



---

Production ML Pipelines  
with Kubeflow

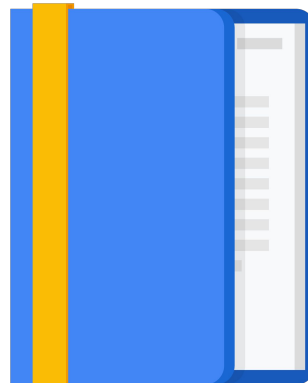
# Agenda

---

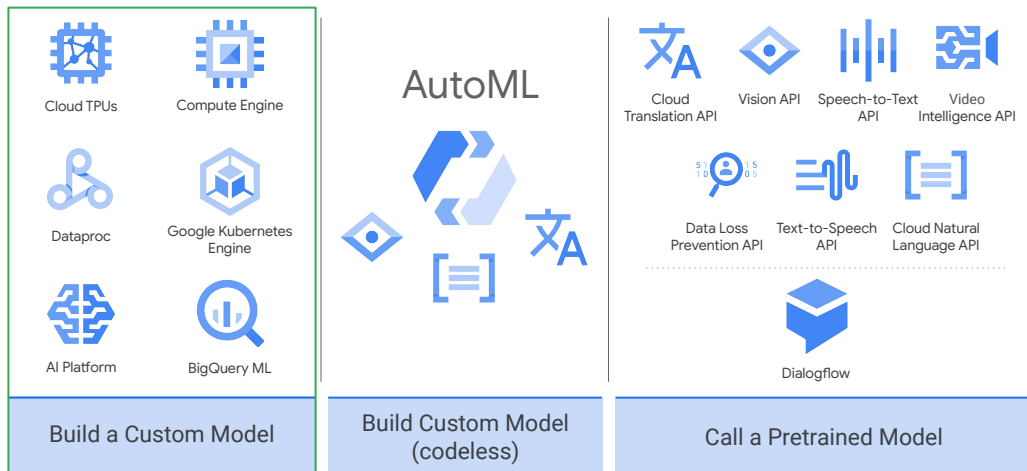
Ways to do ML on GCP

Kubeflow

AI Hub



# Create and deploy custom models with KubeFlow



## AI Platform is a fully managed service for custom machine learning models



- Scales to production
- Batching and distribution of model training
- Performs transformations on input data
- Hyper-parameter tuning
- Host and autoscale predictions
- Serverless - self-tuning - manages overhead

In this course, we don't cover writing TensorFlow models, only ways to operationalize them

[Google Cloud Training - Machine Learning and AI](#)

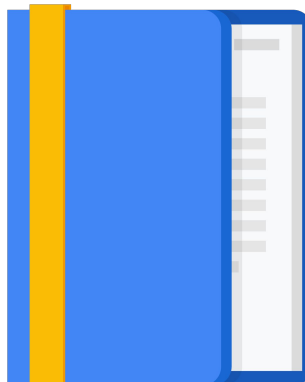
# Agenda

---

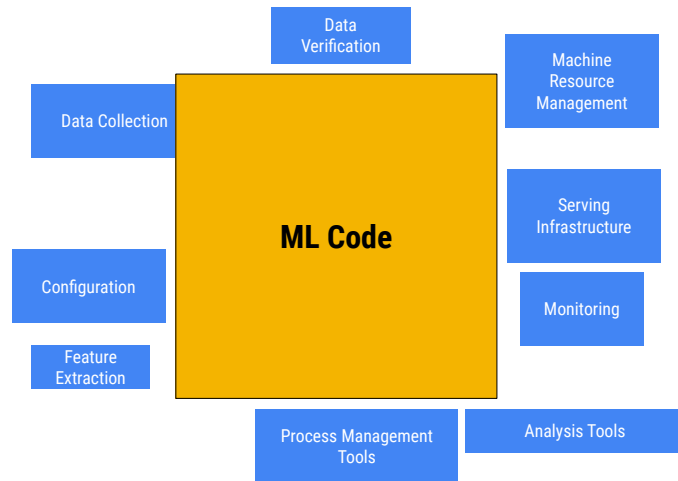
Ways to do ML on GCP

Kubeflow

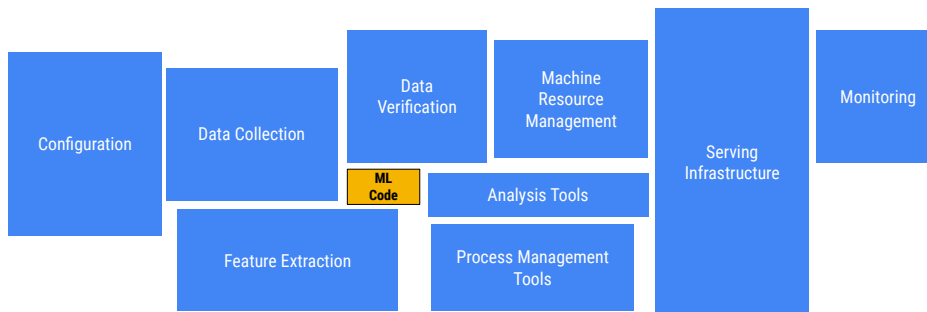
AI Hub



## Perception: ML products are mostly about ML



## Reality: ML Requires lots of DevOps



Source: [Sculley et al.: Hidden Technical Debt in Machine Learning Systems](#)

ML systems are large, complicated distributed systems.  
We started building Kubeflow to tackle these devops challenges using Kubernetes and containers



## Kubeflow provides a platform for building ML products

- Leverage containers and Kubernetes to solve the challenges of building ML products
- Kubeflow = Cloud Native, multi-cloud solution for ML.
- Kubeflow provides a platform for composable, portable and scalable ML pipelines
- If you have a Kubernetes conformant cluster, you can run Kubeflow

## Kubernetes is a great platform for ML

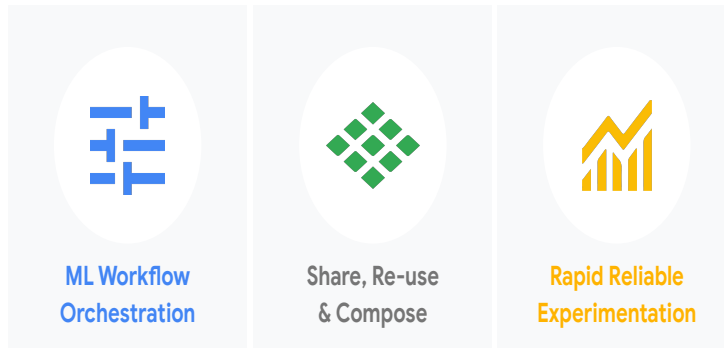
- Containers
- Scaling built in
- Unified architecture
- Easy to integrate building blocks
  - ML APIs
  - Dataflow
- Lots of options for CI/CD
- Portability
  - Dev, On-Prem, Multi-cloud: same stack



Before we dive into the specifics of Kubeflow Pipelines, I should provide additional context.

Kubeflow Pipelines is a part of the open source project Kubeflow. Kubeflow is a platform that provides the tools and scalable services required to develop and deploy ML workloads, all the way from distributed training, to scalable serving, to Notebooks w/ JupyterHub and workflow orchestration and much more. Kubeflow services are built on top on Kubernetes. Kubernetes provides scalability and hybrid protability. You can run Kubeflow anywhere you can run a Kubernetes cluster, and thus applications built on Kubeflow are portable across clouds and on-premise environmetns. On GCP, You can easily deploy Kubeflow on Google Kubernetes Engine.

Kubeflow Pipelines enable:



Ok now lets dive into Kubeflow Pipelines.

The capabilities provided by Kubeflow pipelines can largely be put into three buckets:

# What constitutes a Kubeflow Pipeline

## Containerized implementations of ML Tasks

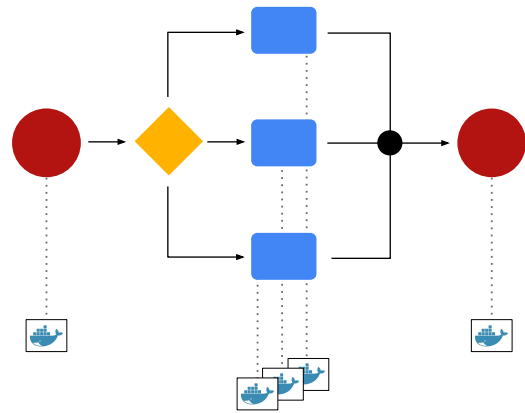
- Containers provide portability, repeatability and encapsulation
- A task can be single node or \*distributed\*
- A containerized task can invoke other services like CMLE, Dataflow or Dataproc

## Specification of the sequence of steps

- Specified via Python SDK

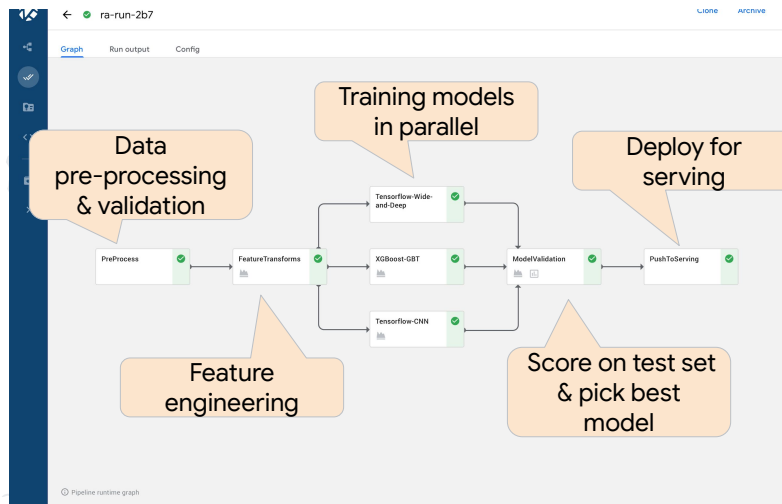
## Input Parameters

- A "Job" = Pipeline invoked w/ specific parameters



Let me provide a peek under the hood:

## Visual depiction of pipeline topology



Google Cloud

To make things more concrete let's look at a screenshot of an illustrative workflow that was run on Kubeflow Pipelines.

This is just an illustrative workflow and users can author and run many different kinds of workflow topologies with different code & tools in the various steps of the workflow

For each workflow that is run on Kubeflow Pipelines, you get a rich visual depiction of the topology so that you know what was executed as part of the workflow

In this workflow we start with a data preprocessing and validation step

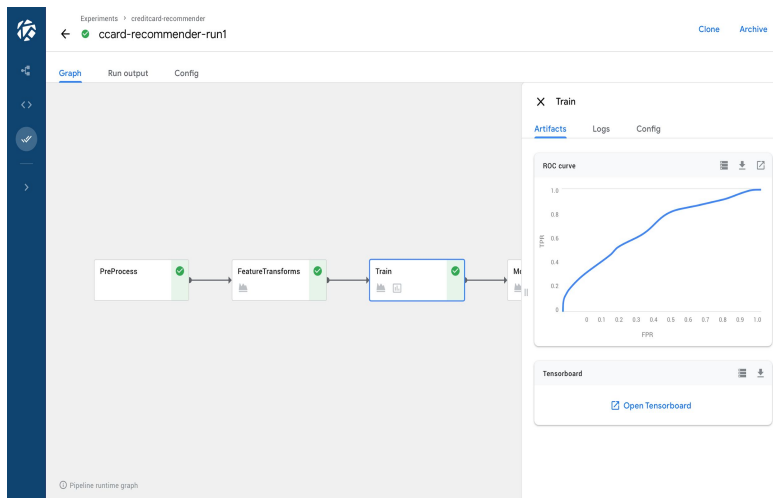
Followed by feature engineering

Following by a fork where we train many different kinds of models

The models that are trained are then analyzed and compared on a test data set

Finally, if an improved model is produced, it is deployed to a serving endpoint

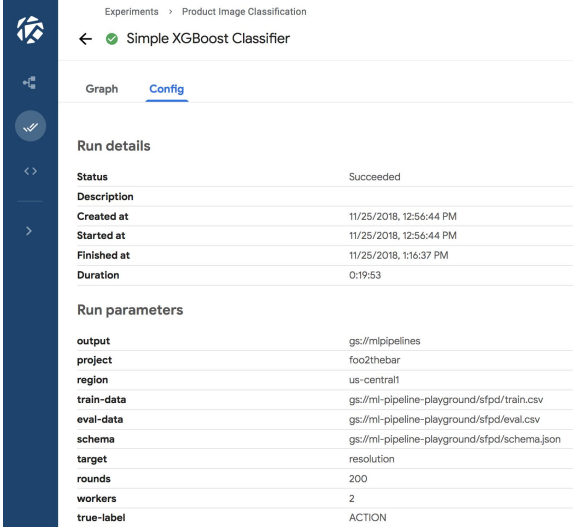
# Rich visualization of metrics



For each step of the workflow, you have rich ML specific information at your finger tips. Just click on a step and visualize relevant metrics produced by that step, such as an ROC curve for example.

If you did model training and produced the rich metadata that can be visualized with TensorBoard, that is just a click away.

## View all configs, inputs and outputs



The screenshot displays the Google Cloud AI Platform console interface. On the left is a dark blue sidebar with navigation icons. The main content area shows the breadcrumb 'Experiments > Product Image Classification' and a back arrow next to the experiment name 'Simple XGBoost Classifier'. Below this are tabs for 'Graph' and 'Config', with 'Config' being the active tab. The 'Run details' section shows a table with the following data:

Run details	
Status	Succeeded
Description	
Created at	11/25/2018, 12:56:44 PM
Started at	11/25/2018, 12:56:44 PM
Finished at	11/25/2018, 1:16:37 PM
Duration	0:19:53

Below the run details is the 'Run parameters' section, which contains a table of configuration parameters:

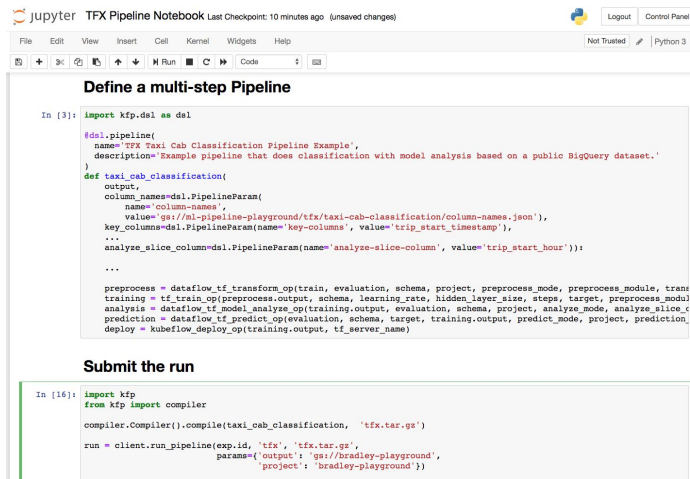
Run parameters	
output	gs://mlpipelines
project	foo2thebar
region	us-central1
train-data	gs://ml-pipeline-playground/sfpd/train.csv
eval-data	gs://ml-pipeline-playground/sfpd/eval.csv
schema	gs://ml-pipeline-playground/sfpd/schema.json
target	resolution
rounds	200
workers	2
true-label	ACTION



For each step of the workflow you can see the precise configuration parameters, and inputs and outputs.

Thus, for a model trained with Kubeflow Pipelines, you never have to wonder, how exactly did I create this model

# Author pipelines with an intuitive Python SDK



The screenshot shows a Jupyter Notebook interface with the title 'TFX Pipeline Notebook' and a status bar indicating 'Last Checkpoint: 10 minutes ago (unsaved changes)'. The notebook has a menu bar with 'File', 'Edit', 'View', 'Insert', 'Cell', 'Kernel', 'Widgets', and 'Help'. Below the menu bar is a toolbar with icons for running, saving, and other actions. The notebook content is divided into two sections: 'Define a multi-step Pipeline' and 'Submit the run'.

```
In [3]: import kfp.dsl as dsl

@dsl.pipeline(
    name='TFX Taxi Cab Classification Pipeline Example',
    description='Example pipeline that does classification with model analysis based on a public BigQuery dataset.'
)
def taxi_cab_classification(
    output,
    column_names=dsl.PipelineParam(
        name='column-names',
        value='gs://ml-pipeline-playground/tfx/taxi-cab-classification/column-names.json',
    ),
    key_column=dsl.PipelineParam(name='key-column', value='trip_start_timestamp'),
    ...
    analyze_slice_column=dsl.PipelineParam(name='analyze-slice-column', value='trip_start_hour'):
    ...

    preprocess = dataflow_tf_transform_op(train, evaluation, schema, project, preprocess_mode, preprocess_module, train
    training = tf_train_op(preprocess.output, schema, learning_rate, hidden_layer_size, steps, target, preprocess_model
    analysis = dataflow_tf_model_analyze_op(training.output, evaluation, schema, project, analyze_mode, analyze_slice_c
    prediction = dataflow_tf_predict_op(evaluation, schema, target, training.output, predict_mode, project, prediction
    deploy = kubernetes_deploy_op(training.output, tf_server_name)
```

**Submit the run**

```
In [16]: import kfp
from kfp import compiler

compiler.Compiler().compile(taxi_cab_classification, 'tfx.tar.gz')

run = client.run_pipeline(exp_id, 'tfx', 'tfx.tar.gz',
    params={'output': 'gs://bradley-playground',
            'project': 'bradley-playground'})
```



Lingua franca of ML practitioners

Python SDK to define the workflow i.e. define every step of the workflow, the inputs and output, and also define how the various steps are connected. The topology of the workflow is implicitly defined by connecting the various steps i.e. connecting the outputs of an upstream step to the inputs of a downstream step. You can also define looping constructs as well as conditional steps.



## Package & share pipelines as zip files

- Upload and execute pipelines via UI (in addition to API/SDK)
- Pipeline steps can be authored as reusable components



### Run details

Artifact:  [Choose](#)

Run name:

Description (optional):

### Run parameters

Specify parameters required by the pipeline

Output:

project:

region:

train-data:

eval-data:

schema:

target:

rounds:

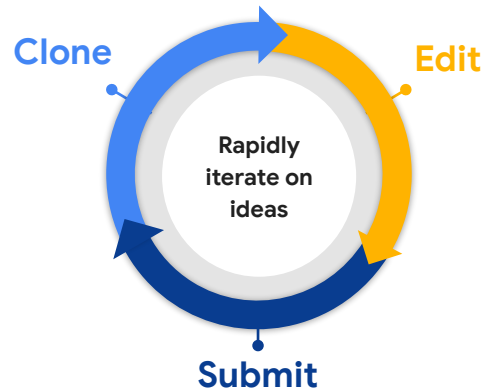
workers:

run-label:

[Create](#) [Cancel](#)

## Rapid, Reliable, Experimentation

- Every run logged with all config params, inputs, outputs & metrics
- Easily search and find old runs
- Clone and re-run or modify



View all current  
and past runs in  
one place

Experiments

[+ Create experiment](#) [Compare runs](#) [Archive](#)

[All experiments](#) [All runs](#)

Filter experiments

<input type="checkbox"/>	Experiment name	Last 5 runs	Created on ↑	Created by
<input type="checkbox"/>	▶ tfma-experiment	🟢	6:17 PM, Aug 24, 2018	John Doe
<input type="checkbox"/>	▶ xgboost-train	🟢🟢🟢	6:17 PM, Aug 24, 2018	John Doe
<input type="checkbox"/>	▶ promo-email	🟢🟢🟢🟢🔴	6:17 PM, Aug 24, 2018	Walter Fisher
<input type="checkbox"/>	▶ data-prep	🟢	6:17 PM, Aug 24, 2018	Walter Fisher
<input type="checkbox"/>	▶ tf-preprocessing	🟢🟢🟢	6:17 PM, Aug 24, 2018	John Doe
<input type="checkbox"/>	▶ tf-training	🟢🟢🟢🟢🔴	6:17 PM, Aug 24, 2018	Walter Fisher
<input type="checkbox"/>	▶ tf-serving	🟢🟢🟢	6:17 PM, Aug 24, 2018	Walter Fisher

Rows per page: 10 1-10 of 241 < >

# Easy comparison and analysis of runs

Experiments

← Image-classifier

Edit

Archive

Fastest run time

Slowest run time

1m 59s

3m 20s

[View run](#)

[View run](#)

All runs

Start new run

Start recurring run

Compare runs

Stop

Archive

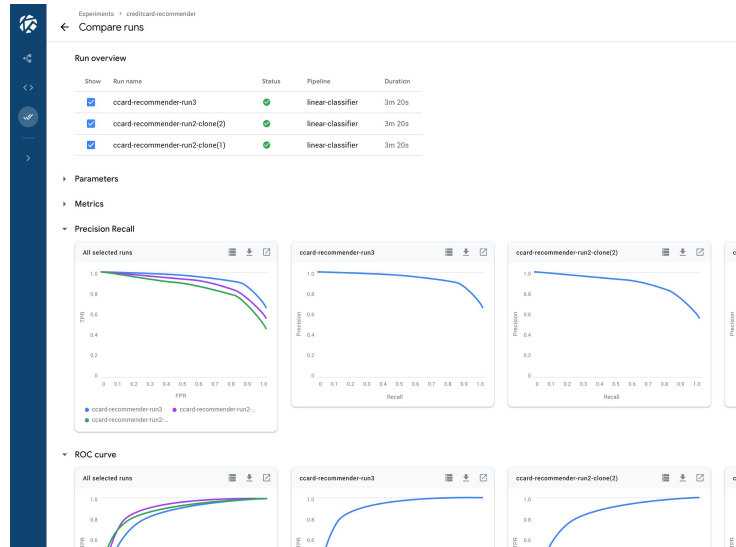
Metrics

Filter runs

<input type="checkbox"/> Runs	Status	Duration	Pipeline	Recurring run config.	Start time ↑	rms	eta
<input type="checkbox"/> ccard-recommender-run3	✓	1m 59s	linear-classifier		9:32 AM, Aug 26, 2018	0.88	0.92
<input type="checkbox"/> ccard-recommender-run2-clone(2)	✓	2m 12s	linear-classifier		11:42 AM, Aug 25, 2018	0.72	0.86
<input type="checkbox"/> ccard-recommender-run2-clone(1)	✓	2m 44s	linear-classifier		10:48 AM, Aug 25, 2018	0.74	0.84
<input type="checkbox"/> ccard-recommender-run2	✓	2m 18s	linear-classifier		10:22 PM, Aug 25, 2018	0.82	0.76
<input type="checkbox"/> ccard-recommender-run1-clone(1)	✓	2m 20s	linear-classifier		10:10 AM, Aug 25, 2018	0.80	0.84
<input type="checkbox"/> ccard-recommender-run1	✓	3m 20s	linear-classifier		6:17 PM, Aug 24, 2018	0.72	0.76

Rows per page: 10 1-10 of 241

# Easy comparison and analysis of runs



Hone in on the technique or parameter that was different

Quickly identify what worked and what did not work

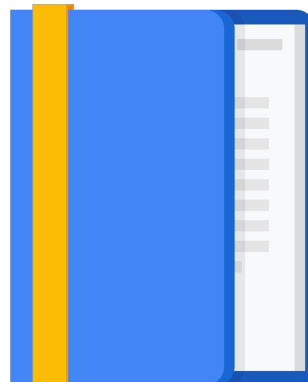
# Agenda

---

Ways to do ML on GCP

Kubeflow

AI Hub



# AI Hub is a repository for AI assets

- Don't reinvent the wheel! Find and deploy ML pipelines

The screenshot displays the Google Cloud AI Hub interface. On the left, there's a navigation sidebar with sections like 'Assets', 'My assets', 'Filter', 'Public', 'Private', 'Company', 'Kubeflow pipeline', 'Notebooks', 'Service', 'TensorFlow module', 'VM image', 'Trained model', 'Technical guide', 'Data type', 'Image', 'Text', 'Video', 'Other', 'No location', 'Data gathering', 'Data preparation', 'Training', 'Inference', 'Documentation', 'Feedback', 'Google Site Terms', 'Terms of service', and 'Privacy'. The main content area features a 'TensorFlow modules' banner, followed by 'Kubeflow pipelines' and 'Notebooks' sections. Each section lists various AI assets with icons, titles, and brief descriptions. For example, under 'Kubeflow pipelines', there are items like 'Submitting a SparkJob job to Cloud Dataproc', 'Data preparation by using the General Purpose Preprocessing component', and 'Batch predicting using Cloud Machine Learning Engine'. Under 'Notebooks', there are 'Text generation using a BERT with eager execution', 'Piano Transcription', and 'Training and Prediction with Kibiboot'. At the bottom, there's a 'Services' section with 'Cloud Text-to-Speech'.



## AI Hub stores various asset types

- Kubeflow pipelines and components
- Jupyter notebooks
- TensorFlow modules
- Trained models
- Services
- VM images



# This is what a typical asset looks like...

AI Hub

Search

Deploying a trained model to Cloud Machine Learning Engine

Scope

Public

Version

1

Category

Kubeflow pipeline

Publisher

Google

Data type

Text

Labels

GCP

ML Engine

Kubeflow

Pipeline

Pipelines are standalone solutions that can integrate into your existing workflow or be used as end-to-end solutions

Learn more

Use this asset

Download

Create a Kubeflow Cluster to use pipelines

Learn more about how to use pipelines

Feedback

Feedback

Twitter Facebook LinkedIn

Documentation

Deploying a trained model to Cloud Machine Learning Engine

A Kubeflow Pipeline component to deploy a trained model from a Cloud Storage path to a Cloud Machine Learning Engine service.

Intended use

Use the component to deploy a trained model to Cloud Machine Learning Engine service. The deployed model can serve online or batch predictions in a KFP pipeline.

Runtime arguments:

Name	Description	Type	Optional	Default
model_uri	The Cloud Storage URI which contains a model file. The commonly used TF model search path (export/exporter) will be used.	GCSPath	No	
project_id	The ID of the parent project of the serving model.	GCPProjectID	No	
model_id	The user-specified name of the model. If it is not provided, the operation uses a random name.	String	Yes	
version_id	The user-specified name of the version. If it is not provided, the operation uses a random name.	String	Yes	
runtime_version	The Cloud Machine Learning Engine's runtime version to use for this deployment. If it is not set, the Cloud ML Engine uses the default stable version, 1.0.	String	Yes	
python_version	The version of Python used in the prediction. If it is not set, the default version is 2.7. Python 3.5 is available when the runtime_version is set to 1.4 and above. Python 2.7 works with all supported runtime versions.	String	Yes	
version	The JSON payload of the new Version.	Dict	Yes	
replace_existing_version	A Boolean flag indicates whether to replace existing version in case of conflict.	Bool	Yes	False
set_default	A Boolean flag indicates whether to set the new version as default version in the model.	Bool	Yes	False
waitInterval	A time interval to wait for in case the operation has a long run time.	Integer	Yes	30

Output:

One-click deployment of ML pipelines via Kubeflow on GCP as platform for AI, or on premise.

## Assets on AI Hub are collected in two scopes: public assets and restricted assets

- Public scope are available to all AI Hub users
- Restricted scope contains AI components that you have uploaded and assets that have been shared with you



## Running ML Pipelines on Kubeflow

---

### Objectives

- Create a Kubernetes cluster and configure AI Platform pipelines
- Launch the pipelines dashboard
- Create and run an experiment from an example end-to-end ML Pipeline
- Examine and verify the output of each step
- Inspect the pipeline graph, various metrics, logs, charts and parameters

In this lab you learn how to install and use Kubeflow Pipelines.

Once Kubeflow Pipelines are installed, you create and run an experiment end-to-end ML Pipeline.

When the pipeline is complete you will examine the pipeline graph, metrics, logs and parameters.

## Module Summary

- Use ML on GCP using either
  - AI Platform (your model, your data)
  - AutoML (our models, your data)
  - Perception API (our models, our data)
- Use Kubeflow to deploy end-to-end ML pipelines
- Don't reinvent the wheel for your ML pipeline! Leverage pipelines on AI Hub