Kedro Airflow K8S Plugin

Release 0.5.1

GetInData

May 17, 2021

CONTENTS:

CHAPTER

ONE

INTRODUCTION

1.1 What is Airflow?

Airflow is a platform to programmatically author, schedule and monitor workflows. Workflows are represented as DAGs. Each DAG is represented by nodes, that define job to be executed. The DAGs are stored in the file storage, allowing user to run the pipeline once or schedule the recurring run.

1.2 What is Kubernetes?

Kubernetes is a platform for managing containerized workloads and services, that facilitates both declarative configuration and automation.

1.3 Why to integrate Kedro project with Airflow nad Kubernetes?

Airflow's 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 Airflow 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-airflow-k8s, 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-airflow-k8s` plugin from `PyPi` with `pip`:

```
pip install --upgrade kedro-airflow-k8s
```

Install from sources

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

```
pip install git+https://github.com/getindata/kedro-airflow-k8s.git@develop
```

2.1.3 Available commands

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

```
$ kedro airflow-k8s

Usage: kedro airflow-k8s [OPTIONS] COMMAND [ARGS]...

Options:
-e, --env TEXT  Environment to use.
-p, --pipeline TEXT  Pipeline name to pick.
```

(continues on next page)

(continued from previous page)

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

Commands:
  compile          Create an Airflow DAG for a project
  init             Initializes configuration for the plugin
  list-pipelines   List pipelines generated by this plugin
  run-once         Uploads pipeline to Airflow and runs once
  schedule         Uploads pipeline to Airflow with given schedule
  ui               Open Apache Airflow UI in new browser tab
  upload-pipeline  Uploads pipeline to Airflow DAG location
```

compile

`compile` command takes one argument, which is the directory name containing configuration (relative to `conf` folder). As an outcome, `dag` directory contains python file with generated DAG.

init

`init` command adds default plugin configuration to the project, based on Apache Airflow CLI input. It also allows optionally adding github actions, to streamline project build and upload.

list-pipelines

`list-pipelines` lists all pipelines generated by this plugin which exist in Airflow server. All generated DAGs are tagged with tag `generated_with_kedro_airflow_k8s:$PLUGIN_VERSION` and the prefix of this tag is used to distinguish among the other tags.

run-once

`run-once` command generates DAG from the pipeline, uploads it Airflow DAG location and triggers the DAG run as soon as the new DAG instance is available. It optionally allows waiting for DAG run completion, checking if `success` status is returned.

schedule

`schedule` command takes three arguments, one is the directory name containing configuration (relative to `conf` folder), the second one is the output location of generated dag, the third is cron like expression that relates to Airflow DAG `schedule_interval`.

ui

`ui` simplifies access to Apache Airflow console. It also allows open UI for the specific DAG.

upload-pipeline

`upload-pipeline` command takes two arguments, one is the directory name containing configuration (relative to `conf` folder), the second one is the output location of generated dag.

2.2 Configuration

Plugin maintains the configuration in the `conf/base/airflow-k8s.yaml` file.

```
# Base url of the Apache Airflow, should include the schema (http/https)
host: https://airflow.example.com

# Directory from where Apache Airflow is reading DAGs definitions
output: gs://airflow-bucket-example-com

# Configuration used to run the pipeline
run_config:

    # Name of the image to run as the pipeline steps
    image: airflow-k8s-plugin-demo

    # Pull policy 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

    # Pod startup timeout in seconds - if timeout passes the pipeline fails, default to␣
↪600
    startup_time: 600

    # Namespace for Airflow pods to be created
    namespace: airflow

    # Name of the Airflow experiment to be created
    experiment_name: Airflow K8S Plugin Demo

    # Name of the dag as it's presented in Airflow
    run_name: airflow-k8s-plugin-demo

    # Apache Airflow cron expression for scheduled runs
    cron_expression: "@daily"

    # Optional pipeline description
    description: "Very Important Pipeline"

    # Optional volume specification
    volume:
        # Storage class - use null (or no value) to use the default storage
```

(continues on next page)

(continued from previous page)

```
        # 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 fsGroup executing pipelines within containers
        # Default: root user group (to avoid issues with volumes in GKE)
        owner: 0
        # If set to True, shared persistent volume will not be created at all and all␣
↪other parameters under
        # `volume` are discarded
        disabled: False

    # Optional resources specification
    resources:
        # Default configuration used by all nodes that do not declare the
        # resource configuration. It's optional. If node does not declare the resource
        # configuration, __default__ is assigned by default, otherwise cluster defaults
        # will be used.
        __default__:
            # Optional labels to be put into pod node selector
            node_selectors:
              #Labels are user provided key value pairs
              node_pool_label/k8s.io: example_value
            # Optional labels to apply on pods
            labels:
              running: airflow
            # Optional annotations to apply on pods
            annotations:
              iam.amazonaws.com/role: airflow
            # Optional list of kubernetes tolerations
            tolerations:
                - key: "group"
                  value: "data-processing"
                  effect: "NoExecute"
                - key: "group"
                  operator: "Equal",
                  value: "data-processing",
                  effect: "NoSchedule"
            requests:
                #Optional amount of cpu resources requested from k8s
                cpu: "1"
                Optional amount of memory resource requested from k8s
                memory: "1Gi"
            limits:
```

(continues on next page)

(continued from previous page)

```
                #Optional amount of cpu resources limit on k8s
                cpu: "1"
                #Optional amount of memory resource limit on k8s
                memory: "1Gi"
        # Other arbitrary configurations to use, for example to indicate some exception␣
↪resources
        huge_machines:
            node_selectors:
                big_node_pool: huge.10x
            requests:
                cpu: "16"
                memory: "128Gi"
            limits:
                cpu: "32"
                memory: "256Gi"
    # Optional external dependencies configuration
    external_dependencies:
        # Can just select dag as a whole
        - dag_id: upstream-dag
        # or detailed
        - dag_id: another-upstream-dag
        # with specific task to wait on
          task_id: with-precise-task
        # Maximum time (minute) to wait for the external dag to finish before this
        # pipeline fails, the default is 1440 == 1 day
          timeout: 2
        # Checks if the external dag exists before waiting for it to finish. If it
        # does not exists, fail this pipeline. By default is set to true.
          check_existence: False
        # Time difference with the previous execution to look at (minutes),
        # the default is 0 meaning no difference
          execution_delta: 10
```

2.2.1 Indicate resources in pipeline nodes

Every node declared in `kedro` pipelines is executed inside pod. Pod definition declares resources to be used based on provided plugin configuration and presence of the tag `resources` in `kedro` node definition.

If no such tag is present, plugin will assign `__default__` from plugin `resources` configuration. If no `__default__` is given in plugin `resources` configuration or no `resources` configuration is given, pod definition will not be given any information on how to allocate resources to pod, thus default k8s cluster values will be used.

```
# train_model node is assigned resources from `huge_machines` configuration, if no such␣
↪configuration exists,
# `__default__` is used, and if __default__ does not exist, k8s cluster default values␣
↪are used
node(func=train_model, inputs=["X_train", "y_train"], outputs="regressor", name='train_
↪model', tags=['resources:huge_machines'])
# evaluate_model node is assigned resources `__default__` configuration and if it does␣
↪not exist,
# k8s cluster default values are used
```

(continues on next page)

(continued from previous page)

```
node(func=evaluate_model, inputs=["X_train", "y_train"], outputs="regressor", name=
↪'evaluate_model')
```

2.2.2 Dynamic configuration support

kedro-airflow-k8s contains hook that enables TemplatedConfigLoader. It allows passing environment variables to configuration files. It reads all environment variables following KEDRO_CONFIG_ 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

Although the plugin does not perform deployment, it's recommended to have access to Airflow DAG directory in order to test run the generated DAG.

3.1.2 Install the toy project with Kedro Airflow K8S 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-airflow-k8s` plugin and kedro-docker.

```
$ pip install 'kedro<0.17' kedro-airflow-k8s 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]: Airflow K8S 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.
 [airflow-k8s-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.
 [airflow_k8s_plugin_demo]:

Change directory to the project generated in ${CWD}/airflow-k8s-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-airflow-k8s plugin is activated:

```
$ cd airflow-k8s-plugin-demo/
$ kedro install
(...)
Requirements installed!
$ kedro airflow-k8s --help
```console
$ kedro airflow-k8s

Usage: kedro airflow-k8s [OPTIONS] COMMAND [ARGS]...

Options:
-e, --env TEXT  Environment to use.
-p, --pipeline TEXT  Pipeline name to pick.
-h, --help      Show this message and exit.

Commands:
  compile          Create an Airflow DAG for a project
  init             Initializes configuration for the plugin
  list-pipelines   List pipelines generated by this plugin
  run-once         Uploads pipeline to Airflow and runs once
  schedule         Uploads pipeline to Airflow with given schedule
  ui               Open Apache Airflow UI in new browser tab
  upload-pipeline  Uploads pipeline to Airflow DAG location
```
```

3.1.3 Build the docker image to be used on Airflow K8S 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 Airflow (and not on local environment) 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 airflow_k8s_plugin_demo:latest remote.repo.url.com/airflow_k8s_plugin_
↪demo:latest
docker push remote.repo.url.com/airflow_k8s_plugin_demo:latest
```

3.1.4 Setup GIT repository

Plugin requires project to be under git repository. Perform repository initialization and commit project files

3.1.5 Compile DAG

Plugin requires configuration to be present. It's best to use:

```
kedor airflow-k8s init --with-github-actions --output ${AIRFLOW_DAG_FOLDER} https://
↪airflow.url
```

This command creates configuration file in `conf/pipelines/airflow-k8s.yaml` with some custom values and reference to Airflow passed in arguments. It also creates some default github actions.

> When using this command, pay attention that the configuration expects `commit_id` and `google_project_id` to be present. Set them up by setting environment variable `KEDRO_CONFIG_COMMIT_ID` and `KEDRO_CONFIG_GOOGLE_PROJECT_ID`.

Also mlflow configuration has to be set up (if required by the project) as described in mlflow section.

Having configuration ready, type:

```
kedro airflow-k8s -e pipelines compile
```

This command compiles pipeline and generates DAG in `dag/airflow_k8s_plugin_demo.py`. This file should be copied manually into Airflow DAG directory, that Airflow periodically scans. After it appears in airflow console, it is ready to be triggered.

As an alternative, one cas use the following:

```
kedro airflow-k8s -e pipelines upload-pipeline -o ${AIRFLOW_DAG_HOME}
```

in order to get DAG copied directly to Airflow DAG folder. Google Cloud Storage locations are also support with `gcs://` or `gs://` prefix in the parameter (this requires plugin to be installed with `pip install kedro-airflow-k8s[gcp]`).

In order to use AWS S3 as storage, prefix output with `s3://` (this requires plugin to be installed with `pip install kedro-airflow-k8s[aws]`).

> It's optional to indicate which pipeline to pick, with `-p` option. By default, pipeline name `__default__` is used. Option `-p` can refer to other pipeline by name it's registered inside kedro hook.

3.1.6 Diagnose execution

Every kedro node is transformed into Airflow DAG task. DAG also contains other, supporting tasks, which are handled by a set of custom operators. In order to diagnose DAG run, every task is logging information with standar python logging library. The outcome is available in Airflow `Log` tab.

3.2 GCP AI Platform support

Google Cloud's AI Platform offers couple services that simplify Machine Learning tasks.

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.

3.3 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` 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/local/mlflow.yml` and adjusting `mlflow_tracking_uri` key. Then, build the image:

```
$ kedro docker build
```

And re-push the image to the remote registry.

> If `kedro-mlflow` is not installed as dependency and configuration is not in place (missing `kedro mlflow init`), the MLflow experiment will not be initialized and available for pipeline tasks in Apache Airflow DAG.

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search