Machine Learning

Command Line Reference

Date published: 2020-07-16
Date modified: 2022-04-11

https://docs.cloudera.com/

Legal Notice

Contents

Command Line Tools in CML

Cloudera Machine Learning ships with the following command line tools. The purpose of each tool differs.

- CDP CLI for Cloudera Machine Learning - If you prefer to work in a terminal window, you can download and configure the CDP client that gives you access to the CDP CLI tool. The CDP CLI allows you to perform the same actions as can be performed from the management console. Use this CLI to create, delete, upgrade, and manage ML workspaces on CDP.

  To view all the available commands, run:

  ```
  cdp ml help
  ```

  To view help for a specific command, run:

  ```
  cdp ml <operation> help
  ```

  If you don't already have the CDP CLI set up, see *Installing the CDP CLI Client*.
- cdswctl - Cloudera Machine Learning also ships with a CLI client that you can download from the Cloudera Machine Learning web UI. This is also referred to as the Model CLI client. The cdswctl client allows you to log in, create an SSH endpoint, launch new sessions, automate model deployment, model updates, and so on.

cdswctl Command Line Interface Client

Cloudera Machine Learning ships with a CLI client that you can download from the Cloudera Machine Learning web UI. The cdswctl client allows you to perform the following tasks:

- Logging in
- Creating an SSH endpoint
- Listing sessions that are starting or running
- Starting or stopping a session
- Creating a model
- Building and deploying models
- Listing model builds and model deployments
- Checking the status of a deployment
- Redeploying a model with updated resources
- Viewing the replica logs for a model

Other actions, such as creating a project, require you to use the Cloudera Machine Learning web UI. For information about the available commands, run the following command:

```
cdswctl --help
```

Download and Configure cdswctl

This topic describes how to download the cdswctl CLI client and configure your SSH public key to authenticate CLI access to sessions.

About this task

Before you begin, ensure that the following prerequisites are met:

- You have an SSH public/private key pair for your local machine.
- You have Contributor permissions for an existing project. Alternatively, create a new project you have access to.
- If you want to configure a third-party editor, make sure the Site Administrator has not disabled remote editing for Cloudera Machine Learning.

(Optional) Generate an SSH Public/Private Key

About this task

This task is optional. If you already have an SSH public/private key pair, skip this task. The steps to create an SSH public/private key pair differ based on your operating system. The following instructions are meant to be an example and are written for macOS using ssh-keygen.

Procedure

1. Open Terminal.
2. Run the following command and complete the fields:

   ```
   ssh-keygen -t rsa -f ~/.ssh/id_rsa
   ```

   Keep the following guidelines in mind:

   - Make sure that the SSH key you generate meets the requirements for the local IDE you want to use. For example, PyCharm requires the -m PEM option because PyCharm does not support modern (RFC 4716) OpenSSH keys.
   - Provide a passphrase when you generate the key pair. Use this passphrase when prompted for the SSH key passphrase.
   - Save the SSH key to the default ~/.ssh location.

Download cdswctl and Add an SSH Key

Procedure

1. Open the Cloudera Machine Learning web UI and go to SettingsRemote Editing for your user account.
2. Download cdswctl client for your operating system.

   Unpack it, and optionally, you can add it to the *PATH* environment variable on your system.
3. Add your SSH public key to SSH public keys for session access.

   Cloudera Machine Learning uses the SSH public key to authenticate your CLI client session, including the SSH endpoint connection to the Cloudera Machine Learning deployment.

   Any SSH endpoints that are running when you add an SSH public key must also be restarted.

Initialize an SSH Endpoint

This topic describes how to establish an SSH endpoint for Cloudera Machine Learning.

About this task

Creating an SSH endpoint is also the first step to configuring a remote editor for an ML workspace.

Procedure

1. Create a configuration file at: $HOME/.cdsw/config.yaml. The contents of config.yaml should be:

   ```
   username: <username>
   url: <ML_workspace_url>
   auth:
   ```

```
  authtype: 1
  basic: null
  apikey: <your_api_key>
```

To collect the values for these fields, first log in to your CML workspace using SSO:

- username: The username with which you are logged into the CML workspace. Found in the top right corner of your ML workspace.
- url: The complete URL used to access the CML workspace. For example: https://ml-<randomly-generated-c luster-name>
- apikey: Go to User Settings API Keys . Copy the value of the Legacy API Key to this field.

2. Create a local SSH endpoint to Cloudera Machine Learning. Run the following command:

```
cdswctl ssh-endpoint -p <username>/<project_name> [-c <CPU_cores>] [-m <
memory_in_GB>] [-g <number_of_GPUs>] [-r <runtime ID> ]
```

If the project is configured to use ML runtimes, the -r parameter must be specified, otherwise it must be omitted. See *Using ML runtimes with cdswctl* documentation page for more information.

The command uses the following defaults for optional parameters:

- CPU cores: 1
- Memory: 1 GB
- GPUs: 0

For example, the following command starts a session for the user milton under the customerchurn project with .5 cores, .75 GB of memory, 0 GPUs, and the Python3 kernel:

```
cdswctl ssh-endpoint -p customerchurn -c 0.5 -m 0.75
```

To create an SSH endpoint in a project owned by another user or a team, for example finance, prepend the username to the project and separate them with a forward slash:

```
cdswctl ssh-endpoint -p finance/customerchurn -c 0.5 -m 0.75
```

This command creates session in the project customerchurn that belongs to the team finance.

Information for the SSH endpoint appears in the output:

```
...
You can SSH to it using
    ssh -p <some_port> cdsw@localhost
...
```

3. Open a new command prompt and run the outputted command from the previous step:

```
ssh -p <some_port> cdsw@localhost
```

For example:

```
ssh -p 9750 cdsw@localhost
```

You will be prompted for the passphrase for the SSH key you entered in the ML workspace web UI.

Once you are connected to the endpoint, you are logged in as the cdsw user and can perform actions as though you are accessing the terminal through the web UI.

4. Test the connection.

If you run ls, the project files associated with the session you created are shown. If you run whoami, the command returns the cdsw user.

5. Leave the SSH endpoint running as long as you want to use a local IDE.

Related Information

Using ML Runtimes with cdswctl

ML Runtimes versus Legacy Engines

Log into cdswctl

This topic describes how to log into cdswctl.

Procedure

1. Open the Model CLI client.
2. Run the following command while specifying the actual values for the variables:

```
cdswctl login -u <workspace_url> -n <username> -y <legacy_api_key>
```

   where

   - *workspace_url* is the workspace URL including the protocol (http(s)://domain.com)
   - *username* is your user name on the workspace
   - *legacy_api_key* is the API key that you can obtain from the Cloudera Machine Learning UI. Go to Settings API Keys and copy the Legacy API Key (and not the API Key).

   To see more information about the login command parameters, run

```
cdswctl login --help
```

   If all goes well, then you'll see "Login succeeded".

Prepare to manage models using the model CLI

Before you can start using the model CLI to automate model deployment or to perform any other tasks, you must install the scikit-learn machine learning library for Python through the Cloudera Machine Learning web UI.

About this task

You must perform this task through the Cloudera Machine Learning web UI.

Procedure

1. Create a new project with Python through the web UI.
   Python provides sample files that you can use to create models using CLI.
2. To start a new session, go to the **Sessions** page from the left navigation panel and click new session.
   The **Start the new session** page is displayed.
3. On **Start the new session** page, select Python 3 from the Engine Kernel drop-down menu, and click Launch Session.
   A new “Untitled Session” is created.
4. From the input prompt, install the scikit-learn machine learning library for Python by running the following command:

```
!pip3 install sklearn
```

5. Open the fit.py file available within your project from the left navigation panel.
   You can use the fit.py file to create a fitted model which creates a model.pkl file that you can use to deploy the actual model.

6. Run the fit.py file by clicking Run Run all .

   The model.pkl directory is created that you can see within your project on the left navigation pane.
7. Close the session by clicking Stop.

Create a model using the CLI

This topic describes how to create models using the model CLI.

Procedure

1. Open a terminal window and log into cdswctl.
2. Obtain the project ID as described in the following steps:
   a) Run the following command:

      ```
      cdswctl projects list
      ```

      The project ID, your username, and the project name are displayed. For example:

      1: john-smith/petal-length-predictor
   b) Note the project ID, which is a number in front of your project name.

      In this case, it is "1".
3. Run the following command while specifying the project name and note the engine image ID:

   Note: The following examples are specific to projects configured to use legacy engines and projects configured to use runtimes. Be sure to use the commands appropriate to your project configuration.

   For projects configured to use legacy engines:

   ```
   cdswctl engine-images list -p <project-name>
   ```

   For example,

   ```
   cdswctl engine-images list -p john-smith/petal-length-predictor
   ```

   For projects configured to use runtimes:

   ```
   cdswctl runtimes list
   ```

   Depending on your local setup, you may get a more readable output by post-processing the result with the following command:

   ```
   cdswctl runtimes list | python3 -m json.tool
   ```

   For this example you should pick a runtime with a Python kernel and Workbench editor. Depending on your local setup, you may filter the results using the following command:

   ```
   cdswctl runtimes list | jq  '.runtimes[] | select((.editor == "Workbench")
    and (.kernel | contains("Python")))'
   ```
4. Create a model by using the following command:

   Note: The following examples are specific to projects configured to use legacy engines and projects configured to use runtimes. Be sure to use the commands appropriate to your project configuration.

   For projects configured to use legacy engines:

   ```
   cdswctl models create
   ```

```
--kernel="python3"
--targetFilePath="predict.py"
--targetFunctionName="predict"
--name="Petal Length Predictor"
--cpuMillicores=1000
--memoryMb=2000
--description="Model of the Iris dataset"
--replicationType=fixed
--numReplicas=1
--visibility="private"
--autoBuildModel
--autoDeployModel
--projectId=<project ID>
--examples='{"request":{"petal_length":1}}'
--engineImageId=<engine image ID from before>
```

For projects configured to use runtimes:

```
cdswctl models create
--targetFilePath="predict.py"
--targetFunctionName="predict"
--name="Petal Length Predictor"
--cpuMillicores=1000
--memoryMb=2000
--description="Model of the Iris dataset"
--replicationType=fixed
--numReplicas=1
--visibility="private"
--autoBuildModel
--autoDeployModel
--projectId=<project ID>
--examples='{"request":{"petal_length":1}}'
--runtimeId=<runtime ID obtained above>
```

If the command runs successfully, the system displays the model details in a JSON format.

5. For more information about the models create command parameters, run the following command:

```
cdswctl models create --help
```

Build and deployment commands for models

Models have separate parameters for builds and deployments. When a model is built, an image is created. Whereas, the deployment is the actual instance of the model. You can list model builds and deployment, and monitor their state using from model CLI client (cdswctl).

Listing a model

To list the models, run the following command:

```
cdswctl models list
```

Monitoring the status of the model

To monitor the status of the build for a particular model, use the following command:

```
cdswctl models listBuild --modelId <model_ID> --projectId <project_ID>
```

You can use the --latestModelDeployment flag to get the build for the latest deployment.

Listing a deployment

To list the deployment for a particular model, run the following command:

```
cdswctl models listDeployments --modelId <model_ID>
```

Checking the status of a deployment

To check the status of your deployment, run the following command:

```
cdswctl models listDeployments --statusSet=deployed
```

Following is a list of arguments for the statusSet parameter:

- deployed
- deploying
- failed
- pending
- stopping
- stopped

Note: You can use the parameter more than once in a command to check multiple statuses of your deployed models. For example,

```
cdswctl models listDeployments --statusSet=deployed --statusSet=stop
ped --statusSet=failed
```

Deploy a new model with updated resources

You can republish a previously-deployed model in a new serving environment with an updated number of replicas or memory/CPU/GPU allocation by providing the model build ID of the model you want to rebuild.

To deploy a new model, use the following command:

```
cdswctl models deploy --modelBuildId=<build_ID> --cpuMilli
cores=<num_of_cpu_cores> --memoryMb=<memory_in_mb> --numRepli
cas=<num_of_replicas> --replicationType=<replication_type>
```

For example:

```
cdswctl models deploy --modelBuildId=<build_ID> --cpuMillicores=1200 --mem
oryMb=2200 --numReplicas=2 --replicationType=fixed
```

Note: You must specify values for all the non-zero resources, even if you do not wish to update their values. For example, in your existing deployment, if you set the cpuMillicores capacity to 1200 and you do not wish to increase or decrease it, you must still specify cpuMillicores=1200 in the command.

View replica logs for a model

When a model is deployed, Cloudera Machine Learning enables you to specify the number of replicas that must be deployed to serve requests. If a replica crashes or fails to come up, you can diagnose it by viewing the logs for every replica using the model CLI.

Procedure

1. Obtain the modelReplicaId by using the following command:

   ```
   cdswctl models listReplicas --modelDeploymentId=<model_deployment_ID>
   ```

   where the *model_deployment_ID* is the ID of a successfully deployed model.
2. To view the replica logs, run the following command:

   ```
   cdswctl models getReplicaLogs --modelDeploymentId=<model_deployment_ID> --
   modelReplicaId="<replica_ID>" --streams=stdout
   ```

   For example:

   ```
   cdswctl models getReplicaLogs --modelDeploymentId=2 --modelReplicaId="pe
   tal-length-predictor-1-2-6d6496b467-hp6tz" --streams=stdout
   ```

   The valid values for the streams parameter are stdout and stderr.

Using ML Runtimes with cdswctl

If a project is configured to use Runtimes, cdswctl workflows for starting sessions or models are slightly different.

Querying the engine type

You can query whether a project is configured using ML Runtimes or Legacy Engine.

Procedure

To determine if a project is configured to use either ML Runtimes or Legacy Engines, use the cdswctl projects get EngineType command and specify the project with the -p parameter.

For example, to determine if configured to use ML Runtimes:

```
cdswctl projects getEngineType -p demouser/runtimeproject
ml_runtime
```

```
cdswctl projects getEngineType -p demouser/legacyproject
legacy_engine
```

Listing runtimes

The first step to working with projects using runtimes is to query the available runtimes using the cdswctl runtimes list command.

About this task

The cdswctl runtimes list command returns all runtimes in a large JSON result. For easier consumption, you can post-process this result with some 3rd-party tool, such as jq or Python's json.tool.

Procedure

To query the available runtimes, use the cdswctl runtimes list command.

Note: The following examples are for presentation purposes only. Neither Python's json.tool nor jq are supported directly by Cloudera.

The following example pipes the cdswctl runtimes list result through Python's json.tool to produce a more readable output:

```
user@host:~ $ cdswctl runtimes list | python3 -m json.tool
{
    "runtimes": [
        {
            "id": 1,
            "imageIdentifier": "docker.repository.cloudera.com/cdsw/ml-run
time-workbench-python3.6-standard:2020.11.1-b6",
            "editor": "Workbench",
            "kernel": "Python 3.6",
            "edition": "Standard",
            "shortVersion": "2020.11",
            "fullVersion": "2020.11.1-b6",
            "maintenanceVersion": 1,
            "description": "Standard edition Python runtime provided by Cl
oudera"
        },
        {
            "id": 2,
            "imageIdentifier": "docker.repository.cloudera.com/cdsw/ml-run
time-jupyterlab-python3.7-standard:2020.11.1-b6",
            "editor": "JupyterLab",
            "kernel": "Python 3.7",
            "edition": "Technical Preview",
            "shortVersion": "2020.11",
            "fullVersion": "2020.11.1-b6",
            "maintenanceVersion": 1,
            "description": "Technical Preview JupyterLab Python runtime pro
vided by Cloudera"
        }
    ]
}
```

The following example pipes the cdswctl runtimes list result through jq to transform the JSON output into arbitrary formats:

```
user@host:~ $ cdswctl runtimes list | jq -r '.runtimes[] | "\(.id) \(.imageI
dentifier)"'
1
docker.repository.cloudera.com/cdsw/ml-runtime-workbench-python3.6-standa
rd:2020.11.1-b6
2
docker.repository.cloudera.com/cdsw/ml-runtime-jupyterlab-python3.7-standa
rd:2020.11.1-b6
```

The following example filters the cdswctl runtimes list result using jq to only show runtimes with specific editors and kernels:

```
user@host:~ $ cdswctl runtimes list | jq  '.runtimes[] | select((.editor ==
"Workbench") and (.kernel | contains("Python")))'

{
  "id": 1,
  "imageIdentifier": "docker.repository.cloudera.com/cdsw/ml-runtime-workben
ch-python3.6-standard:2020.11.1-b6",
  "editor": "Workbench",
  "kernel": "Python 3.6",
  "edition": "Standard",
  "shortVersion": "2020.11",
  "fullVersion": "2020.11.1-b6",
  "maintenanceVersion": 1,
  "description": "Standard edition Python runtime provided by Cloudera"
}
```

Starting sessions and creating SSH endpoints

Once you choose a runtime, you can start a session using the cdswctl sessions start command and create SSH endpoints using the cdswctl   ssh-endpoint command.

About this task

The runtime ID used in the following steps is obtained using the steps outlined in *Listing runtimes*.

Procedure

1. To start a session with a runtime, use the cdswctl sessions start command, specifying the runtime ID with the -r parameter and the project with the -p parameter.

   For example:

   ```
   cdswctl sessions start -r 2 -p demouser/runtimeproject
   ```

2. To specify SSH endpoints for the runtimes sessions, use the cdswctl   ssh-endpoint command and specify the runtime ID using the -r parameter. and the project with the -p parameter.

   For example:

   ```
   cdswctl ssh-endpoint -r 1 -p demouser/runtimeproject
   ```

Creating a model

Creating a model in a project that uses runtimes is similar to model creation with an legacy engine, but you must use a different parameter to specify the runtime ID.

About this task

To create a model in a project that uses runtimes you must use the --runtimeId= parameter to specify a runtime ID (instead of using the --engineImageId= and --kernel= parameters used for a legacy engine).

Procedure

To create a model in a project that uses runtimes use the --runtimeId= parameter to specify a runtime ID.

For example:

```
cdswctl models create --targetFilePath=predict.py --targetFunctionName=predi
ct
  --projectId=4 --name=created-using-cdswctl --description=created-using-cds
wctl
  --memoryMb=1024 --authEnabled --cpuMillicores=250 --autoBuildModel --aut
oDeployModel
  --examples='{"request":{"petal_length":1}}' --runtimeId=1
```

cdswctl command reference

You can manage your Cloudera Machine Learning Workbench cluster with the CLI client (cdswctl) that exists within the Cloudera Machine Learning Workbench. Running `cdswctl` without any arguments prints a brief description of each command.

Table 1: Model CLI Command Reference

Command	Description and usage
cdswctl login	Enables you to log into the model CLI client
cdswctl projects list	Lists the projects
cdswctl models create	Creates a model with the specified parameters
cdswctl models list	Lists all models You can refine the search by specifying the modelId
cdswctl models listBuild	Llists the builds for a model You can monitor the status of the build by specifying the modelId and the projectId
cdswctl models listDeployments	List the deployments for a model You can refine the search by specifying the modelId Use the statusSet parameter to check the status of the model being deployed
cdswctl models deploy	Deploys a model with the specified parameters
cdswctl models listReplicas	Enables you to view the list of model replicas You also need this information to obtain replica logs
cdswctl models getReplicaLogs	Enables you to view the logs for a model replica
cdswctl models restart	Restarts a model Usage: cdswctl models restart --modelDeploymentId=<*deployment_ID*> Note: Running this command does not change the resources if you previously ran the cdswctl models update command
`cdswctl models update`	Changes the name, description, or visibility of the model To change a model’s resources, use the `cdswctl models deploy` command
`cdswctl models delete`	Deletes a model Usage: cdswctl models delete --id=<*model_ID*>

Azure NetApp Files Management with the CLI

You can manage the NetApp Files setup using the CLI. This can be helpful for automating setup and teardown of workspaces as ML project needs change.

Create an Azure NetApp Files account

The following code sample creates an Azure NetApp Files account.

```
 az netappfiles account create \ --account-name my-anf-account \ --resource-
group
    my-cdp-resource-group \ --location westus2
```

Create a capacity pool

A capacity pool is a storage container for volumes, which are accessed directly by CML. The minimum size for an Azure NetApp Files capacity pool is 4 TiB

```
 MINIMUM_POOL_SIZE=4 # 4 TiB is the minimum az netappfiles pool create \
    --account-name my-anf-account \ --pool-name my-anf-pool \ --resource-gr
oup my-cdp-resource-group
    \ --service-level Standard \ --location westus2 \ --size ${MINIMUM_POO
L_SIZE}
```

Create a volume

Create one or more volumes in the capacity pool. The "Usage threshold" is referred to as the "quota" in the Azure web portal. It is measured in GiB. The volume must support the NFSv3 protocol (which is the default).

```
 az netappfiles volume create \ --account-name my-anf-account \ --pool-name
    my-anf-account \ --volume-name my-anf-volume \ --resource-group my-cdp-r
esource-group \
    --location westus2 \ --file-path my-anf-volume \ --usage-threshold 1000
\ --vnet my-cdp-vnet \
    --subnet my-anf-subnet \ --service-level Standard
```

The mount path for this volume, or a dedicated, empty subdirectory inside that volume, must be provided for the "Existing NFS" field when provisioning CML workspaces. It can be found in the "Mount Instructions" blade of the volume in the Azure portal.

Since each capacity pool has a large minimum, and each volume requires a dedicated subnet, users may wish to have a single volume that is shared between workspaces. This can be managed by having a VM that has the Azure volume mounted (instructions for doing this are also in the "Mount Instructions" blade of the volume in the Azure portal). This VM can then be used to quickly manage directories for individual workspaces on a single, shared volume. For instance:

```

    USER=      # username for accessing management VM
    VM=        # IP address or hostname for accessing management VM
    VOLUME=    # NFS volume name
    WORKSPACE= # CML workspace name (or other unique directory name)
    ssh ${USER}@${VM} "sudo mkdir ${VOLUME}/${WORKSPACE}; sudo chown 8536:8
536 ${VOLUME}/${WORKSPACE}"
    # ...
    ssh ${USER}@${VM} "sudo rm -r ${VOLUME}/${WORKSPACE}"

```