

Orchestrating workflows and pipelines with Apache Airflow in Cloudera Data Engineering

Date published: 2020-07-30

Date modified: 2024-03-05

CLOUDERA

Legal Notice

© Cloudera Inc. 2025. All rights reserved.

The documentation is and contains Cloudera proprietary information protected by copyright and other intellectual property rights. No license under copyright or any other intellectual property right is granted herein.

Unless otherwise noted, scripts and sample code are licensed under the Apache License, Version 2.0.

Copyright information for Cloudera software may be found within the documentation accompanying each component in a particular release.

Cloudera software includes software from various open source or other third party projects, and may be released under the Apache Software License 2.0 (“ASLv2”), the Affero General Public License version 3 (AGPLv3), or other license terms. Other software included may be released under the terms of alternative open source licenses. Please review the license and notice files accompanying the software for additional licensing information.

Please visit the Cloudera software product page for more information on Cloudera software. For more information on Cloudera support services, please visit either the Support or Sales page. Feel free to contact us directly to discuss your specific needs.

Cloudera reserves the right to change any products at any time, and without notice. Cloudera assumes no responsibility nor liability arising from the use of products, except as expressly agreed to in writing by Cloudera.

Cloudera, Cloudera Altus, HUE, Impala, Cloudera Impala, and other Cloudera marks are registered or unregistered trademarks in the United States and other countries. All other trademarks are the property of their respective owners.

Disclaimer: EXCEPT AS EXPRESSLY PROVIDED IN A WRITTEN AGREEMENT WITH CLOUDERA, CLOUDERA DOES NOT MAKE NOR GIVE ANY REPRESENTATION, WARRANTY, NOR COVENANT OF ANY KIND, WHETHER EXPRESS OR IMPLIED, IN CONNECTION WITH CLOUDERA TECHNOLOGY OR RELATED SUPPORT PROVIDED IN CONNECTION THEREWITH. CLOUDERA DOES NOT WARRANT THAT CLOUDERA PRODUCTS NOR SOFTWARE WILL OPERATE UNINTERRUPTED NOR THAT IT WILL BE FREE FROM DEFECTS NOR ERRORS, THAT IT WILL PROTECT YOUR DATA FROM LOSS, CORRUPTION NOR UNAVAILABILITY, NOR THAT IT WILL MEET ALL OF CUSTOMER’S BUSINESS REQUIREMENTS. WITHOUT LIMITING THE FOREGOING, AND TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, CLOUDERA EXPRESSLY DISCLAIMS ANY AND ALL IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, QUALITY, NON-INFRINGEMENT, TITLE, AND FITNESS FOR A PARTICULAR PURPOSE AND ANY REPRESENTATION, WARRANTY, OR COVENANT BASED ON COURSE OF DEALING OR USAGE IN TRADE.

Contents

Automating data pipelines using Apache Airflow in Cloudera Data Engineering.....	4
Creating an Airflow DAG using the Pipeline UI.....	6
Managing an Airflow Pipeline using the CDE CLI.....	6
Creating a pipeline using the CDE CLI.....	6
Creating a basic Airflow pipeline using CDE CLI.....	7
Creating a pipeline with additional Airflow configurations using CDE CLI.....	7
Creating an Airflow pipeline with custom files using CDE CLI [technical preview].....	9
Updating a pipeline using the CDE CLI.....	10
Updating a DAG file using the CDE CLI.....	10
Updating the Airflow job configurations using the CDE CLI.....	10
Updating the Airflow file mounts using the CDE CLI [technical preview].....	11
Deleting an Airflow pipeline using the CDE CLI.....	11
Using CDE with an external Apache Airflow deployment.....	11
Supporting Airflow operators and hooks.....	13
Creating a custom Airflow Python environment (Technical Preview).....	14
Creating a custom Airflow Python environment resource.....	14
CDE CLI custom Airflow Python environment flag reference.....	15
Uploading the resource to build the Python environment.....	15
Activating Python environment resources.....	16
Resetting to the default Airflow Python environment.....	16
Deleting Airflow Python environment resources.....	16

Automating data pipelines using Apache Airflow in Cloudera Data Engineering

Cloudera Data Engineering (CDE) enables you to automate a workflow or data pipeline using Apache Airflow Python DAG files. Each CDE virtual cluster includes an embedded instance of Apache Airflow. You can also use CDE with your own Airflow deployment. CDE on CDP Private Cloud currently supports only the CDE job run operator.

Before you begin



Important: Cloudera provides support for Airflow [core operators and hooks](#), but does not provide support for Airflow provider packages. Cloudera Support may require you to remove any installed provider packages during troubleshooting.

About this task

The following instructions are for using the Airflow service provided with each CDE virtual cluster. For instructions on using your own Airflow deployment, see [Using the Cloudera provider for Apache Airflow](#).

Procedure

1. Create an Airflow DAG file in Python. Import the CDE operator and define the tasks and dependencies.
For example, here is a complete DAG file:

```
from dateutil import parser
from datetime import datetime, timedelta
from datetime import timezone
from airflow import DAG
from cloudera.airflow.providers.operators.cde import CdeRunJobOperator

default_args = {
    'owner': 'psherman',
    'retry_delay': timedelta(seconds=5),
    'depends_on_past': False,
    'start_date': datetime(2024, 2, 10, tz="UTC"),
}

example_dag = DAG(
    'airflow-pipeline-demo',
    default_args=default_args,
    schedule_interval='@daily',
    catchup=False,
    is_paused_upon_creation=False
)

ingest_step1 = CDEJobRunOperator(
    connection_id='cde-vc01-dev',
    task_id='ingest',
    retries=3,
    dag=example_dag,
    job_name='etl-ingest-job'
)

prep_step2 = CdeRunJobOperator(
    task_id='data_prep',
    job_name='insurance-claims-job'
)
```

```
ingest_step1 >> prep_step2
```

Here are some examples of things you can define in the DAG file:

CDE job run operator

Use `CDEJobRunOperator` to specify a CDE job to run. This job must already exist in the virtual cluster specified by the `connection_id`. If no `connection_id` is specified, CDE looks for the job in the virtual cluster where the Airflow job runs.

```
from cloudera.cdp.airflow.operators.cde_operator import CDEJobRunOperator
...
ingest_step1 = CDEJobRunOperator(
    connection_id='cde-vc01-dev',
    task_id='ingest',
    retries=3,
    dag=example_dag,
    job_name='etl-ingest-job'
)
```

Email Alerts

Add the following parameters to the DAG `default_args` to send email alerts for job failures or missed service-level agreements or both.

```
'email_on_failure': True,
'email': 'abc@example.com',
'email_on_retry': True,
'sla': timedelta(seconds=30)
```

Task dependencies

After you have defined the tasks, specify the dependencies as follows:

```
ingest_step1 >> prep_step2
```

For more information on task dependencies, see [Task Dependencies](#) in the Apache Airflow documentation.

For a tutorial on creating Apache Airflow DAG files, see the [Apache Airflow documentation](#).

2. Create a CDE job.

- In the Cloudera Data Platform (CDP) console, click the Data Engineering tile. The CDE Home page displays.
- In the left navigation menu click Jobs. The Jobs page is displayed.
- Click Create Job. The Job Details page is displayed.
- Select the Airflow job type.
- Name: Provide a name for the job.
- DAG File: Use an existing file or add a DAG file to an existing resource or create a resource and upload it.
 - Select from Resource: Click Select from Resource to select a DAG file from an existing resource.
 - Upload: Click Upload to upload a DAG file to an existing resource or to a new resource that you can create by selecting Create a resource from the Select a Resource dropdown list. Specify the resource name and upload the DAG file to it.



Note: You must configure the Configure Email Alerting option while creating a virtual cluster to send your email alerts. For more information about configuring email alerts, see [Creating virtual clusters](#).

You can add the email alert parameters to the DAG `default_args` to get email alerts for job failures and missed service-level agreements. An example of email alert configurations is listed in *Step 1*.

3. Click Create and Run to create the job and run it immediately, or click the dropdown button and select Create to create the job.

Related Information

[Provider packages](#)

[Managing Cloudera Data Engineering job resources using the CLI](#)

Creating an Airflow DAG using the Pipeline UI

With the CDE Pipeline UI, you can create multi-step pipelines with a combination of available operators.

About this task



Note: Cloudera supports all major browsers (Google Chrome, Firefox and Safari) for this feature. If you are using a browser in incognito mode, you have to allow all cookies in your browser settings so that you can view Pipelines, Spark, and Airflow pages.

Procedure

1. Go to **Jobs Create Job**.
Under Job details, select **Airflow**.
The UI refreshes, only Airflow-specific options remain.
2. Specify a name for the job.
3. Under DAG File select the **Editor** option.
4. Click **Create**.
You are redirected to the job Editor tab.
5. Build your Airflow pipeline.
 - Drag and drop operators to the canvas from the left hand pane.
 - When selecting an operator, you can configure it in the editor pane that opens up.
On the **Configure** tab you can provide operator-specific settings. The **Advanced** tab allows you to make generic settings that are common to all operators, for example execution timeout or retries.
 - Create dependencies between tasks by selecting them and drawing an arrow from one of the four nodes on their edges to another task. If the dependency is valid the task is highlighted in green. If invalid, it is highlighted in red.
 - To modify DAG-level configuration, select **Configurations** on the upper right.
6. When you are done with building your pipeline, click **Save**.

Managing an Airflow Pipeline using the CDE CLI

Based on your business requirement, you can use Cloudera Data Engineering (CDE) CLI to create basic Airflow pipelines or multi-step pipelines with a combination of available operators, to enable data-driven decisions. You can update these Airflow pipelines by updating the DAG files and job configurations.

Creating a pipeline using the CDE CLI

You can update the following properties in an Airflow pipeline:

Related Information

[Using the Cloudera Data Engineering CLI](#)

Creating a basic Airflow pipeline using CDE CLI

By creating a basic pipeline in Cloudera Data Engineering (CDE) using the CLI, you can create multi-step pipelines with a combination of available operators.

About this task

To create a basic pipeline in CDE, you must upload the Airflow (Directed Acyclic Graph) DAG to a CDE resource and create a CDE Airflow job from this DAG.

Procedure

In the CDE CLI, run the following command:

```
cde resource create --name my_pipeline_resource
cde resource upload --name my_pipeline_resource --local-path my_pipeline_dag
.py
cde job create --name my_pipeline --type airflow --dag-file my_pipeline_dag.
py --mount-1-resource my_pipeline_resource
```

Related Information

[Using the Cloudera Data Engineering CLI](#)

Creating a pipeline with additional Airflow configurations using CDE CLI

By creating a pipeline with additional Airflow configurations using the Cloudera Data Engineering (CDE) CLI, you can create multi-step pipelines with a combination of available operators. There are two ways to create this type of pipeline. The first method detailed below is recommended approach that we highly suggest customers use. The second is the alternative method that customers have used in the past, but is not recommended.

About this task

Airflow DAGs can be defined with parameters at the DAG-level or Task-level. These parameters can be overridden in the case of a manual run. A manual run is triggered explicitly by the user. It is recommended to use the Params approach so that default values can be used by the scheduled job instances as well.

For Params (Recommended)

An example of a DAG definition with additional Airflow configuration is as follows:

1. Create a configuration such as the example shown below:

```
from airflow import DAG
from airflow.models.param import Param

with DAG(
    "my_dag",
    params={
        # an int with a default value
        "int_param": Param(10, type="integer", minimum=0, maximum
=20),

        # a required param which can be of multiple types
        # a param must have a default value
        "dummy": Param(5, type=["null", "number", "string"]),

        # an enum param, must be one of three values
        "enum_param": Param("foo", enum=["foo", "bar", 42]),

        # a param which uses json-schema formatting
```

```

        "email": Param(
            default="example@example.com",
            type="string",
            format="idn-email",
            minLength=5,
            maxLength=255,
        ),
    },
):
    # prints <class 'str'> by default
    # prints <class 'int'> if render_template_as_native_obj=True
    my_operator = PythonOperator(
        task_id="template_type",
        op_args=[
            "{{ params.int_param }}"
        ],
        python_callable=(
            lambda x: print(type(x))
        ),
    )

```

In this case, nothing needs to be done on the cde job create step. Values can be additionally overridden in a manual run, through the `--config` flag of the `cde job run` command. For example:

```
cde job run --name my_pipeline --config key1=my_new_value1
```

For Dag run conf (Not recommended)



Note: This is an alternative method to create a pipeline with additional Airflow configurations, but it is not recommended. This method does not work with scheduled job instances.

For historical reasons CDE supports the `{{ dag_run.conf }}` object as well. In this case, the option, `--config key=value` in the `cde job create` command, is used to define default values whenever the user triggers a manual run using `cde job run` without specifying these parameters in the run command. This config option can be repeated to define multiple parameters.

1. Create a configuration such as the example shown below:

```

cde resource create --name my_pipeline_resource
cde resource upload --name my_pipeline_resource --local-path my_pipeline_dag.py
cde job create --name my_pipeline --type airflow --dag-file my_pipeline_dag.py --mount-1-resource my_pipeline_resource --config key1=value1 --config key2=value2

```

The configuration can be used in a DAG as shown below:

```

my_bash_task = BashOperator(
    task_id="my_bash_task",
    bash_command="echo key1_value: {{ dag_run.conf['key1'] }} key2_value: {{ dag_run.conf['key2'] }}",
    dag=dag,
)

```

The configuration can also be overridden for manual runs in the same manner as described in the Recommended section on this page.

Related Information

[Params](#)

[Using the Cloudera Data Engineering CLI](#)

Creating an Airflow pipeline with custom files using CDE CLI [technical preview]

By creating a pipeline in CDE using the CLI, you can add custom files that are available for tasks. This is a technical preview.

Before you begin

This feature is available in CDE 1.19 and above in new Virtual Cluster installations only.

About this task

For use cases where custom files need to be accessed within an Airflow task, you need to first upload the custom files to a CDE resource, and then specify it in the job creation parameter using the `--airflow-file-mount-<n>-resource` option. These files are available only to the jobs in which they are linked.

Procedure

Run the following commands to upload the custom files to a CDE resource, and then create the job:

```
cde resource create --name my_pipeline_resource
cde resource upload --name my_pipeline_resource --local-path my_pipeline_dag.py
cde resource create --name my_file_resource
cde resource upload --name my_file_resource --local-path my_file.conf

cde job create --name my_pipeline --type airflow --dag-file my_pipeline_dag.py --mount-1-resource my_pipeline_resource --airflow-file-mount-1-resource my_file_resource
```

Example

The files can be reached in Airflow DAGs with the following pattern: `/app/mount/<resource_name> resource_alias/<file_name>`, like in the following example:

```
read_conf = BashOperator(
    task_id=read_conf,
    bash_command="cat /app/mount/my_file_resource/my_file.conf"
)
```



Note: It is possible to change the mount path by specifying the `--airflow-file-mount-N-prefix` `my_custom_prefix` option in the job creation command, like in the following example:

```
cde job create --name my_pipeline --type airflow --dag-file my_pipeline_dag.py --mount-1-resource my_pipeline_resource --airflow-file-mount-1-resource my_file_resource --airflow-file-mount-1-prefix my_custom_prefix
```

In this case, the file is available at:

```
read_conf = BashOperator(
    task_id=read_conf,
    bash_command="cat /app/mount/my_custom_prefix/my_file.conf"
)
```



Note: As a best practice, Cloudera recommends to use the same resource name because it is simpler to follow the DAG without having to look at the job definition. Also, it is possible to use `"/` as a value to mount to `/app/mount` if there is only one Airflow file mounted in the job definition; however, this is not recommended.

Related Information

[Using the Cloudera Data Engineering CLI](#)

Updating a pipeline using the CDE CLI

You can update the following properties in an Airflow pipeline:

Updating a DAG file using the CDE CLI

You can update a Directed Acyclic Graph (DAG) file using the CDE CLI for instances where the DAG needs to be overridden. For use cases where the DAG needs to be overridden, first the DAG needs to be uploaded to the resource to override the previous version, then you must update the job.

About this task

Unlike a Spark job, the Airflow job does not automatically pull in the updated resource. Airflow jobs require a forced update by calling the `job update` command, such that the required files are uploaded to Airflow server for processing.

Choose one of the following options in step 1:



Important: Before updating a DAG, it is recommended to ensure the DAG is paused and is not currently running when the update is being done.

Updating a DAG file that needs to be overridden

Run the following command in the CDE CLI:

```
cde resource upload --name my_pipeline_resource --local-path my_pipeline_dag.py
cde job update --name my_pipeline --dag-file my_pipeline_dag.py --mount-1-resource my_pipeline_resource
```

Updating a DAG in scenarios where the DAG must be set to a different one:

First upload the DAG to any resource and then the job needs to be updated. Run the following command in the CDE CLI:

```
cde resource upload --name my_other_pipeline_resource --local-path my_other_pipeline_dag.py
cde job update --name my_pipeline --dag-file my_other_pipeline_dag.py --mount-1-resource my_other_pipeline_resource
```

Updating the Airflow job configurations using the CDE CLI

In the case where the Airflow job was created with the `--config` option, the Airflow job configuration can be updated with the following command below. For more information, see [Creating a pipeline using the CDE CLI](#) linked below.

Procedure

Run the following command in the CDE CLI:

```
cde job update --name my_pipeline --config key1=new_value1 --config key2=new_value2
```

The new configuration is merged with the existing job configuration.

Related Information

[Creating a pipeline using the CDE CLI](#)

Updating the Airflow file mounts using the CDE CLI [technical preview]

You can update or delete an existing file mount, or add new Airflow file mounts for your pipeline with these commands.

Changing an existing file mount

```
cde job update --name my_pipeline --airflow-file-mount-1-resource my_new_pipeline_resource
```

Changing a file mount prefix

```
cde job update --name my_pipeline --airflow-file-mount-1-prefix my_new_pipeline_resource_prefix
```

Adding a new file mount

Add a new file mount when one already exists:

```
cde job update --name my_pipeline --airflow-file-mount-2-resource my_new_pipeline_resource
```

Removing an existing file mount

```
cde job update --name my_pipeline --unset-airflow-file-mount-index 2
```



Note: You can only delete one mount at a time. After the mount is deleted, they will be re-indexed.

Deleting an Airflow pipeline using the CDE CLI

You can delete a pipeline in CDE using the CLI.

Procedure

To delete a pipeline, give the job name to the delete command:

```
cde job delete --name my_pipeline
```

Using CDE with an external Apache Airflow deployment

Before you begin

The Cloudera [provider](#) for Apache Airflow, available at the [Cloudera GitHub repository](#), provides an Airflow operator for running Cloudera Data Engineering (CDE) jobs. You can install the provider on your existing Apache Airflow deployment to integrate.



Important: CDE on CDP Private Cloud currently supports only the CDE job run operator.

- The Cloudera provider for Apache Airflow is for use with existing Airflow deployments. If you want to use the embedded Airflow service provided by CDE, see [Automating data pipelines with CDE using Apache Airflow](#).

- The provider requires Python 3.6 or higher.
- The provider requires the Python cryptography package version 3.3.2 or higher to address CVE-2020-36242. If an older version is installed, the plugin automatically updates the cryptography library.

About this task

This component provides an Airflow Operator `CDEJobRunOperator` to be integrated in your DAGs. The `CDEJobRunOperator` is for running Cloudera Data Engineering jobs.

Procedure

Install Cloudera Airflow provider on your Airflow servers

1. Run the following pip command on each Airflow server: `pip install cloudera-airflow-provider`

Create a connection using the Airflow UI

Before you can run a CDE job from your Airflow deployment, you must configure a connection using the Airflow UI.

2. In the Cloudera Data Platform (CDP) console, click the Data Engineering tile. The CDE Home page displays.
3. Click Administration in the left navigation menu. The Administration page displays.
4. In the Virtual Clusters column, click the Cluster Details icon.
5. Click JOBS API URL to copy the URL.
6. Go to your Airflow web console (where you installed the Cloudera provider).
7. Go to Admin Connection .
8. Click + Add a new record.
9. Fill in connection details:

Conn Id

Create a unique connection identifier.

Conn Type

The type of the connection. From the drop-down, select

- HTTP (if you are using Apache Airflow version 1)
- HTTP or Cloudera Data engineering (if you are using Apache Airflow version 2)

Host/Virtual API Endpoint

URL of the host where you want the job to run. Paste here the JOBS API URL you copied in a previous step.

Login/CDP Access Key

Provide the CDP access key of the account for running jobs on the CDE VC.

Password/CDP Private Key

Provide the CDP private key of the account for running jobs on the CDE VC.

10. Click Save.

11. In the CDE Home page, click Jobs in the left navigation menu, and then click Create Job.

12. Fill in the Job Details:**Job Type**

Select the option matching your use case.

Name

Specify a name for the job.

DAG File

Provide a DAG file.

Use the `CDEJobRunOperator` to specify a CDE job to run. The job definition in the DAG file must contain:

connection_id

The Conn Id you specified on the Airflow UI when creating the connection.

task_id

The ID that identifies the job within the DAG.

dag

The variable containing the dag object

job_name

The name of the CDE job to run. This job must exist in the CDE virtual cluster you are connecting to.

For example:

```
from cloudera.cdp.airflow.operators.cde_operator import CDEJobRunOperator
t1 = CDEJobRunOperator(
    connection_id='cde-vc01-dev',
    task_id='ingest',
    dag=example_dag,
    job_name='etl-ingest-job'
)
```

13. Click Create and Run to create the job and run it immediately, or click the dropdown button and select Create to create the job.

Supporting Airflow operators and hooks

Apache Airflow Python DAG files can be used to automate workflows or data pipelines in Cloudera Data Engineering (CDE). CDE currently supports a specified list of Airflow operators and hooks that can be used.

Airflow operators

CDE supports the following Airflow operators:

- `airflow.operators.bash`
- `airflow.operators.branch`
- `airflow.operators.datetime`
- `airflow.operators.dummy`
- `airflow.operators.email`
- `airflow.operators.generic_transfer`
- `airflow.operators.latest_only`
- `airflow.operators.python`
- `airflow.operators.sql`

- `airflow.operators.subdag`
- `airflow.operators.trigger_dagrun`

Airflow hooks

CDE supports the following Airflow hooks:

- `airflow.hooks.filesystem`
- `airflow.hooks.subprocess`

Related Tasks

[Automating data pipelines using Apache Airflow in Cloudera Data Engineering](#)

Creating a custom Airflow Python environment (Technical Preview)

To manage job dependencies, Cloudera Data Engineering (CDE) supports creating custom Python environments dedicated to Airflow using the `airflow-python-env` resource type. With this option, you can install custom libraries for running your Directed Acyclic Graphs (DAGs). The supported version is Python 3.8.

A resource is a named collection of files or other resources referenced by a job. The `airflow-python-env` resource type allows you to specify a `requirements.txt` file that defines an environment that you can then activate globally for airflow deployments in a virtual cluster. You can specify any Python package which is compatible with the Airflow python constraints. These constraints can be found at [https://raw.githubusercontent.com/apache/airflow/constraints-\\${AIRFLOW_VERSION}/constraints-\\${PYTHON_VERSION}.txt](https://raw.githubusercontent.com/apache/airflow/constraints-${AIRFLOW_VERSION}/constraints-${PYTHON_VERSION}.txt). The Airflow and Python versions depend on your CDE version.



Important:

Deprecation notice for Spark Python environments

For improved maintainability and extendable API, the following change is made in CDE Resource API:

- For all `/resources` endpoint, `pythonEnv.pyPiMirror` is deprecated. You must use `pythonEnv.pipRepository.url` instead.

Related Information

[Resource](#)

[requirements.txt](#)

Creating a custom Airflow Python environment resource

Create the custom Airflow Python environment resource. You can specify the `pip` repositories if required.

Before you begin

Download and configure the CDE CLI.

Procedure

Create a custom Airflow Python environment resource.

```
cde resource create --name $RESOURCE_NAME --type airflow-python-env
```

For example:

```
cde resource create --name airflow-1 --type airflow-python-env
```

- [Optional] You can specify the custom pip repository using the `--pip-repository-url <custom-pip-repository-url> --pip-repository-cert <path-to-pem-file>` option in the create resource command.
- [Optional] You can specify one or more extra custom pip repositories using the `--extra-pip-repository-<number>-url --<custom-pip-repository-url>-<number>-cert <path-to-pem-file>` option in the create resource command. You can specify up to 10 extra pip repositories.

Example of command with pip repository and extra pip repository:

```
cde resource create --name airflow-custom-pip-repos --type airflow-python-env --pip-repository-url https://pypi.example.com/simple --pip-repository-cert cert.pem --extra-pip-repository-1-url https://extra-pypi.example.com/simple --extra-pip-repository-1-cert extra-cert.pem
```

CDE builds the environment according to the requirements.txt file. During this build time, you cannot run a job associated with the environment. You can check the status of the environment by running the `cde resource list-events --name $RESOURCE_NAME` command.



Note: You can use specific credentials for your custom pip repositories. For more information, see [Using credentials for custom pip repositories](#).

CDE CLI custom Airflow Python environment flag reference

You use the following optional flags when creating an Airflow Python environment.

Table 1:

Flag	Description
<code>--pip-repository-url</code>	Index URL of the pip repository to override the default public pip repository for the <code>python-env</code> or <code>airflow-python-env</code> resource. This option maps to the <code>--index-url</code> flag of the pip install process. If not specified, the public pip repository is used.
<code>--pip-repository-cert</code>	Certificate file associated with the pip repository for the <code>python-env</code> or <code>airflow-python-env</code> resource.
<code>--pip-repository-cred</code>	CDE credential name used to authenticate against the pip repository for the <code>python-env</code> or <code>airflow-python-env</code> resource.
<code>--pip-repository-skip-cert-validation</code>	Skips the certificate validation for the pip repository for the <code>python-env</code> or <code>airflow-python-env</code> resource.
<code>--extra-pip-repository-N-url</code>	Index URL of the N-th extra pip repository for the <code>python-env</code> or <code>airflow-python-env</code> resource. You can specify up to 100 extra repositories. Maps to the <code>extra-index-url</code> field of the pip install command.
<code>--extra-pip-repository-N-cert</code>	Certificate file associated with the N-th extra pip repository for the <code>python-env</code> or <code>airflow-python-env</code> resource.
<code>--extra-pip-repository-N-cred</code>	CDE credential name used to authenticate against the N-th extra pip repository for the <code>python-env</code> or <code>airflow-python-env</code> resource.
<code>--extra-pip-repository-N-skip-cert-validation</code>	Skips the certificate validation for the N-th extra pip repository for the <code>python-env</code> or <code>airflow-python-env</code> resource.

Uploading the resource to build the Python environment

After you create the resource, you have to upload the resource to build your environment.

Before you begin

Create a requirements.txt file specifying the Python package and version dependencies required by your CDE job.

Procedure

Upload the requirements.txt file to the resource.

```
cde resource upload --name $RESOURCE_NAME --local-path requirements.txt
```

Results

This launches a job that builds the Python environment from the requirement.txt file. You can check the status of this job using the `cde resource list-events --name $RESOURCE_NAME` command. The Python resource must be in the Ready state after you upload and before you activate it.

Activating Python environment resources

Airflow-related deployments must be configured to use the new Python environment and have to be restarted to use the newly created Python environment.



Note: Only one Python environment can be active at one time. This command restarts the Airflow services. You must ensure that no jobs are running when activating the Python environment.

Before you begin

The Python resource must be in the Ready state after you upload. You can check the resource status using the `cde resource list-events --name $RESOURCE_NAME` command.

Procedure

After you upload the resource, activate the environment to use it in the Airflow jobs.

```
cde airflow activate-pyenv --pyenv-resource-name $RESOURCE_NAME
```

You can check if the resource is activated using the `cde airflow get-active-pyenv` command.

Resetting to the default Airflow Python environment

You can reset the Python environment to the default Airflow Python environment resources.



Note: This resets the Airflow services. You must ensure that no jobs are running when resetting the Python environment.

Procedure

Reset your Python environment to use the default environment.

```
cde airflow reset-pyenv
```

Deleting Airflow Python environment resources

You can delete the custom Python environment resources when it is not active and if not needed.

Procedure

Delete the custom Airflow Python environment resource.

```
cde resource delete --name $RESOURCE_NAME
```