Cloudera Data Engineering 1.4.1

Accessing the Cloudera Data Engineering service using the CLI

Date published: 2020-07-30
Date modified: 2022-11-18



https://docs.cloudera.com/

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

Using the Cloudera Data Engineering command line interface.....5

Downloading the Cloudera Data Engineering command line interface..... 5

Configuring the CLI client..... 5
Cloudera Data Engineering CLI configuration options..... 6
Cloudera Data Engineering CLI authentication..... 7
Cloudera Data Engineering CLI TLS configuration..... 9

CDE concepts..... 10

Managing Cloudera Data Engineering job resources using the CLI.....10
Creating a Cloudera Data Engineering resource using the CLI..... 11
Uploading files or other assets to a Cloudera Data Engineering resource using the CLI..... 12
Deleting a Cloudera Data Engineering resource using the CLI..... 13
Creating and updating Docker credentials..... 14
Deleting Docker credentials..... 14
Deleting an Airflow DAG..... 15

Managing Cloudera Data Engineering jobs using the CLI.....15
Creating and updating Apache Spark jobs using the CLI..... 15
Creating and updating Apache Airflow jobs using the CLI..... 16
Listing jobs using the CLI..... 17
Submitting a Spark job using the CLI..... 17
Running raw Scala code in Cloudera Data Engineering..... 18
Submitting an Airflow job using the CLI..... 18
Running a Spark job using the CLI..... 18
Running a Airflow job using the CLI..... 19

Scheduling Spark jobs.....20
Enabling, disabling, and pausing scheduled jobs..... 22
Managing the status of scheduled job instances..... 22

CDE Spark job example..... 23

CDE CLI command reference..... 24

CDE CLI Spark flag reference..... 25

CDE CLI Airflow flag reference.....26

CDE CLI list command syntax reference.....26

Using the Cloudera Data Engineering command line interface

Cloudera Data Engineering (CDE) provides a command line interface (CLI) client. You can use the CLI to create and update jobs, view job details, manage job resources, run jobs, and so on.

Note: The CLI client is not forward compatible. Download the client for the version of the cluster you are accessing. The Cluster Details page for every virtual cluster includes a link to download the CLI client for that cluster version.

The CLI client can also use a password file for non-interactive uses, such as automation frameworks.

Related Information

Using CLI-API to Automate Access to Cloudera Data Engineering

Using Cloudera Data Engineering CLI

Downloading the Cloudera Data Engineering command line interface

Cloudera Data Engineering (CDE) provides a command line interface (CLI) client.

In addition to the CDE API, you can use the CDE CLI client to access your CDE service. Using the CLI, you can manage clusters and applications.

Note: The CLI client is not forward compatible. Download the client for the version of the cluster you are accessing. The Cluster Details page for every virtual cluster includes a link to download the CLI client for that cluster version.

To download the CLI client:

1. Navigate to the Cloudera Data Engineering Overview page by clicking the Data Engineering tile in the Cloudera Data Platform (CDP) management console.
2. In the CDE web console, select an environment.
3. Click the Cluster Details icon for the virtual cluster you want to access.
4. Click the link under CLI TOOL to download the CLI client.

Configuring the CLI client

The CDE CLI client uses a configuration file, ~/.cde/config.yaml, to define the default CDE virtual cluster to interact with, as well as other configuration parameters.

Before you begin

Make sure that you have downloaded the CDE CLI client.

Procedure

1. Determine the virtual cluster endpoint URL.
   a) Navigate to the Cloudera Data Engineering Overview page.
   b) In the Environments column, select the environment containing the virtual cluster you want to access using the CLI.
   c) In the Virtual Clusters column on the right, click the Cluster Details icon for the virtual cluster you want to access.
   d) Click JOBS API URL to copy the URL to your clipboard.

   Note: Currently, the URL copied to your clipboard begins with http://, not https://. To use the URL, you must manually change this to https://.

2. On the host with the CLI client, create or edit the configuration file at ~/.cde/config.yaml.

   Note: You can use a custom file location by setting the CDE_CONFIG environment variable.

3. In the configuration file, specify the CDP user and virtual cluster endpoint as follows:

   ```
   user: <CDP_USER>
   vcluster-endpoint: <CDE_VIRTUAL_CLUSTER_ENDPOINT>
   ```

   Important: The CLI in this release does not support TLS validation. You must disable TLS validation by adding the following lines to the CDE configuration file:

   ```
   tls-insecure: true
   ```

   The connection still uses HTTPS, but the TLS certificate is not validated.

   The CDP user is your workload username.
4. Save the configuration file.
5. If you have not done so already, make sure that the cde file is executable by running chmod +x */PATH/TO/*cde.
6. Run cde job list to verify your configuration. Enter your workload password when prompted.

   Note: If the directory containing the cde file is not part of your PATH environment variable, you can either add it to your PATH environment variable or use the full path when running the command.

   You can also configure the CLI to use an access token so that you do not need to enter your password each time. For more information, see Cloudera Data Engineering CLI authentication.

What to do next

See CDE CLI configuration options for other configuration options.

Cloudera Data Engineering CLI configuration options

The Cloudera Data Engineering (CDE) CLI can be configured using a configuration file, environment variables, or by command flags.

Configuration Option	Configuration File (~/.cde/config.yaml)	Environment Variable	Command Flag
User	user: <*USERNAME*>	CDE_USER=<*USER*>	--user <*USERNAME*>
Credentials file	credentials-file: </path/to/credentials>	CDE_CREDENTIALS_FILE=</path/to/credentials>	--credentials-file </path/to/credentials>
Skip credentials file detection	skip-credentials-file: true	CDE_SKIP_CREDENTIALS_FILE=true	--skip-credentials-file

Configuration Option	Configuration File (~/.cde/config.yaml)	Environment Variable	Command Flag
Password file	auth-pass-file: *<PASSWORD_FILE>*	CDE_AUTH_PASS_FILE=*<PASSWORD_FILE>*	--auth-pass-file *<PASSWORD_FILE>*
Virtual cluster endpoint	vcluster-endpoint: *<VIRTUAL_CLUSTER>*	CDE_VCLUSTER_ENDPOINT=*<VIRTUAL_CLUSTER>*	--vcluster-endpoint *<VIRTUAL_CLUSTER>*
Disable authentication token caching	auth-no-cache: true	CDE_AUTH_NO_CACHE=true	--auth-no-cache
Authentication token cache file	auth-cache-file: *<TOKEN_CACHE_FILE>*	CDE_AUTH_CACHE_FILE=*<TOKEN_CACHE_FILE>*	--auth-cach-file *<TOKEN_CACHE_FILE>*

Cloudera Data Engineering CLI authentication

The Cloudera Data Engineering (CDE) CLI tool supports both interactive and transparent authentication. For interactive authentication, if you have configured the CLI with your workload username, you are prompted for a password. For transparent authentication, the CDE CLI supports a password file, Cloudera Data Platform (CDP) access keys, and CDP credentials file.

The CDE CLI provides the following mechanisms for authentication:

- CDP access key stored in a credentials file
- CDP access key specified by CLI flag or environment variable
- Interactive prompt for workload password
- Workload password specified by CLI flag or environment variable

In all cases, the CLI uses the provided credentials to obtain an authentication token for the specified user, and caches it locally in a file on the machine where the CLI is running. You can disable caching of tokens entirely by using the --auth-no-cache CLI flag or the CDE_AUTH_NO_CACHE environment variable.



Important: The minimum required roles to obtain an access token are *DEUSER* and *ENVIRONMENTUSER*. *ENVIRONMENTADMIN* role is not required.

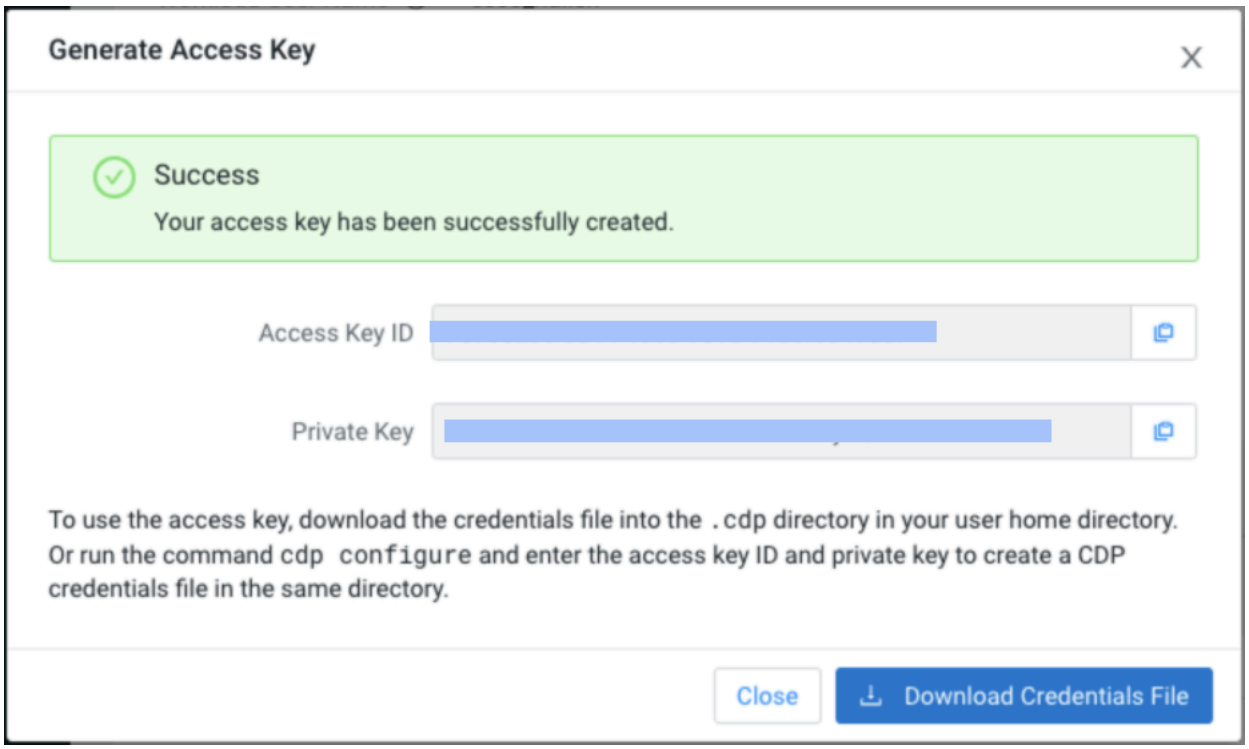
The cache file location is automatically determined based on the default system user cache:

- Linux: $HOME/.cache/cloudera/cde or $XDG_CACHE_HOME/cloudera/cde/
- macOS: $HOME/Library/Caches/cloudera/cde/
- Windows: %LocalAppData%\cloudera\cde\

If you want to use a custom location, specify it with the --auth-cache-file flag or the CDE_AUTH_CACHE_FILE environment variable. You can use the special string $USERCACHE, which is expanded according to the default system user cache (as listed above, without the /cloudera/cde/ suffix).

CDP credentials file

When you generate a CDP access key, you can download it to a credentials file:



The access key is only displayed and available for download when you first generate it. After you close the dialog, there is no way to recover the key.

Save or copy the credentials file to $HOME/.cdp/credentials on the machine where you are running the CDE CLI. Credentials stored in this file are automatically discovered by both the CDE and CDP CLIs. If a credentials file is found, authentication occurs transparently using the discovered CDP access key.

The CDE CLI automatically looks for a CDP access key in the following locations in the order given:

1. ./credentials
2. $HOME/.cde/credentials
3. /etc/cde/credentials
4. $HOME/.cdp/credentials

You can override this by using the --credentials-file *</PATH/TO/CREDENTIALS>* CLI flag to specify a different file location.

You can also skip credential discovery by using the --skip-credentials-file flag.

CDP access key

If you do not want to use the credentials file, you can specify the access key using environment variables or command line flags as follows:

Table 1: CDP access key environment variables and CLI flags

Parameter	Environment variable	CLI flag
Access key ID	CDE_ACCESS_KEY_ID=*<ACCESS_KEY_ID>*	--access-key-id *<ACCESS_KEY_ID>*
Access key secret	CDE_ACCESS_KEY_SECRET=*<ACCESS_KEY_SECRET>*	--access-key-secret string *<ACCESS_KEY_SECRET>*

Along with the above flags, CDE CLI expects CDP endpoint URL to be configured. CDP Endpoint URL is same as the CDP private cloud console URL. You can configure this using environment variables or command line flags as follows:

Table 2: CDP endpoint environment variables and CLI flags

Parameter	Environment variable	CLI flag
CDP Endpoint	CDE_CDP_ENDPOINT=<*CDP_ENDPOINT*>	--cdp-endpoint <*CDP_ENDPOINT*>

Workload password prompt

When the CLI requires a new token for a virtual cluster, you are prompted for the password for the workload user, identified by the --user CLI flag or the CDE_USER environment variable.

The workload password, for both human and machine users, can be set using the CDP User Management console. For more information, see Managing user access and authorization.

Workload password file

If you do not want to be prompted for your workload password, you can provide a password file. A password file is a file containing your workload password, and nothing else.

Note: When using a password file, the CLI strips one trailing newline character. If your password actually includes a newline character at the end, add an extra newline at the end of the file.

You can specify the password file by using an environment variable or a command line flag as follows:

Environment variable

CDE_AUTH_PASS_FILE=<*/PATH/TO/PASSWORD/FILE*>

Command line flag

--auth-pass-file <*/PATH/TO/PASSWORD/FILE*>

Cloudera Data Engineering CLI TLS configuration

Important: The CLI in this release does not support TLS validation. You must disable TLS validation by adding the following lines to the CDE configuration file (~/.cde/config.yaml):

```
tls-insecure: true
```

All CDE virtual cluster endpoints are configured with TLS. In non-production or on-premises environments the TLS certificates are usually signed by a non-production or non-public certificate authority (CA). In these cases, without additional configuration, the CLI tool fails as it attempts to validate the API server's TLS certificate. The CLI provides a TLS configuration when using non-public/non-production CAs.

Specify a file containing the PEM encoded public certificate(s) of the signing CA in one of the following ways:

- add the --tls-ca-certs *[***/PATH/TO/CA.PEM***]* flag on the command line
- define the tls-ca-certs: *[***/PATH/TO/CA.PEM***]* variable in the ~/.cde/config.yaml configuration file
- set the CDE_TLS_CA_CERTS environment variable

Replace *[***/PATH/TO/CA.PEM***]* with the path to a valid ca.pem file.

For public cloud, certificates are issued and signed by LetsEncrypt:

Note: LetsEncrypt Production CA Chain is part of the standard CA bundle therefore you do not need to add it on Linux or macOS. It is however, mandatory on Windows, where you have to concatenate the following into a single CA file:

- https://letsencrypt.org/certs/lets-encrypt-x3-cross-signed.pem.txt
- https://letsencrypt.org/certs/trustid-x3-root.pem.txt

For internal or on-premises environments you need to obtain your CA certificates through your internal process.

Note:

If using the CLI on Windows, ensure you use path styles such as C:\Users\janeblogs\.cde\ca.pem when referencing local files.

CDE concepts

Learn about some basic concepts behind Cloudera Data Engineering (CDE) service to better understand how you can use the command line interface (CLI).

CDE has three main concepts:

job

A 'job' is a definition of something that CDE can run. For example, the information required to run a jar file on Spark with specific configurations.

job run

A 'job run' is an execution of a job. For example, one run of a Spark job on a CDE cluster.

resource

A 'resource' refers to a job dependency that must be available to jobs at runtime. Currently the following resource types are supported:

- files is a directory of files that you can upload to CDE pods into a standard location (/app/mount). This is typically for application (for example, .jar or .py files) and reference files, and not the data that the job run will operate on. Multiple files resources can be referenced in a single job.
- python-env is used to provide custom Python dependencies to the job as a Python virtual environment which is automatically configured. Up to one python-env resource can be specified per job definition.

In addition, to support jobs with custom requirements, CDE also allows users to manage credentials which can be used at job run time. Currently, only custom Docker registry credentials are supported.

Submitting versus running a job

The cde spark submit and cde airflow submit commands automatically create a new job and a new resource, submit the job as a job run, and when the job run terminates they delete the job and resources.

A cde job run requires a job and all necessary resources to be created and uploaded to the CDE cluster beforehand. The advantage of creating resources and jobs ahead of time is that resources can be reused across jobs, and that jobs can be run using only a job name.

Managing Cloudera Data Engineering job resources using the CLI

A *resource* in Cloudera Data Engineering (CDE) is a named collection of files or other resources referenced by a job. The files can include application code, configuration files, or any other arbitrary files required by a job. A resource can also be a Python virtual environment, or a custom Docker container image.

Note: Custom Docker container images is a *Technical Preview* feature. Contact your Cloudera account representative to enable access to this feature.

You can think of resources as any supporting files, libraries, or images that a CDE job requires to run. Resources can be created and deleted, and files can be added to and deleted from a resource as needed.

A resource can also be a Python virtual environment specification (as a requirements.txt file), or a custom Docker container image.

Before continuing, make sure that you have downloaded and configured the CLI client.

Creating a Cloudera Data Engineering resource using the CLI

A *resource* in Cloudera Data Engineering (CDE) is a named collection of files or other assets referenced by a job, including application code, configuration files, or any other arbitrary files required by a job. A resource can also be a Python virtual environment, or a custom Docker container image.

Before you begin

Note: Custom Docker container images is a *Technical Preview* feature. Contact your Cloudera account representative to enable access to this feature.

Make sure that you have downloaded and configured the CLI client.

Procedure

1. Create a resource using the cde resource create command.

   The cde resource create syntax is as follows:

   ```
   cde resource create [flags]
   ```

   You can view the list of flags by running cde resource create --help, or you can view the CDE CLI reference documentation.

   Example: Create a file resource

   ```
   cde resource create --name cde-file-resource --type files
   ```

   Example: Create a Python virtual environment resource

   ```
   cde resource create --name cde-python-env-resource --type python-env --p
   ython-version python3
   ```

   Note:

   You can specify a PyPi mirror for a Python virtual environment resource using the --pypi-mirror flag. Note, that this requires network access to the mirror from the CDP environment.

   Example: Create a custom Docker container image resource

   ```
   cde resource create --name cde-container-image-resource --type custom-ru
   ntime-image
   ```

2. Verify that the resource was created by running cde resource list.

Uploading files or other assets to a Cloudera Data Engineering resource using the CLI

A *resource* in Cloudera Data Engineering (CDE) is a named collection of files or other assets referenced by a job, including application code, configuration files, or any other arbitrary files required by a job. A resource can also be a Python virtual environment, or a custom Docker container image.

Before you begin



Note: Custom Docker container images is a *Technical Preview* feature. Contact your Cloudera account representative to enable access to this feature.

Make sure that you have downloaded and configured the CLI client.

Make sure that you have created a resource.

Procedure

1. Upload assets to a resource using the cde resource upload command.

   The cde resource upload syntax is as follows:

   ```
   cde resource upload [flags]
   ```

   You can view the list of flags by running cde resource upload --help, or you can view the CDE CLI reference documentation.

   Note: For Python environment resources, you can only upload a requirements.txt file. Python environment resources do not support arbitrary file upload. If the local file is named something other than requirements.txt, you must add the flag --resource-path requirements.txt to the command.

   Example: Upload a file resource

   ```
   cde resource upload --name [***RESOURCE_NAME***] --local-pat
   h [***LOCAL_PATH***] [--resource-path [***PATH_IN_RESOURCE***]]
   ```

   Use repeated local path flags, and/or */?/[a-z] wildcards, to specify multiple files. Use quotes around the local path when including wildcards, for example, --local-path "*.jar". For a single file --resource-path is used for the resource filename. For multiple files --resource-path is used for the resource directory.

   Example: Upload a Python virtual environment resource

   ```
   cde resource upload --name cde-python-env-resource --local-path ${HOME}/
   requirements.txt
   ```

   Example: Upload a custom Docker container image resource

   ```
   cde resource upload --name cde-container-image-resource --type custom-ru
   ntime-image
   ```

   Example: Upload a file for a file resource

   ```
   cde resource upload --name cde-file-resource --local-path /path/to/local/
   file
   ```

   Example: Upload and extract archive to resource

   Currently supported archive file formats are : .zip and .tar.gz

   ```
   cde resource upload-archive --name cde-file-resource --local-path /path/
   to/local/file
   ```

2. Verify that the file is included in the resource by running cde resource describe --name *<RESOURCE_NAME>*.

Deleting a Cloudera Data Engineering resource using the CLI

A *resource* in Cloudera Data Engineering (CDE) is a named collection of files or other resources referenced by a job, including application code, configuration files, or any other arbitrary files required by a job. A resource can also be a Python virtual environment, or a custom Docker container image. Resources can be deleted using the CLI.

Before you begin

- Make sure that you have downloaded and configured the CLI client.
- Make sure that the resource you are deleting is no longer needed for any jobs. (Resources cannot be deleted if they are referenced in one or more jobs)

Procedure

1. Run cde resource describe --name <*RESOURCE_NAME*>. View the output and confirm that the resource you want to delete is no longer required, and does not contain any files that you need to retain.
2. Delete the resource by running cde resource delete --name <*RESOURCE_NAME*>
3. Verify that the resource is deleted by running cde resource list and confirming that the resource is no longer listed.

Creating and updating Docker credentials

To allow the use of private Docker registries, Cloudera Data Engineering (CDE) supports the creation and management of credentials. These are stored securely in the Kubernetes cluster as secrets and cannot be accessed by end users directly. Credentials are attached to job runs automatically by the CDE backend.

About this task



Note: Custom Docker container images is a *Technical Preview* feature. Contact your Cloudera account representative to enable access to this feature.

Procedure

1. To create a new Docker credential:

```
cde credential create --name <CRED_NAME> --type docker-basic --docker-se
rver <REGISTRY_URL_OR_HOSTNAME> --docker-username <DOCKER_USER>
```

2. Enter the Docker registry password when you are prompted.

   An optional --description field allows you to annotate the credential with a human readable description.
3. Run cde credential list to verify that the credential was created:

```
cde credential list [--filter <FILTER>]
```

   For more information on filtering syntax, see CDE CLI list command syntax reference on page 26.
4. If you want to update a credential, use the cde credential update command.

   This command allows you to update the secret content, the credential description, or both.

```
cde credential update --name <CRED_NAME> [--docker-serve
r <REGISTRY_URL_OR_HOSTNAME> --docker-username <DOCKER_USER>] [--descrip
tion "<DESC>"]
```

Deleting Docker credentials

To allow the use of private Docker registries, Cloudera Data Engineering (CDE) supports the creation and management of credentials. These are stored securely in the Kubernetes cluster as secrets and cannot be accessed by end users directly. Credentials are attached to job runs automatically by the CDE backend.

Before you begin

- Make sure that you have downloaded and configured the CLI client.
- Make sure that the credential you are deleting is no longer needed for any jobs.

About this task

Note: Custom Docker container images is a *Technical Preview* feature. Contact your Cloudera account representative to enable access to this feature.

Procedure

1. Delete the credential by running cde credential delete --name <*CRED_NAME*>
2. Run cde credential list to verify that the credential was deleted:

```
cde credential list [--filter <FILTER>]
```

For more information on filtering syntax, see CDE CLI list command syntax reference on page 26.

Deleting an Airflow DAG

You can delete unused Airflow DAGs using the Cloudera Data Engineering (CDE) command line interface (CLI).

About this task

The default process of removing CDE resources is to delete them together with the jobs owning them, using the cde job delete command. The cde airflow delete-dag command is a fallback for when Airflow gets into an unexpected situation and you have to remove a DAG with no associated Airflow job.

Procedure

To delete a DAG from Airflow that is not associated with a job, use the cde airflow delete-dag command:

```
cde airflow delete-dag --dag-id <DAG_ID>
```

Managing Cloudera Data Engineering jobs using the CLI

A *job* in Cloudera Data Engineering (CDE) is a definition of something that CDE can run. For example, the information required to run a JAR file on Spark with specific configurations. A 'job run' is an execution of a job. For example, one run of a Spark job on a CDE cluster.

Creating and updating Apache Spark jobs using the CLI

The following example demonstrates how to create a Spark application in Cloudera Data Engineering (CDE) using the command line interface (CLI).

Before you begin

Make sure that you have downloaded the CLI client. For more information, see Using the Cloudera Data Engineering command line interface .

Procedure

1. Run the cde job create command as follows:

```
cde job create --application-file <PATH_TO_APPLICATION_JAR> --c
lass <APPLICATION_CLASS> [--default-variable name=value] --name <JOB_NAME>
 --num-executors <NUM_EXECUTORS> --type spark
```

   To see the full command syntax and supported options, run cde job create --help.

   With [--default-variable] flags you can replace strings in job values. Currently the supported fields are:

   - Spark application name
   - Spark arguments
   - Spark configurations

   For a variable flag name=value any substring {{{name}}} in the value of the supported field gets replaced with value. These can be overriden by the [--variable] flag during the job run.

2. Run cde job describe to verify that the job was created:

```
cde job describe --name <JOB_NAME>
```

3. If you want to update the job configuration, use the cde job update command.
   For example, to change the number of executors:

```
cde job update --name test_job --num-executors 15
```

   To see the full command syntax and supported options, run cde job update --help.

4. To verify the updated configuration, run cde job describe again:

```
cde job describe --name <JOB_NAME>
```

Creating and updating Apache Airflow jobs using the CLI

The following example demonstrates how to create an Airflow DAG in Cloudera Data Engineering (CDE) using the command line interface (CLI).

Before you begin

Make sure that you have downloaded the CLI client. For more information, see Using the Cloudera Data Engineering command line interface.

About this task

Procedure

1. Run the cde job create command as follows:

```
cde job create --name <JOB_NAME> --type airflow --dag-file <DAG_FILE> --m
ount-1-resource <YOUR_DAG_RESOURCE> [OTHER AIRFLOW FLAGS...]
```

   <DAG_file>

   is a reference to a file within a CDE resource

   To see the full command syntax and supported options, run cde job create --help.



Note: Airflow DAGs manage their own schedules and so their schedules cannot be set through the CLI.

2. Run cde job describe to verify that the job was created:

```
cde job describe --name <JOB_NAME>
```

3. If you want to update the job configuration, use the cde job update command.
   For example, to change the number of executors:

```
cde job update --name test_job
```

   To see the full command syntax and supported options, run cde job update --help.

4. To verify the updated configuration, run cde job describe again:

```
cde job describe --name <JOB_NAME>
```

Listing jobs using the CLI

To view existing applications, run cde job list. To view details for a specific application, run cde job describe --name <*JOB_NAME*>

Submitting a Spark job using the CLI

The following example demonstrates how to submit a JAR or Python file to run on CDE Spark in Cloudera Data Engineering (CDE) using the command line interface (CLI).

About this task

Using the cde spark submit command is a quick and efficient way of testing a spark job, as it spares you the task of creating and uploading resources and job definitions before running the job, and cleaning up after running the job.

This command is recommended only for JAR or Python files that need to be run just once, because the the file is removed from Spark at the end of the run. To manage jobs that need to be run more than once, or that contain schedules, use cde job run instead of this command.

Procedure

To submit a JAR or Python file to run on CDE Spark, use the CLI command:

```
cde spark submit <JAR/Python file> [args...] [Spark flags...] [--job-name <j
ob name>] [--hide-logs]
```

To see the full command syntax and supported options, run cde spark submit --help.

For example:

To submit a job with a local JAR file:

```
cde spark submit my-spark-app-0.1.0.jar 100 1000 --class com.company.app.spa
rk.Main
```

The CLI displays the job run ID followed by the driver logs, unless you specified the --hide-logs option. The script returns an exit code of 0 for success or 1 for failure.

Running raw Scala code in Cloudera Data Engineering

Cloudera Data Engineering (CDE) supports running raw Scala code from the command line, without compiling it into a JAR file. You can use the cde spark submit command to run a .scala file. CDE recognizes the file as Scala code and runs it using spark-shell in batch mode rather than spark-submit.

Limitations:

- When setting the Log Level from the user interface, the setting is not applied to the raw Scala jobs.
- Do not use package <something> in the raw Scala job file as Raw Scala File is used for Scripting and not for Jar development and packaging.

Note: CDE does not currently support interactive sessions. The Scala code runs in batch mode spark-shell.

Run cde spark submit as follows to run a Scala file:

```
cde spark submit filename.scala --jar <JAR_DEPENDENCY_1> --j
ar <JAR_DEPENDENCY_2> ...
```

Submitting an Airflow job using the CLI

The following example demonstrates how to submit a DAG file to immediately run on CDE Airflow in Cloudera Data Engineering (CDE) using the command line interface (CLI).

About this task

Using the cde airflow submit command is a quick and efficient way of testing an Airflow job, as it spares you the task of creating and uploading resources and job definitions before running the job, and cleaning up after running the job.

This command is recommended only for Airflow DAGs that need to be run just once, because the DAG is removed from Airflow at the end of the run. To manage Airflow DAGs that need to be run more than once, or that contain schedules, use cde job run instead of this command.

Procedure

To submit a DAG file to run on CDE Airflow, use the CLI command:

```
cde airflow submit <DAG python file> [--config <key=value>]* [--job-name <jo
b name>]
```

To see the full command syntax and supported options, run cde airflow submit --help.

For example:

To submit a job with a local DAG file:

```
cde airflow submit my-dag.py
```

When the job has been submitted the CLI displays the job run ID, waits for the job to terminate, and returns an exit code of 0 for success or 1 for failure.

Running a Spark job using the CLI

The following example demonstrates how to run a Cloudera Data Engineering (CDE) Spark job using the command line interface (CLI).

Before you begin

Make sure that the Spark job has been created and all necessary resources have been created and uploaded.

Note: Custom Docker container images is a *Technical Preview* feature. Contact your Cloudera account representative to enable access to this feature.

About this task

Using the cde job run requires more preparation on the target environment compared to the cde spark submit command. Whereas cde spark submit is a quick and efficient way of testing a Spark job during development, cde job run is suited for production environments where a job is to be run multiple times, therefore removing resources and job definitions after every job run is neither necessary, nor viable.

Procedure

To run a Spark job, run the following command:

```
cde job run --name <job name> [Spark flags...] [--wait] [--variable name=val
ue...]
```

- With [Spark flags...] you can override the corresponding job values. Spark flags that can be repeated replace the original list, except for --conf which only adds or replaces values for the given keys.
- With [--variable] flags you can replace strings in job values. Currently the supported fields are:
  - Spark application name
  - Spark arguments
  - Spark configurations

  For a variable flag name=value any substring {{{name}}} in the value of the supported field gets replaced with value.
- A custom runtime Docker image can be specified for the job using the --runtime-image-resource-name flag, which has to refer to the name of a custom image resource that has already been created.

By default the command returns the job run ID as soon as the job has been submitted.

Optionally, you can use the --wait switch to wait until the job run ends and returns a non-zero exit code if the job run was not successful.

Running a Airflow job using the CLI

The following example demonstrates how to run a Cloudera Data Engineering (CDE) Airflow job using the command line interface (CLI).

Before you begin

Make sure that the job has been created and all necessary resources have been created and uploaded.

Note: Custom Docker container images is a *Technical Preview* feature. Contact your Cloudera account representative to enable access to this feature.

About this task

Using the cde job run requires more preparation on the target environment compared to the cde airflow submit command. Whereas cde airflow submit is a quick and efficient way of testing an Airflow job during development, cde job run is suited for production environments where a job is to be run multiple times, therefore removing resources and job definitions after every job run is neither necessary, nor viable.

Procedure

To run an Airflow job, run the following command:

```
cde job run --name <job name> [--config <key=value>]* [--wait]
```

Airflow configs provided at job run time will override the corresponding job configs.

By default the command returns the job run ID as soon as the job has been submitted.

Optionally, you can use the --wait switch to wait until the job run ends and returns a non-zero exit code if the job run was not successful.

Scheduling Spark jobs

Spark jobs can optionally be scheduled so that they are automatically run on an interval. Cloudera Data Engineering uses the Apache Airflow scheduler to create the schedule instances.

About this task



Note:

Airflow DAGs manage their own schedules, therefore Airflow job schedules cannot be set in this way, other than by using the operational commands pause, unpause, clear, mark-success.

Before you begin

Make sure that the Spark job has been created and all necessary resources have been created and uploaded.



Note: Custom Docker container images is a *Technical Preview* feature. Contact your Cloudera account representative to enable access to this feature.

Procedure

1. Define a running interval for your Spark job:

   The schedule interval is defined by a cron expression. Intervals can be regular, such as daily at 3 a.m., or irregular, such as hourly but only between 2 a.m. and 6 a.m. and only on weekdays. You can provide the cron expression directly or you can generate it using flags.

   Note: Scheduled job runs start at the end of the first full schedule interval after the start date, at the end of the scheduled period. For example, if you schedule a job with a daily interval with a start_date of 14:00, the first scheduled run is triggered at the end of the next day, after 23:59:59. However if the start_date is set to 00:00, it is triggered at the end of the same day, after 23:59:59.

   Available schedule interval flags are:

   --cron-expression

   A cron expression that is provided directly to the scheduler. For example, 0 */1 * * *

   --every-minutes

   Running frequency in minutes. Valid values are 0-59. Only a single value is allowed.

   --every-hours

   Running frequency in hours. Valid values are 0-23. Only a single value is allowed.

   --every-days

   Running frequency in days. Valid values are 1-31. Only a single value is allowed.

   --every-months

   Running frequency in months. Valid values are 1-12. Only a single value is allowed.

   --for-minutes-of-hour

   The minutes of the hour to run on. Valid values are 0-59. Single value, range (e.g.: 1-5), or list (e.g.: 5,10) are allowed.

   --for-hours-of-day

   The hours of the day to run on. Valid values are 0-23. Single value, range (e.g.: 1-5), or list (e.g.: 5,10) are allowed.

   --for-days-of-month

   The days of the month to run on. Valid values are 1-31. Single value, range (e.g.: 1-5), or list (e.g.: 5,10) are allowed.

   --for-months-of-year

   The months of the year to run on. Valid values are 1-12 and JAN-DEC. Single value, range (e.g.: 1-5), or list (e.g.: APR,SEP) are allowed.

   --for-days-of-week

   The days of the week to run on. Valid values are SUN-SAT and 0-6. Single value, range (e.g.: 1-5), or list (e.g. TUE,THU) are allowed.

   For example, to set the interval as hourly but only between 2 a.m. and 6 a.m. and only on weekdays, use the command:

   ```
   cde job create --name test_job --schedule-enabled=true --every-hours 1 --
   for-minutes-of-hour 0 --for-hours-of-day 2-6 --for-days-of-week MON-FRI --
   schedule-start 2021-03-09T00:00:00Z
   ```

   Or, equivalently, using a single cron expression:

   ```
   cde job create --name test_job --schedule-enabled=true --cron-expression
    '0 2-6/1 * * MON-FRI'  --schedule-start 2021-03-09T00:00:00Z
   ```

2. Define a time range for your Spark job:

   The schedule also defines the range of time that instances can be created for. The mandatory --schedule-start flag timestamp tells the scheduler the date and time from which the scheduling begins. The optional --schedule-end flag timestamp tells the scheduler the last date and time at which the schedule is active. If --schedule-end is not specified, the job runs at the scheduled interval until it is stopped manually.

   Note: Timestamps must be specified in ISO-8601 UTC format ('yyyy-MM-ddTHH:mm:ssZ'). UTC offsets are not supported.

   For example, to create a schedule that runs at midnight for each day of a single week, use the following command:

```
cde job create --name test_job --schedule-enabled=true --every-days 1 --
for-minutes-of-hour 0 --for-hours-of-day 0 --schedule-start 2021-03-09T0
0:00:00Z --schedule-end 2021-03-15T00:00:00Z
```

Enabling, disabling, and pausing scheduled jobs

Using the Cloudera Data Engineering (CDE) command line interface (CLI), you can enable, disable, or pause scheduled job runs.

Before you begin

Note:

Disabling the schedule removes all record of prior schedule instances.

Note:

Pausing and unpausing the schedule does not remove the record of prior schedule instances.

Procedure

- To enable or disable a job schedule, use the following command:

```
cde job (create | update) --name <job name> --schedule-enabled=(true | f
alse) ...
```

- To pause a job schedule upon schedule creation:

```
cde job (create | update) --name <job name> --schedule-enabled=true --sc
hedule-paused=true ...
```

- To pause an existing job schedule:

```
cde job schedule pause --name <job name>
```

- To unpause an existing job schedule:

```
cde job schedule unpause --name <job name>
```

Managing the status of scheduled job instances

Using the Cloudera Data Engineering (CDE) command line interface (CLI), you can clear the statuses of a range of scheduled instances or mark a scheduled job instance as successful.

Procedure

- To clear the status of a range of scheduled instances, run the following command:

```
cde job schedule clear [--schedule-start <start of clear period>] [--sch
edule-end <end of clear period>]
```

- To mark a single scheduled instance as successful, run the following command:

```
cde job schedule mark-success --execution-date <execution date of schedu
led instance>
```

  where <execution date of scheduled instance> is the timestamp that the instance was scheduled for, not when it actually ran.

CDE Spark job example

In this example there is a local Spark jar my-app-0.1.0.jar, and a local reference file my-ref.conf that the Spark job opens locally as part of its execution. The Spark job reads data from the location in the first argument and writes data to the location in the second argument. There is also a custom Spark configuration for tuning performance.

1. Make your job available for running in one of the following ways:

   You can submit the job directly to CDE and have it run the job once, using the spark submit command. In this case no permanent resources are created on CDE subsequently no cleanup is necessary after the job run. This is ideal when testing a job.

```
cde spark submit my-app-0.1.0.jar \
  --file my-ref.conf \
  --conf spark.sql.shuffle.partitions=1000
```

   If you plan to run the same job several times it is is a good idea to create and upload the resource and job and then run it on CDE using the job run command. This is the preferable method in production environments.

```
> cde resource create --name my-resource
> cde resource upload --name my-resource --local-path my-app-0.1.0.jar
 109.7MB/109.7MB 100% [===============================================] my-
app-0.1.0.jar
> cde resource upload --name my-resource --local-path my-ref.conf
   135.0b/135.0b 100% [===============================================] my-
ref.conf
> cde job create \
  --name my-job \
  --type spark \
  --mount-1-resource my-resource \
  --application-file my-app-0.1.0.jar \
  --conf spark.sql.shuffle.partitions=1000 \
> cde job run --name my-job
{
  "id": 1
}
> cde run describe --id 1 | jq -r '.status'
starting
...
> cde run describe --id 1 | jq -r '.status'
finished
```

2. Schedule your job:

   As the above created job stays in CDE permanently until you delete it, you can schedule it to run regularly at a predefined time. This example schedules your job to run daily at midnight, starting from January 1, 2021:

```
> cde job update \
  --name my-job \
  --schedule-enabled=true \
  --schedule-start 2021-01-01T00:00:00Z \
  --every-days 1 \
  --for-minutes-of-hour 0 \
  --for-hours-of-day 0
```

CDE CLI command reference

The Cloudera Data Engineering (CDE) command line syntax is shown below. You can view additional syntax help by adding --help after any command.

cde command

```
Usage:
  cde [command]

Available Commands:
  help        Help about any command
  job         Manage CDE jobs
  resource    Manage CDE resources
  run         Manage CDE runs
  spark       Spark commands

Flags:
      --auth-cache-file string     token file cache location (default "$USE
RCACHE/token-cache")
      --auth-no-cache              do not cache authentication tokens
      --auth-pass-file string      authentication password file location
  -h, --help                       help for cde
      --hide-progress-bars         hide progress bars for file uploads
      --insecure                   API does not require authentication
      --tls-ca-certs string        additional PEM-encoded CA certificates
      --tls-insecure               skip verification of API server TLS certi
ficate
      --user string                CDP user to authenticate as
      --vcluster-endpoint string   CDE virtual cluster endpoint
  -v, --verbose                    verbose logging
      --version                    version for cde
Use "cde [command] --help" for more information about a command.
```

cde job command

```
Usage:
  cde job [command]

Available Commands:
  create      Create a job
  delete      Delete a job
  describe    Describe a job
  import      Import a job
  list        List jobs
```

```
  run         Run a job
  schedule    Operate CDE job schedules
  update      Update a job
```

cde resource command

```
Usage:
  cde resource [command]
Available Commands:
  create      Create a resource
  delete      Delete a resource
  delete-file Delete a file from a resource
  describe    Describe resource
  download    Download a file from a resource
  list        List resources
  upload      Upload a file to resource
```

cde run command

```
Usage:
  cde run [command]

Available Commands:
  describe    Describe a run
  kill        Kill a run
  list        List runs
  logs        Retrieve logs for a run
  ui          Open a run in the default browser
```

cde spark command

```
Usage:
  cde spark [command]

Available Commands:
  submit      Run a jar/py file on CDE Spark
```

CDE CLI Spark flag reference

The Cloudera Data Engineering (CDE) command Spark flag reference is shown below.

```
--application-file: application main file
--class: application main class
--arg: Spark argument
--conf: Spark configuration (format key=value) (can be repeated)
--min-executors: minimum number of executors
--max-executors: maximum number of executors
--initial-executors: initial number of executors
--executor-cores: number of cores per executor
--executor-memory: memory per executor
--driver-memory: memory for driver
--driver-cores: number of driver cores
--spark-name: Spark application name
--file: additional file additional file (can be repeated) (will be merged w
ith --files, if provided)
--files: additional files (comma-separated list) (will be merged with all --
file)
```

```
--jar: additional jar (can be repeated) (will be merged with --jars, if prov
ided)
--jars: additional jars (comma-separated list) (will be merged with all --
jar)
--py-file: additional Python file (can be repeated) (will be merged with --
py-file, if provided)
--py-files: additional Python files (comma-separated list) (will be merged
 with all --py-file)
--packages: additional dependencies as comma-separated list of Maven coordi
nates
--repositories: additional repositories/resolvers for retrieving the --pac
kages dependencies
--python-env-resource-name: Python environment resource name
--python-version: Python version ("python3" or "python2")
--log-level: log level for Spark containers (TRACE, DEBUG, INFO, WARN, ERR
OR, FATAL, OFF)
--enable-analysis: enables Spark analysis (see 'Analysis' UI tab for a job r
un)
```

CDE CLI Airflow flag reference

The Cloudera Data Engineering (CDE) command Airflow flag reference is shown below.

```
--dag-file: DAG filename
--config: DAG configuration (can be repeated). Use in DAG using templates.
For example,
for --config hello=world, use in DAG as {{ dag_run.conf['hello'] }} to be
replaced with world.
```

CDE CLI list command syntax reference

You can include flags with the Cloudera Data Engineering (CDE) command line interface (CLI) list command calls to filter the result set.

cde [credential|job|resource|run|...] list [--filter [fieldname[operator]argument]] [--filter [fieldname[operator]argument]] ...

A list command call can include multiple filter flags, where all filters must match for the entry to be returned. You have to enclose filters in quotes.

fieldname

is selected from the top-level fields of the returned entries. Filtering of fields nested within other fields is supported using MySQL 8 JSON path expressions.

operator

is one of: eq, noteq, lte, lt, gte, gt, in, notin, like, rlike. The in and notin operators work on an argument of comma-separated values. The like operator matches using SQL LIKE syntax, e.g. %test%. The rlike operator matches using the SQL REGEXP regular expression syntax.

argument

is the value, list, or expression to match with the operator. If the argument contains commas the filter has to be enclosed in a second set of quotes, for example: '"id[in]12,14,16"'.



Note:

Timestamps must be formatted as MySQL date time literals.

For example:

```
cde run list --filter 'spark.spec.file[rlike]jar'
```