nodeflux

A Scalable Training Approach Using Kubeflow

www.nodeflux.io

I N T R O D U C I N G

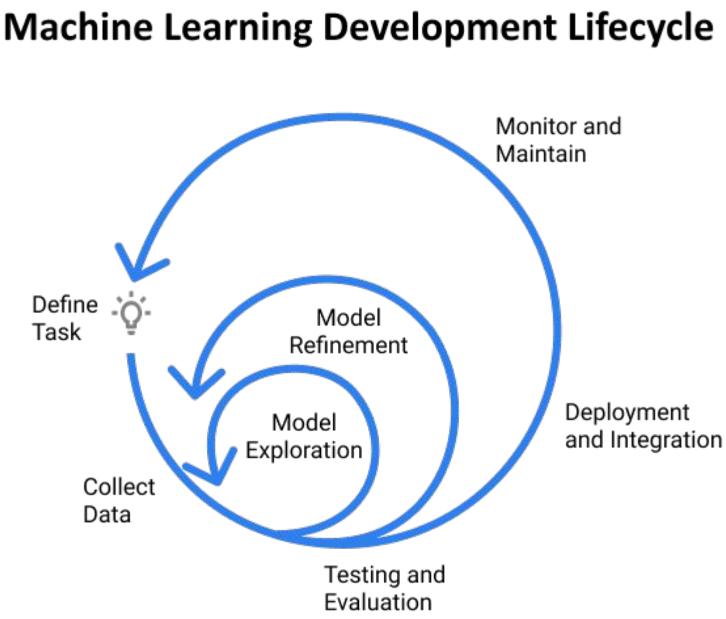
nodeflux

Alvin Prayuda Juniarta Dwiyantoro

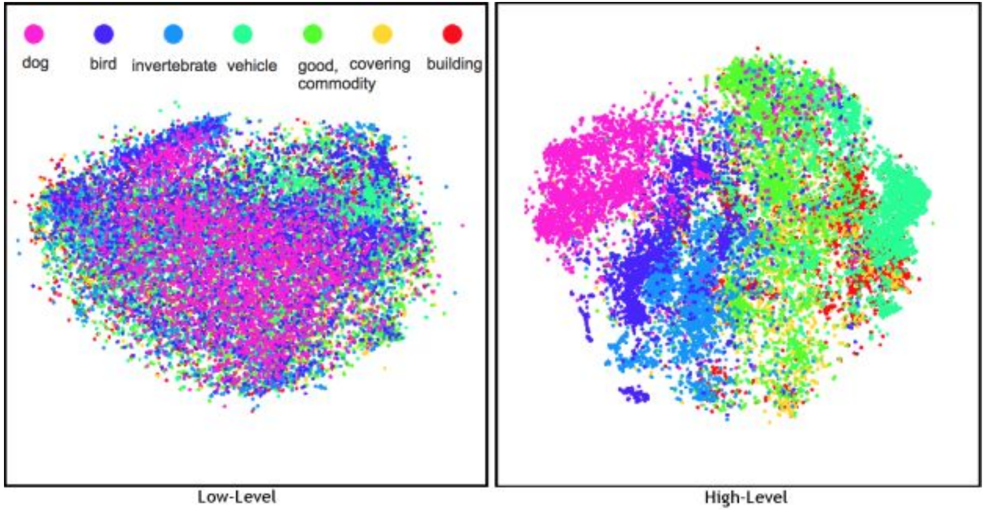
AI Research Group Lead
of Nodeflux

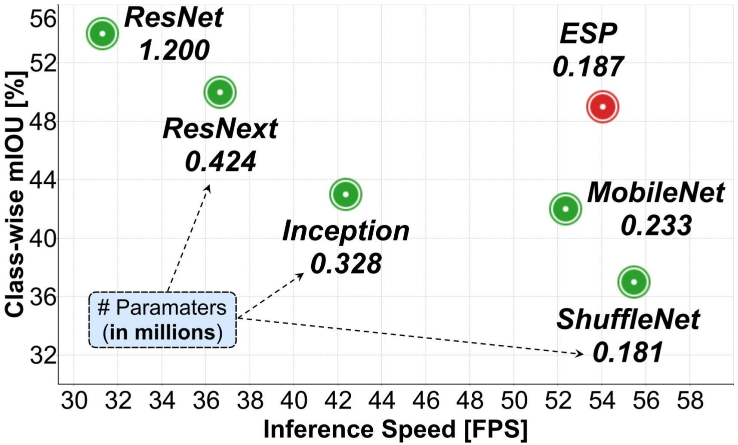
Typical Problem

- Iterative training process
- Multiple training resources
- Data migration
- Training process observation



nodeflux



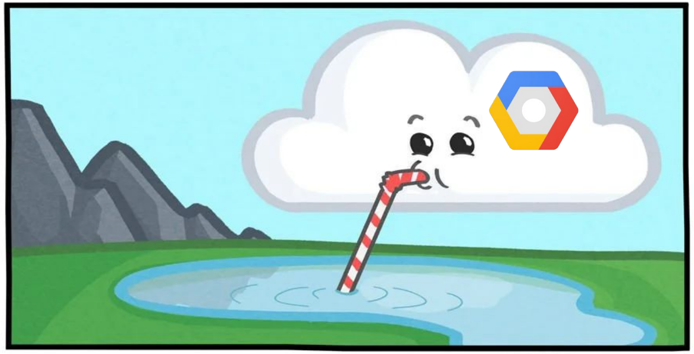


Iterative Training Process?

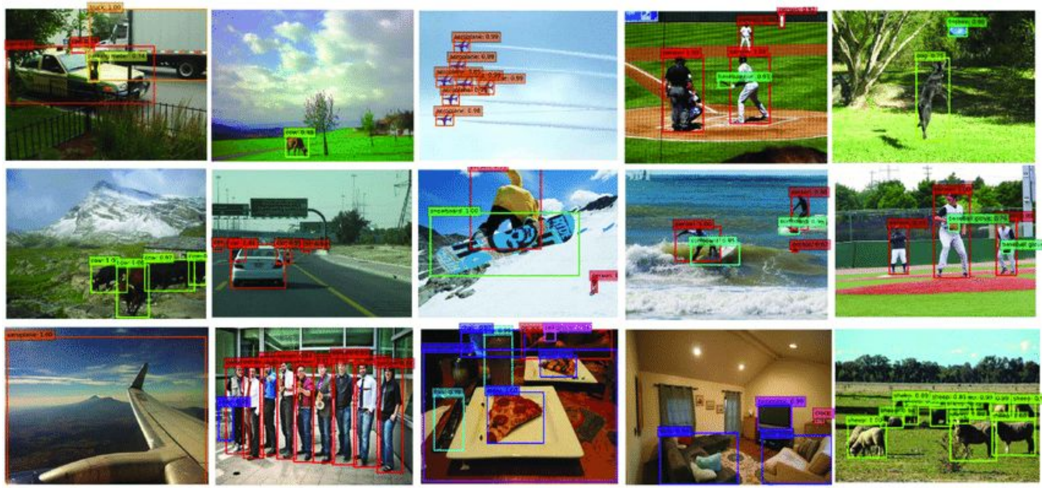
- Same problem definition (classification, regression, object detection), different hyperparameters
  - Data?
  - Method selection?
  - Number of class?
  - Data augmentation?
  - Feature Engineering?

Data Migration

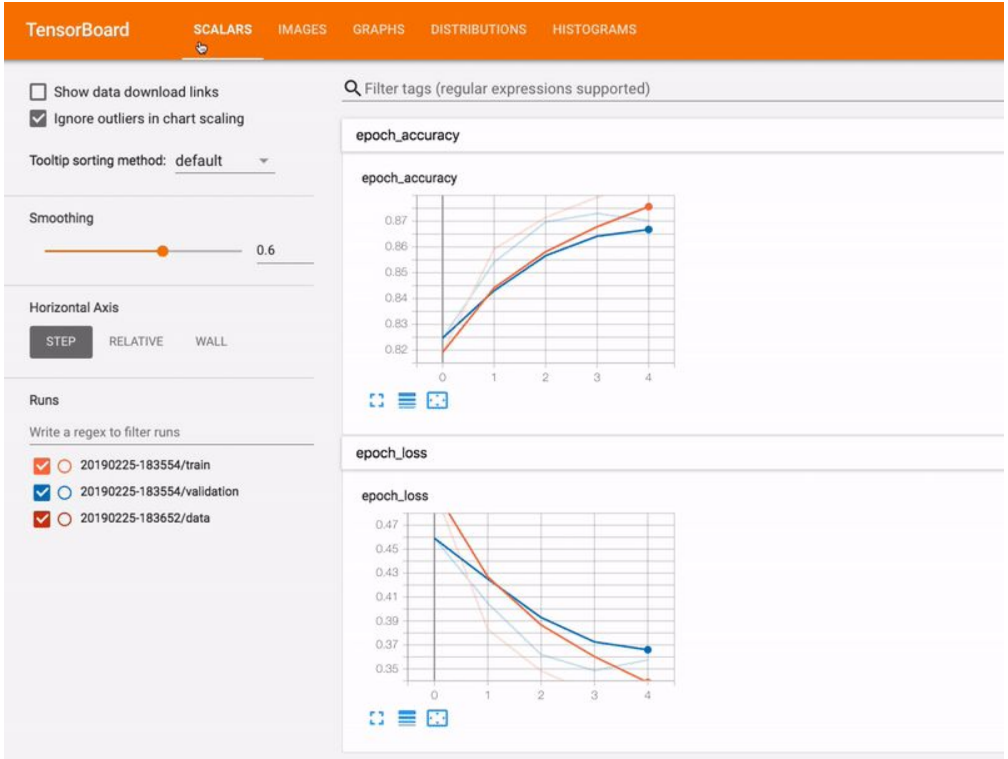
- Huge size dataset
- Prototype in laptop/pc, train in server/cloud?
    - In our case :
        - local development : Indonesia
        - available training server : US (for NVIDIA P100/V100)



MS COCO dataset : 25.2 GB



nodeflux



Training Process Observation

- Loss and metrics visualization
- Error logging

nodeflux

The Solutions

The Orchestrator

Kubeflow

The Observer

comet

The Tracker

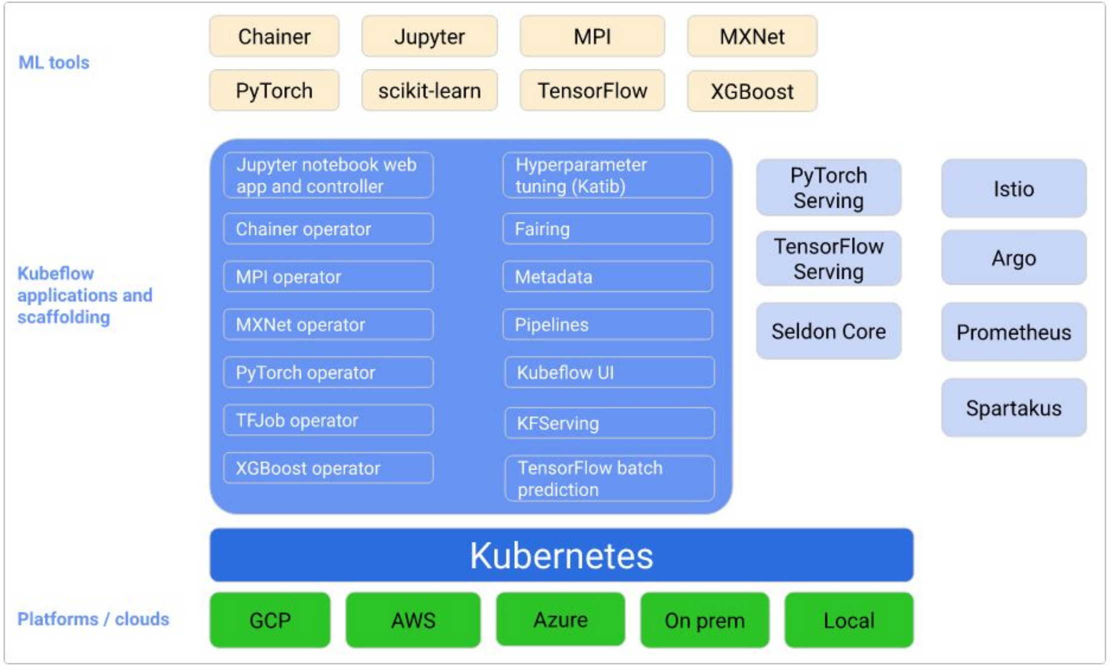
DVC

The Orchestrator

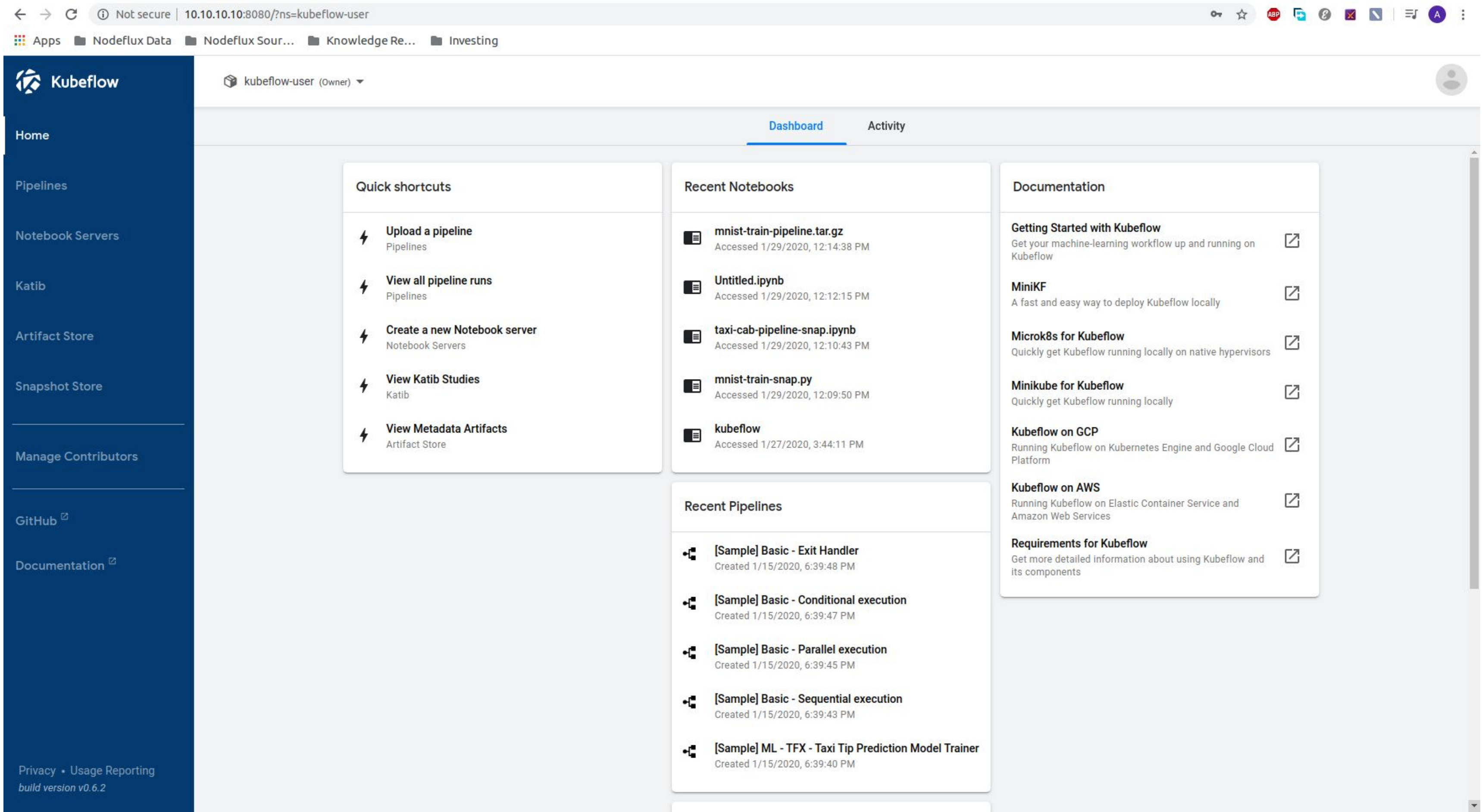
What is Kubeflow?

nodeflux

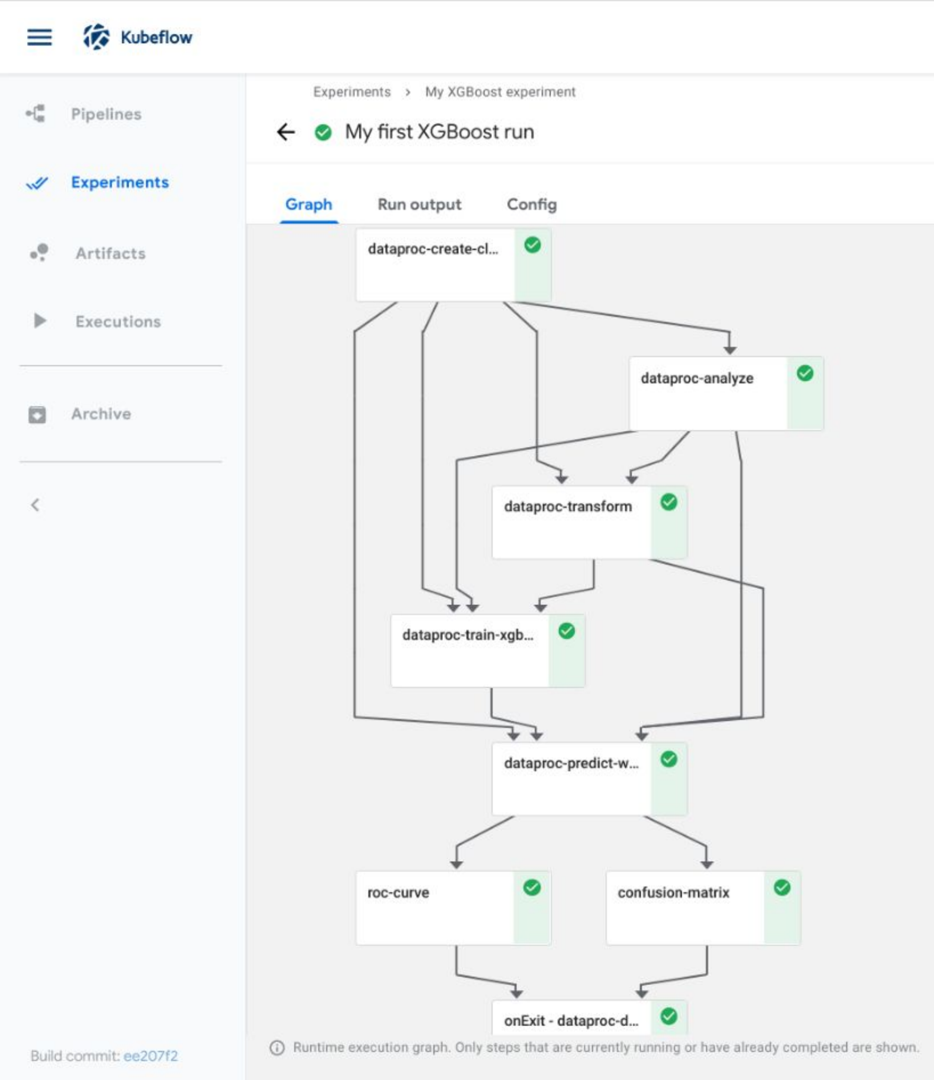
- An open source Kubernetes-native platform for developing, orchestrating, deploying, and running scalable and portable ML workloads
- It supports reproducibility and collaboration in ML workflow lifecycles in multiple or hybrid environments (local → cloud env) as long as kubernetes exist
- Helps reuse building blocks across different workflows



Kubeflow UI



Not secure | 10.10.10.10:8080/?ns=kubeflow-user
Apps
Nodeflux Data
Nodeflux Sour...
Knowledge Re...
Investing
Kubeflow
kubeflow-user (Owner)
Home
Pipelines
Notebook Servers
Katib
Artifact Store
Snapshot Store
Manage Contributors
GitHub
Documentation
Privacy • Usage Reporting
build version v0.6.2
Dashboard
Activity
Quick shortcuts
Upload a pipeline
Pipelines
View all pipeline runs
Pipelines
Create a new Notebook server
Notebook Servers
View Katib Studies
Katib
View Metadata Artifacts
Artifact Store
Recent Notebooks
mnist-train-pipeline.tar.gz
Accessed 1/29/2020, 12:14:38 PM
Untitled.ipynb
Accessed 1/29/2020, 12:12:15 PM
taxi-cab-pipeline-snap.ipynb
Accessed 1/29/2020, 12:10:43 PM
mnist-train-snap.py
Accessed 1/29/2020, 12:09:50 PM
kubeflow
Accessed 1/27/2020, 3:44:11 PM
Recent Pipelines
[Sample] Basic - Exit Handler
Created 1/15/2020, 6:39:48 PM
[Sample] Basic - Conditional execution
Created 1/15/2020, 6:39:47 PM
[Sample] Basic - Parallel execution
Created 1/15/2020, 6:39:45 PM
[Sample] Basic - Sequential execution
Created 1/15/2020, 6:39:43 PM
[Sample] ML - TFX - Taxi Tip Prediction Model Trainer
Created 1/15/2020, 6:39:40 PM
Documentation
Getting Started with Kubeflow
Get your machine-learning workflow up and running on Kubeflow
MiniKF
A fast and easy way to deploy Kubeflow locally
Microk8s for Kubeflow
Quickly get Kubeflow running locally on native hypervisors
Minikube for Kubeflow
Quickly get Kubeflow running locally
Kubeflow on GCP
Running Kubeflow on Kubernetes Engine and Google Cloud Platform
Kubeflow on AWS
Running Kubeflow on Elastic Container Service and Amazon Web Services
Requirements for Kubeflow
Get more detailed information about using Kubeflow and its components

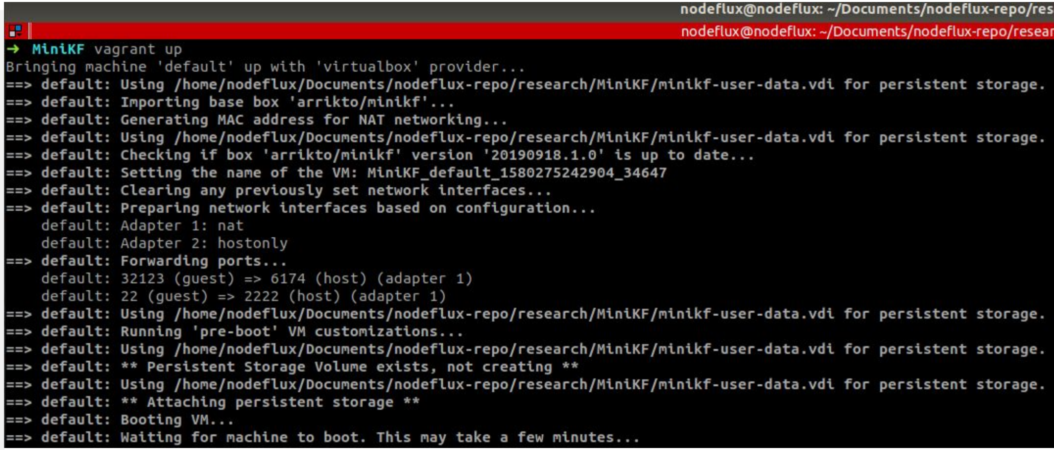


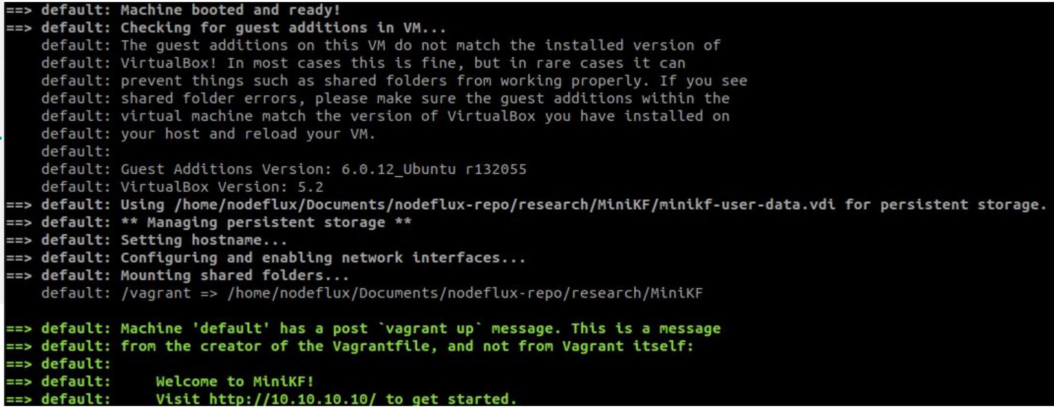
Kubeflow Pipelines

- A platform for building and deploying portable, scalable machine learning (ML) workflows based on Docker containers
  - Provides UI for experiments
  - Engine for scheduling multi-steps ML workflow
  - Interactive Jupyter to interact with pipelines
- The pipelines will consist of several components which is an executable functions inside the docker container

Local Development

- Using MiniKF to deploy Kubeflow locally (on laptop)
- MiniKF utilizing vagrant to package the Kubeflow VM
- MiniKF tutorial :
  - https://medium.com/kubeflow/an-end-to-end-ml-pipeline-on-prem-notebooks-kubeflow-pipelines-on-the-new-minikf-33b7d8e9a836





nodeflux

Local Development

- Accessing http://10.10.10.10 for MiniKF landing page
- We can access Kubeflow and Rok UI page from here
- Rok is used to snapshot the Jupyter volume in which will be used to store the data
- In our case, we only use MiniKF to provide the Kubeflow environment

Step 1 : Develop the Docker Image

app.py

home > nodeflux > Documents > nodeflux-repo > research > MiniKF > mnist_training > app.py > ...

```python
from comet_ml import Experiment
import argparse
from datetime import datetime
import tensorflow as tf
import os

os.environ["GOOGLE_APPLICATION_CREDENTIALS"] = 'AI-training-a9aad66abc4e.json'

experiment = Experiment(api_key="0kZ5vwvCG7xBToHOeptFxtyxu",
                        project_name="test-kubeflow", workspace="hyperion-rg")

parser = argparse.ArgumentParser()
parser.add_argument(
    '--model_file', type=str, required=True, help='Name of the model file.')
parser.add_argument(
    '--bucket', type=str, required=True, help='GCS bucket name.')
args = parser.parse_args()

bucket=args.bucket
model_file=args.model_file

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(512, activation=tf.nn.relu),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

print(model.summary())
```

Training Script Development

- Keep in mind, Kubeflow have several conditions
  - Accept argument parse as input
  - Each outputs must be saved as string and stored in local file inside the container

nodeflux

```
model.fit(x_train, y_train, batch_size=32, epochs=5, callbacks=callbacks,
          validation_data=(x_test, y_test))


model.save(model_file)

from tensorflow import gfile

gcs_path = bucket + "/" + model_file

if gfile.Exists(gcs_path):
    gfile.Remove(gcs_path)

gfile.Copy(model_file, gcs_path)
with open('/output.txt', 'w') as f:
  f.write(gcs_path)
```

Training Script Development

- Keep in mind, Kubeflow have several conditions
  - Accept argument parse as input
  - Each outputs must be saved as string and stored in local file inside the container

Build Docker and Push to Registry

- Create Dockerfile
- Build locally and commit to your selected registry (make sure your Kubeflow can access it later)

Name

- Dockerfile
- build_image.sh
- app.py

Dockerfile

home > nodeflux > Documents > nodeflux-repo > research > Mi

```
FROM tensorflow/tensorflow:1.15.0-py3
RUN pip install comet_ml
WORKDIR /app
COPY . /app
```

build_image.sh

home > nodeflux > Documents > nodeflux-repo > research > MiniKF > mnist_training > build_image.sh

```
docker build -t "mnist_training_kf_pipeline" .
docker tag "mnist_training_kf_pipeline" "gcr.io/ai-training-250314/mnist_training_kf_pipeline:latest"
docker push "gcr.io/ai-training-250314/mnist_training_kf_pipeline:latest"
docker image rm "mnist_training_kf_pipeline"
docker image rm "gcr.io/ai-training-250314/mnist_training_kf_pipeline:latest"
```

Step 2 : Create Reusable Kubeflow Components as a Package

nodeflux

mnist-train-snap.py

```python
import kfp.dsl as dsl
import kfp.gcp as gcp

def mnist_train_op(model_file, bucket):
    return dsl.ContainerOp(
      name="mnist_training_container",
      image='gcr.io/ai-training-250314/mnist_training_kf_pipeline:latest',
      command=['python', '/app/app.py'],
      file_outputs={'outputs': '/output.txt'},
      arguments=['--bucket', bucket, '--model_file', model_file]
    )

# Define the pipeline
@dsl.pipeline(
   name='Mnist pipeline',
   description='A toy pipeline that performs mnist model training.'
)
def mnist_container_pipeline(
    model_file: str = 'mnist_model.h5',
    bucket: str = 'gs://ai-dataset/HYP/Alvin/example_kubeflow'
):
#     mnist_train_op(model_file=model_file, bucket=bucket).apply(gcp.use_gcp_secret('user-gcp-sa'))
    mnist_train_op(model_file=model_file, bucket=bucket)

if __name__ == '__main__':
    import kfp.compiler as compiler

    compiler.Compiler().compile(mnist_container_pipeline, 'mnist-train-pipeline.tar.gz')
```

Kubeflow Components Development

- Kubeflow wrap script interaction inside a container as a python function, we call it ContainerOp
- We can compile the operations and pipeline as a single deployment file (usually ends with .tar.gz) and freely distributed to others so that they can use it

Step 3 : Execute the Deployment Package in the Kubeflow Cluster

nodeflux

Use the Pre-compiled Component

- Via scripting, possible via UI
- Simple execution, fast orchestration

MNIST-train.ipynb

Code

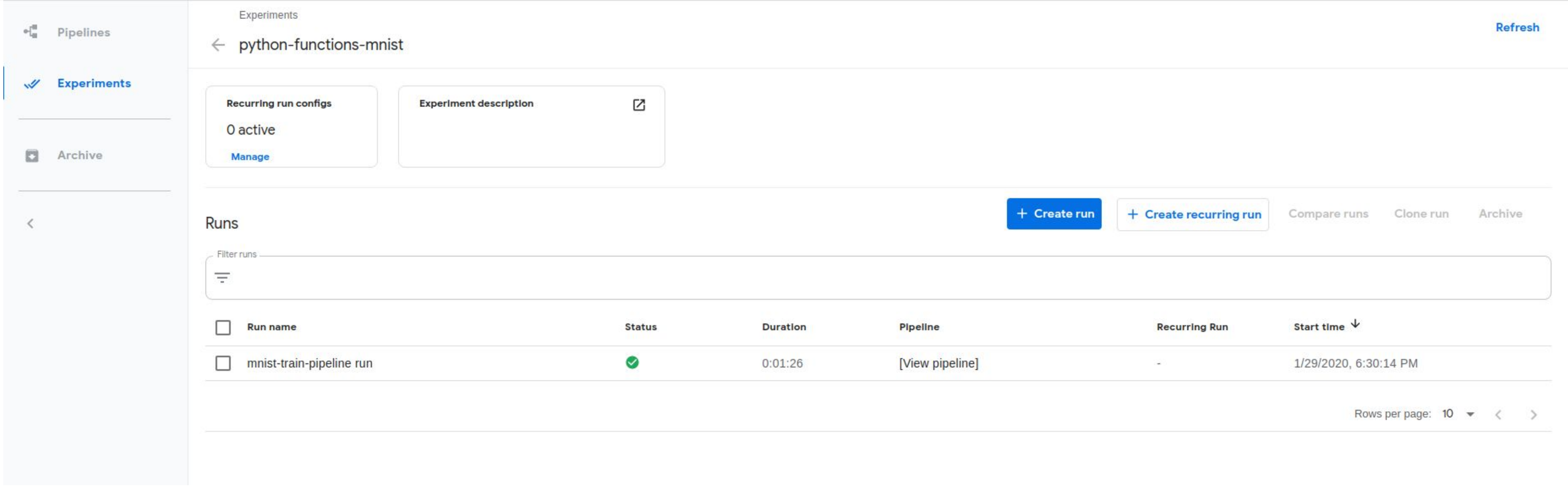
```
[2]: import kfp

client=kfp.Client()

pipeline_filename = 'mnist-train-pipeline.tar.gz'

experiment = client.create_experiment('python-functions-mnist')
run_name = 'mnist-train-pipeline run'

arguments = {"model_file":"mnist_model.h5",
             "bucket":"gs://ai-dataset/HYP/Alvin/example_kubeflow"}

run_result = client.run_pipeline(
    experiment_id=experiment.id,
    job_name=run_name,
    pipeline_package_path=pipeline_filename,
    params=arguments)
```

Experiment link here

Run link here

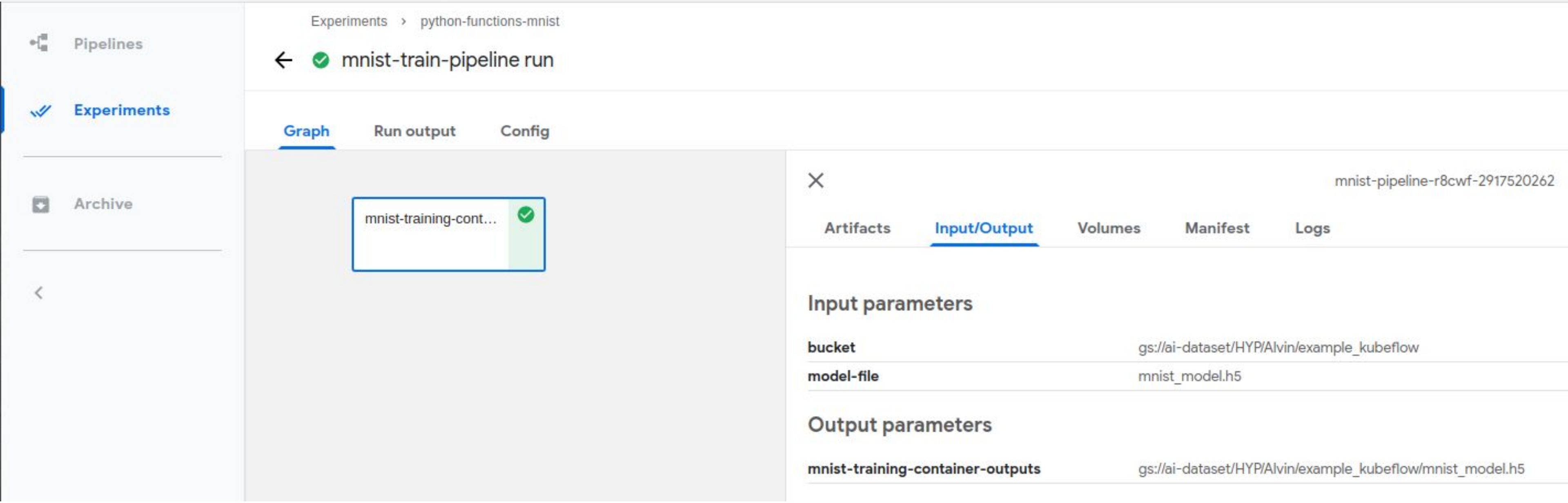
Inspect the Experiments Orchestration

- The executed workflow will appear on the experiments tab
- Running experiment will show the status, duration, etc..



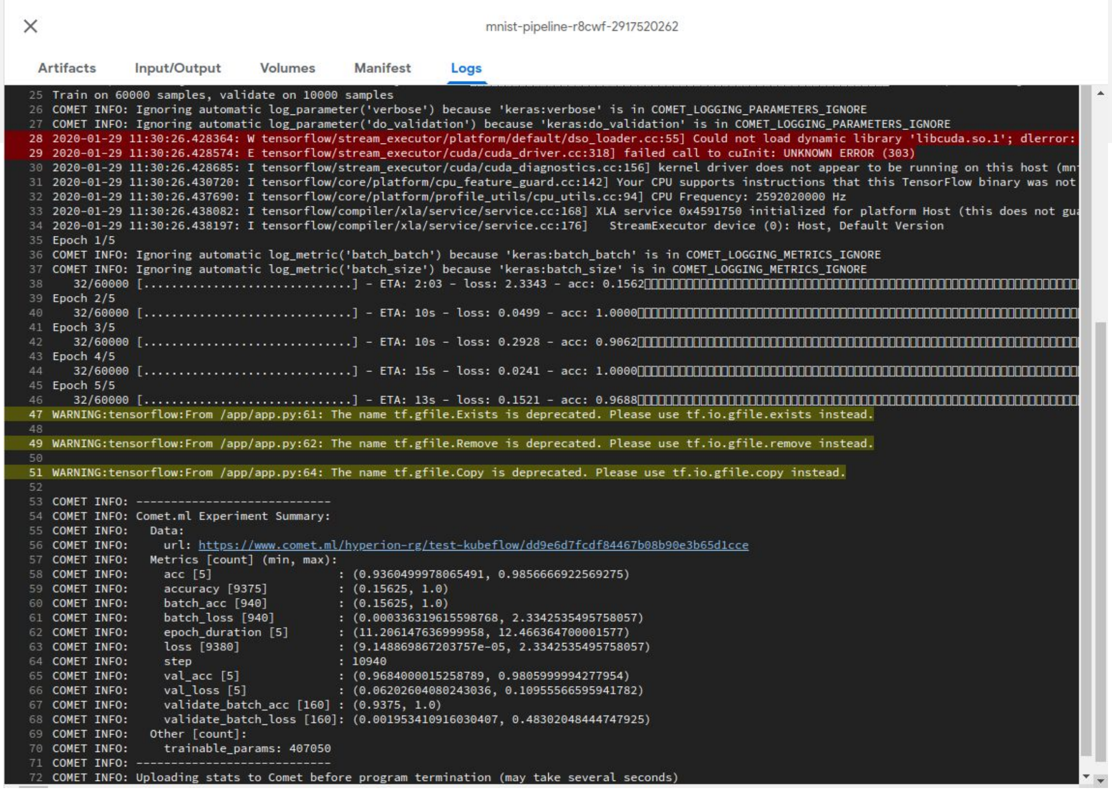
Inspect the Experiments Orchestration

- We can inspect the details of each run
  - Input parameters
  - Output value
  - Logs



Inspect the Experiments Orchestration

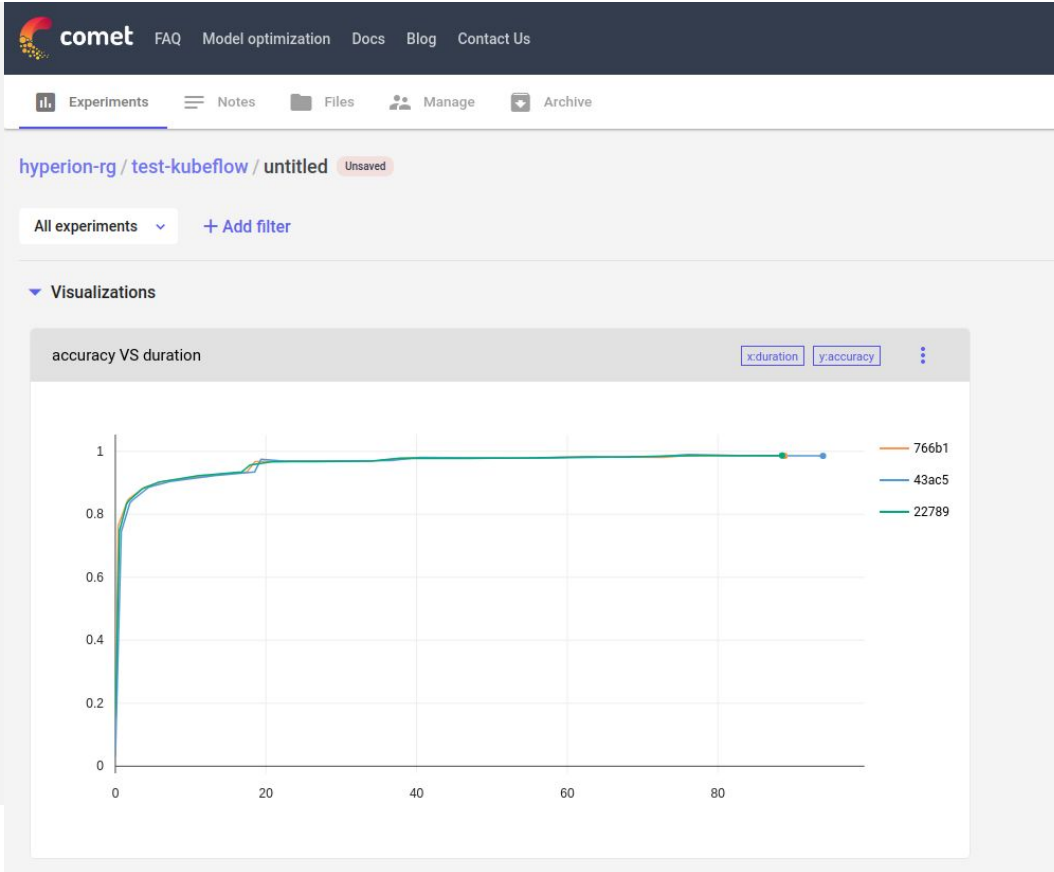
- We can inspect the details of each run
  - Input parameters
  - Output value
  - Logs



The Observer

Comet.ml?

- A freemium framework to track code, experiments, and results
- Easy to integrate with famous deep learning framework
- Enable collaboration with our teams
- Visualization across different experiments



nodeflux

home > nodeflux > Documents > nodeflux-repo > research > MiniKF > mnist_training > app.py > ...

```python
from comet_ml import Experiment
import argparse
from datetime import datetime
import tensorflow as tf
import os

os.environ["GOOGLE_APPLICATION_CREDENTIALS"] = 'AI-training-a9aad66abc4e.json'

experiment = Experiment(api_key="0kZ5vwvCG7xBToHOeptFxtyxu",
                        project_name="test-kubeflow", workspace="hyperion-rg")
```

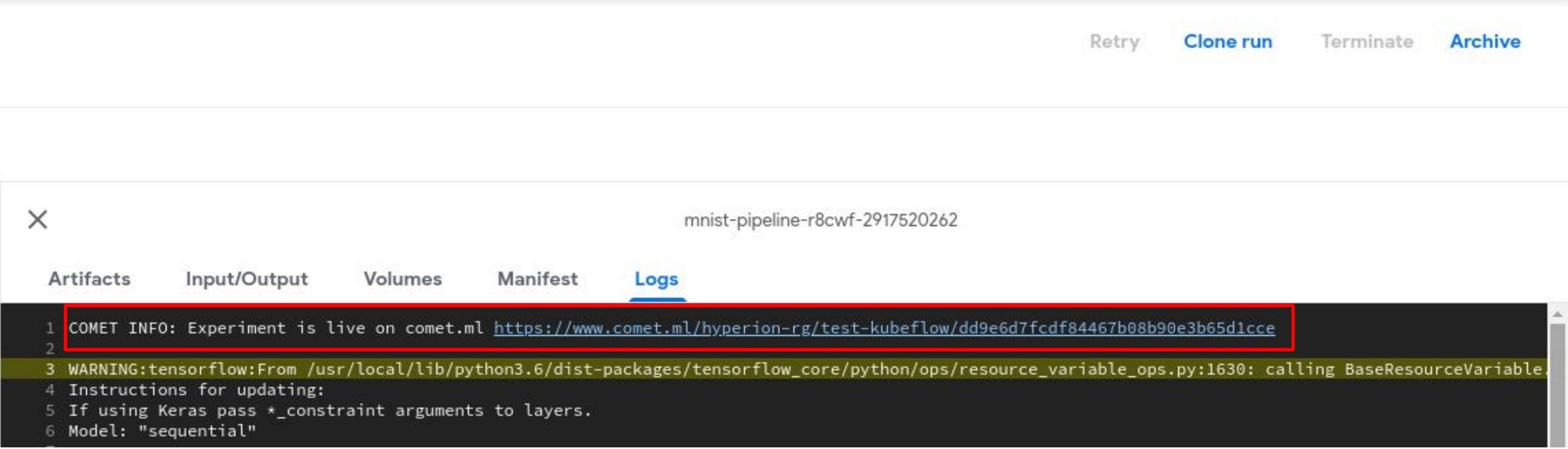
```python
class CometMLCallback(tf.keras.callbacks.Callback):
    def on_train_batch_end(self, batch, logs=None):
        experiment.log_metric("loss", logs['loss'])
        experiment.log_metric("accuracy", logs['acc'])


callbacks = [
    CometMLCallback(),
    # Interrupt training if val_loss stops improving for over 2 epochs
    tf.keras.callbacks.EarlyStopping(patience=2, monitor='val_loss'),
]

model.fit(x_train, y_train, batch_size=32, epochs=5, callbacks=callbacks,
          validation_data=(x_test, y_test))
```

Code Integration

- Kubeflow wrap script interaction inside a container as a python function, we call it ContainerOp
- We can compile the operations and pipeline as a single deployment file (usually ends with .tar.gz) and freely distributed to others so that they can use it

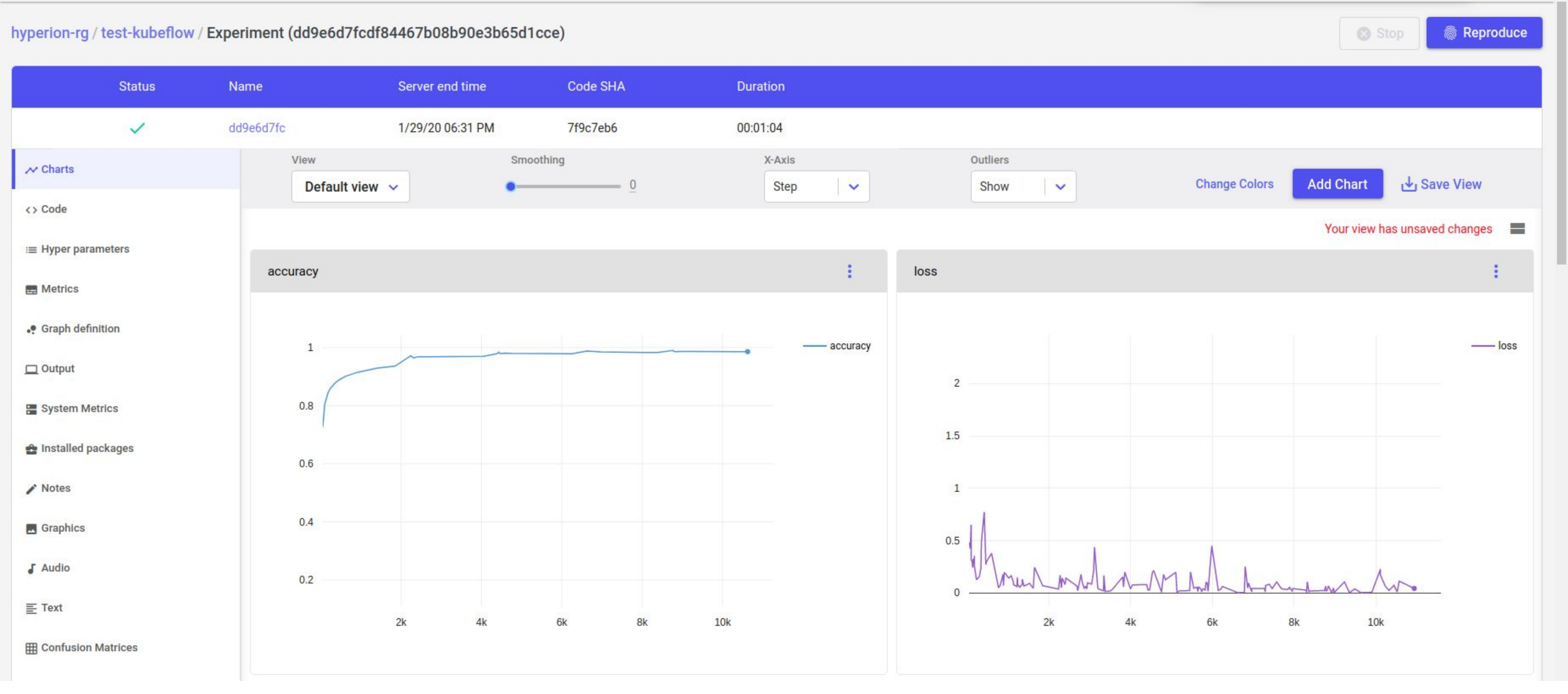
Inspect the Experiments Progress

- Experiment link will spawn on the stdout log
- Will direct you to the comet.ml experiments page



Inspect the Experiments Progress

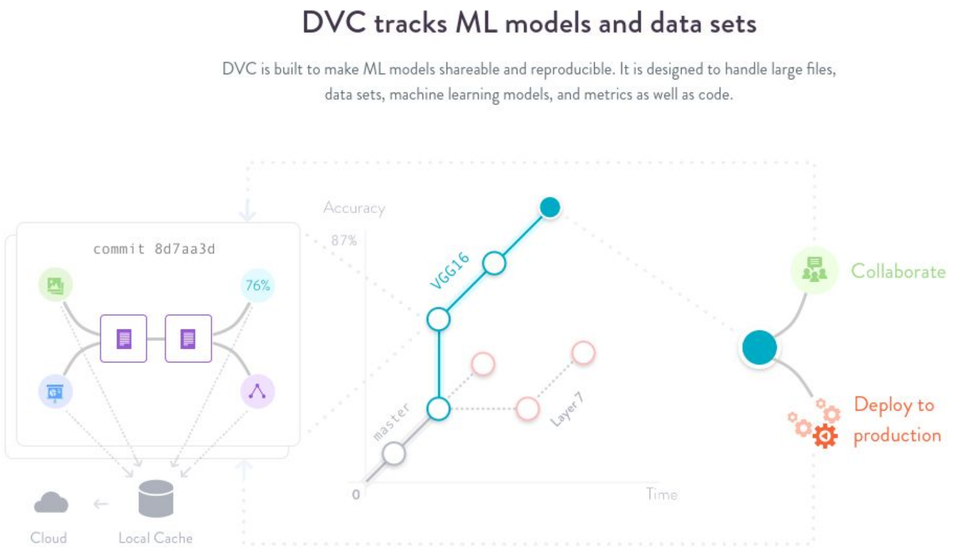
- Similarly it will record logs, parameters, but it will visualize the metric graph over the training time



The Tracker

Version Control for Data

- Download data as easy as cloning repository
- Maintain data version and changes
- Easy integration with Google Cloud Storage
- Avoid data corruption during download (maintain data hashing)



```
$ dvc add images
```

```
$ dvc remote add -d myrepo s3://mybucket
```

```
$ dvc push
```

```
$ git add images.dvc
```

```
$ git commit
```

```
$ git push origin master
```

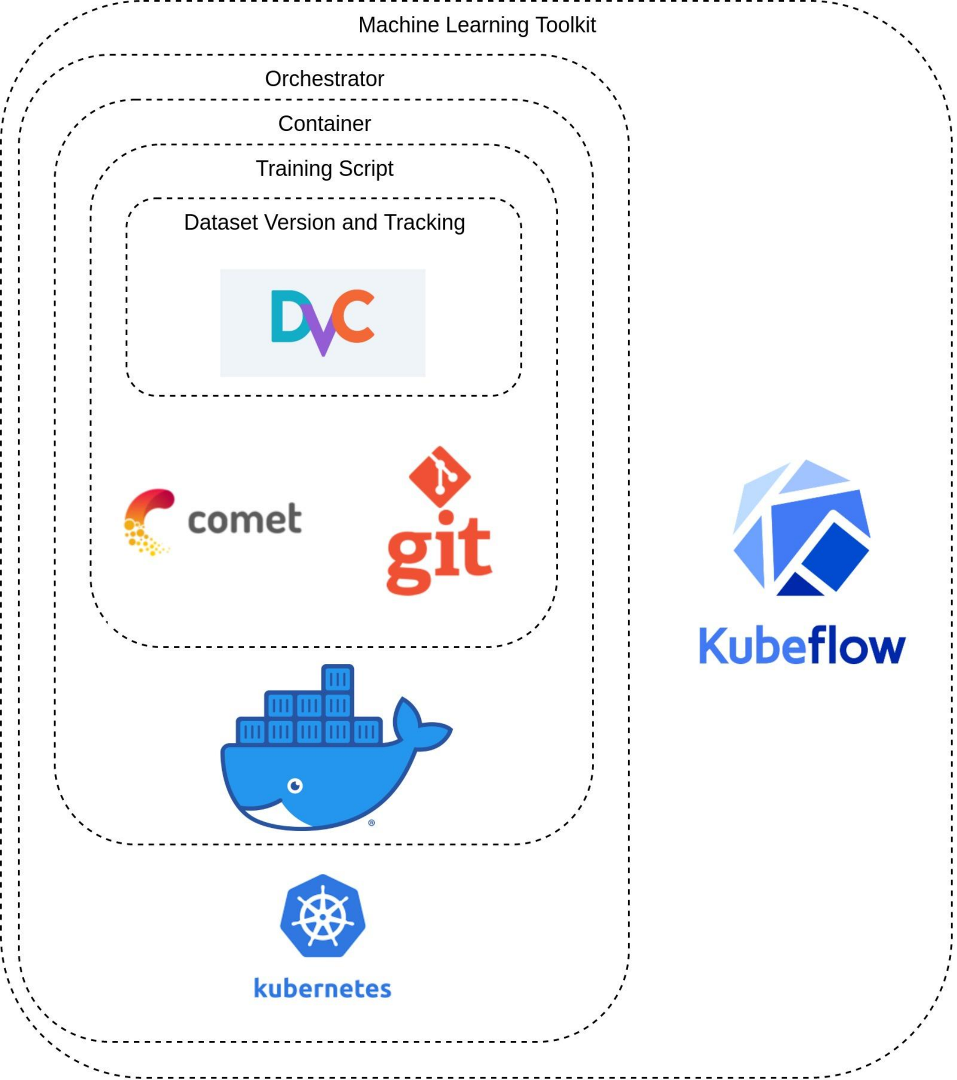
Dataset with Git Integration

- Use it like you develop your code
- Integrate with git

Clone it and Voila !

- As easy as clone a repo, select a version, pull

```
$ git clone git@gitlab.com:myrepo/data.git
$ git checkout v1
$ dvc pull
```


Machine Learning Toolkit
Orchestrator
Container
Training Script
Dataset Version and Tracking
DVC
comet
git
Kubeflow
kubernetes

Summary

nodeflux

Thank You

www.nodeflux.io