Cloudera Data Engineering 1.5.4

Accessing the Cloudera Data Engineering service using the CLI

Date published: 2020-07-30
Date modified: 2024-05-30

CLOUDERA

https://docs.cloudera.com/

Legal Notice

Contents

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.

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.

session

A 'session' is an interactive short-lived development environment for running Spark commands to help you iterate upon and build your Spark workloads.

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/mou nt). 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 24.
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 24.

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 . While creating a job if you want to use the [--data-connector] flag, you must obtain the name of the data connector from the CDE UI by navigating to Administration > click Service Details icon of the CDE Service > Data Connectors tab.

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] [--data-connector
    name] --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.

   Using the [--data-connector] flag, you can specify the name of the data connector. Currently, only the Ozone type data connector is supported and it must be created before 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]
```

You can use [--job-name <job name>] to specify the same CDE job name for consecutive cde spark submit commands. 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-json <json-string>]* [--job-n
ame <job 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.
- GPU Acceleration (Technical Preview): Using [--enable-gpu-acceleration] you can accelerate your Spark jobs using GPUs. You can use [--executor-node-selector "nvidia.com/gpu=A100"] and [--executor-node-toleration "nvidia.com/gpu=true"] options to configure selectors and tolerations if you want to run the job on specific GPU nodes. When this job is run, this particular job will request GPU resources.

  Warning: You must ensure this virtual cluster has been configured with GPU resource quota. Otherwise, the jobs will be in the Pending state as no GPU resource can be allocated to the pod.

  ```
  cde job run --name example-pi \
  --enable-gpu-acceleration \
  --executor-node-selector "nvidia.com/gpu=A100" \
  --executor-node-toleration "nvidia.com/gpu=true"
  ```

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>
```

or

```
cde job schedule pause-all
```

- 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.

Managing Sessions in Cloudera Data Engineering using the CLI

A Cloudera Data Engineering (CDE) Session is an interactive short-lived development environment for running Spark commands to help you iterate upon and build your Spark workloads.

Creating a Session using the CDE CLI

The cde session create command allows you to create a new Session.

Procedure

Run the following command in the CDE CLI:

```
cde session create --name <session-name> --type <pyspark/spark-scala>
```

- You can enable the GPU acceleration using the[--enable-gpu-acceleration] flag during the spark session creation.

  ```
  cde session create --name test-session --type spark-scala \
  --enable-gpu-acceleration
  ```

  - To accelerate session queries on specific hardware, you can use [--executor-node-selector "nvidia.com/gpu=A100"] and [--executor-node-toleration "nvidia.com/gpu=true"] options to configure selectors and tolerations if you want to run the job on specific GPU nodes. You can only provide executor node selectors and tolerations, as GPUs are used by executors only. The selector and tolerations CLI options are optional. You can use either one or both in congestion with --enable-gpu-acceleration flag.

    For example:

    ```
    cde session create --name test-session --type spark-scala \
                        --enable-gpu-acceleration \
                        --executor-node-selector "nvidia.com/gpu=A100" \
                        --executor-node-toleration "nvidia.com/gpu=true"
    ```

Interacting with a Session using the CDE CLI

Once your Session has been created, you can interact with it using the cde sessions interact command.

About this task

Below is an example that demonstrates how to interact with a PySpark or Scala Session in CDE using the CLI.

Procedure

Run the following command in the CDE CLI:

```
cde session interact --name <session-name>
```

Sessions example for the CDE CLI

In this example, a Session is created using the Cloudera Data Engineering (CDE) CLI with resources specified during creation. In this example, python environment, files, Git repository, and workload credentials resources are used.

Note: Access files from the /app/mount after they are mounted. Access Secrets from /etc/dex/secrets/<secret -name>.

```
> cde session create --name resources --type pyspark --python-env-resource-n
ame example-virtual-env --runtime-image-resource-name docker-image --mount-1
-resource octocat --mount-2-resource example-files --mount-3-resource exampl
e-data --workload-credential workload-cred --workload-credential workload-cr
ed-2
{
  "name": "resources",
  "type": "pyspark",
  "creator": "csso_surya.balakrishnan",
  "created": "2023-10-06T03:13:03Z",
  "mounts": [
    {
      "dirPrefix": "/",
      "resourceName": "octocat"
    },
    {
      "dirPrefix": "/",
      "resourceName": "example-files"
    },
    {
      "dirPrefix": "/",
      "resourceName": "example-data"
    }
  ],
  "lastStateUpdated": "2023-10-06T03:13:03Z",
  "state": "starting",
  "interactiveSpark": {
    "id": 1,
    "driverCores": 1,
    "executorCores": 1,
    "driverMemory": "1g",
    "executorMemory": "1g",
    "numExecutors": 1,
    "pythonEnvResourceName": "example-virtual-env"
  },
  "workloadCredentials": [
    "workload-cred",
    "workload-cred-2"
  ],
  "runtimeImageResourceName": "docker-image"
}

> ./cde session interact --name resources
```

```
Starting REPL...
Waiting for the session to go into an available state...
Connected to Cloudera Data Engineering...
Press Ctrl+D (i.e. EOF) to exit
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\
      /_/
Type in expressions to have them evaluated.

>>> os.listdir("/app/mount")
['.git', 'README', 'access-logs-ETL-iceberg.py', 'access-logs-ETL.py', 'a
ccess-logs.txt', 'cdeoperator.py', 'pyspark-batch-job.py', 'pyspark_wordcoun
t.py', 'spark-load-data.py', 'word_count_templates.txt', 'wordcount_input_1.
txt', 'dex-spark-driver-template-txckdpxp.yaml', 'dex-spark-executor-templat
e-txckdpxp.yaml']

>>> sec_path = "/etc/dex/secrets/workload-cred/key1"
>>> with open(sec_path) as f:
...     for line in f:
...         print(line)
value1
>>> sec_path = "/etc/dex/secrets/workload-cred-2/key2"
>>> with open(sec_path) as f:
...     for line in f:
...         print(line)
value2
>>> import pandas

>>> dates = pandas.date_range("20130101", periods=6)

>>> dates
DatetimeIndex(['2013-01-01', '2013-01-02', '2013-01-03', '2013-01-04',
               '2013-01-05', '2013-01-06'],
              dtype='datetime64[ns]', freq='D')
```

Sessions command descriptions

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

Command	Definition
cde session create	Creates a CDE session. Sessions are identified by a user-specified name. Sessions have a type that defines the engine that the Session will run on. The 'spark-scala'[Scala REPL] and 'pyspark'[Python REPL] types are currently supported.
cde session interact	Connects to a running session in a Spark shell similar to the interface and submit statements.
cde session kill	Ends a session. The Spark driver and executor processes are stopped.
cde session delete	Deletes a session and removes all references to the session. Logs will no longer be accessible.
cde session list	Lists all sessions. The --output flag can be used to control the output format. You can use the `--output string output format ("table" or "json") (default "table")` flag to specify whether the Session's output must be in a table or JSON format
cde session statements	Lists session statements. The --output flag can be used to control the output format.

Command	Definition
cde sessions describe –name <session_name>	Describes the session. The command is used as an input to name the session.

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.

```
cde airflow submit --help
Usage:
  cde airflow submit [flags]

Examples:
For a local DAG file 'my-airflow-job.py':
> cde airflow submit my-airflow-job.py

Flags:
      --airflow-file-mount-N-prefix string     mount directory prefix for
airflow file mount N (defaults to "/airflow-file-mount-N-resource-name")
      --airflow-file-mount-N-resource string   resource name for airflow fi
le mount N
      --config stringArray                     DEPRECATED - DAG configura
tion (format key=value) (can be repeated). Use --config-json option instead.
      --config-json string                     DAG configuration in JSON st
ring format
      --config-json-file string                DAG configuration file locati
on in JSON format
      --dag-file string                        DAG filename, path to the DAG
 within the resource
  -h, --help                                   help for submit
      --job-name string                        name of the generated job
```

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]argume
nt]] ...

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'
```