

CML Top Tasks (Private Cloud) Map

Date published: 2023-05-24

Date modified: 2023-06-09

CLOUDERA

Legal Notice

© Cloudera Inc. 2024. 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

Creating a Project with Legacy Engine Variants..... 4

Analytical Applications..... 4

Creating and deploying a Model..... 6

Creating a Project with Legacy Engine Variants

Projects create an independent working environment to hold your code, configuration, and libraries for your analysis. This topic describes how to create a project with Legacy Engine variants in Cloudera Machine Learning.

Procedure

1. Go to Cloudera Machine Learning and on the left sidebar, click Projects.
2. Click New Project.
3. If you are a member of a team, from the drop-down menu, select the **Account** under which you want to create this project. If there is only one account on the deployment, you will not see this option.
4. Enter a Project Name.
5. Select **Project Visibility** from one of the following options.
 - **Private** - Only project collaborators can view or edit the project.
 - **Team** - If the project is created under a team account, all members of the team can view the project. Only explicitly-added collaborators can edit the project.
 - **Public** - All authenticated users of Cloudera Machine Learning will be able to view the project. Collaborators will be able to edit the project.
6. Under **Initial Setup**, you can either create a blank project, or select one of the following sources for your project files.
 - **Built-in Templates** - Template projects contain example code that can help you get started with Cloudera Machine Learning. They are available in R, Python, PySpark, and Scala. Using a template project is not required, but it helps you start using Cloudera Machine Learning right away.
 - **Custom Templates** - Site administrators can add template projects that are customized for their organization's use-cases. For details, see *Custom Template Projects*.
 - **Local** - If you have an existing project on your local disk, use this option to upload compressed files or folders to Cloudera Machine Learning.
 - **Git** - If you already use Git for version control and collaboration, you can continue to do so with Cloudera Machine Learning. Specifying a Git URL will clone the project into Cloudera Machine Learning. To use a password-protected Git repository, see *Creating a project from a password-protected Git repo*.
7. Click Create Project. After the project is created, you can see your project files and the list of jobs defined in your project.

Note that as part of the project filesystem, Cloudera Machine Learning also creates the following .gitignore file.

```
R
node_modules
*.pyc
.*
!.gitignore
```

8. **(Optional)** To work with team members on a project, add them as collaborators to the project.

Analytical Applications

This topic describes how to use a Cloudera Machine Learning Workspace to create long-running web applications.

About this task:

This feature gives data scientists a way to create Cloudera Machine Learning web applications/dashboards and easily share them with other business stakeholders. Applications can range from single visualizations embedded in reports, to rich dashboard solutions such as Tableau. They can be interactive or non-interactive.

Applications stand alongside other existing forms of workloads in Cloudera Machine Learning Workspace (sessions, jobs, experiments, models). Like all other workloads, applications must be created within the scope of a project. Each application is launched within its own isolated engine. Additionally, like models, engines launched for applications do not time out automatically. They will run as long as the web application needs to be accessible by any users and must be stopped manually when needed.

Before you begin:

Testing applications before you deploy

Before you deploy an application using the steps described here, make sure your application has been thoroughly tested. You can use sessions to develop, test, and debug your applications. You can test web apps by embedding them in sessions as described here: [Web Applications Embedded in Sessions](#).

For CML UI

1. Go to a project's **Overview** page.
2. Click Applications.
3. Click New Application.
4. Fill out the following fields.

- **Name:** Enter a name for the application.
- **Run Application as:** If the application is to run in a service account, select Service Account and choose the account from the dropdown menu.
- **Subdomain:** Enter a subdomain that will be used to construct the URL for the web application. For example, if you use test-app as the subdomain, the application will be accessible at test-app.<ml-workspace-domain-name>.

Subdomains should be valid DNS hostname characters: letters from a to z, digits from 0 to 9, and the hyphen.

- **Description:** Enter a description for the application.
- **Script:** Select a script that hosts a web application on either CDSW_READONLY_PORT or CDSW_APP_PORT. Applications running on either of these ports are available to any users with at least read access to the project. The Python template project includes an entry.py script that you can use to test this out.



Note: Cloudera Machine Learning does not prevent you from running an application that allows a read-only user (i.e. Viewers) to modify files belonging to the project. It is up to you to make the application truly read-only in terms of files, models, and other resources belonging to the project.

- **Engine Kernel and Resource Profile:** Select the kernel and computing resources needed for this application.
- **Set Environment Variables:** Click Set Environment Variables, enter the name and value for the new application variable, and click Add.

If there is a conflict between the project-level and application-level environment variables, the application-level environment variables override the project-level environment variables.

5. Click Create Application.

For CML APIv2

To create an application using the API, refer to this example:

Here is an example of using the Application API.

```
application_request = cmlapi.CreateApplicationRequest(
    name = "application_name",
    description = "application_description",
```

```

    project_id = project_id,
    subdomain = "application-subdomain",
    kernel = "python3",
    script = "entry.py",
    environment = {"KEY": "VAL"}
)
app = client.create_application(
    project_id = project_id,
    body = application_request
)

```

Results:

In a few minutes, you should see the application status change to **Running** on the **Applications** page. Click on the name of the application to access the web application interface.

What to do next:

You can Stop, Restart, or Delete an application from the **Applications** page.

If you want to make changes to an existing application, click Overview under the application name. Then go to the Settings tab to make any changes and update the application.

Creating and deploying a Model

Using Cloudera Machine Learning, you can create any function within a script and deploy it to a REST API. In a Cloudera Machine Learning project, this is typically a predict function that accepts an input and returns a prediction based on the model's parameters.

As an example, we create a function that adds two numbers, and deploy it as a model that returns the sum of the numbers. This function will accept two numbers in JSON format as input and return the sum.



Note: In case of PBJ (Powered by Jupyter) Runtime, a specific decorator, the `cml_model` decorator is needed to create a model. This decorator allows a function to work as a model in a PBJ Runtime. The decorator can also be used to enable gathering of model metrics. For more details, see [Example models with PBJ Runtimes](#).

For CML UI

1. Create a new project. Note that models are always created within the context of a project.
2. Click New Session and launch a new Python 3 session.
3. Create a new file within the project called `add_numbers.py`. This is the file where we define the function that will be called when the model is run. For example:

`add_numbers.py`

```

def add(args):
    result = args["a"] + args["b"]
    return result

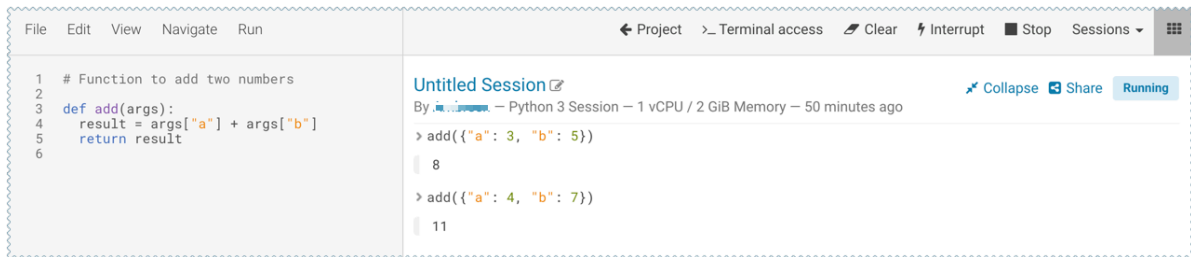
```



Note: In practice, do not assume that users calling the model will provide input in the correct format or enter good values. Always perform input validation.

4. Before deploying the model, test it by running the `add_numbers.py` script, and then by calling the `add` function directly from the interactive workbench session. For example:

```
add({ "a": 3, "b": 5 })
```



5. Deploy the `add` function to a REST endpoint.
- Go to the project Overview page.
 - Click **Models** **New Model**.
 - Give the model a Name and Description.
 - In **Deploy Model** as, if the model is to be deployed in a service account, select **Service Account** and choose the account from the dropdown menu.
 - Enter the details about the model that you want to build, for example:
 - File: `add_numbers.py`
 - Function: `add`
 - Example Input: `{"a": 3, "b": 5}`
 - Example Output: `8`

The screenshot shows the "Deploy Model" form in the Google Cloud console. The form has four sections:

- File ***: A text input field containing `add_numbers.py`.
- Function ***: A text input field containing `add`.
- Example Input ?**: A text area containing a JSON object: `{ "a": 3, "b": 5 }`.
- Example Output ?**: A text area containing the number `8`.

- Select the resources needed to run this model, including any replicas for load balancing. To specify the maximum number of replicas in a model deployment, go to **Site Administration Settings Model Deployment Settings**. The default is 9 replicas, and up to 199 can be set.
- Click **Deploy Model**.

6. Select the model to go on its Overview page. Click Builds to track realtime progress as the model is built and deployed. This process essentially creates a Docker container where the model will live and serve requests.

Add Two Numbers Building Stop Deploy New Build

Overview Deployments **Builds** Monitoring Settings

Build	Status	File	Function	Kernel	Engine Image	Created By	Created At	Comment	Actions
1	Building	add_numbers.py	add	python3	Base Image v	ambreen	Jun 5, 2018, 5:44 PM	Initial revision.	Delete

```

Sending build context to Docker daemon 15.05 MB

Step 1/16 : FROM docker.repository.cloudera.com/cdsw/engine:
----> f8955770daa1
Step 2/16 : ENTRYPOINT node /app/model-runtime/model-server.js
----> Running in 58038f1e58d5

```

7. Once the model has been deployed, go back to the model Overview page and use the Test Model widget to make sure the model works as expected.

If you entered example input when creating the model, the Input field will be pre-populated with those values. Click Test. The result returned includes the output response from the model, as well as the ID of the replica that served the request.

Model response times depend largely on your model code. That is, how long it takes the model function to perform the computation needed to return a prediction. Note that model replicas can only process one request at a time. Concurrent requests will be queued until the model can process them.

For CML APIv2

Create and deploy a model using the Models API as instructed in this example:

This example demonstrates the use of the Models API. To run this example, first do the following:

1. Create a project with the Python template and a legacy engine.
2. Start a session.
3. Run `!pip3 install sklearn`
4. Run `fit.py`

The example script first obtains the project ID, then creates and deploys a model.

```

projects = client.list_projects(search_filter=json.dumps({"name": "<your
project name>"}))
project = projects.projects[0] # assuming only one project is returned by
the above query
model_body = cmlapi.CreateModelRequest(project_id=project.id, name="Demo
Model", description="A simple model")
model = client.create_model(model_body, project.id)
model_build_body = cmlapi.CreateModelBuildRequest(project_id=project.id,
model_id=model.id, file_path="predict.py", function_name="predict", ker
nel="python3")
model_build = client.create_model_build(model_build_body, project.id, mod
el.id)
while model_build.status not in ["built", "build failed"]:
    print("waiting for model to build...")
    time.sleep(10)
    model_build = client.get_model_build(project.id, model.id, model_build
.id)
if model_build.status == "build failed":

```



```
print("model build failed, see UI for more information")
sys.exit(1)
print("model built successfully!")
model_deployment_body = cmlapi.CreateModelDeploymentRequest(project_id=p
project.id, model_id=model.id, build_id=model_build.id)
model_deployment = client.create_model_deployment(model_deployment_body,
project.id, model.id, build.id)
while model_deployment.status not in ["stopped", "failed", "deployed"]:
    print("waiting for model to deploy...")
    time.sleep(10)
    model_deployment = client.get_model_deployment(project.id, model.id, m
odel_build.id, model_deployment.id)
if model_deployment.status != "deployed":
    print("model deployment failed, see UI for more information")
    sys.exit(1)
print("model deployed successfully!")
```