

Worker creates and launches a connector instance based on a REST request. To create a connector, you need the following:

- **Connector plugin** is a JAR file containing connector.class. Connector plugins do not exist or work messages in Kafka, but rather provide an interface architecture between Kafka and the external system. Kafka Connect supports the use of several connectors at the same time. Connector plugin path
- **Connector class** — connector class based on the SourceConnector or SinkConnector interface, used to create the connector instance itself.
- **Task class** — a task class based on the SourceTask or SinkTask interface is used to create the task instances. The methods used in these classes ensure that the connector receives data for tasks from partitions.

The minimum parameters that must be provided in the request to successfully create a connector are:

- name — the unique name of the connector. The name must not be repeated for connectors created in this cluster
- connector.class — the name of the connector class
- tasks.max — the maximum number of threads (tasks) in which the connector can run (default is 1).

Below are examples of creating a connector using several technologies.

Kafka Connect REST API

Each Kafka Connect worker provides its own REST API requests to create connectors and manage connectors and tasks.

For example, a request to create a simple FileStreamSource connector (copies data from a local file to a Kafka topic) looks like this:

```
$ curl -X POST --data-binary "@source.json" -H "Content-Type: application/json" http://localhost:8083/connectors | jq
```

where @source.json is a file that contains all the parameters describing the connector in JSON format:

```
{
  "name": "filetream-kafka",
  "config": {
    "connector.class": "FileStreamSource",
    "file": "/data/source.txt",
    "topic": "data"
  }
}
```

ksqlDB

In the ksqlDB environment, connectors are created using several CREATE CONNECTOR operators. For example, the connector creates a Debezium connector to communicate with Postgres inside the DB:

```
CREATE SOURCE CONNECTOR _debezium WITH (
  'connector.class' = 'io.debezium.connector.postgresql.PostgresConnector',
  'database.hostname' = 'postgres',
  'database.port' = '5432',
  'database.user' = 'postgres',
  'database.password' = 'postgres',
  'database.dbname' = 'postgres',
  'database.server.name' = 'debezium',
  'table.whitelist' = 'public.customers',
  'topic.prefix' = 'postgres',
  'key.converter' = 'true',
  'transforms.unwrap.type' = 'io.debezium.transforms.ExtractNewRecordState',
  'transforms.unwrap.drop.tombstones' = 'false',
  'transforms.unwrap.delete.handling.mode' = 'rewrite',
  'plugin.name' = 'pgoutput',
  'slot.name' = '1'
);
```

ADS Control

For ADS, it is possible to create connectors via the Kafka Connects page of the ADS Control user interface.

The following connectors are available for use:

- **MirrorCheckpointConnector**, **MirrorHeartbeatConnector**, **MirrorSourceConnector** — connectors designed for replication of topics via the Mirror Maker 2 mechanism. For information on creating these connectors in the ADS Control interface, see the section Mirror Maker 2 in ADS Control.
- PostgreSQL — Debezium connector for PostgreSQL Server.
- MS SQL — Debezium connector for MS SQL Server.
- **Kafka Sink Connector** — connector for writing data from Kafka to Iceberg tables.

Task

Task is an implementation of data copying between the external system and Kafka. Each task instance is created inside a connector instance using a special task class and parameters, also creating the connector.

Each task runs in its own Java thread.

The figure below shows how data flows through the source task in Kafka Connect.

Kafka Connect task

A task instance (Source Task) for each record creates an instance of the SourceRecord class, which transmits a message data, which is received by the Kafka Connect platform (worker) for writing to Kafka, while sourceOffset is used to store information about the position in the data source to which the record is linked.

The record instance transmits to the worker, via special fields, information about where the data is stored in Kafka (topic, kafkaPartition) and can also transmit data about the record key and value. The sourceOffset field contains information about the data source. For example, file name, table name, etc., and in sourcePartition contains the position of sourceOffset. The data can be used to resume copying data from the source in a previous location in case of failures or restarts for maintenance.

The message and offset are then sent to the Kafka Connect framework for the worker to write Kafka topics using the Kafka API.

The tasks themselves have no state stored in them. The task status is stored in special Kafka topics defined by the config.storage.topic and status.storage.topic parameters, and is managed by the corresponding connectors. Tasks can be stopped, started or restarted at any time using the worker's REST API.

Converter

Converter is a ready-made component that converts data formats. Converters provide a mechanism for converting data from the internal data types used by Kafka Connect to data types received as AVRO, PROTOBUF or JSON Schema.

Converters do:

- serialization and deserialization of data, connecting variations and deserializers from Kafka client libraries.
- saving and retrieving AVRO, PROTOBUF, or JSON schemas from Schema Registry.
- interaction of the connector with the Schema Registry can be for saving and retrieving schemas.

The converter class is specified in the configuration parameters of the Kafka Connect service:

- key.converter
- value.converter

Also, the converter class can be defined separately for each connector during creation.

The figure below shows how data is serialized when using the source connector.

Kafka Connect converter

When connecting for the first time, connectors create schemas for incoming data and use a converter to register the schema in the Schema Registry. During the process of copying data, converters connect the desired schema and serialize the data before writing it to a Kafka topic.

Transform

The single message transform (SMT) module is a ready-made component with simple logic for changing a single message.

SMT transformations are optional and are used to modify data before it is written to Kafka, and to modify data read from Kafka before it is written to the sink.

The removed transformations can be configured with connectors or configured globally.

Examples of using SMT transformations:

- Removing fields from input data, such as personally identifiable information (PII), if prescribed by system requirements.
- Adding metadata information such as provenance to data received through Kafka Connect.
- Changing field data types.
- Changing topic names to include timestamp.
- Renaming fields.

For more complex transformations, such as aggregation or merging with other topics, it is recommended to use stream processing in ksqlDB or Kafka Streams.

Handle errors

Possible error-handling options are described below:

- **Fail fast**, or error-and-handling strategy in which any error during the operation of the Kafka Connect task and connectors causes the connector to fail.
  - This strategy is configured by default using the errors.tolerance parameter, if its value is none. When the connector value is NONE or error occurs, the connector continues its work. In this case, there are two possible scenarios for dealing with errors: ignoring errors and redirecting problematic records to the dead letter queue.
- **Ignore errors** is a type of error-handling strategy, a strategy designed for processing the processing of which an error occurred. Unobserved messages are recorded in special log.

The dead letter queue is configured using the following parameters:

- errors.log.enable — if the parameter value is TRUE, problematic entries that cause an error are written to a special log, in the same time, metadata will also be registered for records that cause errors, such as topic, partition, offset, and timestamp, for the source connector. But the value and key data will not be written. Otherwise, all elements, as well as the timestamp, Kafka topic, Kafka partition, source partition, and source offset.
- errors.log.include.messages — if the parameter value is TRUE, problematic entries that cause an error are written to a special log, in the same time, metadata will also be registered for records that cause errors, such as topic, partition, offset, and timestamp, for the source connector. But the value and key data will not be written. Otherwise, all elements, as well as the timestamp, Kafka topic, Kafka partition, source partition, and source offset.

Kafka Connect in ADS

Connection

After starting and configuring of the **Kafka Connect** service as part of ADS cluster, you can connect to Kafka Connect using the **Kafka Connect Worker** assignment is located, and the port specified in the **connect.distributed.properties** group on the Configuration page of the **Kafka Connect** service.

Information about the Kafka Connect service in the ADCM interface

Kafka

In ADS, the **Kafka Connect** service can be installed only after installing the **Kafka** service. After installing **Kafka Connect**, the bootstrap.servers parameter in the mirror.kafka section automatically receives the value of the bootstrap.servers parameter in the root.kafka section, as well as other values for the interaction between Kafka and Kafka Connect. For example, for correct Kafka Connect operation, the following parameters are set automatically in the **Kafka** section:

Schema Registry

The **Schema Registry** service is installed in ADS simultaneously with the **Kafka Connect** service, provides the ability to convert data from internal data types used by Kafka Connect into data in AVRO, PROTOBUF or JSON Schema and vice versa.

For example, to work with AVRO data in the **connect.distributed.properties** configuration file, you should set a special Kafka Connect service as the **key.converter** and/or **value.converter** parameters, as well as for the **key.converter.schema.registry.parameter.url** and/or **value.converter.schema.registry.url** as the Schema Registry service URL.

ksqlDB

The **ksqlDB** service, installed in ADS simultaneously with the **Kafka Connect** service, provides the ability to manage Kafka Connect connectors. Configuring the interaction between two services is performed on the side of the **ksqlDB** service. For more information, see ksqlDB Connect.

Configure Kafka Connect

Configure Kafka Connect parameters in the ADCM interface is performed on the configuration page of the **Kafka Connect** service.

To configure the parameters of the file, which determines the operating current distributed properties configuration file, set the **Show advanced** switch to active, expand the **connect distributed properties** table and enter new values for the desired parameters in the **Value** column. Kafka Connect parameters that are not available in ADS interface, use the **Add key/value** field, specify the **Key** and **Value** and click the **+** button to add the parameter to the list.

After changing the parameters using the ADCM interface, restart the **Kafka Connect** service. To do this, expand the **Action** action by clicking ⋮ in the **Actions** column, and select the **Restart** action.

Debezium overview

Olga Semme

Contents

To Table of Contents

Debezium features

Debezium and Kafka Connect

Features of Debezium connectors for different DBMS

- PostgreSQL
- MS SQL

Create the Debezium connector

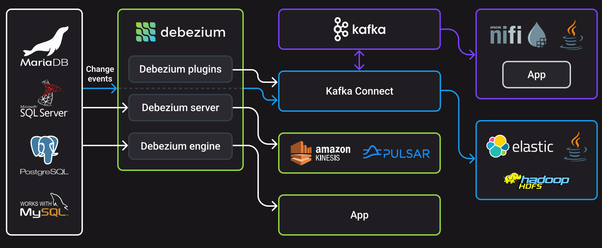
Debezium features

Debezium is a distributed open source platform based on the change data capture (CDC) pattern.

The Debezium connector collects changes in databases and transmits them to external applications.

Debezium is used by external applications to record and process data changes in the DBMS: insertion, update, and deletion events.

The figure below shows the capabilities of the Debezium platform.



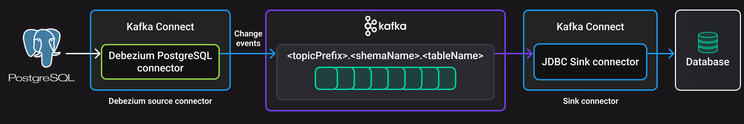
Debezium features

Debezium connectors can be launched using the following methods:

- Via Kafka Connect, using Debezium ready plugins to create connectors. In this case, all changes are written to Kafka topics. This method is recommended if the system uses event streaming to/from Kafka to bypass third-party intermediaries.
- Using Debezium server — a ready-made application that transmits change events from the source database to various messaging infrastructures.
- Using Debezium engine, which allows you to embed Debezium connectors directly into the application space using a special module *debezium-api*. At the same time, there is no such level of fault tolerance and reliability as when using the Kafka Connect service, but there are no intermediaries such as Kafka brokers and the Kafka Connect service.

Debezium and Kafka Connect

The figure below shows what the architecture of a change data capture pipeline looks like using the Debezium connector, for example, when transferring events from a PostgreSQL server to any database that supports JBDC.



Change data capture architecture based on Debezium

The Debezium connector, created with Kafka Connect, writes change events from PostgreSQL to a Kafka topic whose default name consists of a user-specified prefix, the database schema name, and the table name from which changes are collected. Once change event records are stored in Kafka, the various sinks created by Kafka Connect can push the records to other databases and data stores.

Debezium provides a set of connectors for transferring data to Kafka topics from various DBMS.

Features of Debezium connectors for different DBMS

The following describes the functionality of the connectors available by default in ADS Control.

PostgreSQL

Capture of data changes on the PostgreSQL server is provided using a logical replication mechanism based on the write ahead log (WAL). This log is stored on disk and stores all data change events for `INSERT`, `UPDATE`, and `DELETE` queries. These changes are processed using the output plugin:

- decoderbufs — plugin based on Protobuf;
- pgoutput — standard logical decoding output plugin in PostgreSQL 10+, used by PostgreSQL itself for logical replication.

The Debezium connector for PostgreSQL has limitations, including those related to the PostgreSQL logical decoding function used:

- DDL changes are not supported — events associated with `CREATE`, `ALTER`, `DROP`, `TRUNCATE` requests.
- If PostgreSQL is organized as a cluster, the connector can only work with the main server, and if it fails, the connector stops. After the primary server is restored, you can restart the connector. If another PostgreSQL server has been promoted to primary, you must manually configure the connector before restarting.
- The `pgoutput` logic decoding output module does not capture values for generated columns.
- Currently, Debezium only supports databases with UTF-8 character encoding. A single-byte character encoding cannot correctly process strings containing extended ASCII characters.

When the PostgreSQL connector first connects to a PostgreSQL database, it creates a consistent snapshot of each of the database schemas. The connector then streams changes from the location where the snapshot was created.

Streaming replication parameters of the PostgreSQL server that are important for the operation of the connector are listed below:

- `wal_level` — to enable support for logical decoding, it must be set to `logical`;
- parameters affecting the number of connectors that can simultaneously access the server:
  - `max_replication_slots` — the maximum number of replication slots that the server can support.
  - `max_wal_senders` — the maximum number of simultaneous connections from backup servers or streaming basic backup clients.
- `wal_keep_size` — the maximum WAL size limit that a replication slot will keep.

Configuring PostgreSQL Server to run the Debezium connector when using the `pgoutput` plugin requires a database user who has the following privileges:

- `REPLICATION`
- `LOGIN`

Additionally, when using the `pgoutput` plugin, the user requires the following privileges:

- `CREATE` in the database to add publications.
- `SELECT` on tables to copy the original table data.

MS SQL

To allow the Debezium Connector for SQL Server to capture change event records for database operations, you must first enable the change data capture feature available starting with SQL Server 2016. CDC must be enabled on the database and on each table that needs to be captured.

The connector tracks `INSERT`, `UPDATE`, and `DELETE` operations at the row level and writes event records for each table to a separate Kafka topic.

When the connector first connects to an SQL Server database, it takes a consistent schema snapshot of all the tables specified in its configuration and passes it to Kafka. Next, the connector continuously monitors changes at the row level.

Create the Debezium connector

Debezium connectors, like other connectors, can be created using multiple ways through Kafka Connect.

On the Kafka Connect page of the ADS Control user interface, plugins for creating some Debezium connectors for transferring data to Kafka cluster topics are available for selection ADS, as well as the ability to control the operation of connectors:

- PostgreSQL
- MS SQL

The remaining connectors can be installed independently. To do this, you need to create a JAR file using the connector code downloaded from the Debezium repository and save it in the folder, which path is specified as the ***plugin.path*** parameter in the ***connect-distributed.properties*** group on the configuration page of the ***Kafka Connect*** service.

Kafka REST Proxy overview

Olga Semme

Contents

To Table of Contents

- Kafka REST Proxy features
- Kafka REST Proxy architecture
  - HTTP requests to Kafka REST Proxy
  - Serialization and deserialization
  - Security
  - Kafka REST Proxy limitations
- Kafka REST Proxy in ADS
  - Connection
  - Kafka
  - Schema Registry
  - Security
  - Configure Kafka REST Proxy

Kafka REST Proxy features

Kafka REST Proxy is a special proxy server created by Confluent. Kafka REST Proxy provides a RESTful interface to interact with Kafka clusters, that allows users to write and read messages via HTTP/HTTPS requests.

Interaction between external systems and Kafka brokers is done by default via a specific set of API and CLI tools provided by the Java client library. Also client libraries are supported for some programming languages.

Kafka REST Proxy helps you overcome the limitations associated with the narrow choice of ways to interact with Kafka. Any application, including not based on the JVM, can be configured to work with a Kafka cluster, regardless of programming language, as long as the application has the ability to send HTTP/HTTPS requests.

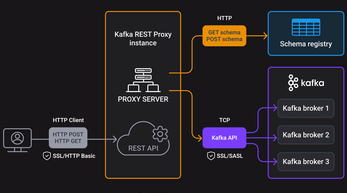
Kafka REST Proxy features

The main functionality of Kafka REST Proxy is listed below:

- The HTTP RESTful interface allows clients to interact with Kafka using standard HTTP methods, thereby ensuring compatibility with various programming languages.
- Interaction with Kafka API via a proxy server is provided:
  - creating and sending messages to Kafka topics in various formats;
  - subscription to topics and reading messages in real time;
  - obtaining data about the cluster and topics.
- Integration with Schema Registry for creating and developing data schemas is supported.
- Authentication and authorization mechanisms are supported to protect access to Kafka clusters.

Kafka REST Proxy architecture

The figure below shows the architecture of Kafka REST Proxy.



Kafka REST Proxy architecture

HTTP requests to Kafka REST Proxy

Each Kafka REST Proxy instance is a separate proxy server, which is an additional layer between the client and the Kafka broker.

The client makes an HTTP request to the Kafka REST Proxy server's REST API using the IP address of the Kafka REST Proxy server, not the Kafka broker.

A write request is a `POST` method that specifies metadata (the location where the message will be stored) and a request body with the contents of the message. A request to retrieve data from Kafka is a `GET` method specifying metadata (the location from which the message should be read).

> **NOTE**
>
> For examples of requests to the Kafka REST Proxy server, you can refer to the Kafka REST Proxy usage example article.

The Kafka REST Proxy server converts received requests into Kafka API application instances (Producer API, Consumer API, or Admin API) and returns the response generated by the application's execution to the user.

Serialization and deserialization

To perform data serialization and deserialization, the Kafka REST Proxy server connects to the Schema Registry server's REST API and writes a new one or retrieves an existing schema.

For this purpose, special headers are used in user requests to the Kafka REST Proxy server:

- `Content-Type` — specifies the type of content that should be written to Kafka.
- `Accept` — specifies the type of content that the client can understand.

These headers indicate content type in the format used by the Schema Registry API.

For example, below is an example query that writes data to a Kafka topic and also registers a schema in the Schema Registry:

```
$ curl -X POST -H "Content-Type: application/vnd.kafka.avro.v2+json" \
      -H "Accept: application/vnd.kafka.v2+json" \
      --data '{"value_schema": "{\"type\": \"record\", \"name\": \"User\", \"fields\": [{\"name\": \"name\", \"type\": \"string\"}]}", "records": [{"value": {"name": "testUser"}}]}' \
      "http://localhost:8082/topics/avrotest"
```

Security

On the Kafka REST Proxy server side, authentication is configured for two connections:

- **Connection between client and Kafka REST Proxy server.**
  This connection is configured with basic HTTP authentication using a pair `<username>:<password>`. User authentication data is passed to the Kafka REST Proxy in the HTTP request using the `Authorization` header.

  To enable mutual SSL/TLS authentication, configure:

  - support HTTPS for the Kafka REST Proxy server;
  - Kafka REST Proxy truststore for validating incoming client certificates.

- **Connection between Kafka REST Proxy and Kafka brokers.**
  If Kafka REST Proxy is used to communicate with a Kafka broker that is already configured with SASL authentication, for each Kafka REST Proxy server, you need to configure the appropriate parameters to authenticate the Kafka REST Proxy user in Kafka.

  If SSL/TLS authentication is configured for the Kafka broker, for each Kafka REST Proxy server, you need to configure the SSL parameters to authenticate the Kafka REST Proxy user in Kafka.

Kafka REST Proxy limitations

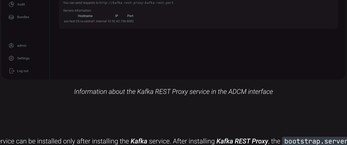
When using Kafka REST Proxy in data processing systems, you should remember the following limitations:

- Requests to write messages to several topics simultaneously are not supported — each HTTP request from the manufacturer can only access one topic or partition.
- Most producer and consumer configuration overrides are not available using HTTP requests.
- It is not possible to use different serializers for a key and a value in a message.
- Requests to Kafka REST Proxy are processed using a synchronous request-response protocol. At the same time, Kafka can connect up to tens of parallel instances of Kafka REST Proxy, which is enough for different use cases.

Kafka REST Proxy in ADS

Connection

After addition and installation of the ***Kafka REST Proxy*** service as part of an ADS cluster, the request to the Kafka REST Proxy server must specify the host where the ***Kafka REST Proxy*** component is located, and the port specified as the ***rest_listener_port*** parameter (default `8082`) in the ***Main*** group on the configuration page of the ***Kafka REST Proxy*** service page.



Information about the Kafka REST Proxy service in the ADCM interface

Kafka

In ADS, the ***Kafka REST Proxy*** service can be installed only after installing the ***Kafka*** service. After installing ***Kafka REST Proxy***, the `bootstrap.servers` parameter in the */etc/kafka-rest/kafka-rest.properties* configuration file is automatically set to communicate with the Kafka broker.

Schema Registry

The ***Schema Registry*** service, installed in ADS simultaneously with the ***Kafka REST Proxy*** service, allows you to serialize and deserialize data when working with Kafka topics via the RESTful interface. After installing both services, the `schema.registry.url` parameter in the */etc/kafka-rest/kafka-rest.properties* configuration file is automatically set.

Security

Most security options for Kafka REST Proxy can be configured using the ADCM interface:

- In the ADS cluster, basic HTTP authentication for Kafka REST Proxy can be configured by enabling Basic authentication.
- In the ADS cluster, SASL authentication is enabled using the Manage Kerberos action or by running the SASL_PLAINTEXT mechanism.
- In the ADS cluster, the kafka-rest-security special module provides the ability to authenticate in Kafka with different principals when connecting to Kafka via Kafka REST Proxy after enabling SASL authentication.
- In an ADS cluster, SSL authentication is enabled using the Manage SSL action.

Configure Kafka REST Proxy

Configuring Kafka REST Proxy parameters in the ADCM interface is performed on the configuration page of the ***Kafka REST Proxy*** service.

To configure the parameters of the */etc/kafka-rest/kafka-rest.properties* configuration files, set the ***Show advanced*** switch to active, expand the ***kafka-rest.properties*** node, and enter new values for the parameters. To change Kafka REST Proxy parameters that are not available in the ADCM interface, use the ***Add key,value*** field. Select ***Add property*** and enter the name of the parameter and its value.

After changing the parameters using the ADCM interface, restart the ***Kafka REST Proxy*** service. To do this, apply the Restart action by clicking in the ***Actions*** column.

ksqlDB overview

Olga Semme

Contents

To Table of Contents

ksqlDB features

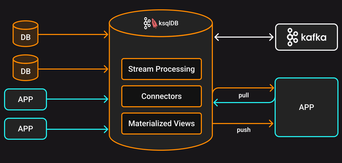
ksqlDB architecture

ksqlDB in ADS

- Connection
- Kafka
- Schema Registry
- Kafka Connect
- Configure KsqlDB

ksqlDB features

ksqlDB is a Kafka database used for stream processing. ksqlDB processes data stored in Kafka and combines Kafka Streams for stream processing and Kafka Connect for collecting and processing events from various data sources. The Kafka Streams processing applications inside ksqlDB are a set of SQL queries.



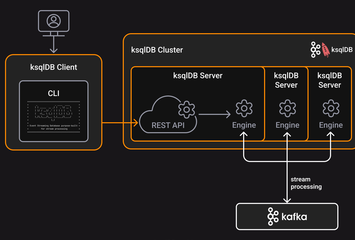
ksqlDB features

The main functionality of ksqlDB is listed below:

- Modeling data stored in Kafka into streams or tables via SQL.
- Executing push queries for data transfer. Using push requests, you can subscribe to changes occurring in the database and track changes in real time. A push request can be sent via the command line interface (CLI) or as an HTTP request to the ksqlDB REST API.
- Executing pull queries to retrieve data. A pull query can retrieve the current value from a materialized view, table, or stream.
- Creating materialized views from streams and tables. The materialized view in the background collects data from the Kafka table, converts it into the required format, and places it into a pre-created target table. You can bind multiple materialized views to a single Kafka table to store data at different levels of granularity across multiple tables.
- Creating connectors for integration with external data stores.

ksqlDB architecture

The figure below shows the architecture of ksqlDB.



ksqlDB architecture

Every ksqlDB server runs the Kafka Streams applications and is essentially a separate instance of the Kafka Streams application. As part of a cluster, several ksqlDB servers share the load created by the Kafka Streams topology.

The ksqlDB server consists of two components:

- **ksqlDB engine** — executes SQL statements and queries. The engine parses SQL statements, converts them to Kafka Streams topology, and launches the Kafka Streams applications.
- **REST API ksqlDB** — provides client access to the ksqlDB engine to send requests.

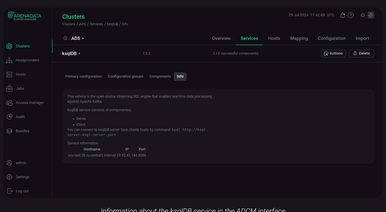
To connect to the ksqlDB REST interface, the client can use the following types of the ksqlDB interface:

- the ksqlDB CLI command line interface — a console in which the SQL query are entered using the ksqlDB operators;
- the ksqlDB user interface — allows you to develop the ksqlDB applications in Control Center for Confluent Platform;
- the REST interface itself — allows you to make requests to the server using the ksqlDB REST API with the cURL command line tool.

ksqlDB in ADS

Connection

After adding and installation of the ***ksqlDB*** service as part of an ADS cluster, you can connect to the ksqlDB server from the hosts where the ***ksqlDB Client*** component is located using the following command: `ksql http://ksql-server:ksql-server-port`.



Information about the ksqlDB service in the ADCM interface

Kafka

In ADS, the ***ksqlDB*** service can be installed only after installing the ***Kafka*** service. After installing ***ksqlDB***, the `bootstrap.servers` parameter in the */etc/ksqldb/ksql-server.properties* configuration file is automatically set for communication with the Kafka broker, as well as others options for interaction between Kafka and ksqlDB (for example, for internal ksqlDB topics created in Kafka).

Schema Registry

The ***ksqlDB*** service, installed in ADS simultaneously with the Schema Registry service, allows registration, reading of schemas, as well as serialization of data using specified (with an identifier) schemas. This makes it easier to work with data serialization, since there is no need to manually define columns and data types in ksqlDB. The format can be created for both keys and values. For example, when creating a stream based on a Kafka topic, the use of schemas is described by the `KEY_SCHEMA_ID` or/and `VALUE_SCHEMA_ID` properties of the thread as shown below.

```
CREATE STREAM pageviews
  WITH (
    KAFKA_TOPIC='avro-topic',
    VALUE_FORMAT='AVRO',
    VALUE_SCHEMA_ID=1
  );
```

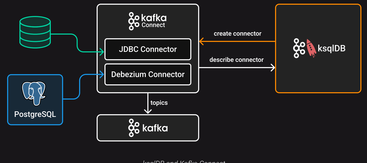
ksqlDB may contain columns whose contents do not match the schema format. In this case, the output of columns corresponding to the schema format can be displaying.

After installing the ***ksqlDB*** service, the `key.converter.schema.registry.url` and `value.converter.schema.registry.url` parameters, which are responsible for the interaction between ksqlDB and Schema Registry, are set in the */etc/ksqldb/connect.properties* configuration file.

Kafka Connect

The ***ksqlDB*** service, installed in ADS simultaneously with the ***Kafka Connect*** service, provides the ability to manage the Kafka Connect connectors by performing the following actions:

- create connectors;
- describe connectors;
- drop connectors;
- import topics created by Kafka Connect into its own database for use in stream processing.



ksqlDB and Kafka Connect

Examples of connectors that can be created using ksqlDB:

- Debezium PostgreSQL connector — a source connector that receives a snapshot from a PostgreSQL database, then tracks all the subsequent changes of this data at the row level for each table and writes it to a separate Kafka topic;
- JDBC connectors — connectors that work with any database with a JDBC driver with data import into Kafka topics (JDBC Source connector) or exporting data from Kafka topics (JDBC Sink connector).

For interaction between the ***ksqlDB*** and ***Kafka Connect*** services, configure the `ksql.connect.url` parameter in the */etc/ksqldb/connect.properties* configuration file.

Configure KsqlDB

Configuring ksqlDB parameters in the ADCM interface is performed on the configuration page of the ***ksqlDB*** service.

To configure the parameters of the */etc/ksqldb/ksql-server.properties* and */etc/ksqldb/connect.properties* configuration files, activate the ***Show advanced*** switch, expand the ***server.properties*** or ***connect.properties*** node and enter new values for the parameters. To change the ksqlDB parameters that are not available in the ADCM interface, use the ***Add key,value*** field. Select ***Add property*** and enter the name of the parameter and its value.

After changing the parameters using the ADCM interface, restart the ***ksqlDB*** service. To do this, apply the Restart action by clicking in the ***Actions*** column.

MiNiFi overview

Olga Semme

Contents

To Table of Contents

MiNiFi features

MiNiFi architecture

- C2 Server
- MiNiFi Agent

MiNiFi implementation

MiNiFi in ADS

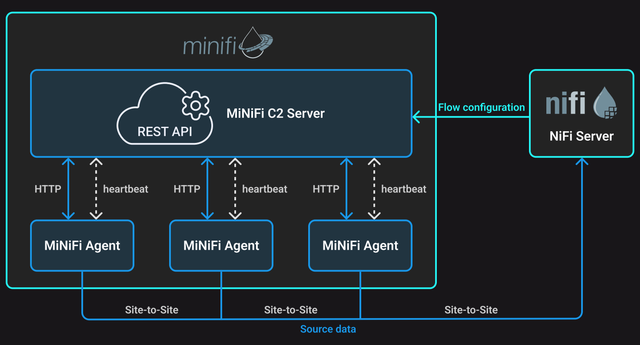
MiNiFi features

MiNiFi — a subproject that extends the functionality of NiFi. MiNiFi provides the ability to collect data from the most remote peripheral devices where it is impossible to install a full-fledged NiFi Server, such as cell towers, network access devices, routers, etc. Collecting data directly at the source reduces the likelihood of data loss during transmission.

To ensure lightweight, MiNiFi lacks many of the features present in NiFi. For example, due to the lack of a user interface in MiNiFi, data flow construction can only be done in full-fledged NiFi to create templates that will be used in MiNiFi.

MiNiFi architecture

The figure below shows the architecture of MiNiFi.



MiNiFi architecture

MiNiFi works by building a custom NiFi data flow. Multiple agents (MiNiFi Agent) collect data from sensors or other devices (for example, via the MQTT protocol) and transmit them into the flow using specially configured processors (for example, ConsumeMQTT). Once the data flow is created and the required attributes for the FlowFile (sensor parameters, location, etc.) are added, the data is sent to the main NiFi instance via the Remote Process Group.

C2 Server

MiNiFi C2 Server — a central server that manages the configuration and operation of multiple agents via the C2 protocol. MiNiFi C2 Server is installed on the server that must be accessible from MiNiFi agents. Multiple C2 servers can be used for complex network structures.

The main functions of MiNiFi C2 Server are described below:

- The server monitors heartbeats from agents and in response, if necessary, sends commands to update or write the flow configuration.
- Each new agent is asked by the server for a manifest (all pre-installed configurations, processors, and controller services). The agent manifest is used in the C2 protocol to design a flow for an agent class — a group of agents with identical manifests.
- The server manages MiNiFi flow configurations and publishes them via its own REST API. MiNiFi agents can connect to this API to update their own configuration at intervals defined by the `nifi.minifi.notifier.ingestors.pull.http.period.ms` parameter.
  The flow configuration is a YML file (*config.yml*) and can be created manually. Since this method is laborious, the most common flow is one created using the NiFI user interface and saved as template in XML file format. MiNiFi C2 Server accesses templates via the NiFi REST API. The template can be converted to YML format using the `minifi-toolkit` utility. The template name usually corresponds to the agent class name.

MiNiFi Agent

MiNiFi Agent — an agent that runs data flows and is essentially a lightweight version of NiFi.

The main functions of MiNiFi Agent are described below:

- In response to heartbeat messages, the agent receives a command from the C2 server to update the configuration.
- The agent sends confirmation of all control commands received from the server via the C2 protocol.
- The agent downloads the flow using the server's REST API. The agent class is specified using the `nifi.minifi.notifier.ingestors.pull.http.query` parameter.
- The agent applies the new flow and sends the collected data via the Site-to-Site protocol to the remote NiFi Server instance using the Remote Process Group.

MiNiFi implementation

MiNiFi is available in two variants:

- MiNiFi C++ — a binary, uses little system memory, but can run a limited subset of NiFi processors.
- MiNiFi Java — contains most of the available NiFi processors, but is a larger binary distribution and uses more system resources.

MiNiFi in ADS

The ***MiNiFi*** service in the ADS cluster can only be installed simultaneously with the ***NiFi*** service or provided that ***NiFi*** is already installed.

After adding and installing the ***MiNiFi*** service in the ADS cluster, the following parameters are automatically set for each service component:

- For the ***MiNiFi C2 Server*** component, the */etc/minifi-c2/conf/minifi-c2-context.xml* file specifies the API address of the NiFi instance installed in the same ADS cluster, with which the flow templates will be loaded (`http://<hostname>:9090/nifi-api`).
- For the ***MiNiFi Agent*** component, the */etc/minifi/conf/bootstrap.conf* file automatically specifies the parameters for updating flow configurations and communicating with the MiNiFi C2 Server, and also loads the current flow configuration (if any) into the */etc/minifi/conf/config.yml* file.

Configuring parameters and changing default parameters is performed on the configuration page of the ***MiNiFi*** service in the ADCM interface.

To connect to a NiFi instance via the Site-to-Site protocol, the ***NiFi*** service has parameters responsible for remote connection (with the `nifi.remote` prefix) set in the */etc/nifi/conf/nifi.properties* file.

Descriptions of the parameters are given in the ***MiNiFi*** and ***NiFi*** sections of the ADS configuration parameters article.

After changing the parameters using the ADCM interface, restart the service. To do this, apply the ***Restart*** action by clicking in the ***Actions*** column.

NiFi overview

Olga Semme

Contents

To Table of Contents

Features

Architecture

- Core components of a NiFi node
- NiFi cluster architecture

Configure NiFi

Features

NiFi is a platform created to automate the management of data flows between systems.

The main functionality of NiFi is listed below:

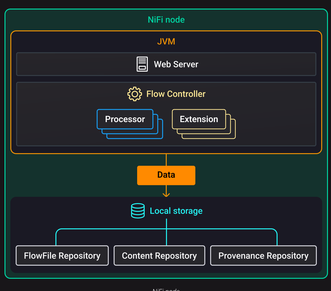
- Data flows are managed using special processors, each of those has its own functional purpose. At the same time, to ensure data availability, guarantee data delivery and resilience when handling reboots and unexpected system failures, the following functions are supported:
  - Use a **Write-Ahead Log (WAL)**, that fixes every change that occurs to the FlowFile as a transactional unit, records the successful metadata change in the FlowFile repository, and in case of system failure it restores its state.
  - The ability to flexibly configure data buffering in a queue using backpressure when the queue reaches certain parameters.
  - The ability to configure schemes for prioritizing data processing from a queue.
- Features that make flow management easy:
  - NiFi user interface provides the ability to visually create and manage data streams in real time. To create new components in the system or make changes to existing ones, there is no need to stop all components.
  - Flow templates allow you to create and share flow projects.
  - Storing in the Data Provenance repository information about the provenance of the data and all the operations to which they were subjected while passing through the system. This allows you to analyze the system for compliance with specified requirements, as well as optimization and troubleshooting.
  - Ability to view content stored in content repository at any time during the stream processing cycle.
- Features that ensure system security:
  - Possibility of using encrypted protocols (two-way SSL) when transferring data between two systems.
  - NiFi supports SSL authentication and provides pluggable authorization in the user interface or via Apache Ranger. **Multi-tenant authorization** can also be configured, allowing different groups of users to configure access to different parts of the data flow with different levels of authorization.
- Extension options:
  - NiFi provides the ability to connect extensions through the following points: processors, controller services, reporting tasks, prioritizers, and client user interfaces.
  - Limited set of dependencies for creating extensions and use of NiFi's own class loader model.
  - Possibility of connecting between NiFi instances using the **Site-to-Site (S2S)** communication protocol, which ensures secure and efficient data transfer, as well as the use of extension libraries for data transfer.
- Scaling options:
  - NiFi is designed to scale by clustering multiple nodes. This allows you to configure thread balancing as well as failover between nodes.
  - Throughput throttling can be done by changing the number of tasks on the SCHEDULING tab when configuring the processor.

Architecture

Core components of a NiFi node

NiFi runs inside a JVM on the host operating system.

The components of a NiFi node are shown in the figure below.



NiFi node

The NiFi node contains the following components:

- **Web Server** is a server that accepts HTTP requests from clients to manage NiFi streams and produces the result in the form of demonstrating stream changes visually (using the user interface) or HTTP responses.
- **FlowFile** is a NiFi object representing a package of information moving through the system. For every package NiFi keeps track of attributes as key/value pairs and content associated with it.
- **Processor** is a NiFi component that performs the FlowFiles data processing actions stated in its function, for example, retrieving or publishing data. Processors can use one or more FlowFiles. Processors have access to the attributes of a given FlowFile and its contents.
- **Extension** is a library that increases or changes the functionality of an object and is loaded when extension points are launched. Extensions in NiFi are launched and executed inside the JVM. Extension points include: processors, controller services, reporting tasks, prioritizers, and client user interfaces.
- **Flow Controller** is a NiFi component that performs the functions of managing the exchange of FlowFiles between processors and controlling the launch of extensions. A thread controller can be configured to control a specific set of threads.
- **NiFi repositories** are repositories that store data for running NiFi streams. Repositories are directories in the NiFi node's local storage:
  - **FlowFile Repository** is a repository that contains metadata for all current FlowFiles in a flow, which tracks the entire path of the FlowFile through the system. The repository implementation is pluggable. By default, Write-Ahead Log is used to store data. The path to the repository on disk is specified using the `nifi.flowfile.repository.directory` parameter.
  - **Content Repository** is a repository that contains the actual bytes of the content of this FlowFile. The repository implementation is pluggable. By default, a repository is a mechanism that stores blocks of data on the file system. The path to the repository on disk is specified using the `nifi.content.repository.directory` parameter. You can specify more than one file system storage location.
  - **Provenance Repository** is a repository that contains all data about the occurrence of events, the history of FlowFiles. Event data at each location is indexed and searchable. The repository implementation is pluggable. The path to the repository on disk is specified using the `nifi.provenance.repository.directory` parameter. You can specify one or more physical disk volumes.

NiFi cluster architecture

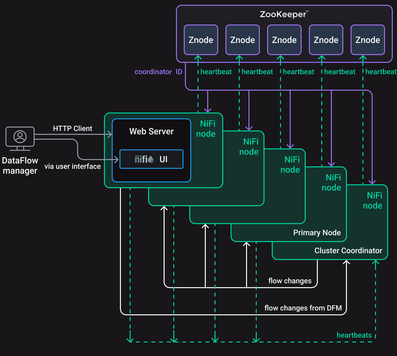
Starting with version 1.0, NiFi uses the **Zero-Leader Clustering** paradigm, in which there is no single system leader, and accordingly, there is no single point of failure.

To coordinate the operation of NiFi nodes in a distributed system (in a cluster), ZooKeeper is used, which ensures the availability of flow data in case of failures or maintenance of nodes.

NOTE

- The concepts and main components of ZooKeeper are described in the ZooKeeper article.
- The mechanism of the leader election using ZooKeeper is described in the Leader election in ZooKeeper article.

The figure below shows how the NiFi nodes interact with each other and with ZooKeeper in the cluster.



NiFi cluster

When NiFi nodes start up as part of a cluster, each one registers itself in ZooKeeper — creates an ephemeral ZooKeeper node (znode) that specifies a node ID, hostname, and port. ZooKeeper monitors node availability.

Changes to the NiFi flow can be made by the DataFlow manager (DFM) via the NiFi user interface page or by connecting to the NiFi node web server (for example, using NiFi REST API). In this case, DFM can use any node from the cluster. Every change made is replicated to all connected nodes in the cluster.

ZooKeeper, in accordance with the ZAB consensus algorithm protocol (ZooKeeper Atomic BroadCast), elects key cluster nodes:

- **Cluster Coordinator** is a node in the NiFi cluster that performs the following operations in the NiFi cluster:
  - Processes heartbeat signals, through which cluster nodes report information about their status, and disabling nodes that do not receive a heartbeat signal within the time specified in the `nifi.cluster.protocol.heartbeat.interval` parameter (default value `5 s`)
  - Provides the current actual stream to new joining nodes (the decision about whether a node can join is made based on the configured firewall file *nifi.cluster.firewall.file*).
  - Determines the correct version of a thread when starting a NiFi cluster using the **leader election** mechanism — voting on node threads. In this case, the versions of streams of different nodes are compared and if the streams of two nodes coincide, a vote is given for this stream. Flow elections are controlled by the `nifi.cluster.flow.election.max` and `nifi.cluster.flow.election.max.wait.time` parameters. On the node whose flow conflicts with the flow of the "election winner" node, a backup copy of the flow configuration file *flow.xml.gz* is created (the file is stored in accordance with the `nifi.flow.configuration.file` parameter), and the file itself is replaced by the "correct stream" file.
  - Coordinates flows on all nodes. All web requests from each node are forwarded to the coordinator, which then replicates the request to all nodes in the cluster.
- **Primary Node** is a node where you can run **isolated processors**, which work on only one cluster node. By default, in a NiFi cluster, the same data stream runs on all nodes. In a situation where the processor is processing files from a remote directory, it may be necessary to limit data transfer to one channel to avoid errors when duplicating data. Enabling processor "isolation" is done using the ***Execution*** parameter on the SCHEDULING tab when configuring the processor.

The cluster coordinator and primary node are displayed on the Cluster page of the global UI menu.

ZooKeeper monitors the availability of nodes and automatically selects a new cluster coordinator and master node if the previous ones become unavailable. Also from ZooKeeper, all nodes receive the ID of the cluster coordinator to send heartbeat signals.

Configure NiFi

After adding and installing the ***NiFi*** service as part of ADS cluster, you can configure NiFi parameters on the configuration page of the ***NiFi*** service via ADCM.

To configure backpressure parameters, expand the ***Main*** node in the configuration settings tree and enter new values for the parameters.

To change storage locations for NiFi repositories, expand the ***Directories*** node in the configuration settings tree and enter new values for the parameters.

To configure the NiFi cluster operating parameters, set the ***Show advanced*** switch to active, expand the ***nifi.properties*** node and enter new values for the parameters. To change NiFi parameters that are not available in the ADCM interface, use the ***Add key,value*** field in the ***nifi.properties*** node. Select ***Add property*** and enter the name of the parameter and its value.

Schema Registry overview

By Tilgin Almuca

Contents

To Table of Contents

Schema Registry features
Schema Registry architecture
- Overview
- Schema Registry cluster
- Producers
- Consumers

Schema Registry in ADS
- Connection
- Kafka
- ksqlDB
- Kafka Connect
- Kafka REST Proxy
- NiFi
- Configure Schema Registry

Schema Registry features

Schema Registry is a centralized repository used to ensure consistency of data formats between message producers and consumers in Kafka.

Schema Registry features

The main functionality of Schema Registry is listed below:

- Ensuring serialization and deserialization of data via special serializers provided to Kafka clients.
- Interaction with the Kafka clients to write and read serialized messages.
- with producers;
- with consumers;
- with various schema managers (e.g. the Schema Registry Mixin plugin) as part of CI/CD technology.
- Registration and storage of data schemas for each subject.
- Support for the evolution of schemas with storage of each version with its own identifier and the ability to check version compatibility for each subject.
- Ensuring accessibility for performing actions with schemas and subjects via the RESTful interface using Schema Registry API.

Schema Registry architecture

Overview

Basic Schema Registry terminology:

- **Serialization** is the process of writing data from a format-structure that is semantically understandable to humans into a stream/byte array in a binary code format for machine processing. Schema Registry supports the following serialization formats: JSON, Avro, and Protobuf.
  Benefits of the serialization:
  - ability to use different data formats in Kafka messages;
  - ability to exchange data between applications implemented in different programming languages;
  - reduction in the total volume of stored data.
- **Deserialization** is the restoration of a data structure from a byte stream.
- **Serializer and deserializer** are objects (or classes) that represent a set of functions and methods that allow the client to interact with the Schema Registry to provide serialization and deserialization. There are built-in Kafka serializers and deserializers for different schema formats. The Kafka API also supports the creation of custom serializers and deserializers based on the Serializer interface. The serializer is defined in the producer configuration using the `key.serializer` and/or `value.serializer` parameters. The deserializer is defined in the consumer configuration using the `key.deserializer` and/or `value.deserializer` parameters.
- **Subject** is the area in which schemas can evolve. The subject name is created when the schema is first registered and is used:
  - when changing (evolving) the schema as a namespace for different versions;
  - to check the compatibility of circuit versions;
  - to get the schema from the repository.

  There are three configurable ways to name a subject:
  - `TopicNameStrategy` is the default setting. Schemas that are registered by producers will be registered under the topic name with the addition of `-key` or `-value`. (for example, for a topic `Topic`, which only serializes values, the subject will be named `Topic-value`).
  - `RecordNameStrategy` – allows you to use schemas in one topic, since each individual record will have its own schema. In this case, you will have to use the same schema and the same version in all topics of the cluster for this particular record type, since it is impossible to determine which topic the record belongs to.
  - `TopicRecordNameStrategy` – a combination of the first two strategies.

  A non-default subject name is set using the producer parameter when registering a new schema — `key.subject.name.strategy` and/or `value.subject.name.strategy`.
- **Schema** is the format structure of the serialized data. The schema describes the structure of the data written to the topic and what type of information they contain. This information connects producers and consumers. There are specific requirements for each format, for example, the AVRO format schema must comply with AVRO format specification.

  Below are examples of supported format schemas for the same sample data used – `{"id":"100", "amount":500 }`:

JSON

```
{
  "type": "object",
  "properties": {
    "id": {
      "type": "string"
    },
    "amount": {
      "type": "number"
    }
  }
}
```

AVRO

```
{
  "type": "record",
  "name": "Transaction",
  "fields": [
    {
      "name": "id",
      "type": "string"
    },
    {
      "name": "amount",
      "type": "double"
    }
  ]
}
```

PROTOBUF

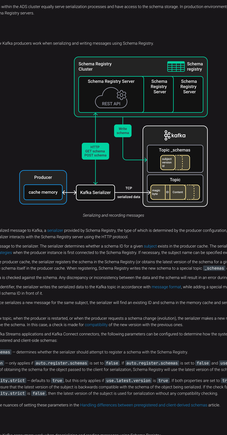
```
syntax = "proto3";

message MyRecord {
  string id = 1;
  float amount = 2;
}
```

- **Schema compatibility** – compliance of the new version of the schema with previous versions for a given subject. Multiple compatibility types may be defined depending on what data has been changed or deleted. Compatibility types are essentially templates for how a schema can evolve so that a new version of it can be successfully processed by producers and consumers. The compatibility type is specified using the schema.compatibility.level Schema Registry server parameter, which is specified in the schema registry properties file.

Schema Registry cluster

Every Schema Registry server is essentially a REST API server. Schema Registry servers provide APIs for managing schemas, as well as serializers and deserializers for reading and writing data according to schemas.

All Schema Registry servers within the ADS cluster equally serve serialization processes and have access to the schema storage. In production environments, it is recommended to specify the URL of all Schema Registry servers.



Serializing and recording messages

To serialize and write a serialized message to Kafka, a serializer provided by Schema Registry, the type of which is determined by the producer configuration, is connected to the producer instance. The serializer interacts with the Schema Registry server using the HTTP protocol.

The producer sends the message to the serializer. The serializer determines whether a schema ID for a given subject exists in the producer cache. The serializer obtains the subject name based on strategy (for example, `TopicNameStrategy`) and checks the Schema Registry. If necessary, the subject name can be specified explicitly.

If the schema ID is not in the producer cache, the serializer registers the schema in the Schema Registry (or obtains the latest version of the schema for a given subject) and stores the resulting schema ID and the schema itself in the producer cache. When registering, Schema Registry writes the new schema to a special topic (`_schemas` on the Kafka broker by default).

Before serialization, the data is checked against the schema. Any discrepancy or inconsistency between the data and the schema will result in an error during serialization.

After receiving the schema identifier, the serializer writes the serialized data to the Kafka topic in accordance with message format, while adding a special magic byte (serialization format version number) and schema ID in front of it.

If the same producer instance serializes a new message for the same subject, the serializer will find an existing ID and a schema in the memory cache and serialize the message according to it.

To write a message to a new topic, when the producer is restarted, or when the producer requests a schema change (evolution), the serializer makes a new request to the Schema Registry to register or retrieve the schema. In this case, a check is made for compatibility of the new version with the previous ones.

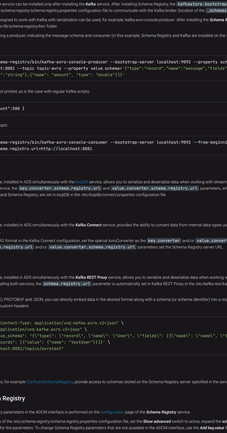
For producers, including Kafka Streams applications and Kafka Connect connectors, the following parameters can be configured to determine how the system behaves when there are differences between pre-registered and new schemas:

- `auto.register.schemas` – determines whether the serializer should attempt to register a schema with the Schema Registry.
- `use.latest.version` – only applied if `auto.register.schemas` is set to `false`. If `auto.register.schemas` is set to `false` and `use.latest.version` is set to `true`, then instead of obtaining the schema for the object passed to the client for serialization, Schema Registry will use the latest version of the schema in subject.
- `latest.compatibility.strict` – defaults to `true`, but this only applies if `use.latest.version` = `true`. If both properties are set to `true`, a check is performed during serialization to ensure that the latest version of the subject is backwards compatible with the schema of the object being serialized. If the check fails, an error is thrown. If `latest.compatibility.strict` = `false`, then the latest version of the subject is used for serialization without any compatibility checking.

You can read more about the nuances of setting these parameters in the Handling differences between preregistered and client-derived schemas article.

Consumers

The figure below shows how Kafka consumers work when deserializing and reading messages using Schema Registry.



Deserializing and reading messages

To deserialize a message in Kafka, a deserializer provided by Schema Registry, the type of which is determined by the consumer configuration, is connected to the producer instance. The deserializer interacts with the Schema Registry server via the HTTP protocol.

The consumer receives data from the Kafka cluster (serialized message), passes it to the deserializer. The deserializer checks for the presence of the magic byte and extracts the schema identifier from the message.

The deserializer then reads the schema ID and checks whether a matching schema exists in the consumer cache. If exists, deserialization occurs with this schema. Otherwise, the deserializer retrieves the schema from Schema Registry by the schema ID. Once the schema is ready, the deserializer starts deserializing.

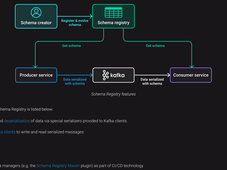
If the serialized data does not match the expected schema (e.g. missing fields), deserialization will fail and an error will be raised.

For consumers, a mechanism has a value of `use.latest.version`, which determines how the system behaves when there are differences between pre-registered and client schemas.

Schema Registry in ADS

Connection

After activation and installation of the **Schema Registry** service as part of an ADS cluster, you can connect to the Schema Registry server using its URL in the `http://schema-registry:<schema-registry-port>` format with the host and port specified on the service information page in the ADCM interface.



Information about the Schema Registry service in the ADCM interface

Kafka

In ADS, the **Schema Registry** service can be installed only after installing the **Kafka** service. After installing Schema Registry, the `kafkastore.bootstrap.servers` parameter is automatically set in the /etc/schema-registry/schema-registry.properties configuration file to communicate with the Kafka broker (location of the `_schemas` service topic).

For test purposes, scripts designed to work with Kafka with serialization can be used, for example, *kafka-avro-console-producer*. After installing the **Schema Registry** service, such scripts are located in the usr/lib/schema-registry/bin folder.

Below are examples of running a producer, indicating the schema registry and consumer (in this example, Schema Registry and Kafka are installed on the same host):

Recording a message:

```
$ sudo /usr/lib/schema-registry/bin/kafka-avro-console-producer --bootstrap-server localhost:9092 --property schema.registry.url=http://localhost:8081 --topic topic-avro --property value.schema='{"type":"record","name":"message","fields":[{"name":"id","type":"string"},{"name": "amount", "type": "double"}]}'
```

To write a message, `{` is not printed, as in the case with regular Kafka scripts:

```
{"id":"100", "amount":500 }
```

Reading messages from a topic:

```
$ sudo /usr/lib/schema-registry/bin/kafka-avro-console-consumer --bootstrap-server localhost:9092 --from-beginning --topic topic-avro --property schema.registry.url=http://localhost:8081
```

ksqlDB

The **Schema Registry** service, installed in ADS simultaneously with the ksqlDB service, allows you to serialize and deserialize data when working with streams and tables in ksqlDB. After integrating the **ksqlDB** service, the `ksql.schema.registry.url` and `value_converter.schema.registry.url` parameters, which are responsible for the interaction between ksqlDB and Schema Registry, are set in ksqldb in the /etc/ksqldb/connect.properties configuration file.

Kafka Connect

The **Schema Registry** service, installed in ADS simultaneously with the **Kafka Connect** service, provides the ability to convert data from internal data types used by Kafka Connect to AVRO format data and back.

To work with data in the AVRO format in the Kafka Connect configuration, set the special AvroConverter as the `key.converter` and/or `value.converter` parameter, and for the `key.converter.schema.registry.url` and/or `value.converter.schema.registry.url` parameters set the Schema Registry server URL.

Kafka REST Proxy

The **Schema Registry** service, installed in ADS simultaneously with the **Kafka REST Proxy** service, allows you to serialize and deserialize data when working with Kafka topics via the RESTful interface. After installing both services, the `schema.registry.url` parameter is automatically set in Kafka REST Proxy in the /etc/kafka-rest/kafka-rest.properties configuration file.

Using the built-in types AVRO, PROTOBUF, and JSON, you can directly embed data in the desired format along with a schema (or schema identifier) into a request by defining the appropriate content type in custom headers:

```
$ curl -X POST -H "Content-Type: application/vnd.kafka.avro.v2+json" \
    -H "Accept: application/vnd.kafka.avro.v2+json" \
    --data '{"value_schema": "{\"type\": \"record\", \"name\": \"User\", \"fields\": [{\"name\": \"name\", \"type\": \"string\"}]}", "records": [{"value": {"name": "testUser"}}]}' \
    "http://localhost:8082/topics/avrotest"
```

NiFi

Some NiFi controller services, for example, ConfluentSchemaRegistry, provide access to schemas stored on the Schema Registry server specified in the service parameters.

Configure Schema Registry

Configure Schema Registry parameters in the ADCM interface is performed on the configuration page of the **Schema Registry** service.

To configure the parameters of the /etc/schema-registry/schema-registry.properties configuration file, set the **Show advanced** switch to active, expand the **schema-registry.properties** node, and enter new values for the parameters. To change Schema Registry parameters that are not available in the ADCM interface, use the **Add key value** field: listed, **Add property** and enter the name of the parameter and its value.

After changing the parameters using the ADCM interface, restart the **Schema Registry** service. To do this, apply the Restart action by clicking ▷ in the **Actions** column.

Service discovery in ZooKeeper

Olga Semme

Contents

To Table of Contents

Overview

Service Registration

Service Discovery

NOTE

The concepts and main components of ZooKeeper are described in the ZooKeeper article.

Overview

Service Discovery is a ZooKeeper service mechanism that acts as a registry, keeping track of the addresses of all instances. Instances have dynamically assigned network paths.

The Service Discovery system provides a mechanism for:

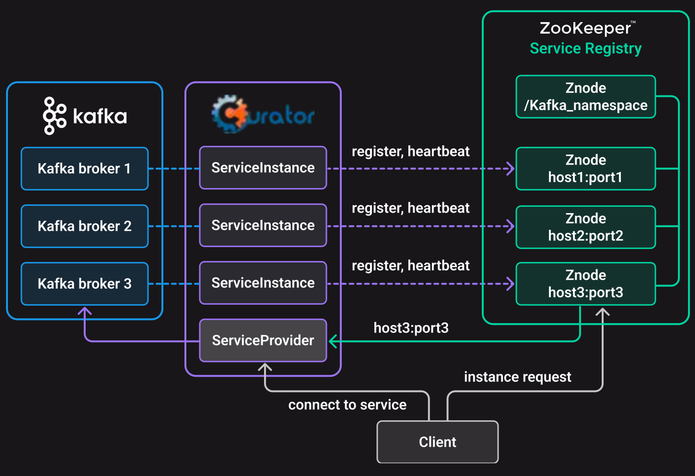
- registration of services and instances of services;
- search for instances of a particular service;
- notifications about changes to service instances.

The entire Service Discovery procedure consists of two phases:

1. Service Registration.
2. Service Discovery.

The Service Discovery mechanism with load balancing uses the curator-x-discovery package from the Apache Curator — Java/JVM client library for Apache ZooKeeper.

The Service Discovery mechanism is shown in the following diagram.



Service Discovery in ZooKeeper

The stages of Service Discovery are described below.

Service Registration

When a Kafka service instance appears, it registers with the **Service Registry** ZooKeeper in the namespace of its service, adding its ephemeral znode. Znode stores the `host:port` of the server as its data. The service instance sends heartbeat requests to save its registration.

Service Registry is a component that contains a database of all available service instances. It stores information about the currently available instances of each service and their network data for establishing a connection.

The Service Registry monitors running instances for changes by polling the deployment environment or by subscribing to events. When the Service Registry detects a new service instance available, it writes it to its database. The Service Registry also unregisters failed (disabled) service instances.

Apache Curator represents a service instance as a `ServiceInstance` class. `ServiceInstances` have a name, ID, address, port and/or ssl port, and an optional (user-defined) payload. `ServiceInstances` are serialized and stored in ZooKeeper like this:

```
base path
       |_______ service A name
                    |__________ instance 1 id --> (serialized ServiceInstance)
                    |__________ instance 2 id --> (serialized ServiceInstance)
                    |__________ ...
       |_______ service B name
                    |__________ instance 1 id --> (serialized ServiceInstance)
                    |__________ instance 2 id --> (serialized ServiceInstance)
                    |__________ ...
       |_______ ...
```

Service Discovery

After registering service instances, service discovery occurs in the following sequence:

1. The client connects to the ZooKeeper ensemble and requests the desired service.
2. Using the `ServiceProvider` class used in Apache Curator, ZooKeeper uses a load balancing algorithm to select one of the available service instances and returns a `host:port` for one of the registered instances.
3. The client connects to the service instance.

Leader election in ZooKeeper

D. Olga Gerenie

Contents

To Table of Contents

Consensus algorithm protocol
- Message exchange
- Message delivery principles
- Controller epoch

Kafka controller election logic

Kafka replicated log leader

NOTE

- The concepts of storage and the main components of Kafka are described in the Storage concepts in Kafka article.
- The concepts and the main components of ZooKeeper are described in the ZooKeeper article.

ZooKeeper is a centralized service used to store configuration information, provide distributed synchronization of Kafka brokers.

The atomic messaging system at the heart of ZooKeeper is based on the principles of the consensus algorithm.

Consensus algorithm – a set of principles and rules, due to which all nodes participating in the cluster automatically come to a consensus on the current state of the network. ZooKeeper implements one of the consensus algorithm protocols — ZAB (ZooKeeper Atomic BroadCast).

Consensus algorithm protocol

The ZAB protocol ensures that ZooKeeper replication occurs in order, and is also responsible for selecting lead nodes and recovering any failed nodes.

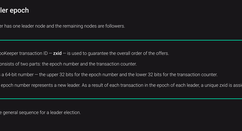
The main provisions of ZAB are described below.

Message exchange

In the messaging part, ZAB uses the following concepts:

- **Packet** is a sequence of bytes sent over the FIFO channel.
- **Message** is a sequence of bytes that will be atomically broadcast to all ZooKeeper servers. The message is placed in the offer and agreed upon before it is delivered.
- **Proposal** is a unit of agreement. Proposal are agreed by exchanging packets when there is a quorum of ZooKeeper servers.

The proposal approval messaging model used by ZooKeeper is shown below.



Messaging model

To communicate with replicas, proposal approval uses a model in which each new record requires at least three transactions: a proposal (propose), an acknowledgment (ack), and a commit.

Message delivery principles

The ZAB protocol involves the creation of FIFO channels (in each session, the server executes client requests one by one in the order in which they are received) of the point-to-point type between servers. The FIFO channel is for communication, providing the following properties:

- **Reliable delivery** – if message `m` is delivered by one server, it will eventually be delivered by all servers.
- **Common order** – if message `m` is delivered before message `h` by one server, then `m` will be delivered before `h` by all servers. If messages `m` and `h` are delivered, then either `m` will be delivered before `h` or `h` will be delivered before `m`.
- **Causal order** – if message `h` is sent after message `m` was delivered by sender `h`, message `m` must be ordered before `h`. if the sender sends `m` after sending `h`, `m` must be ordered after `h`.
- **Ordered delivery** – data is delivered in the same order as it was sent, and message `m` is only delivered after all messages sent before `m` have been delivered. The implication of this is that if the `m` message is lost, all messages after `m` will be lost.
- **No message after close** – once a FIFO channel is closed, no messages will be received from it.

The use of **timeouts** ensures that consensus is reached in the presence of failures.

Controller epoch

Every cluster has one leader node and the remaining nodes are followers.

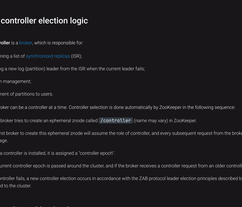
The ZooKeeper transaction ID – **zxid** – is used to guarantee the overall order of the offers.

Zxid consists of two parts: the epoch number and the transaction counter.

Zxid is a 64-bit number – the upper 32 bits for the epoch number and the lower 32 bits for the transaction counter.

A new epoch number represents a new leader. As a result of each transaction in the epoch of each leader, a unique zxid is assigned.

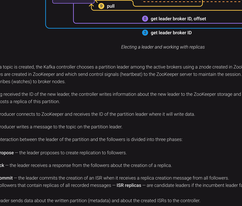
Below is the general sequence for a leader election.



Leader election in ZooKeeper

1. A proposal (propose) of a new epoch arrives from the server, and each proposal is assigned its own zxid.
2. Every node acknowledges (ack) the offer only if it does not know any other leader with a higher epoch number in zxid, or if the epoch is the same, with a higher transaction counter. Before a leader can be chosen, it must collect votes from a quorum of nodes. Quorums to determine the leader must be `(n/2+1)`, where `n` is the number of servers that make up the ZooKeeper service.
3. After the quorum is confirmed, the new leader commits the creation of a new epoch and sets the next zxid to use.
4. The follower will record the formation of a new epoch.

Each phase of the controller epoch is described below.



Controller epoch

Each epoch consists of three phases, and each node can be in one of these three phases at any given time:

- **Leader election** – leader election phase based on current quorum configurations. There can be at most one leader at any given time. The end of the phase comes after the confirmation of the new epoch by all followers.
- **Synchronization** – synchronization phase, during which the new leader synchronizes the available replicas with the previous epoch and with all the followers as a leader. The end of the phase occurs after the quorum of followers has recognized that they are in sync with the leader.
- **Broadcast** – translation phase, the normal mode of operation in which the leader continues to offer new client requests. The end of the phase occurs after the leader failure.

Kafka controller election logic

Kafka **controller** is a broker, which is responsible for:

- maintaining a list of synchronized replicas (ISR);
- choosing a new log (partition) leader from the ISR when the current leader fails;
- partition management;
- assignment of partitions to users.

Only one broker can be a controller at a time. Controller selection is done automatically by ZooKeeper in the following sequence:

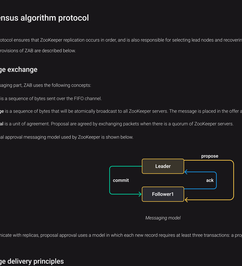
1. Every broker tries to create an ephemeral znode called `/controller` (name may vary) in ZooKeeper.
2. The first broker to create this ephemeral znode will assume the role of controller, and every subsequent request from the broker will receive a `node already exists` message.
3. Once a controller is installed, it is assigned a 'controller epoch'.
4. The current controller epoch is passed around the cluster, and if the broker receives a controller request from an older controller epoch, it is ignored.
5. If a controller fails, a new controller election occurs in accordance with the ZAB protocol leader election principles described by above, and a new controller epoch is created and passed to the cluster.

Kafka replicated log leader

Leader of a journal (partition) is a broker that works with clients. It is the leader that works with message producers. Requests to the leader are made by **followers** – brokers that store a replica of all these partitions. Consumers can read messages from the leader and from the followers.

The ZAB distributed consensus algorithm is also used to negotiate between message writing and reading sequence and managing partition replication in Kafka for availability and system reliability.

The following describes how, in general, leader selection and partition replication management in Kafka is automatically performed using the ZooKeeper service:



Electing a leader and working with replicas

1. After a topic is created, the Kafka controller chooses a partition leader among the active brokers using a znode created in ZooKeeper. Active brokers are those which ephemeral znodes are created in ZooKeeper and which send control signals (heartbeat) to the ZooKeeper server to maintain the session. To track the status of brokers, the controller node subscribes (watches) to broker nodes.
2. Having received the ID of the new leader, the controller writes information about the new leader to the ZooKeeper storage and sends the data about the new leader to each broker that hosts a replica of this partition.
3. The producer connects to ZooKeeper and receives the ID of the partition leader where it will write data.
4. The producer writes a message to the topic on the partition leader.
5. The interaction between the leader of the partition and the followers is divided into three phases:
   - **propose** – the leader proposes to create replication to followers.
   - **ack** – the leader receives a response from the followers about the creation of a replica.
   - **commit** – the leader commits the creation of an ISR, when it receives a replica creation message from all followers.
     Followers that contain replicas of all recorded messages — **ISR replicas** — are candidate leaders if the incumbent leader fails.
6. The leader sends data about the written partition (metadata) and about the created ISRs to the controller.
7. The controller writes information about the created ISRs to the ZooKeeper repository, as well as partition metadata.
8. The consumer connects to ZooKeeper and gets the ID of the leader or followers for the requested partition, as well as information about the offset from which to read.
9. The consumer requests (pull) messages from the leader or followers.
10. The leader receives the offset data and passes it to ZooKeeper.

The following describes how leader election is performed in Kafka in the event of a failure of the partition leader.

Partition leader failure

1. If the leader fails, its ephemeral znode is deleted.
2. The controller receives information from its znode about the failure of the leader.
3. The controller selects a new leader from the number of ISR replicas.
4. Having received the ID of the new leader, the controller records the information about the new leader in the ZooKeeper storage and sends the data about the new leader to each broker that hosts a replica of this partition.

Further work with replicas, producer, and consumer is performed in the same way as described for the general case above.

This leader election is **clean** — it guarantees no data loss.

NOTE

After broker failures, uneven distribution can occur, and most partition replica leaders will end up on the same cluster node. In this case, you can manually rebalance the cluster using the `./usr/lib/kafka/bin/kafka-leader-election.sh` script with the `--election-type` option set to `PREFERRED` on the host where the Kafka service is installed.

The following describes the development of events in Kafka in the event of failure of the broker-follower.

Follower failure

1. The leader waits for a response about creating a replica (ack) from the follower after the specified time has elapsed (the response time is determined by the Kafka broker parameter `replica.lag.time.max.ms`, by default 30000 ms).
2. When time expires, the leader broker considers the replica to be out of sync and removes it from the ISR.
3. Information about ISR removal in broker controller and ZooKeeper. This follower ceases to be a candidate for the leader of the partition.

Further work with replicas, producer, and consumer is performed in the same way as described for the general case above.

CAUTION

If you enable the Kafka broker option `unclean.leader.election.enable`, new leaders for partitions will be created not from the ISR list. This will lose any out-of-sync messages that were sent to the old leader. In the case when there is no ISR, except for the leader that has become inaccessible, and it is impossible to return the leader to the network, this is the only option to leave the topic available to consumers. For a separate topic, this can be done using the `/usr/lib/kafka/bin/kafka-configs.sh` script using the `--add-config` option with the `unclean.leader.election.enable=true` parameter.

Message format in Kafka

Olga Semme

Contents

To Table of Contents

Record messages

Message format

Record batch format

Store a record on a broker

This article describes the message format in Kafka.

Message, **record** is a fixation in the system of an event that has occurred, including the state of an object, the value of a physical quantity or any other parameter that requires tracking, storage, or transmission to another system.

In the process of configuring components that take part in recording and reading messages, it is important to understand the message format and configure the parameters associated with it.

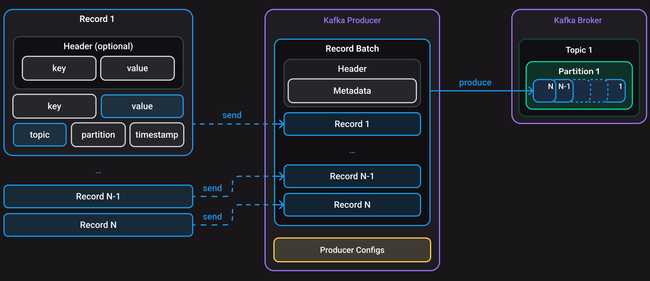
Record messages

Messages in Kafka are stored (buffered) in a **record batch** and then sent to the Kafka broker based on producer configurations. A batch can contain one or several messages. Messages from one batch are written to the same partition.

The batch message processing is regulated via the following parameters:

- max.in.flight.requests.per.connection – the maximum number of unconfirmed requests (number of batches) per entry that can be sent simultaneously (default value is `5`);
- linger.ms – the artificial delay time for sending batchs to combine records into a batch (default value is `0`);
- batch.size – the upper limit of the batch size in bytes (default value is `16384`).

The principle of batch message processing in Kafka is shown in the figure below.



Recording messages in Kafka

Message format

The core of a Kafka message is a **key/value** pair. Both the key and value are a variable length byte array. The value can be array of any data, including a certain number of key/value pairs. The key may be `null` or not specified at all.

Both the message key and value can be serialized in one of the following formats: json, avro, protobuf. Serialization occurs when the producer instance is started.

When creating a message, specify the following parameters along with the key/value pair:

- name of the **topic** where the message should be written;
- number of the **partition** in which the message should be written (may not be specified);
- **timestamp** specified by the user (may not be specified).

Based on the specified key and partition number when creating a message, one of the strategies for writing a message to a specific partition of the topic is determined:

- writing a message to the specified partition if this partition exists;
- recording messages with the same key into one partition, which is calculated using the hash function of the key (if the partition number is not specified and the key is present);
- writing a message to a partition selected **in a cyclic method (round-robin)** (neither partition nor key is specified).

If the user did not set a timestamp when creating a message, the message will be marked with the current time.

Using the **message header**, it is possible to add metadata for each message. The header is a key/value pair, where the key is always an encoded string, and the value can be array of any data, including a certain number of key/value pairs.

Creation of a record indicating the necessary Kafka parameters is performed via the ProducerRecord class.

Below is the disk format of the record and record header, indicating which data types are used for all values.

Disc recording format

```
length: varint
attributes: int8
    bit 0~7: unused
timestampDelta: varlong
offsetDelta: varint
keyLength: varint
key: byte[]
valueLen: varint
value: byte[]
Headers => [Header]
```

Disc header format

```
headerKeyLength: varint
headerKey: String
headerValueLength: varint
Value: byte[]
```

Record batch format

After the message is created, a KafkaProducer instance is launched, which performs:

1. Serialization of records in accordance with the specified key.serializer and value.serializer serialization parameters .
2. Partitioning records according to the selected strategy.
3. Buffering of records and batch sorting.
4. Performing the necessary compression of messages within the batch in accordance with the specified compression.type parameter.

A record batch contains one to several records and a **header** – the metadata section of the record batch. The header always contains 61 bytes. The header contains information about the type of compression used, the batch creation timestamp, the number of records, and other metadata.

Below is the disk format of the record batch header, indicating the data types used for all values.

Disk record batch header format

```
baseOffset: int64
batchLength: int32
partitionLeaderEpoch: int32
magic: int8 (current magic value is 2)
crc: uint32
attributes: int16
    bit 0~2:
        0: no compression
        1: gzip
        2: snappy
        3: lz4
        4: zstd
    bit 3: timestampType
    bit 4: isTransactional (0 means not transactional)
    bit 5: isControlBatch (0 means not a control batch)
    bit 6: hasDeleteHorizonMs (0 means baseTimestamp is not set as the delete horizon for compaction)
    bit 7~15: unused
lastOffsetDelta: int32
baseTimestamp: int64
maxTimestamp: int64
producerId: int64
producerEpoch: int16
baseSequence: int32
records: [Record]
```

Store a record on a broker

After writing to the Kafka broker, the message is saved with all the metadata that was written during the message writing process, while:

- The message on the broker has timestamp based on the log.message.timestamp.type broker parameter or message.timestamp.type topic parameter.
- Compression of recorded messages on the broker is performed based on the configured compression.type broker parameter or compression.type topic parameter. The message is stored in compressed form in the broker and decompressed only at the consumer.
- The message on the broker is saved in serialized form based on the serialization parameters set by the producer.

The RecordMetadata class returns the metadata of a record successfully written to the broker.

NiFi objects

Olga Semeno

Contents

To Table of Contents

- FlowFile
- Processor
- Connection
- Repositories
  - FlowFile Repository
  - Content Repository
  - Provenance Repository
- Process FlowFile
- Process Group
- Flow Controller
- Configure NiFi

This article describes the architecture of the core NiFi objects and their interaction.

FlowFile

FlowFile is a NiFi component, which is a data package, a unit of processing and transmission of information in NiFi.

Every FlowFile consists of two parts:

- **content** – an array of bytes that should be converted, extracted, or transmitted using NiFi tools. The content itself is stored in Content Repository. FlowFile tracks the path to where the content is stored.
- **attributes** – key/value pairs that contain information about the content and other data or parameters needed for proper processing.

Processor

Processor is the NiFi component that processes FlowFiles. The processor consists of a *logical block implemented in Java* in accordance with the specified purpose of the processor, and an interface with which it is possible to:

- configure processor configuration parameters that specify how data will be processed and define communication with external processes;
- manage a processor;
- perform diagnostics of the processor state.

Connection

Connection is a NiFi component that connects processors to transfer FlowFiles between them. In NiFi, a connection acts as a buffer or queue of FlowFiles. After FlowFiles are processed by one processor, they accumulate in the connection queue until they are accepted for processing by another processor. Queues allow processors to communicate at different speeds.

A NiFi connection consists of a FlowFiles queue and an interface through which it is possible to:

- configure connection parameters that determine how queues will be processed, including:
  - configure backpressure applications;
  - configure FlowFiles priority;
- manage connection;
- perform diagnostics queues.

Repositories

FlowFile Repository

The **FlowFile Repository** stores data of each FlowFile in the **write ahead log format (WAL)**. Each FlowFile change is a transactional operation, which guarantees the ability to recover without data loss in case of failures.

Information about storing the FlowFile in the repository is available by viewing FlowFile data.

The system periodically creates a snapshot of the current state of the FlowFile (attributes and path to the contents on the disk) and writes the snapshot data to the repository as a file with the extension *.partial*. These files are intermediate records of FlowFile processing transactions. When a **checkpoint** occurs, the last snapshot *.partial* is renamed to *snapshot* (the interval between checkpoints is controlled by the `nifi.flowfile.repository.checkpoint.interval` parameter). In case of failures, the *.partial* files written after the last *snapshot* file are deleted, and the last *snapshot* file is considered the recovery point for the node. If the *snapshot* file does not exist, the last *.partial* is renamed to *snapshot*.

To increase performance, processed FlowFiles can be stored in JVM memory, which can lead to data loss during failures. Given the use of WAL, risks can be minimized by setting parameters according to project requirements.

The process of NiFi working with JVM memory (data storage and file swapping during processing) can be adjusted via the FlowFile Repository parameters:

- `nifi.flowfile.repository.implementation` – determines the storage location of processed files;
- `nifi.flowfile.repository.always.sync` – determines whether any change in the repository should be synchronized with disk;
- `nifi.queue.swap.threshold` – queue threshold (number of FlowFiles) at which NiFi starts swapping FlowFile information to disk.

Content Repository

The content of each FlowFile is stored in a **Content Repository**. A repository consists of a set of files on disk. Each of the files has its own **identifier**. The contents of each FlowFile are stored in a file at a certain **offset**.

Below is the hierarchy of data storage in Content Repository:

```
base path
      |__Container 0
                  |__Section 1
                              |__identifier
                                          |__offset
                  |__Section 2
                              |__identifier
                                          |__offset
                  |__________ ...
      |__Container 1
                  |__Section 1
                  |__Section 2
                  |__ ...
      |__...
```

This repository uses the **immutability** and **copy-on-write** paradigms. This means that the data written to the repository does not change and data location on disk also does not change.

Content location is tracked by Java **Resource Claims** objects, which specify the path to the content on disk (Container → Section → identifier → offset) and pass this data as a **Content Claim** object, which is part of the FlowFile metadata when stored in FlowFile Repository.

Resource Claim tracks the same path for multiple FlowFiles that have the same content. If the processor changes content, the new data is written to a new fragment on disk and Resource Claim begins tracking the new data storage path for transmission to Content Claim.

If content references created by Resource Claim objects are not used in any Content Claim, the content will be deleted or archived (archiving is enabled by the `nifi.content.repository.archive.enabled` parameter and the retention time of archived data by the `nifi.content.repository.archive.max.retention.period` parameter).

Tracking of unused links occurs every time a checkpoint is reached in the **FlowFile Repository**.

Provenance Repository

The **Provenance Repository** stores snapshots of data of every FlowFile each time a FlowFile has been modified by the processor. This allows you to track the chronology of FlowFile changes.

Such a snapshot in NiFi is called **Provenance Event**.

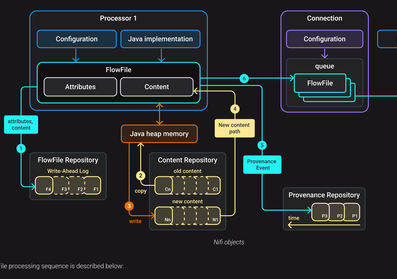
All provenance events are displayed on the Data Provenance page of the global UI menu.

Old versions of FlowFile content may be deleted before the origin events referencing them are deleted. In this case, the user will have access to FlowFile metadata and attributes for analyzing the flow of data.

The storage period for data provenance information is controlled by the `nifi.provenance.repository.max.storage.time` parameter.

Process FlowFile

The figure below shows how data is processed and modified in NiFi.



Nifi objects

The FlowFile processing sequence is described below:

1. Snapshots of FlowFile data are written to FlowFile Repository (`F1–F4`) at a certain frequency.
2. If the processor logic expects the content to change, FlowFile requests the content (`C1–Cn`) from Content Repository according to Content Claim and reads it into JVM memory.
3. The changed contents (`N1–Nn`) are written to a new disk fragment.
4. The Resource Claim object starts tracking the new content path and updates the data in the Content Claim.
5. A new snapshot of the FlowFile metadata and attributes (provenance event) is written to Provenance Repository.
6. FlowFile enters the connection queue for processing in the next processor. A new snapshot is written to the FlowFile Repository with updated FlowFile data.

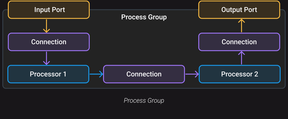
If a failure occurs during one of the steps, the system will revert to the state corresponding to the latest *snapshot* file in the FlowFile Repository. In this case, new data written to the disk will be deleted, since the link does not have time to be updated and access to old data remains unchanged due to copy-on-write and immutability.

Process Group

Process Group is a NiFi component that allows you to combine the functionality of different processors and other NiFi elements into one component, which is essentially a new processor. The process group must contain an **Input Port**, **Output Port** and can contain any number of NiFi elements and connections between them.

A process group has an interface that allows you to:

- configure process group configuration parameters;
- manage process group;
- perform diagnostics of the process group state.



Process Group

Flow Controller

Flow Controller is an entity that unites all processes in NiFi. It is a broker for the operation of processors, which controls the operation of Nifi extensions and plans the allocation of resources for the operation of processors and extensions.

The flow controller allows you to add controller services.

Controller Services – services intended for setting up connections to external systems and distributing to multiple processors or process groups.

Configuring the flow controller and controller services is done on the Controller Settings page of the global user interface menu.

Configure NiFi

After adding and installing the ***NiFi*** service as part of ADS cluster, you can configure NiFi parameters on the configuration page of the ***NiFi*** service via ADCM.

To configure the parameters of the NiFi service, set the ***Show advanced*** switch to active, expand the ***nifi.properties*** node and enter new values for the parameters. To change NiFi parameters that are not available in the ADCM interface, use the ***Add key,value*** field in the ***nifi.properties*** node. Select ***Add property*** and enter the name of the parameter and its value.

Processor types in NiFi

A. Vigil Dennies

Contents

To Table of Contents

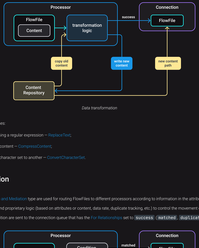
Data transformation
Routing and mediation
Database access
Attribute extraction
System interaction
Data Ingestion
Data egress/Sending data
Splitting and aggregation
HTTP
Amazon Web Services

This article describes the existing types of NiFi processors.

> **NOTE**
> For information about the architecture of the core NiFi objects and their interaction, you can refer to the NiFi objects article.

Data transformation

Processors belonging to the Data Transformation type transform FlowFile contents.
The processor copies the content from Content Repository, transforms the data according to its own logic, writes the new data to the Content Repository, and stores a link to the changed data in the FlowFile.
FlowFiles whose contents were successfully transformed are sent to the connection queue, which has the For Relationships set to `success`.



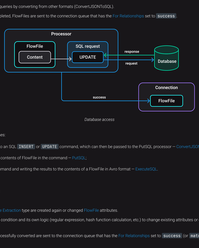
Data transformation

Examples of processors and their uses:

- replacing all or part of the text using a regular expression → ReplaceText;
- compressing or decompressing content → CompressContent;
- converting characters from one character set to another → ConvertCharacterSet.

Routing and mediation

Processors belonging to the Routing and Mediation type are used for routing FlowFiles to different processors according to information in the attributes or contents of those FlowFiles.
The processor uses user-specified and generated logic (based on attributes or content, data rate, duplicate tracking, etc.) to control the movement of FlowFiles.
FlowFiles that match the given condition are sent to the connection queue that has the For Relationships set to `success` `matched` `duplicate` (depending on the processor).



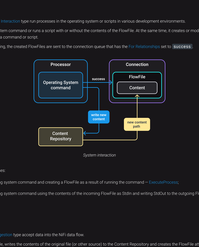
Routing and mediation

Examples of processors and their uses:

- FlowFiles throughput throttling based on configured speed rate → ControlRate;
- tracking duplicate FlowFiles based on the user-specified criteria → DetectDuplicate;
- routing FlowFiles based on their attributes using NiFi Expression Language→ RouteOnAttribute.

Database access

Processors belonging to the Database Access type are designed to operate with databases.
The processor executes an SQL request, depending on its own logic, and can use the contents of the incoming FlowFile as the contents of the request, or write the result of the query as the contents of the outgoing FlowFile.
Also, some processors can prepare queries by converting from other formats (ConvertJSONToSQL).
When a request is successfully completed, FlowFiles are sent to the connection queue that has the For Relationships set to `success`.



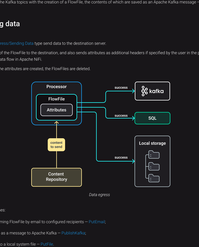
Database access

Examples of processors and their uses:

- converting a JSON document into an SQL `INSERT` or `UPDATE` command, which can then be passed to the PutSQL processor → ConvertJSONToSQL;
- updating the database using the contents of FlowFile in the command → PutSQL;
- executing an SQL `SELECT` command and writing the results to the contents of FlowFile in Avro format → ExecuteSQL.

Attribute extraction

Processors belonging to the Attribute Extraction type are created again or changed FlowFile attributes.
The processor uses a user-specified condition and its own logic (regular expression, hash function calculation, etc.) to change existing attributes or extract attributes from the contents of the FlowFile.
FlowFiles whose attributes were successfully converted are sent to the connection queue that has the For Relationships set to `success` `matched` (depending on the processor).



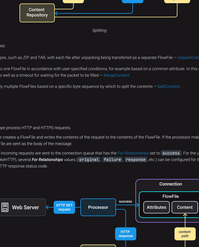
Attribute extraction

Examples of processors and their uses:

- extracting attributes from the FlowFile content text according to regular expressions specified in custom processor properties → ExtractText;
- updating FlowFile attributes using NiFi Expression Language or based on user-specified regular expressions → UpdateAttribute;
- calculating a cryptographic hash value for the FlowFile content using the specified algorithm and writing the value to the output attribute → CryptographicHashContent.

System interaction

Processors belonging to the System Interaction type run processes in the operating system or scripts in various development environments.
The processor runs an operating system command or runs a script with or without the contents of the FlowFile. At the same time, it creates or modifies a FlowFile, the contents of which record the result of executing a command or script.
Upon successful command processing, the created FlowFiles are sent to the connection queue that has the For Relationships set to `success`.



System interaction

Examples of processors and their uses:

- running a user-specified operating system command and creating a FlowFile as a result of running the command → ExecuteProcess;
- running a user-specified operating system command using the contents of the incoming FlowFile as its Stdin and writing StdOut to the outgoing FlowFile → ExecuteStreamCommand.

Data ingestion

Processors belonging to the Data Ingestion type accept data into the NiFi data flow.
The processor creates a new FlowFile, writes the contents of the original file (or other source) to the Content Repository and creates the FlowFile attributes. Typically, such processors are the starting point of the data flow in Apache NiFi.
If contents are successfully written and attributes are created, FlowFiles are sent to the connection queue that has For Relationships set to `success`.

Data ingestion

Examples of processors and their uses:

- transferring the contents of a file from a local drive (or network drive) as the contents of the created FlowFile and then deleting the original file → GetFile;
- subscribing to one or more Apache Kafka topics with the creation of a FlowFile, the contents of which are saved as an Apache Kafka message → ConsumeKafka.

Data egress/Sending data

Processors belonging to the Data Egress/Sending Data type send data to the destination server.
The processor passes the contents of the FlowFile to the destination, and also sends attributes as additional headers if specified by the user in the processor settings. Typically, such processors are the endpoint of the data flow in Apache NiFi.
If the data is sent successfully and the attributes are created, the FlowFiles are deleted.

Data egress

Examples of processors and their uses:

- sending the contents of the incoming FlowFile by email to configured recipients → PutEmail;
- sending the contents of FlowFile as a message to Apache Kafka → PublishKafka;
- writing the contents of FlowFile to a local system file → PutFile.

Splitting and aggregation

Processors belonging to the Splitting and Aggregation type split or aggregate contents of FlowFile.
When splitting, the processor creates new FlowFiles, the contents of which are written to the data obtained as a result of the split. The division occurs in accordance with a user-specified condition and the processors own logic. The attributes for each new FlowFile are usually data describing the resulting segment of content — size, ID, number of lines, etc.
During aggregation, the processor creates a new FlowFile, the contents of which combine the contents of all incoming files. The attributes for the new FlowFile are data describing the file resulting from the merge: file name, number of merged FlowFiles, packet lifetime before sending, etc.
The processor writes the contents of new FlowFiles to the Content Repository and stores a reference to them in each FlowFile.
When splitting or aggregating content, new FlowFiles are sent to the connection queue that has For Relationships set to `success` `splits` `segments` `merged` — depending on the processor).

Splitting

Examples of processors and their uses:

- Unpacking archives of various types, such as ZIP and TAR, with each file after unpacking being transferred as a separate FlowFile → UnpackContent;
- Combining multiple FlowFiles into one FlowFile in accordance with user-specified conditions, for example based on a common attribute. In this case, the minimum and maximum size of each packet can be set, as well as a timeout for waiting for the packet to be filled → MergeContent;
- Splitting a FlowFile into potentially multiple FlowFiles based on a specific byte sequence by which to split the contents → SplitContent.

HTTP

Processors belonging to the HTTP type process HTTP and HTTPS requests.
For incoming requests, the processor creates a FlowFile and writes the contents of the request to the contents of the FlowFile. If the processor makes an outgoing request (PUT, POST, or PATCH), the contents of the FlowFile are sent as the body of the message.
Successfully created FlowFiles from incoming requests are sent to the connection queue that has the For Relationships set to `success`. For the universal processor responsible for receiving and sending requests (InvokeHTTP), several **For Relationships** values (`original` `failure` `response`, etc.) can be configured for the connection, where FlowFiles can be distributed in depending on the HTTP response status code.

HTTP requests

Examples of processors and their uses:

- running an HTTP (or HTTPS) server and listening for incoming connections → ListenHTTP;
- interacting with a custom HTTP endpoint using the GET method, where the response body becomes the content of the generated FlowFile → InvokeHTTP.

Amazon Web Services

Processors belonging to the Amazon Web Services type are responsible for interacting with the web system Amazon services.
The processor performs actions in accordance with its own logic — retrieves the contents of AWS objects or messages, sends the contents of FlowFile as the contents of an AWS object or notifications. When saving data from AWS to the contents of a FlowFile, parameters that describe the data are saved as attributes: identifier, hash sum, etc.
When a request is successfully completed, FlowFiles are sent to a connection queue that has the For Relationships set to `success`.

AWS

Examples of processors and their uses:

- retrieving the contents of an object stored in Amazon Simple Storage → FetchS3Object;
- writing the contents of FlowFile to to an Amazon S3 object → PutS3Object.

ADS supported services

Olga Semme

To Table of Contents

This article describes a list of services included into the current version of ADS. The ***Version*** column contains references to the release notes of each component.

Service	Component	Version
Kafka	Kafka Broker	3.9.1_arenadata1 Kafka Tiered Storage 1.0.0_arenadata1
	Kafka Controller	3.9.1_arenadata1
	Kafka Cruise Control	3.9.1_arenadata1
Kafka Connect	Kafka Connect Worker	3.9.1_arenadata1 Debezium Connector 3.1 Iceberg Sink Connector 1.6.1_arenadata3
Kafka REST Proxy	Kafka REST Proxy	7.9.1
ksqlDB	ksqlDB Client	7.9.1
	ksqlDB Server	7.9.1
Schema-Registry	Schema-Registry	7.9.1
NiFi	NiFi Registry	1.28.1_arenadata2
	NiFi Server	1.28.1_arenadata2
NiFi2	NiFi2 Registry	2.5.0_arenadata1
	NiFi2 Server	2.5.0_arenadata1
MiNiFi	MiNiFi Agent	1.28.1_arenadata2
	MiNiFi C2 Server	1.28.1_arenadata2
ZooKeeper	ZooKeeper Server	3.8.4_arenadata1

Version compatibility matrix

Olga Semme

To Table of Contents

ADS version	ADPS min version	ADPS max version	ADCM min version
3.9.1.1_b1	1.2.1_b1	—	2.7.0
3.9.0.1_b1	1.2.1_b1	—	2.6.0
3.7.2.1_b1	1.2.0_b1	1.2.1_b1	2.5.0
3.6.2.2_b1	1.2.0_b1	1.2.1_b1	2.3.0
3.6.2.1_b1	1.2.0_b1	1.2.1_b1	2.2.0
3.3.2.2_b1	1.1.0_b1	1.2.1_b1	2.0.0
3.3.2.1_b1	1.1.0_b1	1.2.1_b1	2023.10.10.08
2.8.1.1_b1	1.0.3_b1	1.0.4_b6	2023.10.10.08
1.8.2_b1	1.0.3_b1	1.0.4_b6	2023.06.14.16
1.8.1_b1	1.0.3_b1	1.0.4_b6	2022.12.07.10
1.8.0_b1	1.0.3_b1	1.0.4_b6	2022.12.07.10
1.7.2_b2	1.0.3_b1	1.0.4_b6	2022.04.18.13