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

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.

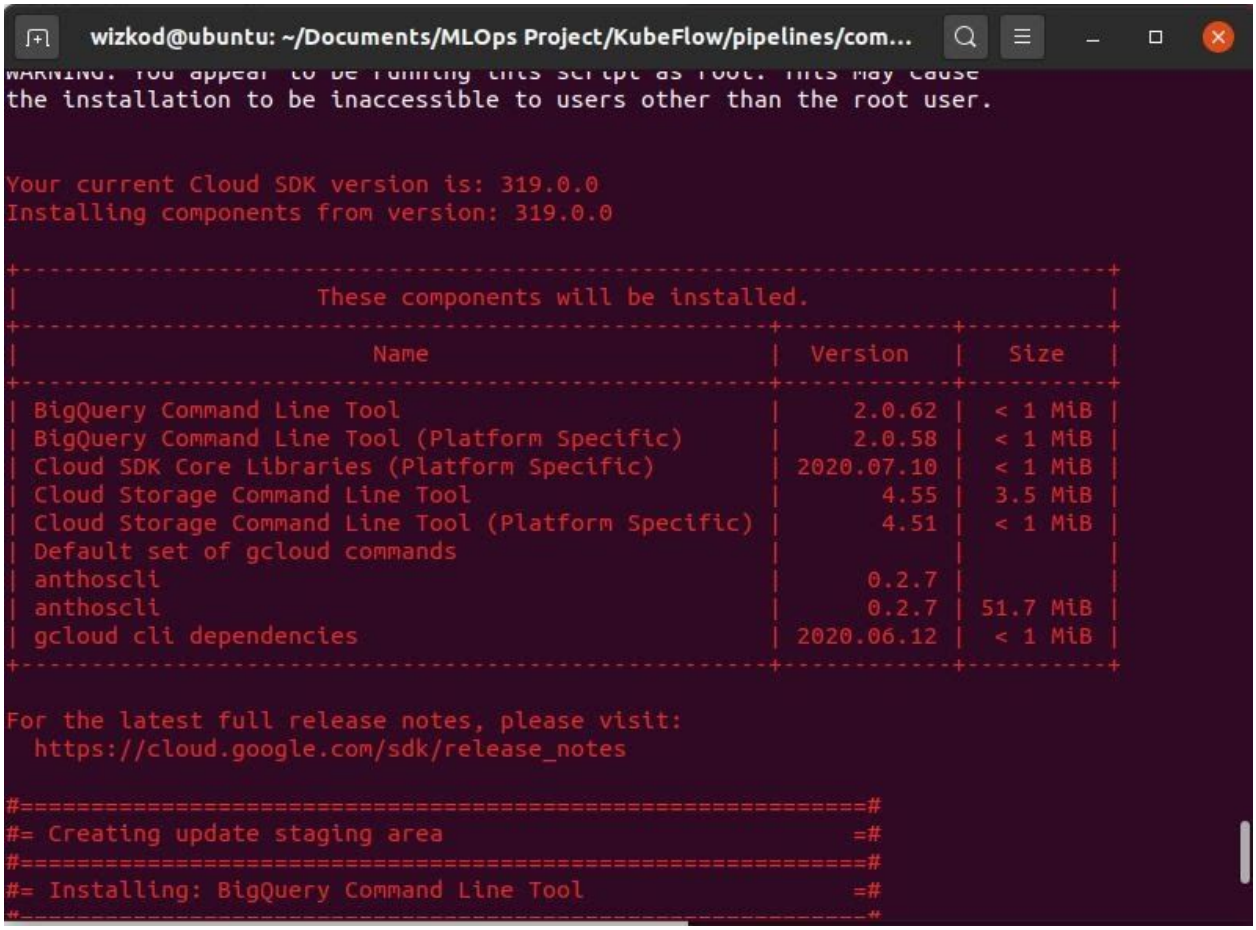
3. Install docker engine
after setup docker repository you can install docker with this command
`$ sudo apt-get install docker-ce docker-ce-cli containerd.io`

This tutorial shows step by step how to install docker engine on Ubuntu.

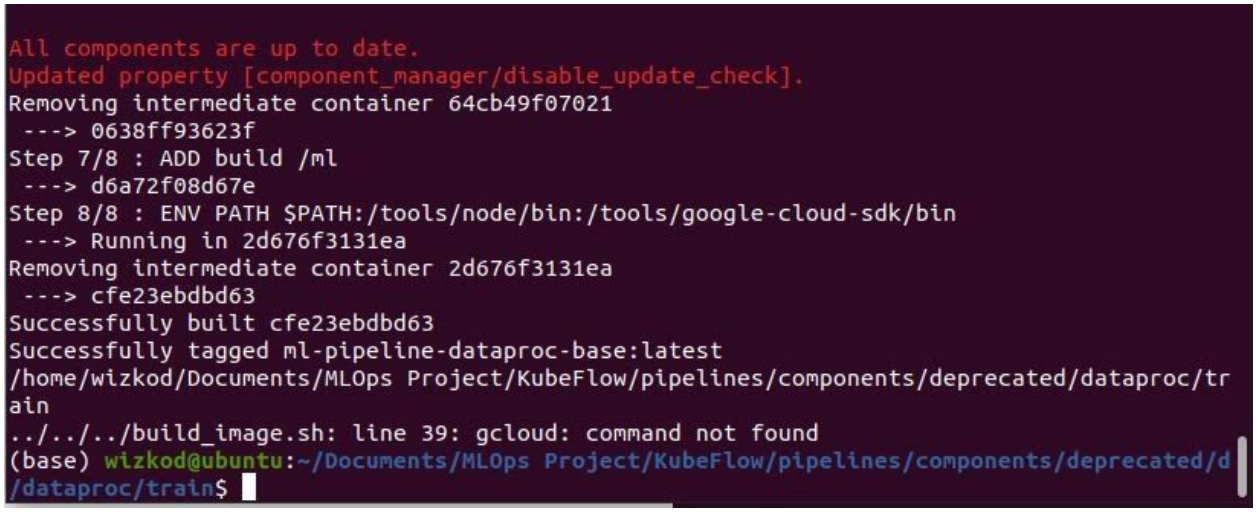
4. Clone mlflow projects examples
`$ git clone https://github.com/kubeflow/pipelines.git`

5.Create a Docker container image that packages your program
you have to execute the build_image script on the git repository

The creation process will look like this after few minutes

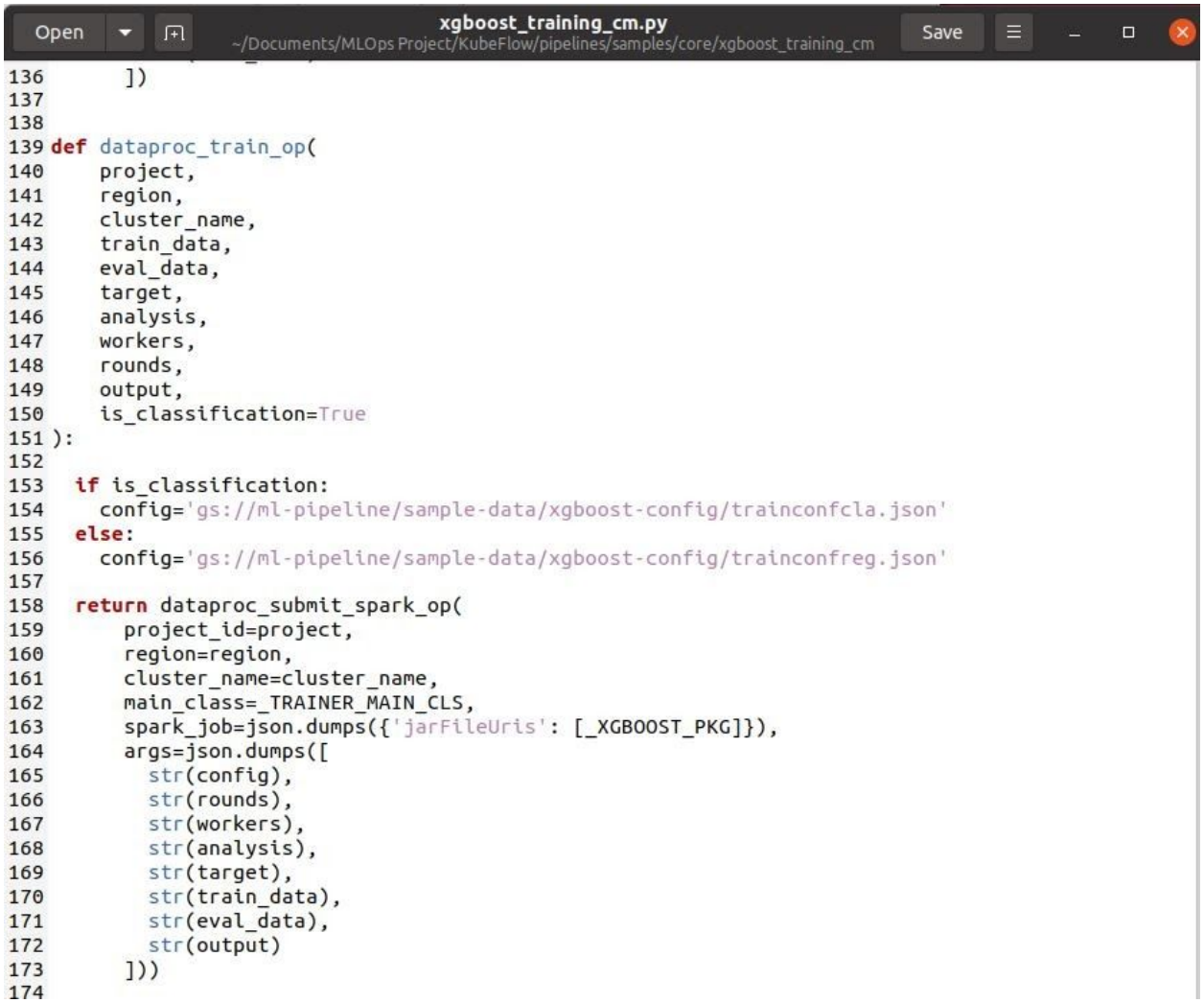


At the end :



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




```
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({'jarFileUris': [_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
```

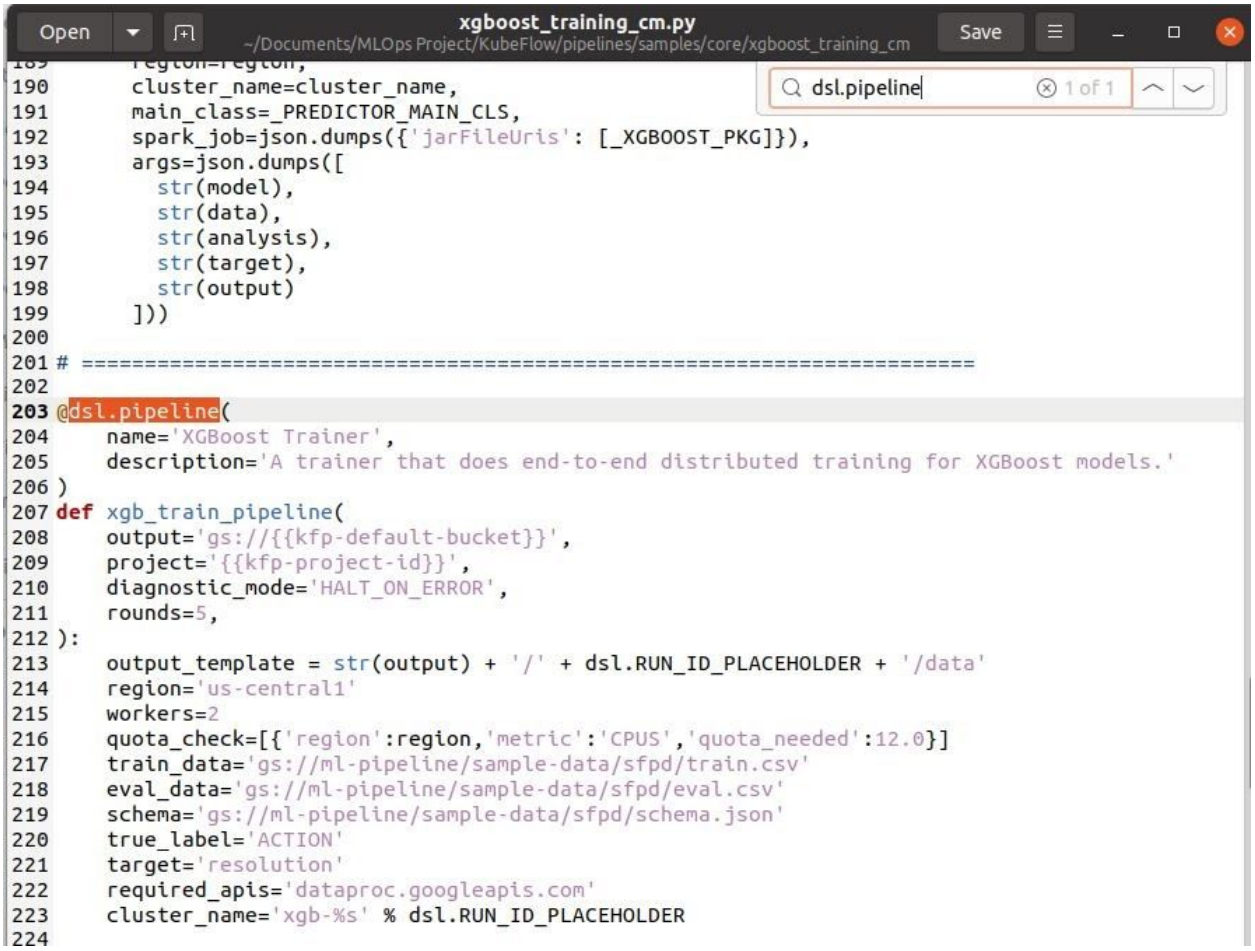
- Each component must inherit from `dsl.ContainerOp`.
- Values in the arguments list that's used by the `dsl.ContainerOp` constructor above must be either Python scalar types (such as str and int) or dsl.PipelineParam types. Each `dsl.PipelineParam` represents a parameter whose value is usually only 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.



- **@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.

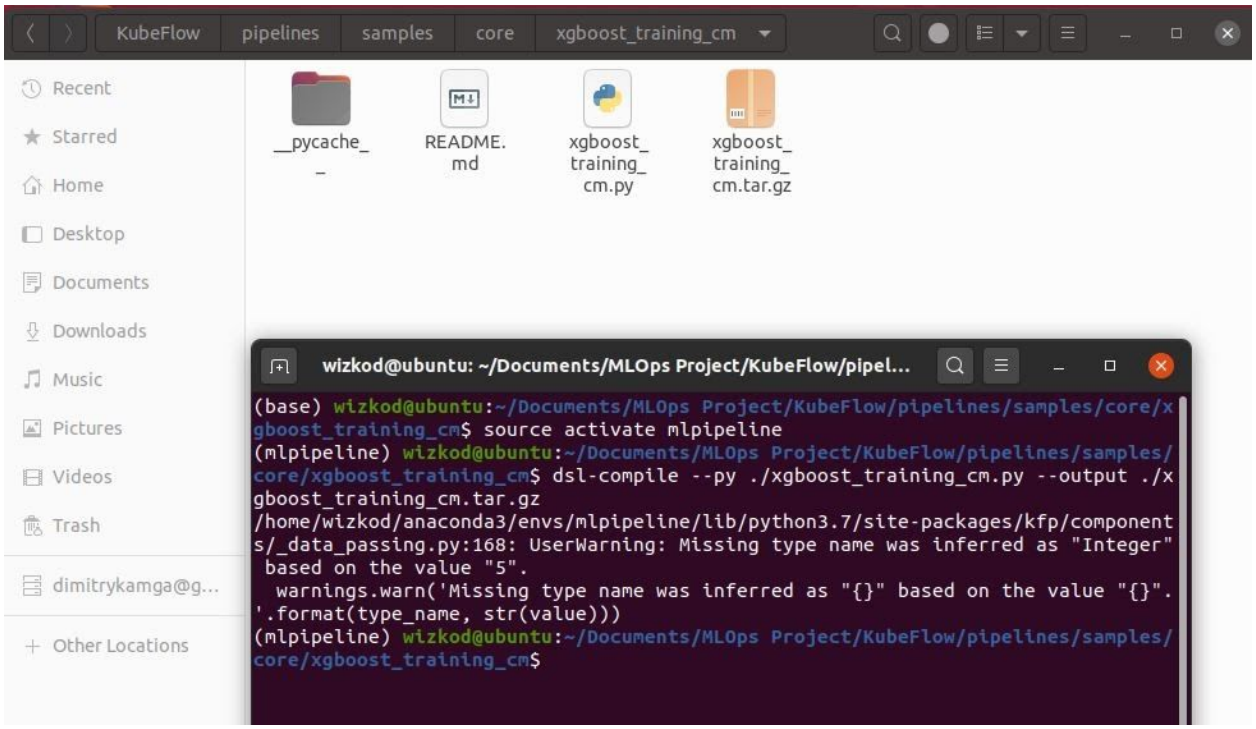
8. Compile the pipeline

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`

- now we have to install kubectl with this simple tutorial or just enter

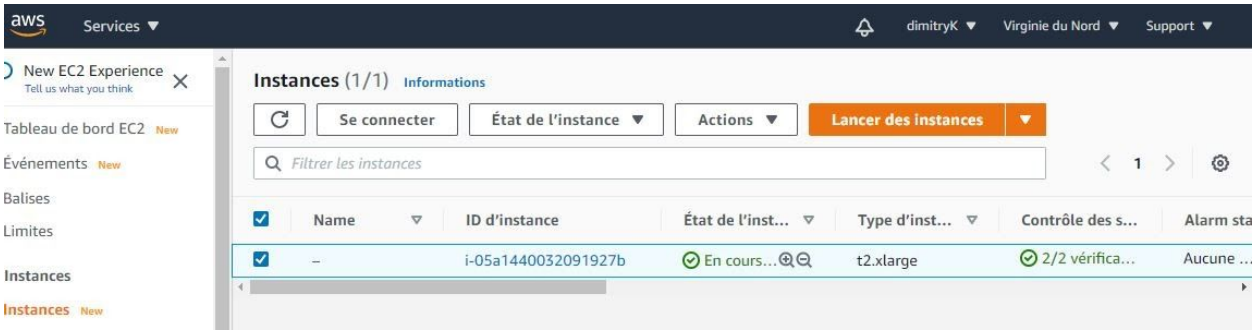
```
(base) wizkod@ubuntu:~/Desktop$ snap install kubectl --classic
kubectl 1.19.4 from Canonical✓ installed
```

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.
```

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



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