Kedro Kubeflow Plugin

Release 0.5.1

GetInData

Jan 28, 2022

CONTENTS:

1 Introduction **1**
- 1.1 What is Kubeflow Pipelines? . . . 1
- 1.2 Why to integrate Kedro project with Pipelines? . . . 1

2 Installation **3**
- 2.1 Installation guide . . . 3
- 2.2 Configuration . . . 5

3 Getting started **7**
- 3.1 Quickstart . . . 7
- 3.2 GCP AI Platform support . . . 12
- 3.3 Mlflow support . . . 15
- 3.4 Continuous Deployment . . . 17
- 3.5 Authenticating to Kubeflow Pipelines API . . . 17

4 Indices and tables **19**

CHAPTER

ONE

INTRODUCTION

1.1 What is Kubeflow Pipelines?

Kubeflow Pipelines is a platform for building and deploying portable, scalable machine learning (ML) workflows based on Docker containers. It works by defining pipelines with nodes (Kubernetes objects, like pod or volume) and edges (dependencies between the nodes, like passing output data as input). The pipelines are stored in the versioned database, allowing user to run the pipeline once or schedule the recurring run.

1.2 Why to integrate Kedro project with Pipelines?

Kubeflow Pipelines' main attitude is the portability. Once you define a pipeline, it can be started on any Kubernetes cluster. The code to execute is stored inside docker images that cover not only the source itself, but all the libraries and entire execution environment. Portability is also one of key Kedro aspects, as the pieplines must be versionable and packagebale. Kedro, with Kedro-docker plugin do a fantastic job to achieve this and Kubeflow looks like a nice addon to run the pipelines on powerful remote Kubernetes clusters.

CHAPTER

TWO

INSTALLATION

2.1 Installation guide

2.1.1 Kedro setup

First, you need to install base Kedro package in `<17.0` version

> Kedro 17.0 is supported by kedro-kubeflow, but not by kedro-mlflow yet, so the latest version from 0.16 family is recommended.

```
$ pip install 'kedro<0.17'
```

2.1.2 Plugin installation

Install from PyPI

You can install `kedro-kubeflow` plugin from `PyPi` with `pip`:

```
pip install --upgrade kedro-kubeflow
```

Install from sources

You may want to install the develop branch which has unreleased features:

```
pip install git+https://github.com/getindata/kedro-kubeflow.git@develop
```

2.1.3 Available commands

You can check available commands by going into project directory and runnning:

```
$ kedro kubeflow
Usage: kedro kubeflow [OPTIONS] COMMAND [ARGS]...

  Interact with Kubeflow Pipelines

Options:
  -e, --env TEXT  Environment to use.
```

(continues on next page)

(continued from previous page)

```
  -h, --help      Show this message and exit.

Commands:
  compile          Translates Kedro pipeline into YAML file with Kubeflow...
  init             Initializes configuration for the plugin
  list-pipelines   List deployed pipeline definitions
  run-once         Deploy pipeline as a single run within given experiment.
  schedule         Schedules recurring execution of latest version of the...
  ui               Open Kubeflow Pipelines UI in new browser tab
  upload-pipeline  Uploads pipeline to Kubeflow server
```

init

`init` command takes one argument (that is the kubeflow pipelines root url) and generates sample configuration file in `conf/base/kubeflow.yaml`. The YAML file content is described in the Configuration section.

ui

`ui` command opens a web browser pointing to the currently configured Kubeflow Pipelines UI. It's super useful for debugging, especially while working on multiple Kubeflow installations.

list-pipelines

`list-pipelines` uses Kubeflow Pipelines to retrieve all registered pipelines

compile

`compile` transforms Kedro pipeline into Argo workflow (Argo is the engine that powers Kubeflow Pipelines). The resulting `yaml` file can be uploaded to Kubeflow Pipelines via web UI.

upload-pipeline

`upload-pipeline` compiles the pipeline and uploads it as a new pipeline version. The pipeline name is equal to the project name for simplicity.

schedule

`schedule` creates recurring run of the previously uploaded pipeline. The cron expression (required parameter) is used to define at what schedule the pipeline should run.

`run-once`

`run-once` is all-in-one command to compile the pipeline and run it in the Kubeflow environment.

2.2 Configuration

Plugin maintains the configuration in the `conf/base/kubeflow.yaml` file. Sample configuration can be generated using `kedro kubeflow init`:

```yaml
# Base url of the Kubeflow Pipelines, should include the schema (http/https)
host: https://kubeflow.example.com/pipelines

# Configuration used to run the pipeline
run_config:

  # Name of the image to run as the pipeline steps
  image: kubeflow-plugin-demo

  # Pull pilicy to be used for the steps. Use Always if you push the images
  # on the same tag, or Never if you use only local images
  image_pull_policy: IfNotPresent

  # Location of Vertex AI GCS root, required only for vertex ai pipelines configuration
  root: bucket_name/gcs_suffix

  # Name of the kubeflow experiment to be created
  experiment_name: Kubeflow Plugin Demo

  # Name of the run for run-once
  run_name: Kubeflow Plugin Demo Run

  # Optional pipeline description
  description: Very Important Pipeline

  # Flag indicating if the run-once should wait for the pipeline to finish
  wait_for_completion: False

  # How long to keep underlying Argo workflow (together with pods and data
  # volume after pipeline finishes) [in seconds]. Default: 1 week
  ttl: 604800

  # Optional volume specification
  volume:

    # Storage class - use null (or no value) to use the default storage
    # class deployed on the Kubernetes cluster
    storageclass: # default

    # The size of the volume that is created. Applicable for some storage
    # classes
    size: 1Gi
```

(continues on next page)

(continued from previous page)

```
    # Access mode of the volume used to exchange data. ReadWriteMany is
    # preferred, but it is not supported on some environements (like GKE)
    # Default value: ReadWriteOnce
    #access_modes: [ReadWriteMany]

    # Flag indicating if the data-volume-init step (copying raw data to the
    # fresh volume) should be skipped
    skip_init: False

    # Allows to specify user executing pipelines within containers
    # Default: root user (to avoid issues with volumes in GKE)
    owner: 0

    # Flak indicating if volume for inter-node data exchange should be
    # kept after the pipeline is deleted
    keep: False

  # Optional section allowing adjustment of the resources
  # reservations and limits for the nodes
  resources:

    # For nodes that require more RAM you can increase the "memory"
    data_import_step:
      memory: 2Gi

    # Training nodes can utilize more than one CPU if the algoritm
    # supports it
    model_training:
      cpu: 8
      memory: 1Gi

    # GPU-capable nodes can request 1 GPU slot
    tensorflow_step:
      nvidia.com/gpu: 1

    # Default settings for the nodes
    __default__:
      cpu: 200m
      memory: 64Mi
```

2.2.1 Dynamic configuration support

`kedro-kubeflow` contains hook that enables TemplatedConfigLoader. It allows passing environment variables to configuration files. It reads all environment variables following `KEDRO_CONFIG_<NAME>` pattern, which you can later inject in configuration file using `${name}` syntax.

There are two special variables `KEDRO_CONFIG_COMMIT_ID`, `KEDRO_CONFIG_BRANCH_NAME` with support specifying default when variable is not set, e.g. `${commit_id|dirty}`

CHAPTER

THREE

GETTING STARTED

3.1 Quickstart

3.1.1 Preprequisites

The quickstart assumes user have access to Kubeflow Pipelines deployment. Pipelines can be dedployed on any Kubernetes cluster, including local cluster.

3.1.2 Install the toy project with Kubeflow Pipelines support

It is a good practice to start by creating a new virtualenv before installing new packages. Therefore, use `virtalenv` command to create new env and activate it:

```
$ virtualenv venv-demo
created virtual environment CPython3.8.5.final.0-64 in 145ms
  creator CPython3Posix(dest=/home/mario/kedro/venv-demo, clear=False, no_vcs_
↪ignore=False, global=False)
  seeder FromAppData(download=False, pip=bundle, setuptools=bundle, wheel=bundle,␣
↪via=copy, app_data_dir=/home/mario/.local/share/virtualenv)
    added seed packages: pip==20.3.1, setuptools==51.0.0, wheel==0.36.2
  activators BashActivator,CShellActivator,FishActivator,PowerShellActivator,
↪PythonActivator,XonshActivator
$ source venv-demo/bin/activate
```

Then, `kedro` must be present to enable cloning the starter project, along with the latest version of `kedro-kubeflow` plugina and kedro-docker (required to build docker images with the Kedro pipeline nodes):

```
$ pip install 'kedro<0.17' kedro-kubeflow kedro-docker
```

With the dependencies in place, let's create a new project:

```
$ kedro new --starter=git+https://github.com/getindata/kedro-starter-spaceflights.git --
↪checkout allow_nodes_with_commas
Project Name:
=============
Please enter a human readable name for your new project.
Spaces and punctuation are allowed.
 [New Kedro Project]: Kubeflow Plugin Demo

Repository Name:
```

(continues on next page)

(continued from previous page)

```
================
Please enter a directory name for your new project repository.
Alphanumeric characters, hyphens and underscores are allowed.
Lowercase is recommended.
 [kubeflow-plugin-demo]:

Python Package Name:
====================
Please enter a valid Python package name for your project package.
Alphanumeric characters and underscores are allowed.
Lowercase is recommended. Package name must start with a letter or underscore.
 [kubeflow_plugin_demo]:

Change directory to the project generated in /home/mario/kedro/kubeflow-plugin-demo

A best-practice setup includes initialising git and creating a virtual environment␣
↪before running `kedro install` to install project-specific dependencies. Refer to the␣
↪Kedro documentation: https://kedro.readthedocs.io/
```

TODO: switch to the official `spaceflights` starter after https://github.com/quantumblacklabs/kedro-starter-spaceflights/pull/10 is merged

Finally, go the demo project directory and ensure that kedro-kubeflow plugin is activated:

```
$ cd kubeflow-plugin-demo/
$ kedro install
(...)
Requirements installed!
$ kedro kubeflow --help
Usage: kedro kubeflow [OPTIONS] COMMAND [ARGS]...

  Interact with Kubeflow Pipelines

Options:
  -e, --env TEXT  Environment to use.
  -h, --help      Show this message and exit.

Commands:
  compile          Translates Kedro pipeline into YAML file with Kubeflow...
  init             Initializes configuration for the plugin
  list-pipelines   List deployed pipeline definitions
  mlflow-start
  run-once         Deploy pipeline as a single run within given experiment.
  schedule         Schedules recurring execution of latest version of the...
  ui               Open Kubeflow Pipelines UI in new browser tab
  upload-pipeline  Uploads pipeline to Kubeflow server
```

3.1.3 Build the docker image to be used on Kubeflow Pipelines runs

First, initialize the project with `kedro-docker` configuration by running:

```
$ kedro docker init
```

This command creates a several files, including `.dockerignore`. This file ensures that transient files are not included in the docker image and it requires small adjustment. Open it in your favourite text editor and extend the section `# except the following` by adding there:

```
!data/01_raw
```

This change enforces raw data existence in the image. Also, one of the limitations of running the Kedro pipeline on Kubeflow (and not on local environemt) is inability to use MemoryDataSets, as the pipeline nodes do not share memory, so every artifact should be stored as file. The `spaceflights` demo configures four datasets as in-memory, so let's change the behaviour by adding these lines to `conf/base/catalog.yml`:

```
X_train:
  type: pickle.PickleDataSet
  filepath: data/05_model_input/X_train.pickle
  layer: model_input

y_train:
  type: pickle.PickleDataSet
  filepath: data/05_model_input/y_train.pickle
  layer: model_input

X_test:
  type: pickle.PickleDataSet
  filepath: data/05_model_input/X_test.pickle
  layer: model_input

y_test:
  type: pickle.PickleDataSet
  filepath: data/05_model_input/y_test.pickle
  layer: model_input
```

Finally, build the image:

```
kedro docker build
```

When execution finishes, your docker image is ready. If you don't use local cluster, you should push the image to the remote repository:

```
docker tag kubeflow_plugin_demo:latest remote.repo.url.com/kubeflow_plugin_demo:latest
docker push remote.repo.url.com/kubeflow_plugin_demo:latest
```

3.1.4 Run the pipeline on Kubeflow

First, run `init` script to create the sample configuration. A parameter value should reflect the kubeflow base path **as seen from the system** (so no internal Kubernetes IP unless you run the local cluster):

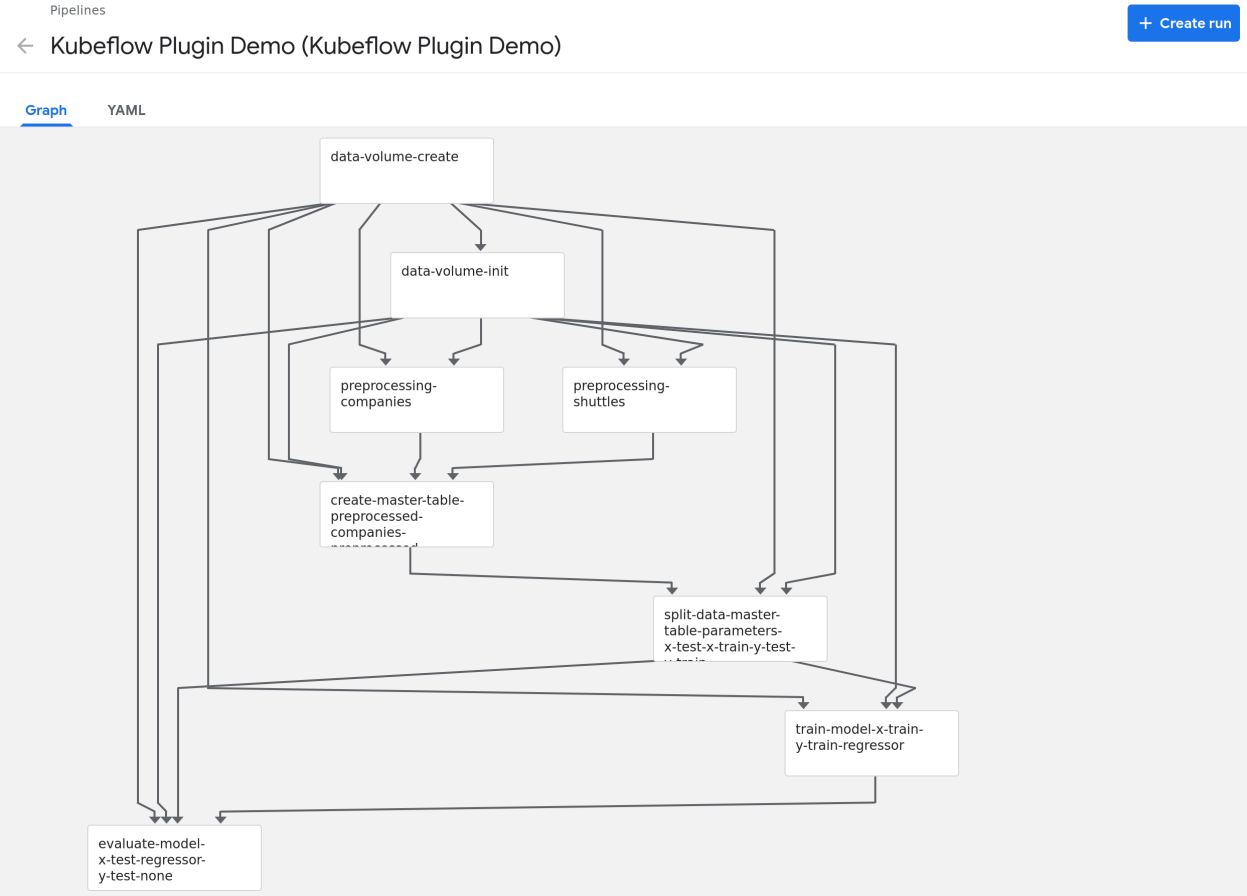
```
kedro kubeflow init https://kubeflow.cluster.com
(...)
Configuration generated in /home/mario/kedro/kubeflow-plugin-demo/conf/base/kubeflow.yaml
```

Then, if needed, adjust the `conf/base/kubeflow.yaml`. For example, the `image:` key should point to the full image name (like `remote.repo.url.com/kubeflow_plugin_demo:latest` if you pushed the image at this name). Depending on the storage classes availability in Kubernetes cluster, you may want to modify `volume.storageclass` and `volume.access_modes` (please consult with Kubernetes admin what values should be there).

Finally, everything is set to run the pipeline on Kubeflow. Run `upload-pipeline`:

```
$ kedro kubeflow upload-pipeline
2021-01-12 09:47:35,132 - kedro_kubeflow.kfpclient - INFO - No IAP_CLIENT_ID provided,␣
↪skipping custom IAP authentication
2021-01-12 09:47:35,209 - kedro_kubeflow.kfpclient - INFO - Pipeline created
2021-01-12 09:47:35,209 - kedro_kubeflow.kfpclient - INFO - Pipeline link: https://
↪kubeflow.cluster.com/#/pipelines/details/9a3e4e16-1897-48b5-9752-d350b1d1faac/version/
↪9a3e4e16-1897-48b5-9752-d350b1d1faac
```

As you can see, the pipeline was compiled and uploaded into Kubeflow. Let's visit the link:



Uploaded pipeline

The Kubeflow pipeline reflects the Kedro pipeline with two extra steps:

- `data-volume-create` - creates an empty volume in Kubernetes cluster as a persistence layer for inter-steps data access
- `data-volume-init` - initialized the volume with `01_raw` data when the pipeline starts

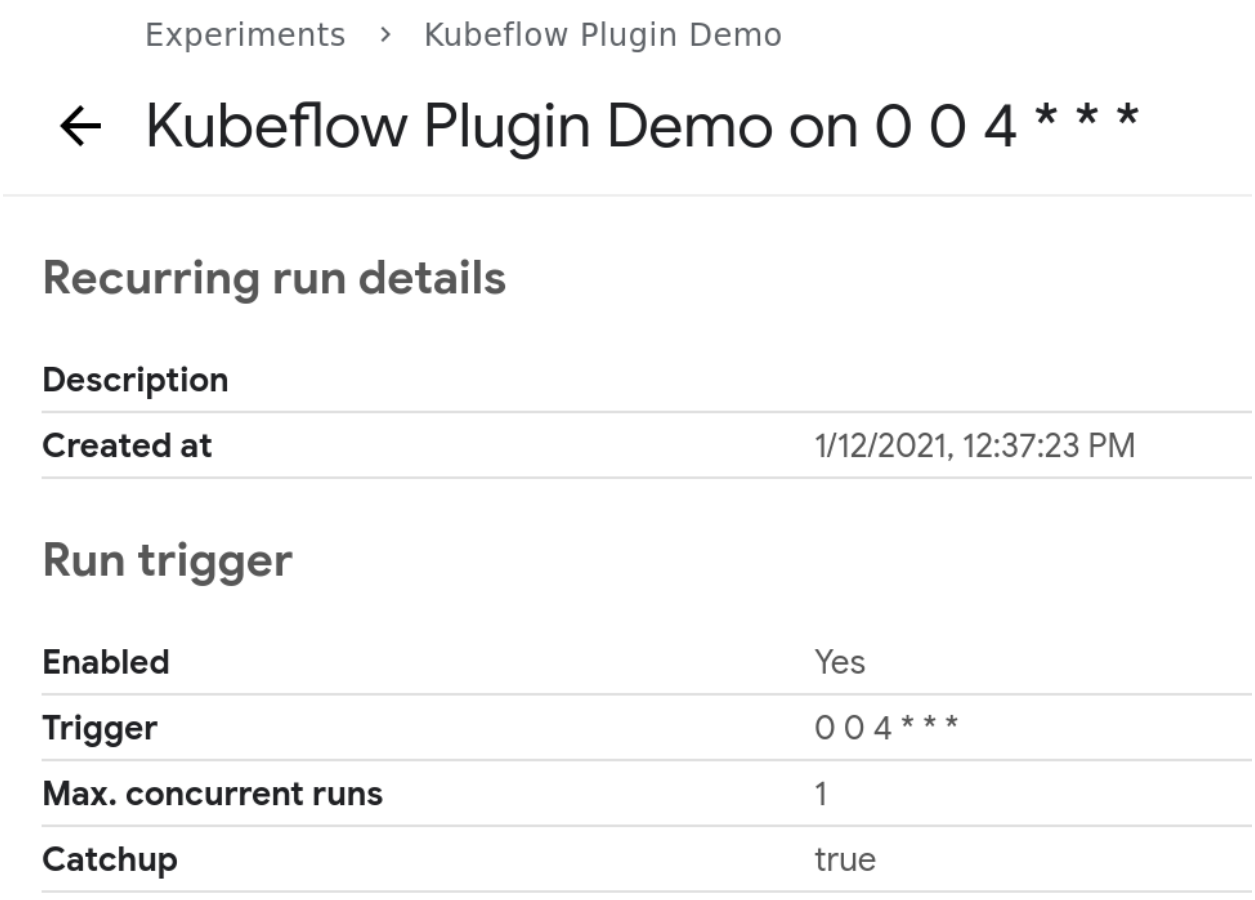
By using `Create run` button you can start a run of the pipeline on the cluster. A run behaves like `kedro run` command, but the steps are executed on the remote cluster. The outputs are stored on the persistent volume, and passed as the inputs accordingly to how Kedro nodes need them.

Pipeline run

From the UI you can access the logs of the execution. If everything seems fine, use `schedule to create a recurring run:

```
$ kedro kubeflow schedule --cron-expression '0 0 4 * * *'
(...)
2021-01-12 12:37:23,086 - kedro_kubeflow.kfpclient - INFO - No IAP_CLIENT_ID provided,␣
↪skipping custom IAP authentication
2021-01-12 12:37:23,096 - root - INFO - Creating experiment Kubeflow Plugin Demo.
2021-01-12 12:37:23,103 - kedro_kubeflow.kfpclient - INFO - New experiment created:␣
↪2123c082-b336-4093-bf3f-ce73f68b66b4
2021-01-12 12:37:23,147 - kedro_kubeflow.kfpclient - INFO - Pipeline scheduled to 0 0 4␣
↪* * *
```

You can see that the new experiment was created (that will group the runs) and the pipeline was scheduled. Please note, that Kubeflow uses 6-places cron expression (as opposite to Linux's cron with 5-places), where first place is the second indicator.



Scheduled run

3.2 GCP AI Platform support

Google Cloud's AI Platform offers couple services that simplify Machine Learning tasks with use of Kubeflow based components.

3.2.1 Using `kedro` with AI Platform Notebooks

AI Platform Notebooks provides an easy way to manage and host JupyterLab based data science workbench environment. What we've found out is that the default images provided by a service cause some dependency conflicts. To avoid this issues make sure you use isolated virtual environment, e.g. virtualenv. New virual environment can be created by simply invoking `python -m virtualenv venv` command.

3.2.2 Using `kedro-kubeflow` with AI Platform Pipelines

AI Platform Pipelines is a service that allows to easily deploy Kubeflow Pipelines on new or existing Google Kubernetes Engine clusters.

In general `kedro-kubeflow` plugin should work with AI Platform Pipelines out of the box, with the only exception is that it requires authentication. Note that the `host` variable should point to a dashbard URL generated by AI Platform Pipelines service (e.g. https://653hddae86eb7b0-dot-europe-west1.pipelines.googleusercontent.com/), just open the dashboard from the service page and copy url from the browser.

Below is the list of authentication scenarios supported so far:

1. Connecting to AI Pipelines from AI Platform Notebooks

In this scenario authentication works out of the box with *default credentials* mechanism.

2. Authentication to AI Pipelines from local environment

To interact with AI Platform Pipelines from local environment you can use the mechanisms provided by Google Cloud SDK. After installing the SDK run `google cloud application-default login` to initialize *default credentials* on your local machine.

You can use service account key for authentication as well. To make that work just set `GOOGLE_APPLICATION_CREDENTIALS` environment variable to the path of where the service account key file is stored.

3. Authenticating through IAP Proxy

Identity Aware Proxy is a product that allows securing your cloud based applications with Google Identity.

To authenticate with IAP find out which *oauth client ID* is the proxy configured with and then save it in `IAP_CLIENT_ID` environment variable. The authentication should work seamlessly assuming identity you are using has been granted access to the application.

The above will work if you are connecting from within GCP VM or locally with specified service account credentials. It will *NOT* work for credentials obtained with `google cloud application-default login`.

3.2.3 Using `kedro-kubeflow` with Vertex AI Pipelines (EXPERIMENTAL)

Vertex AI Pipelines is a fully managed service that allows to easily deploy Kubeflow Pipelines on a serverless Google service. Vertex AI Pipelines was still in a Preview mode when this plugin version was released, therefore plugin capability is also limited.

1. Preparing configuration

In order the plugin picks Vertex AI Pipelines as a target infrastructure, it has to be indicated in configuration. As the solution is serverless, no URL is to be provided. Instead, special set of parameters has to be passed, so that connection is established with proper GCP service.

```
host: vertex-ai-pipelines
project_id: hosting-project
region: europe-west4
run_config:
  root: vertex-ai-pipelines-accessible-gcs-bucket/pipelines-specific-path
```

If the pipeline requires access to services that are not exposed to public internet, you need to configure VPC peering between Vertex internal network and VPC that hosts the internal service and then set the VPC identifier in the configuration. Optionally, you can add custom host aliases:

```
run_config:
  vertex_ai_networking:
    vpc: projects/12345/global/networks/name-of-vpc
    host_aliases:
    - ip: 10.10.10.10
      hostnames: ['mlflow.internal']
    - ip: 10.10.20.20
      hostnames: ['featurestore.internal']
```

2. Preparing environment variables

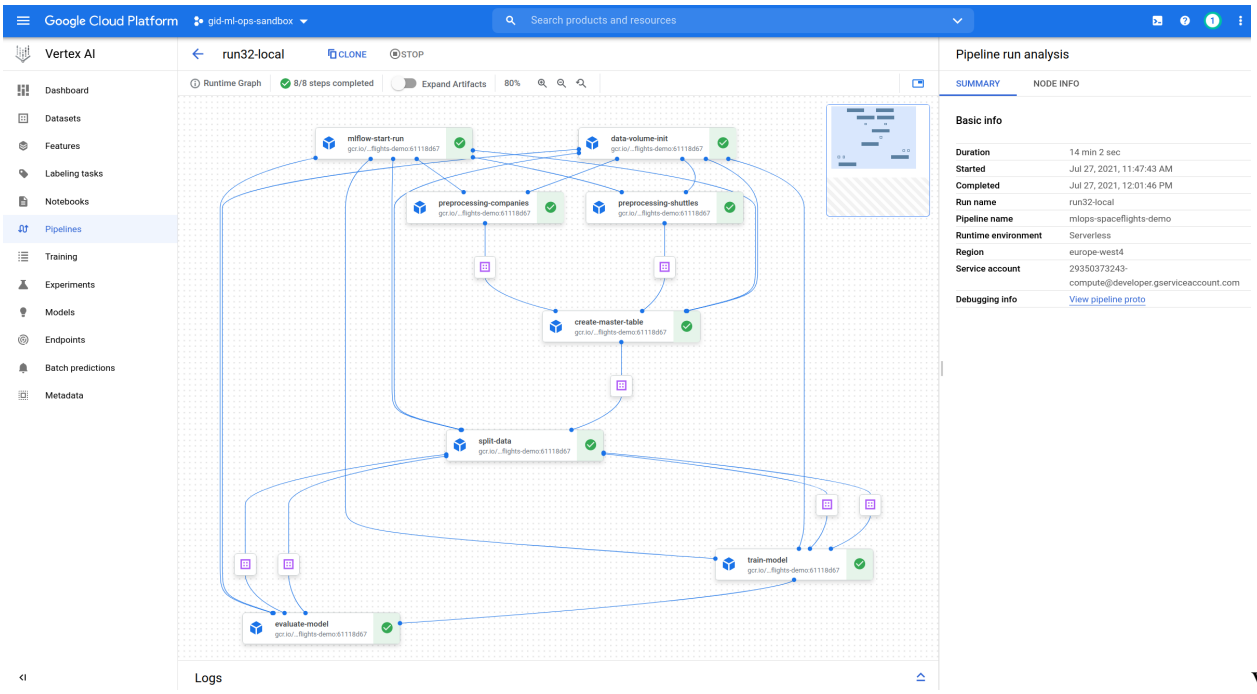
There're the following specific environment variables required for the pipeline to run correctly:

- SERVICE_ACCOUNT - full email of service account that job will use to run the pipeline. Account has to have access to `run_config.root` path. Variable is optional, if no given, project compute account is used
- MLFLOW_TRACKING_TOKEN - identity token required if MLFlow is used inside the project and mlflow access is protected. Token is passed as it is to kedro nodes in order to authenticate against MLFlow service. Can be generated via `gcloud auth print-identity-token` command.

3. Supported commands

Following commands are supported:

```
kedro kubeflow compile
kedro kubeflow run-once
kedro kubeflow schedule
kedro kubeflow list-pipelines
```



Vertex_AI_Pipeli

3.3 Mlflow support

If you use MLflow and kedro-mlflow for the Kedro pipeline runs monitoring, the plugin will automatically enable support for:

- starting the experiment when the pipeline starts,
- logging all the parameters, tags, metrics and artifacts under unified MLFlow run.

To make sure that the plugin discovery mechanism works, add `kedro-mlflow` and `kedro-kubeflow` as a dependencies to `src/requirements.in` and run:

```
$ pip-compile src/requirements.in > src/requirements.txt
$ kedro install
$ kedro mlflow init
```

Then, adjust the kedro-mlflow configuration and point to the mlflow server by editing `conf/base/mlflow.yml` and adjusting `mlflow_tracking_uri` key. Then, build the image:

```
$ kedro docker build
```

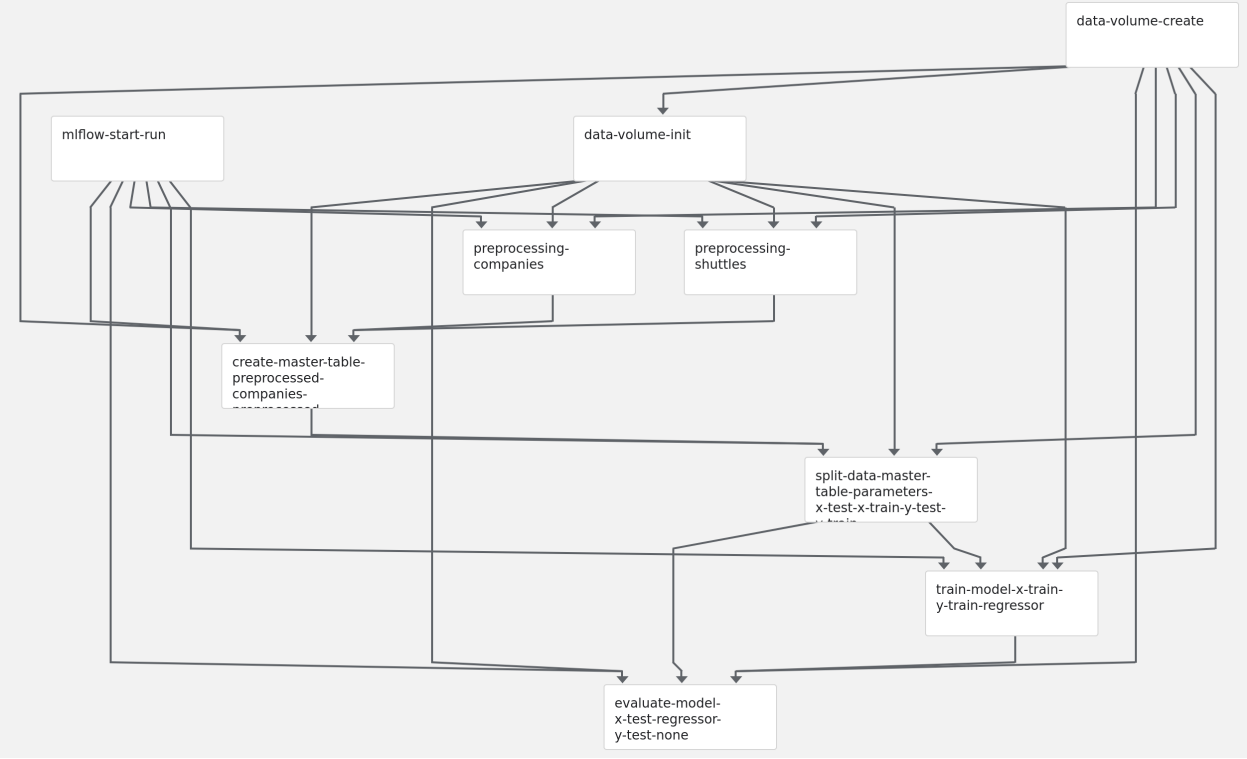
And re-push the image to the remote registry. Finally, reupload the pipeline:

```
$ kedro kubeflow upload-pipeline
(...)
2021-01-12 13:05:56,879 - kedro_kubeflow.kfpclient - INFO - No IAP_CLIENT_ID provided,␣
↪skipping custom IAP authentication
2021-01-12 13:05:56,973 - kedro_kubeflow.kfpclient - INFO - New version of pipeline␣
↪created: ba3a05c2-2f19-40c5-809e-0ed7c2989000
2021-01-12 13:05:56,973 - kedro_kubeflow.kfpclient - INFO - Pipeline link: http://10.43.
↪54.89/#/pipelines/details/9a3e4e16-1897-48b5-9752-d350b1d1faac/version/ba3a05c2-2f19-
↪40c5-809e-0ed7c2989000
```

(continues on next page)

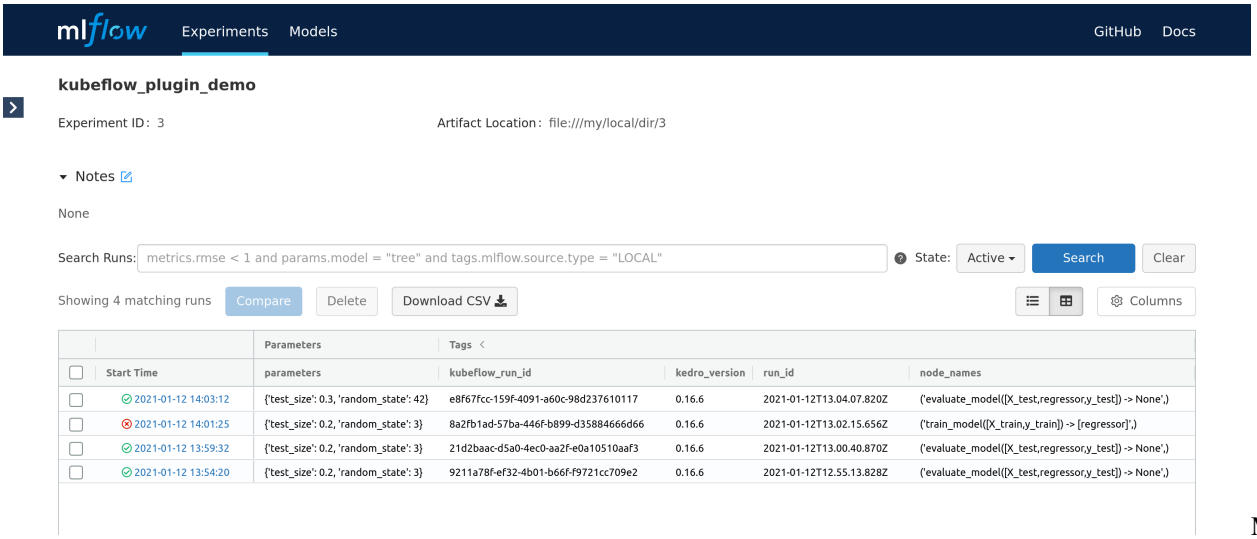
(continued from previous page)

And verify how does it look in the Kubeflow UI. You should notice `mlflow-start-run` step on the very top:



Pipeline with Mlflow

Finally, start the pipeline. While it executes, the new Mlflow run is started and it's constantly updated with the attributes provided by the next steps. Finally, the experiments runs page looks like:



Mlflow UI

The UI presents the pipeline stauts (in form of the icon) and latest node that was run (for failed runs in indicates at what step did the pipeline fail). Also, the `kubeflow_run_id` tag can be used to correlate Mlflow run with the Kubeflow pipeline execution.

3.4 Continuous Deployment

With kedro pipelines started on the remote Kubeflow Pipelnes clusters, changes in the code require re-building docker images and (sometimes) changing the pipeline structure. To simplify this workflow, Kedro-kubeflow plugin is capable of creating configuration for the most popular CI/CD automation tools.

The autogenerated configuration defines these actions:

- on any new push to the repository - image is re-built and the pipeline is started using `run-once`,
- on merge to master - image is re-built, the pipeline is registered in the Pipelines and scheduled to execute on the daily basis.

The behaviour and parameters (like schedule expression) can be adjusted by editing the genrated files. The configuration assumes that Google Container Registry is used to store the images, but users can freely adapt it to any (private or public) docker images registry.

3.4.1 Github Actions

If the Kedro project is stored on github (either in private or public repository), Github Actions can be used to automate the Continuous Deployment. To configure the repository, go to Settings->Secrets and add there:

- `GKE_PROJECT`: ID of the google project.
- `GKE_SA_KEY`: service account key, encoded with base64 (this service account must have access to push images into registry),
- `IAP_CLIENT_ID`: id of the IAP proxy client to communicate with rest APIs.

Next, re-configure the project using

```
kedro kubeflow init --with-github-actions https://<endpoint_name>.endpoints.<project-
↪name>.cloud.goog/pipelines
```

This command will generate Github Actions in `.github/workflows` directory. Then push the code to any branch and go to "Actions" tab in Github interface.

3.5 Authenticating to Kubeflow Pipelines API

Plugin supports 2 ways of authenticating to Kubeflow Pipelines API:

3.5.1 1. KFP behind IAP proxy on Google Cloud

It's already described in GCP AI Platform support chapter.

3.5.2 2. KFP behind Dex+authservice

Dex is the recommended authentication mechanism for on-premise Kubeflow clusters. The usual setup looks in a way that:

- oidc-autheservice redirect unauthenticated users to Dex,
- Dex authenticates user in remote system, like LDAP or OpenID and also acts as OpenID provider,
- oidc-autheservice asks Dex for token and creates the session used across entire Kubeflow.

In order to use `kedro-kubeflow` behind Dex-secured clusters, use the following manual:

1. Setup staticPassword authentication method and add a user that you're going to use as CI/CD account.
2. Point your Kedro project to `/pipeline` API on Kubeflow, for example: `https://kubeflow.local/pipeline`
3. Set environment variables `DEX_USERNAME` and `DEX_PASSWORD` before calling `kedro kubeflow`

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search