ADCM

Arenadata Orchestrator

Arenadata Orchestrator (ADO) is a platform for setting up and operating data pipelines in a production environment. The central component of the platform is Apache Airflow, an open source tool used to programmatically create, schedule, and monitor process and task sequences (DAGs).

Version: CURRENT Language: EN

Contents

1. Concepts
   - Airflow architecture

2. Planning guide
   - Network requirements
   - Software requirements

3. Get started
   - Installation
     - Online installation
       - Install ADCM
       - Prepare hosts
       - Install ADO cluster
         - Create a cluster
         - Add services
         - Add hosts to a cluster
         - Add components
         - Configure services
         - Configure a cluster
         - Install a cluster
       - Install monitoring
     - Offline installation
       - Install ADCM
       - Prepare hosts
       - Install Enterprise Tools cluster
         - Create a cluster
         - Add services
         - Add hosts to a cluster
         - Add components
         - Install a cluster
       - Install ADO cluster
         - Create a cluster
         - Add services
         - Add hosts to a cluster
         - Add components
         - Configure services
         - Configure a cluster
         - Import ET settings
         - Install a cluster
       - Install monitoring
   - Connect to Airflow
     - CLI
     - REST API

4. Cluster maintenance
   - Cluster management via ADCM
     - Cluster actions
       - Add/Remove components
       - Check
       - Manage SSL
       - Manage Kerberos
       - Reinstall status-checker
       - Start
       - Stop
     - Service actions
       - ADPG
       - Airflow
       - Monitoring
       - Redis
   - Security
     - Integration with HashiCorp Vault to store secrets
   - Logging
   - Manage dependencies via ADCM
   - Performance tuning

5. Web user interface
   - Overview
   - Work with DAGs
     - Create a simple DAG
     - Work with TaskFlow
     - Use sensors
     - Customize DAG scheduling
     - Add custom operators and hooks
     - Generate DAGs dynamically
     - Use templates

6. References
   - Configuration parameters
   - Airflow CLI
     - Miscellaneous commands
       - cheat-sheet
       - dag-processor
       - info
       - kerberos
       - plugins
       - rotate-fernet-key
       - scheduler
       - standalone
       - sync-perm
       - triggerer
       - version
       - webserver
     - Celery commands
       - flower
       - stop
       - worker
     - View configuration
       - get-value
       - list
     - Manage connections
       - add
       - delete
       - export
       - get
       - import
       - list
       - test
     - Manage DAGs
       - backfill
       - delete
       - details
       - list
       - list-import-errors
       - list-jobs
       - list-runs
       - next-execution
       - pause
       - report
       - reserialize
       - show
       - show-dependencies
       - state
       - test
       - trigger
       - unpause
     - Database operations
       - check
       - check-migrations
       - clean
       - downgrade
       - drop-archived
       - export-archived
       - init
       - reset
       - shell
       - upgrade
     - Manage jobs
       - check
     - Manage KubernetesExecutor
       - cleanup-pods
       - generate-dag-yaml
     - Manage pools
       - delete
       - export
       - get
       - import
       - list
       - set
     - Display providers
       - auth
       - behaviours
       - get
       - hooks
       - links
       - list
       - logging
       - secrets
       - widgets
     - Manage roles
       - add-perms
       - create
       - del-perms
       - delete
       - export
       - import
       - list
     - Manage tasks
       - clear
       - failed-deps
       - list
       - render
       - run
       - state
       - states-for-dag-run
       - test
     - Manage users
       - add-role
       - create
       - delete
       - export
       - import
       - list
       - remove-role
     - Manage variables
       - delete
       - export
       - get
       - import
       - list
       - set

7. Release notes
   - ADO releases
   - Supported services

Airflow architecture

Daria Barysheva

ADO

Contents

To Table of Contents

Data model
- DAG (Directed acyclic graph)
- Task
- Operator

Idempotence

Architecture
- ADO-specific components

Airflow is a platform that allows to develop, plan, run, and monitor complex workflows. It fits perfectly with ETL/ELT processes and also can be useful if you need to periodically run any processes and monitor their execution. In order to describe processes, Aiflow uses the Python programming language.

Data model

The Airflow data model uses the following basic objects.

DAG (Directed acyclic graph)

It is the core concept of Airflow representing a semantic association of Tasks that should be performed in the strictly defined order — according to the specified schedule. Visually, the DAG looks like a directed acyclic graph, i.e. a graph without cyclic dependencies.

The simple DAG shown below consists of three Tasks. You can see that it defines the order of running Tasks and dependencies between them. Using Airflow you can also configure the scheduler for this DAG, actions in case of failure, number of retries, etc. The DAG does not care about what happens with data — this function is performed by Tasks themselves.

A simple DAG

Task

A task is a DAG's node that describes operations to be applied to the data. For example, loading data from external sources, aggregating and indexing, removing duplicates, and other typical ETL operations.

Operator

This is a template for completing Tasks. While Tasks describe what actions to perform with data, Operators define how to perform these actions at the code level — using Python functions, Bash scripts, etc.

Airflow supports a lot of built-in core Operators. Besides that, you can use a huge set of community provider packages. Also, you can create your own Operators by extending the `BaseOperator` class.

A separate type of Operators — **Sensors** — allow users to define a reaction to certain events: the occurrence of a specific time, loading some data, starting of other DAGs/Tasks, etc.

Examples of useful Operators are listed below.

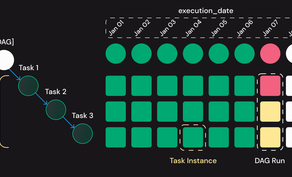
Airflow Operators

Operator	Description
PythonOperator	Execution of the Python code
BranchPythonOperator	Execution of the code depending on the specified conditions
BashOperator	Running Bash scripts
SimpleHttpOperator	Sending HTTP requests
PostgresOperator	Sending queries to a PostgreSQL database
MySqlOperator	Sending queries to a MySQL database
SqlSensor	Verifying the SQL query execution
DockerOperator	Running a Docker container for the Task
KubernetesPodOperator	Creating a separate Kubernetes pod for the Task
EmailOperator	Sending Emails
DummyOperator	The "empty" Operator that is used for grouping Tasks

Idempotence

One of the most important concepts underlying AirFlow is storing information about every DAG's start time according to the specified schedule. For example, if you specify that the DAG should start at 01.01.2022 00:00:00 once a day, AirFlow will store information about running of this DAG using the following timestamps: 01.01.2022 00:00:00, 02.01.2022 00:00:00, 03.01.2022 00:00:00, and so on. The timestamps used in this example are called **execution_date**, the corresponding DAG instances — **DAG Runs**, and the Task entities associated with specific DAG Runs — **Task instances**.

This concept is very important for maintaining the idempotence: start or restart of the Task with the same dataset for some date in the past allows you to reproduce the results obtained earlier. In addition, it is possible to run Tasks of one DAG for different timestamps simultaneously.



The concept of Task instances and Dag Runs

Architecture

Airflow architecture is based on the following components:

- **Web Server**. Provides a convenient user interface that allows to run and monitor DAGs and Tasks, track their schedule, execution status, and so on.
- **Metadata Database**. A shared database that is used by Scheduler, Executor, and Web Server components to store the system metadata: global variables, data source connection settings, statuses of Task Instances and DAG Runs, etc. As this repository is based on the SQLAlchemy library, it requires the installation of a compatible database. For example, MySQL or PostgreSQL.
- **Scheduler**. A service responsible for planning in Airflow. It handles both triggering scheduled workflows and submitting Tasks to the Executor for running.
  The Scheduler analyzes the results of DAGs parsing and checks the existence of Tasks ready to run regularly (by default, once a minute). When the predefined conditions for Tasks running are met, the scheduler initializes Task Instances. To perform active Tasks, the scheduler uses the Executor specified in the settings.
  For certain Metadata Database versions (PostgreSQL 9.6+ and MySQL 8+), using of several schedulers is supported. It helps to provide the maximum fault tolerance.
- **Executor**. A mechanism that runs Task Instances. It works in conjunction with the Scheduler. In the default Airflow installation, the Executor runs everything inside the Scheduler (within the same process). But for production needs it is recommended to use Executors that push Tasks execution out to Workers.

Executor types

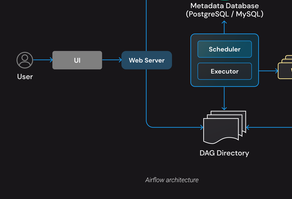
Type	Description
SequentialExecutor	Launches the Tasks sequentially and suspends the Scheduler for the duration of their execution. That is why this Executor is recommended only for testing — it is not suitable for a production environment
LocalExecutor	Starts a new child process for each Task, that allows to process multiple Tasks in parallel. It perfectly simulates a production environment in a test environment, but it is also not recommended for real use since the low fault tolerance: in case of a failure on the machine where Executor is running, the Task is not transferred to other nodes
CeleryExecutor	Allows to have multiple Workers running on different nodes. The executor uses Celery and requires additional installation of the message broker. For example, Redis or RabbitMQ. When the load increases, it is enough to connect an additional Worker. If any Worker fails, its work is transferred to the rest of the nodes
DaskExecutor	It is similar to CeleryExecutor, but instead of the Celery for parallel computing it uses the Dask library
KubernetesExecutor	Launches a new Worker on a separate pod in Kubernetes for each Task Instance
CeleryKubernetesExecutor	Allows to run CeleryExecutor and KubernetesExecutor simultaneously. The specific type is selected depending on the status of the Tasks queue
DebugExecutor	Used to run and debug pipelines from the IDE

- **Worker**. A separate process in which Tasks are executed. The location of the Worker is determined by the selected type of the Executor.
- **DAG Directory**. A folder of DAG files that can be read by the Scheduler, Executor, and any Worker that the Executor has.

IMPORTANT

All DAGs in the form of python-scripts should be located in the DAG Directory. Web UI is used only for monitoring DAGs, not for adding or editing.

The high-level architecture view of Airflow is shown below. Depending on the Executor type, this scheme can contain additional components: for example, the Message Queue between the Executor and its Workers for the CeleryExecutor. However, in such particular cases you can still think of an Executor and its Workers as a single logical component that handles the Tasks execution.



Airflow architecture

NOTE

For more details about Airflow concepts, see Airflow documentation.

ADO-specific components

The Airflow architecture in ADO also includes several additional components that extend the base functionality and provide better scalability and observability.

- **DAG Processor**. A separate component responsible for parsing DAG files. It reads DAG definitions from the directory and prepares them for scheduling. In newer versions of Airflow, this logic is no longer part of the Scheduler, which allows more flexible and scalable setups.
- **Triggerer**. A component that handles deferred tasks and asynchronous operations. It is used for operators that wait for external events without occupying worker slots. This helps to reduce resource usage and improves cluster efficiency in long-running workflows.
- **Broker (Redis)**. A message broker used for communication between the Scheduler and Workers when using CeleryExecutor. In ADO, Redis is used as the default broker implementation. It is responsible for queueing tasks and delivering them to workers.
- **Airflow Flower**. A web-based monitoring tool for Celery. It provides visibility into task execution, worker status, and queues. Flower can be used to troubleshoot task execution and monitor cluster activity in real time.

Network requirements

Elena Kostyuchenko

Contents

To Table of Contents

ADCM

Arenadata PostgreSQL

Airflow2

Redis

Monitoring

The tables below list ports of ADO service components and servers. The ***Config*** column contains a file name where the port number can be redefined.

ADCM

Component/Server	Port	Protocol	Description
ADCM	22	TCP	ADCM connection port

Arenadata PostgreSQL

Component/Server	Port	Protocol	Description	Parameter	Config
Arenadata PostgreSQL	5432	TCP	PostgreSQL port for DB connections	port	postgresql.conf

Airflow2

Component/Server	Port	Protocol	Description	Parameter	Config
Webserver	8080/8080	HTTP/HTTPS	Airflow Server Web UI port	web_server_port (***server_port*** in ADCM)	airflow.cfg
Worker	8793/8793	HTTP/HTTPS	Celery Worker API port	worker_log_server_port (***worker_port*** in ADCM)	airflow.cfg
Flower	5555/5555	HTTP/HTTPS	Flower Web UI port	flower_port	airflow.cfg
Metastore	5432	TCP	PostgreSQL service port	sql_alchemy_conn, result_backend	airflow.cfg

Redis

Component/Server	Port	Protocol	Description
Redis	6379	TCP	Redis port for DB connections

Monitoring

Component/Server	Port	Protocol	Description
Graphana	11210	TCP	Port to send metrics
Prometheus	11200	TCP	Port to send metrics
Node Exporter	11203	TCP	Port to send metrics

Software requirements

Elena Kostyuchenko

To Table of Contents

Component	Requirements
Platform	Intel x86_64
OS	■ CentOS 7.9. ■ RHEL 7.9. ■ AltLinux 8 SP (ADO Enterprise edition). ■ AltLinux 10 SP (ADO Enterprise edition, starting with the version 2.10.5.1). ■ AstraLinux 1.7.6 "Orel" SE with Axiom JDK (ADO Enterprise edition). The `LC_TIME` locale should be set to `en_US.utf8`. ■ AstraLinux 1.7.6 "Voronezh" SE with Axiom JDK (ADO Enterprise edition, starting with the version 2.6.3.2). ■ Ubuntu 22.04.2 LTS. ■ RED OS 7.3 Certified edition (ADO Enterprise edition).
Browser	■ Internet Explorer ■ Firefox ■ Google Chrome ■ Safari
Software	■ RPM + YUM / DPKG + APT ■ scp ■ curl ■ unzip ■ tar
Python	By default, ADO includes Python 3.10. Interpreter path: `/opt/python3.10/bin/python3.10`
Metastore	■ PostgreSQL: 12, 13, 14, 15, 16 ■ MySQL: 8.0, Innovation ■ SQLite: 3.15.0+ We recommend using ADPG (Postgres). ADPG 16 is available as an ADO service
Broker	Airflow with the Celery executor supports RabbitMQ, Redis and experimentally a SQLAlchemy database. Redis is available as a service as part of ADO
Arenadata Cluster Manager (ADCM)	Version 2.5.0 and higher

Installation

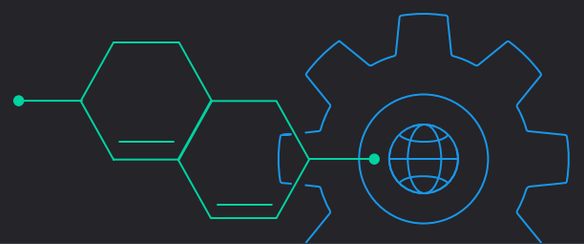
To Table of Contents

This tutorial guides you through the process of installing an Arenadata Orchestrator (ADO) 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 ADO via ADCM in two ways.

Online installation

on hosts with access to the Internet



Offline installation

on hosts without access to the Internet

Online installation

To Table of Contents

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

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

Step 1.

Install ADCM

Step 2.

Prepare hosts

Step 3.

Install ADO cluster

Step 4.

Install monitoring

Install AZUM

Table of Contents

Contents

- Step 1. Check prerequisites
  - Software and Hardware requirement
  - Install Docker
  - Install Git
  - Install Node.js
  - Check ports that the Server will be communicating
- Step 2. Install AZUM
- Step 3. Start AZUM
  - Check Web Browser compatibility
  - Check the AZUM server interface
  - Set the AZUM URL
- Step 4. Update AZUM
  - Online update
  - Offline update

Step 1. Check prerequisites

Software and Hardware requirement

Before installing AZUM, please check the software and hardware requirement. Check as well.

Software requirement

- Docker
- Git
- Node.js
- Python 3.7 or later
- Python 2.7 (optional)
- Python 3.7 (optional)

Hardware requirement

- CPU: 4 cores or more
- RAM: 8GB or more
- Disk: 100GB or more

Check ports that the Server will be communicating

Please check the ports that the Server will be communicating. For example, if you want to use the default port, 443, please check if the port is not used by other applications.

Install Docker

Install Docker

Please refer to the Docker installation guide for your OS.

Check if Docker has the right version

Please check if the Docker version is 19.03.0 or later and **Docker Compose** is installed.

```
docker -v
docker-compose -v
```

Install Git

Install Git

Please refer to the Git installation guide for your OS.

Check if Git has the right version

Please check if the Git version is 2.0 or later.

```
git --version
```

Install Node.js

Install Node.js

Please refer to the Node.js installation guide for your OS.

Check if Node.js has the right version

Please check if the Node.js version is 12.0 or later.

```
node -v
```

Install Python

Install Python

Please refer to the Python installation guide for your OS.

Check if Python has the right version

Please check if the Python version is 3.7 or later.

```
python3 -V
```

Install Python 2.7 (optional)

Install Python 2.7

Please refer to the Python installation guide for your OS.

Check if Python has the right version

Please check if the Python version is 2.7.

```
python2 -V
```

Install Python 3.7 (optional)

Install Python 3.7

Please refer to the Python installation guide for your OS.

Check if Python has the right version

Please check if the Python version is 3.7.

```
python3.7 -V
```

Install Docker CE

Install Docker CE

Please refer to the Docker installation guide for your OS.

Check if Docker has the right version

Please check if the Docker version is 19.03.0 or later and **Docker Compose** is installed.

```
docker -v
docker-compose -v
```

Install Git

Install Git

Please refer to the Git installation guide for your OS.

Check if Git has the right version

Please check if the Git version is 2.0 or later.

```
git --version
```

Install Node.js

Install Node.js

Please refer to the Node.js installation guide for your OS.

Check if Node.js has the right version

Please check if the Node.js version is 12.0 or later.

```
node -v
```

Install Python

Install Python

Please refer to the Python installation guide for your OS.

Check if Python has the right version

Please check if the Python version is 3.7 or later.

```
python3 -V
```

Install Python 2.7 (optional)

Install Python 2.7

Please refer to the Python installation guide for your OS.

Check if Python has the right version

Please check if the Python version is 2.7.

```
python2 -V
```

Install Python 3.7 (optional)

Install Python 3.7

Please refer to the Python installation guide for your OS.

Check if Python has the right version

Please check if the Python version is 3.7.

```
python3.7 -V
```

Disable SELinux

Add

Create the file /etc/sysctl.d/99-azum.conf and add the following contents.

```
vm.max_map_count = 262144
```

To apply the changed configuration file:

```
sysctl -p /etc/sysctl.d/99-azum.conf
```

Check if SELinux is disabled

```
getenforce
```

If SELinux is disabled, Disabled is displayed.

Stop Firewall

Add

Stop the firewall.

```
systemctl stop firewalld
```

Disable the firewall from starting automatically.

```
systemctl disable firewalld
```

Check if the firewall is stopped.

```
systemctl status firewalld
```

If the firewall is stopped, inactive (dead) is displayed.

Allow access to the Docker TLS default port (2376/tcp)

Add

Allow access to the Docker TLS default port (2376/tcp).

```
firewall-cmd --zone=public --add-port=2376/tcp --permanent
firewall-cmd --reload
```

Allow access to the Docker TLS default port (2376/tcp) and configure Docker to start on boot with sudo privileges

Add

Allow access to the Docker TLS default port (2376/tcp).

```
sudo firewall-cmd --zone=public --add-port=2376/tcp --permanent
sudo firewall-cmd --reload
```

Configure Docker to start on boot with sudo privileges.

```
sudo systemctl enable docker
```

Stop Firewall

Add

Stop the firewall.

```
sudo systemctl stop firewalld
```

Disable the firewall from starting automatically.

```
sudo systemctl disable firewalld
```

Check if the firewall is stopped.

```
sudo systemctl status firewalld
```

If the firewall is stopped, inactive (dead) is displayed.

Install Git

Add

Please refer to the Git installation guide for your OS.

Check if Git has the right version

Please check if the Git version is 2.0 or later.

```
git --version
```

Install Node.js

Add

Please refer to the Node.js installation guide for your OS.

Check if Node.js has the right version

Please check if the Node.js version is 12.0 or later.

```
node -v
```

Install Python

Add

Please refer to the Python installation guide for your OS.

Check if Python has the right version

Please check if the Python version is 3.7 or later.

```
python3 -V
```

Install Python 2.7 (optional)

Add

Please refer to the Python installation guide for your OS.

Check if Python has the right version

Please check if the Python version is 2.7.

```
python2 -V
```

Install Python 3.7 (optional)

Add

Please refer to the Python installation guide for your OS.

Check if Python has the right version

Please check if the Python version is 3.7.

```
python3.7 -V
```

Disable SELinux

Add

Create the file /etc/sysctl.d/99-azum.conf and add the following contents.

```
vm.max_map_count = 262144
```

To apply the changed configuration file:

```
sysctl -p /etc/sysctl.d/99-azum.conf
```

Check if SELinux is disabled

```
getenforce
```

If SELinux is disabled, Disabled is displayed.

Stop Firewall

Add

Stop the firewall.

```
systemctl stop firewalld
```

Disable the firewall from starting automatically.

```
systemctl disable firewalld
```

Check if the firewall is stopped.

```
systemctl status firewalld
```

If the firewall is stopped, inactive (dead) is displayed.

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

When creating ADO hosts via ADCM, you should specify their FQDNs.

Create a cluster

Elena Kostyuchenko, 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.

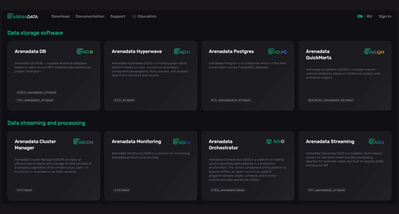
The steps for adding a cluster to ADCM are listed below.

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 Orchestrator***.



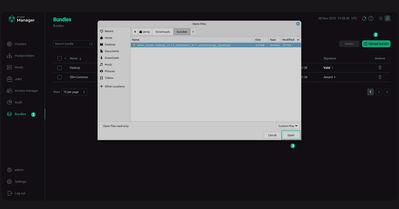
Switch to the bundle download page

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

Step 2. Upload a cluster bundle to ADCM

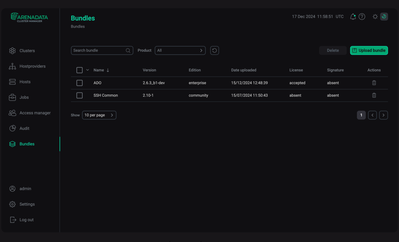
To upload a bundle to ADCM, follow the steps:

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

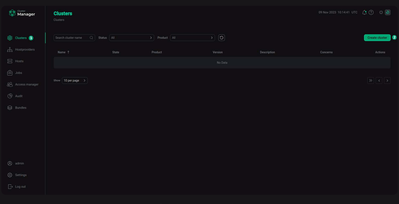
Step 3. Create a cluster based on the uploaded bundle

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.

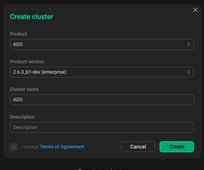
The steps for cluster creation are given below:

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



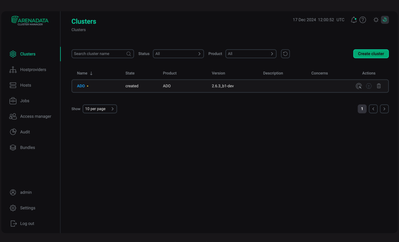
Switch to cluster creation

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

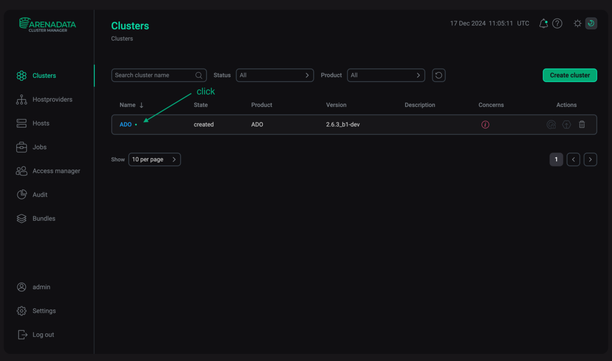
Add services

Elena Kostyuchenko, Daria Barysheva

To Table of Contents

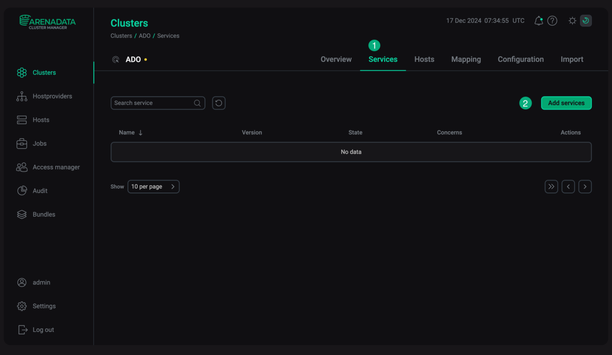
In ADCM, a **service** means a software that performs some function. Examples of services for ADO clusters: Airflow, ADPG, etc. The steps for adding services to a cluster are listed below:

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



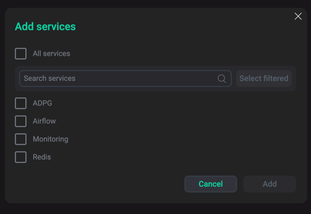
Select a cluster

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***.



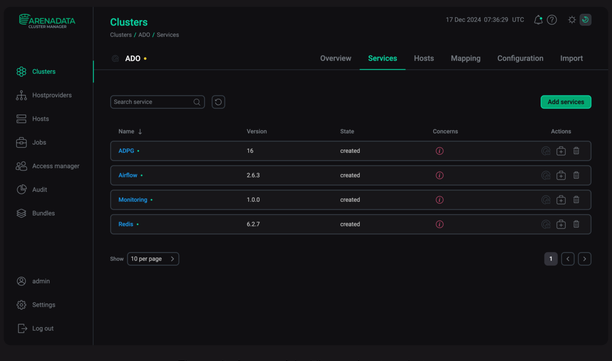
Select services

The brief description of available services is listed below.

Services that can be added to the ADO cluster

Service	Purpose
Airflow	A service used for creation, scheduling, and monitoring workflows in the form of Directed Acyclic Graphs (DAGs) of tasks. Can be used in Arenadata Orchestrator clusters for building ETL/ELT processes
Monitoring	A service that should be added if monitoring of the ADO cluster is planned
Redis	Redis is an open source, in-memory, NoSQL data store used primarily as an application cache or quick-response database
ADPG	Arenadata Postgres (ADPG) is a commercial distribution of the PostgreSQL database with additional Enterprise functionality

4. As a result, the added services are displayed at 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.

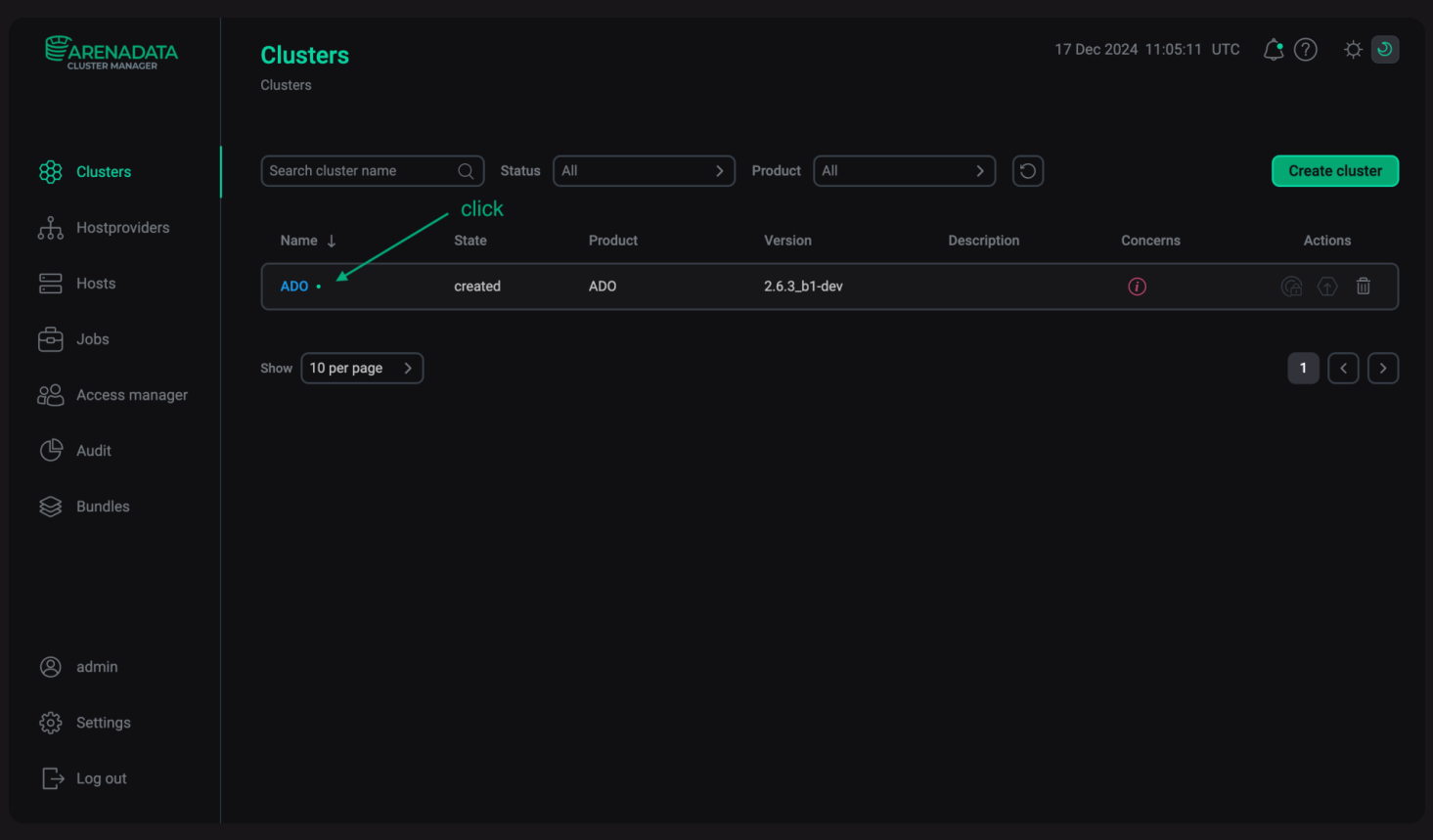
Add hosts to a cluster

Elena Kostyuchenko, Daria Barysheva

To Table of Contents

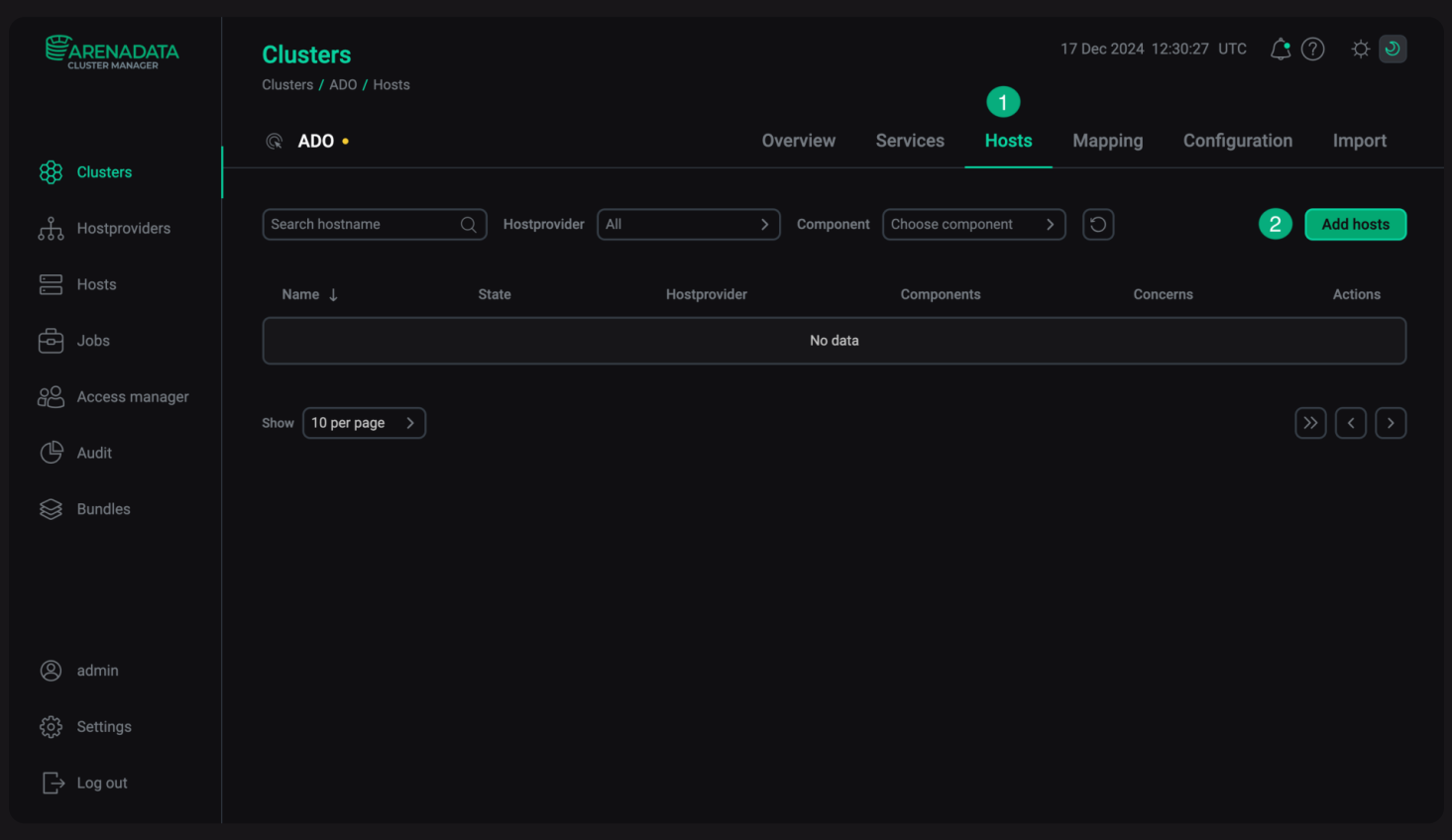
The steps for adding hosts to a cluster are listed below:

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



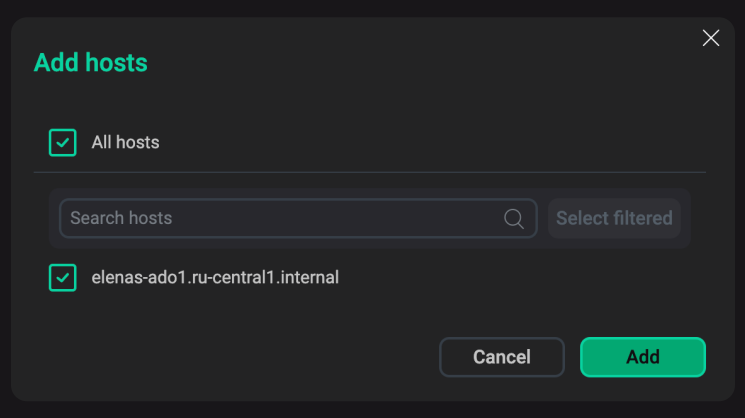
Select a cluster

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



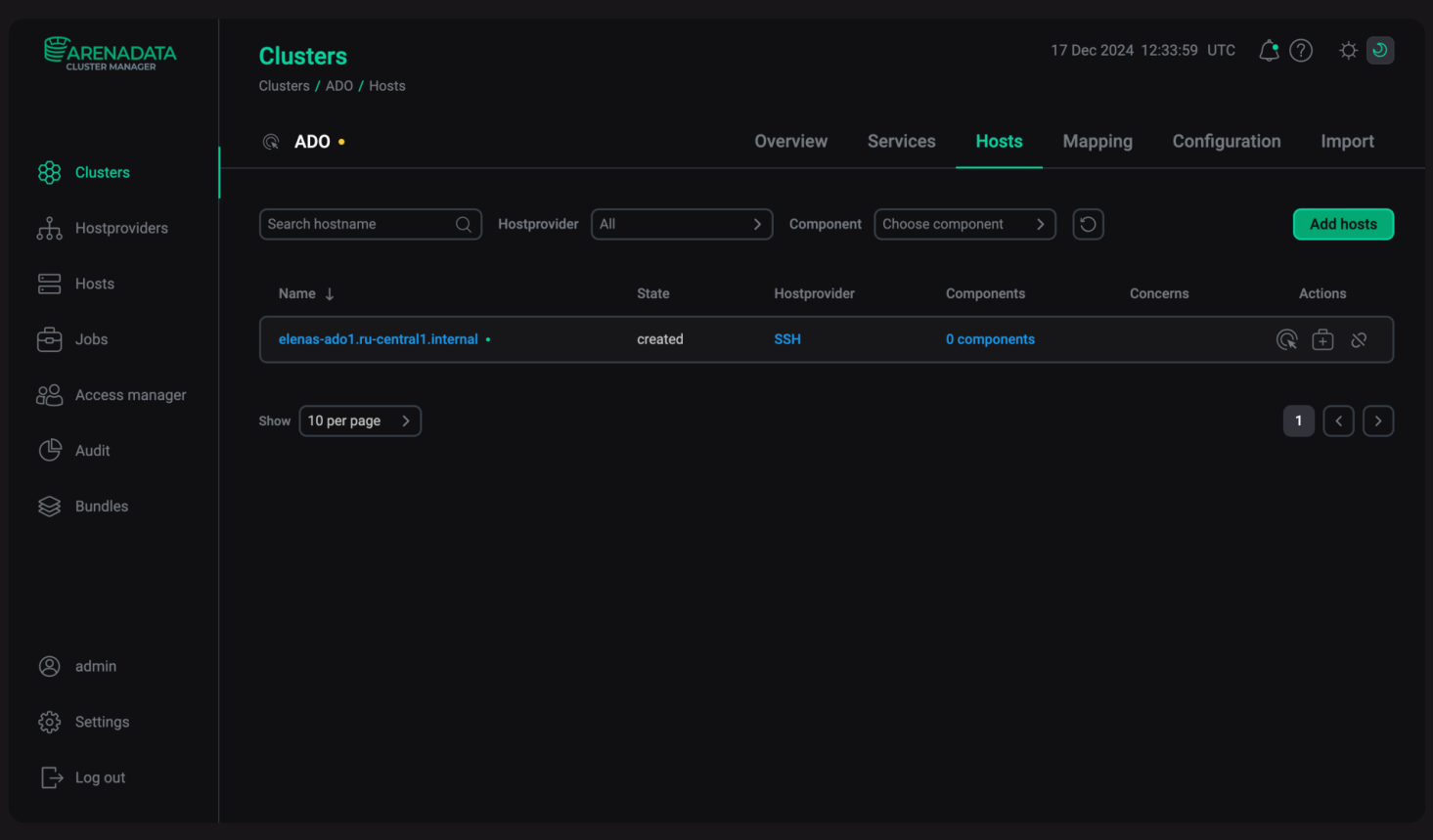
Switch to adding hosts

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

Elena Kostyuchenko, Daria Barysheva

To Table of Contents

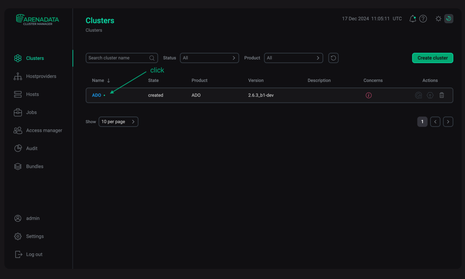
In ADCM, a **component** means a part of a service that should be deployed on one or several cluster hosts. For example, Airflow service can consist of such components as Airflow Server, Airflow Scheduler, Airflow Worker, and others.

IMPORTANT

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

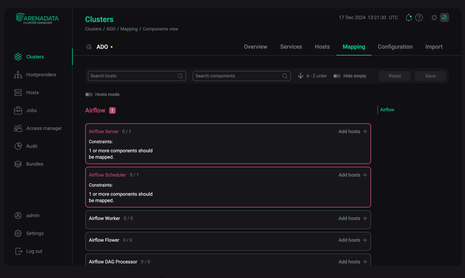
The steps for adding components to hosts are listed below:

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



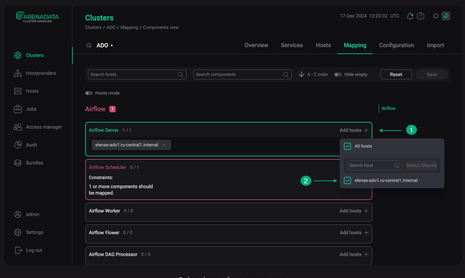
Select a cluster

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



Switch to mapping service components

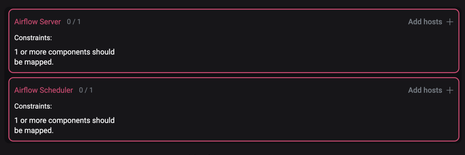
3. In the window that opens, map components to hosts. To assign hosts for each component, click ***Add hosts*** and select hosts in the pop-up window.



Select hosts for a component

Mandatory components

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



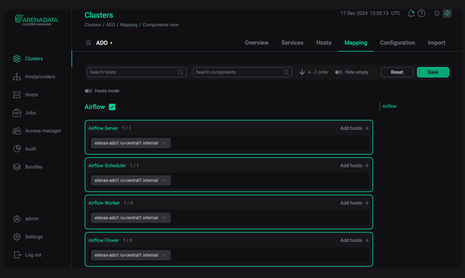
Mandatory components

CAUTION

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

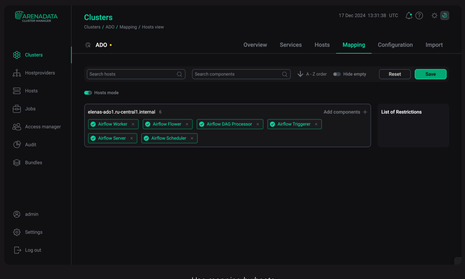
You can install all components in an ADO cluster on a single host. You can add a number of hosts depending on your requirements and distribute the components among them.

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.



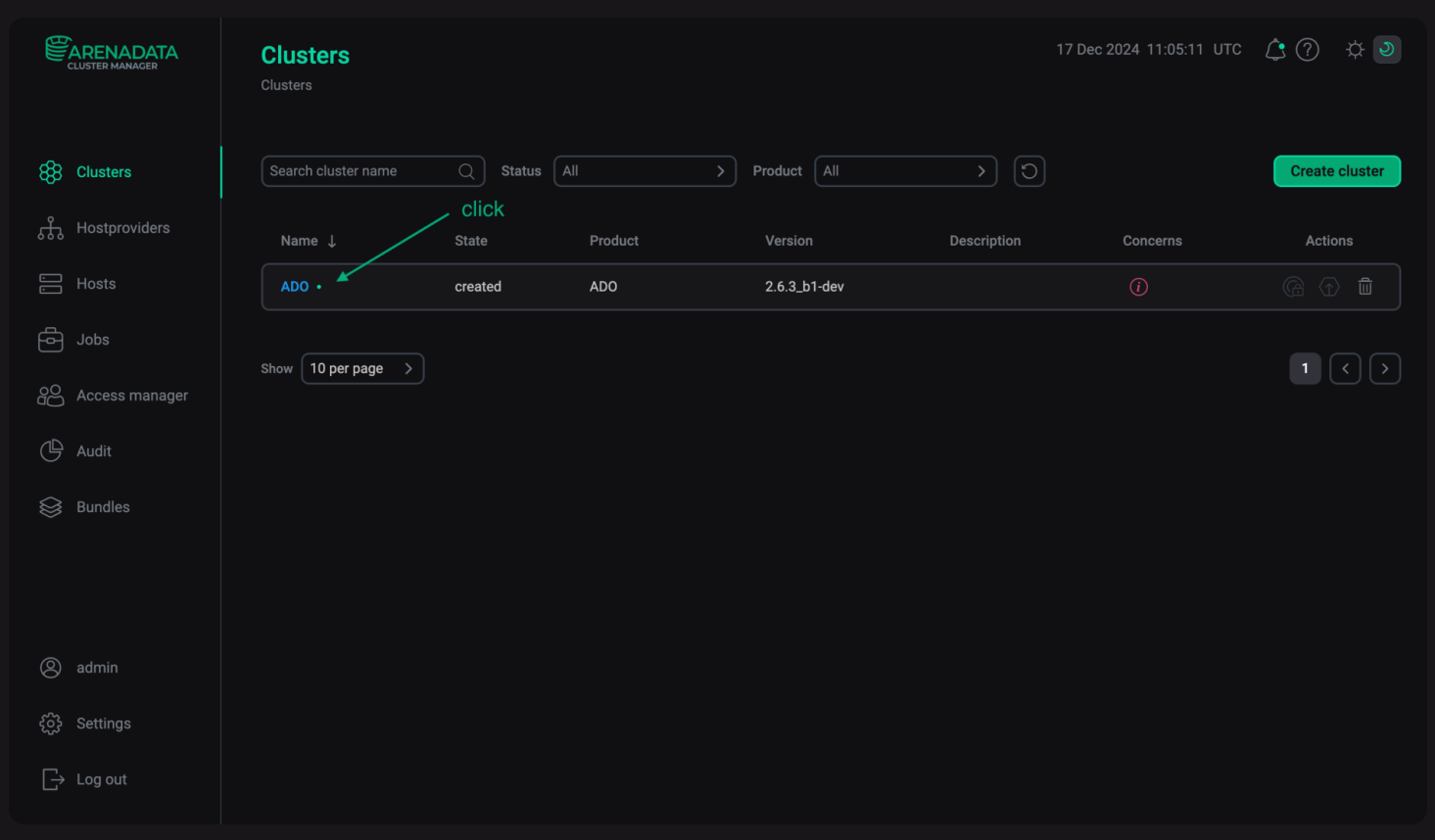
Use mapping by hosts

Configure services

Elena Kostyuchenko, Daria Barysheva

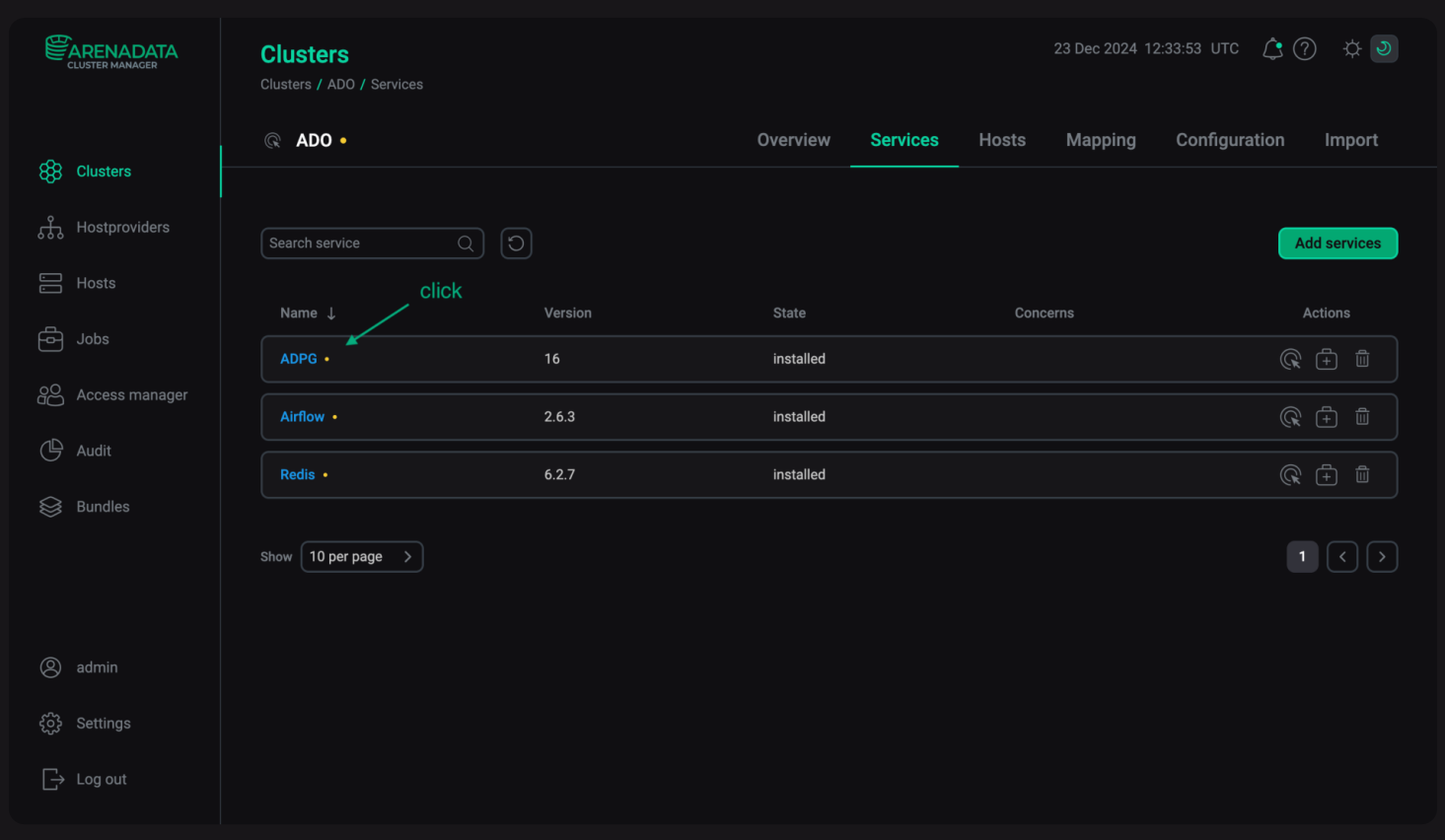
To Table of Contents

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



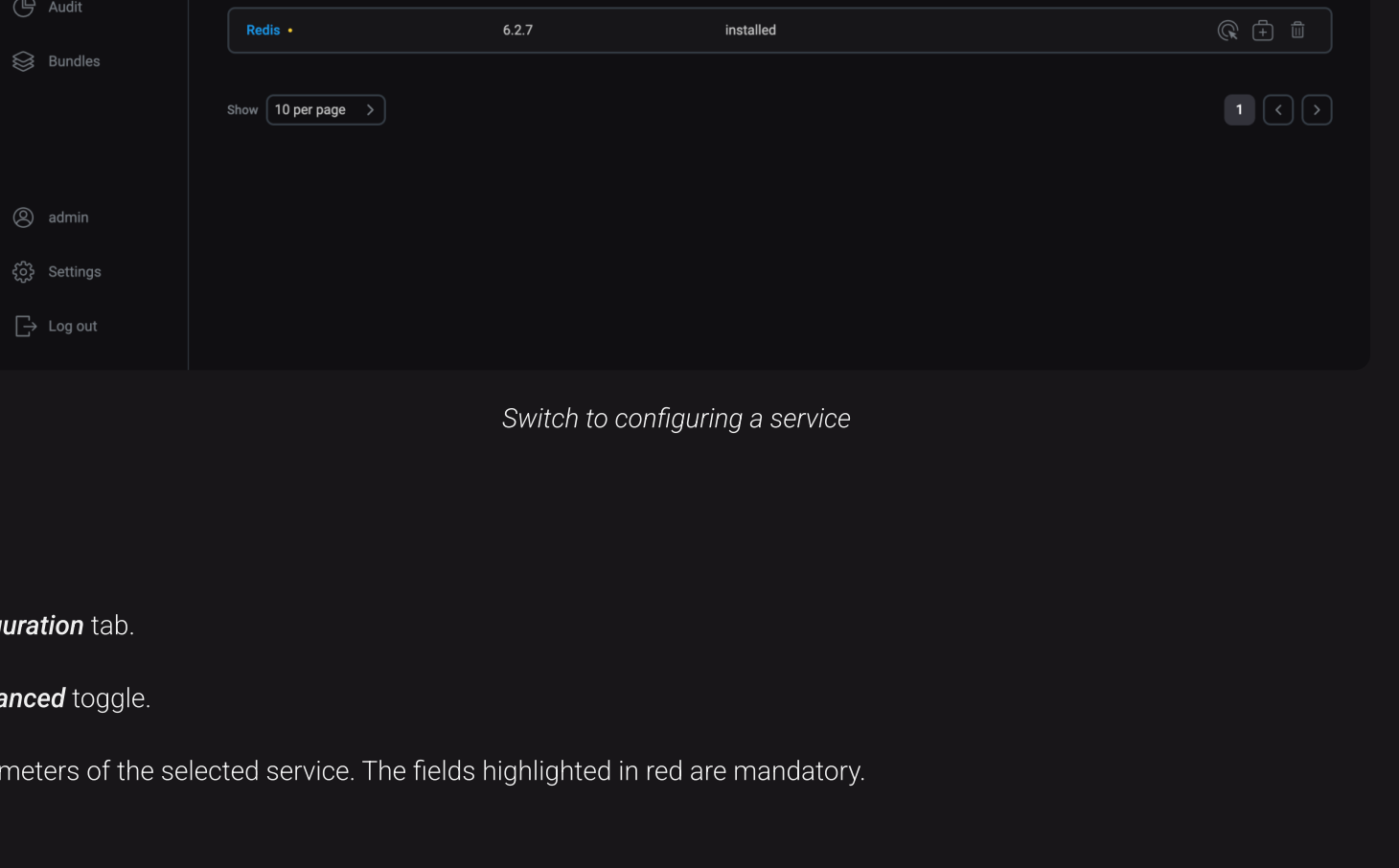
Select a cluster

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

NOTE

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

Configure a cluster

Elena Kostyuchenko, Daria Borysheva

Contents

To Table of Contents

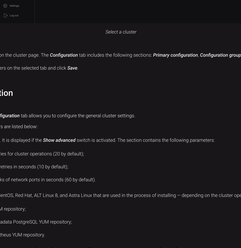
Primary configuration
Configuration groups
Ansible settings

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.

Available configuration parameters are listed below:

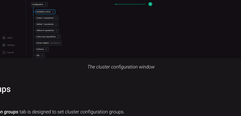
- The ***Reliability control*** section. It is displayed if the ***Show advanced*** switch is activated. The section contains the following parameters:
  - ***retries*** — the number of retries for cluster operations (20 by default),
  - ***delay*** — the delay between retries in seconds (10 by default),
  - ***timeout*** — timeout for checks of network ports in seconds (60 by default).
- Paths to the repositories for CentOS, Red Hat, ALT Linux 8, and Astra Linux that are used in the process of installing — depending on the cluster operation system:
  - ***arenadata*** — Arenadata YUM repository,
  - ***arenadata_postgres*** — Arenadata PostgreSQL YUM repository,
  - ***prometheus_repo*** — Prometheus YUM repository.
- The ***Docker registry*** parameter that specifies a Docker Registry address. The default value is hub.adsw.io — Docker Registry from the Arenadata repository.
- The ***Kerberos*** section that includes settings listed in the table below. Some options are visible only with the ***Show advanced*** option enabled.

Parameter	Description	Default value
Enable Kerberos	Enables the Kerberos authentication	False
Authentication on WEB UIs	Enables the Kerberos authentication on WEB UIs	False
SPNEGO Signature Value	A secret key that is used to generate and verify the signature in the HTTP authentication protocol (SPNEGO (in the Base64 format))	—
Kerberos KDC type	A KDC type. For more information, see Kerberos	MIT
KDC hosts	One or more KDC hosts with running FreeIPA server(s)	—
Realm	A Kerberos realm to connect to the FreeIPA server	—
Domains	One or more domains associated with FreeIPA	—
Kadmin server	A host where `kadmin` is running	—
Kadmin principal	A principal name used to connect via `kadmin`, for example: `admin@EXAMPLE.INTERNAL`	—
Kadmin password	An IPA Admin password	—
Keytabs directory	A directory with keytab files	/etc/security/keytabs
Additional realms	Additional Kerberos realms	—
Delay between kinit invocation attempts	Delay between kinit invocation attempts	5
Number of retries for kinit invocation attempts	Number of retries for kinit invocation attempts	10
Trusted Active Directory server	An Active Directory server for one-way cross-realm trust from the MIT Kerberos KDC	—
Trusted Active Directory realm	An Active Directory realm for one-way cross-realm trust from the MIT Kerberos KDC	—
Custom krb5.conf	Enables adding custom parameters to the *krb5.conf* file	False
krb5.conf	Additional parameters to write to the *krb5.conf* file	—
Admin DN	A full DistinguishedName of the admin user with rights to `create/modify/delete/patchange` user accounts in the target Organizational Unit	—
LDAP URL	LDAP URL that consists of the `ldap://` or `ldaps://` protocol prefix, hostname or IP address, and a port (if any). For example: `ldaps://192.168.4.2:636`	—
Container DN	A container distinguished name	—
TLS CA certificate Path	A CA certificate path on the host filesystem that is pre-located or should be written from the **TLS CA certificate** field	—
TLS CA certificate (optional)	A CA certificate that is written on a host by the path from ***TLS CA certificate Path*** during the execution of the ***Manage Kerberos*** action. The certificate is not saved in ADCM	—
Custom ldap.conf	Enables adding custom parameters to the *ldap.conf* file	False
ldap.conf	Additional parameters that are written to the *ldap.conf* file	—
IpaClient No NTP Autoconf	Disables the NTP configuration during the IPA client installation	False
IpaClient No DNS Lookup	Disables the DNS lookup for the FreeIPA server during the IPA client installation	True
Custom smb.conf	Usage of custom configuration file *smb.conf*	False
smb.conf	Custom configuration file *smb.conf*	—
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	True
Configure services and clients	Enables updating of services and clients configuration	True
Run service checks	Enables service check runs	True

- **SSL** section with the settings listed in the table below:

Parameter	Description	Default value
Enable SSL	Enables SSL	false
Keystore path	A path to the keystore file	/etc/ssl/keystore.jks
Keystore password	A password for the keystore file	—
Truststore path	A path to the truststore file	/etc/ssl/truststore.jks
Truststore password	A password for the truststore file	—
TLS Version	Version of the TLS protocol	TLSv1.2
Run service checks	Enables service check runs	true

- The ***Precheck packages*** parameter that specifies whether the availability of the packages in the repository should be checked before installation. `False` by default.
- The ***Cluster UUID*** displays the cluster ID information.

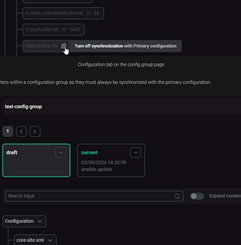


The cluster configuration window

Configuration groups

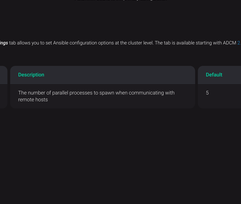
The **Configuration** → ***Configuration groups*** tab is designed to set cluster configuration groups.

To configure parameters in a configuration group, hover over the parameters that you wish to configure differently, click the **Turn off synchronization** icon, edit them, and click ***Save***.



Configuration tab on the config group page

You cannot change some parameters within a configuration group as they must always be synchronized with the primary configuration.



Parameter bound to the primary configuration

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

2. Elena Kostyuchenko, Daria Baryshena

Contents

To Table of Contents

Step 1. Installation
- Whole cluster
- Single services

Step 2. View installation results

Step 3. Check the cluster state after the installation

Step 1. 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.

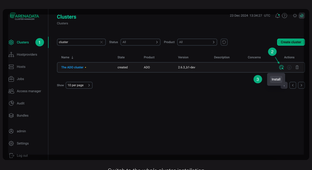
IMPORTANT

The installation must be performed on a clean host without prior installation of related or similar software.

Whole cluster

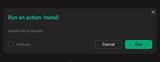
To install all cluster services within a single action, follow the steps:

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. Verify the requested action in the opened window.



Verify the action

3. Wait until the job is completed.

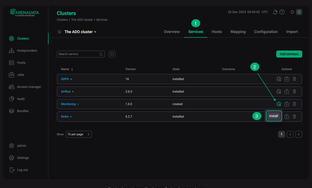
Single services

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

CAUTION

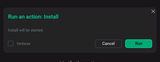
Notice that services are installed one by one. Do not install another service until the installation of the previous one is completed.

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



Switch to installation of single services

2. Verify the action in the opened window.

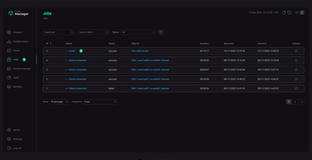


Verify the action

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

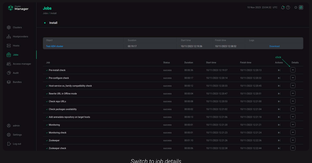
Step 2. View installation results

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. The window with job inner steps is opened. Click the ellipsis in the ***Details*** column for additional information on the current step.

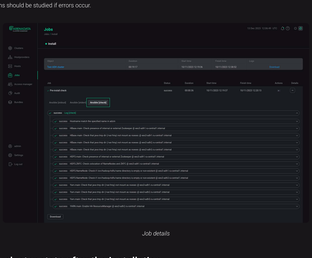


Switch to job details

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.

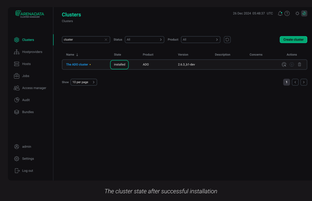


Job details

Step 3. Check the cluster state after the installation

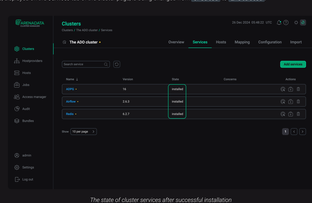
As a result of successful installation, the cluster and its services change their state according to the following rules:

- 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

Contents

To Table of Contents

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

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

Overview

When you add the Monitoring service to an ADO cluster, node exporter is installed on all hosts. It is a monitoring agent that reads host system metrics that Prometheus will collect. Prometheus will also collect metrics from ADO services. These metrics will be available in the Prometheus format on ports and endpoints specified in service configuration options. 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 ADO metrics. To do this, configure access to ADO metrics in your monitoring system using ADO's Prometheus parameters specified in the Monitoring service configuration.
> - You can also use the Federation mechanism to migrate all metrics from the Prometheus server deployed in ADO to your Prometheus.

Step 1. Add the service

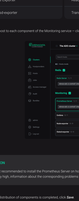
1. In the ADO UI interface, open the **Clusters** page and click your ADO cluster name. On the cluster page that opens, switch to the **Services** tab and click **Add services**.

Switch to adding services

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

Select the service

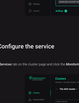
As a result, the added service is displayed in the **Services** tab.



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.



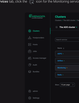
Switch to mapping service components

Each component should be installed on one host of the cluster.

Monitoring service components

Component	Description
Prometheus Server	Deploys a Prometheus server that serves as a - proxy channel for all collections of metrics on a host, - storage for all metrics of a cluster, - generator of alerts based on collected metrics.
Grafana	Visualizes ADO metrics as graphs and charts organized into dashboards
Node Exporter	Reads host system metrics that Prometheus will collect
Statsd-exporter	Translates StatsD-style metrics to Prometheus metrics via configured mapping rules

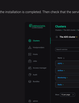
2. Assign a host to each component of the Monitoring service – click **Add hosts** and select the desired host in the pop-up window.



Select a host for a component

> **CAUTION**
>
> It is not recommended to install the Prometheus Server on hosts with ADO – use a separate host for it. Otherwise, if an ADO host fails and/or the load on it is critically high, information about the corresponding problems will not be saved.

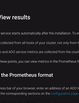
3. After the distribution of components is completed, click **Save**.



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 configuration parameters – see the **Monitoring** section in the 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 the ⋮ icon 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 completed. Then check that the service state has changed from `prepared` → `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 the 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 system and ADO service metrics are collected from the cluster hosts.

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.

Metrics in the Prometheus format

1. In the address bar of your browser, enter an address of an ADO host with the specified port and endpoint to listen for service or system metrics. Port numbers and endpoints are defined in the corresponding sections on the configuration page of the Monitoring service:
   - **ADO's service metric settings** – settings for access to monitoring metrics of ADO services.
   - **Node Exporter settings** – settings for access to system metrics of the ADO cluster hosts.

   For example, http://10.62.48.107:11003/metrics is an address to view metrics of the ADO service on a host with IP `10.62.48.107`.
2. The page that opens will display monitoring metrics from the specified host of the ADO cluster in the Prometheus format.

Airflow server metrics in the Prometheus format

Prometheus web interface

1. In the address bar of your browser, enter an IP address of a host where the Prometheus Server component is installed. Specify also a port number from the **Alerts_address** parameter in the **Prometheus settings** section of the Monitoring service configuration (the default port is `19011`). For example, http://10.62.48.107:19011. To log into the Prometheus interface, use a username and password that are also specified in the **Prometheus settings** section – the **Prometheus users to login/input to Prometheus** setting.
2. In the **Expression** field, you can select a metric and click **Execute** – values of the metric on all hosts of the ADO cluster will be shown in the histogram.

Prometheus web interface

Grafana web interface

1. In the address bar of your browser, enter an address of a host on which Grafana is deployed and add a port number – the value of the **Grafana Web port** parameter located in the **Grafana settings** section of the Monitoring service configuration (the default value is `19300`). For example, http://10.62.48.107:19300. To log in, use **admin** as a username, and the **Grafana administrator's password** parameter value (also found in the **Grafana settings** section of service configuration parameters) as a password.
2. In the window that opens, navigate to **Home** → **Dashboards** and expand the **ADO Dashboard -ADO_cluster_name** section. In this section, you can select one of available dashboards to view service or system metrics coming from your ADO cluster.

View ADO metrics in Grafana

Offline installation

To Table of Contents

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

ADCM

Step 1.

Install ADCM

ADCM

Step 2.

Prepare hosts

ADCM

Step 3.

Install ET cluster

ADCM

Step 4.

Install ADO cluster

ADCM

Step 5.

Install monitoring

Install ASCM

Contents

- Step 1. Check prerequisites
  - Software and hardware requirements
  - Install Docker
  - Install Docker Compose
  - Install ASCM
  - Allow access to the Docker CLI without root privileges
- Step 2. Install ASCM
- Step 3. Start ASCM
- Step 4. Check installation
  - Check the Docker container status
  - Check the ASCM port availability
  - Check the ASCM web interface
  - Set the ASCM URL
- Step 5. Update ASCM
  - Online update
  - Offline update

Table of contents

ASCM supports deployment on Linux platforms that use systemd and provides [systemctl](#) commands to manage ASCM services.

Step 1. Check prerequisites

Software and hardware requirements

Verify that your hardware and software meet the requirements listed in the following table.

Item	Requirement
Linux OS	Red Hat Enterprise Linux 7.3 or later CentOS 7.3 or later Oracle Linux 7.3 or later Ubuntu 16.04 LTS or later SUSE Linux Enterprise Server 12 SP2 or later openSUSE Leap 15 or later Debian 9.1 or later Amazon Linux 2 or later
CPU	4 cores or more
Memory	8 GB or more
Disk	100 GB or more
Network	Access to the Internet is required during installation and upgrade. The following domains must be accessible: hub.docker.com, registry-1.docker.io, and auth.docker.io.

NOTICE

The following requirements are **optional**. They are required only when you enable the corresponding features.

Item	Requirement
CPU	8 cores or more
Memory	16 GB or more
Disk	200 GB or more

NOTICE

Red Hat Enterprise Linux, CentOS, and Oracle Linux are not supported in ARM, IPv6, and FIPS environments. Ubuntu is not supported in IPv6 and FIPS environments. SUSE Linux Enterprise Server is not supported in FIPS environments. openSUSE Leap is not supported in IPv6 and FIPS environments. Debian is not supported in IPv6 and FIPS environments. Amazon Linux 2 is not supported in IPv6 and FIPS environments.

To get the root permissions, switch to the root user or run the **sudo** command.

Check if SELinux is disabled

Run the **sestatus** command to check whether SELinux is disabled. If SELinux is enabled, run the **setenforce 0** command to disable it.

```
sestatus
```

Check if firewalld is disabled

Run the **systemctl status firewalld** command to check whether firewalld is disabled. If firewalld is enabled, run the **systemctl stop firewalld** command to disable it.

```
systemctl status firewalld
```

Check if AppArmor is disabled (Ubuntu)

Run the **systemctl status apparmor** command to check whether AppArmor is disabled. If AppArmor is enabled, run the **systemctl stop apparmor** command to disable it.

```
systemctl status apparmor
```

Check if SUSEfirewall2 is disabled (SUSE Linux)

Run the **systemctl status SuSEfirewall2** command to check whether SUSEfirewall2 is disabled. If SUSEfirewall2 is enabled, run the **systemctl stop SuSEfirewall2** command to disable it.

```
systemctl status SuSEfirewall2
```

Check if AA-enforced is disabled (SUSE Linux)

Run the **aa-status** command to check whether AA-enforced is disabled. If AA-enforced is enabled, run the **systemctl stop apparmor** command to disable it.

```
aa-status
```

Install Docker

NOTICE

Docker 18.06.0-ce or later is required. Check whether Docker is installed on the server. If Docker is not installed, install Docker by referring to the following steps.

CentOS / Red Hat

1. Run **yum** to install Docker packages from a specific mirror site.

```
yum -y install docker
```

2. Start Docker and enable it to start automatically at boot.

```
systemctl start docker
systemctl enable docker
```

Ubuntu

1. Run **apt-get** to install Docker packages from a specific mirror site.

```
apt-get -y install docker.io
```

2. Start Docker and enable it to start automatically at boot.

```
systemctl start docker
systemctl enable docker
```

SUSE Linux

1. Run **zypper** to install Docker packages from a specific mirror site.

```
zypper -n install docker
```

2. Start Docker and enable it to start automatically at boot.

```
systemctl start docker
systemctl enable docker
```

Amazon Linux 2

1. Run **yum** to install Docker packages from a specific mirror site.

```
yum -y install docker
```

2. Start Docker and enable it to start automatically at boot.

```
systemctl start docker
systemctl enable docker
```

Amazon Linux AMI 2018.03 or earlier

1. Run **yum** to install Docker packages from a specific mirror site.

```
yum -y install docker-18.06.1ce-2.61.amzn1
```

2. Start Docker and enable it to start automatically at boot.

```
systemctl start docker
systemctl enable docker
```

Debian

1. Run **apt-get** to install Docker packages from a specific mirror site.

```
apt-get -y install docker.io
```

2. Start Docker and enable it to start automatically at boot.

```
systemctl start docker
systemctl enable docker
```

Oracle Linux 7.5 or later

1. Run **yum** to install Docker packages from a specific mirror site.

```
yum -y install docker-engine
```

2. Start Docker and enable it to start automatically at boot.

```
systemctl start docker
systemctl enable docker
```

Oracle Linux 7.4 or earlier

1. Run **yum** to install Docker packages from a specific mirror site.

```
yum -y install docker
```

2. Start Docker and enable it to start automatically at boot.

```
systemctl start docker
systemctl enable docker
```

Install Docker Compose

NOTICE

Docker Compose 1.18.0 or later is required. Check whether Docker Compose is installed on the server. If Docker Compose is not installed, install Docker Compose by referring to the following steps.

1. Run **curl** to download Docker Compose from a specific mirror site.

```
curl -L https://github.com/docker/compose/releases/download/1.24.1/docker-compose-`uname -s`-`uname -m` -o /usr/local/bin/docker-compose
```

2. Add the execute permission to Docker Compose.

```
chmod +x /usr/local/bin/docker-compose
```

Install ASCM

NOTICE

Install ASCM as the root user to ensure normal running.

1. Run **tar** to decompress the ASCM installation package.

```
tar -zxvf ascm-x.x.x.tar.gz
```

2. Run **cd** to enter the decompressed directory.

```
cd ascm-x.x.x
```

3. Run **./ascmctl install** to install ASCM.

```
./ascmctl install
```

Allow access to the Docker CLI without root privileges

By default, only the root user has the permission to access the Docker CLI. To allow a non-root user to access the Docker CLI, add the user to the **docker** user group.

1. Run **usermod** to add the user to the **docker** user group.

```
usermod -aG docker username
```

2. Run **su** to switch to the user.

```
su username
```

3. Run **docker info** to verify that the user has the permission to access the Docker CLI.

```
docker info
```

Step 2. Install ASCM

Perform the following steps to install ASCM:

1. Run **tar** to decompress the ASCM installation package.

```
tar -zxvf ascm-x.x.x.tar.gz
```

2. Run **cd** to enter the decompressed directory.

```
cd ascm-x.x.x
```

3. Run **./ascmctl install** to install ASCM.

```
./ascmctl install
```

Step 3. Start ASCM

Perform the following steps to start ASCM:

1. Run **./ascmctl start** to start ASCM.

```
./ascmctl start
```

2. Run **./ascmctl status** to check the ASCM status.

```
./ascmctl status
```

Step 4. Check installation

Check the Docker container status

Run **docker ps** to check whether the ASCM containers are running properly.

```
docker ps
```

The following figure shows an example of the output:

Check the ASCM port availability

After ASCM is installed, it uses the following ports by default: **80**, **443**, **8080**, and **8443**. Run **netstat -tlnp** to check whether these ports are occupied.

```
netstat -tlnp
```

The following figure shows an example of the output:

Check the ASCM web interface

Open a browser and enter the ASCM web interface. The following steps use Chrome as an example.

1. Open Chrome.
2. Enter **https://IP address:8443**.

Figure 4-1 ASCM

Log in with the default account and password. The default account is **admin**, and the default password is **Admin@123**.

Figure 4-2 ASCM login page

Change the default password.

Figure 4-3 Changing the default password

Set the ASCM URL

After logging in to the ASCM web interface, choose **System > Basic Settings** in the left navigation pane. The **Basic Settings** page is displayed. Set the **Access Address** to the IP address or domain name that is used to access ASCM, and click **Save**. This address is used to access ASCM in the browser and is also used by managed nodes to connect to ASCM. Therefore, ensure that the managed nodes can access this address.

Figure 4-4 Setting the ASCM URL

1. Choose **System > Basic Settings**.
2. Set the **Access Address**.
3. Click **Save**.

Figure 4-5 Saving the ASCM URL

Step 5. Update ASCM

NOTICE

- Back up ASCM data before the update. For details, see [Backup and Restore](#).
- Before updating, check the version compatibility between ASCM and the managed nodes. For details, see [Version Compatibility](#).
- Update ASCM as the root user to ensure normal running.
- Unlike installation, the update does not require [Docker](#) and [Docker Compose](#) to be installed on the server.

Online update

Perform the following steps to update ASCM online:

1. Run **tar** to decompress the ASCM installation package.

```
tar -zxvf ascm-x.x.x.tar.gz
```

2. Run **cd** to enter the decompressed directory.

```
cd ascm-x.x.x
```

3. Run **./ascmctl update** to update ASCM.

```
./ascmctl update
```

NOTICE

- By default, the update retains the configurations of the previous version.
- To update ASCM to a specific version, run the **./ascmctl update -t x.x.x** command.

IMPORTANT

- To update ASCM to a version earlier than 1.2.0, run the **./ascmctl update -t x.x.x** command.
- To update ASCM to version 1.2.0 or later, run the **./ascmctl update** command.

Offline update

Perform the following steps to update ASCM offline:

1. Download the installation package for the target version of ASCM.
2. Run **tar** to decompress the ASCM installation package.

```
tar -zxvf ascm-x.x.x.tar.gz
```

3. Run **cd** to enter the decompressed directory.

```
cd ascm-x.x.x
```

4. Run **./ascmctl update -p /IP/ascm-x.x.x.tar** to update ASCM.

```
./ascmctl update -p /IP/ascm-x.x.x.tar
```

NOTICE

- By default, the update retains the configurations of the previous version.
- To update ASCM to a specific version, run the **./ascmctl update -t x.x.x -p /IP/ascm-x.x.x.tar** command.

IMPORTANT

- To update ASCM to a version earlier than 1.2.0, run the **./ascmctl update -t x.x.x -p /IP/ascm-x.x.x.tar** command.
- To update ASCM to version 1.2.0 or later, run the **./ascmctl update -p /IP/ascm-x.x.x.tar** command.

NOTICE

- Replace **/IP/** in the preceding commands with the actual path of the installation package.
- Replace **ascm-x.x.x.tar** in the preceding commands with the actual name of the installation package.

5. Confirm whether the update is successful.

```
./ascmctl status
```

6. Check whether the ASCM containers are running properly.

```
docker ps
```

NOTICE

- Run **./ascmctl update -t x.x.x -p /IP/ascm-x.x.x.tar** to update ASCM to a specific version.
- Run **./ascmctl update -p /IP/ascm-x.x.x.tar** to update ASCM to the latest version.
- Replace **/IP/** in the preceding commands with the actual path of the installation package.
- Replace **ascm-x.x.x.tar** in the preceding commands with the actual name of the installation package.

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

When creating ADO hosts via ADCM, you should specify their FQDNs.

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.

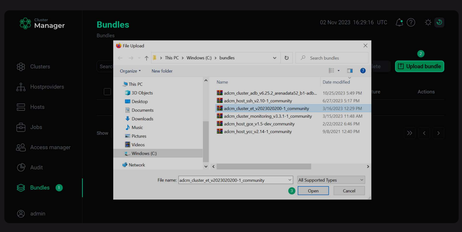
Step 1. Get the Enterprise version of a cluster bundle

Please get a cluster bundle from the Arenadata support team.

Step 2. Upload a cluster bundle to ADCM

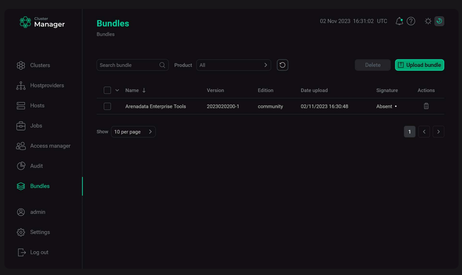
To upload a bundle to ADCM, follow the steps:

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

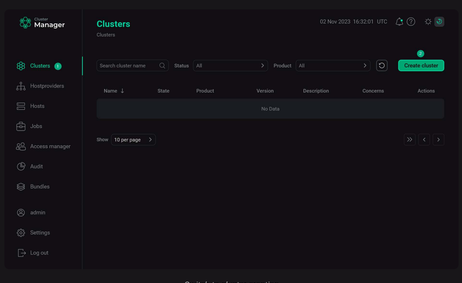
Step 3. Create a cluster based on the uploaded bundle

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.

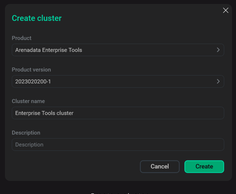
The steps for cluster creation are given below:

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



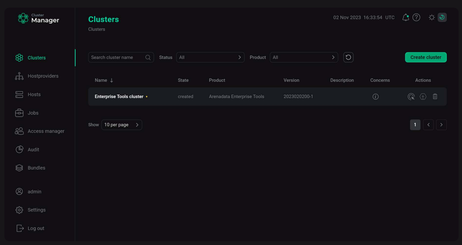
Switch to cluster creation

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

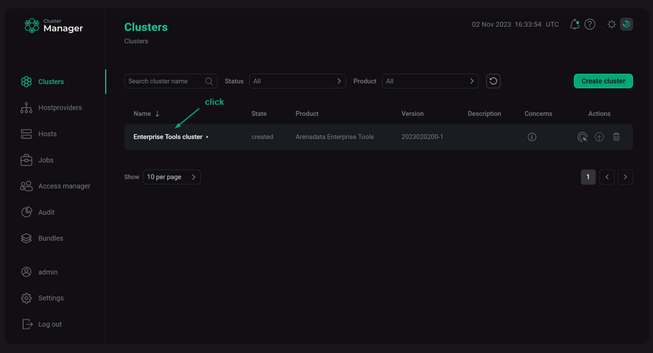
Add services

Daria Barysheva

To Table of Contents

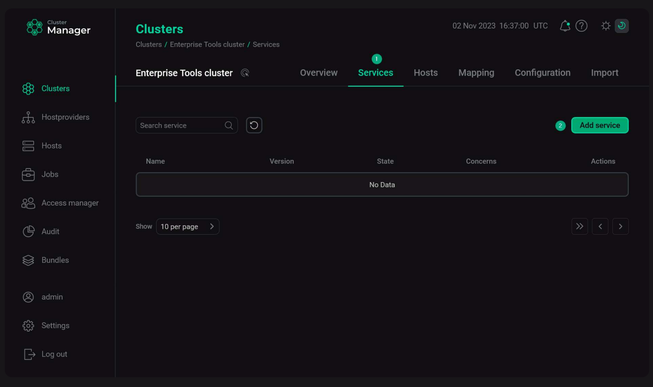
In ADCM, a **service** means a software that performs some function. The steps for adding services to a cluster are listed below:

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



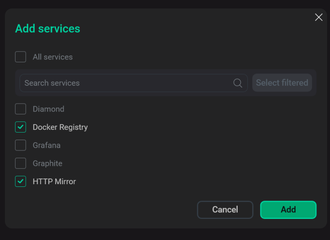
Select a cluster

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.

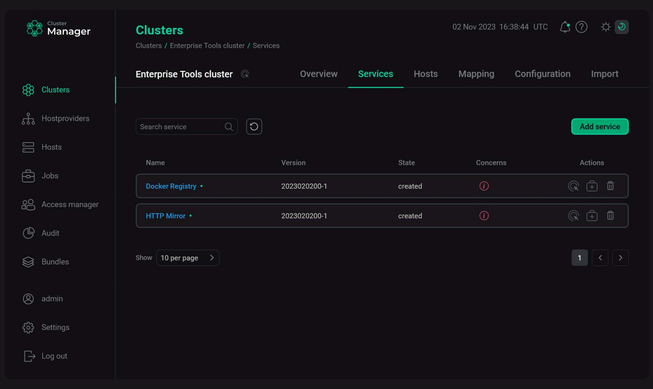
Services that should be added to the Enterprise Tools cluster

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

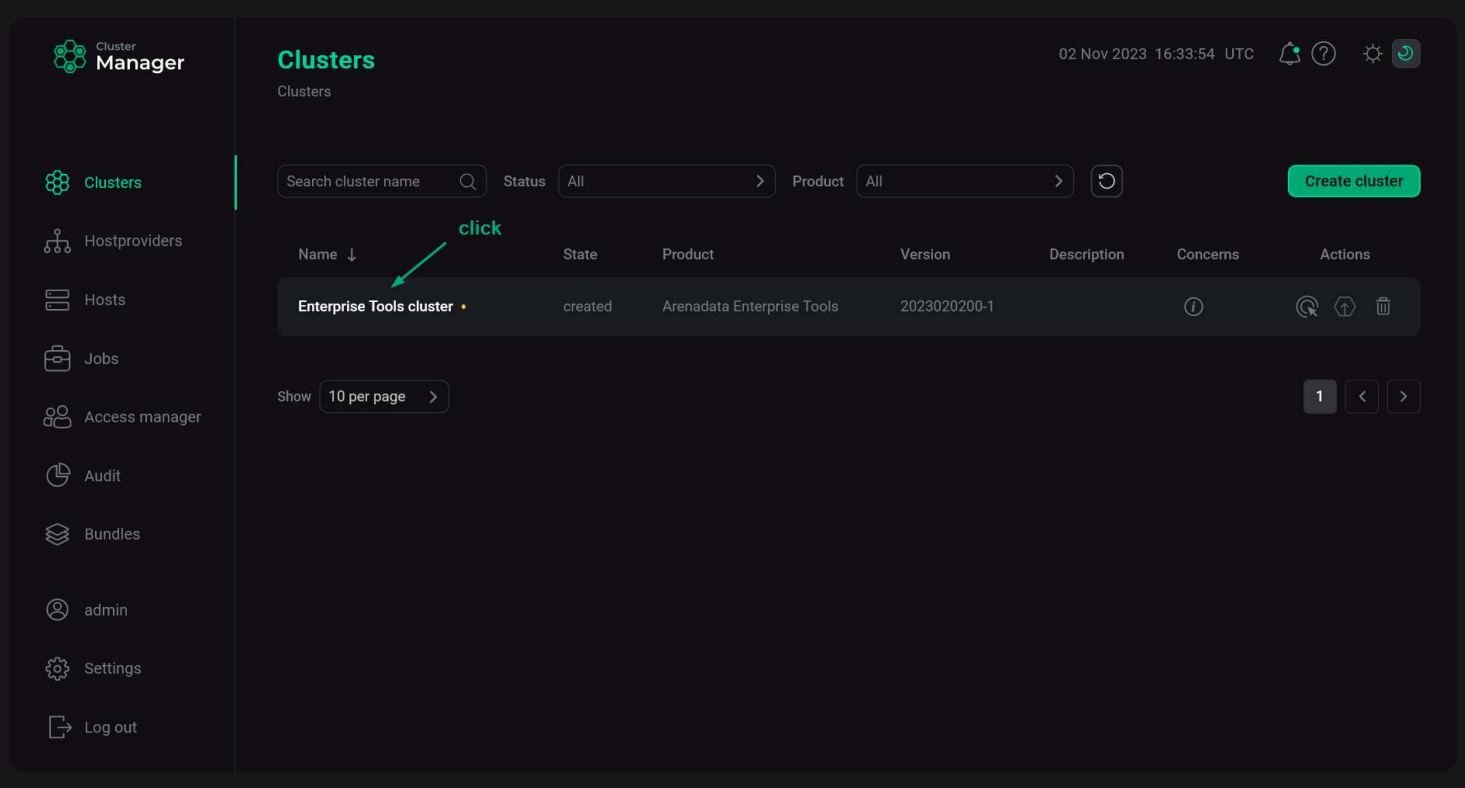
Add hosts to a cluster

Daria Barysheva

To Table of Contents

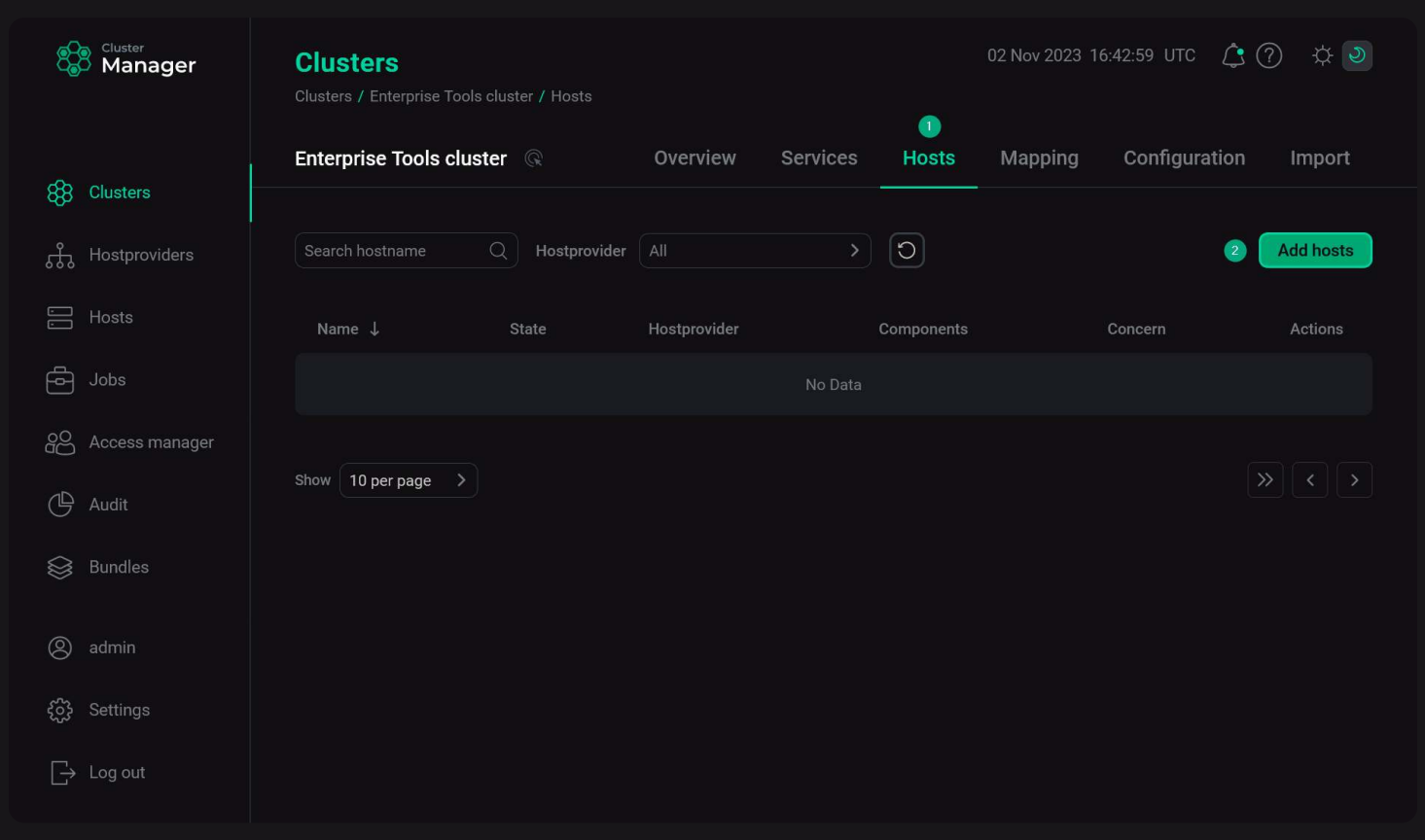
The steps for adding hosts to a cluster are listed below:

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



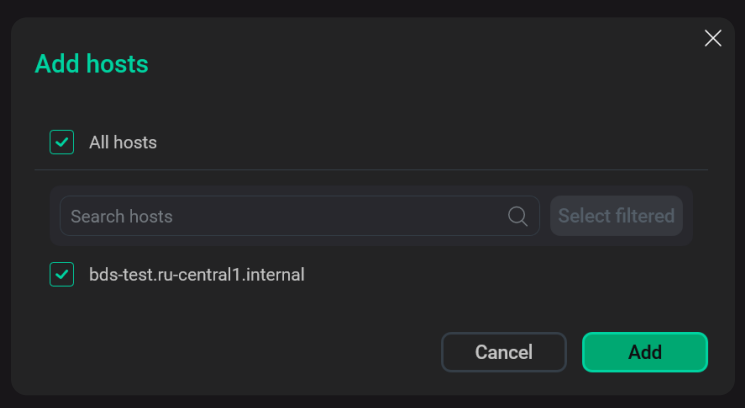
Select a cluster

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



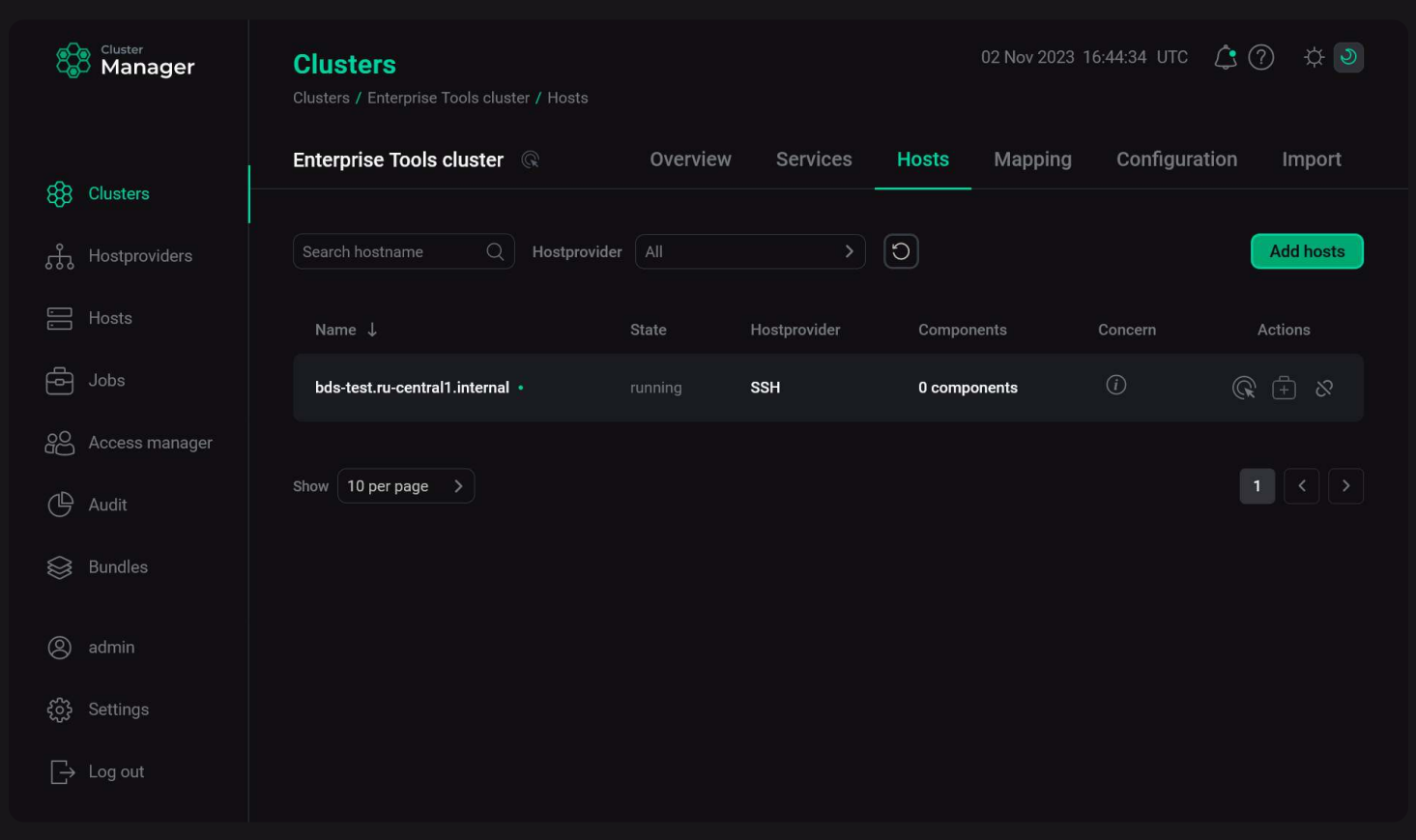
Switch to adding hosts

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

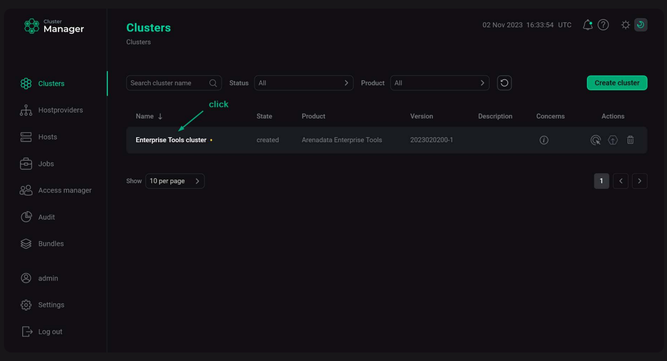
In ADCM, a **component** means a part of a service that should be deployed on one or several cluster hosts.

IMPORTANT

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

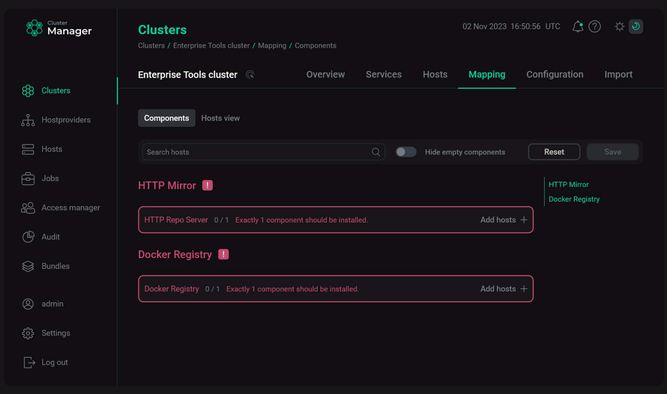
The steps for adding components to hosts are listed below:

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



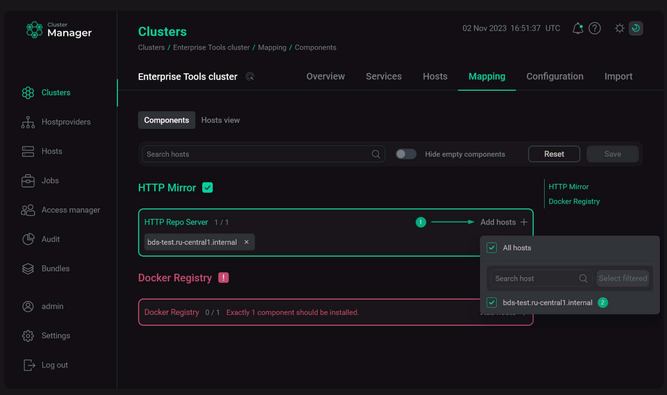
Select a cluster

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



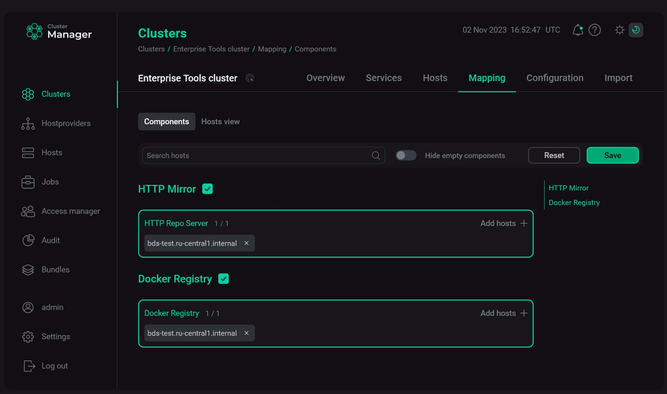
Switch to mapping service components

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

Daria Barysheva

Contents

To Table of Contents

Step 1. Run installation

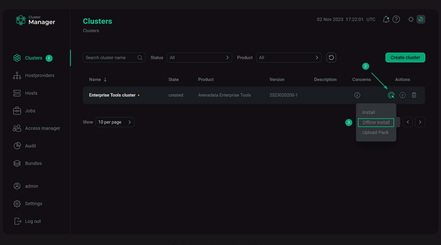
Step 2. Upload packages for offline installation

Step 3. Verify results

Step 1. Run installation

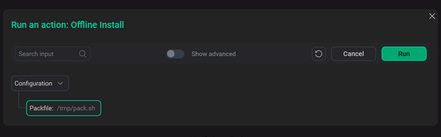
The last step of adding a new Enterprise Tools cluster is to install its services. To do this:

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***.

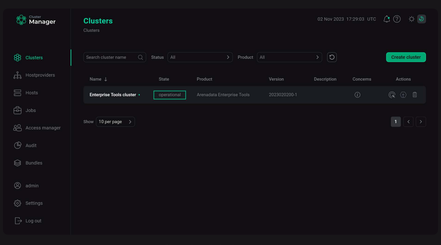


Verify installation

3. Wait until the installation process is completed.

NOTE

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

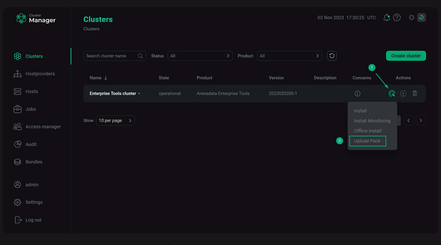


Cluster installation is completed

Step 2. Upload packages for offline installation

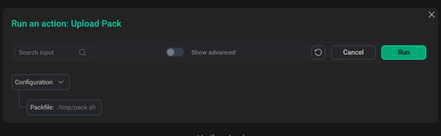
In addition to installing the cluster, you should process the shell script that is received from the Arenadata support team. It is necessary to enable offline installation of products in the future. To do this:

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 ADO installation.

```
total 0
drwxr-xr-x 5 root root 45 Sep 16 13:35 .
drwxr-xr-x 3 root root 28 Sep 16 13:35 ..
drwxr-xr-x 4 root root 38 Sep 16 13:35 ADO
drwxr-xr-x 4 root root 28 Sep 16 13:35 ADM
drwxr-xr-x 3 root root 20 Sep 16 13:35 zookeeper
```

Create a cluster

Elena Kostyuchenko, 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.

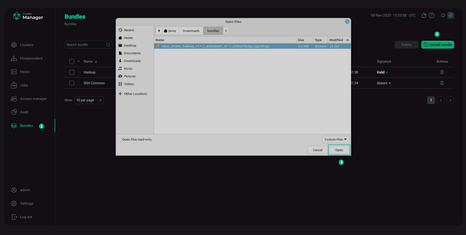
Step 1. Get the Enterprise version of a cluster bundle

Please get a cluster bundle from the Arenadata support team.

Step 2. Upload a cluster bundle to ADCM

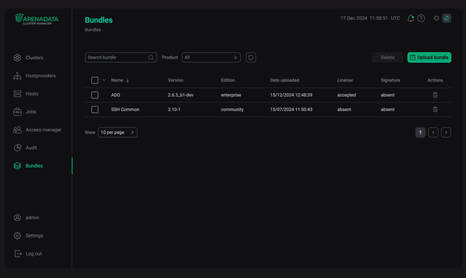
To upload a bundle to ADCM, follow the steps:

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

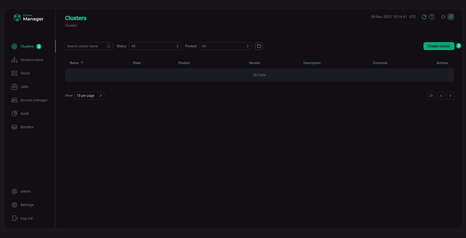
Step 3. Create a cluster based on the uploaded bundle

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:

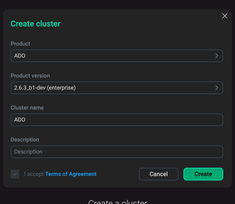
1. Select the ***Clusters*** item in the left navigation menu and click ***Create cluster***.



Switch to cluster creation

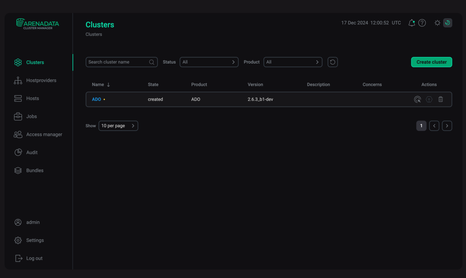
2. In the opened dialog, describe a new cluster instance:

   - 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

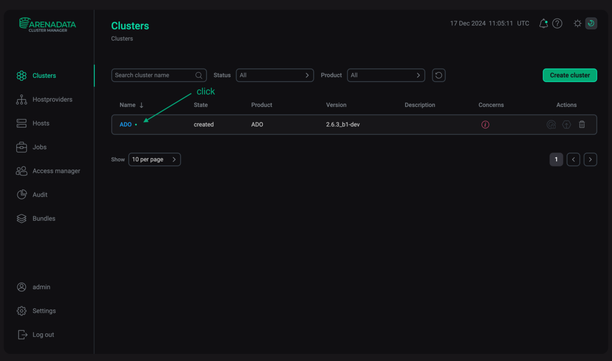
Add services

Elena Kostyuchenko, Daria Barysheva

To Table of Contents

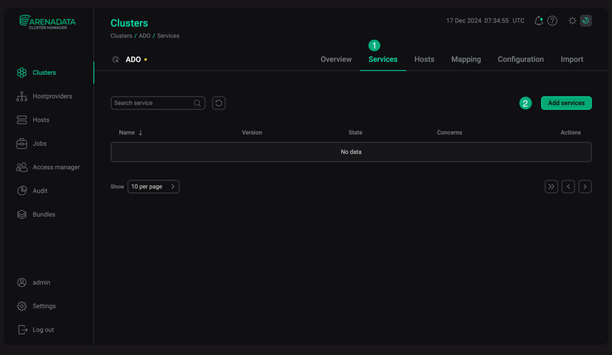
In ADCM, a **service** means a software that performs some function. Examples of services for ADO clusters: Airflow, ADPG, etc. The steps for adding services to a cluster are listed below:

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



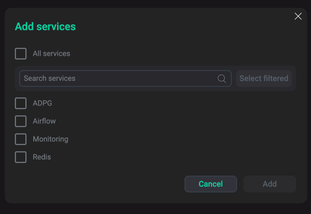
Select a cluster

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***.



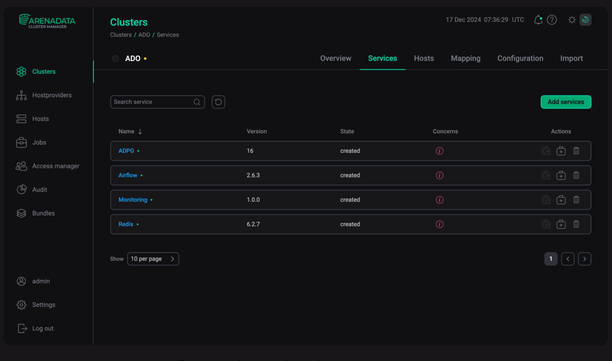
Select services

The brief description of available services is listed below.

Services that can be added to the ADO cluster

Service	Purpose
Airflow	A service used for creation, scheduling, and monitoring workflows in the form of Directed Acyclic Graphs (DAGs) of tasks. Can be used in Arenadata Orchestrator clusters for building ETL/ELT processes
Monitoring	A service that should be added if monitoring of the ADO cluster is planned
Redis	Redis is an open source, in-memory, NoSQL data store used primarily as an application cache or quick-response database
ADPG	Arenadata Postgres (ADPG) is a commercial distribution of the PostgreSQL database with additional Enterprise functionality

4. As a result, the added services are displayed at 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.

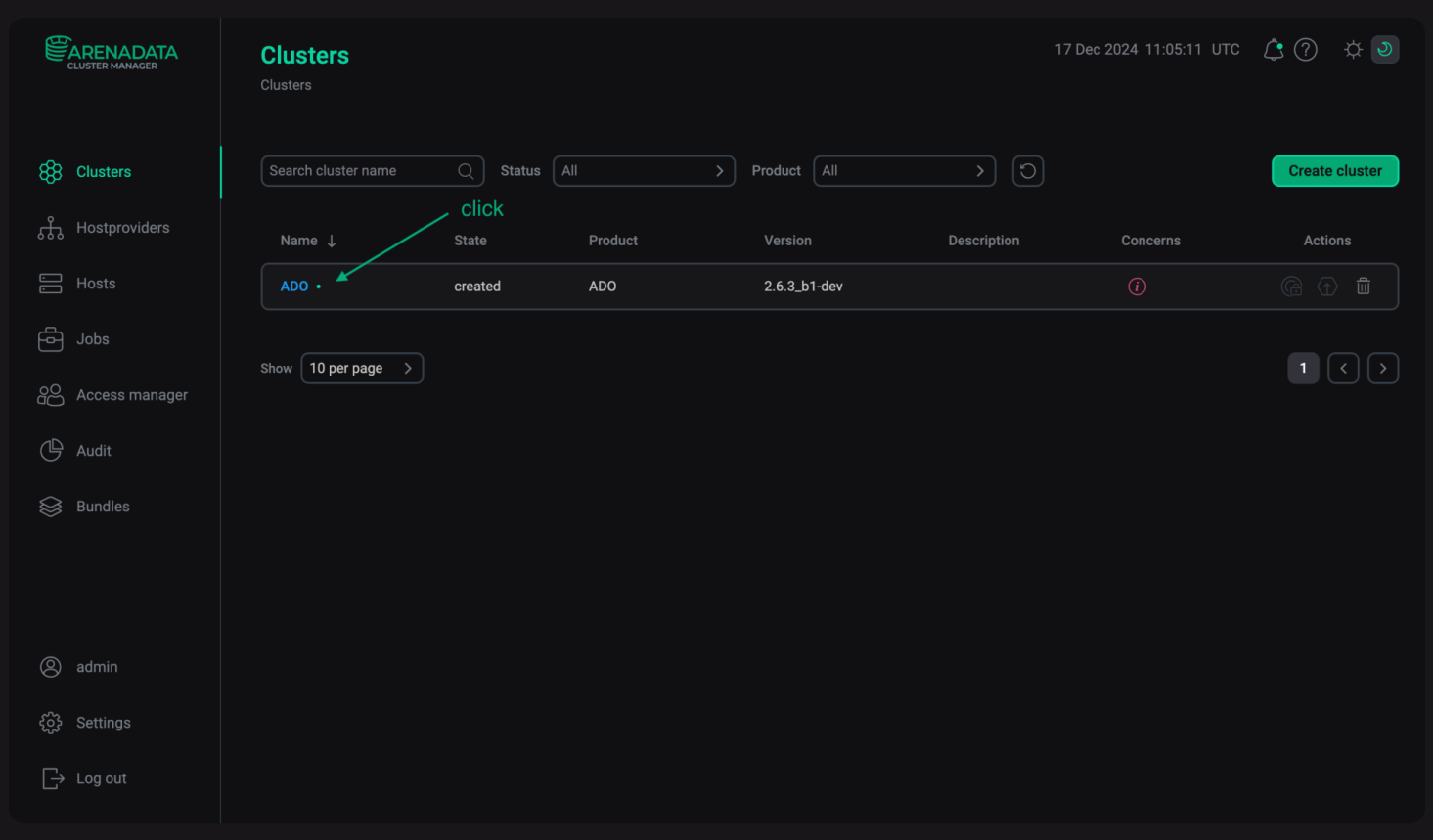
Add hosts to a cluster

Elena Kostyuchenko, Daria Barysheva

To Table of Contents

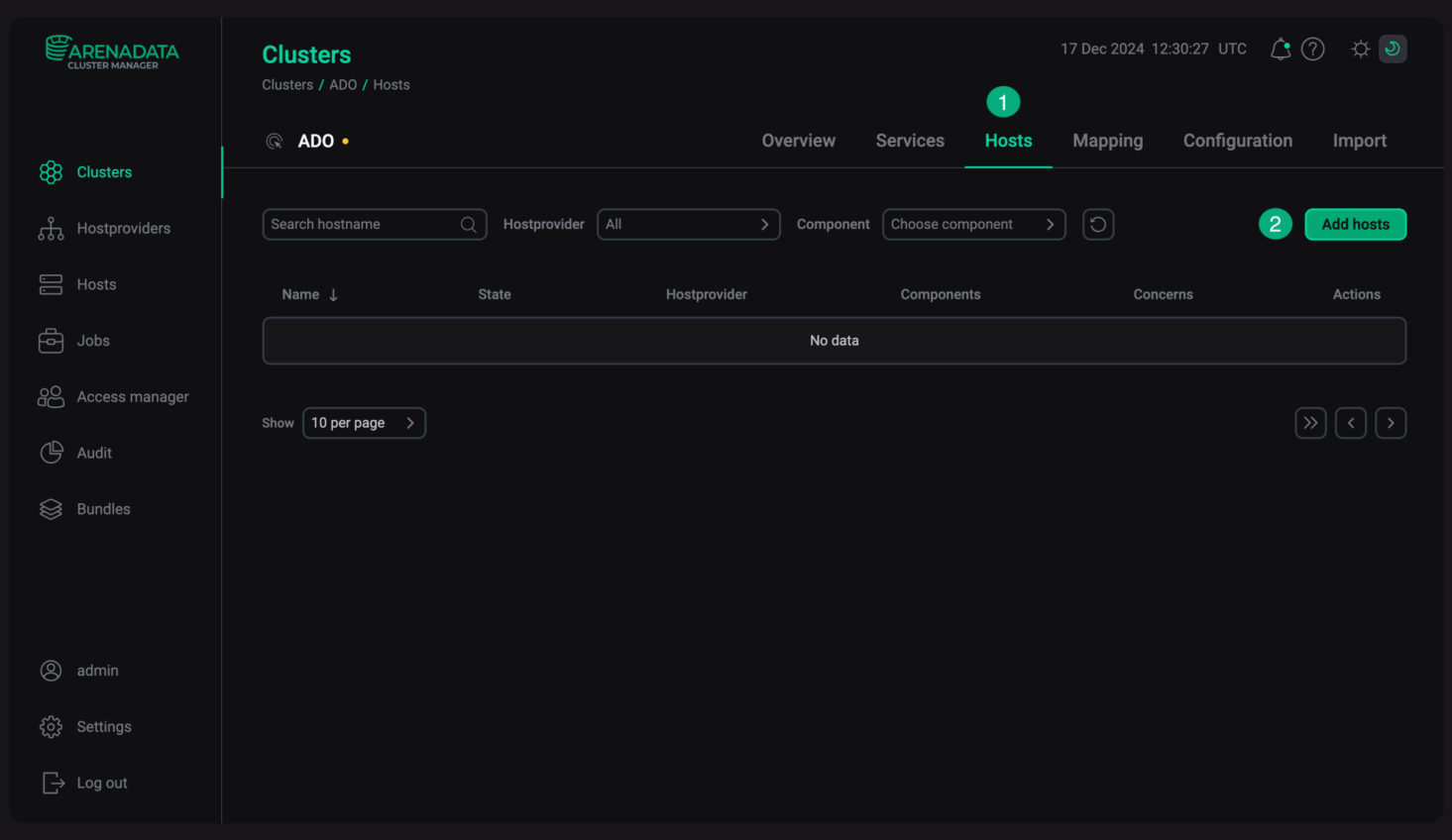
The steps for adding hosts to a cluster are listed below:

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



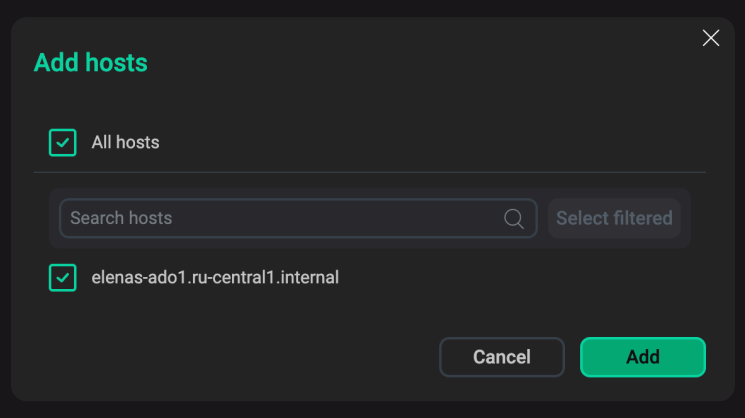
Select a cluster

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



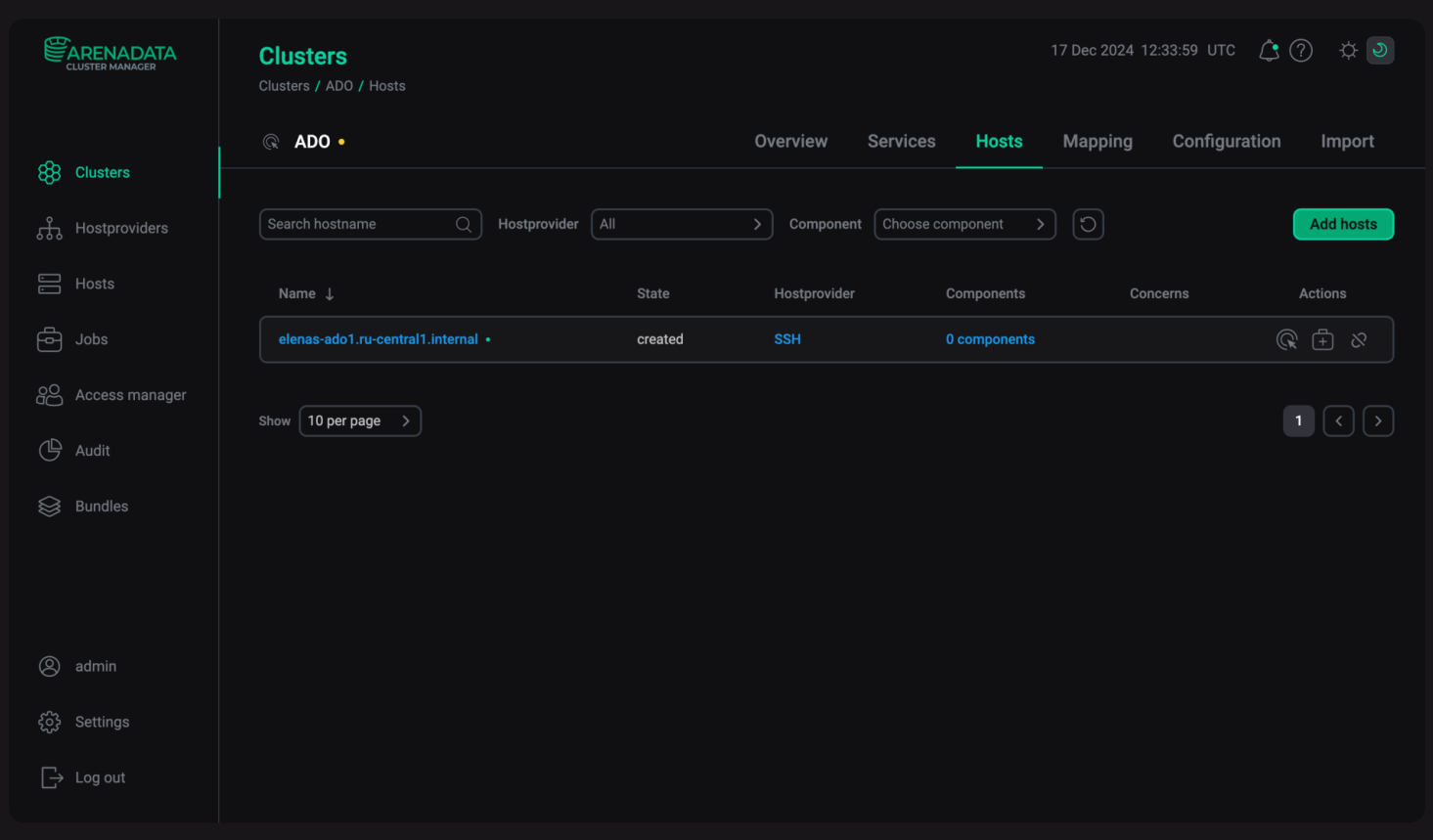
Switch to adding hosts

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

Elena Kostyuchenko, Daria Barysheva

To Table of Contents

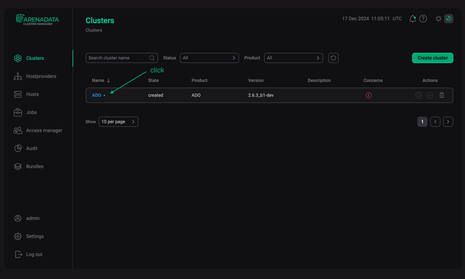
In ADCM, a **component** means a part of a service that should be deployed on one or several cluster hosts. For example, Airflow service can consist of such components as Airflow Server, Airflow Scheduler, Airflow Worker, and others.

IMPORTANT

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

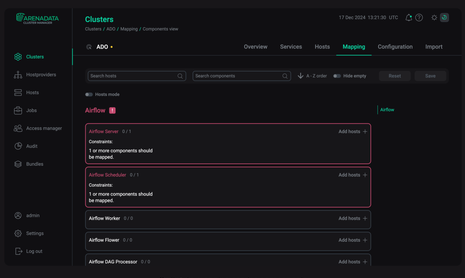
The steps for adding components to hosts are listed below:

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



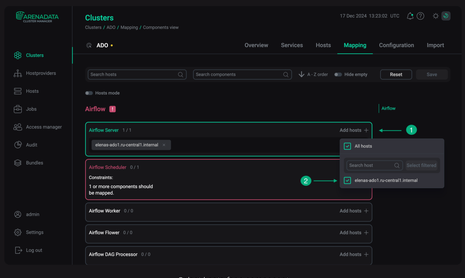
Select a cluster

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



Switch to mapping service components

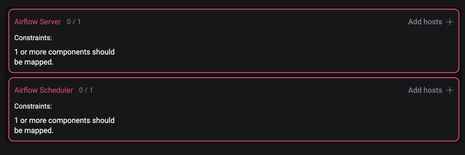
3. In the window that opens, map components to hosts. To assign hosts for each component, click ***Add hosts*** and select hosts in the pop-up window.



Select hosts for a component

Mandatory components

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



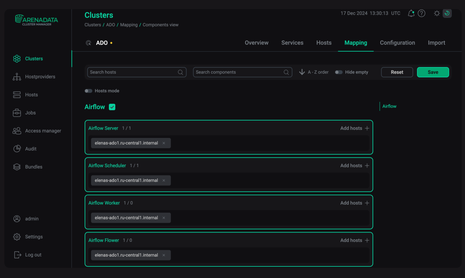
Mandatory components

CAUTION

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

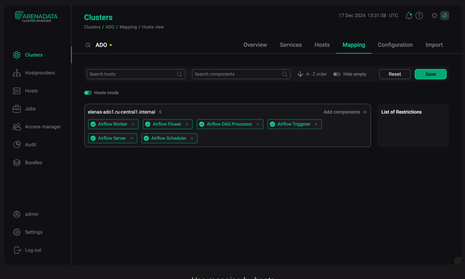
You can install all components in an ADO cluster on a single host. You can add a number of hosts depending on your requirements and distribute the components among them.

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.



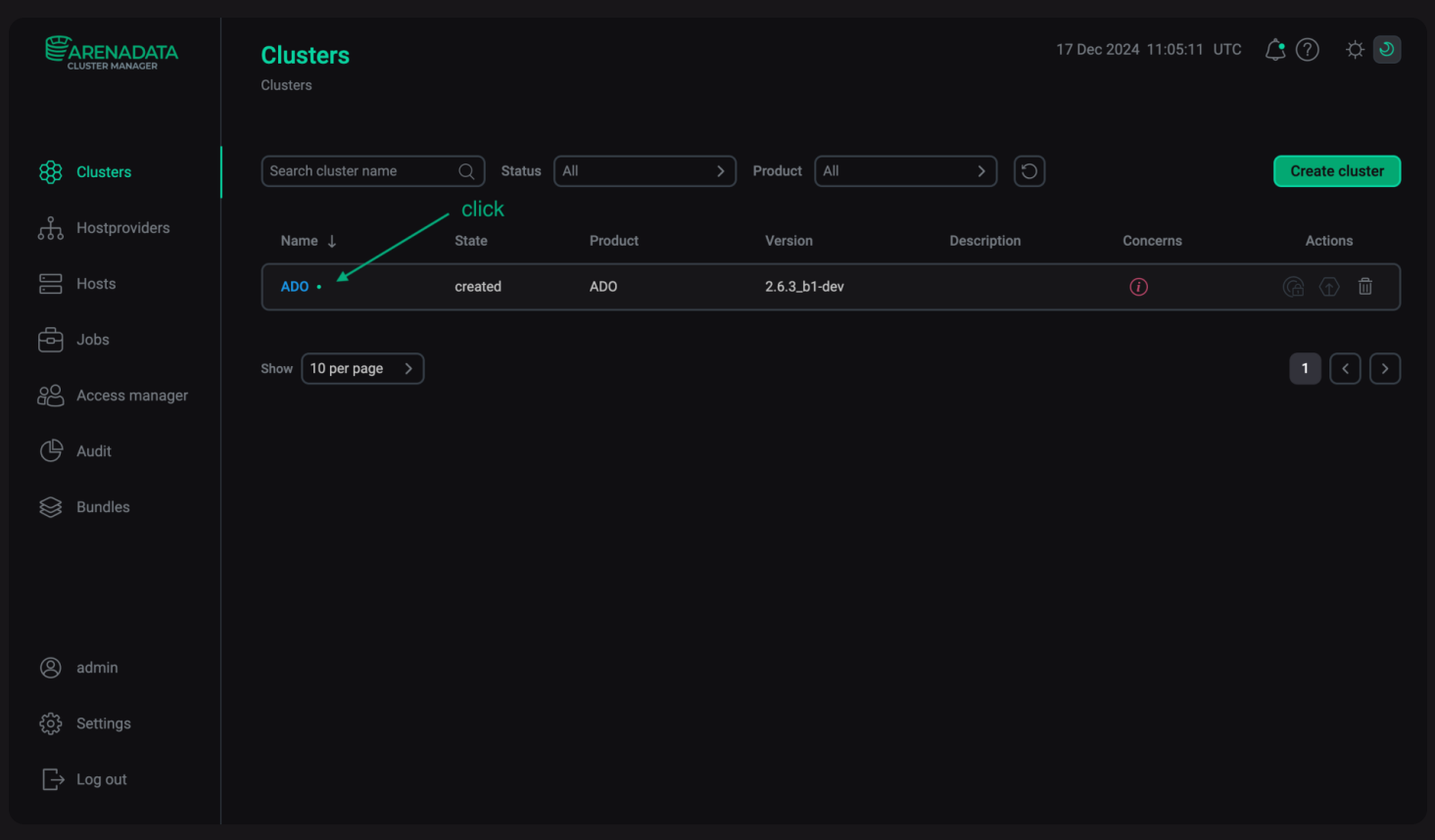
Use mapping by hosts

Configure services

Elena Kostyuchenko, Daria Barysheva

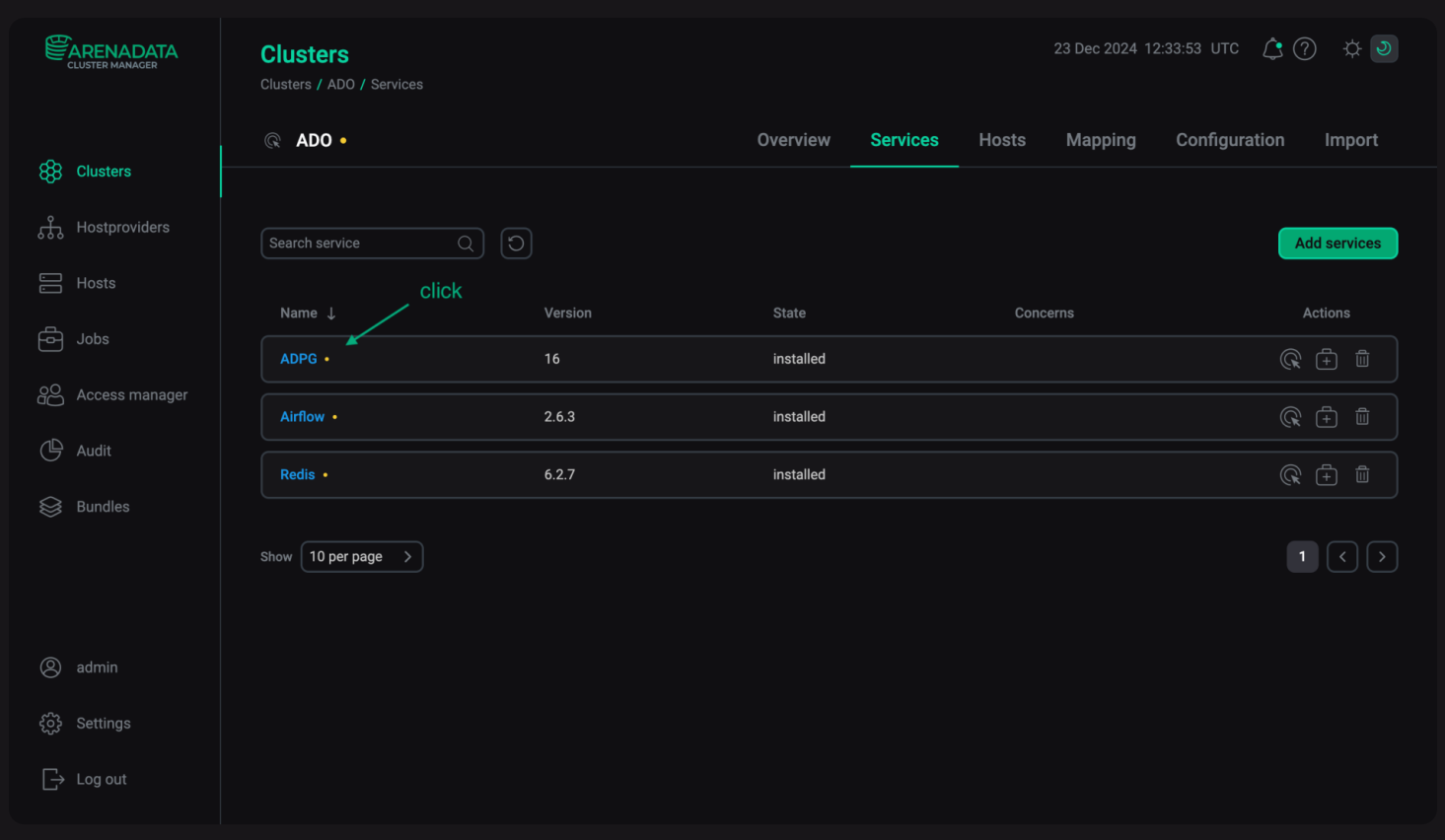
To Table of Contents

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



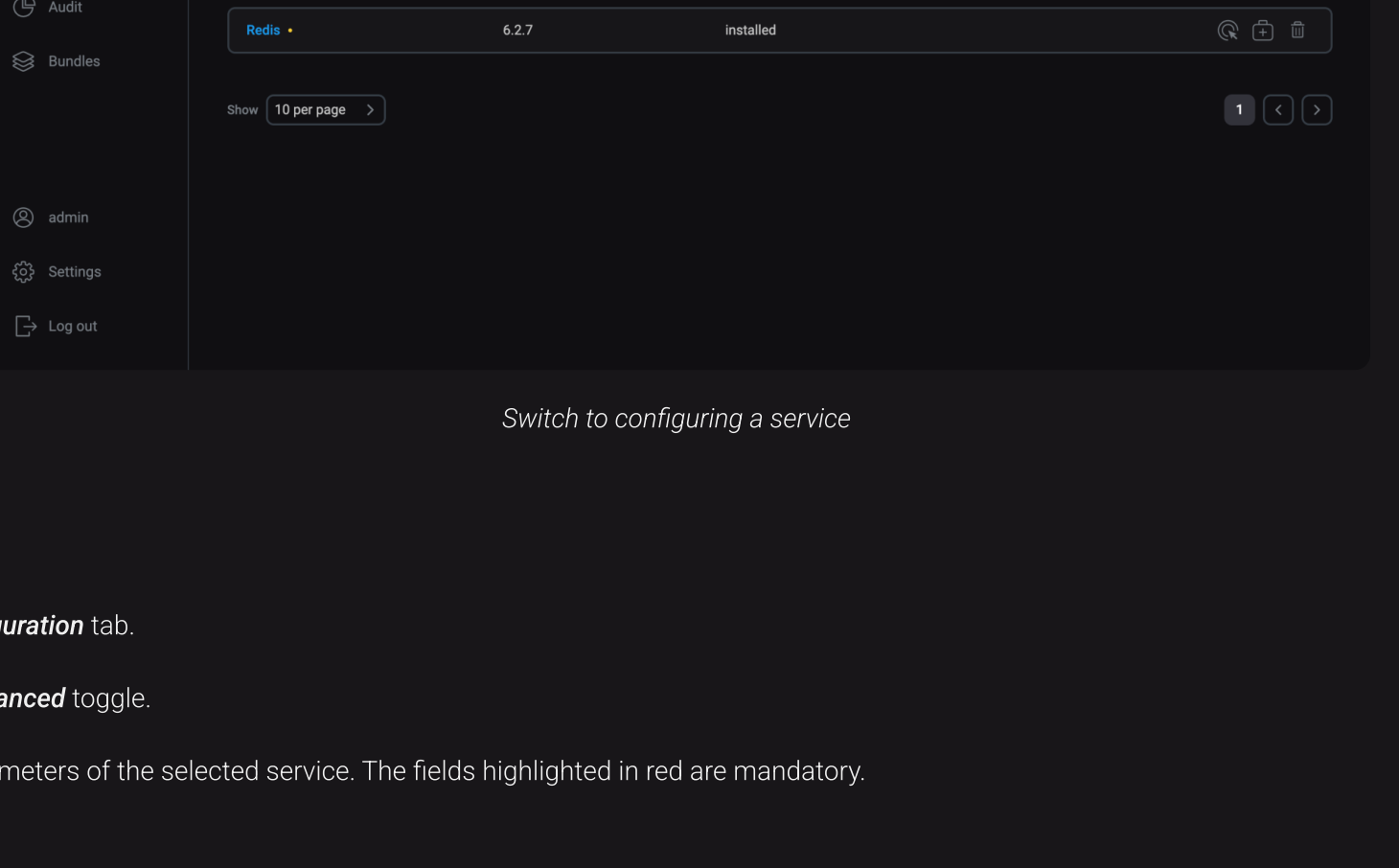
Select a cluster

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

NOTE

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

Configure a cluster

Elena Kostyuchenko, Daria Borysheva

Contents

To Table of Contents

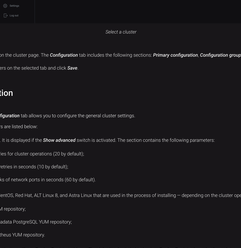
Primary configuration
Configuration groups
Ansible settings

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.

Available configuration parameters are listed below:

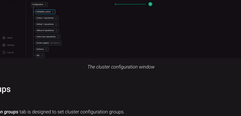
- The ***Reliability control*** section. It is displayed if the ***Show advanced*** switch is activated. The section contains the following parameters:
  - ***retries*** — the number of retries for cluster operations (20 by default),
  - ***delay*** — the delay between retries in seconds (10 by default),
  - ***timeout*** — timeout for checks of network ports in seconds (60 by default).
- Paths to the repositories for CentOS, Red Hat, ALT Linux 8, and Astra Linux that are used in the process of installing — depending on the cluster operation system:
  - ***arenadata*** — Arenadata YUM repository,
  - ***arenadata_postgres*** — Arenadata PostgreSQL YUM repository,
  - ***prometheus_repo*** — Prometheus YUM repository.
- The ***Docker registry*** parameter that specifies a Docker Registry address. The default value is hub.adsw.io — Docker Registry from the Arenadata repository.
- The ***Kerberos*** section that includes settings listed in the table below. Some options are visible only with the ***Show advanced*** option enabled.

Parameter	Description	Default value
Enable Kerberos	Enables the Kerberos authentication	False
Authentication on WEB UIs	Enables the Kerberos authentication on WEB UIs	False
SPNEGO Signature Value	A secret key that is used to generate and verify the signature in the HTTP authentication protocol (SPNEGO (in the Base64 format))	—
Kerberos KDC type	A KDC type. For more information, see Kerberos	MIT
KDC hosts	One or more KDC hosts with running FreeIPA server(s)	—
Realm	A Kerberos realm to connect to the FreeIPA server	—
Domains	One or more domains associated with FreeIPA	—
Kadmin server	A host where `kadmin` is running	—
Kadmin principal	A principal name used to connect via `kadmin`, for example: `admin@EXAMPLE.INTERNAL`	—
Kadmin password	An IPA Admin password	—
Keytabs directory	A directory with keytab files	/etc/security/keytabs
Additional realms	Additional Kerberos realms	—
Delay between kinit invocation attempts	Delay between kinit invocation attempts	5
Number of retries for kinit invocation attempts	Number of retries for kinit invocation attempts	10
Trusted Active Directory server	An Active Directory server for one-way cross-realm trust from the MIT Kerberos KDC	—
Trusted Active Directory realm	An Active Directory realm for one-way cross-realm trust from the MIT Kerberos KDC	—
Custom krb5.conf	Enables adding custom parameters to the *krb5.conf* file	False
krb5.conf	Additional parameters to write to the *krb5.conf* file	—
Admin DN	A full DistinguishedName of the admin user with rights to `create/modify/delete/patchange` user accounts in the target Organizational Unit	—
LDAP URL	LDAP URL that consists of the `ldap://` or `ldaps://` protocol prefix, hostname or IP address, and a port (if any). For example: `ldaps://192.168.4.2:636`	—
Container DN	A container distinguished name	—
TLS CA certificate Path	A CA certificate path on the host filesystem that is pre-located or should be written from the **TLS CA certificate** field	—
TLS CA certificate (optional)	A CA certificate that is written on a host by the path from ***TLS CA certificate Path*** during the execution of the ***Manage Kerberos*** action. The certificate is not saved in ADCM	—
Custom ldap.conf	Enables adding custom parameters to the *ldap.conf* file	False
ldap.conf	Additional parameters that are written to the *ldap.conf* file	—
IpaClient No NTP Autoconf	Disables the NTP configuration during the IPA client installation	False
IpaClient No DNS Lookup	Disables the DNS lookup for the FreeIPA server during the IPA client installation	True
Custom smb.conf	Usage of custom configuration file *smb.conf*	False
smb.conf	Custom configuration file *smb.conf*	—
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	True
Configure services and clients	Enables updating of services and clients configuration	True
Run service checks	Enables service check runs	True

- **SSL** section with the settings listed in the table below:

Parameter	Description	Default value
Enable SSL	Enables SSL	false
Keystore path	A path to the keystore file	/etc/ssl/keystore.jks
Keystore password	A password for the keystore file	—
Truststore path	A path to the truststore file	/etc/ssl/truststore.jks
Truststore password	A password for the truststore file	—
TLS Version	Version of the TLS protocol	TLSv1.2
Run service checks	Enables service check runs	true

- The ***Precheck packages*** parameter that specifies whether the availability of the packages in the repository should be checked before installation. `False` by default.
- The ***Cluster UUID*** displays the cluster ID information.

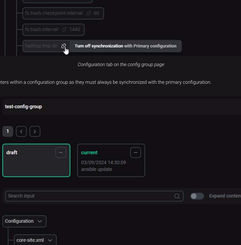


The cluster configuration window

Configuration groups

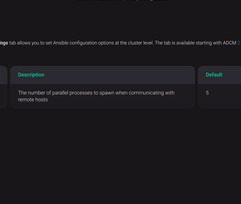
The **Configuration** → ***Configuration groups*** tab is designed to set cluster configuration groups.

To configure parameters in a configuration group, hover over the parameters that you wish to configure differently, click the **Turn off synchronization** icon, edit them, and click ***Save***.



Configuration tab on the config group page

You cannot change some parameters within a configuration group as they must always be synchronized with the primary configuration.



Parameter bound to the primary configuration

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

Elena Kostyuchenko, Daria Barysheva

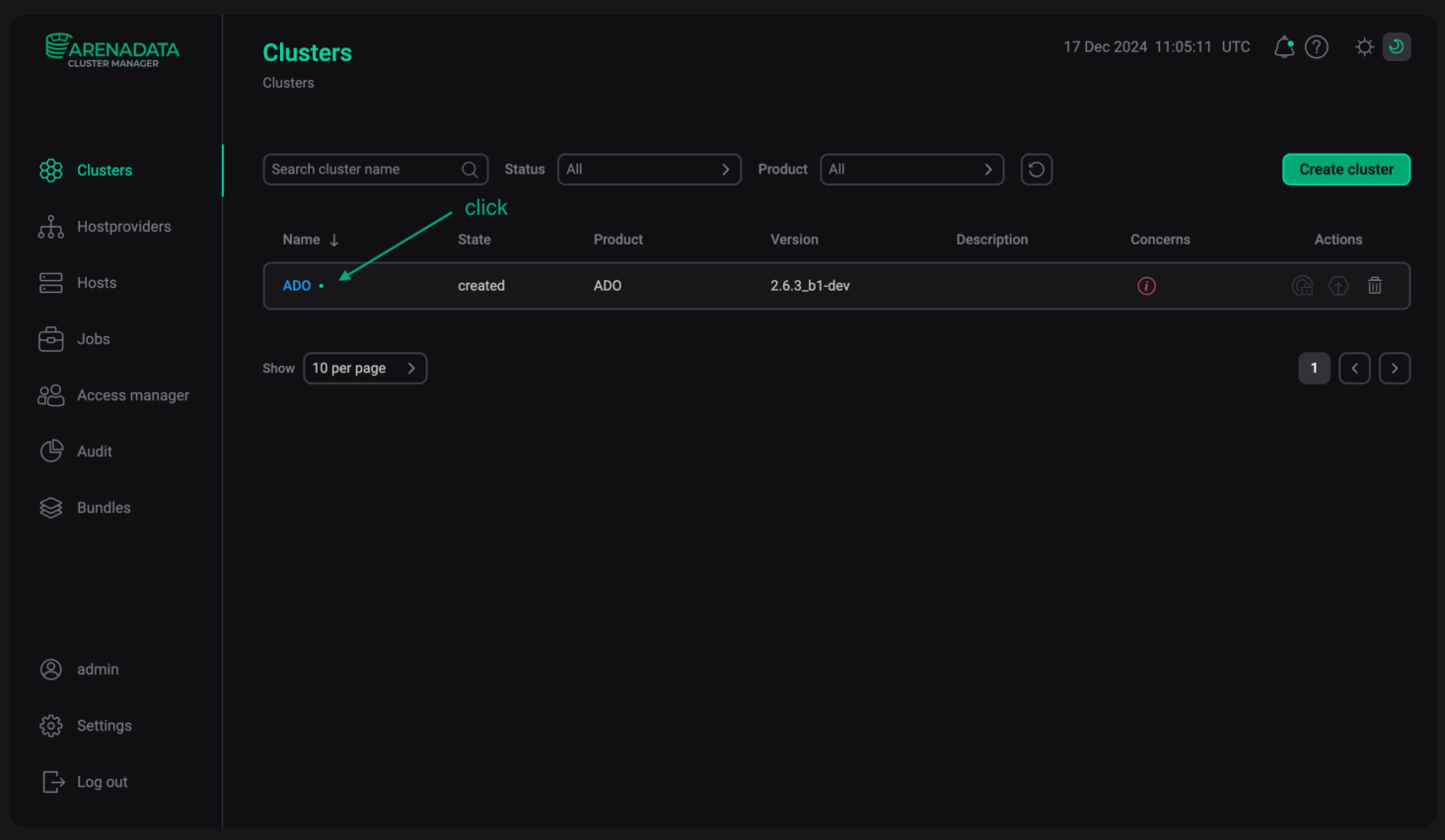
To Table of Contents

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.

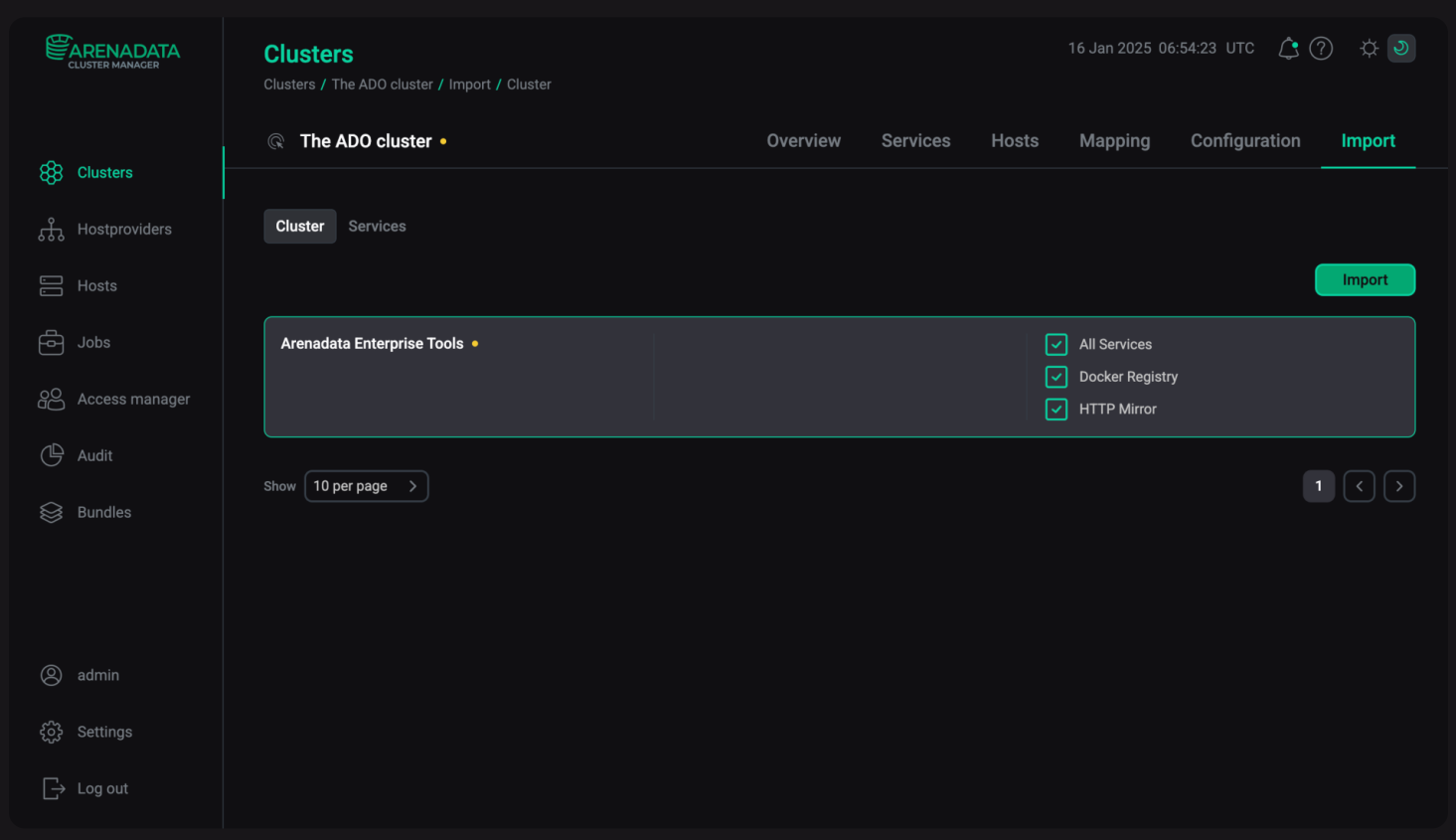


Select a cluster

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

2. Elena Kostyuchenko, Daria Baryshena

Contents

To Table of Contents

Step 1. Installation
- Whole cluster
- Single services

Step 2. View installation results

Step 3. Check the cluster state after the installation

Step 1. 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.

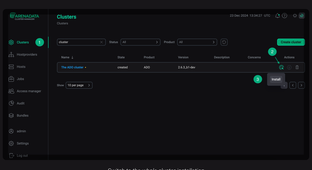
IMPORTANT

The installation must be performed on a clean host without prior installation of related or similar software.

Whole cluster

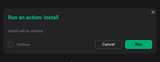
To install all cluster services within a single action, follow the steps:

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. Verify the requested action in the opened window.



Verify the action

3. Wait until the job is completed.

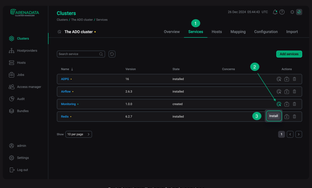
Single services

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

CAUTION

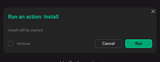
Notice that services are installed one by one. Do not install another service until the installation of the previous one is completed.

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



Switch to installation of single services

2. Verify the action in the opened window.

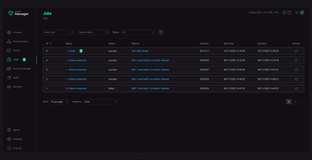


Verify the action

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

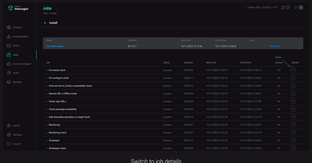
Step 2. View installation results

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. The window with job inner steps is opened. Click the ellipsis in the ***Details*** column for additional information on the current step.

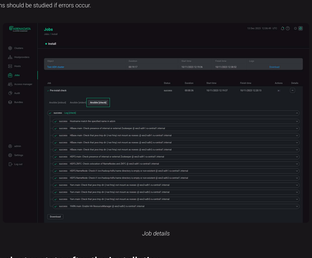


Switch to job details

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.

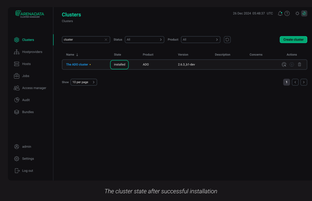


Job details

Step 3. Check the cluster state after the installation

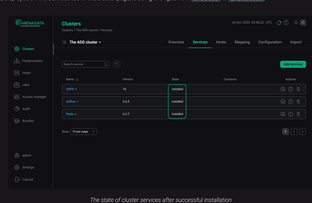
As a result of successful installation, the cluster and its services change their state according to the following rules:

- 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

Contents

To Table of Contents

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

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

Overview

When you add the Monitoring service to an ADO cluster, node exporter is installed on all hosts. It is a monitoring agent that reads host system metrics that Prometheus will collect. Prometheus will also collect metrics from ADO services. These metrics will be available in the Prometheus format on ports and endpoints specified in service configuration options. 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 ADO metrics. To do this, configure access to ADO metrics in your monitoring system using ADO's Prometheus parameters specified in the Monitoring service configuration.
> - You can also use the Federation mechanism to migrate all metrics from the Prometheus server deployed in ADO to your Prometheus.

Step 1. Add the service

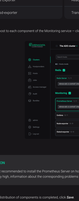
1. In the ADO UI interface, open the **Clusters** page and click your ADO cluster name. On the cluster page that opens, switch to the **Services** tab and click **Add services**.

Switch to adding services

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

Select the service

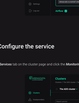
As a result, the added service is displayed in the **Services** tab.



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.



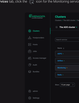
Switch to mapping service components

Each component should be installed on one host of the cluster.

Monitoring service components

Component	Description
Prometheus Server	Deploys a Prometheus server that serves as a - proxy channel for all collections of metrics on a host, - storage for all metrics of a cluster, - generator of alerts based on collected metrics.
Grafana	Visualizes ADO metrics as graphs and charts organized into dashboards
Node Exporter	Reads host system metrics that Prometheus will collect
Statsd-exporter	Translates StatsD-style metrics to Prometheus metrics via configured mapping rules

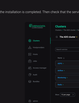
2. Assign a host to each component of the Monitoring service – click **Add hosts** and select the desired host in the pop-up window.



Select a host for a component

> **CAUTION**
>
> It is not recommended to install the Prometheus Server on hosts with ADO – use a separate host for it. Otherwise, if an ADO host fails and/or the load on it is critically high, information about the corresponding problems will not be saved.

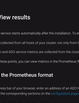
3. After the distribution of components is completed, click **Save**.



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 configuration parameters – see the **Monitoring** section in the 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 the ⋮ icon 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 completed. Then check that the service state has changed from `prepared` → `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 the 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 system and ADO service metrics are collected from the cluster hosts.

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.

Metrics in the Prometheus format

1. In the address bar of your browser, enter an address of an ADO host with the specified port and endpoint to listen for service or system metrics. Port numbers and endpoints are defined in the corresponding sections on the configuration page of the Monitoring service:
   - **ADO's service metric settings** – settings for access to monitoring metrics of ADO services.
   - **Node Exporter settings** – settings for access to system metrics of the ADO cluster hosts.

   For example, http://10.62.48.107:11003/metrics is an address to view metrics of the ADO service on a host with IP `10.62.48.107`.
2. The page that opens will display monitoring metrics from the specified host of the ADO cluster in the Prometheus format.

Airflow server metrics in the Prometheus format

Prometheus web interface

1. In the address bar of your browser, enter an IP address of a host where the Prometheus Server component is installed. Specify also a port number from the **Alerts_address** parameter in the **Prometheus settings** section of the Monitoring service configuration (the default port is `19011`). For example, http://10.62.48.107:19011. To log into the Prometheus interface, use a username and password that are also specified in the **Prometheus settings** section – the **Prometheus users to login/input to Prometheus** setting.
2. In the **Expression** field, you can select a metric and click **Execute** – values of the metric on all hosts of the ADO cluster will be shown in the histogram.

Prometheus web interface

Grafana web interface

1. In the address bar of your browser, enter an address of a host on which Grafana is deployed and add a port number – the value of the **Grafana Web port** parameter located in the **Grafana settings** section of the Monitoring service configuration (the default value is `19300`). For example, http://10.62.48.107:19300. To log in, use **admin** as a username, and the **Grafana administrator's password** parameter value (also found in the **Grafana settings** section of service configuration parameters) as a password.
2. In the window that opens, navigate to **Home** → **Dashboards** and expand the **ADO Dashboard -ADO_cluster_name** section. In this section, you can select one of available dashboards to view service or system metrics coming from your ADO cluster.

View ADO metrics in Grafana

Connect to Airflow via CLI

Elena Kostyuchenko

Contents

To Table of Contents

Overview

Usage examples

Overview

Airflow provides a command line interface implemented as the *bin/airflow* script located in the */opt/airflow/bin* directory.

While it is recommended to manage Airflow via ADCM or the Airflow web UI, the CLI lets you access an extended number of additional options for working with Airflow.

To interact with Airflow via CLI, connect to a cluster host with an Airflow component via SSH and execute the script by running:

```
$ sudo /opt/airflow/bin/airflow
```

The output looks like this:

```
Usage: airflow [-h] GROUP_OR_COMMAND ...

Positional Arguments:
  GROUP_OR_COMMAND

    Groups:
      celery         Celery components
      config         View configuration
      connections    Manage connections
      dags           Manage DAGs
      db             Database operations
      jobs           Manage jobs
      kubernetes     Tools to help run the KubernetesExecutor
      pools          Manage pools
      providers      Display providers
      roles          Manage roles
      tasks          Manage tasks
      users          Manage users
      variables      Manage variables

    Commands:
      cheat-sheet    Display cheat sheet
      dag-processor  Start a standalone Dag Processor instance
      info           Show information about current Airflow and environment
      kerberos       Start a kerberos ticket renewer
      plugins        Dump information about loaded plugins
      rotate-fernet-key
                     Rotate encrypted connection credentials and variables
      scheduler      Start a scheduler instance
      standalone     Run an all-in-one copy of Airflow
      sync-perm      Update permissions for existing roles and optionally DAGs
      triggerer      Start a triggerer instance
      version        Show the version
      webserver      Start a Airflow webserver instance

Options:
  -h, --help         show this help message and exit
```

The detailed description of each command is available in the Airflow2 CLI command reference.

For example, check the version by running:

```
$ sudo /opt/airflow/bin/airflow version
```

The output:

```
2.6.3
```

You can display help for a specific command by using the `--help` option, for example:

```
$ sudo /opt/airflow/bin/airflow dags --help
```

The commands have the following syntax:

```
$ airflow [GROUP_OR_COMMAND] [SUB_COMMAND] [OPTIONS]
```

Where:

- `GROUP_OR_COMMAND` — an Airflow CLI command or a group of commands like `dags`, `jobs`, etc.
- `SUB_COMMAND` — a command related to a group, for example, `dags list`.
- `OPTIONS` — the selected command's options.

Usage examples

Airflow provides several tools, one of which is the `dags` command, which allows you to manage DAGs using the CLI.

For example, to get a list of existing DAGs, run:

```
$ sudo /opt/airflow/bin/airflow dags list
```

Example output:

```
dag_id                                                | filepath
| owner   | paused
=====================================================+============================================================================================================+=========+=======
dataset_consumes_1                                    | /opt/airflow/lib/python3.10/site-
packages/airflow/example_dags/example_datasets.py | airflow | None
dataset_consumes_1_and_2                              | /opt/airflow/lib/python3.10/site-
packages/airflow/example_dags/example_datasets.py | airflow | None
dataset_consumes_1_never_scheduled                    | /opt/airflow/lib/python3.10/site-
packages/airflow/example_dags/example_datasets.py | airflow | None
dataset_consumes_unknown_never_scheduled              | /opt/airflow/lib/python3.10/site-
packages/airflow/example_dags/example_datasets.py | airflow | None
dataset_produces_1                                    | /opt/airflow/lib/python3.10/site-
packages/airflow/example_dags/example_datasets.py | airflow | None
dataset_produces_2                                    | /opt/airflow/lib/python3.10/site-
packages/airflow/example_dags/example_datasets.py | airflow | None
example_bash_operator                                 | /opt/airflow/lib/python3.10/site-
packages/airflow/example_dags/example_bash_operat | airflow | None
                                                      | or.py
|         |
example_branch_datetime_operator                      | /opt/airflow/lib/python3.10/site-
packages/airflow/example_dags/example_branch_date | airflow | None
                                                      | time_operator.py
|         |
example_branch_datetime_operator_2                    | /opt/airflow/lib/python3.10/site-
packages/airflow/example_dags/example_branch_date | airflow | None
                                                      | time_operator.py
|         |
example_branch_datetime_operator_3                    | /opt/airflow/lib/python3.10/site-
packages/airflow/example_dags/example_branch_date | airflow | None
                                                      | time_operator.py
|         |
example_kubernetes_executor                           | /opt/airflow/lib/python3.10/site-
packages/airflow/example_dags/example_kubernetes_ | airflow | None
                                                      | executor.py
|         |
example_local_kubernetes_executor                     | /opt/airflow/lib/python3.10/site-
packages/airflow/example_dags/example_local_kuber | airflow | None
                                                      | netes_executor.py
|         |
example_nested_branch_dag                             | /opt/airflow/lib/python3.10/site-
packages/airflow/example_dags/example_nested_bran | airflow | None
                                                      | ch_dag.py
|         |
example_params_trigger_ui                             | /opt/airflow/lib/python3.10/site-
packages/airflow/example_dags/example_params_trig | airflow | None
                                                      | ger_ui.py
|         |
tutorial                                              | /opt/airflow/lib/python3.10/site-
packages/airflow/example_dags/tutorial.py         | airflow | None
tutorial_dag                                          | /opt/airflow/lib/python3.10/site-
packages/airflow/example_dags/tutorial_dag.py     | airflow | None
tutorial_taskflow_api                                 | /opt/airflow/lib/python3.10/site-
packages/airflow/example_dags/tutorial_taskflow_a | airflow | None
                                                      | pi.py
|         |
```

To export the list of DAGs to a file, use the `--output` option and provide one of the following formats as the argument: `json`, `yaml`, `plain`, or `table`.

To run a DAG, use the `trigger` command:

```
$ sudo /opt/airflow/bin/airflow dags trigger tutorial
```

Here `tutorial` is the ID of the DAG to run.

Connect to Airflow via REST API

by Ana Gallegos Ortiz

Contents

To Table of Contents

Overview

Example requests

- List pools
- Check import issues
- Get information about a DAG
- Trigger a DAG run
- Get tasks

Overview

Along with the web interface, Airflow provides access to its REST API. This article contains a few examples of how to work with DAGs via the API using curl. The full list of endpoints is available in the Airflow REST API Specification.

A URL for Airflow API requests has the following format:

```
http://<server-address>:<port>/api/v1/<endpoint>
```

Where:

- **<server-address>** — the IP address or the host name of the Airflow server;
- **<port>** — the port of the Airflow server;
- **<endpoint>** — the endpoint of the desired method.

Most endpoints accept JSON as input and return JSON in response. Add the following header when making requests:

```
Content-Type: application/json
```

The example request:

```
$ curl -L 'http://localhost:8080/api/v1/dags' -H 'Content-Type: application/json' --user "admin:admin"
```

This command requests information about existing DAGs in the JSON format.

Output sample:


```
{
    "dag_id": "example_params_ui_tutorial",
    "default_view": "grid",
    "description": "DAG demonstrating various options for a trigger form generated by DAG params",
    "file_token": "
eJwNyIEOgCAMRD/GLuIsZPsmXOBmE2Hd5Fnu/JmcWxYOqeHSeFlPB_sYUmTnYIOvmFJBmQrFqiMqZtIQuIbPOhDDcjIA
MdqqqSBtRLfFCHVC_PVqUEsSnXuKgA",
    "fileloc": "/opt/airflow/lib/python3.10/site-packages/airflow/example_dags/example_params_ui_tutorial.py",
    "has_import_errors": false,
    "has_task_concurrency_limits": false,
    "is_active": true,
    "is_paused": true,
    "is_subdag": false,
    "last_expired": null,
    "last_parsed_time": "2024-08-29T13:16:21.591951+00:00",
    "last_pickled": null,
    "max_active_runs": 16,
    "max_consecutive_failed_dag_runs": 0,
    "next_dagrun": null,
    "next_dagrun_create_after": null,
    "next_dagrun_data_interval_end": null,
    "next_dagrun_data_interval_start": null,
    "owners": [
      "airflow"
    ],
    "pickle_id": null,
    "root_dag_id": null,
    "scheduler_lock": null,
    "tags": [
      {
        "name": "example_ui"
      }
    ],
    "timetable_description": "Never, external triggers only"
  },
```


Example requests

Here are the examples of Airflow REST API requests. The examples that contain the **<dag-id>** placeholder require a DAG ID. Provide the ID of the required DAG instead of the placeholder.

For authorization, examples use the default credentials: the **admin** user with the password **admin**.

List pools

To list the existing Airflow pools, execute a **GET** request to the **/pools** endpoint.

For example:

```
$ curl -X GET 'http://localhost:8080/api/v1/pools' \
  -H 'Content-Type: application/json' --user "admin:admin"
```

The output:


```
{
  "pools": [
    {
      "description": "Default pool",
      "name": "default_pool",
      "occupied_slots": 0,
      "open_slots": 128,
      "queued_slots": 0,
      "running_slots": 0,
      "scheduled_slots": 7,
      "slots": 128
    }
  ],
  "total_entries": 1
}
```


Check import issues

To check if there are any DAG import issues, send a request to the **/importErrors** endpoint.

For example:

```
$ curl --verbose 'http://localhost:8080/api/v1/importErrors' -H 'Content-Type: application/json' --user "admin:admin"
```

The output:


```
{
  "import_errors": [
    {
      "filename": "/opt/airflow/dags/second_example.py",
      "import_error_id": 1,
      "stack_trace": "Traceback (most recent call last):\n  File \"<frozen importlib._bootstrap_external>\", line 947, in
source_to_code\n  File \"<frozen importlib._bootstrap>\", line 241, in _call_with_frames_removed\n  File
\"/opt/airflow/dags/second_example.py\", line 24\n    import pandas as pdin\n                  ^^^^^^\nSyntaxError: invalid syntax\n",
      "timestamp": "2024-09-03T13:34:12.000000+00:00"
    }
  ],
  "total_entries": 1
}
```


Get information about a DAG

To retrieve information about a specific DAG, send a **GET** request with the DAG's ID to the **/details** endpoint.

For example:

```
$ curl -v -X GET http://localhost:8080/api/v1/dags/<dag-id>/details \
  -H 'Content-Type: application/json' \
  --user "admin:admin"
```

The output:


```
{
  "catchup": false,
  "concurrency": 16,
  "dag_id": "tutorial_dag",
  "dag_run_timeout": null,
  "default_view": "grid",
  "description": "DAG tutorial",
  "doc_md": "\nDAG 101: Demonstrating\n\n\nThis DAG is demonstrating an Extract -> Transform -> Load pipeline\n",
  "end_date": null,
  "file_token": "eJwNyAIQEAMAN_2cCeYJKUTYnHo3C_lXuHr_yAqqY1JdKHdnbGAeSsVnQNQv-bY-fnSzQE9U_BnQFCbVqqcl-vHVE-MBnkEk=.dAQ
eqvvvIJ0eSSEFBJbFldrDrDIBM",
  "fileloc": "/opt/airflow/lib/python3.10/site-packages/airflow/example_dags/tutorial_dag.py",
  "is_active": true,
  "is_paused": true,
  "is_subdag": false,
  "last_parsed": "2024-09-03T13:28:29.979933+00:00",
  "max_active_runs": 16,
  "max_active_tasks": 16,
  "orientation": "LR",
  "owners": [
    "airflow"
  ],
  "params": {},
  "pickle_id": null,
  "render_template_as_native_obj": false,
  "schedule_interval": null,
  "start_date": "2021-01-01T00:00:00+00:00",
  "tags": [
    {
      "name": "example"
    }
  ],
  "timezone": "Timezone('UTC')"
}
```


Trigger a DAG run

To trigger a DAG, send a **POST** request that contains the DAG run parameters to the **/dagRuns** endpoint.

For example:

```
$ curl -v -X POST http://localhost:8080/api/v1/dags/<dag-id>/dagRuns \
  -H 'Content-Type: application/json' \
  --user "admin:admin" \
  --data-raw '{}'
```

Where **{}** are the parameters of the DAG run in the JSON format.

For example:

```
{
  "dag_run_id": "string",
  "logical_date": "2019-08-24T14:15:22Z",
  "execution_date": "2019-08-24T14:15:22Z",
  "conf": {},
  "note": "string"
}
```

To set a specific execution date (**logical_date**) for your DAG, pass a timestamp in the **YYYY-MM-DDTHH:mm:ssZ** format.

The output:


```
{
  "conf": {},
  "dag_id": "tutorial_dag",
  "dag_run_id": "string",
  "data_interval_end": "2019-08-24T14:15:22+00:00",
  "data_interval_start": "2019-08-24T14:15:22+00:00",
  "end_date": null,
  "execution_date": "2019-08-24T14:15:22+00:00",
  "external_trigger": true,
  "last_scheduling_decision": null,
  "logical_date": "2019-08-24T14:15:22+00:00",
  "note": "string",
  "run_type": "manual",
  "start_date": null,
  "state": "queued"
}
```


Get tasks

To see the list of tasks for a DAG, send a **GET** request to the DAG's **/tasks** endpoint.

The example request:

```
$ curl -v -X GET http://localhost:8080/api/v1/dags/<dag-id>/tasks \
  -H 'Content-Type: application/json' \
  --user "admin:admin"
```

The output


```
{
  "tasks": [
    {
      "class_ref": {
        "class_name": "PythonOperator",
        "module_path": "airflow.operators.python"
      },
      "depends_on_past": false,
      "downstream_task_ids": [
        "transform_people"
      ],
      "end_date": null,
      "execution_timeout": null,
      "extra_links": [],
      "is_mapped": false,
      "operator_name": "PythonOperator",
      "owner": "airflow",
      "params": {},
      "pool": "default_pool",
      "pool_slots": 1.0,
      "priority_weight": 1.0,
      "queue": "default",
      "retries": 0.0,
      "retry_delay": {
        "__type": "TimeDelta",
        "days": 0,
        "microseconds": 0,
        "seconds": 300
      },
      "retry_exponential_backoff": false,
      "start_date": "2021-01-17T00:00:00+00:00",
      "task_id": "extract_people",
      "template_fields": [
        "templates_dict",
        "op_args",
        "op_kwargs"
      ],
      "trigger_rule": "all_success",
      "ui_color": "#ffefeb",
      "ui_fgcolor": "#000",
      "wait_for_downstream": false,
      "weight_rule": "downstream"
    },
    {
      "class_ref": {
        "class_name": "PythonOperator",
        "module_path": "airflow.operators.python"
      },
      "depends_on_past": false,
      "downstream_task_ids": [],
      "end_date": null,
      "execution_timeout": null,
      "extra_links": [],
      "is_mapped": false,
      "operator_name": "PythonOperator",
      "owner": "airflow",
      "params": {},
      "pool": "default_pool",
      "pool_slots": 1.0,
      "priority_weight": 1.0,
      "queue": "default",
      "retries": 0.0,
      "retry_delay": {
        "__type": "TimeDelta",
        "days": 0,
        "microseconds": 0,
        "seconds": 300
      },
      "retry_exponential_backoff": false,
      "start_date": "2021-01-17T00:00:00+00:00",
      "task_id": "transform_people",
      "template_fields": [
        "templates_dict",
        "op_args",
        "op_kwargs"
      ],
      "trigger_rule": "all_success",
      "ui_color": "#ffefeb",
      "ui_fgcolor": "#000",
      "wait_for_downstream": false,
      "weight_rule": "downstream"
    },
    {
      "class_ref": {
        "class_name": "PythonOperator",
        "module_path": "airflow.operators.python"
      },
      "depends_on_past": false,
      "downstream_task_ids": [
        "load_people"
      ],
      "end_date": null,
      "execution_timeout": null,
      "extra_links": [],
      "is_mapped": false,
      "operator_name": "PythonOperator",
      "owner": "airflow",
      "params": {},
      "pool": "default_pool",
      "pool_slots": 1.0,
      "priority_weight": 1.0,
      "queue": "default",
      "retries": 0.0,
      "retry_delay": {
        "__type": "TimeDelta",
        "days": 0,
        "microseconds": 0,
        "seconds": 300
      },
      "retry_exponential_backoff": false,
      "start_date": "2021-01-17T00:00:00+00:00",
      "task_id": "transform_people",
      "template_fields": [
        "templates_dict",
        "op_args",
        "op_kwargs"
      ],
      "trigger_rule": "all_success",
      "ui_color": "#ffefeb",
      "ui_fgcolor": "#000",
      "wait_for_downstream": false,
      "weight_rule": "downstream"
    }
  ],
  "total_entries": 3
}
```

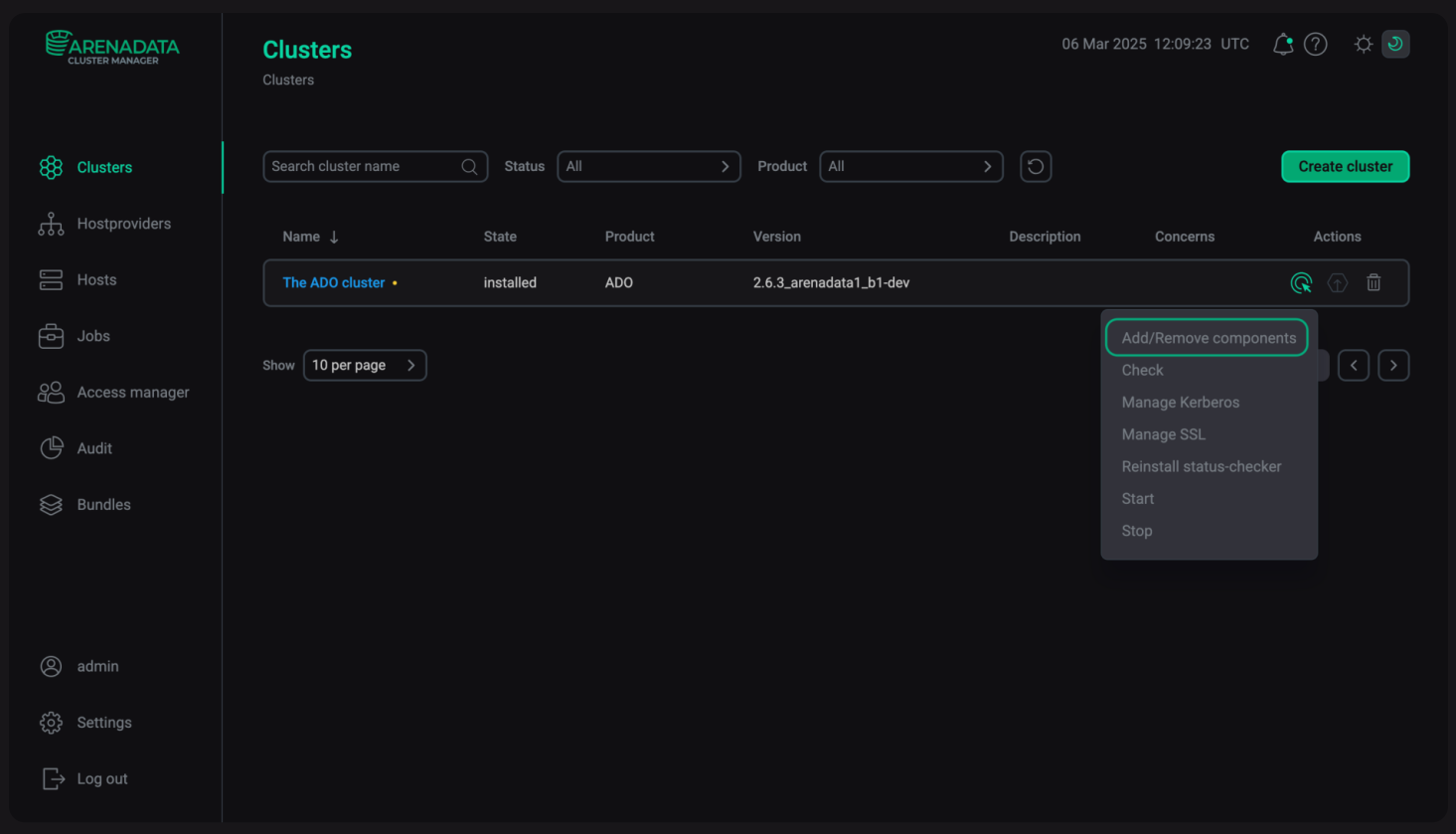
Add/Remove components

Elena Kostyuchenko

To Table of Contents

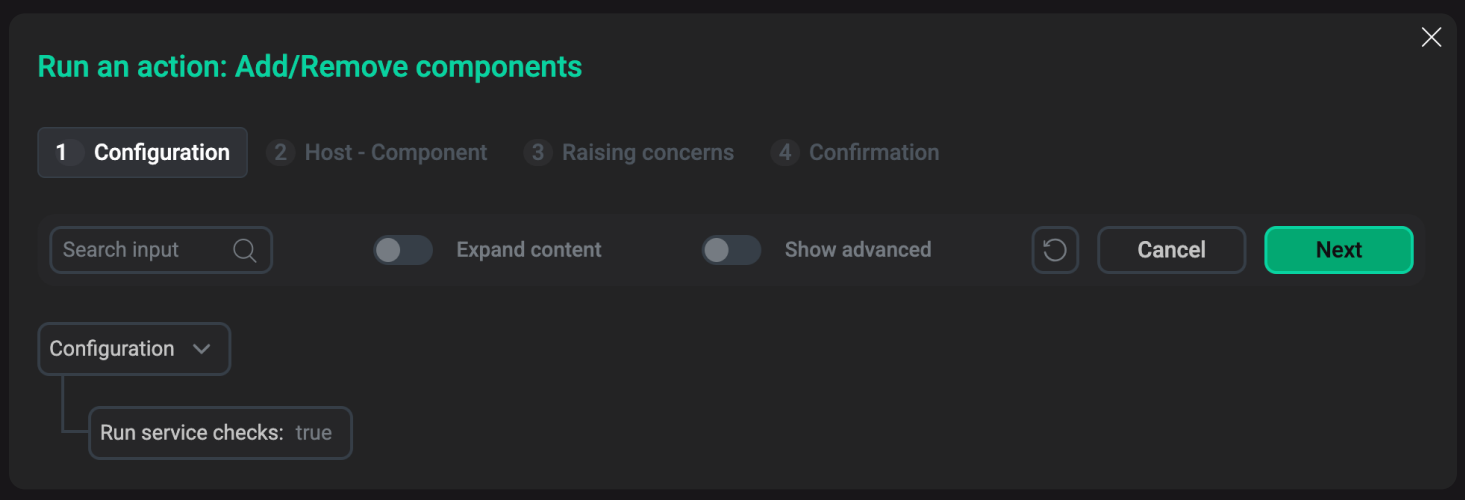
The *Add/Remove components* action opens the omponent-host mapping interfac where you can add, remove, and distribute service components.

To run an action, go to the *Clusters* page, click in the *Actions* column, and select the action from the drop-down list that appears.



Execute the Add/Remove components action

The dialog window of the *Add/Remove components* action contains the *Run service checks* option that determines whether to run the hec action for services after adding or removing components.



The Run an action window

Alternatively, you can use the *Add/Remove components* action of a particular service to manually change the number of components.

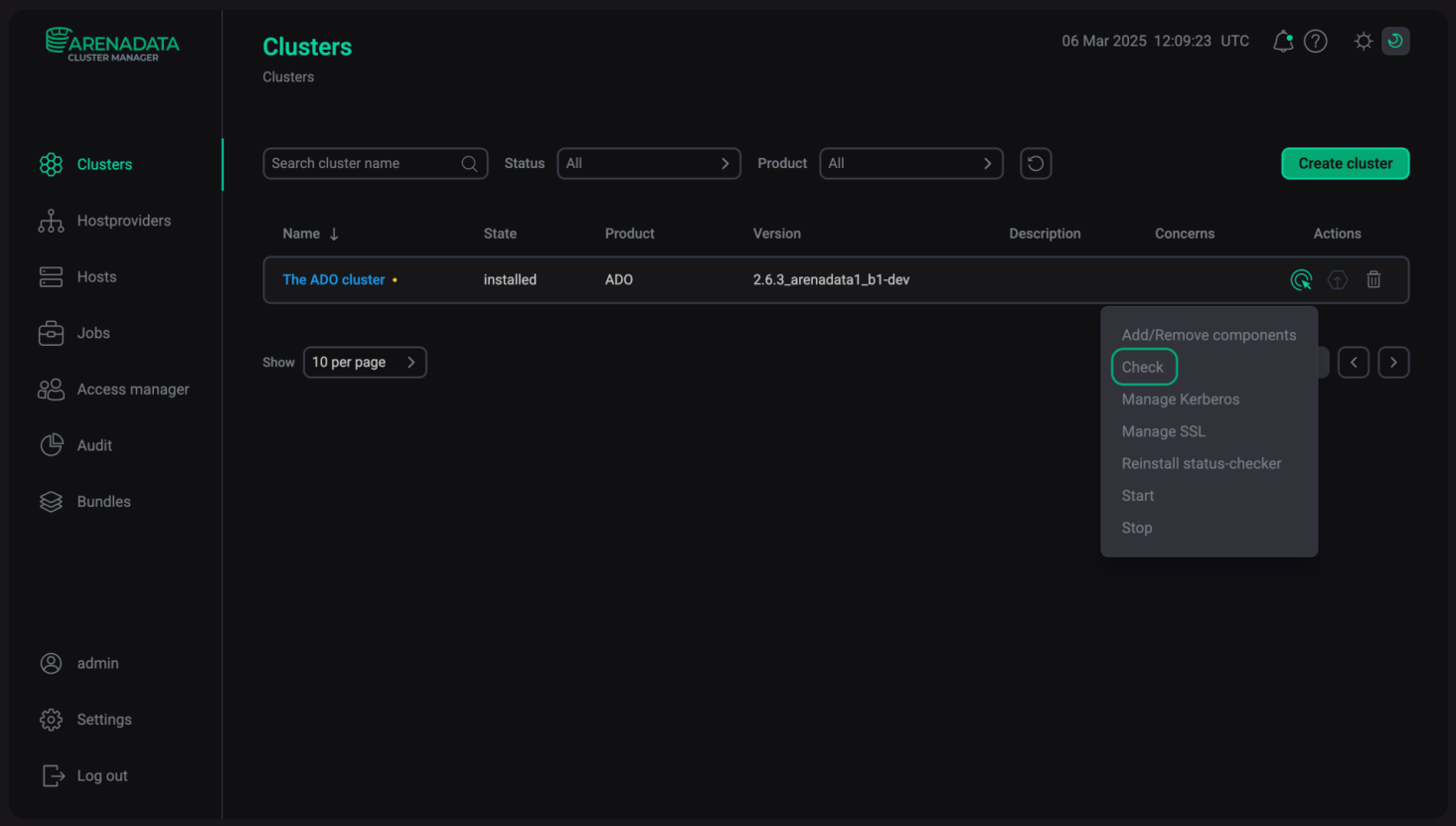
Check

Eugenia Kuzina, Elena Kostyuchenko

To Table of Contents

The ***Check*** action verifies whether ADO services work correctly. It sequentially runs service-specific tests for each service to check the health of the service and its components.

To run an action, go to the ***Clusters*** page, click in the ***Actions*** column, and select the action from the drop-down list that appears.



Execute the Check action

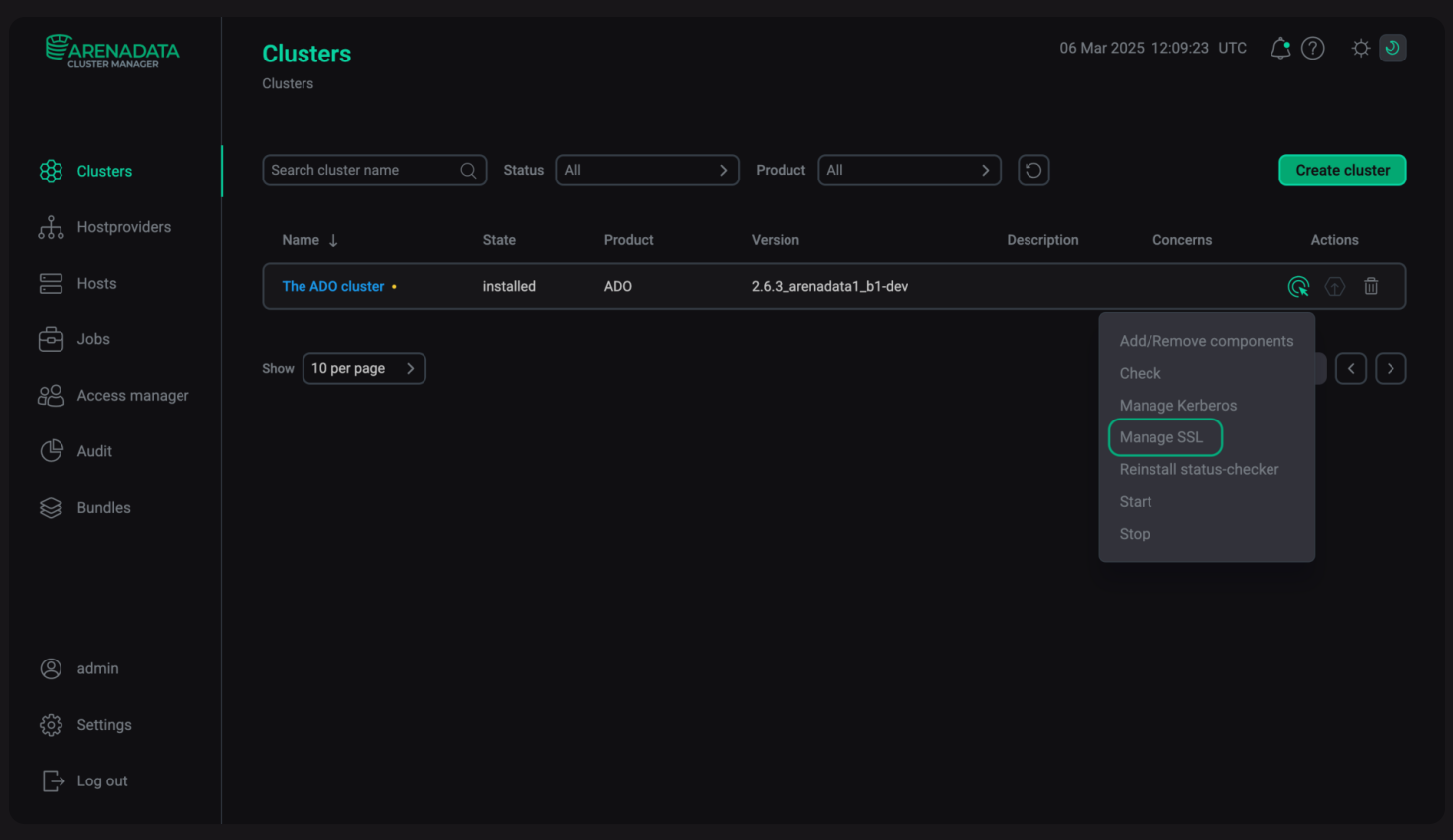
Manage SSL

Sergei Tikhomirov, Eugenia Kuzina, Elena Kostyuchenko

To Table of Contents

The ***Manage SSL*** action allows you to control the SSL encryption of all services in the ADO cluster. For additional information on SSL settings, see Requirements for SSL certificates.

To run an action, go to the ***Clusters*** page, click in the ***Actions*** column, and select the action from the drop-down list that appears.



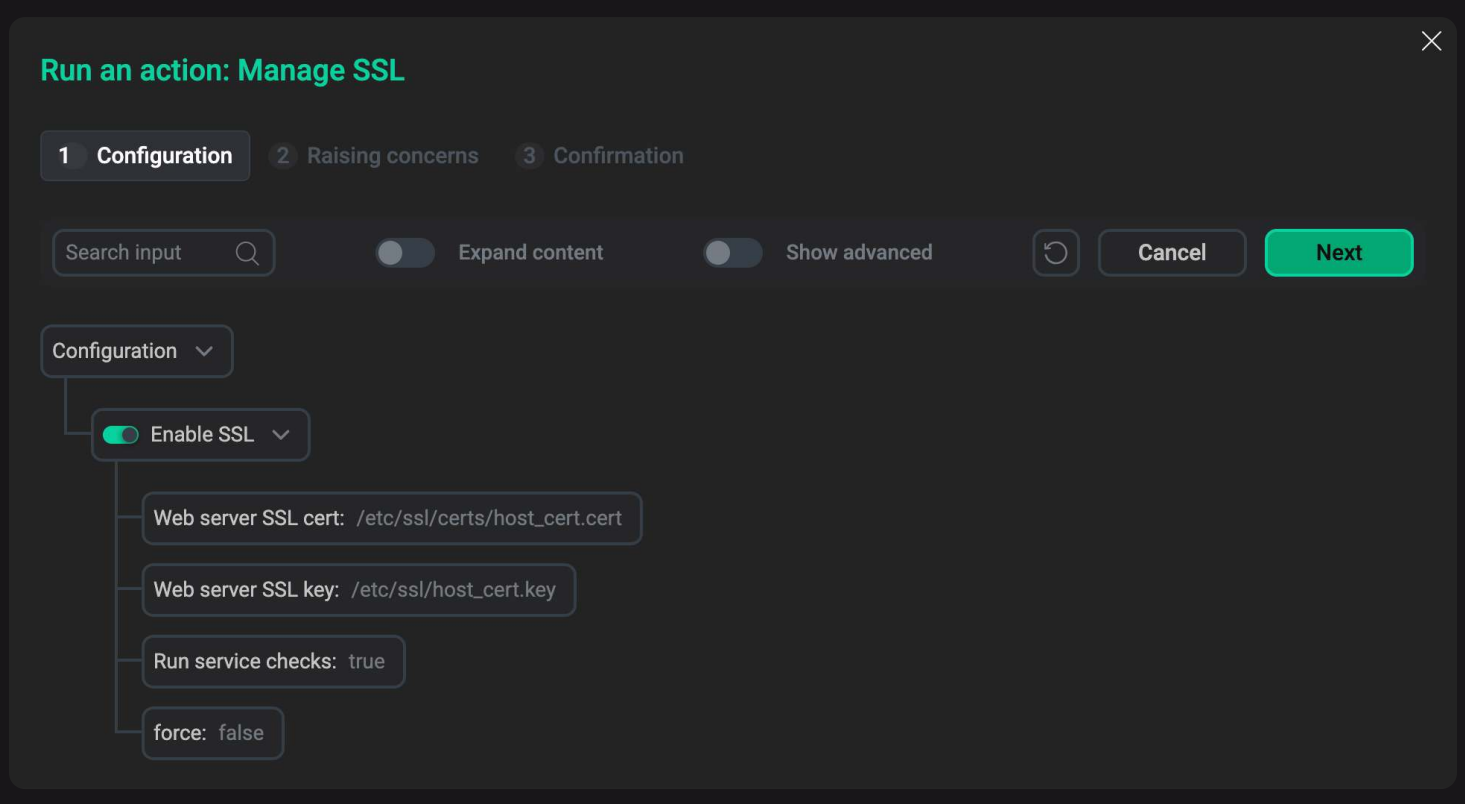
Execute the "Manage SSL" action

When you choose the action, ADCM displays a dialog window where you can specify SSL settings listed in the table below.

SSL settings

Parameter	Description	Default value
Web server SSL cert	The path to the web server SSL certificate	/etc/ssl/certs/host_cert.cert
Web server SSL key	The path to the web server SSL key	/etc/ssl/host_cert.key
Run service checks	Runs the service checks after the setting change	True
force	Defines whether current SSL parameters should be overwritten with the specified values	False

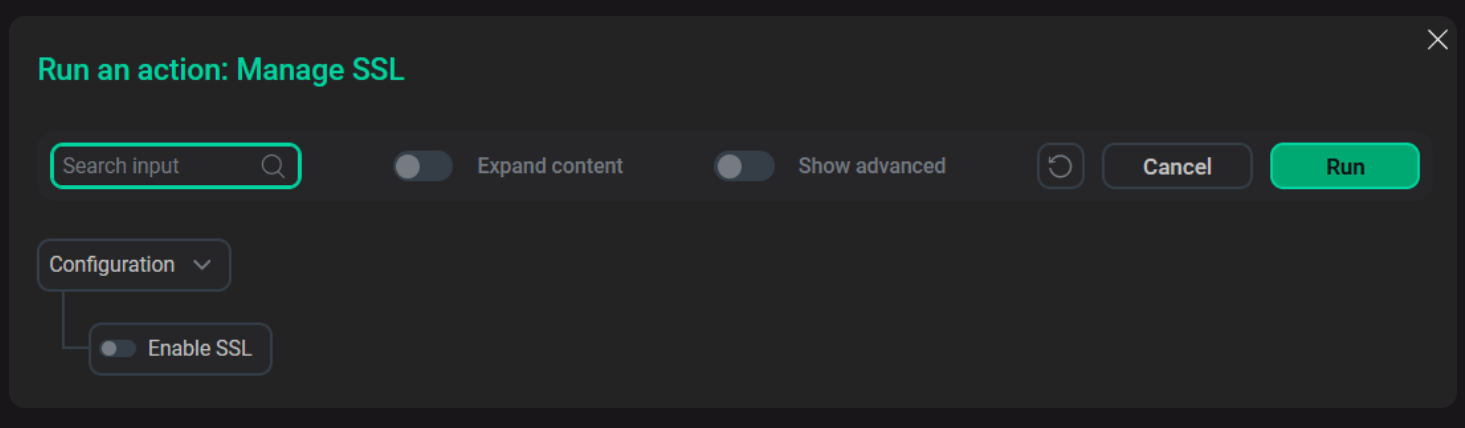
To enable SSL, run the action with the ***Enable SSL*** parameter group active and the parameters filled in.



Enable SSL with the "Manage SSL" action

After you specify the parameters, click ***Run***. In the action confirmation window that appears, you can select the ***Verbose*** checkbox to see additional execution details.

To disable SSL, run the ***Manage SSL*** action with the ***Enable SSL*** parameter group inactive.



Disable SSL with the "Manage SSL" action

During the execution of the ***Manage SSL*** action, ADCM stops all services one by one, reconfigures them to enable or disable SSL encryption, then starts the services and checks them.

Manage Kerberos

Sergei Tikhomirov, Elena Kostyuchenko

Contents

To Table of Contents

Manage Kerberos action

Custom kerberization settings

Custom keytab recommendations

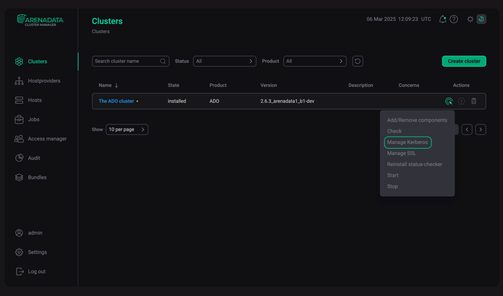
Manage Kerberos action

The ***Manage Kerberos*** action encapsulates enabling, reconfiguring, and disabling of Kerberos. To run it, you need an installed ADO cluster (see Get started with Arenadata Orchestrator) and one or more KDCs (Key Distribution Center). Before you proceed to managing Kerberos, it is recommended to read the Kerberos overview article about kerberization requirements.

Among other things, ***Manage Kerberos*** is designed to allow the following scenarios:

- Enabling Kerberos without administrator credentials using the existing client, principals and/or keytabs.
- Quickly changing the cluster state from kerberized to non-kerberized in case kerberization fails in the beginning (e.g. bad kadmin credentials) without having to reconfigure services, remove keytabs and principals.
- Postponed service reconfiguration — configure Kerberos at the moment and reconfigure service later.

To run the action in the ADCM web UI, go to the ***Clusters*** page. Select an installed and prepared ADO cluster, and choose the ***Manage Kerberos*** action.

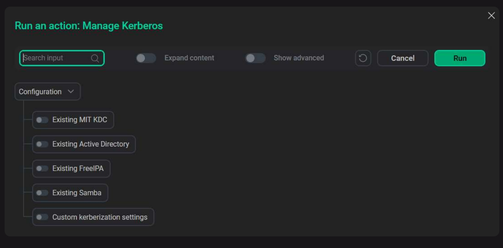


Manage Kerberos

The pop-up window suggests several options to run the action:

- Using an existing MIT KDC.
- Using an existing MS Active Directory.
- Using an existing FreeIPA.
- Using an existing Samba.
- Using the Custom kerberization settings option.

Each of these options can be combined with the ***Custom kerberization settings*** option.



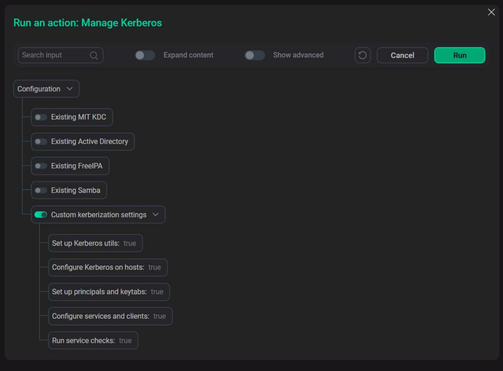
Ways to manage Kerberos

IMPORTANT

Running the action with one KDC type enabled will trigger Kerberos activation.

Custom kerberization settings

The ***Custom kerberization settings*** option allows the user to choose kerberization steps, for example, creation of principals and keytabs.



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	True
Configure services and clients	Enables updating of services and clients configuration	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
airflow	airflow	airflow	600

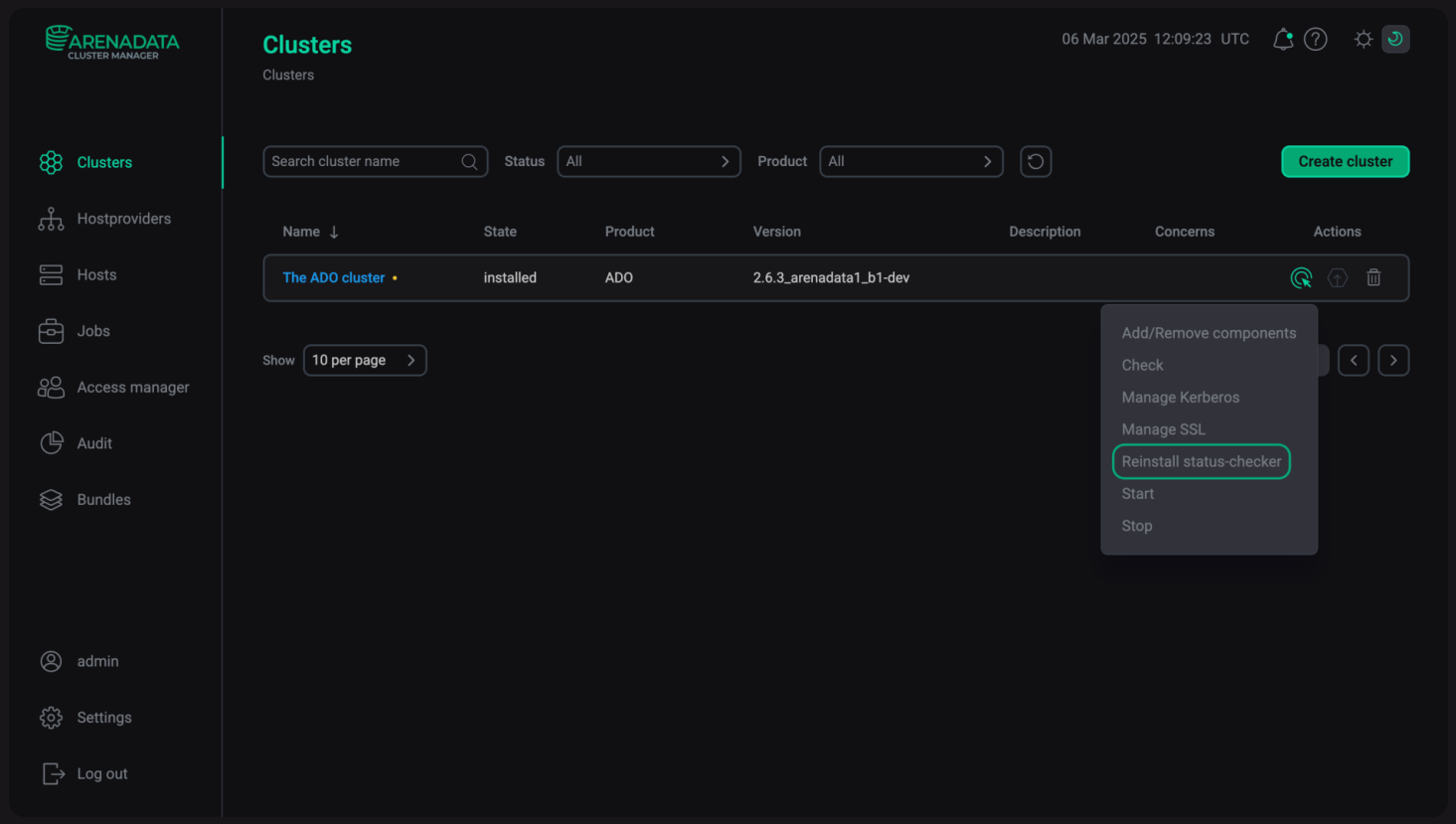
Reinstall status-checker

Eugenia Kuzina, Elena Kostyuchenko

To Table of Contents

The ***Reinstall status-checker*** action reinstalls status-checker for all services one by one.

To run an action, go to the ***Clusters*** page, click in the ***Actions*** column, and select the action from the drop-down list that appears.



Execute the "Reinstall status-checker" action

Start

Eugenia Kuzina, Elena Kostyuchenko

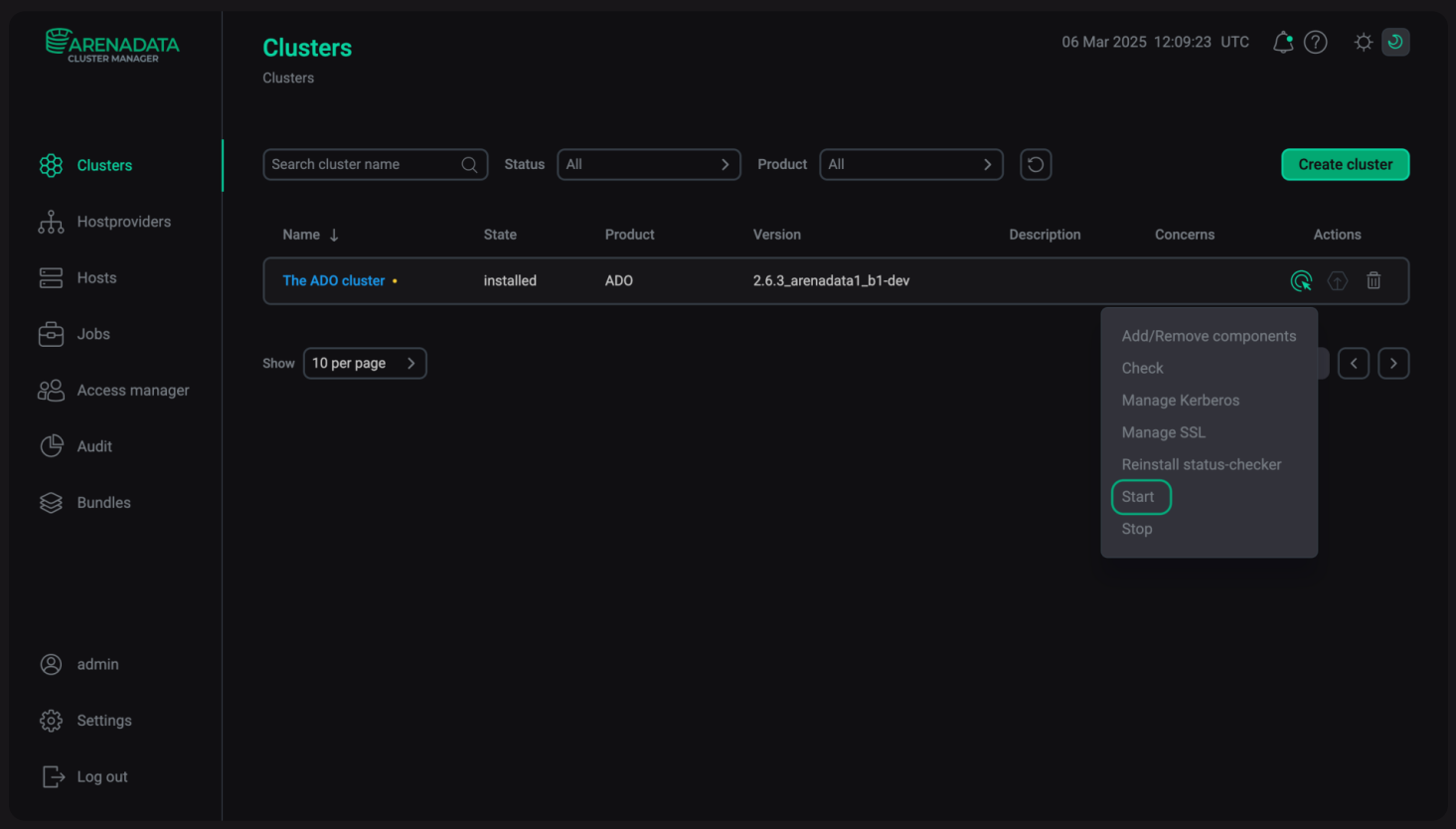
To Table of Contents

IMPORTANT

If you installed a monitoring cluster and want to use it with the ADO cluster, start the monitoring cluster first. To do this, run the ***Start*** action of the monitoring cluster in the same way as described for the ADO cluster.

The ***Start*** action starts all the services installed in the ADO cluster, one by one, in the correct order.

To run an action, go to the ***Clusters*** page, click in the ***Actions*** column, and select the action from the drop-down list that appears.



Execute the "Start" action

Alternatively, you can use the ***Start*** service action of a particular service to manually start only required services.

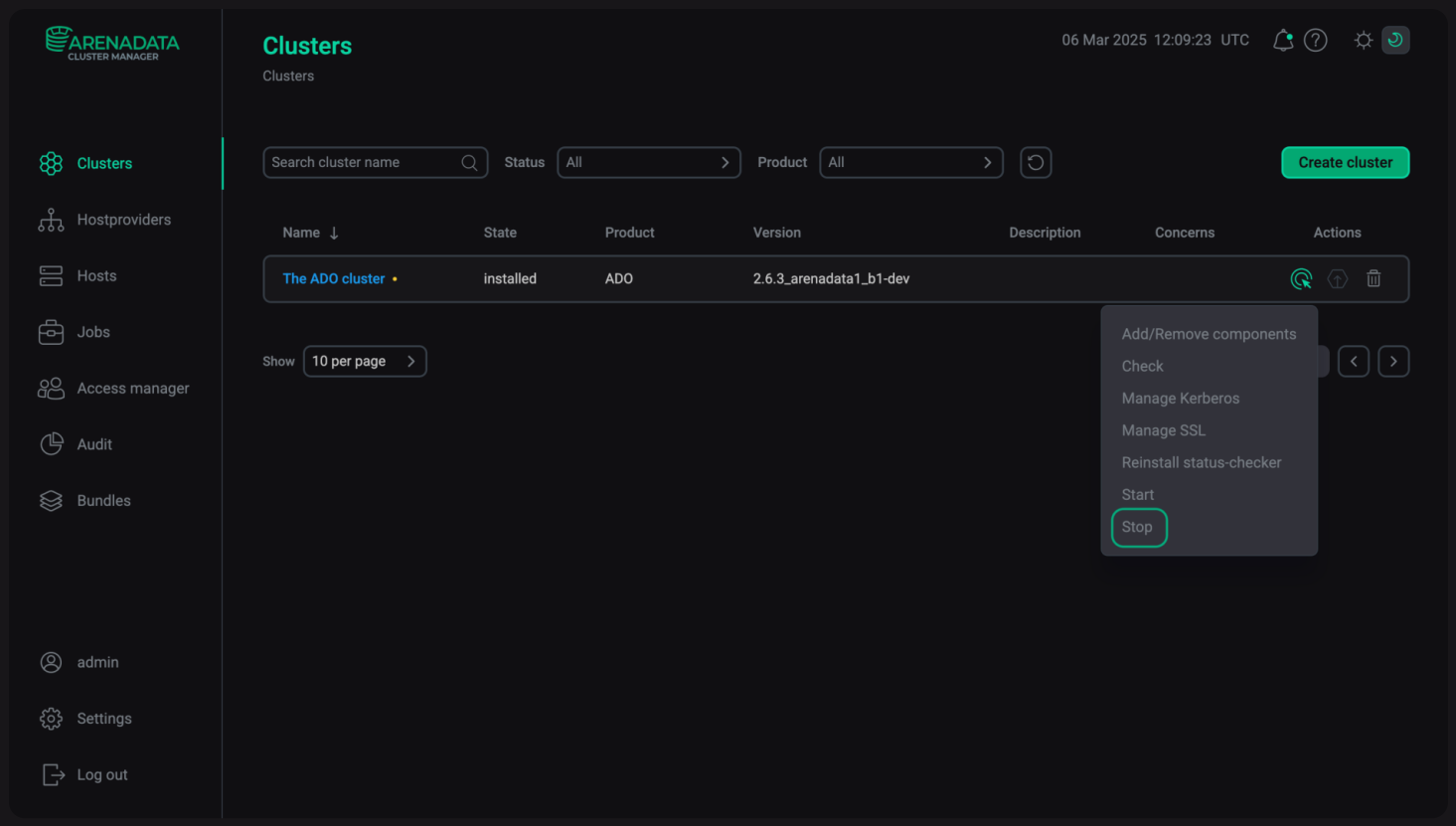
Stop

Eugenia Kuzina, Elena Kostyuchenko

To Table of Contents

The ***Stop*** action stops all the services installed in the ADO cluster in the correct order.

To run an action, go to the ***Clusters*** page, click in the ***Actions*** column, and select the action from the drop-down list that appears.



Execute the "Stop" action

Alternatively, you can use the ***Stop*** service action of a particular service to manually stop only required services or the entire cluster.

ADPG service management via ADCM

Elena Kostyuchenko

To Table of Contents

The ADCM UI provides actions to manage the Arenadata Postgres (ADPG) service. The actions are listed in the table below.

Action	Description
Add/Remove components	Running this action opens the component-host mapping interface where you can add, remove, and distribute ADPG components
Check ADPG	Runs service-specific tests to check the health of the service and its components
Restart ADPG	Restarts the service. When you run this action, the option ***Apply configs from ADCM*** is available. 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
Remove	Removes the service from the cluster. This action should be used to remove already installed services. Whereas the control can be used to remove a non-mapped service (a service which components have not been distributed among cluster hosts)
Start ADPG	Starts the service. When you run this action, the option ***Apply configs from ADCM*** is available. If it is set to `true`, all service configurations defined in ADCM settings will be applied on the service startup. Otherwise, the service starts without applying configurations from ADCM
Stop ADPG	Stops the service

Airflow service management via ADCM

Elena Kostyuchenko

To Table of Contents

The ADCM UI provides actions to manage the Airflow service and its components. The actions available for the Airflow service are listed in the table below.

For more information about Airflow components, see Airflow.

Action	Description
Add/Remove components	Running this action opens the component-host mapping interface where you can add, remove, and distribute Airflow components
Check	Runs service-specific tests to check the health of the service and its components
Start	Starts the service. When you run this action, the option ***Apply configs from ADCM*** is available. If it is set to `true`, all service configurations defined in ADCM settings will be applied on the service startup. Otherwise, the service starts without applying configurations from ADCM
Stop	Stops the service
Restart	Restarts the service. When you run this action, the option ***Apply configs from ADCM*** is available. 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
Remove	Removes the service from the cluster. This action should be used to remove already installed services. Whereas the control can be used to remove a non-mapped service (a service which components have not been distributed among cluster hosts)
Rotate fernet key	Rotates the Fernet encryption key. If ***Autogenerate new fernet key*** is set to `True`, ADCM will generate a new Fernet key automatically. The result depends on the ***Manage sensitive configuration data*** parameter, see Fernet key rotation. This action will require several restarts of the Airflow service
Sync permission	Updates permissions for existing roles
Sync requirements	Updates Airflow dependencies on all hosts. ***Restart*** is required to apply the changes

Some of the above actions are also available for Airflow on a per-component basis. Available component actions are listed in the table below.

Component	Start	Stop	Restart	Check
Airflow DAG Processor	+	+	+	—
Airflow Flower	+	+	+	+
Airflow Scheduler	+	+	+	—
Airflow Server	+	+	+	+
Airflow Triggerer	+	+	+	—
Airflow Worker	+	+	+	—

Monitoring service management via ADCM

Elena Kostyuchenko

To Table of Contents

The ADCM UI provides actions to manage the Monitoring service and its components. The actions are listed in the table below.

Action	Description
Add/Remove components	Running this action opens the component-host mapping interface where you can add, remove, and distribute Monitoring components
Check	Runs service-specific tests to check the health of the service and its components
Reconfigure and restart	When the action is launched, all service configurations defined in ADCM settings will be applied during the service restart
Reinstall	Reinstalls the service with the default configuration
Start	Starts the service
Stop	Stops the service

For each particular Statsd-exporter component, you can run the following actions on a per-component basis: ***Restart Statsd-exporter***, ***Start Statsd-exporter***, and ***Stop Statsd-exporter***.

Redis service management via ADCM

Elena Kostyuchenko

To Table of Contents

The ADCM UI provides actions to manage the Redis service and its components. The actions are listed in the table below.

Action	Description
Add/Remove components	Running this action opens the component-host mapping interface where you can add, remove, and distribute Redis components
Check Redis	Runs service-specific tests to check the health of the service and its components
Restart Redis	Restarts the service. When you run this action, the option ***Apply configs from ADCM*** is available. 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
Remove	Removes the service from the cluster. This action should be used to remove already installed services. Whereas the control can be used to remove a non-mapped service (a service which components have not been distributed among cluster hosts)
Start Redis	Starts the service. When you run this action, the option ***Apply configs from ADCM*** is available. If it is set to `true`, all service configurations defined in ADCM settings will be applied on the service startup. Otherwise, the service starts without applying configurations from ADCM
Stop Redis	Stops the service

For each particular Redis server, you can run the following actions on a per-component basis: ***Check***, ***Restart***, ***Start***, and ***Stop***.

Integration with HashiCorp Vault to store secrets

Sergei Tikhomirov

Contents

To Table of Contents

Overview

Prerequisites

Configuration in ADCM

Fernet key rotation

Overview

In Airflow, secret backed is a mechanism of securely storing and retrieving sensitive information such as passwords, API keys, etc. Airflow can be configured to use various types of secret backends (environment variables, metastore database, local filesystem, etc.), but it's strongly recommended to use HashiCorp Vault due to enhanced security.

Prerequisites

Before configuring the secret backend in ADCM, have a Vault instance configured, running, and available (see Vault installation).

Follow these steps to prepare for the ADO integration:

1. Create a mount point for Airflow:

```
$ vault secret enable -path=airflow -version=2 kv
```

2. Generate credentials for authentication (here, an authentication token is used):

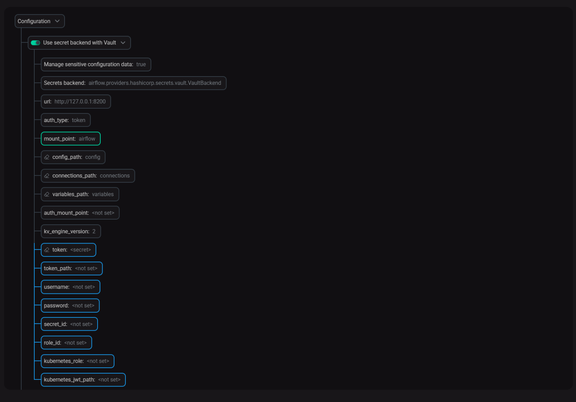
```
$ vault token create -period=30m
```

Save the generated token or copy it as you will need it later on.

Configuration in ADCM

To configure ADO integration with HashiCorp Vault, follow the steps below:

1. In ADCM, select your ADO cluster and click ***Services*** → ***Airflow***.
2. Activate the ***Show advanced*** switch.
3. Activate the ***Use secret backend with Vault*** switch and fill the parameters inside (see Airflow configuration parameters):



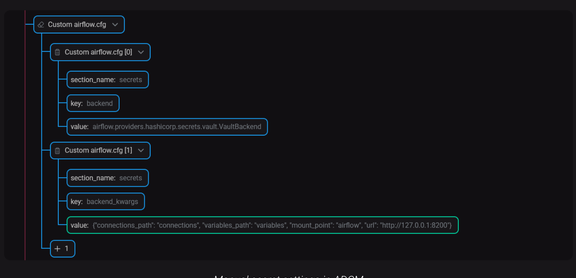
Vault settings in ADCM

NOTE

When the ***Manage sensitive configuration data*** parameter is enabled, the product takes over the creation of secrets (transferring them from configurations to Vault) as well as updating them. As a limitation, this requires the appropriate rights to create secrets in Vault.

4. Save the configuration and restart Airflow.

To manually generate the ***airflow.cfg[secrets]*** block, you have to add the necessary data in the ***Custom airflow.cfg*** parameter.



Manual secret settings in ADCM

Custom parameter values overwrite the values specified in the ***Use secret backend with Vault*** section.

Fernet key rotation

With the HashiCorp Vault integration enabled, the behavior of the ***Rotate fernet key*** action differs.

	Autogenerate new fernet key enabled	***Autogenerate new fernet key*** disabled
Manage sensitive configuration data enabled	The fernet key will get changed in Vault and no value will be kept on a host	With an empty value provided as a custom fernet key, the action succeeds, but no changes are made to Vault and no value is kept on a host
Manage sensitive configuration data disabled	The fernet key will get changed a host and no updates will happen in Vault	With an empty value provided as a custom fernet key, the action fails as it cannot rotate a key with an empty value

In all the other cases (e.g. with a non-empty valid/invalid custom fernet key), the action behaves the same way regardless of the parameters mentioned above.

Logging in Airflow

Daria Barysheva, Elena Kostyuchenko

Contents

To Table of Contents

Overview

Grep logs

Logging levels

Logging configuration

Overview

Airflow writes text logs used for analyzing errors that can occur while running DAGs. These logs are located in the */var/log/airflow/* directory of the Airflow server's host, but they are also available in the Airflow UIs.

Log paths

The full path to each DAG log file looks like this:

/var/log/airflow/dag_id=<DAG_ID>/run_id=<DAG_Run_ID>/task_id=<Task_ID>/<log_number>.log

where:

- `<DAG_ID>` is the DAG identifier.
- `<DAG_Run_ID>` is the identifier of the DAG run, which combines in itself the type of run and a timestamp. For example, *run_id=scheduled__2024-07-14T14:18:33.254657+00:00*.
- `<Task_ID>` is the task identifier.
- `<log_number>` is the sequence number of the log (begins with 1).

The full path to the process manager log files is */var/log/airflow/dag_processor_manager/*.

The full path to the scheduler logs is */var/log/airflow/scheduler/*.

To view Airflow logs on the host:

1. Connect to the Airflow server via SSH and run the following command:

```
$ ls -la /var/log/airflow/
```

   The output looks similar to this:

```
total 12
drwxr-xr-x.  5 airflow airflow   77 Jul 15 14:18 .
drwxr-xr-x. 16 root    root    4096 Aug  5 08:44 ..
drwxr-xr-x. 13 airflow airflow 4096 Jul 30 10:12 dag_id=adcm_check
drwxr-xr-x.  2 airflow airflow  109 Jul 29 23:54 dag_processor_manager
drwxr-xr-x. 20 airflow airflow 4096 Aug  5 07:56 scheduler
```

2. View the desired log file:

```
$ cat /var/log/airflow/dag_id=adcm_check/run_id=manual__2024-07-15T14:18:43.743847+00:00/task_id=runme_1/attempt=1.log
```

   The output is similar to this:

```
[2024-07-15T14:18:46.143+0000] {taskinstance.py:1103} INFO - Dependencies all met for dep_context=requeueable deps ti=
<TaskInstance: adcm_check.runme_1 manual__2024-07-15T14:18:43.743847+00:00 [queued]>
[2024-07-15T14:18:46.144+0000] {taskinstance.py:1308} INFO - Starting attempt 1 of 4
[2024-07-15T14:18:46.157+0000] {taskinstance.py:1327} INFO - Executing <Task(BashOperator): runme_1> on 2024-07-15 14:18:43.
743847+00:00
[2024-07-15T14:18:46.164+0000] {standard_task_runner.py:57} INFO - Started process 7082 to run task
[2024-07-15T14:18:46.168+0000] {standard_task_runner.py:84} INFO - Running: ['airflow', 'tasks', 'run', 'adcm_check', 'runme_1',
'manual__2024-07-15T14:18:43.743847+00:00', '--job-id', '12', '--raw', '--subdir', 'DAGS_FOLDER/adcm_check.py', '--cfg-path',
'/tmp/tmpr_3qshf4']
[2024-07-15T14:18:46.171+0000] {standard_task_runner.py:85} INFO - Job 12: Subtask runme_1
[2024-07-15T14:18:46.233+0000] {task_command.py:410} INFO - Running <TaskInstance: adcm_check.runme_1 manual__2024-07-15T14:18:43.
743847+00:00 [running]> on host elenas-ado3.ru-central1.internal
[2024-07-15T14:18:46.326+0000] {taskinstance.py:1545} INFO - Exporting env vars: AIRFLOW_CTX_DAG_OWNER='airflow'
AIRFLOW_CTX_DAG_ID='adcm_check' AIRFLOW_CTX_TASK_ID='runme_1' AIRFLOW_CTX_EXECUTION_DATE='2024-07-15T14:18:43.743847+00:00'
AIRFLOW_CTX_TRY_NUMBER='1' AIRFLOW_CTX_DAG_RUN_ID='manual__2024-07-15T14:18:43.743847+00:00'
```

Grep logs

You can search through the logs for specific information, like error messages. To do this, connect to the host with the logs you want to inspect and use a `grep` command.

For example:

```
$ cat /var/log/airflow/dag_processor_manager/dag_processor_manager.log | grep -i -A3 -B1 error | grep -i -A3 -B1 error
```

This command searches for messages containing the word `error` in the process manager log. The `-i` option allows you to ignore case distinctions. The `-A3 -B1` options expand the output to one line before and three lines after the line containing the error.

Example output:

```
File Path                                                
```

Manage dependencies via ADCM

Elena Kostyuchenko

Contents

To Table of Contents

Overview

Define dependencies in ADCM

- Add a Python package
- Add a package from a private repository

Dependency constraints

Overview

Some Airflow DAGs require additional Python packages that are not included in the base environment. These packages may come from the public Python Package Index (PyPI), private repositories, or local distributions.

ADCM provides a centralized way to manage these dependencies. You can define and update Python package requirements directly from the ADCM UI using configuration parameters available for the Airflow service. ADCM ensures that these configurations are applied to all hosts in the cluster.

This feature supports:

- dependencies from public packages from PyPI (e.g. `requests`, `pandas`);
- dependencies from private or internal Python packages hosted on authenticated repositories;
- accessing repositories through a proxy;
- dependency version constraints.

The Airflow configuration in ADCM provides editable fields that map directly to the parameters used by `pip install`.

This guide describes how to configure dependency management for Airflow using ADCM.

Define dependencies in ADCM

Let's consider a DAG that has the following dependencies:

```
from airflow import DAG
from airflow.operators.python import PythonOperator
from datetime import datetime
import requests (1)
import pandas as pd (2)
import my_private_module (3)
```

(1) Python package.

(2) Python package of a specific version.

(3) Package from a private repository.

Add a Python package

ADCM provides the ***Extra requirements*** field in Airflow configuration that functions as a *requirements.txt* file for managing dependencies.

To install the required Python packages for DAGs via ADCM:

1. On the ***Clusters*** page, select your ADO cluster.
2. Go to the ***Services*** tab and select the ***Airflow*** service.
3. Toggle the ***Show advanced*** option and unfold the ***Dependency Management*** section.
4. In the ***Extra requirements*** field, list the packages in the following format: `<package-name>==<version>`. For example:

```
requests
pandas==2.2.2
```

   If a version is omitted, the latest is installed by default. For more information on how to specify packages, see the Requirements File Format description.

5. Confirm changes to configuration by clicking ***Save***.
6. In the ***Actions*** drop-down menu, select ***Sync requirements*** and click ***Run***.
7. In the ***Actions*** drop-down menu, select ***Restart***, make sure the ***Apply configs from ADCM*** option is set to `true`, and click ***Run***.

Add a package from a private repository

When your DAGs rely on a package hosted in a private repository, you can configure the package source host and authentication credentials in ADCM.

To install the required Python packages from a private repository via ADCM:

1. On the ***Clusters*** page, select your ADO cluster.
2. Go to the ***Services*** tab and select the ***Airflow*** service.
3. Toggle the ***Show advanced*** option and unfold the ***Dependency Management*** section.
4. In the ***Extra requirements*** field, list the packages in the following format: `<package-name>==<version>`. For example:

```
my-private-pkg==0.1.0
```

5. Configure the following fields:
   - ***index-url*** — repository URL;
   - ***index-url-user*** — username for authentication;
   - ***index-url-password*** — password for authentication.
6. Confirm changes to configuration by clicking ***Save***.
7. In the ***Actions*** drop-down menu, select ***Sync requirements*** and click ***Run***.
8. In the ***Actions*** drop-down menu, select ***Restart***, make sure the ***Apply configs from ADCM*** option is set to `true`, and click ***Run***.

Optionally, if your cluster requires a proxy access to fetch public packages, you can enter the proxy parameters in the following fields:

- ***proxy*** — proxy server address;
- ***proxy-user*** — username for authentication via proxy;
- ***proxy-password*** — password for authentication via proxy.

If the proxy does not support HTTPS, enter the IP address of the host or the `<host IP>:<port>` pair in the ***trusted-host*** field.

Dependency constraints

ADCM provides a mechanism to control the versions of critical Python packages using constraints files. This helps to prevent conflicts when adding new dependencies or unintended upgrades of essential components such as Apache Airflow.

Constraints files differ from requirements files in that they define which versions are allowed, but do not install them automatically.

The ***Dependency Management*** Airflow configuration group includes the following fields for version control:

- ***Constraints file*** — editable list of version constraints defined by the user;
- ***Base constraints file*** — read-only list of constraints required for proper Airflow operation, which cannot be overridden by ***Constraints file***.

To define version constraints via ADCM:

1. On the ***Clusters*** page, select your ADO cluster.
2. Go to the ***Services*** tab and select the ***Airflow*** service.
3. Toggle the ***Show advanced*** option and unfold the ***Dependency Management*** section.
4. In the ***Constraints file*** field, list the packages in the following format: `<package-name>==<version>`. For example:

```
apache-airflow==2.10.5
pandas==2.2.2
requests<3.0.0
```

5. Confirm changes to the configuration by clicking ***Save***.
6. In the ***Actions*** drop-down menu, select ***Sync requirements*** and click ***Run***.
7. In the ***Actions*** drop-down menu, select ***Restart***, make sure the ***Apply configs from ADCM*** option is set to `true`, and click ***Run***.

ADCM validates all dependencies against the combined constraints file during the ***Sync requirements*** action.

If a package installation violates defined constraints, the synchronization process fails and an error is logged.

Examples of constraint violations include:

- attempting to install a package that requires a higher version of Apache Airflow than allowed;
- specifying a package version that conflicts with the base constraints;
- manually installing incompatible packages in the Airflow virtual environment.

In all such cases, dependency synchronization will be blocked to maintain cluster stability.

Airflow performance tuning

J. Elena Kostyuchenko

Contents

To Table of Contents

- Scheduler tuning
  - Key settings
  - Scheduler parallelism, HA, and worker coordination
- DAG design recommendations
  - Parsing and top-level code
  - Dynamic task mapping
  - XComs and inter-task data transfer
- Executor and worker tuning
- Metadata database (PostgreSQL) tuning
  - Connection pooling & SQLAlchemy
- Concurrency controls, pools, and priorities
  - Pools
  - Priorities and SLA
- Web Server sizing
- Logging
- Evaluate tuning results

This article describes optimization techniques and best practices that can help you improve the performance of the Airflow service in Annadata Orchestrator clusters. The described approaches are based on the latest Apache Airflow recommendations and are applicable to the 2.10.5 version of Airflow or higher.

Scheduler tuning

The scheduler is the primary throughput limiter for Airflow at scale. When the scheduler can't keep up with DAG parsing and task scheduling, queues grow, task wait time increases and leads to the metadata DB overload. Tuning the scheduler first yields the largest benefits in terms of throughput optimization.

If scheduler loop time increases, you can find bottlenecks by checking DAG parse time, top-level imports in DAGs, and values in the ***store_serialized_dags*** and ***min_file_process_interval*** parameters.

If tasks stay in a queue for a long time, check the state of the Executor, Celery broker queue length, the values in the ***worker_concurrency*** parameter, and DB connection pool exhaustion.

Key settings

Configuration property	Description	Suggested starting value
store_serialized_dags	Stores serialized DAGs in the metadata DB so the scheduler reads pre-parsed DAGs instead of re-parsing Python on every loop. Reduces parse CPU and I/O by letting the scheduler load a compact serialized DAG model from the DB instead of running Python. This is very effective for large numbers of DAG files	True
min_file_process_interval	Minimum seconds between DAG file parsing cycles. Increases parse efficiency at expense of immediate DAG update propagation	30s
dag_dir_list_interval	Interval for scanning the dags folder for new or removed files. Set a higher value to reduce I/O overhead	60s
max_tis_per_query	Batch size of TaskInstance rows fetched per scheduling loop. Lower values make the scheduler more responsive. Experiment to find a value that balances between the DB query cost and scheduler throughput. Recent Airflow releases changed defaults to favor responsiveness. Must be lower than ***core.parallelism***	16–128
max_dagruns_to_create_per_loop	Limit DAGs for which runs are created per loop (helps distribute work across multiple schedulers)	10–50

Scheduler parallelism, HA, and worker coordination

For better performance, run schedulers on CPU-optimized nodes. For high scale, run multiple schedulers in HA mode with ***use_row_level_locking=True*** and tune ***max_dagruns_to_create_per_loop*** so creation of DAG runs is spread across schedulers.

Increase scheduler CPU and memory if you parse many DAGs or use dynamic DAG generation, but first refactor DAGs to minimize top-level work.

DAG design recommendations

Flawed DAG design can multiply scheduler and worker cost. To make sure that DAGs do not impede your Airflow optimization efforts, consider best practices for writing DAGs described in this section.

Parsing and top-level code

When writing DAG files in Airflow, it is important to avoid heavy imports or blocking calls at the module's top level (outside of task functions or operators). This is because the scheduler and Web Server need to continuously parse and load DAG files to keep track of available workflows.

Every time a DAG is parsed, all the top-level code in the file is executed. If the file contains expensive imports, database queries, or network calls at the top level, this significantly increases the parsing time and can put unnecessary strain on the scheduler and Web Server. Under high load with many DAGs, these delays compound, leading to slower scheduling loops, missed SLAs, and reduced cluster throughput.

Another risk is that blocking calls or large dependencies at the module level can introduce instability. For example, if a DAG file imports a library that takes several seconds to load or attempts to fetch data from an external service, the scheduler may hang or even fail to parse the DAG, marking it as broken. This can prevent critical workflows from being scheduled at all.

To avoid these issues, heavy operations should be moved inside task functions or operators, where they will only execute when the task runs. Keeping DAG files lightweight ensures that parsing remains fast, predictable, and resilient even in environments with thousands of workflows and frequent code updates.

An example of a problematic DAG:

```
import pandas as pd  1
heavy = heavy_initialization()  2

with DAG(
    dag_id="bad_dag",
    start_date=datetime(2025, 1, 1),
    schedule_interval=None,
    catchup=False,
):

    @task()
    def do_work_bad():
        heavy.process()

    do_work_bad()
```

1. Placing an import here will slow down the parsing.
2. This top-level function will be executed during parsing, which will also slow it down.

An example of how to fix that DAG:

```
with DAG(
    dag_id="good_dag",
    start_date=datetime(2025, 1, 1),
    schedule_interval=None,
    catchup=False,
):

    @task()
    def do_work_good():
        from mylib import heavy_initialization  1
        heavy = heavy_initialization()  2
        heavy.process()

    do_work_good()
```

1. Put the import inside the task.
2. Put the function call inside the task.

Removing top-level work in this way reduces parse latency and scheduler CPU spikes.

Dynamic task mapping

Dynamic task mapping enables optimizing Airflow performance in environments with many parameterized or repetitive tasks through a DAG design. It enables creating tasks at runtime rather than generating task instances at parse time.

When generating DAGs this way, you define a single mapped task and supply it with a list of inputs instead of creating all task objects manually. When the DAG runs, Airflow automatically expands that task into multiple parallel instances, then it creates the individual task instances in the metadata database. This reduces scheduler load, speeds up DAG parsing, and keeps the DAG lightweight.

XComs and inter-task data transfer

It's not recommended to push large payloads to XCom (metadata DB). For large artifacts it's better to use object storage (for example, S3 or HDFS) and push only pointers, like paths and URIs, via XCom.

If you need structured small messages, use XCom JSON serialization or a remote XCom backend designed for larger payloads.

Executor and worker tuning

The Executor determines how Airflow runs tasks in parallel. In ADO, the recommended Executors are `CeleryExecutor` and `KubernetesExecutor`, depending on workload characteristics.

Key parameters for tuning Executors in ADO are listed below.

Parameter	Description	Default value
parallelism	Global limit for the number of task instances that can run in parallel across the entire Airflow environment	32
dag_concurrency	Maximum number of task instances allowed to run concurrently per DAG. Prevents a single DAG from exhausting cluster resources	16
max_active_runs_per_dag	Maximum number of active DAG runs for each DAG. Setting too high values may lead to scheduler and worker overload	16
worker_concurrency	Maximum number of task instances a single worker can execute simultaneously. Adjust based on available CPU cores and memory	16

Worker concurrency and resource sizing parameters directly determine how efficiently tasks are executed across your cluster.

The configuration of workers defines the throughput, responsiveness, and stability of the entire system. Worker concurrency controls how many tasks a single worker process can execute at the same time. If concurrency is set too low, your cluster might underutilize available CPU and memory resources, leading to slow DAG runs and idle infrastructure. If it's set too high, the worker may attempt to run more tasks than the host machine can handle, causing CPU thrashing, memory exhaustion, and task failures.

For example, a worker with 8 CPU cores and a concurrency setting (***worker_concurrency***) of 64 might try to run far too many Python processes in parallel, leading to context switching overhead and degraded performance. But setting concurrency to 2 on the same machine would leave resources underused, forcing tasks to queue unnecessarily.

For optimal performance, set concurrency to a level that matches the available hardware resources (cores, memory) and the typical resource profile of your tasks. For lightweight tasks (like database queries or API calls), higher concurrency may work well, but for heavy ETL or ML tasks, lower concurrency ensures stability and better throughput.

Airflow workers can be scaled both vertically (increasing CPU, memory, or disk resources per worker) and horizontally (adding more worker nodes). It's recommended to align the capacity of your hardware with the type of the workloads:

- CPU-intensive tasks (e.g. data transformations, Spark job submission) benefit from more cores but require careful concurrency tuning so that CPU-bound tasks don't overwhelm the infrastructure.
- Memory-heavy tasks (e.g. large Pandas dataframes) require larger memory allocations and lower concurrency settings to prevent out-of-memory errors.
- Mixed workloads often benefit from splitting workers into pools or queues, so different task types run on workers optimized for their needs.

Metadata database (PostgreSQL) tuning

The metadata database is often the first and most critical bottleneck when scaling Airflow. Every scheduler heartbeat, task state update, log reference, and DAG parse results in reads and writes to this database. Because Airflow's workload consists of a very high volume of small, frequent inserts, updates, and queries, the database must be tuned specifically for high write and read concurrency.

Recommendations for DB optimization:

- Set the value for ***max_connections*** high enough to handle Airflow components (scheduler, Web Server, workers, CLI), but balance it with connection pooling to avoid exhausting memory.
- Increase ***wal_buffers*** and set the right value for ***checkpoint_completion_target*** so that write-ahead log (WAL) flushing doesn't block frequent updates.
- Ensure the storage is optimized for fast random writes (e.g. SSDs).
- Airflow's metadata tables, such as `task_instance`, `dag_run`, and `log`, grow quickly. If autovacuum isn't productive enough, table bloat might cause slow queries. Increase frequency of autovacuum on these large tables, or schedule manual `VACUUM ANALYZE` for predictable performance.
- Partitioning or periodically archiving old task history and logs keeps table sizes manageable.

It is also important to scale the database vertically (CPU, memory, IOPS) as workflows grow. For more information on optimizing Postgres, see the Performance tuning article.

Connection pooling & SQLAlchemy

Set SQLAlchemy pool values so connections are available for scheduler, Web Server, workers, and external tooling.

Parameter	Description	Suggested starting value
sql_alchemy_pool_size	The SQLAlchemy pool size is the maximum number of database connections in the pool. `0` indicates no limit	20
sql_alchemy_max_overflow	The maximum overflow size of the pool. When the number of checked-out connections reaches the size set in `pool_size`, additional connections will be returned up to this limit. When those additional connections are returned to the pool, they are disconnected and discarded. The total number of simultaneous connections the pool will allow is `pool_size` + `max_overflow`, and the total number of sleeping connections the pool will allow is `pool_size`. `max_overflow` can be set to `-1` to indicate no overflow limit; no limit will be placed on the total number of concurrent connections. Defaults to `10`	30
sql_alchemy_pool_recycle	The SQLAlchemy pool recycle is the number of seconds a connection can be idle in the pool before it is invalidated. If the number of DB connections is ever exceeded, a lower config value will allow the system to recover faster	1800

Increasing ***sql_alchemy_pool_size*** reduces wait time for DB connections. Ensure your Postgres ***max_connections*** is sized to accommodate total Airflow connections plus other DB users.

Concurrency controls, pools, and priorities

Pools

By default, Airflow will schedule as many tasks as allowed by the Executor's capacity and worker concurrency. While this maximizes throughput, it can easily lead to resource contention. For example, too many tasks hammering a database, API rate limits being exceeded, or I/O heavy jobs monopolizing a worker. Pools act as a concurrency throttle: they group tasks into resource categories and limit how many of them can run at once across the entire Airflow cluster.

One of the better ways to utilize this feature is to create pools for resource-bounded operations (e.g. `hadoop_jobs`, `db_ingest`, `api_calls`), set explicit slot counts, and assign tasks to pools via pool argument in operators.

You can configure pools using Airflow CLI or UI.

Priorities and SLA

Use ***priority_weight*** on tasks to influence slot allocation when pools are saturated. Combine with pools to ensure high-priority jobs get scheduled sooner.

Using DAG-level caps such as ***max_active_runs_per_dag*** together with pools helps to smooth out bursts of task executions and prevents resource exhaustion. By default, if a DAG is triggered frequently or has a large backlog of catchup runs, Airflow may attempt to schedule many DAG runs at once. Setting ***max_active_runs_per_dag*** limits how many runs of that DAG can execute in parallel, ensuring that execution is paced and system resources are not overwhelmed. When combined with pools, DAG-level settings control how a single workflow consumes cluster capacity, while pools coordinate concurrency across different workflows and resource types.

You can also implement a short task-rate limiter via an upstream DAG or queue consumer that writes to an intermediate queue (e.g. Kafka) for some cases.

Web Server sizing

The Airflow Web Server serves the UI, API requests, and DAG/task metadata queries, all of which increase linearly with the number of users and DAG activity. Under-provisioned webservers can lead to slow page loads, timeouts, or even service unavailability during peak usage.

To optimize performance, it is important to scale the Web Server both vertically (allocating sufficient CPU, memory, and network bandwidth) and horizontally (running multiple replicas behind a load balancer). Additionally, you can enable `cluster` worker tuning to be able to adjust the number of worker processes and threads to match available cores. This ensures that requests are served efficiently without overloading the CPU or exhausting memory.

For very large deployments, caching static content, limiting the number of DAGs loaded per user session, and offloading heavy API queries to dedicated endpoints can further reduce load, ensuring that the Airflow UI remains responsive even under high user concurrency.

Logging

Avoid local-only logs if you scale workers: configure remote logging (S3/GCS/Elasticsearch) to centralize logs and reduce disk I/O on workers. Configure ***remote_base_log_folder*** and ***remote_log_conn_id*** parameters via ADCM.

Evaluate tuning results

Before large config changes (e.g. raising parallelism or DB pool sizes), it is recommended to run load tests with representative DAGs to measure effects on scheduler, DB, and workers. The best way to evaluate the effectiveness of Airflow performance tuning would be using A/B comparison tests to isolate impact.

Airflow UI overview

Contents

Table of Contents

- Airflow UI
  - DAGs
  - Datasets
  - Security
  - Browse
  - Admin
  - Docs
- Flower UI

Airflow UI

Airflow's UI provides the ability to monitor DAGs, work with Datasets, change security settings, configure the metadata database options (for example, connections to external systems), and more.

Prerequisites: To access the Airflow UI, you must have a working Airflow project. See Install Airflow.

Web UI URLs

System	URL	Documentation
Airflow Web UI	http://localhost:8080 (Port 8080)	DAGs
Flower	http://localhost:5555 (Port 5555)	Flower

If you have an Astro API key, you can also configure your Astro Cloud UI instance of Airflow 2.0+. This is the same Airflow UI that is available locally, but with a few additional features. See Configure Astro Cloud UI.

Airflow's grid view with multiple DAGs

Airflow UI

DAGs

The DAGs view is the main page of the Airflow UI. It displays a list of all pipelines with information about them, several actions, and controls to pull pages.

- DAG - the unique DAG identifier. Next to the DAG name, you can find toggle buttons to pause the data pipeline. By clicking on the DAG name, you can navigate to the DAG's page.
- Owner - the owner of the DAG, as defined in the DAG's code. By clicking on the owner's name, you can filter the DAGs view to show only those belonging to that owner.
- Runs - the success of previous DAG runs: queued, success, running, failed.
- Schedule - the DAG's schedule, as a cron expression or a cron preset. Hover over the schedule to see a timetable.
- Last Run - the start time of the last DAG run. You can also hover over the DAG run to see the exact start time and date.
- Next Run - the next scheduled DAG run. Hover over the schedule to see the exact start time and date. Learn more about how the Next Run is calculated in Scheduling & Triggers.
- Recent Tasks - a summary of the most recent task runs in the DAG.
- Actions - quick actions for the DAG, such as triggering a DAG run. You can also delete DAG runs and clear task instances. Learn more in Rerunning DAGs.
- Links - links to any custom external resources or URLs that were added to the DAG.

Additionally, you can use search or filter the list of DAGs by tag.

Airflow UI's DAGs page

The DAGs page can also be accessed programmatically with the DAG's endpoint.

- Pause / unpause DAGs: Use the toggle button to the left of the DAG name. DAGs are set to unpaused by default and the DAG is active. To use schedules for a specific task, click on the DAG name to open the DAG's page, then click on the Code button. Here you can see the DAG's code and confirm that the schedule is correct.
- Search DAGs: Use the search bar at the top of the DAGs page to search DAGs by name.
- Filter DAGs by tag: Click on a tag to filter the DAGs list to only those with that tag. To see all DAGs again, click Reset.
- Trigger a DAG: Click the play button under the Actions column.
- Delete a DAG: Click the delete button under the Actions column.

Airflow UI's DAGs page

Datasets

Click the Datasets page, you can see a list of all existing datasets and the past 30 days' representation of relationships between datasets and DAGs on the right. When you click on a dataset, the dependency graph is highlighted and the dataset's properties are displayed.

The page provides the same Filter by DAG function as the DAGs view. Click on a DAG to see all of its upstream datasets.

Airflow UI's Datasets page

Security

The Security View provides access to the information about users, roles, actions, and permissions.

List Users

The List Users page displays a list of the users. Each page has the following fields: first name, last name, email, active, and roles. To add a user, click the blue + button next to the search bar.

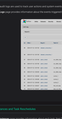
Airflow UI's List Users page

List Roles

The List Roles page displays a list of user roles. Each role has a name and a list of permissions given to the role.

You can view, edit, or delete a role by selecting the corresponding action.

To add a role, click the blue + button next to the search bar and fill out the form.



Airflow UI's List Roles page

User's Statistics

The User's Statistics page displays a chart of users' login counts. You can change the view by selecting either login counts or failed login attempts.

View's Permissions / Resources

- The Actions page displays a list of actions that can be granted to a role.
- The Resources page displays a list of resources to which you can grant access.
- The Permissions page displays a list of actions that can be granted to a particular resource.

Browse

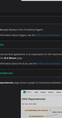
The Browse menu provides a set of pages with lists that allow you to view/control various parts of Airflow.

DAG Runs

The DAG Runs page displays a list of all DAG runs.

You can manage several DAG runs by selecting the necessary DAG runs and selecting an action to apply to the them.

By clicking on the Map ID == -1 or Run ID, you can navigate to the DAG run page.



Airflow UI's DAG Runs page

Jobs

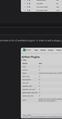
The Jobs page displays a list of all jobs with information about their DAG, current state, start and finish times, latest heartbeat, executor class, hostname, and unixname.

Airflow UI's Jobs page

Audit Logs

In Airflow, audit logs are used to track user actions and system events that occur during the execution of DAGs and tasks.

The Audit Logs page provides information about the event triggered by the user.



Airflow UI's Audit Logs page

Task Instances and Task Reschedules

The Task Instances page provides information about each task: their DAG, start and finish times, and more.

You can manage several tasks at once by selecting the necessary tasks and choosing an action to apply to them.

By clicking on the Map ID, Task ID, or Run ID, you can navigate to the task's DAG page.

The Task Reschedules page displays a list of all times that a task was rescheduled.

Airflow UI's Task Instances page

Triggers

The Triggers page displays a list of existing triggers.

For more information about triggers, see the Deferrable Operators.

SLA Misses

The SLA Misses page displays a list of all tasks that missed their SLA (service level agreement). If a task takes longer than the time it's set to in the SLA, it's an SLA miss.

For more information about the SLAs, see Notifications.

DAG Dependencies

The DAG Dependencies page displays a graph of dependencies between DAGs, triggers, sensors, and datasets.

Airflow UI's DAG Dependencies page

Admin

The Admin menu provides access common options and information about the Airflow configuration.

Variables

A variable is a simple key-value pair that is stored in Airflow's server. On the page, you can create variables that can be used in DAGs and both code and templates. Click on the + button next to the search bar to create the following record: key, val, description, and is_encrypted. Click on the edit button to edit a record, or the delete button to delete a record.

To upload several variables at once click on the Choose file... button and select the file, and then click on Import Variables.

Airflow UI's Variables page

Configurations

The page displays the table with the configuration of your Airflow instance.

Connections

The connections enable integration of Airflow with other services. The page displays a list of created connections.

To create a connection, click on the + button next to the search bar. For more information, see Connections. Also a connection can be created by running Test.

Airflow UI's Connections page

Plugins

The page provides a list of installed plugins. In order to add a plugin, you have to add a connection first.

Airflow UI's Plugins page

Providers

The page displays available providers for integration with other services.

By clicking on the providers's name you can navigate to the Airflow doc website's page to find out more. The docs in the documentation's style for Services URLs and usages.

Airflow UI's Providers page

Pools

The page displays a list of existing pools. In Airflow, pools are used to limit the concurrency of a set of tasks. By default, all tasks run through a default pool (but you have the following properties):

- Pool - the name of the pool. By clicking on it a pool's name you can navigate to the list of tasks that belong to that pool.
- Slots - the number of slots in the pool.
- Running Slots - the number of running slots by the activity of the pool or pool, you can navigate to the list of tasks that occupy those slots.
- Queued Slots - the number of queued slots by the activity of the pool or pool, you can navigate to the list of tasks that are in the state like running those slots.
- Scheduled Slots - the number of slots reported for the scheduled tasks. By clicking on the pool's name of slots, you can navigate to the list of tasks that occupy those slots.

To create a pool, click on the + button next to the search bar and fill out the form with the following fields:

You can edit or delete a pool by choosing the corresponding actions.

Airflow UI's Pools page

XComs

In Airflow, XComs are used to pass data between tasks. On the page, you can see a list of available XComs.

Airflow UI's XComs page

Docs

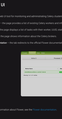
The Docs menu provides links to the following helpful resources:

- Documentation - the official Airflow documentation.
- Airflow Website - the official Airflow website.
- GitHub Repo - the Airflow project repository on GitHub.
- REST API Reference (Swagger UI) - the Swagger UI interface documentation for the Swagger format.
- REST API Reference (Redoc) - the REST API interface documentation for the Redoc format.

Flower UI

Flower is a web UI tool for monitoring and administering Celery clusters. The Flower web UI has four tabs:

- Workers - The page provides a list of currently running workers and information about them.
- Tasks - The page displays a list of all tasks with their current state, UUID, name and some information.
- Broker - The page shows information about the Celery brokers.
- Documentation - The tab redirects to the official Flower documentation.



Flower homepage

For more information about Flower see the Flower documentation.

Create a simple DAG

by Ivo Bernardo

Contents

Step 1. Prepare a DAG file
Step 2. Run the DAG via Web UI
Step 3. Check the results

To Table of Contents

This article explains how to create and run your first DAG to process CSV files. This DAG will simulate an ETL process and include the following tasks:

- **Extract**: This task loads the records from a source CSV file and transfers them to the next task. We will take the test file [people_ages_titles.csv](#) containing a list of 1000 people with their names and ages.
- **Transform**: This task selects the records received from the previous task according to the following criterion: the age of the people should be less or equal to 18.
- **Load**: This task loads the transformed records received from the previous task to a new CSV file.

> **NOTE**
>
> - In practice, you can modify this example by changing input files to database connections.
> - See more examples in the [official documentation](#).
> - The example described in this section refers to the Airflow version 1.10.11.

Step 1. Prepare a DAG file

Step 1 is divided by following these steps:

1. Add the `ETL_basic.py` file to your DAG folder that is usually located in the home AirFlow folder and called *dags*. The content of the file is described below:

```
import pandas as pd
from datetime import timedelta, datetime
from airflow import DAG
from airflow.operators.python_operator import PythonOperator

dag = DAG( ITL_test',
    schedule_interval=timedelta(days=1),
    start_date=datetime(2021, 12, 1), )

def extract_people(path: str, **context) -> None:
    extracted_people = pd.read_csv(path,sep=',',header=1)
    context['ti'].xcom_push(key='extracted_people', value=extracted_people)

def transform_people(**context) -> None:
    extracted_people = context['ti'].xcom_pull(key='extracted_people', task_ids=['extract_people'][0])
    transformed_people = []
    for index, row in extracted_people.values:
        if int(person[1]) <= 18:
            transformed_people.append(
                {'name': person[0],
                 'age': person[1],
                }
            )
    context['ti'].xcom_push(key='transformed_people', value=transformed_people)

def load_people(path: str, **context) -> None:
    transformed_people = context['ti'].xcom_pull(key='transformed_people', task_ids=['transform_people'])
    loaded_people = pd.DataFrame(transformed_people[0])
    loaded_people.to_csv(path, index=False)

task_extract_people = PythonOperator(
    task_id='extract_people',
    python_callable=extract_people,
    op_kwargs={'path' : '/opt/airflow/input/people_ages_titles.csv'},
    dag=dag,
    provide_context=True
)

task_transform_people = PythonOperator(
    task_id='transform_people',
    python_callable=transform_people,
    dag=dag,
    provide_context=True
)

task_load_people = PythonOperator(
    task_id='load_people',
    python_callable=load_people,
    op_kwargs={'path' : '/opt/airflow/output/loaded_people.csv'},
    dag=dag,
    provide_context=True
)

task_extract_people >> task_transform_people >> task_load_people
```

- **Importing**. We import [pandas](#) to work with CSV files.
- **The DAG definition**. We create the `ETL_test` object of type DAG and make it run every day since 2021-12-17 00:00:00
- **The first Python function**. The first task `extract_people` will use it. The function gets the CSV file from the defined `path` and loads it using the `pandas` module. It uses the `context` object to transfer the loaded data through the XCOM data transport service to the next task `transform_people`.
- **The second Python function**. The second task `transform_people` will use it. This function gets the extracted data from the first task `extract_people` and finds in it the records with the specified value of the criterion: `Age` (one or equal to 18). Then it copies the found data to an array and makes that array available for the next task `load_people`.
- **The third Python function**. The third task `load_people` will use it. The function gets the transformed data from the second task `transform_people` and saves it in a new file located at the specified `path`. For this purpose, it uses the module `pandas`.
- **Declaration of tasks**. They pair defines operators, functions, and input parameters that the tasks will use. It also binds the tasks to the DAG defined earlier.
- **Dependency definition**. This code defines dependencies between created tasks.

> **IMPORTANT**
>
> If you use AirFlow installed in a Docker container (or AOD, it comes exactly like this), be careful with paths. For example, use */opt/airflow/* instead of */src/airflow/home/*.

2. Make sure that the file is located in the DAG directory by running this command:

```
$ ls -la /src/airflow/home/dags/
```

The output looks similar to this:

```
total 8
drwxr-xr-x 2 50000 50000   45 Dec 15 11:12
drwxr-xr-x 4 root  root   128 Dec 15 11:12
-rw-r--r-- 1 50000 50000  999 Dec 14 07:39 ETL_check.py
-rw-r--r-- 1 root  root  1771 Dec 15 11:33 ETL_test.py
drwxr-xr-x 2 50000 50000    7k Dec 15 11:16 __pycache__
```

3. Make the input and output folders in the AirFlow home directory:

```
$ sudo mkdir /src/airflow/home/input
$ sudo mkdir /src/airflow/home/output
```

Make sure the folders are created by listing the home directory:

```
$ ls -la /src/airflow/home/
```

The output looks similar to this:

```
total 48
drwxr-xr-x 6 50000 50000  126 Dec 15 13:50 .
drwxr-xr-x 4 root  root   128 Dec 15 11:12 ..
-rw-r--r-- 1 root  root  37324 Dec 15 07:33 airflow.cfg
-rw-r--r-- 1 root  root  17661 Dec 15 07:33 airflow-webserver.pid
drwxr-xr-x 5 50000 50000   45 Dec 15 11:12 dags
drwxr-xr-x 2 root  root    6 Dec 15 13:57 input
drwxr-xr-x 5 50000 50000  135 Dec 15 13:56 logs
drwxr-xr-x 2 root  root    6 Dec 15 13:50 output
-rw-r--r-- 1 50000 50000 2520 Dec 14 07:39 unittests.cfg
```

4. Download the source file [people_ages_titles.csv](#) to the local file system of the AirFlow server. Make sure the file is uploaded successfully using the following command:

```
$ ls -la ~
```

The result looks similar to this:

```
total 64
drwx------ 7 deoha deoha  296 Dec 17 15:14 .
drwxr-xr-x 3 root  root    19 Aug 31 11:55 ..
drwx------ 3 deoha deoha   17 Aug 31 15:29 .ansible
drwx------ 2 deoha deoha 20503 Dec 17 10:57 .bash_history
-rw-r--r-- 1 deoha deoha   18 Apr  1  2020 .bash_logout
-rw-r--r-- 1 deoha deoha  141 Apr  1  2020 .bash_profile
-rw-r--r-- 1 deoha deoha  372 Apr  1  2020 .bashrc
drwx------ 3 deoha deoha   17 Dec  7 11:44 .cache
-rw-r--r-- 1 deoha deoha 7227 Nov 25 07:05 .conta
-rw-r--r-- 1 deoha deoha 17661 Dec 17 15:14 people_ages_titles.csv
drwx------ 3 deoha deoha   19 Dec  1 14:56 .pki
drwxrwxr-x 2 deoha deoha   21 Nov 30 09:25 .ssClive
drwxr-xr-x 2 deoha deoha   29 Aug 31 11:55 .ssh
```

5. Copy the *people_ages_titles.csv* file to the previously created folder *input*

```
$ sudo cp ~/people_ages_titles.csv /src/airflow/home/input
```

Make sure the operation is successful by running the following command:

```
$ ls -la /src/airflow/home/input
```

The output looks similar to this:

```
total 20
drwxr-xr-x 2 root root    36 Dec 18 14:03
drwxr-xr-x 5 50000 50000 135 Dec 18 13:58 ..
-rw-r--r-- 1 root root 17661 Dec 18 14:03 people_ages_titles.csv
```

6. Change the owner of the created *input* and *output* folders and their contents to `airflow`. Otherwise, errors may occur when accessing files during the DAG execution. Use the following commands:

```
$ sudo chown -R 50000:50000 /src/airflow/home/input
$ sudo chown -R 50000:50000 /src/airflow/home/output
```

To check the result, run this command:

```
$ ls -la /src/airflow/home/
```

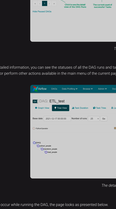
The output looks similar to this:

```
total 48
drwxr-xr-x 6 50000 50000  126 Dec 18 13:50 .
drwxr-xr-x 4 root  root   128 Dec 18 11:12 ..
-rw-r--r-- 1 root  root  37324 Dec 18 07:33 airflow.cfg
-rw-r--r-- 1 root  root  17661 Dec 18 07:33 airflow-webserver.pid
drwxr-xr-x 5 50000 50000   45 Dec 18 11:12 dags
drwxr-xr-x 2 50000 50000   36 Dec 18 14:03 input
drwxr-xr-x 5 50000 50000  135 Dec 18 13:56 logs
drwxr-xr-x 2 50000 50000    6 Dec 18 13:50 output
-rw-r--r-- 1 50000 50000 2520 Dec 14 07:39 unittests.cfg
```

Step 2. Run the DAG via Web UI

To run and monitor the DAG created earlier, follow these steps:

1. Open the home page of the AirFlow Web UI. If everything is configured correctly, you will see the `ETL_test` DAG at the **DAGs** tab. Otherwise, the errors will be displayed:



The new DAG is displayed

2. Activate the DAG using one of the following ways:

- Click the toggle and wait for the next running according to the configured schedule.
- Open the DAG and click the icon **Trigger Dag** to start the DAG immediately.

Activating the DAG

3. After a DAG run starts, you can see the status information at the home page: the full number of successful, failed, running, and other types of DAG runs and task instances. To view the detailed information about the DAG runs, click the name of the DAG:

The DAG is running

4. In the detailed information, you can see the statuses of all the DAG runs and task instances. If necessary, you can change the view mode to **Graph View** in order to analyze task duration or perform other actions available in the main menu of the current page:

The detailed view of the DAG Runs

If some errors occur while running the DAG, the page looks as presented below:

The view with errors

Step 3. Check the results

In conclusion, let's check whether the DAG has completed all the tasks correctly. Make sure that the output file *loaded_people.csv* is located in the output folder:

1. Check the contents of the output folder:

```
$ ls -la /src/airflow/home/output/
```

The list of files looks similar to this:

```
total 8
drwxr-xr-x  2 50000 50000   31 Dec 18 14:25 .
drwxr-xr-x  6 50000 50000  126 Dec 18 13:50 ..
-rw-r--r--  1 50000 50000 832 Dec 18 14:25 loaded_people.csv
```

2. Get the output data:

```
$ cat /src/airflow/home/output/loaded_people.csv
```

The output looks similar to this:

```
name,Age
Briana Bettis,18
Campbell Tibbs,15
Thompson Barren,16
Stokes Millis,18
Davia Kian,15
Vaughn Belle,18
Rumal Bizaare,18
Morton Berrins,15
Bellamy Bettis,18
Bolton Cimarron,16
James Loretta,16
Myers Rizay,18
Torres Elsie,18
Phelps Rona,16
Hill Katherine,18
Mcgee Ashley,16
Fitzgerald,Timothy,16
Miller Francis,16
Banks Catherine,16
Casey Lincoln,18
```

Work with TaskFlow

Elena Kvetsyuzhenko

Contents

To Table of Contents

Use TaskFlow with traditional operators
- Pass data between tasks
- Cross-functional dependencies

TaskFlow DAG structure

The TaskFlow API, added in Airflow 2, introduces a new approach for creating DAGs that enables direct function-to-function parameter passing and helps to make the DAGs code more concise and easier to read.

In previous Airflow versions, the most common way to write a DAG was using Python operators. When this approach is used, the tasks would be defined as follows:

```
def extract_people(path: str, **context) -> None:
    extracted_people = pd.read_csv(path,sep=',',header=1)
    context['ti'].xcom_push(key='extracted_people', value=extracted_people)

def load_people(path: str, **context) -> None:
    extracted_people = context['ti'].xcom_pull(key='extracted_people',task_ids=['extract_people'])[0]
    loaded_people = pd.DataFrame(extracted_people[0])
    loaded_people.to_csv(path, index=None)

task_extract_people = PythonOperator(
    task_id='extract_people',
    python_callable=extract_people,
    op_kwargs={'path': '/opt/airflow/input/people_ages_titles.csv'},
    dag=dag,
    provide_context=True
)

task_load_people = PythonOperator(
    task_id='load_people',
    python_callable=load_people,
    op_kwargs={'path': '/opt/airflow/output/loaded_people.csv'},
    dag=dag,
    provide_context=True
)

task_extract_people >> task_load_people
```

This is a simplified version of the traditional DAG. The full example is available in the Create a simple DAG article.

The example above illustrates limitations of the traditional approach that could lead to bloated code:

- It is necessary to use a context object to transfer the loaded data through the XCom data transport service to the next task.
- You have to separately define operators, functions, and input parameters that the tasks will use.

The TaskFlow API introduces specific decorators that handle the data transferring between tasks using the same XCom (but without the need of explicitly calling it) and eliminate the need to instantiate Python operators.

When you write a decorated function, you don't need to specify `task_id` or `python_callable`. TaskFlow uses the name of the function as a task ID, but you can pass a `task_id` to the decorator as a parameter.

Airflow supports decorators for tasks, groups of tasks, and DAGs. In the TaskFlow paradigm, the same example would look like this:

```
@task(task_id="extract_people", retries=2) 1
def extract_people(path: str) -> pd.DataFrame: 2
    extracted_people = pd.read_csv(path, sep=',', header=1)
    return extracted_people 3

@task()
def transform_people(extracted_people: pd.DataFrame) -> list:
    transformed_people = []
    for person in extracted_people.values:
        if int(person[1]) <= 18:
            transformed_people.append({
                'Name': person[0],
                'Age': person[1]
            })
    return transformed_people

extracted_data = extract_people(SOURCE_PATH) 4
transformed_data = transform_people(extracted_data)
```

1. Define a task by using the `@task` decorator. If required, pass the necessary task-level parameters to the decorator.
2. Define the task's function.
3. There's no need to use `xcom_pull` and `xcom_push` since the function's return value is automatically passed to other tasks via XCom when the dependencies are defined.
4. Define your task dependencies by calling the task functions.

Use TaskFlow with traditional operators

While traditional operators offer more fine-grained control over data pipelines, TaskFlow makes cleaner transitions in workflows and more readable code.

In Airflow, you can use both traditional operators and the TaskFlow API.

Pass data between tasks

To transfer data from a decorated task to a traditional one, use the `.output` attribute of the task object when calling the decorated function.

For example:

```
def traditional_function():
    return 10

@task
def taskflow_function(x):
    return x + 5

sum_task = PythonOperator(
    task_id="sum_task",
    python_callable=traditional_function,
)

taskflow_function(sum_task.output)  # the task will return 15
```

To transfer data from a traditional task to a decorated one, you can call the decorated function in the traditional operator.

For example:

```
@task
def taskflow_function():
    return 10

def traditional_function(x):
    return x + 5

sum_task = PythonOperator(
    task_id="sum_task",
    python_callable=traditional_function,
    op_args=[taskflow_function()]
)

traditional_function >> taskflow_function()

# sum_task will return 15
```

Additionally, you can use `xcom_pull` and `xcom_push` between traditional and decorated functions as well.

Since TaskFlow decorators use the same traditional XCom functionality to save a function's return values, you can pull from a decorated task using the same `xcom_pull` syntax.

For example:

```
@task
def taskflow_function():
    return 10

def traditional_functions(**kwargs):
    ti = kwargs['ti']
    received_data = ti.xcom_pull(task_ids="taskflow_function")
    print(received_data)

traditional_task = PythonOperator(
    task_id="traditional_task",
    python_callable=traditional_function,
)

traditional_function >> taskflow_function()

# sum_task will return 15
```

IMPORTANT

TaskFlow saves the return value of a function only if the return value is used elsewhere in the DAG.

Cross-functional dependencies

When using both types of functions in the same DAG, you have to define dependencies according to each function type: function-to-function parameter passing for TaskFlow and using bit-shift operators for traditional functions.

For example:

```
@task
def taskflow_function_10():
    return 10

@task
def taskflow_function_20(x):
    return x + 20

def traditional_function(x):
    return x + 30

sum_task = PythonOperator(
    task_id="sum_task",
    python_callable=traditional_function,
)

taskflow_function_20(taskflow_function_10()) >> traditional_function

# sum_task will return 60
```

TaskFlow DAG structure

In Airflow 2, tasks are written as Python functions, decorated with `@task`. When creating a DAG, the task functions must be located inside the DAG function, which is decorated with `@dag`.

For example:

```
from datetime import timedelta, datetime
import pandas as pd
from airflow.decorators import dag, task 1


SOURCE_PATH = '/opt/airflow/input/people_ages_titles.csv'
TARGET_PATH = '/opt/airflow/output/loaded_people.csv'

@dag( 2
    schedule_interval=timedelta(days=1),
    start_date=datetime(2021, 12, 17),
    catchup=False,
    tags=['example'],
)
def etl_example():

    @task() 3
    def extract_people(path: str) -> pd.DataFrame:
        extracted_people = pd.read_csv(path, sep=',', header=1)
        return extracted_people

    @task()
    def transform_people(extracted_people: pd.DataFrame) -> list:
        transformed_people = []
        for person in extracted_people.values:
            if int(person[1]) <= 18:
                transformed_people.append({
                    'Name': person[0],
                    'Age': person[1]
                })
        return transformed_people

    @task()
    def load_people(transformed_people: list, path: str) -> None:
        loaded_people = pd.DataFrame(transformed_people)
        loaded_people.to_csv(path, index=None)

    extracted_data = extract_people(SOURCE_PATH) 4
    transformed_data = transform_people(extracted_data)
    load_people(transformed_data, TARGET_PATH)

etl_dag = etl_example() 5
```

1. Import the task and DAG decorators.
2. Instantiate a DAG using the `@dag` decorator and its Python function `example_etl`. By default, the name of the function acts as the DAG identifier.
3. Inside the DAG function, define your task functions.
4. Define the dependencies to ensure data transfer between the tasks.
5. Invoke the DAG function. All the decorated functions in your DAG must be called in that DAG's file so that Airflow can register them.

Use sensors in Airflow

Elena Kostyuchenko

Contents

To Table of Contents

Sensor parameters

Work with sensors

Write a custom sensor

In Airflow, a sensor is an operator that is used when, in a DAG, it is required to wait until a condition is met to continue with the tasks.

The conditions depend on the type of sensor. Airflow provides standard operators that, for example, allow a task to wait for a certain time to pass (TimeDeltaSensor), wait until a file appears in the system (FileSensor), or wait until another task in another DAG has finished (ExternalTaskSensor). When a sensor's condition is met, it can trigger downstream tasks.

Sensor parameters

Sensors have the following base parameters:

- `poke_interval` – the time that the job should wait in between each try. The default is 60 seconds.
- `timeout` – the maximum number of seconds a sensor is allowed to run for. The default timeout is 7 days, or 60 * 60 * 24 * 7 seconds, after which the sensor's task will be terminated. The value must be less than the schedule interval of your DAG to avoid hanging tasks.
- `mode` – the indicator of how long a sensor task will occupy a worker slot. Can have one of two values:
  - `poke` – the sensor takes up a worker slot until it gets completed;
  - `reschedule` – the sensor takes up a worker slot only during checks and, in between each `poke_interval`, the scheduler releases the slot.

Work with sensors

Sensor operators are derived from the `airflow.sensors.base` class and inherit its attributes. Airflow provides several sensors, which are described in the table below, and some of the sensors can be available through different providers.

Standard Airflow sensors

Name	Description
airflow.sensors.bash	Executes a bash command or a script that can be used as a custom sensor. The command should return `0` when it succeeds, any other value otherwise
airflow.sensors.date_time	Waits until the specified datetime. An advantage of this sensor is idempotence for the `target_time`. It can be applied to cases where TimeSensor and TimeDeltaSensor are not suited
airflow.sensors.external_task	Waits for a task in another DAG to complete before getting itself completed
airflow.sensors.filesystem	Waits for a file or folder to land in a filesystem. If the path given is a directory, then this sensor will return `True` if the directory is not empty
airflow.sensors.python	Executes an arbitrary callable and waits for it to finish. The `@task.sensor` decorator is recommended over the classic PythonSensor to execute Python callables to check for `True` condition
airflow.sensors.time_delta	Waits for a specified time period
airflow.sensors.time_sensor	Waits until the specified time of the day
airflow.sensors.weekday	Waits until the first specified day of the week. For example, if the execution day of the task is `2018-12-22` (Saturday) and you pass `FRIDAY`, the task will wait until next Friday

A more detailed description of each sensor is available in the Airflow documentation.

Here's an example of a possible sensor usage in a DAG:

```
from datetime import datetime, timedelta

from airflow import DAG
from airflow.sensors.time_delta import TimeDeltaSensor, TimeDeltaSensorAsync (1)

dag = DAG(
    dag_id="sensor_example",
    schedule=None,
    start_date=datetime(2023, 6, 4),
)

time_sensor_example = TimeDeltaSensor( (2)
   task_id='time_sensor_example',
   dag=dag,
   delta=timedelta(minutes=1), (3)
   poke_interval=10, (4)
   timeout=30, (5)
   soft_fail=True, (6)
)
```

1. Import the sensor operators. `TimeDeltaSensorAsync` is an async version of the `TimeDeltaSensor` operator.
2. Instantiate the sensor. If the sensor mode is not defined, it defaults to `poke`.
3. Specifies for how long to wait before the task can be marked as succeeded.
4. Defines how often a sensor checks the conditions (in seconds).
5. Defines for how long the sensor will be working.
6. Marks the sensor's task as `SKIPPED` if it fails.

To avoid deadlocks and other performance issues, it is recommended to check the following conditions:

- Set up a timeout for your sensor that will work for each particular case. The default value for this parameter might be too long in some cases.
- Use the reschedule mode for long-running sensors so that they do not constantly occupy a working slot, and the poke mode for a short polling interval (i.e., less than 5 minutes) to avoid scheduler overload.
- Make sure that the `poke_interval` parameter is applicable to your case. The default value might be unnecessarily frequent.

Write a custom sensor

To create a custom sensor, you have to inherit the `BaseSensorOperator` class in your Python function.

In this example, we will create a new sensor that checks the existence of a file.

1. In the *dags* directory of your Airflow home, add a new Python file that will contain your custom sensor:

```
$ sudo vi /opt/airflow/dags/custom_sensor.py
```

2. Add the code for your sensor:

```
import os
from airflow.sensors.base import BaseSensorOperator (1)

class MyCustomFileSensor(BaseSensorOperator): (2)
    def __init__(self, filepath, *args, **kwargs): (3)
        self.filepath = filepath
        super().__init__(*args, **kwargs)

    def poke(self, context): (4)
        print(f'checking if a file {self.filepath} exists...')
        return os.path.exists(self.filepath) (5)
```

1. Import the base sensor class.
2. Inherit the base class in your custom sensor.
3. Add additional arguments and keep the arguments of the parent class.
4. Override the `poke` method.
5. Add the logic for checking the existence of the file.

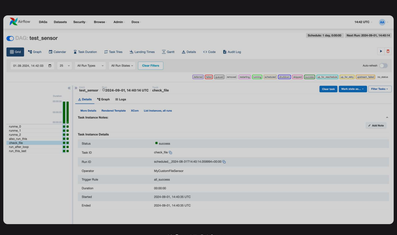
3. Call your custom sensor inside a DAG:

```
from custom_sensor import MyCustomFileSensor (1)

check_file = MyCustomFileSensor(    (2)
    task_id='check_file',
    filepath='/opt/airflow/test_file.txt',
    poke_interval=10,
    timeout=60,
    dag=dag
)
```

1. Import the custom sensor class.
2. Call the custom sensor function with the required parameters.

4. Go to the Airflow web UI to trigger the DAG and make sure that the custom sensor function is working.



Airflow UI: DAG page

The DAG page shows that the `check_file` task with the `MyCustomFileSensor` class has finished successfully.

Customize DAG scheduling

Elena Kosyachenko

Contents

To Table of Contents

Overview
Cron-based schedule
Timedelta schedule
Events timetable
Custom timetables

Overview

Airflow provides the ability to run data pipelines on a schedule, allowing you to define the data interval and logical date of each DAG run.

A DAG schedule can be defined with one of the following tools:

- built-in timedelta object of the Python `datetime` module;
- Linux Cron;
- custom timetable classes.

All the schedule parameters, like data interval, date and time, and other, are determined in a DAG's internal timetable. DAGs scheduled with a cron expression or a timedelta object are internally converted into a timetable as well.

Airflow has default timetables that handle cron expressions or timedeltas, but for DAGs with a more specific schedule, it might be required to create a custom timetable class and pass it to the DAG's schedule argument.

The examples of cases when a custom timetable is needed:

- Data pipelines start at different times each day.
- The DAG schedule requires the use of a different calendar.
- DAG runs with overlapping data intervals. For example, a DAG run must cover the period of the previous runs.
- DAG runs with uneven intervals.

Cron-based schedule

Cron expressions in Airflow follow the standard cron syntax, which consists of five fields separated by whitespaces, where each field can accept values listed in the table below.

A cron expression syntax is `<1> <2> <3> <4> <5>`, where:

- `<1>` - the minute in the range: 0-59.
- `<2>` - the hour in the range: 0-23.
- `<3>` - the day of the month in the range: 1-31.
- `<4>` - the month in the year. The range is from 1 to 12, where 1 is January.
- `<5>` - the numeric indicator of a day in the week. Each day is represented by a number in a range from 0 to 6, where 0 is Sunday.

For example, a DAG's schedule represented by a cron expression `0 18 * * 5` runs every Friday at 6 PM.

Possible value	Example
A single value	5
A range of values defined using a dash	1-5
A list of values separated by commas	1,3,5
Every possible value is represented by an asterisk	*

Airflow has two timetables: `CronTriggerTimetable` and `CronDataIntervalTimetable` for handling cron schedules. Both timetables trigger DAG runs at the same time but the timestamp for the `run_id` for DAGs will be different.

For example, if a DAG with the schedule defined as `0 0 * * *` (12 AM every day) is enabled at 3 PM on January 31, `CronTriggerTimetable` will trigger a new DAG run at 12 AM on February 1, but `CronDataIntervalTimetable` will trigger the DAG immediately, because the DAG run for the daily time interval beginning at 12 AM on January 31 did not occur yet.

An example DAG for `CronTriggerTimetable` that triggers a DAG run at 1 AM on Wednesday:

```
from datetime import timedelta
from airflow import DAG
from airflow.timetables.trigger import CronTriggerTimetable

def cron_timetable(*args, **kwargs):
    return CronTriggerTimetable(*args[ 'cron_expression'])

dag = DAG(
    dag_id= my_dag',
    timetable=cron_timetable(cron_expression='0 1 * * 3', timezone="UTC"),
    start_date=days_ago(2),
    tags['example'],
)

@dag(schedule=CronTriggerTimetable("0 1 * * 3", timezone="UTC"), ...)
def example_dag():
    pass
```

When using `CronDataIntervalTimetable`, you can specify a static data interval using the `interval` argument. It must use `datetime.timedelta` or `dateutil.relativedelta.relativedelta` objects.

If specified, the DAG's data interval spans the specified duration and ends with the trigger time.

For example:

```
from datetime import timedelta
from airflow.timetables.trigger import CronTriggerTimetable

@dag(
    schedule=CronTriggerTimetable(
        "0 18 * * 5",
        timezone="UTC",
        interval=timedelta(days=4, hours=9),
    ),
    ...
)
def example_dag():
    pass
```

The DAG from the above example will run every Friday at 6 PM.

Timedelta schedule

The timedelta schedule uses a timetable that runs DAGs with a specified time interval. This timetable relies on the data interval value provided by the user and does not necessarily align execution dates with the start of the day or an hour.

You can use a timedelta schedule by providing a `datetime.timedelta` or `dateutil.relativedelta.relativedelta` values to the schedule parameter.

For example:

```
@dag(schedule=datetime.timedelta(minutes=30))
def example_dag():
    pass
```

Events timetable

Events timetable enables running DAGs at specified datetimes. The list of dates should have a moderate size since it must be loaded every time the DAG is parsed.

For example:

```
from airflow import DAG
import pendulum
from airflow.timetables.events import EventsTimetable

@dag(
    schedule=EventsTimetable(
        event_dates=[
            pendulum.datetime(2024, 4, 5, 8, 27, tz="America/Vancouver"),
            pendulum.datetime(2024, 4, 17, 8, 27, tz="America/Vancouver"),
            pendulum.datetime(2024, 4, 22, 20, 50, tz="America/Vancouver"),
        ],
        description="Example dates",
        restrict_to_events=False,
    ),
    ...
)
def example_dag():
    pass
```

Custom timetables

Custom timetables allow users to define their own schedules in Python code when cron-based and timedelta timetables are not sufficient.

To create a custom timetable, you need to:

- create a class that inherits from a timetable;
- implement the `next_dagrun_info()` and `infer_manual_data_interval()` methods, both of which return a `DataInterval` object;
- register the custom timetable as part of an Airflow plugin.

The `next_dagrun_info()` method returns the data interval for the DAG's regular schedule and specifies the logic to calculate it. The method also contains the logic for calculating the `start_date`, `end_date`, and `catchup` parameters.

The `infer_manual_data_interval()` method determines how to define the interval in the case of manual execution.

The example below illustrates how to write a DAG with a custom timetable.

To create a DAG with a custom timetable:

1. In the *plugins* directory of your Airflow home, add a new Python file that will contain your custom timetable:

```
$ sudo vi /opt/airflow/plugins/custom_timetable.py
```

2. In the *custom_timetable.py* file, define the structure of the timetable:

```
from datetime import timedelta
from typing import Optional
from pendulum import Date, DateTime, Time, timezone
from airflow.plugins_manager import AirflowPlugin
from airflow.timetables.base import DagRunInfo, DataInterval, TimeRestriction, Timetable 1

UTC = timezone("UTC") 2

class CustomTimetable(Timetable): 3
    pass

class CustomTimetablePlugin(AirflowPlugin): 4
    name = "custom_timetable_plugin"
    timetables = [CustomTimetable]
```

1. Import the dependencies.
2. Declare global variables if necessary.
3. Inherit the base class in your custom timetable.
4. Register your timetable as an Airflow plugin.

3. In the `CustomTimetable` class, write the implementation of the `next_dagrun_info()` method:

```
class CustomTimetable(Timetable):

    def next_dagrun_info(
        self,
        *,
        last_automated_data_interval: Optional[DataInterval],
        restriction: TimeRestriction,
    ) -> Optional[DagRunInfo]:
        if last_automated_data_interval is not None:  1
            last_start = last_automated_data_interval.start
            delta = timedelta(days=1)
            if last_start.hour == 0:
                next_start = last_start.set(hour=16, minute=30).replace(tzinfo=UTC)
                next_end = (last_start+delta).replace(tzinfo=UTC)
            else:
                next_start = (last_start+delta).set(hour=0, minute=0).replace(tzinfo=UTC)
                next_end = (last_start+delta).replace(tzinfo=UTC)
        else:  2
            next_start = restriction.earliest
            if next_start is None:
                return None
            if not restriction.catchup:  3
                next_start = max(next_start, DateTime.combine(Date.today(), Time.min).replace(tzinfo=UTC))
            next_start = next_start.set(hour=0, minute=0).replace(tzinfo=UTC)
            next_end = next_start.set(hour=16, minute=30).replace(tzinfo=UTC)
        if restriction.latest is not None and next_start > restriction.latest:  4
            return None
        return DagRunInfo.interval(start=next_start, end=next_end)
```

1. Write the logic for the case when there was a previous run.
2. Write the logic for the first run.
3. Implement the catchup behavior.
4. Make sure the next run won't start out of schedule.

4. In the `CustomTimetable` class, write the logic for scheduling runs in case of a manual start in the `infer_manual_data_interval()` method:

```
class CustomTimetable(Timetable):

    def infer_manual_data_interval(self, run_after: DateTime) -> DataInterval:
        delta = timedelta(days=1)
        if run_after >= run_after.set(hour=0, minute=0) and run_after <= run_after.set(hour=16, minute=30):
            start = (run_after-delta).set(hour=16, minute=30, second=0).replace(tzinfo=UTC)
            end = run_after.set(hour=0, minute=0, second=0).replace(tzinfo=UTC)
        elif run_after >= run_after.set(hour=16, minute=30) and run_after.hour <= 23:
            start = run_after.set(hour=0, minute=0, second=0).replace(tzinfo=UTC)
            end = run_after.set(hour=16, minute=30, second=0).replace(tzinfo=UTC)
        else:
            start = (run_after-delta).set(hour=0, minute=0).replace(tzinfo=UTC)
            end = (run_after-delta).set(hour=16, minute=30).replace(tzinfo=UTC)
        return DataInterval(start=start, end=end)
```

5. Use the custom timetable in a DAG:

```
from custom_timetable import CustomTimetable 1

@dag(
    dag_id="example_dag",
    start_date=datetime(2024, 3, 10),
    schedule=CustomTimetable(), 2
    default_args={
        "retries": 1,
        "retry_delay": duration(minutes=3),
    },
    catchup=True
)
```

1. Import the custom timetable.
2. Call the custom timetable class to use it as the DAG's schedule.

For more information on how to run a DAG, see the Create a simple DAG article.

IMPORTANT

Adding or updating an Airflow plugin, including custom timetables, requires a restart of Airflow Scheduler and the web server.

Add custom operators and hooks in Airflow

Elena Kostyuchenko

AD.O

Contents

To Table of Contents

Overview

Create a custom operator

Create a custom hook

Overview

Airflow provides standard operators that, for example, allow users to execute bash commands, call arbitrary Python functions, create a connection to an external app, or check for a specific condition.

The list of operators that Airflow provides out-of-the-box is extensive, but in some cases it is necessary to modify an operator or create your own.

This article describes how to write custom operators and hooks. To learn about custom sensors, see the Use sensors in Airflow article.

When creating any custom Airflow operator, you need to:

1. Write the code for a new operator in the `__init__` method.
2. Inherit an Airflow class and either rewrite or extend its functionality by overriding methods in a derived class:
   - In the constructor, define the parameters of your operator.
   - Provide the code to execute when the runner calls the operator.
3. Place the file in a directory that is present in the Airflow's `PYTHONPATH`. By default, it's the following directories: *dags*, *_plugins*, and *config*.

The examples below illustrate implementations of these steps for writing a custom operator and a hook.

Division of the logic between operators and hooks

An operator should contain the business logic of your task and in-memory states. If you need to establish communication with an external service (API, database, etc.), it's better to write it as a hook.

A hook should not contain the logic of the operator but only expose methods that will then be used by an operator for interfacing with an external service.

By keeping the logic separated into two components, you will be able to build multiple operators that use the same hook without duplicating the code.

Operators and hooks interaction path

Create a custom operator

Here's an example of creating a custom operator based on the `airflow.models.baseoperator.BaseOperator` class:

1. In the *dags* directory of your Airflow home, add a new Python file that will contain your custom operator:

```
$ sudo vi /opt/airflow/dags/custom_operator.py
```

2. Add the code for your operator:

```
from airflow.models.baseoperator import BaseOperator  (1)

class CustomOperator(BaseOperator):  (2)
    def __init__(self, name: str, **kwargs) -> None:  (3)
        super().__init__(**kwargs)
        self.name = name

    def execute(self, context):  (4)
        message = f"Hello, {self.name}!"
        print(message)
        return message
```

(1) Import the base operator class.

(2) Inherit the base class in your custom operator.

(3) Add new arguments and keep the arguments of the parent class.

(4) Override the `execute` method with the desired functionality.

3. Call your custom operator inside a DAG:

```
from custom_operator import CustomOperator (1)

print_message = CustomOperator(    (2)
    task_id='print_message',
    name='CustomOperator',
    poke_interval=10,
    timeout=60,
    dag=dag
)
```

(1) Import the custom operator class.

(2) Call the custom operator function.

Create a custom hook

Creating a custom hook is different from the process of writing a custom operator in three ways:

- The custom hook must inherit from a hook class, for example, the `BaseHook` class or any other existing hook.
- You can include a `.get_conn()` method wrapping around a call to the `.get_connection()` method to retrieve information from an Airflow connection.
- There must exist an Airflow connection to be used by the hook.

Below is an example of how to create a custom hook function that initiates a new connection to an external tool:

1. In the *dags* directory of your Airflow home, add a new Python file that will contain your custom hook:

```
$ sudo vi /opt/airflow/dags/custom_hook.py
```

2. Add the code for your hook:

```
from airflow.hooks.base import BaseHook (1)

class CustomHook(BaseHook): (2)

    conn_name_attr = "custom_conn_id" (3)
    default_conn_name = "custom_conn_default" (4)
    conn_type = "http" (5)
    hook_name = "CustomHook" (6)

    def __init__(
        self, custom_conn_id: str = default_conn_name, *args, **kwargs
    ) -> None: (7)
        super().__init__(*args, **kwargs) (8)
        self.custom_conn_id = custom_conn_id (9)
        self.get_conn() (10)

    def get_conn(self):
        conn_id = getattr(self, self.conn_name_attr)
        conn = self.get_connection(conn_id)
        return conn
```

(1) Import the base hook class.

(2) Define the class to inherit from the base class.

(3) Add the parameter that receives the connection ID.

(4) Specify a default connection ID.

(5) Specify the connection type.

(6) Specify the name of the hook.

(7) Define the `.init()` method.

(8) Initialize the parent hook.

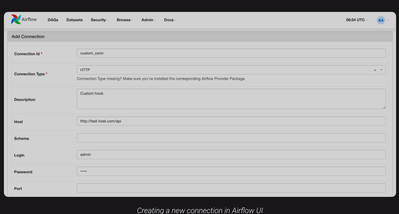
(9) Assign class variables.

(10) Optionally, call the `.get_conn()` method upon initialization.

3. Call your custom hook inside a DAG:

```
from custom_hook import CustomHook
```

4. To use the custom hook, you need to create an Airflow connection with the parameters specified in the hook code.



Creating a new connection in Airflow UI

For more information about the Airflow web interface, see the Airflow UI overview article.

Generate DAGs dynamically in Airflow

Elena Kostyuchenko

Contents

To Table of Contents

Overview

Dynamic DAGs generation

Dynamic task mapping

Overview

Traditional DAGs are convenient to use for a single-source pipeline. For example, to extract data from a file, perform the necessary actions, and load the results.

For cases where you need to run the same data pipeline with different parameters on several data sources, it's more useful to generate DAGs dynamically for each case.

There are two ways of creating DAGs for variable scenarios:

- **Dynamic DAGs generation**
  In dynamically generated DAGs, the structure of the DAG changes based on static predefined values (configuration files, environment variables, file names, etc.).
- **Dynamic task mapping**
  Dynamic task mapping enables generating parallel tasks at runtime based on the output of a previous task.

The examples in this article are written using the TaskFlow API for convenience. You can generate DAGs dynamically using traditional syntax as well.

Dynamic DAGs generation

Creating a dynamically generated DAG is similar to the process of creating a single DAG. Since Airflow executes all Python code in the dags folder, you can execute any Python code that generates DAG objects.

The DAGs are generated every `min_file_process_interval` seconds, which by default is every 30 seconds. You can change the value of this parameter by editing the *cfg_properties_template* field via ADCM.

To edit the `min_file_process_interval` property via ADCM:

1. Go to the ADCM UI and select your ADO cluster.
2. Navigate to ***Services → Airflow2 → Primary configuration*** and toggle ***Show advanced***.
3. Open the ***airflow.cfg*** section and click ***cfg_properties_template***.
4. Set the desired value for `min_file_process_interval`.
5. Save the configuration by clicking ***Save → Create*** and restart the service by clicking ***Actions → Restart***.

> CAUTION
> If the value of the `min_file_process_interval` parameter is too low, many generated DAGs might overload the scheduler.

To create a dynamic DAG:

1. In the dags directory of your Airflow home, add a new Python file that will contain your DAG generator:

```
$ sudo vi /opt/airflow/dags/dag_generator.py
```

2. Add the code for the DAG generator:

```
from airflow import DAG
from airflow.decorators import task
from datetime import datetime                                    (1)

for file in ["dev_data.csv", "test_data.csv", "prod_data.csv"]:  (2)
    dag_id = f"generated_dag_{file}"

    @dag(dag_id=dag_id, schedule="@daily",
    default_args=default_args, catchup=False,
    start_date=datetime(2025,1,1))                               (3)

    def create_dag(filename):

        @task                                                    (4)
        def extract(filename):
            return filename

        @task
        def process(filename):
            return filename

        @task
        def load(filename):
            print(filename)
            return filename

        load(process(extract(filename)))                         (5)

    create_dag(filename)                                         (6)
```

1. Import the dependencies.
2. Create the loop for generating DAGs.
3. Create the DAG function.
4. Add tasks.
5. Define your task dependencies by calling the task functions.
6. Invoke the DAG function.

If you want to use variables to configure your code, it's better to use environment variables rather than Airflow variables.

Using Airflow variables in top-level code creates a connection to the metadata database, which can slow down parsing and place extra load on the database.

You can also import variables from one of several external sources, such as configuration files.

For example, you have two JSON configuration files in the *include/dag-configs/* directory of your Airflow home. The files have the following contents:

- *dev_config.json*:

```
{
    "dag_id": "development_dag",
    "schedule": "@daily",
    "input": "dev_data.csv"
}
```

- *prod_config.json*:

```
{
    "dag_id": "production_dag",
    "schedule": "@daily",
    "input": "prod_data.csv"
}
```

You can create a Python script that will generate separate DAG files for each configuration file using the DAG file you specified as a template.

To generate DAGs using a template:

1. In the Airflow home directory, add a new Python file that will contain your DAG template:

```
$ sudo vi /opt/airflow/include/templates/dag_template.py
```

2. Add the code that the generator will use as a template for DAGs. For example, you can use the code from the previous example and substitute the parameters you want to change with placeholders:

```
from airflow import DAG
from airflow.decorators import task
from datetime import datetime

@dag(dag_id = "dag_id_placeholder", schedule="schedule_placeholder",
catchup=False, start_date=datetime(2025,1,1))                    (1)

def create_dag(filename):

    @task
    def extract(filename):
        return filename

    @task
    def process(filename):
        return filename

    @task
    def load(filename):
        print(filename)
        return filename

    load(process(extract("input_placeholder")))                  (2)

create_dag(filename)
```

1. Add placeholders for the `dag_id` and `schedule_interval` parameters.
2. Add a placeholder for the input.

3. In the Airflow home directory, add a Python file that will generate DAGs using the template:

```
$ sudo vi /opt/airflow/include/scripts/dag_generator.py
```

4. Write the code for the DAG generator:

```
import json
import os
import shutil
import fileinput

config_filepath = "include/dag-configs/"                         (1)
template_filepath = "include/templates/dag_template.py"          (2)

for filename in os.listdir(config_filepath):                     (3)
    if filename.endswith('.json'):
        config = json.load(open(f"include/data/{filename}"))
        generated_dag = f"dags/generated_dag_{config['dag_id']}.py"
        shutil.copyfile(template_filepath, generated_dag)
        for line in fileinput.input(generated_dag, inplace=True):
            line = line.replace("dag_id_placeholder", config['dag_id'])
            line = line.replace("schedule_placeholder", config['schedule'])
            line = line.replace("input_placeholder", config['input'])
            print(line, end="")
```

1. Specify the path to the config files.
2. Specify the path to the template file.
3. Describe the logic of substituting parameter placeholders with correct values from the configs.

When the script is executed, it will check the *dag-configs* directory for available JSON configuration files and generate a DAG file for each configuration file based on the specified template. In the template, it will match the placeholders with the fields from the configuration files and will substitute the values.

Once the DAG files have been generated, Airflow will parse and load them as regular DAGs.

Dynamic task mapping

The above examples describe how to generate DAGs with different parameters, but the number of tasks for generated DAGs stays the same.

You can create tasks based on the unexpected output of the other task using dynamic task mapping.

This method employs the MapReduce framework and works using the `expand()` method, which creates tasks at runtime based on the output by multiplying the caller task using the provided arguments.

The `expand()` method is used with an operator and accepts a list or a dictionary of values as input. These values can also be an output of the other task in the `XComArg` format.

The example of dynamic task mapping:

```
from airflow.decorators import dag, task
from datetime import datetime

@dag(dag_id = "dynamic_task_dag", schedule="@daily", catchup=False, start_date=datetime(2025,1,1))
def create_tasks():

    @task
    def download_files(file: str):                               (1)
        print(file)

    files = download_files.expand(file=["file_a", "file_b", "file_c"])  (2)

    download_files(file)

create_tasks()
```

1. Create a task function. For example, a function that prints a file.
2. Call `expand()` and pass the list of files as an argument. The DAG will generate three tasks for each file.

The example of dynamic task mapping when the output of the previous task is unexpected:

```
from airflow.decorators import dag, task
from datetime import datetime
import random

@dag(dag_id = "dynamic_task_dag", schedule="@daily",
catchup=False, start_date=datetime(2025,1,1))

def create_tasks():

    @task
    def get_files():
        return ["file_{}" for i in range(random.randint(1,5))]   (1)

    @task
    def download_files(file: str):
        print(file)

    files = download_files.expand(file=get_files())              (2)

create_tasks()
```

1. Create a function that has a random output.
2. Call `expand()` and pass the output of the random function as an argument. The DAG will generate a task for each file.

Use Jinja templates in Airflow

Elena Kosteyucheva

Contents

To Table of Contents

Overview

Jinja in Airflow

Template usage restrictions

- Template search path
- PythonOperators

Macros and filters

Create a DAG using Jinja templates

- Step 1. Prepare the template file
- Step 2. Write the Python code for the DAG
- Step 3. Create the reports directory
- Step 4. Trigger the DAG

Overview

Airflow supports templating, which enables dynamic content generation at runtime using variables, macros, and control structures. This feature enables DAGs and tasks to adapt based on execution context such as dates, task instances, and custom logic.

The templating framework used in Airflow is Jinja. It's a Python templating engine that allows you to embed dynamic expressions and control structures inside text files.

This article focuses on how to use Jinja templates within DAGs and operators in Airflow, diving into core usage patterns, templated fields, macros, and examples.

Jinja in Airflow

Airflow renders specific fields called templated fields using Jinja at runtime. These fields are defined on the operator level. Common operators such as `BashOperator`, `PythonOperator`, `EmailOperator`, and `SqlOperator` have templated fields like `bash_command`, `sql`, and `subject`.

For all available options, see the Airflow template reference.

An example of using Jinja templates:

```
from airflow.operators.bash import BashOperator
from airflow import DAG
from datetime import datetime (1)

with DAG("templating_example", start_date=datetime(2023, 1, 1), schedule_interval="@daily", catchup=False) as dag: (2)
    t1 = BashOperator(
        task_id='print_date',
        bash_command='echo "Execution date is {{ ds }}"', (3)
    )
```

1. Import the dependencies.
2. Initialize a DAG.
3. Add a template in the output string. In this example, `{{ ds }}` is replaced with the execution date at runtime (e.g. `2025-04-22`).

Template usage restrictions

To inject dynamic content into operator parameters, use the `{{ … }}` and `{% … %}` syntax.

Templating is allowed in:

- Variables. For example: `{{ ds }}`, `{{ task_instance }}`, and others.
- Control structures. For example: `{% if … %}`, `{% for … %}`.
- Custom expressions. For example: `{{ macros.ds_add(ds, 7) }}`.

Check the operator documentation for which fields are templated. You can also define custom templated fields by extending an operator.

Template search path

Airflow can load external template files from a directory. This allows you to separate your SQL queries and report templates into different files.

For example:

```
dag = DAG(
    dag_id='template_path_example',
    start_date=datetime(2023, 1, 1),
    schedule_interval='@daily',
    template_searchpath=['/opt/airflow/templates/sql']
)
```

where `template_searchpath` — path to SQL templates.

PythonOperators

Jinja can be combined with Python functions by rendering templates before task execution. You can pass templated strings via `op_kwargs`.

For example:

```
from airflow.operators.python import PythonOperator

def greet_user(execution_date, **kwargs):
    print(f"Hello! Today is {execution_date}")

PythonOperator(
    task_id='greet',
    python_callable=greet_user,
    op_kwargs={"execution_date": "{{ ds }}"},
    dag=dag,
)
```

Airflow will render `"{{ ds }}"` into the actual date before calling `greet_user`.

Macros and filters

Airflow includes built-in macros like `macros.ds_add`, `macros.datetime`, filters like `| ds_format`, `| replace`, and others.

For example:

```
BashOperator(
    task_id='seven_days_later',
    bash_command='echo "In 7 days: {{ macros.ds_add(ds, 7) }}"',
)
```

Custom macros can also be defined in your DAG or loaded via plugins.

Create a DAG using Jinja templates

Below is an example of how to build a DAG that generates a dynamic daily report in Markdown using Jinja templates.

Step 1. Prepare the template file

Create a Jinja template file that will be used to generate reports.

1. Create a folder for templates inside your Airflow environment:

```
$ sudo mkdir -p /opt/airflow/templates
```

2. In the *templates* directory, create a file called *report_template.md* with the following contents:

```
# Daily report

**Date:** {{ ds }}
**Task Status:** {{ task_instance.state }}

## Summary
- DAG ID: {{ dag.dag_id }}
- Task ID: {{ task.task_id }}
```

The values in curly brackets are placeholders, which will be filled dynamically at runtime.

Step 2. Write the Python code for the DAG

Create a new Python file *dynamic_report_dag.py* in your DAGs folder, and add this code:

```
from airflow import DAG (1)
from airflow.operators.python import PythonOperator
from datetime import datetime
from jinja2 import Environment, FileSystemLoader

def render_report(ds, **context): (2)
    env = Environment(loader=FileSystemLoader('/opt/airflow/templates'))
    template = env.get_template('report_template.md')
    report = template.render(
        ds=ds,
        dag=context['dag'],
        task=context['task'],
        task_instance=context['ti']
    )

    output_path = f"/opt/airflow/reports/report_{ds}.md" (3)
    with open(output_path, "w") as f:
        f.write(report)

with DAG(  (4)
    dag_id="dynamic_report_dag",
    start_date=datetime(2023, 1, 1),
    schedule_interval="@daily",
    catchup=False,
    template_searchpath=['/opt/airflow/templates'],
) as dag:

    generate_report = PythonOperator(
        task_id='generate_report',
        python_callable=render_report,
        provide_context=True,
    )
```

1. Import the dependencies.
2. Define the function to render the Jinja template.
3. Save the rendered report.
4. Define the DAG. For more information on how to create a DAG, see the Create a simple DAG article.

Step 3. Create the reports directory

1. Create a directory where the generated reports will be saved.

```
$ sudo mkdir -p /opt/airflow/reports
```

2. Set the necessary access settings:

```
$ sudo chmod -R 755 /opt/airflow/reports/
$ sudo chown -R airflow:airflow /opt/airflow/reports
```

Step 4. Trigger the DAG

1. Go to the Airflow web UI.
2. Find the `dynamic_report_dag` DAG and unpause it.
3. Trigger the DAG.

Once the DAG finishes running, you will see the file in the *reports* directory with the current date in it's name and the similar content:

```
# Daily report

**Date:** 2025-04-28
**Task Status:** running

## Summary
- DAG ID: dynamic_report_dag
- Task ID: generate_report
```

The file contains the execution date, task status, DAG ID, and task ID based on the run context.

Airflow CLI

Reference documentation for Airflow CLI

Contents

To Table of Contents

- Overview
- Miscellaneous commands
- Celery components
- View configuration
- Manage connections
- Manage DAGs
- Database operations
- Manage jobs
- KubernetesExecutor tools
- Manage pools
- Display providers
- Manage roles
- Manage tasks
- Manage users
- Manage variables

Overview

This section includes reference documentation for the Airflow shell CLI.

The usage is as follows:

```
$ airflow COMMAND [GENERIC_OPTIONS] SUBCOMMAND [SUBCOMMAND_OPTIONS]
```

Miscellaneous commands

General purpose Airflow commands.

Command	Description
cheat-sheet	Displays cheat sheet
dag-processor	Starts a standalone DAG Processor instance
info	Shows information about the current Airflow and environment
kerberos	Starts a Kerberos ticket renewer
plugins	Dumps information about the loaded plugins
rotate-fernet-key	Rotates encrypted connection credentials and variables
scheduler	Starts a scheduler instance
standalone	Runs an all-in-one copy of Airflow
sync-perm	Upgrades permissions for existing roles and optionally DAGs
triggerer	Starts a triggerer instance
version	Shows the Airflow version
webserver	Starts an Airflow webserver instance

Celery components

The commands for managing Airflow Celery

Command	Description
flower	Starts a Celery Flower
stop	Stops the Celery worker gracefully
worker	Starts a Celery worker node

View configuration

The commands for viewing Airflow configuration.

Command	Description
get-value	Prints the value of the configuration
list	Prints configuration options

Manage connections

The commands for managing Airflow connections.

Command	Description
add	Adds a connection
webserver	Starts an Airflow webserver instance
delete	Deletes a connection
export	Exports connections
get	Displays a connection
import	Imports connections from a file
list	Lists connections
test	Tests a connection

Manage DAGs

The commands for managing DAGs

Command	Description
backfill	Runs subsections of a DAG for a specified date range
delete	Deletes all DB records related to the specified DAG
details	Displays information about a DAG
list	Lists all DAGs
list-import-errors	Lists all DAGs that have import errors
list-jobs	Lists jobs
list-runs	Lists DAG runs
next-execution	Displays the next DAG execution dates
pause	Pauses a DAG
report	Shows DAG loading report
reserialize	Re-serializes all DAGs by parsing the DAG files
show	Displays DAG tasks with their dependencies
show-dependencies	Displays DAGs with their dependencies
state	Displays the status of a DAG run
test	Executes a single DAG run
trigger	Triggers a DAG run
unpause	Resumes a paused DAG

Database operations

The commands for managing the Airflow database

Command	Description
check	Checks if the database can be reached
check-migrations	Checks if a migration has finished
clean	Purges old records in the metastore tables
downgrade	Downgrades the schema of the metadata database
drop-archived	Drops archived tables created through the `db clean` command
export-archived	Exports archived data from the archive tables
init	Initializes the metadata database
reset	Drops and recreates the metadata database
shell	Runs a shell to access the database
upgrade	Upgrades the metadata database to latest version

Manage jobs

The commands for managing Airflow jobs.

Command	Description
check	Checks if a job is still alive

KubernetesExecutor tools

The commands for managing KubernetesExecutor

Command	Description
cleanup-pods	Clean up Kubernetes pods (created by KubernetesExecutor/KubernetesPodOperator) in `evicted` `failed` `succeeded` `pending` states
generate-dag-yaml	Generates YAML files for all tasks in a DAG. Useful for debugging tasks without launching them in a cluster

Manage pools

The commands for managing Airflow pools

Command	Description
delete	Deletes a pool
export	Exports all pools
get	Displays pool size
import	Imports pools
list	Displays the list of pools
set	Configures a pool

Display providers

The commands for displaying information about providers.

Command	Description
auth	Displays information about API auth backends
behaviours	Displays information about registered connection types with custom behaviour
get	Displays detailed information about a provider
hooks	Lists registered provider hooks
links	Lists extra links registered by the providers
list	Lists installed providers
logging	Displays information about task logging handlers
secrets	Displays information about secret managing backends
widgets	Displays information about registered widgets connections

Manage roles

The commands for managing Airflow roles

Command	Description
add-perms	Adds permissions to a role
create	Creates a role
del-perms	Deletes role permissions
delete	Deletes a role
export	Exports roles (without permissions) from the database to a JSON file
import	Imports roles (without permissions) from a JSON file to the database
list	Lists roles

Manage tasks

The commands for managing Airflow tasks

Command	Description
clear	Clears a set of task instances
failed-deps	Returns the unmet dependencies for a task instance
list	Displays the tasks within a DAG
render	Displays a task instance templates
run	Runs a single task instance
state	Displays the status of a task instance
states-for-dag-run	Displays the status of all task instances in a DAG run
test	Tests a task instance

Manage users

The commands for managing Airflow users.

Command	Description
add-role	Adds a role to a user
create	Creates a user
delete	Deletes a user
export	Exports all users
import	Imports users
list	Lists users
remove-role	Removes a role from a user

Manage variables

The commands for managing Airflow variables.

Command	Description
delete	Deletes a variable
export	Exports all variables
get	Displays a variable
import	Imports variables
list	Lists variables
set	Edits a variable

cheat-sheet

Elena Kostyuchenko

Contents

To Table of Contents

Definition
Usage
└ Arguments
Examples

Definition

Displays a cheat sheet of Airflow CLI commands with descriptions, grouped by command type.

Usage

```
$ airflow cheat-sheet [-h] [-v]
```

Arguments

Parameter	Description
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

Examples

To display the cheat sheet, run:

```
$ airflow cheat-sheet
```

Output example

```
Miscellaneous commands
airflow cheat-sheet                     | Display cheat sheet
airflow dag-processor                   | Start a standalone Dag Processor instance
airflow info                            | Show information about current Airflow and environment
airflow kerberos                        | Start a kerberos ticket renewer
airflow plugins                         | Dump information about loaded plugins
airflow rotate-fernet-key               | Rotate encrypted connection credentials and variables
airflow scheduler                       | Start a scheduler instance
airflow standalone                      | Run an all-in-one copy of Airflow
airflow sync-perm                       | Update permissions for existing roles and optionally DAGs
airflow triggerer                       | Start a triggerer instance
airflow version                         | Show the version
airflow webserver                       | Start a Airflow webserver instance

Celery components
airflow celery flower                   | Start a Celery Flower
airflow celery stop                     | Stop the Celery worker gracefully
airflow celery worker                   | Start a Celery worker node

View configuration
airflow config get-value                | Print the value of the configuration
airflow config list                     | List options for the configuration

Manage connections
airflow connections add                 | Add a connection
airflow connections delete              | Delete a connection
airflow connections export              | Export all connections
airflow connections get                 | Get a connection
airflow connections import              | Import connections from a file
airflow connections list                | List connections
airflow connections test                | Test a connection

Manage DAGs
airflow dags backfill                   | Run subsections of a DAG for a specified date range
airflow dags delete                     | Delete all DB records related to the specified DAG
airflow dags details                    | Get DAG details given a DAG id
airflow dags list                       | List all the DAGs
airflow dags list-import-errors         | List all the DAGs that have import errors
airflow dags list-jobs                  | List the jobs
airflow dags list-runs                  | List DAG runs given a DAG id
airflow dags next-execution             | Get the next execution datetimes of a DAG
airflow dags pause                      | Pause a DAG
airflow dags report                     | Show DagBag loading report
airflow dags reserialize                | Reserialize all DAGs by parsing the DagBag files
airflow dags show                       | Displays DAG's tasks with their dependencies
airflow dags show-dependencies          | Displays DAGs with their dependencies
airflow dags state                      | Get the status of a dag run
airflow dags test                       | Execute one single DagRun
airflow dags trigger                    | Trigger a DAG run
airflow dags unpause                    | Resume a paused DAG

Database operations
airflow db check                        | Check if the database can be reached
airflow db check-migrations             | Check if migration have finished
airflow db clean                        | Purge old records in metastore tables
airflow db downgrade                    | Downgrade the schema of the metadata database.
airflow db drop-archived                | Drop archived tables created through the db clean command
airflow db export-archived              | Export archived data from the archive tables
airflow db init                         | Initialize the metadata database
airflow db reset                        | Burn down and rebuild the metadata database
airflow db shell                        | Runs a shell to access the database
airflow db upgrade                      | Upgrade the metadata database to latest version

Manage jobs
airflow jobs check                      | Checks if job(s) are still alive

Tools to help run the KubernetesExecutor
airflow kubernetes cleanup-pods         | Clean up Kubernetes pods (created by KubernetesExecutor/KubernetesPodOperator) in
                                        | evicted/failed/succeeded/pending states
airflow kubernetes generate-dag-yaml    | Generate YAML files for all tasks in DAG. Useful for debugging tasks without
launching into a cluster

Manage pools
airflow pools delete                    | Delete pool
airflow pools export                    | Export all pools
airflow pools get                       | Get pool size
airflow pools import                    | Import pools
airflow pools list                      | List pools
airflow pools set                       | Configure pool

Display providers
airflow providers auth                  | Get information about API auth backends provided
airflow providers behaviours            | Get information about registered connection types with custom behaviours
airflow providers get                   | Get detailed information about a provider
airflow providers hooks                 | List registered provider hooks
airflow providers links                 | List extra links registered by the providers
airflow providers list                  | List installed providers
airflow providers logging               | Get information about task logging handlers provided
airflow providers secrets               | Get information about secrets backends provided
airflow providers widgets               | Get information about registered connection form widgets

Manage roles
airflow roles add-perms                 | Add roles permissions
airflow roles create                    | Create role
airflow roles del-perms                 | Delete roles permissions
airflow roles delete                    | Delete role
airflow roles export                    | Export roles (without permissions) from db to JSON file
airflow roles import                    | Import roles (without permissions) from JSON file to db
airflow roles list                      | List roles

Manage tasks
airflow tasks clear                     | Clear a set of task instance, as if they never ran
airflow tasks failed-deps               | Returns the unmet dependencies for a task instance
airflow tasks list                      | List the tasks within a DAG
airflow tasks render                    | Render a task instance's template(s)
airflow tasks run                       | Run a single task instance
airflow tasks state                     | Get the status of a task instance
airflow tasks states-for-dag-run        | Get the status of all task instances in a dag run
airflow tasks test                      | Test a task instance

Manage users
airflow users add-role                  | Add role to a user
airflow users create                    | Create a user
airflow users delete                    | Delete a user
airflow users export                    | Export all users
airflow users import                    | Import users
airflow users list                      | List users
airflow users remove-role               | Remove role from a user

Manage variables
airflow variables delete                | Delete variable
airflow variables export                | Export all variables
airflow variables get                   | Get variable
airflow variables import                | Import variables
airflow variables list                  | List variables
airflow variables set                   | Set variable
```

dag-processor

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

└ Arguments

Definition

Starts a standalone DAG Processor instance. You can run this command by using the ***Add/Remove components*** action in ADCM.

Usage

```
$ airflow dag-processor [-h] [-D] [-p] [-l LOG_FILE] [-n NUM_RUNS] [--pid [PID]]
                        [--stderr STDERR] [--stdout STDOUT] [-S SUBDIR] [-v]
```

Arguments

Parameter	Description
-D, --daemon	Run as a daemon instead of running in the foreground
-p, --do-pickle	Attempt to pickle the DAG object to send over to the workers, instead of letting workers run their version of the code
-l, --log-file	Location of the log file
-n, --num-runs	Set the number of runs to execute before exiting. Default: `-1` (do not attempt additional runs)
--pid	PID file location
--stderr	Redirect stderr to the specified file
--stdout	Redirect stdout to the specified file
-S, --subdir	File or directory location for DAGs. Defaults to `[AIRFLOW_HOME]/dags`, where `[AIRFLOW_HOME]` is the value set for `AIRFLOW_HOME` config in *airflow.cfg*
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

info

Elena Kostyuchenko

Contents

To Table of Contents

Definition
Usage
└ Arguments
Examples

Definition

Provides system information about Airflow.

Usage

```
$ airflow info [-h] [-v]
```

Arguments

Parameter	Description
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

Examples

To get system information, run:

```
$ airflow info
```

Output example

```
Apache Airflow
version                | 2.6.3
executor               | CeleryExecutor
task_logging_handler   | airflow.utils.log.file_task_handler.FileTaskHandler
sql_alchemy_conn       | postgresql+psycopg2://airflow:admin@host:5432/airflow
dags_folder            | /opt/airflow/dags
plugins_folder         | /opt/airflow/plugins
base_log_folder        | /var/log/airflow
remote_base_log_folder |


System info
OS              | Linux
architecture    | x86_64
uname           | uname_result(system='Linux', node='host', release='3.10.0-1160.119.1.el7.x86_64', version='#1 SMP Tue
                | Jun 4 14:43:51 UTC 2024', machine='x86_64')
locale          | ('en_US', 'UTF-8')
python_version  | 3.10.4 (main, Feb 24 2025, 09:32:37) [GCC 4.8.5 20150623 (Red Hat 4.8.5-44)]
python_location | /opt/airflow/bin/python3.10


Tools info
git             | NOT AVAILABLE
ssh             | OpenSSH_7.4p1, OpenSSL 1.0.2k-fips  26 Jan 2017
kubectl         | NOT AVAILABLE
gcloud          | NOT AVAILABLE
cloud_sql_proxy | NOT AVAILABLE
mysql           | NOT AVAILABLE
sqlite3         | NOT AVAILABLE
psql            | NOT AVAILABLE


Paths info
airflow_home    | /opt/airflow
system_path     | /sbin:/bin:/usr/sbin:/usr/bin
python_path     | /opt/airflow/bin:/opt/python3.10/lib/python310.zip:/opt/python3.10/lib/python3.10:/opt/python3.
10/lib/python3.10/lib-dynload:/opt/
                | airflow/lib/python3.10/site-packages:/opt/airflow/dags:/opt/airflow/config:/opt/airflow/plugins
airflow_on_path | True


Providers info
apache-airflow-providers-alibaba          | 2.6.0
apache-airflow-providers-amazon           | 8.12.0
apache-airflow-providers-apache-beam      | 5.3.0
apache-airflow-providers-apache-cassandra | 3.3.0
apache-airflow-providers-apache-drill     | 2.5.0
apache-airflow-providers-apache-druid     | 3.6.0
apache-airflow-providers-apache-flink     | 1.2.0
apache-airflow-providers-apache-hdfs      | 4.2.0
apache-airflow-providers-apache-hive      | 6.2.0
apache-airflow-providers-apache-impala    | 1.2.1
apache-airflow-providers-apache-kafka     | 1.2.0
apache-airflow-providers-apache-kylin     | 3.3.0
apache-airflow-providers-apache-livy      | 3.6.0
apache-airflow-providers-apache-spark     | 4.4.0
apache-airflow-providers-apache-sqoop     | 4.1.0
apache-airflow-providers-apprise          | 1.1.0
apache-airflow-providers-arangodb         | 2.3.0
apache-airflow-providers-asana            | 2.3.0
apache-airflow-providers-atlassian-jira   | 2.3.0
apache-airflow-providers-celery           | 3.4.1
apache-airflow-providers-cloudant         | 3.3.0
apache-airflow-providers-cncf-kubernetes  | 7.10.0
apache-airflow-providers-common-sql       | 1.8.1
apache-airflow-providers-dbt-cloud        | 3.4.1
apache-airflow-providers-dingding         | 3.3.0
apache-airflow-providers-discord          | 3.4.1
apache-airflow-providers-docker           | 3.8.2
apache-airflow-providers-elasticsearch    | 5.2.0
apache-airflow-providers-exasol           | 4.3.0
apache-airflow-providers-facebook         | 3.3.0
apache-airflow-providers-ftp              | 3.4.2
apache-airflow-providers-github           | 2.4.0
apache-airflow-providers-google           | 10.12.0
apache-airflow-providers-grpc             | 3.3.0
apache-airflow-providers-hashicorp        | 3.4.1
apache-airflow-providers-http             | 4.4.2
apache-airflow-providers-imap             | 3.2.2
apache-airflow-providers-influxdb         | 2.3.0
apache-airflow-providers-jdbc             | 4.1.0
apache-airflow-providers-jenkins          | 3.4.0
apache-airflow-providers-microsoft-azure  | 8.3.0
apache-airflow-providers-microsoft-mssql  | 3.5.0
apache-airflow-providers-microsoft-psrp   | 2.4.0
apache-airflow-providers-microsoft-winrm  | 3.3.0
apache-airflow-providers-mongo            | 3.4.0
apache-airflow-providers-mysql            | 5.4.0
apache-airflow-providers-odbc             | 4.2.0
apache-airflow-providers-oracle           | 3.8.0
apache-airflow-providers-postgres         | 5.8.0
apache-airflow-providers-redis            | 3.4.1
apache-airflow-providers-salesforce       | 5.5.1
apache-airflow-providers-samba            | 4.3.0
apache-airflow-providers-segment          | 3.3.0
apache-airflow-providers-sendgrid         | 3.3.0
apache-airflow-providers-sftp             | 4.7.0
apache-airflow-providers-singularity      | 3.3.0
apache-airflow-providers-slack            | 8.4.0
apache-airflow-providers-smtp             | 1.4.1
apache-airflow-providers-sqlite           | 3.4.2
apache-airflow-providers-ssh              | 3.8.1
apache-airflow-providers-tableau          | 4.3.0
apache-airflow-providers-tabular          | 1.3.0
apache-airflow-providers-telegram         | 4.2.0
apache-airflow-providers-vertica          | 3.6.0
apache-airflow-providers-yandex           | 3.6.0
```

kerberos

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

Arguments

Examples

Definition

Starts a Kerberos ticket renewer.

Usage

```
$ airflow kerberos [-h] [-D] [-k [KEYTAB]] [-l LOG_FILE] [-o] [--pid [PID]]
                   [--stderr STDERR] [--stdout STDOUT] [-v]
                   [principal]
```

Arguments

Parameter	Description
principal	Kerberos principal
-D, --daemon	Run as a daemon
-k, --keytab	Keytab to use. Default: `airflow.keytab`
-l, --log-file	Location of the log file
-o, --one-time	Run Airflow Kerberos once, not repeatedly
--pid	PID file location
--stderr	Redirect stderr to the specified file
--stdout	Redirect stdout to the specified file
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

Examples

An example command for starting a Kerberos ticket renewer:

```
$ airflow kerberos --keytab custom.keytab airflow
```

plugins

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

└ Arguments

Examples

Definition

Displays information about the loaded plugins or dumps it into a file.

Usage

```
$ airflow plugins [-h] [-o table, json, yaml, plain] [-v]
```

Arguments

Parameter	Description
-o, --output	Output format. Possible values: `table`, `json`, `yaml`, `plain`. Default: `table`
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

Examples

To display information about the plugins, run:

```
$ airflow plugins
```

Output example:

```
name | source                                                                    | macros
=====+===========================================================================+=================================================
==================
hive | apache-airflow-providers-apache-hive==6.2.0: EntryPoint(name='hive',  | airflow.providers.apache.hive.macros.hive.
max_partition,airflow.provi
     | value='airflow.providers.apache.hive.plugins.hive:HivePlugin',       | ders.apache.hive.macros.hive.closest_ds_partition
     | group='airflow.plugins')                                              |
```

rotate-fernet-key

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

Definition

Rotates all encrypted connection credentials and variables. You can run this command by using the ***Rotate fernet key*** action in ADCM.

Usage

```
$ airflow rotate-fernet-key [-h]
```

scheduler

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

└ Arguments

Definition

Starts a scheduler instance. You can run this command by using the ***Add/Remove components*** action in ADCM.

Usage

```
$ airflow scheduler [-h] [-D] [-p] [-l LOG_FILE] [-n NUM_RUNS] [--pid [PID]]
                    [-s] [--stderr STDERR] [--stdout STDOUT] [-S SUBDIR] [-v]
```

Arguments

Parameter	Description
-D, --daemon	Run as daemon instead of running in the foreground
-p, --do-pickle	Attempt to pickle the DAG object to send over to the workers, instead of letting workers run their version of the code
-l, --log-file	Location of the log file
-n, --num-runs	The number of runs to execute before exiting
--pid	PID file location
-s, --skip-serve-logs	Do not start the logs maintenance process along with the workers
--stderr	Redirect stderr to the specified file
--stdout	Redirect stdout to the specified file
-S, --subdir	File or directory location for DAGs. Defaults to `[AIRFLOW_HOME]/dags`, where `[AIRFLOW_HOME]` is the value set for `AIRFLOW_HOME` config in *airflow.cfg*
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

standalone

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

Definition

Runs an all-in-one copy of Airflow.

Usage

```
$ airflow standalone [-h]
```

sync-perm

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

└ Arguments

Definition

Updates permissions for existing roles and optionally DAGs. You can run this command by using the ***Sync permission*** action in ADCM.

Usage

```
$ airflow sync-perm [-h] [--include-dags] [-v]
```

Arguments

Parameter	Description
--include-dags	Sync DAG-specific permissions
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

triggerer

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

└ Arguments

Definition

Starts a triggerer instance. You can run this command by using the ***Add/Remove components*** action in ADCM.

Usage

```
$ airflow triggerer [-h] [--capacity CAPACITY] [-D] [-l LOG_FILE] [--pid [PID]]
                    [-s] [--stderr STDERR] [--stdout STDOUT] [-v]
```

Arguments

Parameter	Description
--capacity	The maximum number of triggers that can be run simultaneously
-D, --daemon	Run as a daemon instead of running in the foreground
-l, --log-file	Location of the log file
--pid	PID file location
-s, --skip-serve-logs	Do not start the logs maintenance process along with the workers
--stderr	Redirect stderr to the specified file
--stdout	Redirect stdout to the specified file
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

version

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

Definition

Displays Airflow version.

Usage

```
$ airflow version [-h]
```

webserver

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

└ Arguments

Examples

Definition

Starts an Airflow webserver instance. You can run this command by using the ***Add/Remove components*** action in ADCM.

Usage

```
$ airflow webserver [-h] [-A ACCESS_LOGFILE] [-L ACCESS_LOGFORMAT] [-D] [-d]
                    [-E ERROR_LOGFILE] [-H HOSTNAME] [-l LOG_FILE] [--pid [PID]]
                    [-p PORT] [--ssl-cert SSL_CERT] [--ssl-key SSL_KEY]
                    [--stderr STDERR] [--stdout STDOUT] [-t WORKER_TIMEOUT]
                    [-k {sync,eventlet,gevent,tornado}] [-w WORKERS]
```

Arguments

Parameter	Description
-A, --access-logfile	The logfile to store the webserver access log. Use `-` to print to stdout
-L, --access-logformat	The access log format for gunicorn logs
-D, --daemon	Run as a daemon instead of running in the foreground
-d, --debug	Use the server that is shipped with Flask in debug mode
-E, --error-logfile	The logfile to store the webserver error log. Use `-` to print to stderr
-H, --hostname	Hostname on which to run the webserver
-l, --log-file	Location of the log file
--pid	PID file location
-p, --port	Port on which to run the server
--ssl-cert	Path to the SSL certificate for the webserver
--ssl-key	Path to the key to use with the SSL certificate
--stderr	Redirect stderr to the specified file
--stdout	Redirect stdout to the specified file
-t, --worker-timeout	The timeout for waiting on webserver workers
-k, --workerclass	The worker class to use for Gunicorn. Possible values: `sync`, `eventlet`, `gevent`, `tornado`. Default: `sync`
-w, --workers	Number of workers to run the webserver on

Examples

Example command for starting a webserver:

```
$ airflow webserver -p 8080 -H 0.0.0.0 -w 4 -D -A /logs/access.log -E /logs/error.log --ssl-cert /etc/ssl/certs/airflow.crt --ssl-key /etc/ssl/private/airflow.key
```

flower

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

└ Arguments

Definition

Starts a Celery Flower. You can run this command by using the ***Add/Remove components*** action in ADCM.

Usage

```
$ airflow celery flower [-h] [-A BASIC_AUTH] [-a BROKER_API] [-D]
                        [-c FLOWER_CONF] [-H HOSTNAME] [-l LOG_FILE]
                        [--pid [PID]] [-p PORT] [--stderr STDERR]
                        [--stdout STDOUT] [-u URL_PREFIX] [-v]
```

Arguments

Parameter	Description
-A, --basic-auth	Secure Flower with basic authentication. Accepts `user:password` pairs separated by a comma. For example: `flower_basic_auth = user1:password1,user2:password2`
-a, --broker-api	Broker API
-D, --daemon	Run as a daemon instead of running in the foreground
-c, --flower-conf	Configuration file for Flower
-H, --hostname	Set the name of the host on which to run the server
-l, --log-file	Location of the log file
--pid	PID file location
-p, --port	The port on which to run the server. Default: `5555`
--stderr	Redirect stderr to this file
--stdout	Redirect stdout to this file
-u, --url-prefix	URL prefix for Flower
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

stop

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

└ Arguments

Definition

Gracefully stops the Celery worker.

Usage

```
$ airflow celery stop
                    [-h]
                    [--pid [PID]]
                    [-v]
```

Arguments

Parameter	Description
--pid	PID file location
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

worker

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

└ Arguments

Definition

Starts a Celery worker node. You can run this command by using the ***Add/Remove components*** action in ADCM.

Usage

```
$ airflow celery worker [-h] [-a AUTOSCALE] [-H CELERY_HOSTNAME]
                        [-c CONCURRENCY] [-D] [-l LOG_FILE] [--pid [PID]]
                        [-q QUEUES] [-s] [--stderr STDERR] [--stdout STDOUT]
                        [-u UMASK] [-v] [--without-gossip] [--without-mingle]
```

Arguments

Parameter	Description
-a, --autoscale	Minimum and maximum number of workers to autoscale
-H, --celery-hostname	Hostname of Celery worker if you have multiple workers on a single machine
-c, --concurrency	The number of worker processes. Default: `16`
-D, --daemon	Run as a daemon instead of running in the foreground. Default: `False`
-l, --log-file	Location of the log file
--pid	PID file location
-q, --queues	Comma delimited list of queues to serve. Default: `default`
-s, --skip-serve-logs	Do not start the serve logs process along with the workers. Default: `False`
--stderr	Redirect stderr to this file
--stdout	Redirect stdout to this file
-u, --umask	Set the umask of Celery worker in daemon mode
--without-gossip	Do not subscribe to other workers events. Default: `False`
--without-mingle	Do not synchronize with other workers at start-up. Default: `False`
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

get-value

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

Arguments

Examples

Definition

Prints the value of the configuration parameter.

Usage

```
$ airflow config get-value [-h] [-v] section option
```

Arguments

Parameter	Description
section	Section name in the configuration
option	Option name
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

Examples

To get the value of the `dags_folder` parameter in the core settings, run:

```
$ airflow config get-value core dags_folder
```

Output example:

```
/opt/airflow/dags
```

list

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

Arguments

Examples

Definition

Lists options for the configuration.

Usage

```
$ airflow config list [-h] [--color {off,auto,on}] [-c] [-a] [-p] [-d] [-V] [-e]
                      [-s] [--section SECTION] [-v]
```

Arguments

Parameter	Description
--color	Use colored output. Possible values: `off`, `auto`, `on`. Default: `auto`
-c, --comment-out-everything	Comment out all configuration options. Useful as starting point for new installation
-a, --defaults	Show only defaults, do not show local configuration, sources, descriptions of includes, examples, variables
-p, --exclude-providers	Exclude provider configuration (included by default)
-d, --include-descriptions	Show descriptions for the configuration variables
-V, --include-env-vars	Show the environment variable for each option
-e, --include-examples	Show examples for the configuration variables
-s, --include-sources	Show the source of the configuration variables
--section	The section name
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

Examples

To print out the core configuration, run:

```
$ airflow config list --section core
```

Output example:

```
[core]
dags_folder = /opt/airflow/dags
hostname_callable = airflow.utils.net.getfqdn
might_contain_dag_callable = airflow.utils.file.might_contain_dag_via_default_heuristic
default_timezone = utc
executor = CeleryExecutor
parallelism = 32
max_active_tasks_per_dag = 16
dags_are_paused_at_creation = True
max_active_runs_per_dag = 16
load_examples = True
plugins_folder = /opt/airflow/plugins
execute_tasks_new_python_interpreter = False
fernet_key = 9smzpoHUQlm4eu_E2vufmVUWgm3A6tClkEuhYu48aC8=
donot_pickle = True
dagbag_import_timeout = 30
dagbag_import_error_tracebacks = True
dagbag_import_error_traceback_depth = 2
dag_file_processor_timeout = 50
task_runner = StandardTaskRunner
default_impersonation =
security =
unit_test_mode = False
enable_xcom_pickling = False
allowed_deserialization_classes = airflow\..*
killed_task_cleanup_time = 60
dag_run_conf_overrides_params = True
dag_discovery_safe_mode = True
dag_ignore_file_syntax = regexp
default_task_retries = 0
default_task_retry_delay = 300
max_task_retry_delay = 86400
default_task_weight_rule = downstream
default_task_execution_timeout =
min_serialized_dag_update_interval = 30
compress_serialized_dags = False
min_serialized_dag_fetch_interval = 10
max_num_rendered_ti_fields_per_task = 30
check_slas = True
xcom_backend = airflow.models.xcom.BaseXCom
lazy_load_plugins = True
lazy_discover_providers = True
hide_sensitive_var_conn_fields = True
sensitive_var_conn_names =
default_pool_task_slot_count = 128
max_map_length = 1024
daemon_umask = 0o077
database_access_isolation = False
fernet_key_secret = fernet_key_value
```

add

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

Arguments

Examples

Definition

Adds a connection.

Usage

```
$ airflow connections add [-h] [--conn-description CONN_DESCRIPTION]
                          [--conn-extra CONN_EXTRA] [--conn-host CONN_HOST]
                          [--conn-json CONN_JSON] [--conn-login CONN_LOGIN]
                          [--conn-password CONN_PASSWORD]
                          [--conn-port CONN_PORT] [--conn-schema CONN_SCHEMA]
                          [--conn-type CONN_TYPE] [--conn-uri CONN_URI]
                          conn_id
```

Arguments

Parameter	Description
conn_id	Connection ID required to add a connection
--conn-description	Connection description (optional)
--conn-extra	Connection extra field (optional)
--conn-host	Connection host (optional)
--conn-json	Connection JSON. Required when adding a connection using JSON representation
--conn-login	Connection login (optional)
--conn-password	Connection password (optional)
--conn-port	Connection port (optional)
--conn-schema	Connection schema (optional)
--conn-type	Connection type, required when adding a connection without `conn_uri`
--conn-uri	Connection URI. Required when adding a connection without `conn_type`

Examples

To add a connection, run:

```
$ airflow connections add --conn-type http test_connection
```

Output example:

```
Successfully added `conn_id`=test_connection
```

delete

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

Arguments

Examples

Definition

Deletes a connection.

Usage

```
$ airflow connections delete [-h] [--color {off,auto,on}] [-v] conn_id
```

Arguments

Parameter	Description
conn_id	Connection ID required to delete a connection
--color	Use colored output. Possible values: `off`, `auto`, `on`. Default: `auto`
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

Examples

To delete the specified connection, run:

```
$ airflow connections delete test_connection
```

Output example:

```
Successfully deleted connection with `conn_id`=test_connection
```

export

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

Arguments

Examples

Definition

Exports all connections to stdout or a file.

Usage

```
$ airflow connections export [-h] [--file-format {json,yaml,env}]
                             [--format {json,yaml,env}]
                             [--serialization-format {json,uri}] [-v]
                             file
```

Arguments

Parameter	Description
file	Output file path for exporting connections
--file-format	File format for the export. Possible values: `json`, `yaml`, `env`
--format	File format to use for the export. Deprecated. Use `--file-format` instead
--serialization-format	When exporting the ENV format, defines how connections should be serialized. Possible values: `json`, `uri` (default)
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

Examples

The following command will export the connections into JSON format:

```
$ airflow connections export /tmp/connections.json
```

Output example:

```
Connections successfully exported to /tmp/connections.json.
```

get

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

Arguments

Examples

Definition

Displays information about a connection.

Usage

```
$ airflow connections get [-h] [--color {off,auto,on}]
                          [-o table, json, yaml, plain] [-v]
                          conn_id
```

Arguments

Parameter	Description
conn_id	Connection ID needed to display connection information
--color	Use colored output. Possible values: `off`, `auto`, `on`. Default: `auto`
-o, --output	Output format. Possible values: `table`, `json`, `yaml`, `plain`. Default: `table`
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

Examples

To display information about a connection, run:

```
$ airflow connections get ftp_default
```

Output example:

```
  |           |          |             |          |        |         |          |      | is_encrypte | is_extra_en |
extra_dejso |
id | conn_id     | conn_type | description | host      | schema | login   | password | port | d           | crypted     | n
| get_uri
===+=============+===========+=============+===========+========+=========+==========+======+=============+=============+========
=====+=============
18 | ftp_default | ftp       | None        | localhost | None   | airflow | airflow  | 21   | True        | True        |
{'key_file' | ftp://airflo
  |             |           |             |           |        |         |          |      |             |             | :
| w:airflow@lo
  |             |           |             |           |        |         |          |      |             |             | '~/.
ssh/id_ | calhost:21/?
  |             |           |             |           |        |         |          |      |             |             | rsa',
| __extra__=%7
  |             |           |             |           |        |         |          |      |             |             |
'no_host_ke | B%22key_file
  |             |           |             |           |        |         |          |      |             |             |
y_check':   | %22%3A+%22~%
  |             |           |             |           |        |         |          |      |             |             | True}
| 2F.ssh%2Fid_
  |             |           |             |           |        |         |          |      |             |             |
| rsa%22%2C+%2
  |             |           |             |           |        |         |          |      |             |             |
| 2no_host_key
  |             |           |             |           |        |         |          |      |             |             |
| _check%22%3A
  |             |           |             |           |        |         |          |      |             |             |
| +true%7D
```

import

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

- Arguments

Examples

Definition

Import connections to Airflow from a file. This file can be the output of the Airflow export command.

The file type specified in the command must be JSON, YAML, or ENV, and will be automatically detected.

Usage

```
$ airflow connections import [-h] [--overwrite] [-v] file
```

Arguments

Parameter	Description
file	Path to a file with the connections to import
--overwrite	Overwrite existing entries if a conflict occurs
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

Examples

To import connections, which will overwrite the existing ones, from a file, run:

```
$ airflow connections import --overwrite /tmp/connections.json
```

Output example:

```
Imported connection airflow_db
Imported connection aws_default
Imported connection azure_batch_default
Imported connection azure_cosmos_default
Imported connection azure_data_explorer_default
Imported connection azure_data_lake_default
Imported connection azure_default
Imported connection cassandra_default
Imported connection databricks_default
Imported connection dingding_default
Imported connection drill_default
Imported connection druid_broker_default
Imported connection druid_ingest_default
Imported connection elasticsearch_default
Imported connection emr_default
Imported connection facebook_default
Imported connection fs_default
Imported connection ftp_default
Imported connection google_cloud_default
Imported connection hive_cli_default
Imported connection hiveserver2_default
Imported connection http_default
Imported connection impala_default
Imported connection kubernetes_default
Imported connection kylin_default
Imported connection leveldb_default
Imported connection livy_default
Imported connection local_mysql
Imported connection metastore_default
Imported connection mongo_default
Imported connection mssql_default
Imported connection mysql_default
Imported connection opsgenie_default
Imported connection oracle_default
Imported connection oss_default
Imported connection pig_cli_default
Imported connection pinot_admin_default
Imported connection pinot_broker_default
Imported connection postgres_default
Imported connection presto_default
Imported connection qubole_default
Imported connection redis_default
Imported connection redshift_default
Imported connection segment_default
Imported connection sftp_default
Imported connection spark_default
Imported connection sqlite_default
Imported connection sqoop_default
Imported connection ssh_default
Imported connection tableau_default
Imported connection tabular_default
Imported connection test_connection
Imported connection trino_default
Imported connection vertica_default
Imported connection wasb_default
Imported connection webhdfs_default
Imported connection yandexcloud_default
```

list

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

└ Arguments

Examples

Definition

Displays information about connections.

Usage

```
$ airflow connections list [-h] [--conn-id CONN_ID]
                           [-o table, json, yaml, plain] [-v]
```

Arguments

Parameter	Description
--conn-id	If passed, only items with the specified connection ID will be displayed
-o, --output	Output format. Possible values: `table` (default), `json`, `yaml`, `plain`. Default: `table`
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

Examples

To display information about the specified connection, run:

```
$ airflow connections list --conn-id ssh_default
```

Output example:

```
   |           |          |             |          |       |      |          |      |              | is_extra_en |
|
id | conn_id     | conn_type | description | host      | schema | login | password | port | is_encrypted | crypted     |
extra_dejson | get_uri
===+=============+===========+=============+===========+========+=======+==========+======+==============+=============+=========
=====+=============
50 | ssh_default | ssh       | None        | localhost | None   | None  | None     | None | False        | False       | {}
| ssh://localh
   |             |           |             |           |        |       |          |      |              |             |
| ost
```

test

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

Arguments

Examples

Definition

Tests a connection.

Usage

```
$ airflow connections test [-h] [-v] conn_id
```

Arguments

Parameter	Description
conn_id	ID of the connection to test
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

Examples

To test the specified connection, run:

```
$ airflow connections test --conn-id ssh_default
```

backfill

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

- Arguments

Examples

Definition

Backfill is a sub-command in the `dag` command group. It allows you to run subsections of a DAG for a specified date range.

If `reset_dag_run` option is used, the command will first prompt whether Airflow should clear all the previous DAG runs and task instances within the provided date range.

If `rerun_failed_tasks` is used, Airflow will re-run the previous failed task instances within the provided date range.

Usage

```
$ airflow dags backfill [-h] [-c CONF] [--continue-on-failures]
                        [--delay-on-limit DELAY_ON_LIMIT] [--disable-retry] [-x]
                        [-n] [-e END_DATE] [-i] [-I] [-l] [-m] [--pool POOL]
                        [--rerun-failed-tasks] [--reset-dagruns] [-B]
                        [-s START_DATE] [-S SUBDIR] [-t TASK_REGEX]
                        [--treat-dag-as-regex] [--treat-dag-id-as-regex] [-v]
                        [-y]
                        dag_id
```

Arguments

Parameter	Description
dag_id	The ID of the DAG to backfill
-s, --start-date	The start date of the backfill period in `YYYY-MM-DD` format
-e, --end-date	The end date of the backfill period in `YYYY-MM-DD` format
-m, --mark-success	Mark jobs as succeeded without running them
-i, --ignore-dependencies	Skip upstream tasks. Run only the tasks matching the regex
-t, --task-regex	Regex pattern to filter specific task IDs to backfill
-I, --ignore-first-depends-on-past	Ignore `depends_on_past` dependencies for the first set of tasks only
-l, --local	Run the task using the LocalExecutor
-x, --donot-pickle	Do not pickle the DAG object to send to workers. Workers will run their version of the code
-n, --dry-run	Perform a dry run
--pool	Resource pool to use
--rerun-failed-tasks	Re-run previously failed task instances within the date range
--reset-dagruns	Delete existing backfill-related DAG runs and start fresh
-B, --run-backwards	Run tasks from the most recent date first. Incompatible with `depends_on_past`
--delay-on-limit	Time (in seconds) to wait before retrying when max active DAG runs limit is reached
--disable-retry	Mark failed tasks as failed without retrying
-c, --conf	JSON string to be stored in the DAG run's `conf` attribute
-y	Skip confirmation prompt when resetting DAG runs
-S, --subdir	Path to the DAG files directory. Defaults to `[AIRFLOW_HOME]/dags`
--treat-dag-as-regex	Interpret `dag_id argument` as a regex pattern to match multiple DAGs
--treat-dag-id-as-regex	Alias for `--treat-dag-as-regex`
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

Examples

To run a single selected task from the specified DAG, run:

```
$ airflow dags backfill example_complex -s 2025-04-03 -e 2025-04-04 -t get_entry_group
```

delete

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

- Arguments

Examples

Definition

Deletes all database records related to the specified DAG.

Usage

```
$ airflow dags delete [-h] [-v] [-y] dag_id
```

Arguments

Parameter	Description
dag_id	The ID of the DAG to delete from the database
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose
-y, --yes	Do not prompt for confirmation

Examples

To delete information about the specified DAG, run:

```
$ airflow dags delete test_dag
```

Output example:

```
[2025-04-03T14:38:17.121+0000] {__init__.py:42} INFO - Loaded API auth backend: airflow.api.auth.backend.session
[2025-04-03T14:38:17.123+0000] {__init__.py:42} INFO - Loaded API auth backend: airflow.api.auth.backend.basic_auth
This will drop all existing records related to the specified DAG. Proceed? (y/n)
[2025-04-03T14:38:44.671+0000] {delete_dag.py:49} INFO - Deleting DAG: test_dag
Removed 1 record(s)
```

details

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

Arguments

Examples

Definition

Shows details about the specified DAG.

Usage

```
$ airflow dags details [-h] [-o table, json, yaml, plain] [-v] dag_id
```

Arguments

Parameter	Description
dag_id	The ID of the DAG
-o, --output	Output format. Possible values: `table` (default), `json`, `yaml`, `plain`. Default: `table`
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

Examples

To display information about the specified DAG, run:

```
$ airflow dags details tutorial
```

Output example:

```
property_name                  | property_value
===============================+================================================================================
==================
schedule_interval              | {'__type': 'TimeDelta', 'days': 1, 'seconds': 0, 'microseconds': 0}
last_expired                   | None
scheduler_lock                 | None
has_task_concurrency_limits    | False
next_dagrun                    | 2025-04-13T14:44:10.740990+00:00
next_dagrun_create_after       | 2025-04-14T14:44:10.740990+00:00
next_dagrun_data_interval_start | 2025-04-13T14:44:10.740990+00:00
pickle_id                      | None
file_token                     | .
eJw9ykESQDAMAMC_9K5hPMgEURkhmSaG_t7NnjeBWgBy3UQfEJ7BWux6jXnowTmoM1wOLOR_ohdPE5pWLA5xh1ZGydbSByZCHCk.Cm0xzMLrV1KcD
                               | KI1UB5RJx-4EOs
has_import_errors              | False
is_paused                      | True
next_dagrun_data_interval_end  | 2025-04-14T14:44:10.740990+00:00
max_active_tasks               | 16
dag_id                         | tutorial
is_subdag                      | False
is_active                      | True
timetable_description          |
description                    | A simple tutorial DAG
fileloc                        | /opt/airflow/lib/python3.10/site-packages/airflow/example_dags/tutorial.py
last_pickled                   | None
owners                         | airflow
root_dag_id                    | None
default_view                   | grid
max_active_runs                | 16
tags                           | {'name': 'example'}
last_parsed_time               | 2025-04-14T14:44:10.740850+00:00
```

list

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

Arguments

Examples

Definition

Lists all the DAGs.

Usage

```
$ airflow dags list [-h] [-o table, json, yaml, plain]
                    [-S SUBDIR] [-v]
```

Arguments

Parameter	Description
-S, --subdir	File or directory location for DAGs. Defaults to `[AIRFLOW_HOME]/dags`, where `[AIRFLOW_HOME]` is the value set for `AIRFLOW_HOME` config in *airflow.cfg*
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose
-y, --yes	Do not prompt for confirmation

Examples

To get the list of DAGs, run:

```
$ airflow dags list
```

Output example:

```
dag_id                                                 | filepath
| owner   | paused
=======================================================+=================================================================================
=+=========+=======
adcm_check                                             | adcm_check.py
| airflow | True
dataset_consumes_1                                     | /opt/airflow/lib/python3.10/site-packages/airflow/example_dags/example_
| airflow | True
                                                       | datasets.py
|         |
dataset_consumes_1_and_2                               | /opt/airflow/lib/python3.10/site-packages/airflow/example_dags/example_
| airflow | True
                                                       | datasets.py
|         |
dataset_consumes_1_never_scheduled                     | /opt/airflow/lib/python3.10/site-packages/airflow/example_dags/example_
| airflow | True
                                                       | datasets.py
|         |
dataset_consumes_unknown_never_scheduled               | /opt/airflow/lib/python3.10/site-packages/airflow/example_dags/example_
| airflow | True
```

list-import-errors

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

└ Arguments

Definition

Lists all the DAGs that have import errors.

Usage

```
$ airflow dags list-import-errors [-h] [-o table, json, yaml, plain] [-S SUBDIR]
                                  [-v]
```

Arguments

Parameter	Description
-S, --subdir	File or directory location for DAGs. Defaults to `[AIRFLOW_HOME]/dags`, where `[AIRFLOW_HOME]` is the value set for `AIRFLOW_HOME` config in *airflow.cfg*
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose
-y, --yes	Do not prompt for confirmation

list-jobs

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

Arguments

Examples

Definition

Displays information about jobs.

Usage

```
$ airflow dags list-jobs [-h] [-d DAG_ID] [--limit LIMIT]
                         [-o table, json, yaml, plain]
                         [--state running, success, restarting, failed] [-v]
```

Arguments

Parameter	Description
-d, --dag-id	The ID of the DAG
--limit	Return a limited number of records
--state	Only list the jobs with a specific state. Possible values: `running`, `success`, `restarting`, `failed`
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose
-y, --yes	Do not prompt for confirmation

Examples

To display the failed jobs, run:

```
$ airflow dags list-jobs --state failed
```

Output example:

```
dag_id | state  | job_type     | start_date                        | end_date
=======+========+==============+===================================+=========
None   | failed | SchedulerJob | 2025-04-13 06:05:32.765139+00:00 | None
None   | failed | SchedulerJob | 2025-04-04 07:04:45.232394+00:00 | None
None   | failed | SchedulerJob | 2025-03-20 08:14:14.726727+00:00 | None
None   | failed | SchedulerJob | 2025-03-18 08:42:55.290413+00:00 | None
None   | failed | SchedulerJob | 2025-03-12 12:26:23.617197+00:00 | None
None   | failed | SchedulerJob | 2025-03-10 08:30:22.448855+00:00 | None
None   | failed | SchedulerJob | 2025-03-06 10:55:55.349423+00:00 | None
```

list-runs

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

Arguments

Examples

Definition

Displays information about DAG runs for the specified DAG ID.

Usage

```
$ airflow dags list-runs [-h] -d DAG_ID [-e END_DATE] [--no-backfill]
                         [-o table, json, yaml, plain] [-s START_DATE]
                         [--state queued, running, success, failed] [-v]
```

Arguments

Parameter	Description
-d, --dag-id	The ID of the DAG
-s, --start-date	If `start_date` is provided, all the DAG runs that were executed before this date will be filtered out
-e, --end-date	If `end_date` is provided, all the DAG runs that were executed after this date will be filtered out
--no-backfill	If `no_backfill` option is used, all backfill DAG runs for given DAG ID will be filtered out
--state	If state option is used, only the DAG runs with the given state will be searched for. Possible values: `queued`, `running`, `success`, `failed`
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose
-y, --yes	Do not prompt for confirmation

Examples

The following command displays information about the running DAGs with the `example_complex` ID:

```
$ airflow dags list-runs -d example_complex --state running
```

Output example:

```
dag_id          | run_id                         | state   | execution_date               | start_date                   | end_date
================+================================+=========+==============================+==============================+===========
====================
example_complex | manual__2025-04-14T15:29:17 | success | 2025-04-14T15:29:17.298007+0 | 2025-04-14T15:29:17.556324+ | 2025-04-
14T15:29:43.118988+00
                | .298007+00:00               |         | 0:00                         | 00:00                        | :00
example_complex | manual__2025-04-14T14:33:55 | success | 2025-04-14T14:33:55.429076+0 | 2025-04-14T14:33:55.901567+ | 2025-04-
14T14:34:23.912882+00
                | .429076+00:00               |         | 0:00                         | 00:00                        | :00
```

next-execution

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

- Arguments

Examples

Definition

Gets time and date of the next DAG execution. Returns one execution unless the `--num-executions` option is provided.

Usage

```
$ airflow dags next-execution [-h] [-n NUM_EXECUTIONS] [-S SUBDIR] [-v] dag_id
```

Arguments

Parameter	Description
dag_id	The ID of the DAG
-n, --num-executions	The number of next execution dates to show. Default: `1`
-S, --subdir	File or directory location of the DAG. Defaults to `[AIRFLOW_HOME]/dags`, where `[AIRFLOW_HOME]` is the value set for `AIRFLOW_HOME` in *airflow.cfg*
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

Examples

To get the date of the next DAG execution, run:

```
$ airflow dags next-execution adcm_check
```

Output example:

```
[INFO] Please be reminded this DAG is PAUSED now.
2025-04-04T04:37:12.246784+00:00
```

pause

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

Arguments

Examples

Definition

Pauses one or more DAGs. This command allows you to halt the execution of specified DAGs, disabling further task scheduling.

Use the `-treat-dag-id-as-regex` option to target multiple DAGs by treating the `-dag-id` as a regex.

Usage

```
$ airflow dags pause [-h] [-o table, json, yaml, plain] [-S SUBDIR]
                     [--treat-dag-id-as-regex] [-v] [-y]
                     dag_id
```

Arguments

Parameter	Description
dag_id	The ID of the DAG
-S, --subdir	File or directory location for DAGs. Defaults to `[AIRFLOW_HOME]/dags`, where `[AIRFLOW_HOME]` is the value set for `AIRFLOW_HOME` config in *airflow.cfg*
--treat-dag-id-as-regex	If set, `dag_id` will be treated as regex instead of a string
-o, --output	Output format. Possible values: `table`, `json`, `yaml`, `plain`. Default: `table`
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose
-y, --yes	Do not prompt for confirmation

Examples

To pause a DAG, run:

```
$ airflow dags pause tutorial
```

Output example:

```
Dag: tutorial, paused: True
```

report

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

Arguments

Examples

Definition

Shows the DAG folder loading report.

Usage

```
$ airflow dags report [-h] [-o table, json, yaml, plain] [-S SUBDIR] [-v]
```

Arguments

Parameter	Description
-S, --subdir	File or directory location for DAGs. Defaults to `[AIRFLOW_HOME]/dags`, where `[AIRFLOW_HOME]` is the value set for `AIRFLOW_HOME` config in *airflow.cfg*
-o, --output	Output format. Possible values: `table`, `json`, `yaml`, `plain`. Default: `table`
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

Examples

To display the DAG loading report, run:

```
$ airflow dags report
```

Output example:

```
=================================================+===============+=========+==========+=================================
==================
/opt/airflow/lib/python3.10/site-packages/airflow/exa | 0:00:00.424125 | 0       | 0        |
mple_dags/plugins/workday.py                          |               |         |          |
/opt/airflow/lib/python3.10/site-packages/airflow/exa | 0:00:00.201374 | 1       | 7        | example_kubernetes_executor
mple_dags/example_kubernetes_executor.py              |               |         |          |
/opt/airflow/lib/python3.10/site-packages/airflow/exa | 0:00:00.031733 | 1       | 6        | example_xcom
mple_dags/example_xcom.py                             |               |         |          |
/opt/airflow/lib/python3.10/site-packages/airflow/exa | 0:00:00.013801 | 1       | 1        | example_params_ui_tutorial
mple_dags/example_params_ui_tutorial.py               |               |         |          |
```

reserialize

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

└ Arguments

Definition

Drops all serialized DAGs from the metadata DB. This will cause all DAGs to be re-serialized from the DAG folder.

This can be helpful if serialized DAGs get out of sync with the version of Airflow that you are running.

Usage

```
$ airflow dags reserialize [-h] [--clear-only] [-S SUBDIR] [-v]
```

Arguments

Parameter	Description
--clear-only	If passed, serialized DAGs will be cleared but not re-serialized
-S, --subdir	File or directory location for DAGs. Defaults to `[AIRFLOW_HOME]/dags`, where `[AIRFLOW_HOME]` is the value set for `AIRFLOW_HOME` config in *airflow.cfg*
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

show

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

- Arguments

Examples

Definition

Generates a visual representation of tasks and dependencies in a DAG.

Usage

```
$ airflow dags show [-h] [--imgcat] [-s SAVE] [-S SUBDIR] [-v]
```

Arguments

Parameter	Description
--imgcat	Displays a graph using the imgcat tool. This option only works in iTerm
-s, --save	Saves the result to the specified file. The file format is determined by the file extension. For more information about supported format, see Output Formats
-S, --subdir	File or directory location for DAGs. Defaults to `[AIRFLOW_HOME]/dags`, where `[AIRFLOW_HOME]` is the value set for `AIRFLOW_HOME` config in *airflow.cfg*
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

Examples

To create a PNG file with the DAG information graph, execute the following command:

```
$ airflow dags show --save output.png
```

show-dependencies

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

Arguments

Examples

Definition

Generates a visual representation of dependencies.

Usage

```
$ airflow dags show-dependencies [-h] [--imgcat] [-s SAVE] [-S SUBDIR] [-v]
```

Arguments

Parameter	Description
--imgcat	Displays a graph using the imgcat tool. This option only works in iTerm
-s, --save	Saves the result to the specified file. The file format is determined by the file extension. For more information about supported format, see Output Formats
-S, --subdir	File or directory location for DAGs. Defaults to `[AIRFLOW_HOME]/dags`, where `[AIRFLOW_HOME]` is the value set for `AIRFLOW_HOME` config in *airflow.cfg*
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

Examples

To display dependencies, run:

```
$ airflow dags show-dependencies
```

Output example:

```
digraph {
        graph [rankdir=LR]
        subgraph dataset_produces_2 {
                graph [label=dataset_produces_2 labelloc=t rankdir=LR]
                dataset_produces_2 -> s3://dag2/output_1.txt
                s3://dag2/output_1.txt -> dataset
        }
}
```

state

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

└ Arguments

Examples

Definition

Shows the status of a DAG run.

Usage

```
$ airflow dags state [-h] [-S SUBDIR] [-v] dag_id execution_date
```

Arguments

Parameter	Description
dag_id	The ID of the DAG
-S, --subdir	File or directory location for DAGs. Defaults to `[AIRFLOW_HOME]/dags`, where `[AIRFLOW_HOME]` is the value set for `AIRFLOW_HOME` config in *airflow.cfg*
execution_date	Date of the DAG execution
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

Examples

To get the status of a DAG run, execute the following:

```
$ airflow dags state tutorial 04.04.2025
```

test

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

└ Arguments

Definition

Executes one single DAG run for a given DAG and execution date.

Usage

```
$ airflow dags test [-h] [-c CONF] [--imgcat-dagrun]
[--mark-success-pattern MARK_SUCCESS_PATTERN]
[--save-dagrun SAVE_DAGRUN] [--show-dagrun] [-S SUBDIR]
[--use-executor] [-v]
dag_id [execution_date]
```

Arguments

Parameter	Description
dag_id	The ID of the DAG
execution_date	The DAG execution date (optional)
-S, --subdir	File or directory location for DAGs. Defaults to `[AIRFLOW_HOME]/dags`, where `[AIRFLOW_HOME]` is the value set for `AIRFLOW_HOME` config in *airflow.cfg*
-c, --conf	JSON string that gets pickled into the DAG run `conf` attribute
--imgcat-dagrun	After completing the DAG run, prints a diagram on the screen for the current DAG run using the imgcat tool. This option only works in iTerm
--mark-success-pattern	Do not run `task_ids` matching the regex `<MARK_SUCCESS_PATTERN>`, mark them as successful instead. Can be used to skip dependency check sensors or cleanup steps in local testing
--save-dagrun	After completing the backfill, saves the diagram for current DAG run to the indicated file
--show-dagrun	After completing the backfill, shows the diagram for the current DAG run. The diagram is in the DOT format
--use-executor	Use an executor to test the DAG. By default the DAG is run without an executor. If set, the executor configured in the environment is used
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

trigger

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

└ Arguments

Examples

Definition

Triggers a new DAG run. If the DAG is paused, then DAG run state will remain queued and the task will not run.

Usage

```
$ airflow dags trigger [-h] [-c CONF] [-e EXEC_DATE] [--no-replace-microseconds]
                       [-o table, json, yaml, plain] [-r RUN_ID] [-S SUBDIR]
                       [-v]
                       dag_id
```

Arguments

Parameter	Description
dag_id	The ID of the DAG
-c, --conf	JSON string that gets pickled into the DAG run `conf` attribute
-e, --exec-date	DAG execution date
--no-replace-microseconds	Whether microseconds should be zeroed
-r, --run-id	Run ID
-S, --subdir	File or directory location for DAGs. Defaults to `[AIRFLOW_HOME]/dags`, where `[AIRFLOW_HOME]` is the value set for `AIRFLOW_HOME` config in *airflow.cfg*
-o, --output	Output format. Possible values: `table`, `json`, `yaml`, `plain`. Default: `table`
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

Examples

To trigger the tutorial DAG, run:

```
$ airflow dags trigger tutorial
```

Output example:

```
    |          |              | data_interva | data_interva |          | external_tri | last_schedul | logical_dat |          |
conf | dag_id   | dag_run_id   | l_end        | l_start      | end_date | gger         | ing_decision | e           | run_type |
start_date | state
=====+==========+==============+==============+==============+==========+==============+==============+=============+==========+=
===========+=======
{}   | tutorial | manual__2025 | 2025-04-04T0 | 2025-04-04T0 | None     | True         | None         | 2025-04-04T | manual   |
None       | queued
     |          | -04-04T05:16 | 5:16:28+00:0 | 5:16:28+00:0 |          |              |              | 05:16:28+00 |          |
|
     |          | :28+00:00    | 0            | 0            |          |              |              | :00         |          |
|
```

unpause

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

Arguments

Examples

Definition

Resumes one or more DAGs. This command allows to restore the execution of specified DAGs, enabling further task scheduling.

Usage

```
$ airflow dags unpause [-h] [-o table, json, yaml, plain] [-S SUBDIR]
                       [--treat-dag-id-as-regex] [-v] [-y]
                       dag_id
```

Arguments

Parameter	Description
dag_id	ID of the DAG
--treat-dag-id-as-regex	If set, `dag_id` will be treated as regex instead of a string
-o, --output	Output format. Possible values: `table`, `json`, `yaml`, `plain`. Default: `table`
-S, --subdir	File or directory location for DAGs. Defaults to `[AIRFLOW_HOME]/dags`, where `[AIRFLOW_HOME]` is the value set for `AIRFLOW_HOME` config in *airflow.cfg*
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose
-y, --yes	Do not prompt for confirmation

Examples

To unpause the tutorial DAG, run:

```
$ airflow dags unpause tutorial
```

Output example:

```
Dag: tutorial, paused: False
```

check

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

- Arguments

Examples

Definition

Checks if the database can be reached.

Usage

```
$ airflow db check
                  [-h]
                  [--retry RETRY]
                  [--retry-delay RETRY_DELAY]
                  [-v]
```

Arguments

Parameter	Description
--retry	Retry database check upon failure. Default value is `0` (no retry)
--retry-delay	Wait time between retries in seconds. Default: `1`
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

Examples

To perform the database connection check, run:

```
$ airflow db check
```

Output example:

```
[2025-04-08T05:20:05.969+0000] {db.py:1741} INFO - Connection successful.
```

check-migrations

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

Arguments

Examples

Definition

Checks if the database migration has finished. The check can be one-time or continuous until the timeout expires.

Usage

```
$ airflow db check-migrations
                            [-h]
                            [-t MIGRATION_WAIT_TIMEOUT]
                            [-v]
```

Arguments

Parameter	Description
-t, --migration-wait-timeout	Timeout to wait in seconds for a database to migrate. Default: `60`
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

Examples

To check if the database migration have finished in 5 minutes, run:

```
$ airflow db check-migrations -t 300
```

clean

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

└ Arguments

Examples

Definition

Purges old records from metastore tables.

Usage

```
$ airflow db clean [-h] --clean-before-timestamp CLEAN_BEFORE_TIMESTAMP
                   [--dry-run]
                   [--skip-archive]
                   [-t TABLES]
                   [-v]
                   [-y]
```

Arguments

Parameter	Description
--clean-before-timestamp	Purge data created before the specified timestamp. If no timezone info is supplied, then the dates are assumed to be in Airflow default timezone. Example: `2022-01-01 00:00:00+01:00`
--dry-run	Perform a dry run. Default: `False`
--skip-archive	Do not preserve purged records in an archive table. Default: `False`
-t, --tables	The names of the tables to be dropped. Accepts a comma-separated list. Possible values: `callback_request`, `celery_taskmeta`, `celery_tasksetmeta`, `dag`, `dag_run`, `dataset_event`, `import_error`, `job`, `log`, `session`, `sla_miss`, `task_fail`, `task_instance`, `task_instance_history`, `task_reschedule`, `trigger`, `xcom`
-y, --yes	Do not prompt to confirm. Default: `False`
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

Examples

To purge records from the `task_fail` table, run:

```
$ airflow db clean --clean-before-timestamp 2025-04-03T12:42:04.284+0000 -t task_fail
```

Output example:

```

You have requested that we purge all data prior to 2025-04-03T12:42:04.284000+00:00 for tables ['task_fail'].
This is irreversible.  Consider backing up the tables first and / or doing a dry run with option --dry-run.
Enter 'delete rows' (without quotes) to proceed.
delete rows
Checking table task_fail
Found 1 rows meeting deletion criteria.
Performing Delete...
Moving data to table _airflow_deleted__task_fail__20250403043257
Finished Performing Delete
```

downgrade

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

└ Arguments

Definition

Downgrades the schema of the metadata database.

You can downgrade to a particular Airflow version using the `--to-version` option. Alternatively you may provide an Alembic revision ID to downgrade to using the `--to-revision` option.

Options `--from-revision` and `--from-version` may only be used in conjunction with the `--show-sql-only` option because when actually running migrations, the downgrade must only be performed from current revision.

If you want to preview the commands but not execute them, use `--show-sql-only` option.

For a mapping between Airflow version and Alembic revision see Reference for Database Migrations.

TIP

It is recommended that you backup your database before running a downgrade or any other database operation.

Usage

```
$ airflow db downgrade [-h] [--from-revision FROM_REVISION]
                       [--from-version FROM_VERSION] [-s] [-r TO_REVISION]
                       [-n TO_VERSION] [-v] [-y]
```

Arguments

Parameter	Description
--from-revision	Downgrade from the specified Alembic revision (optional)
--from-version	Downgrade from the specified version (optional)
-s, --show-sql-only	Print out SQL scripts for offline migration instead of running an actual migration. Required when using either `--from-revision` or `--from-version` options
-r, --to-revision	The Alembic revision to downgrade to
-n, --to-version	If provided, only run migrations up to this version
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose
-y, --yes	Do not prompt for confirmation

drop-archived

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

- Arguments

Examples

Definition

Drops archived tables created through the db clean command.

Usage

```
$ airflow db drop-archived [-h] [-t TABLES] [-y]
```

Arguments

Parameter	Description
-t, --tables	Names of the tables to delete (use comma-separated list). Options: `callback_request`, `celery_taskmeta`, `celery_tasksetmeta`, `dag`, `dag_run`, `dataset_event`, `import_error`, `job`, `log`, `session`, `sla_miss`, `task_fail`, `task_instance`, `task_instance_history`, `task_reschedule`, `trigger`, `xcom`
-y, --yes	Do not prompt for confirmation

Examples

To drop the archived `session` tables, run:

```
$ airflow db drop-archived -t session
```

export-archived

Elena Kostyuchenko

Contents

To Table of Contents

- Definition
- Usage
  - Arguments
- Examples

Definition

Exports data from the archive tables.

Usage

```
$ airflow db export-archived [-h] [--drop-archives] [--export-format {csv}]
                             --output-path DIRPATH [-t TABLES] [-y]
```

Arguments

Parameter	Description
--drop-archives	Drop the archived tables after exporting
--export-format	The file format to export the archive data. Default: `csv`
--output-path	The path to the output directory to export the archive data. Must be an existing directory
-t, --tables	Table names to perform maintenance on (use comma-separated list). Options: `callback_request`, `celery_taskmeta`, `celery_tasksetmeta`, `dag`, `dag_run`, `dataset_event`, `import_error`, `job`, `log`, `session`, `sla_miss`, `task_fail`, `task_instance`, `task_instance_history`, `task_reschedule`, `trigger`, `xcom`
-y, --yes	Do not prompt for confirmation

Examples

To export the archived `session` table, run:

```
$ airflow db export-archived --output-path /tmp/exported-sessions.json session
```

init

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

└ Arguments

Examples

Definition

Initializes the metadata database.

Usage

```
$ airflow db init [-h] [-v]
```

Arguments

Parameter	Description
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

Examples

To initialize a metadata database, run:

```
$ airflow db init
```

reset

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

└ Arguments

Definition

Drops and then rebuilds the metadata database.

Usage

```
$ airflow db reset [-h] [-s] [-m] [-v] [-y]
```

Arguments

Parameter	Description
-s, --skip-init	Only remove tables and not perform `db init`
-m, --use-migration-files	Use migration files to perform migration
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose
-y, --yes	Do not prompt for confirmation

shell

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

└ Arguments

Definition

Runs a shell to access the database.

Usage

```
$ airflow db shell [-h] [-v]
```

Arguments

Parameter	Description
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

upgrade

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

- Arguments

Examples

Definition

Upgrades the metadata database to a selected version.

Usage

```
$ airflow db upgrade [-h] [--from-revision FROM_REVISION]
                     [--from-version FROM_VERSION] [-s] [-r TO_REVISION]
                     [-n TO_VERSION] [-m] [-v]
```

Arguments

Parameter	Description
--from-revision	(Optional) If generating SQL, select the Alembic revision to upgrade from
--from-version	(Optional) If generating SQL, select the version to upgrade from
-s, --show-sql-only	Print out SQL scripts for offline migration instead of running an actual migration. Required if using either `--from-revision` or `--from-version` options
-r, --to-revision	Migrate up to and including the specified Alembic revision
-n, --to-version	The Airflow version to upgrade to
-m, --use-migration-files	Use migration files to perform migration
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

Examples

Example command that displays SQL instructions for upgrading a database:

```
$ airflow db upgrade --from-version 2.3.2 --to-version 2.3.3  --show-sql-only
```

Output example:

```
DB: postgresql+psycopg2://airflow:***@elenas-ado.ru-central1.internal:5432/airflow
Generating sql for upgrade -- upgrade commands will *not* be submitted.
BEGIN;

-- Running upgrade 3c94c427fdf6 -> f5fcbda3e651

CREATE INDEX idx_task_fail_task_instance ON task_fail (dag_id, task_id, run_id, map_index);

CREATE INDEX idx_task_reschedule_dag_run ON task_reschedule (dag_id, run_id);

CREATE INDEX idx_xcom_task_instance ON xcom (dag_id, task_id, run_id, map_index);

UPDATE alembic_version SET version_num='f5fcbda3e651' WHERE alembic_version.version_num = '3c94c427fdf6';

COMMIT;
```

jobs check

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

Arguments

Examples

Definition

Checks the status of all jobs in Airflow.

Usage

```
$ airflow jobs check [-h] [--allow-multiple] [--hostname HOSTNAME]
                     [--job-type {BackfillJob,LocalTaskJob,SchedulerJob,TriggererJob,DagProcessorJob}]
                     [--limit LIMIT] [--local] [-v]
```

Arguments

Parameter	Description
--allow-multiple	Run the command even if multiple matching alive jobs are found
--hostname	Check jobs on the specified host
--job-type	The type of jobs to check. Possible values: `BackfillJob`, `LocalTaskJob`, `SchedulerJob`, `TriggererJob`, `DagProcessorJob`
--limit	The number of recent jobs that will be checked. To disable limit, set `0`. Default: `1`
--local	Show jobs only from the local host (those with a hostname matching what `hostname_callable` returns)
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

Examples

To check the status of all jobs on a host, run:

```
$ airflow jobs check --hostname 10.92.41.34
```

cleanup-pods

Elena Kostyuchenko

Contents

To Table of Contents

- Definition
- Usage
  - Arguments
- Examples

Definition

Cleans up Kubernetes pods (created by KubernetesExecutor/KubernetesPodOperator) in `evicted` / `failed` / `succeeded` / `pending` states.

Usage

```
$ airflow kubernetes cleanup-pods [-h]
                                  [--min-pending-minutes MIN_PENDING_MINUTES]
                                  [--namespace NAMESPACE] [-v]
```

Arguments

Parameter	Description
--min-pending-minutes	Deletes the pending pods created before the selected time interval. Default value is 30 minutes (`30`). The minimum value is `5`
--namespace	Kubernetes namespace. Default value is Kubernetes namespace in Airflow configuration
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

Examples

To delete pods created more than an hour ago, run:

```
$ airflow kubernetes cleanup-pods --min-pending-minutes 60
```

generate-dag-yaml

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

Arguments

Examples

Definition

Generates YAML files for all tasks in a DAG. This command can be useful for debugging tasks without launching them in a cluster.

Usage

```
$ airflow kubernetes generate-dag-yaml [-h] [-o OUTPUT_PATH] [-S SUBDIR] [-v]
                                       dag_id logical_date
```

Arguments

Parameter	Description
dag_id	The ID of the DAG
logical_date	The logical date of the DAG
-o, --output-path	The output path for generated YAML files. Default: `[CWD]`
-S, --subdir	File or directory location for DAGs. Defaults to `[AIRFLOW_HOME]/dags`, where `[AIRFLOW_HOME]` is the value set for `AIRFLOW_HOME` config in *airflow.cfg*
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

Examples

To generate YAML files for the `tutorial` DAG, run:

```
$ airflow kubernetes generate-dag-yaml tutorial
```

delete

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

└ Arguments

Definition

Deletes a pool.

Usage

```
$ airflow pools delete [-h] [-o table, json, yaml, plain] [-v] NAME
```

Arguments

Parameter	Description
NAME	Pool name
-o, --output	Output format. Possible values: `table`, `json`, `yaml`, `plain`. Default: `table`
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

export

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

Arguments

Examples

Definition

Exports pools to a file.

Usage

```
$ airflow pools export [-h] [-v] FILEPATH
```

Arguments

Parameter	Description
FILEPATH	Export all pools to a JSON file specified by the path
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

Examples

To export pools to a file, run:

```
$ airflow pools export /tmp/exported-pools.json
```

get

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

Arguments

Examples

Definition

Displays pool size.

Usage

```
$ airflow pools get [-h] [-o table, json, yaml, plain] [-v] NAME
```

Arguments

Parameter	Description
NAME	Pool name
-o, --output	Output format. Possible values: `table`, `json`, `yaml`, `plain`. Default: `table`
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

Examples

To get information about the default pool, run:

```
$ airflow pools get default_pool
```

Output example:

```
pool         | slots | description
=============+=======+=============
default_pool | 128   | Default pool
```

import

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

Arguments

Examples

Definition

Imports pools from a JSON file to the database.

Usage

```
$ airflow pools import [-h] [-v] file
```

Arguments

Parameter	Description
file	JSON file to import the pools from
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

Examples

To import pools from a file, run:

```
$ airflow pools import /tmp/pools.json
```

list

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

Arguments

Examples

Definition

Displays the list of existing pools in the selected format.

Usage

```
$ airflow pools list [-h] [-o table, json, yaml, plain] [-v]
```

Arguments

Parameter	Description
-o, --output	Output format. Possible values: `table`, `json`, `yaml`, `plain`. Default: `table`
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

Examples

To get the list of existing pools as a console output, run:

```
$ airflow pools list
```

Output example:

```
pool         | slots | description
=============+=======+=============
default_pool | 128   | Default pool
```

set

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

└ Arguments

Examples

Definition

Configures a pool.

Usage

```
$ airflow pools set [-h] [--include-deferred] [-o table, json, yaml, plain] [-v]
                    NAME slots description
```

Arguments

Parameter	Description
NAME	Pool name
slots	Pool slots
description	Pool description
--include-deferred	Include deferred tasks in calculations for the pool
-o, --output	Output format. Possible values: `table`, `json`, `yaml`, `plain`. Default: `table`
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

Examples

Command example for pool configuration:

```
$ airflow pools set test_pool 50
```

auth

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

- Arguments

Examples

Definition

Displays information about the provided API authentication backends.

Usage

```
$ airflow providers auth
                        [-h]
                        [-o table, json, yaml, plain]
                        [-v]
```

Arguments

Parameter	Description
-o, --output	Output format. Possible values: `table`, `json`, `yaml`, `plain`. Default: `table`
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

Examples

To display information about the provided API Auth backends, run the command:

```
$ airflow providers auth
```

Output example:

```
api_auth_backand_module
=======================================================
airflow.providers.google.common.auth_backend.google_openid
```

behaviours

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

Arguments

Examples

Definition

Displays information about registered connection types with custom behaviours.

Usage

```
$ airflow providers behaviours
                                [-h]
                                [-o table, json, yaml, plain]
                                [-v]
```

Arguments

Parameter	Description
-o, --output	Output format. Possible values: `table`, `json`, `yaml`, `plain`. Default: `table`
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

Examples

To display information about registered connections, run:

```
$ airflow providers behaviours
```

Output example:

```
field_behaviours
========================
adls
apprise
arangodb
asana
aws
azure
azure_batch
azure_container_registry
azure_container_volume
azure_cosmos
azure_data_explorer
azure_data_factory
azure_data_lake
azure_fileshare
azure_service_bus
azure_synapse
azure_synapse_pipeline
chime
cloudant
dbt_cloud
docker
emr
gcpssh
github
google_cloud_platform
jdbc
jenkins
kafka
kubernetes
postgres
redshift
salesforce
sftp
slack
slackwebhook
smtp
spark
ssh
tabular
vault
wasb
yandexcloud
```

get

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

Arguments

Examples

Definition

Displays information about a provider.

Usage

```
$ airflow providers get [-h] [--color {auto,on,off}] [-f]
                        [-o table, json, yaml, plain] [-v]
                        provider_name
```

Arguments

Parameter	Description
provider_name	Provider ID in Airflow
--color	Use colored output. Possible values: `off`, `auto`, `on`. Default: `auto`
-o, --output	Output format. Possible values: `table`, `json`, `yaml`, `plain`. Default: `table`
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

Examples

To display information about a provider, run:

```
$ airflow providers get test_provider
```

hooks

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

└ Arguments

Definition

Lists registered provider hooks.

Usage

```
$ airflow providers hooks [-h] [-o table, json, yaml, plain] [-v]
```

Arguments

Parameter	Description
-o, --output	Output format. Possible values: `table`, `json`, `yaml`, `plain`. Default: `table`
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

links

Elena Kostyuchenko

Contents

To Table of Contents

- Definition
- Usage
  - Arguments
- Examples

Definition

Displays extra links registered by the providers.

Usage

```
$ airflow providers links [-h] [-o table, json, yaml, plain] [-v]
```

Arguments

Parameter	Description
-o, --output	Output format. Possible values: `table`, `json`, `yaml`, `plain`. Default: `table`
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

Examples

To display provider links, run:

```
$ airflow providers links
```

Output example:

```
extra_link_class_name
================================================================================================
airflow.providers.amazon.aws.links.batch.BatchJobDefinitionLink
airflow.providers.amazon.aws.links.batch.BatchJobDetailsLink
airflow.providers.amazon.aws.links.batch.BatchJobQueueLink
airflow.providers.amazon.aws.links.emr.EmrClusterLink
airflow.providers.amazon.aws.links.emr.EmrLogsLink
```

list

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

- Arguments

Examples

Definition

Displays installed providers list.

Usage

```
$ airflow providers list [-h] [-o table, json, yaml, plain] [-v]
```

Arguments

Parameter	Description
-o, --output	Output format. Possible values: `table`, `json`, `yaml`, `plain`. Default: `table`
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

Examples

To display the list of installed providers:

```
$ airflow providers list -o table
```

logging

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

- Arguments

Examples

Definition

Displays information about the task logging handlers.

Usage

```
$ airflow providers logging [-h] [-o table, json, yaml, plain] [-v]
```

Arguments

Parameter	Description
-o, --output	Output format. Possible values: `table`, `json`, `yaml`, `plain`. Default: `table`
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

Examples

To display information about the task logging providers, run:

```
$ airflow providers logging
```

Output example:

```
logging_class_name
================================================================================
airflow.providers.alibaba.cloud.log.oss_task_handler.OSSTaskHandler
airflow.providers.amazon.aws.log.cloudwatch_task_handler.CloudwatchTaskHandler
airflow.providers.amazon.aws.log.s3_task_handler.S3TaskHandler
airflow.providers.elasticsearch.log.es_task_handler.ElasticsearchTaskHandler
airflow.providers.google.cloud.log.gcs_task_handler.GCSTaskHandler
airflow.providers.google.cloud.log.stackdriver_task_handler.StackdriverTaskHandler
airflow.providers.microsoft.azure.log.wasb_task_handler.WasbTaskHandler
```

secrets

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

- Arguments

Examples

Definition

Displays information about secrets backends provided.

Usage

```
$ airflow providers secrets [-h] [-o table, json, yaml, plain] [-v]
```

Arguments

Parameter	Description
-o, --output	Output format. Possible values: `table`, `json`, `yaml`, `plain`. Default: `table`
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

Examples

To display information about available secrets manager backends, run:

```
$ airflow providers secrets
```

Output example:

```
secrets_backend_class_name
================================================================================
airflow.providers.amazon.aws.secrets.secrets_manager.SecretsManagerBackend
airflow.providers.amazon.aws.secrets.systems_manager.SystemsManagerParameterStoreBackend
airflow.providers.google.cloud.secrets.secret_manager.CloudSecretManagerBackend
airflow.providers.hashicorp.secrets.vault.VaultBackend
airflow.providers.microsoft.azure.secrets.key_vault.AzureKeyVaultBackend
```

widgets

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

Arguments

Examples

Definition

Displays information about registered connections from widgets.

Usage

```
$ airflow providers widgets [-h] [-o table, json, yaml, plain] [-v]
```

Arguments

Parameter	Description
-o, --output	Output format. Possible values: `table`, `json`, `yaml`, `plain`. Default: `table`
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

Examples

To get the list of registered widget connections, run:

```
$ airflow providers widgets
```

Output example:

```
connection_parameter_name                                  | class                       | package_name
| field_type
===========================================================+=============================+=====================================
====+==============
extra__adls__connection_string                             | AzureDataLakeStorageV2Hook | apache-airflow-providers-microsoft-
azure | PasswordField
extra__adls__managed_identity_client_id                    | AzureDataLakeStorageV2Hook | apache-airflow-providers-microsoft-
azure | StringField
extra__adls__tenant_id                                     | AzureDataLakeStorageV2Hook | apache-airflow-providers-microsoft-
azure | StringField
extra__adls__workload_identity_tenant_id                   | AzureDataLakeStorageV2Hook | apache-airflow-providers-microsoft-
azure | StringField
extra__apprise__config                                     | AppriseHook                 | apache-airflow-providers-apprise
| PasswordField
```

add-perms

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

Arguments

Examples

Definition

Adds permissions to roles.

Usage

```
$ airflow roles add-perms [role ...]
                          [-h] -a
                          [ACTION ...]
                          -r [RESOURCE ...]
                          [-v]
```

Arguments

Parameter	Description
role	The name of the role for which permissions need to be added
-a, --action	The action to be allowed with the selected resource
-r, --resource	The resource to which permissions should be granted
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

Examples

To allow users with the `viewer` role to edit connections, run:

```
$ airflow roles add-perms Viewer -a can_edit -r Connections
```

Output example:

```
Added can edit on Connections to role Viewer
Added 1 permission(s)
```

create

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

Arguments

Examples

Definition

Creates a role.

Usage

```
$ airflow roles create [-h] [-v] [role ...]
```

Arguments

Parameter	Description
role	The name of a role
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

Examples

To create a test role, run:

```
$ airflow roles create test_role
```

Output example:

```
Added 1 role(s)
```

del-perms

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

Arguments

Examples

Definition

Deletes permissions for a role.

Usage

```
$ airflow roles del-perms [-h] [-a [ACTION ...]] -r [RESOURCE ...] [-v]
                          [role ...]
```

Arguments

Parameter	Description
role	The name of a role
-a, --action	The action of permissions
-r, --resource	The resource of permissions
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

Examples

To revoke the ability to edit connections from the test role, run:

```
$ airflow roles del-perms can_edit connections test_role
```

delete

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

Arguments

Examples

Definition

Deletes a role.

Usage

```
$ airflow roles delete [-h] [-v] [role ...]
```

Arguments

Parameter	Description
role	The name of a role
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

Examples

To delete the test role, run:

```
$ airflow roles delete test_role
```

export

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

- Arguments

Examples

Definition

Exports roles from a database to a JSON file without copying their permissions.

Usage

```
$ airflow roles export [-h] [-p] [-v] file
```

Arguments

Parameter	Description
file	The name of the JSON file with the exported roles
-p, --pretty	Format output JSON file by sorting role names and indenting by 4 spaces
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

Examples

Example command to export roles:

```
$ airflow roles export /tmp/exported-roles.json
```

import

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

Arguments

Examples

Definition

Imports roles from a JSON file to the database. The roles will be imported without their permissions.

Usage

```
$ airflow roles import [-h] [-v] file
```

Arguments

Parameter	Description
file	JSON file to import the roles from
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

Examples

To import roles from a file, run:

```
$ airflow roles import /tmp/roles.json
```

list

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

└ Arguments

Examples

Definition

Displays the list of roles.

Usage

```
$ airflow roles list [-h] [-o table, json, yaml, plain] [-p] [-v]
```

Arguments

Parameter	Description
-p, --permission	Show role permissions
-o, --output	Output format. Possible values: `table`, `json`, `yaml`, `plain`. Default: `table`
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

Examples

To display the list of roles, run:

```
$ airflow roles list
```

Output example:

```
name
=========
Admin
Op
Public
User
Viewer
test_role
```

clear

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

Arguments

Examples

Definition

Deletes execution information of task instances, returning them to the state before they ever ran.

Usage

```
$ airflow tasks clear [-h] [-R] [-d] [-e END_DATE] [-X] [-x] [-f] [-r]
                      [-s START_DATE] [-S SUBDIR] [-t TASK_REGEX] [-u] [-v] [-y]
                      dag_id
```

Arguments

Parameter	Description
dag_id	ID of the DAG
-R, --dag-regex	Search `dag_id` as regex instead of exact string
-d, --downstream	Include downstream tasks
-e, --end-date	Override `end_date`. The date format is `YYYY-MM-DD`
-X, --exclude-parentdag	Exclude parent DAGs if the task cleared is a part of a sub-DAG
-x, --exclude-subdags	Exclude sub-DAGs
-f, --only-failed	Only failed jobs
-r, --only-running	Only running jobs
-s, --start-date	Override `start_date`. The date format is `YYYY-MM-DD`
-S, --subdir	File or directory location for DAGs. Defaults to `[AIRFLOW_HOME]/dags`, where `[AIRFLOW_HOME]` is the value set for `AIRFLOW_HOME` config in *airflow.cfg*
-t, --task-regex	Regex to filter specific `task_ids`
-u, --upstream	Include upstream tasks
-y, --yes	Do not prompt to confirm
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

Examples

To clear all failed tasks for the specified DAG, run:

```
$ airflow tasks clear example_dag_decorator --only-failed
```

Output example:

```
[2025-04-03T04:50:11.935+0000] {dagbag.py:541} INFO - Filling up the DagBag from /opt/airflow/dags
You are about to delete these 1 tasks:
<TaskInstance: example_dag_decorator.send_email manual__2025-04-03T11:27:54.722760+00:00 [failed]>

Are you sure? [y/n]
```

failed-deps

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

- Arguments

Examples

Definition

Returns the unmet dependencies for a task instance from the perspective of the scheduler. This command helps to determine why a task instance is not scheduled and queued by the scheduler.

Usage

```
$ airflow tasks failed-deps [-h] [--map-index MAP_INDEX] [-S SUBDIR] [-v]
                            dag_id task_id execution_date_or_run_id
```

Arguments

Parameter	Description
dag_id	The ID of the DAG
task_id	The ID of the task
execution_date_or_run_id	The `execution_date` of the DAG or `run_id` of the DAG run
--map-index	Mapped task index. Default: `-1`
-S, --subdir	File or directory location for DAGs. Defaults to `[AIRFLOW_HOME]/dags`, where `[AIRFLOW_HOME]` is the value set for `AIRFLOW_HOME` config in *airflow.cfg*
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

Examples

An example command for displaying unmet dependencies for a DAG:

```
$ airflow tasks failed-deps example_dag extract_people manual__2025-04-14T15:29:17.298007+00:00
```

list

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

- Arguments

Examples

Definition

Displays the list of tasks within a DAG.

Usage

```
$ airflow tasks list [-h] [-S SUBDIR] [-t] [-v] dag_id
```

Arguments

Parameter	Description
dag_id	The ID of the DAG
-S, --subdir	File or directory location for DAGs. Defaults to `[AIRFLOW_HOME]/dags`, where `[AIRFLOW_HOME]` is the value set for `AIRFLOW_HOME` config in *airflow.cfg*
-t, --tree	Deprecated. Use `dags show` instead. Displays tasks in a tree. Note that generating the tree can be slow and the output can be very large for some DAGs
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

Examples

To display the list of tasks in the tutorial DAG, run:

```
$ airflow tasks list tutorial
```

Output example:

```
print_date
sleep
templated
```

render

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

- Arguments

Examples

Definition

Renders templates of a task instance.

Usage

```
$ airflow tasks render [-h] [--map-index MAP_INDEX] [-S SUBDIR] [-v]
                       dag_id task_id execution_date_or_run_id
```

Arguments

Parameter	Description
dag_id	ID of the DAG
task_id	ID of the task
execution_date_or_run_id	The `execution_date` of the DAG or `run_id` of the DAG run
--map-index	Mapped task index. Default: `-1`
-S, --subdir	File or directory location for DAGs. Defaults to `[AIRFLOW_HOME]/dags`, where `[AIRFLOW_HOME]` is the value set for `AIRFLOW_HOME` config in *airflow.cfg*
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

Examples

Example command to display task instance templates:

```
$ airflow tasks render example_dag extract_people manual__2025-04-14T15:29:17.298007+00:00
```

run

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

Arguments

Examples

Definition

Runs a single task instance in a DAG.

Usage

```
$ airflow tasks run [-h] [--cfg-path CFG_PATH] [-d {ignore,wait,check}] [-f]
                    [-A] [-i] [-I] [-N] [-l] [--map-index MAP_INDEX] [-m]
                    [-p PICKLE] [--pool POOL] [--read-from-db] [--ship-dag]
                    [-S SUBDIR] [-v]
                    dag_id task_id execution_date_or_run_id
```

Arguments

Parameter	Description
dag_id	ID of the DAG
task_id	ID of the task
execution_date_or_run_id	The `execution_date` of the DAG or `run_id` of the DAG run
--cfg-path	Path to config file to use instead of *airflow.cfg*
-d, --depends-on-past	Determine how Airflow should deal with past dependencies. Possible values: ■ `check` (default) — Airflow will check if the past dependencies are met for the tasks having `depends_on_past=True` before running them. ■ `ignore` — the past dependencies will be ignored. ■ `wait` — if `depends_on_past=True`, Airflow will wait until the past dependencies are met before running or skipping the task.
-f, --force	Ignore previous task instance state, re-run regardless if task has already succeeded or failed
-A, --ignore-all-dependencies	Ignores all non-critical dependencies, including `ignore_ti_state` and `ignore_task_deps`
-i, --ignore-dependencies	Ignore task-specific dependencies, e.g. upstream, depends_on_past, and retry delay dependencies
-I, --ignore-depends-on-past	Deprecated. Use `--depends-on-past ignore` instead
-N, --interactive	Do not capture standard output and error streams (useful for interactive debugging)
-l, --local	Run the task using the LocalExecutor
--map-index	Mapped task index
-m, --mark-success	Mark jobs as succeeded without running them
-p, --pickle	Serialized pickle object of the entire DAG (used internally)
--pool	Resource pool to use
--read-from-db	Read DAG from DB instead of the DAG file
--ship-dag	Pickles (serializes) the DAG and ships it to the worker
-S, --subdir	File or directory location for DAGs. Defaults to `[AIRFLOW_HOME]/dags`, where `[AIRFLOW_HOME]` is the value set for `AIRFLOW_HOME` config in *airflow.cfg*
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

Examples

Example command for running a task separately:

```
$ airflow tasks run tutorial print_date 2025-04-04
```

state

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

Arguments

Examples

Definition

Displays the status of a task instance.

Usage

```
$ airflow tasks state [-h] [--map-index MAP_INDEX] [-S SUBDIR] [-v]
                      dag_id task_id execution_date_or_run_id
```

Arguments

Parameter	Description
dag_id	ID of the DAG
task_id	ID of the task
execution_date_or_run_id	The `execution_date` of the DAG or `run_id` of the DAG run
--map-index	Mapped task index
-S, --subdir	File or directory location for DAGs. Defaults to `[AIRFLOW_HOME]/dags`, where `[AIRFLOW_HOME]` is the value set for `AIRFLOW_HOME` config in *airflow.cfg*
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

Examples

To display the status of the `print_date` task instance, run:

```
$ airflow tasks state tutorial print_date scheduled__2025-04-13T15:26:57.044494+00:00
```

Output example:

```
success
```

states-for-dag-run

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

- Arguments

Examples

Definition

Displays the status of all task instances in a DAG run.

Usage

```
$ airflow tasks states-for-dag-run [-h] [-o table, json, yaml, plain] [-v]
                                   dag_id execution_date_or_run_id
```

Arguments

Parameter	Description
dag_id	ID of the DAG
execution_date_or_run_id	The `execution_date` of the DAG or `run_id` of the DAG run
-o, --output	Output format. Possible values: `table`, `json`, `yaml`, `plain`. Default: `table`
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

Examples

Example command for displaying task status information:

```
$ airflow tasks states-for-dag-run tutorial scheduled__2025-04-13T15:26:57.044494+00:00
```

Output example:

```
dag_id   | execution_date                   | task_id    | state   | start_date                       | end_date
=========+==================================+============+=========+==================================+==========================
=======
tutorial | 2025-04-13T15:26:57.044494+00:00 | print_date | success | 2025-04-14T15:27:12.075391+00:00 | 2025-04-14T15:27:12.
311052+00:00
tutorial | 2025-04-13T15:26:57.044494+00:00 | templated  | success | 2025-04-14T15:27:13.274360+00:00 | 2025-04-14T15:27:13.
533015+00:00
tutorial | 2025-04-13T15:26:57.044494+00:00 | sleep      | success | 2025-04-14T15:27:13.245757+00:00 | 2025-04-14T15:27:18.
510164+00:00
```

test

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

└ Arguments

Examples

Definition

Tests a task instance. This will run a task without checking for dependencies or recording its state in the database.

Usage

```
$ airflow tasks test [-h] [-n] [--env-vars ENV_VARS] [--map-index MAP_INDEX]
                     [-m] [-S SUBDIR] [-t TASK_PARAMS] [-v]
                     dag_id task_id [execution_date_or_run_id]
```

Arguments

Parameter	Description
dag_id	ID of the DAG
task_id	ID of the task
execution_date_or_run_id	The `execution_date` of the DAG or `run_id` of the DAG run (optional)
-n, --dry-run	Perform a dry run for each task. Only renders Template Fields for each task, nothing else
--env-vars	Set environment variable in both parsing time and runtime for each entry supplied in a JSON dictionary
--map-index	Mapped task index
-m, --post-mortem	Open debugger on uncaught exception
-t, --task-params	Send a JSON parameters dictionary to the task
-S, --subdir	File or directory location for DAGs. Defaults to `[AIRFLOW_HOME]/dags`, where `[AIRFLOW_HOME]` is the value set for `AIRFLOW_HOME` config in *airflow.cfg*
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

Examples

Example command for testing a task instance:

```
$ airflow tasks test tutorial print_date scheduled__2025-04-13T15:26:57.044494+00:00
```

Output example:

```
[2025-04-15T07:41:32.631+0000] {dagbag.py:541} INFO - Filling up the DagBag from /opt/airflow/dags
[2025-04-15T07:41:33.134+0000] {example_python_operator.py:90} WARNING - The virtalenv_python example task requires virtualenv,
please install it.
[2025-04-15T07:41:33.729+0000] {tutorial_taskflow_api_virtualenv.py:29} WARNING - The tutorial_taskflow_api_virtualenv example
DAG requires virtualenv, please install it.
[2025-04-15T07:41:34.220+0000] {taskinstance.py:1103} INFO - Dependencies all met for dep_context=non-requeueable deps ti=
<TaskInstance: tutorial.print_date scheduled__2025-04-13T15:26:57.044494+00:00 [success]>
[2025-04-15T07:41:34.227+0000] {taskinstance.py:1103} INFO - Dependencies all met for dep_context=requeueable deps ti=
<TaskInstance: tutorial.print_date scheduled__2025-04-13T15:26:57.044494+00:00 [success]>
[2025-04-15T07:41:34.227+0000] {taskinstance.py:1308} INFO - Starting attempt 2 of 2
[2025-04-15T07:41:34.232+0000] {taskinstance.py:1327} INFO - Executing <Task(BashOperator): print_date> on 2025-04-13 15:26:57.
044494+00:00
[2025-04-15T07:41:34.281+0000] {taskinstance.py:1545} INFO - Exporting env vars: AIRFLOW_CTX_DAG_EMAIL='airflow@example.com'
AIRFLOW_CTX_DAG_OWNER='airflow' AIRFLOW_CTX_DAG_ID='tutorial' AIRFLOW_CTX_TASK_ID='print_date' AIRFLOW_CTX_EXECUTION_DATE='2025-
04-13T15:26:57.044494+00:00' AIRFLOW_CTX_TRY_NUMBER='2' AIRFLOW_CTX_DAG_RUN_ID='scheduled__2025-04-13T15:26:57.044494+00:00'
[2025-04-15T07:41:34.282+0000] {subprocess.py:63} INFO - Tmp dir root location: /tmp
[2025-04-15T07:41:34.283+0000] {subprocess.py:75} INFO - Running command: ['/bin/bash', '-c', 'date']
[2025-04-15T07:41:34.291+0000] {subprocess.py:86} INFO - Output:
[2025-04-15T07:41:34.293+0000] {subprocess.py:93} INFO - Tue Apr 15 07:41:34 UTC 2025
[2025-04-15T07:41:34.293+0000] {subprocess.py:97} INFO - Command exited with return code 0
[2025-04-15T07:41:34.307+0000] {taskinstance.py:1345} INFO - Marking task as SUCCESS. dag_id=tutorial, task_id=print_date,
execution_date=20250413T152657, start_date=20250414T152712, end_date=20250415T074134
```

add-role

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

- Arguments

Examples

Definition

Adds a role to a user.

Usage

```
$ airflow users add-role [-h]
                         [-e EMAIL]
                         -r ROLE
                         [-u USERNAME]
                         [-v]
```

Arguments

Parameter	Description
-e, --email	Email of the user. Specify either `--username` or `--email`, but not both
-r, --role	Role to add for the user. Examples: `Admin`, `User`, `Op`, `Viewer`, `Public`
-u, --username	Name of the user. Specify either `--username` or `--email`, but not both
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

Examples

To add the administrator role to a user, run:

```
$ airflow users add-role -e user2@user.com -r Admin
```

Output example:

```
[2025-04-03T12:42:04.284+0000] {manager.py:226} INFO - Updated user john doe
User "user2" added to role "Admin"
```

create

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

Arguments

Examples

Definition

Creates a user.

Usage

```
$ airflow users create [-h] -e EMAIL -f FIRSTNAME -l LASTNAME [-p PASSWORD] -r
                       ROLE [--use-random-password] -u USERNAME [-v]
```

Arguments

Parameter	Description
-e, --email	Email of the user
-f, --firstname	First name of the user
-l, --lastname	Last name of the user
-p, --password	Password of the user, required to create a user without using the `--use-random-password` option
-r, --role	Role of the user. Existing roles include `Admin`, `User`, `Op`, `Viewer`, and `Public`
--use-random-password	Use random string as a password. Required when creating a user without the `--password` option
-u, --username	Username of the user
-v, --verbose	Make logging output more verbose. Default: `False`

Examples

To create a user with the admin role, run:

```
$ airflow users create -e user@user.com -f john -l doe -p user1 -r Admin -u user1
```

Output example:

```
[2025-04-03T11:13:35.841+0000] {manager.py:212} INFO - Added user user1
User "user1" created with role "Admin"
```

delete

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

- Arguments

Examples

Definition

Deletes a user.

Usage

```
$ airflow users delete [-h] [-e EMAIL] [-u USERNAME] [-v]
```

Arguments

Parameter	Description
-e, --email	Email of the user
-u, --username	Username of the user
-v, --verbose	Make logging output more verbose. Default: `False`

Examples

To delete the `john` user, run:

```
$ airflow users delete john
```

export

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

└ Arguments

Definition

Exports variables to a JSON file.

Usage

```
$ airflow users export [-h] [-v] FILEPATH
```

Arguments

Parameter	Description
FILEPATH	The name of a destination JSON file for exported users
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

import

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

- Arguments

Examples

Definition

Imports users from a JSON file.

Usage

```
$ airflow users import [-h] [-v] file
```

Arguments

Parameter	Description
file	The name of the JSON file to import users from
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

Example format for a users JSON file:

```
[
    {
        "email": "foo@bar.org",
        "firstname": "Jon",
        "lastname": "Doe",
        "roles": ["Public"],
        "username": "jondoe"
    }
]
```

Examples

To import users from a file, run:

```
$ airflow users import /tmp/users.json
```

list

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

- Arguments

Examples

Definition

Displays the list of users in the selected format.

Usage

```
$ airflow users list [-h] [-o table, json, yaml, plain] [-v]
```

Arguments

Parameter	Description
-o, --output	Output format. Possible values: `table`, `json`, `yaml`, `plain`. Default: `table`
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

Examples

To get the list of users as a console output, run:

```
$ airflow users list
```

remove-role

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

- Arguments

Examples

Definition

Removes a role from a user.

Usage

```
$ airflow users remove-role [-h] [-e EMAIL] -r ROLE [-u USERNAME] [-v]
```

Arguments

Parameter	Description
-e, --email	Email of the user
-r, --role	Role of the user. Existing roles include: `Admin`, `User`, `Op`, `Viewer`, `Public`
-u, --username	Name of the user
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

Examples

To remove the admin role from the test user, run:

```
$ airflow users remove-role -e test@user.com -r admin
```

delete

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

└ Arguments

Definition

Deletes a variable.

Usage

```
$ airflow variables delete [-h] [-v] key
```

Arguments

Parameter	Description
key	Variable key
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

export

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

└ Arguments

Definition

Exports variables to a file or stdout.

Usage

```
$ airflow variables export [-h] [-v] file
```

Arguments

Parameter	Description
file	The name of a destination JSON file for variables
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

get

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

Arguments

Examples

Definition

Displays information about a variable.

Usage

```
$ airflow variables get [-h] [-d VAL] [-j] [-v] key
```

Arguments

Parameter	Description
key	Variable key
-d, --default	Return default value if variable does not exist
-j, --json	Deserialize JSON variable
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

Examples

To get information about the `test_variable`, run:

```
$ airflow variables get test_variable
```

import

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

Arguments

Examples

Definition

Imports variables from a JSON file.

Usage

```
$ airflow variables import [-h] [-a {overwrite,fail,skip}] [-v] file
```

Arguments

Parameter	Description
file	The name of the JSON file to import variables from
-a, --action-on-existing-key	Action to take if a variable already exists. Possible values: `overwrite`, `fail`, `skip`. Default: `overwrite`
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

Examples

To import variables from a file, run:

```
$ airflow variables import /tmp/variables.json
```

list

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

- Arguments

Examples

Definition

Displays the list of variables.

Usage

```
$ airflow variables list [-h] [-o table, json, yaml, plain] [-v]
```

Arguments

Parameter	Description
-o, --output	Output format. Possible values: `table`, `json`, `yaml`, `plain`. Default: `table`
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

Examples

To display the list of variables:

```
$ airflow variables list -o table
```

set

Elena Kostyuchenko

Contents

To Table of Contents

Definition

Usage

- Arguments

Examples

Definition

Creates a variable.

Usage

```
$ airflow variables set [-h] [-j] [-v] key VALUE
```

Arguments

Parameter	Description
key	Variable key
VALUE	Variable value
--description	Variable description (optional)
-j, --json	Serialize JSON variable
-h, --help	Print out the help manual for this command
-v, --verbose	Make logging output more verbose

Examples

Example command that creates a test variable:

```
$ airflow variables set x 1
```

Output example:

```
Variable x created
```

ADO releases

Elena Kostyuchenko

Contents

To Table of Contents

2.10.5.1
2.6.3.2
2.6.3.1

2.10.5.1

Date: 25.03.2026

Upgrades

- Upgraded the Airflow version to 2.10.5_arenadata1
- Upgraded the Monitoring version to 1.3
- Upgraded the duckdb version to 1.4.4

New features

- Added support for integration with services requiring client access on workers (for example, ADH Spark) via shared hosts concept
- Added support for connection strings to Metastore HA clusters
- Added proxy server support for package repositories
- Implemented support for constraints when managing Python dependencies
- The ***Info*** tab now takes custom configuration settings into account
- Added support for AltLinux 10 SP
- Added support for Ansible 2.16
- Implemented automatic removal of status check artifacts during host-component mapping changes to prevent false alerts

Bug fixes

- Airflow and Redis no longer fail to start on new hosts when moving components from hosts in maintenance mode
- Fixed failure of the ***Sync requirements*** action on AltLinux

Misc/Internal

- Minimum required ADCM version is now 2.9.0

2.6.3.2

Date: 22.07.2025

Upgrades

- Prometheus 1.2_arenadata1

New features

- Implemented support for maintenance mode for services
- Added the ability to manage Python dependencies via ADCM
- Added support for Astra Linux 1.7.6 "Voronezh" SE

Improvements

- ***Manage SSL*** action now also manages monitoring options
- Monitoring configurations can now be set granularly using `scrape_config_files`
- Internal refactoring has been performed for optimization and safe operation

Bug fixes

- Flower no longer fails to switch to HTTPS after enabling SSL
- Fixed missing `airflow.flower` group check in the *flower_scraper.yml* template

Misc/Internal

- Minimum required ADCM version is now 2.6.0
- The Airflow bundle and package were refactored to add the ADO version rollback feature. This will be available starting with the next release

2.6.3.1

Date: 18.03.2025

New services

- Redis is now a cluster service for ease of configuration and deployment
- A new monitoring service based on Prometheus and Grafana has been implemented
- Added support for Node Exporter out of the box
- Added support for statsd-exporter out of the box
- Added dashboards in Grafana that display host metrics, Airflow cluster metrics (servers and workers), running DAGs, and Celery queues

New features

- DAG Processor and Triggerer are divided into separate Airflow components for granular cluster setup
- Added new actions for the Airflow service in ADCM:
  - ***Sync permission***
  - ***Rotate fernet key***
- Flower component configuration became more detailed and offers more options
- Added the ability to natively configure a secret backend using integration with HashiCorp Vault
- The link to the Flower web interface is now displayed in the info block

Improvements

- The service configuration now explicitly displays all settings with their descriptions. Additional parameters can be set through the custom section
- Improved handling of sensitive information:
  - All sensitive data (keys, passwords, tokens, etc.) have the secret type and are not displayed explicitly
  - All secrets are generated — `fernet_key`, `secret_key`, etc.
- Introduced a single action for changing the service topology (Add, remove, move components, etc). You can run it both for a service and for the entire cluster

Bug fixes

- Fixed errors that occurred when parsing Airflow config with special characters in `secret_key`
- Fixed an error in the ulimits configuration

Misc/Internal

- The minimum ADCM version now is 2.5.0

Supported services

Elena Kostyuchenko

To Table of Contents

This article lists services included into the current version of ADO. The ***Version*** column contains references to the release notes of each component.

Service	Component	Version
Apache Airflow2	Airflow DAG Processor	2.10.5_arenadata1
	Airflow Flower	2.10.5_arenadata1
	Airflow Scheduler	2.10.5_arenadata1
	Airflow Server	2.10.5_arenadata1
	Airflow Triggerer	2.10.5_arenadata1
	Airflow Worker	2.10.5_arenadata1
Redis	Redis Server	6.2.7
Arenadata Postgres	Arenadata PostgreSQL	16
Arenadata Monitoring	Grafana	1.3
	Prometheus	1.3
	Node Exporter	1.3