Kedro Kubeflow Plugin

Release 0.7.4

GetInData

Feb 27, 2023

CONTENTS:

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 integrate a 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 also all the libraries and entire execution environment. Portability is also one of the key Kedro qualities, as the pipelines must be versionable and packageable. Kedro, with Kedro-docker plugin does a fantastic job to achieve this and Kubeflow looks like a nice add-on 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 kedro<0.19.0,>=0.18.1 version

```
$ pip install 'kedro<0.19.0,>=0.18.1'
```

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 running:

```
$ kedro kubeflow
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:
```

(continues on next page)

(continued from previous page)

```
  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`:

```
# 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

  # Name of the kubeflow experiment to be created
  experiment_name: Kubeflow Plugin Demo [${branch_name|local}]

  # Name of the run for run-once, templated with the run-once parameters
  run_name: Kubeflow Plugin Demo Run ${pipeline_name} ${branch_name|local} ${commit_
↪id|local}

  # Name of the scheduled run, templated with the schedule parameters
  scheduled_run_name: Kubeflow Plugin Demo Recurring Run ${pipeline_name}

  # 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

  # What Kedro pipeline should be run as the last step regardless of the
  # pipeline status. Used to send notifications or raise the alerts
  # on_exit_pipeline: notify_via_slack

  # This sets the caching option for pipeline using
  # execution_options.caching_strategy.max_cache_staleness
  # See https://en.wikipedia.org/wiki/ISO_8601 in section 'Duration'
  #max_cache_staleness: P0D

  # Set to false to disable kfp artifacts exposal
  # This setting can be useful if you don't want to store
  # intermediate results in the MLMD
  #store_kedro_outputs_as_kfp_artifacts: True

  # Strategy used to generate Kubeflow pipeline nodes from Kedro nodes
```

(continues on next page)

(continued from previous page)

```
  # Available strategies:
  #  * none (default) - nodes in Kedro pipeline are mapped to separate nodes
  #                     in Kubeflow pipelines. This strategy allows to inspect
  #                     a whole processing graph in Kubeflow UI and override
  #                     resources for each node (because they are run in separate pods)
  #                     Although, performance may not be optimal due to potential
  #                     sharing of intermediate datasets through disk.
  #  * full - nodes in Kedro pipeline are mapped to one node in Kubeflow pipelines.
  #           This strategy mitigate potential performance issues with `none` strategy
  #           but at the cost of degraded user experience within Kubeflow UI: a graph
  #           is collapsed to one node.
  #node_merge_strategy: none

  # 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

    # 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 tolerations for the nodes
  tolerations:
    __default__:
    - key: "dedicated"
      operator: "Equal"
      value: "ml-ops"
      effect: "NoSchedule"
    node_a:
    - key: "dedicated"
      operator: "Equal"
      value: "gpu_workload"
```

(continues on next page)

(continued from previous page)

```
    effect: "NoSchedule"

# Optional section to allow mounting additional volumes (such as EmptyDir)
# to specific nodes
extra_volumes:
  tensorflow_step:
  - mount_path: /dev/shm
    volume:
      name: shared_memory
      empty_dir:
        cls: V1EmptyDirVolumeSource
        params:
          medium: Memory

# 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: 1
    memory: 1Gi

# Optional section to provide retry policy for the steps
# and default policy for steps with no policy specified
retry_policy:
  # 90 retries every 5 minutes
  wait_for_partition_availability:
    num_retries: 90
    backoff_duration: 5m
    backoff_factor: 1

  # 4 retries after: 1 minute, 2 minutes, 4 minutes, 8 minutes
  __default__:
    num_retries: 4
    backoff_duration: 60s
    backoff_factor: 2
```

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}`

2.2.2 Extra volumes

You can mount additional volumes (such as `emptyDir`) to specific nodes by using `extra_volumes` config node. The syntax of the configuration allows to define k8s SDK compatible class hierarchy similar to the way you would define it in the KFP DSL, e.g:

```
# KFP DSL
volume = dsl.PipelineVolume(volume=k8s.client.V1Volume(
    name="shared_memory",
    empty_dir=k8s.client.V1EmptyDirVolumeSource(medium='Memory')))

training_op.add_pvolumes({'/dev/shm': volume})
```

will translate to the following Kedro-Kubeflow config:

```
extra_volumes:
  training_op:
  - mount_path: /dev/shm
    volume:
      name: shared_memory
      empty_dir:
        cls: V1EmptyDirVolumeSource
        params:
          medium: Memory
```

In general, the `volume` key accepts a dictionary with the keys being the named parameters for the V1Volume and values being one of:

- dictionary with `cls` and `params` keys (to define nested objects) - see `kedro_kubeflow.config.ObjectKwargs`
- list of values / list of dictionaries (`kedro_kubeflow.config.ObjectKwargs`) as described above
- values (`str`, `int` etc.)

CHAPTER

THREE

GETTING STARTED

3.1 Quickstart

3.1.1 Prerequisites

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

Local kubeflow cluster

There is also an option to test locally with running Kubernetes in docker (kind). After going through that guide you should have Kubeflow up and running available at `http://localhost:9000`.

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` plugin and `kedro-docker` (required to build docker images with the Kedro pipeline nodes):

```
 $ pip install 'kedro<0.19.0,>=0.18.1' kedro-kubeflow kedro-docker
```

With the dependencies in place, let's create a new project (with the latest supported kedro version - 0.17.7):

```
$ kedro new --starter=spaceflights --checkout=0.17.7
```

```
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:
================
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/
```

Next go the demo project directory:

```
$ cd kubeflow-plugin-demo/
```

Before installing the dependencies, add the `kedro-kubeflow` to `requirements.*` in src:

```
$ echo kedro-kubeflow >> src/requirements*
```

Finally, ensure that kedro-kubeflow plugin is activated:

```
$ pip install -r src/requirements.txt
(...)
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
```

(continues on next page)

(continued from previous page)

```
  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 favorite text editor and extend the section `# except the following` by adding there:

```
$ echo !data/01_raw >> .dockerignore
```

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 and intermediate data step should be stored as a file. The `spaceflights` demo configures four datasets as in-memory, so we need to change that. Replace the `conf/base/catalog.yml` with the following:

```
companies:
  type: pandas.CSVDataSet
  filepath: data/01_raw/companies.csv
  layer: raw

reviews:
  type: pandas.CSVDataSet
  filepath: data/01_raw/reviews.csv
  layer: raw

shuttles:
  type: pandas.ExcelDataSet
  filepath: data/01_raw/shuttles.xlsx
  layer: raw
  load_args:
    engine: openpyxl

data_processing.preprocessed_companies:
  type: pandas.ParquetDataSet
  filepath: data/02_intermediate/preprocessed_companies.pq
  layer: intermediate

data_processing.preprocessed_shuttles:
  type: pandas.ParquetDataSet
  filepath: data/02_intermediate/preprocessed_shuttles.pq
  layer: intermediate

model_input_table:
  type: pandas.ParquetDataSet
```

(continues on next page)

(continued from previous page)

```
  filepath: data/03_primary/model_input_table.pq
  layer: primary

data_science.active_modelling_pipeline.regressor:
  type: pickle.PickleDataSet
  filepath: data/06_models/regressor_active.pickle
  versioned: true
  layer: models

data_science.candidate_modelling_pipeline.regressor:
  type: pickle.PickleDataSet
  filepath: data/06_models/regressor_candidate.pickle
  versioned: true
  layer: models

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

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

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

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

data_science.active_modelling_pipeline.regressor:
  type: pickle.PickleDataSet
  filepath: data/06_models/regressor.pickle
  versioned: true
  layer: models

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

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

data_science.candidate_modelling_pipeline.X_test:
```

(continues on next page)

(continued from previous page)

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

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

data_science.candidate_modelling_pipeline.regressor:
  type: pickle.PickleDataSet
  filepath: data/06_models/regressor.pickle
  versioned: true
  layer: models
```

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
```

Local cluster testing

The `kind` has its own docker registry that you need to upload the image to. However, since it does not have any connection to other registry we want to prevent it from trying to pull any image (see the docs). In order to do that, we need to tag the built docker image with any specific version. Let's use `demo` tag, as any tag other than `latest` will do.

Locate your image name (it should be the same as kedro project name) with:

```
$ docker images
```

Then tag your image with the following command:

```
$ docker tag <image>:latest <image>:demo
```

Then you need to upload the image from local registry to the kind registry. Here `kfp` is the cluster name (the same as *in linked guide*. Default cluster name is `kind`.

```
$ kind load docker-image <image>:demo --name kfp
```

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/user/kedro/kubeflow-plugin-demo/conf/base/kubeflow.yaml
```

Local cluster testing

For local cluster the link is the following: `http://localhost:9000`

Warning: Since kedro 0.17 there have been introduced name spaces to datasets which are not yet fully supported by this plugin as it causes issues within naming conventions of kfp artifacts. For now it's best to disable storage of kfp artifacts by adding/uncommenting the following line in `conf/base/kubeflow.yaml`:

```
store_kedro_outputs_as_kfp_artifacts: False
```

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've 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).

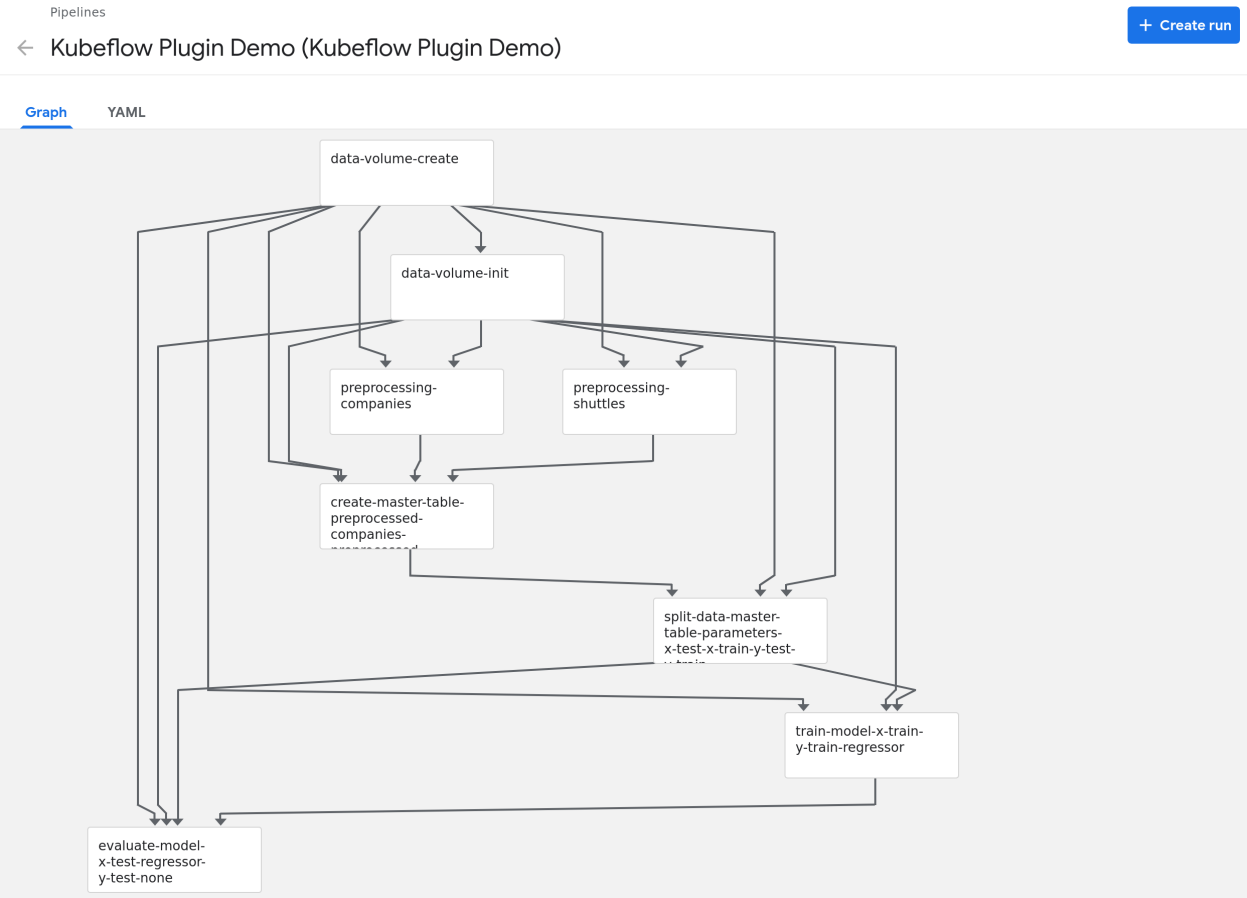
Local cluster testing

In this example you also need to update the tag of the `image:` part to also use `demo` instead latest.

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:



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.

Tip: You can also schedule a single run by using

```
$ kedro kubeflow run-once
```

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

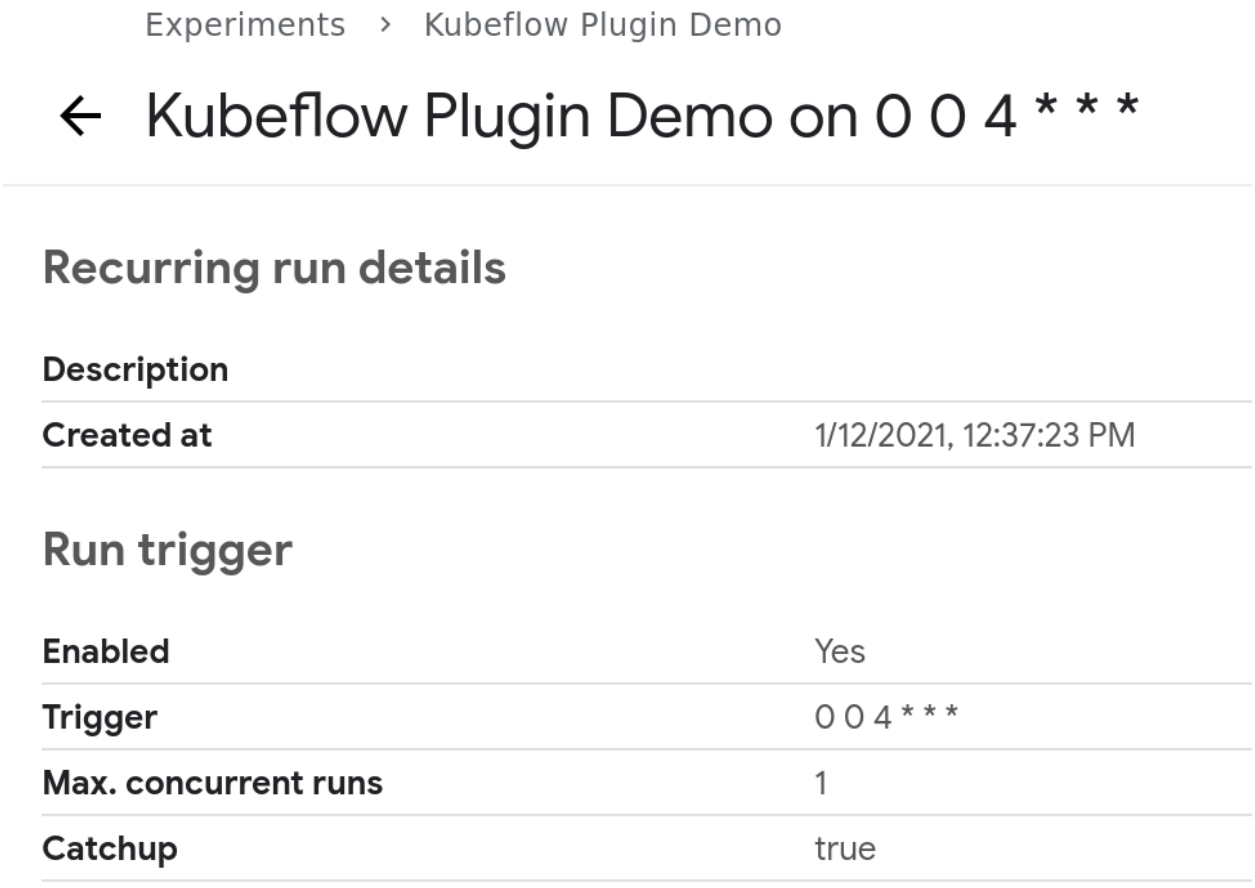
```
$ kedro kubeflow schedule --cron-expression '0 0 4 * * *'
(...)
2021-01-12 12:37:23,086 - kedro_kubeflow.kfpclient - INFO - No IAP_CLIENT_ID provided,␣
↪skipping custom IAP authentication
2021-01-12 12:37:23,096 - root - INFO - Creating experiment Kubeflow Plugin Demo.
2021-01-12 12:37:23,103 - kedro_kubeflow.kfpclient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

(continues on next page)

(continued from previous page)

```
↪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.



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 virtual 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 dashboard 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 (DEPRECATED)

Vertex AI Pipelines support in `kedro-kubeflow` has been deprecated in favor of the new plugin kedro-vertexai

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
```

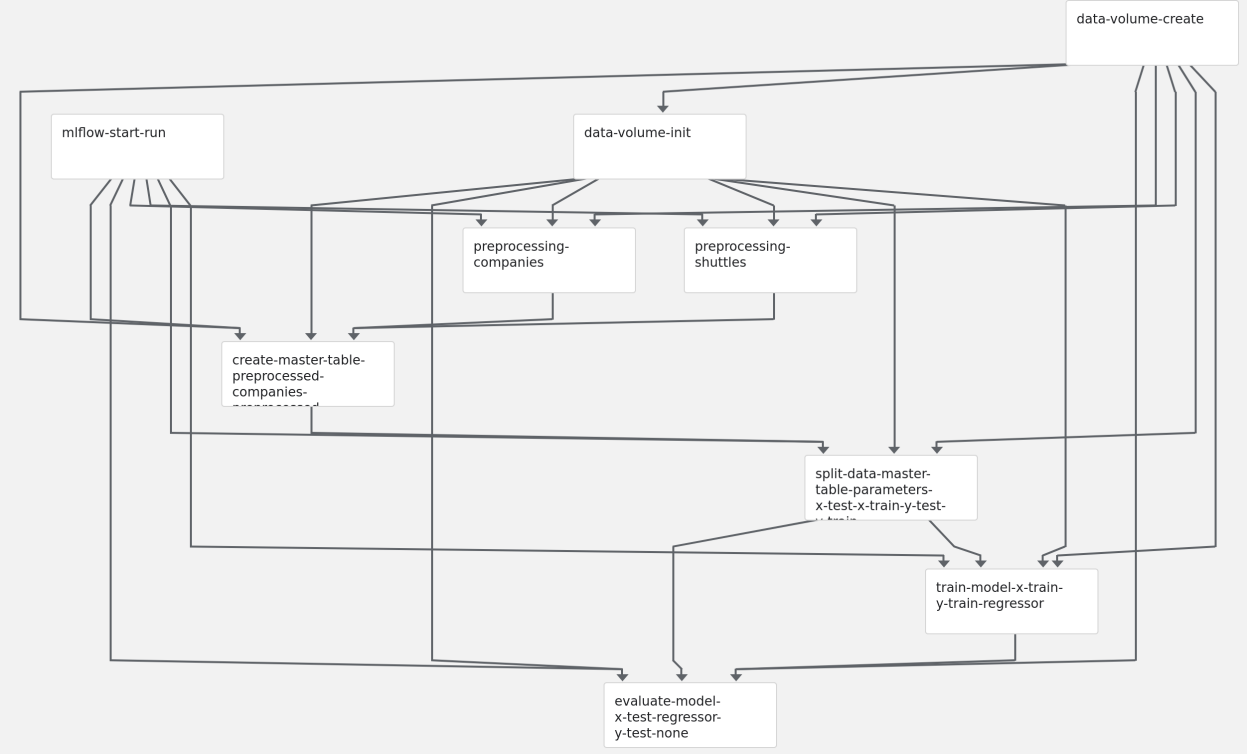
Then, adjust the kedro-mlflow configuration and point to the mlflow server by editing `conf/base/mlflow.yml` and adjusting `mlflow_tracking_uri` key. Then, build the image:

```
$ kedro docker build
```

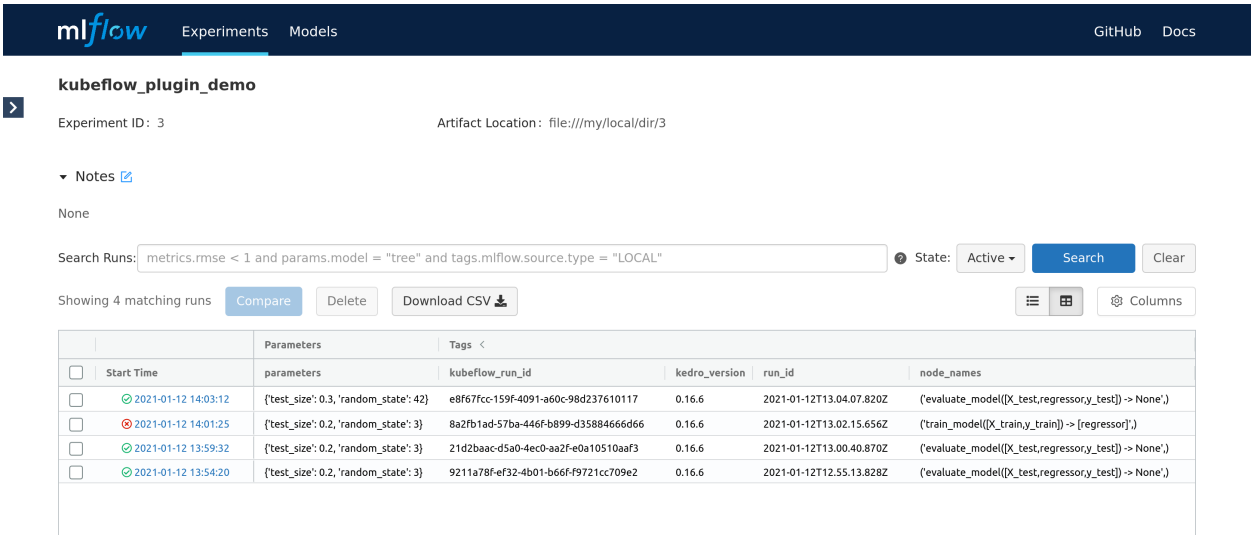
And re-push the image to the remote registry. Finally, re-upload the pipeline:

```
$ 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
```

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



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:



The UI presents the pipeline status (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 auto-generated 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 generated 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 with `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 a 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

CONTRIBUTING

4.1 PR Guidelines

1. Fork branch from `develop`.
2. Ensure to provide unit tests for new functionality.
3. Install dev requirements: `poetry install` and setup a hook: `pre-commit install`
4. Update documentation accordingly.
5. Update changelog according to "Keep a changelog" guidelines.
6. Squash changes with a single commit as much as possible and ensure verbose PR name. Open a PR against `develop`

- We reserve the right to take over and modify or abandon PRs that do not match the workflow or are abandoned.*

4.2 Release workflow

1. Create the release candidate:
   - Go to the Prepare release action.
   - Click "Run workflow"
   - Enter the part of the version to bump (one of `<major>.<minor>.<patch>`). Minor (x.**x**.x) is a default.
2. If the workflow has run successfully:
   - Go to the newly opened PR named `Release candidate `
   - Check that changelog and version have been properly updated. If not pull the branch and apply manual changes if necessary.
   - Merge the PR to master (merge commit, NOT squash)
3. Checkout the Publish workflow to see if:
   - The package has been uploaded on PyPI successfully
   - The changes have been merged back to develop

4.3 Local testing

4.3.1 Unit tests

The plugin has unit tests that can be run with `tox`, for example for python 3.8:

```
$ pip install tox-pip-version
$ tox -v -e py38
```

List available python environments to test with using `tox -l`.

You can also run them manually by executing `python -m unittest` in the root folder. They are also executed with github action on pull requests to test the stability of new changes. See `.github/workflows/python-package.yml`.

4.3.2 E2E tests

There is also a set up with Kubeflow running on team-maintained Google Cloud Platform. It tests the execution on said Kubeflow platform with `spaceflight` kedro starter. They are also automated with github action. See `.github/workflows/e2e-tests.yml`.

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search