

Kubeflow Tutorial

Objective : run a **XGBoost model** project on Kubeflow. Define and record all the steps necessary (manual, automatic and semi-automatic) to transform a GitHub repository to use Kubeflow.

Get Started on Ubuntu

Create a clean Python 3 environment with a name of your choosing. This example uses Python 3.7 and an environment name of `mlpipeline`.

```
conda create --name mlpipeline python=3.7
conda activate mlpipeline
```

1. pip install Kubeflow after activate the conda mlpipeline

```
(base) wizek@ubuntu:~/Desktop$ conda activate mlpipeline
(mlpipeline) wizek@ubuntu:~/Desktop$ pip3 install kfp --upgrade
```

2. Install Anaconda3-2019

`cd /tmp`

`curl -O`

https://repo.anaconda.com/archive/Anaconda3-2019.03-Linux-x86_64.sh

This [tutorial](#) shows step by step how to install conda on Ubuntu.

3. Install docker engine

after setup docker repository you can install docker with this command

```
$ sudo apt-get install docker-ce docker-ce-cli containerd.io
```

This [tutorial](#) shows step by step how to install docker engine on Ubuntu.

4. Clone mlflow projects examples

```
$ git clone https://github.com/kubeflow/pipelines.git
```

5. Create a Docker container image that packages your program

you have to execute the `build_image` script on the git repository

```
(base) wizek@ubuntu:~/Documents/MLops Project/KubeFlow/pipelines/components/deprecated
/dataproc/train$ sudo ./build_image.sh
```

The creation process will look like this after few minutes

```
wizkod@ubuntu: ~/Documents/MLOps Project/KubeFlow/pipelines/com...
WARNING: You appear to be running this script as root. This may cause
the installation to be inaccessible to users other than the root user.

Your current Cloud SDK version is: 319.0.0
Installing components from version: 319.0.0

-----+
|                                     |
|               These components will be installed.               |
|                                     |
|-----+-----+-----+-----+
| Name                                     | Version | Size |
|-----+-----+-----+-----+
| BigQuery Command Line Tool              | 2.0.62  | < 1 MiB |
| BigQuery Command Line Tool (Platform Specific) | 2.0.58  | < 1 MiB |
| Cloud SDK Core Libraries (Platform Specific) | 2020.07.10 | < 1 MiB |
| Cloud Storage Command Line Tool         | 4.55    | 3.5 MiB |
| Cloud Storage Command Line Tool (Platform Specific) | 4.51    | < 1 MiB |
| Default set of gcloud commands          |         |         |
| anthoscli                               | 0.2.7   |         |
| anthoscli                               | 0.2.7   | 51.7 MiB |
| gcloud cli dependencies                 | 2020.06.12 | < 1 MiB |
|-----+-----+-----+-----+

For the latest full release notes, please visit:
https://cloud.google.com/sdk/release\_notes

#=====
#- Creating update staging area                      =#
#=====
#- Installing: BigQuery Command Line Tool           =#
#=====
```

At the end :

```
All components are up to date.
Updated property [component_manager/disable_update_check].
Removing intermediate container 64cb49f07021
---> 0638ff93623f
Step 7/8 : ADD build /ml
---> d6a72f08d67e
Step 8/8 : ENV PATH $PATH:/tools/node/bin:/tools/google-cloud-sdk/bin
---> Running in 2d676f3131ea
Removing intermediate container 2d676f3131ea
---> cfe23ebdbd63
Successfully built cfe23ebdbd63
Successfully tagged ml-pipeline-dataproc-base:latest
/home/wizkod/Documents/MLOps Project/KubeFlow/pipelines/components/deprecated/dataproc/train
.././../build_image.sh: line 39: gcloud: command not found
(base) wizkod@ubuntu:~/Documents/MLOps Project/KubeFlow/pipelines/components/deprecated/d
/dataproc/train$
```

That means our docker image has been successfully installed and we will be able to run it.

6. Define a Python function to wrap your component to describe the interactions with the Docker container image that contains your pipeline component.

In our project, the following python function describes a component that trains an Xboost model

```
Open  ▾  +  xgboost_training_cm.py  Save  ≡  -  □  ×
~/Documents/MLOps Project/KubeFlow/pipelines/samples/core/xgboost_training_cm

136     })
137
138
139 def dataproc_train_op(
140     project,
141     region,
142     cluster_name,
143     train_data,
144     eval_data,
145     target,
146     analysis,
147     workers,
148     rounds,
149     output,
150     is_classification=True
151 ):
152
153     if is_classification:
154         config='gs://ml-pipeline/sample-data/xgboost-config/trainconfcla.json'
155     else:
156         config='gs://ml-pipeline/sample-data/xgboost-config/trainconfreg.json'
157
158     return dataproc_submit_spark_op(
159         project_id=project,
160         region=region,
161         cluster_name=cluster_name,
162         main_class=_TRAINER_MAIN_CLS,
163         spark_job=json.dumps({'jarFileUri': [_XGBOOST_PKG]}),
164         args=json.dumps([
165             str(config),
166             str(rounds),
167             str(workers),
168             str(analysis),
169             str(target),
170             str(train_data),
171             str(eval_data),
172             str(output)
173         ])
174     )
```

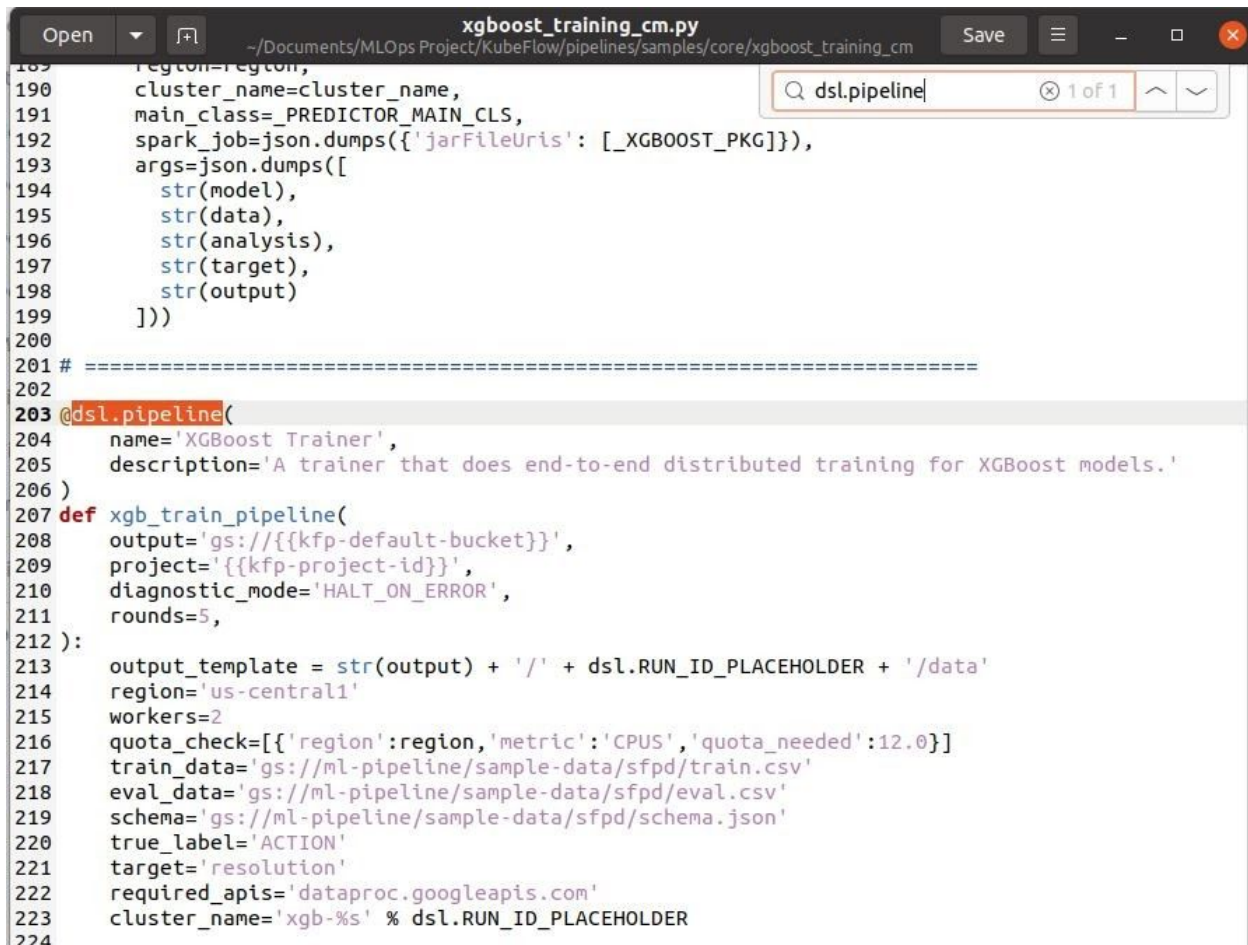
- Each component must inherit from `dsl.ContainerOp`.
- Values in the arguments list that's used by the `dsl.ContainerOp` constructor above must be either Python scalar types (such as `str` and `int`) or `dsl.PipelineParam` types. Each `dsl.PipelineParam` represents a parameter whose value is usually only known at run time. The value is either provided by the user at pipeline run time or received as an output from an upstream component.
- `file_outputs` is a mapping between labels and local file paths. In the above example, the content of `/output.txt` contains the string output of the component. To reference the output in code:

```
op = dataproc_train_op(...)
```

```
op.outputs['label']
```

7. Describe each pipeline as a Python function

See the full code in the [XGBoost Spark pipeline sample](#).

A screenshot of a code editor window titled 'xgboost_training_cm.py'. The editor shows Python code for a Kubeflow pipeline. The code includes imports, a function definition for 'xgb_train_pipeline', and a decorator '@dsl.pipeline'. The function 'xgb_train_pipeline' takes several arguments: 'output', 'project', 'diagnostic_mode', 'rounds', 'output_template', 'region', 'workers', 'quota_check', 'train_data', 'eval_data', 'schema', 'true_label', 'target', 'required_apis', and 'cluster_name'. The code is well-commented and includes a search bar in the top right corner with the text 'dsl.pipeline'.

```
189 region=region,
190 cluster_name=cluster_name,
191 main_class=_PREDICTOR_MAIN_CLS,
192 spark_job=json.dumps({'jarFileUri': [_XGBOOST_PKG]}),
193 args=json.dumps([
194     str(model),
195     str(data),
196     str(analysis),
197     str(target),
198     str(output)
199 ]))
200
201 # =====
202
203 @dsl.pipeline(
204     name='XGBoost Trainer',
205     description='A trainer that does end-to-end distributed training for XGBoost models.'
206 )
207 def xgb_train_pipeline(
208     output='gs://{{kfp-default-bucket}}',
209     project='{{kfp-project-id}}',
210     diagnostic_mode='HALT_ON_ERROR',
211     rounds=5,
212 ):
213     output_template = str(output) + '/' + dsl.RUN_ID_PLACEHOLDER + '/data'
214     region='us-central1'
215     workers=2
216     quota_check=[{'region':region, 'metric':'CPUS', 'quota_needed':12.0}]
217     train_data='gs://ml-pipeline/sample-data/sfpd/train.csv'
218     eval_data='gs://ml-pipeline/sample-data/sfpd/eval.csv'
219     schema='gs://ml-pipeline/sample-data/sfpd/schema.json'
220     true_label='ACTION'
221     target='resolution'
222     required_apis='dataproc.googleapis.com'
223     cluster_name='xgb-%s' % dsl.RUN_ID_PLACEHOLDER
224
```

- **@dsl.pipeline** is a required decoration including the **name** and **description** properties.
- Input arguments show up as pipeline parameters on the Kubeflow Pipelines UI. As a Python rule, positional arguments appear first, followed by keyword arguments.
- Each function argument is of type `dsl.PipelineParam`. The default values should all be of that type. The default values show up in the Kubeflow Pipelines UI but the user can override them.

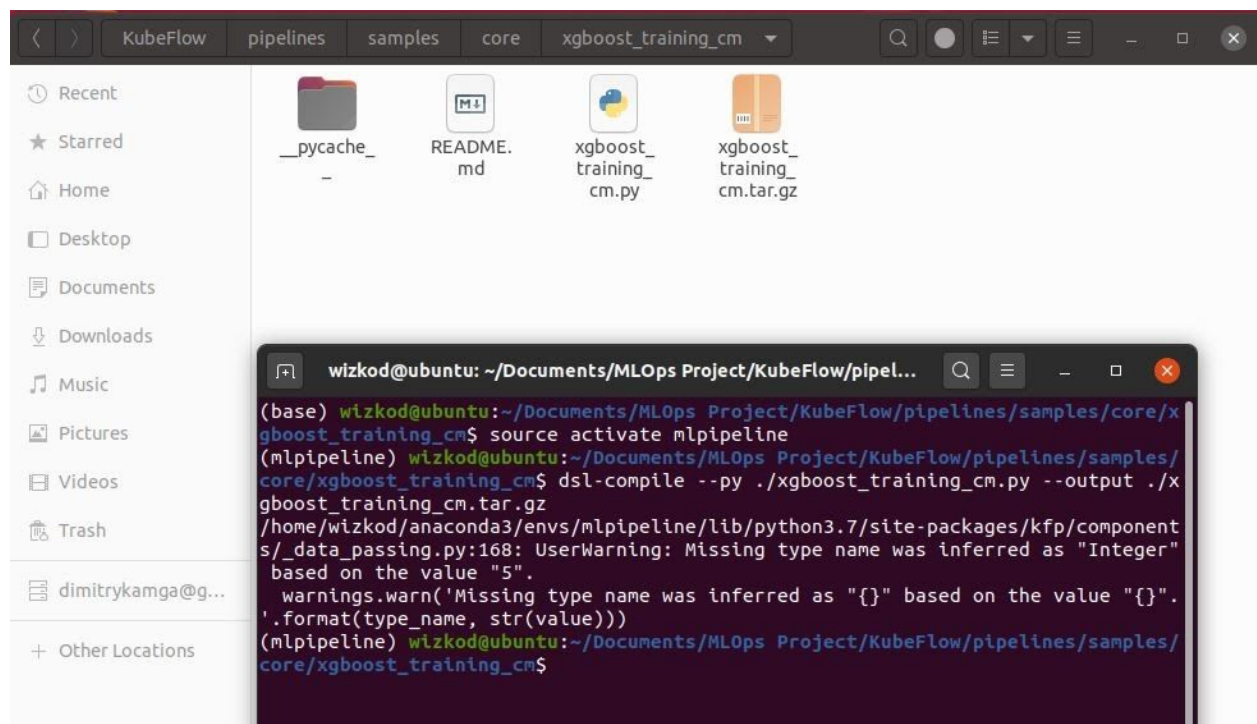
8. Compile the pipeline

After defining the pipeline in Python as described above, you must compile the pipeline to an intermediate representation before you can submit it to the Kubeflow Pipelines service. The intermediate representation is a workflow specification in the form of a YAML file compressed into a `.tar.gz` file.

-- activate the `mlpipeline` variable on your folder with `source activate mlpipeline`

-- Use the `dsl-compile --py ./xgboost_training_cm.py --output ./xgboost_training_cm.tar.gz` command to compile your pipeline:

and then you will get the `.tar.gz` file



9. Deploy the pipeline

Upload the generated `.tar.gz` file through the Kubeflow Pipelines UI.

Before we need to install some google sdk.

`sudo snap install google-cloud-sdk -- classic`

```
(base) wizkod@ubuntu:~/Desktop$ sudo snap install google-cloud-sdk --classic
google-cloud-sdk 319.0.0 from Cloud SDK (google-cloud-sdk✓) installed
```

- now we have to install kubectl with this simple [tutorial](#) or just enter

```
(base) wizkod@ubuntu:~/Desktop$ snap install kubectl --classic
kubectl 1.19.4 from Canonical✓ installed
```

I choose AWS to deploy it but , You can alternatively deploy it on Google Cloud Platform with following this [tutorial](#) but please notice that :

Due to kubeflow/pipelines#1700, the container builder in Kubeflow Pipelines currently prepares credentials for Google Cloud Platform (GCP) only. As a result, the container builder supports only Google Container Registry. However, you can store the container images on other registries, provided you set up the credentials correctly to fetch the image.

Unfortunately my computer can't launch kubeflow locally because it's so heavy and requires important resources, we talk about **16RAM 50G space + 4CPUs on VM**, so here I got two another [tutorial](#) or check the [video tutorial](#) to launch it with microk8s (Ubuntu).

```
multipass 1.5.0 from Canonical✓ installed
(base) wizkod@ubuntu:~/Desktop$ multipass shell kubeflow
shell failed: instance "kubeflow" does not exist
(base) wizkod@ubuntu:~/Desktop$ multipass launch --name kubeflow --mem 16G --disk 50G --cpus 4
launch failed: CPU does not support KVM extensions.
(base) wizkod@ubuntu:~/Desktop$
```

I have a AWS account so I create an instance who can store the deployment test there it is:

The screenshot shows the AWS Management Console interface. On the left, there's a sidebar with navigation options: 'New EC2 Experience', 'Tableau de bord EC2', 'Événements', 'Balises', 'Limites', 'Instances', and 'Instances New'. The main area is titled 'Instances (1/1) Informations'. It contains a search bar 'Filtrer les instances' and a table with one instance. The table has columns: 'Name', 'ID d'instance', 'État de l'inst...', 'Type d'inst...', 'Contrôle des s...', and 'Alarm sta'. The instance row shows a checkmark, a dash, the ID 'i-05a1440032091927b', the state 'En cours...', the type 't2.xlarge', and '2/2 vérifica...'. Above the table, there are buttons: 'Se connecter', 'État de l'instance', 'Actions', and 'Lancer des instances'.

So if anyone would like to deploy it, I have the ssh for the public domain.