Continuous training pipeline with KFP and Cloud AI Platform

Learning Objectives:

1. Learn how to use KF pre-build components (BiqQuery, CAIP training and predictions)
2. Learn how to use KF lightweight python components
3. Learn how to build a KF pipeline with these components
4. Learn how to compile, upload, and run a KF pipeline with the command line

In this lab, you will build, deploy, and run a KFP pipeline that orchestrates **BigQuery** and **Cloud AI Platform** services to train, tune, and deploy a **scikit-learn** model.

Understanding the pipeline design

The workflow implemented by the pipeline is defined using a Python based Domain Specific Language (DSL). The pipeline's DSL is in the `covertype_training_pipeline.py` file that we will generate below.

The pipeline's DSL has been designed to avoid hardcoding any environment specific settings like file paths or connection strings. These settings are provided to the pipeline code through a set of environment variables.

In [ ]:

```
!grep 'BASE_IMAGE =' -A 5 pipeline/covertype_training_pipeline.py
```

The pipeline uses a mix of custom and pre-build components.

- Pre-build components. The pipeline uses the following pre-build components that are included with the KFP distribution:
  - BigQuery query component
  - AI Platform Training component
  - AI Platform Deploy component
- Custom components. The pipeline uses two custom helper components that encapsulate functionality not available in any of the pre-build components. The components are implemented using the KFP SDK's Lightweight Python Components mechanism. The code for the components is in the `helper_components.py` file:
  - **Retrieve Best Run**. This component retrieves a tuning metric and hyperparameter values for the best run of a AI Platform Training hyperparameter tuning job.
  - **Evaluate Model**. This component evaluates a *sklearn* trained model using a provided metric and a testing dataset.

In [ ]:

```
%%writefile ./pipeline/covertype_training_pipeline.py
# Copyright 2019 Google LLC
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#      http://www.apache.org/licenses/LICENSE-2.0
#
```

```
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
"""KFP pipeline orchestrating BigQuery and Cloud AI Platform services."""

import os

from helper_components import evaluate_model
from helper_components import retrieve_best_run
from jinja2 import Template
import kfp
from kfp.components import func_to_container_op
from kfp.dsl.types import Dict
from kfp.dsl.types import GCPProjectID
from kfp.dsl.types import GCPRegion
from kfp.dsl.types import GCSPath
from kfp.dsl.types import String
from kfp.gcp import use_gcp_secret

# Defaults and environment settings
BASE_IMAGE = os.getenv('BASE_IMAGE')
TRAINER_IMAGE = os.getenv('TRAINER_IMAGE')
RUNTIME_VERSION = os.getenv('RUNTIME_VERSION')
PYTHON_VERSION = os.getenv('PYTHON_VERSION')
COMPONENT_URL_SEARCH_PREFIX = os.getenv('COMPONENT_URL_SEARCH_PREFIX')
USE_KFP_SA = os.getenv('USE_KFP_SA')

TRAINING_FILE_PATH = 'datasets/training/data.csv'
VALIDATION_FILE_PATH = 'datasets/validation/data.csv'
TESTING_FILE_PATH = 'datasets/testing/data.csv'

# Parameter defaults
SPLITS_DATASET_ID = 'splits'
HYPERTUNE_SETTINGS = """
{
    "hyperparameters":  {
        "goal": "MAXIMIZE",
        "maxTrials": 6,
        "maxParallelTrials": 3,
        "hyperparameterMetricTag": "accuracy",
        "enableTrialEarlyStopping": True,
        "params": [
            {
                "parameterName": "max_iter",
                "type": "DISCRETE",
                "discreteValues": [500, 1000]
            },
            {
                "parameterName": "alpha",
                "type": "DOUBLE",
                "minValue": 0.0001,
                "maxValue": 0.001,
                "scaleType": "UNIT_LINEAR_SCALE"
            }
        ]
    }
}
"""



# Helper functions
def generate_sampling_query(source_table_name, num_lots, lots):
    """Prepares the data sampling query."""
```

```
    sampling_query_template = """
         SELECT *
         FROM
             `{{ source_table }}` AS cover
         WHERE
         MOD(ABS(FARM_FINGERPRINT(TO_JSON_STRING(cover))), {{ num_lots }}) IN
         """
    query = Template(sampling_query_template).render(
        source_table=source_table_name, num_lots=num_lots, lots=str(lots)[1:-

    return query


# Create component factories
component_store = kfp.components.ComponentStore(
    local_search_paths=None, url_search_prefixes=[COMPONENT_URL_SEARCH_PREFIX

bigquery_query_op = component_store.load_component('bigquery/query')
mlengine_train_op = component_store.load_component('ml_engine/train')
mlengine_deploy_op = component_store.load_component('ml_engine/deploy')
retrieve_best_run_op = func_to_container_op(
    retrieve_best_run, base_image=BASE_IMAGE)
evaluate_model_op = func_to_container_op(evaluate_model, base_image=BASE_IMAG


@kfp.dsl.pipeline(
    name='Covertype Classifier Training',
    description='The pipeline training and deploying the Covertype classifier
)
def covertype_train(project_id,
                    region,
                    source_table_name,
                    gcs_root,
                    dataset_id,
                    evaluation_metric_name,
                    evaluation_metric_threshold,
                    model_id,
                    version_id,
                    replace_existing_version,
                    hypertune_settings=HYPERTUNE_SETTINGS,
                    dataset_location='US'):
    """Orchestrates training and deployment of an sklearn model."""

    # Create the training split
    query = generate_sampling_query(
        source_table_name=source_table_name, num_lots=10, lots=[1, 2, 3, 4])

    training_file_path = '{}/{}'.format(gcs_root, TRAINING_FILE_PATH)

    create_training_split = bigquery_query_op(
        query=query,
        project_id=project_id,
        dataset_id=dataset_id,
        table_id='',
        output_gcs_path=training_file_path,
        dataset_location=dataset_location)

    # Create the validation split
    query = generate_sampling_query(
        source_table_name=source_table_name, num_lots=10, lots=[8])

    validation_file_path = '{}/{}'.format(gcs_root, VALIDATION_FILE_PATH)

    create_validation_split = bigquery_query_op(
```

```
        query=query,
        project_id=project_id,
        dataset_id=dataset_id,
        table_id='',
        output_gcs_path=validation_file_path,
        dataset_location=dataset_location)

    # Create the testing split
    query = generate_sampling_query(
        source_table_name=source_table_name, num_lots=10, lots=[9])

    testing_file_path = '{}/{}'.format(gcs_root, TESTING_FILE_PATH)

    create_testing_split = bigquery_query_op(
        query=query,
        project_id=project_id,
        dataset_id=dataset_id,
        table_id='',
        output_gcs_path=testing_file_path,
        dataset_location=dataset_location)

    # Tune hyperparameters
    tune_args = [
        '--training_dataset_path',
        create_training_split.outputs['output_gcs_path'],
        '--validation_dataset_path',
        create_validation_split.outputs['output_gcs_path'], '--hptune', 'True
    ]

    job_dir = '{}/{}/{}'.format(gcs_root, 'jobdir/hypertune',
                                kfp.dsl.RUN_ID_PLACEHOLDER)

    hypertune = mlengine_train_op(
        project_id=project_id,
        region=region,
        master_image_uri=TRAINER_IMAGE,
        job_dir=job_dir,
        args=tune_args,
        training_input=hypertune_settings)

    # Retrieve the best trial
    get_best_trial = retrieve_best_run_op(
            project_id, hypertune.outputs['job_id'])

    # Train the model on a combined training and validation datasets
    job_dir = '{}/{}/{}'.format(gcs_root, 'jobdir', kfp.dsl.RUN_ID_PLACEHOLDE

    train_args = [
        '--training_dataset_path',
        create_training_split.outputs['output_gcs_path'],
        '--validation_dataset_path',
        create_validation_split.outputs['output_gcs_path'], '--alpha',
        get_best_trial.outputs['alpha'], '--max_iter',
        get_best_trial.outputs['max_iter'], '--hptune', 'False'
    ]

    train_model = mlengine_train_op(
        project_id=project_id,
        region=region,
        master_image_uri=TRAINER_IMAGE,
        job_dir=job_dir,
        args=train_args)

    # Evaluate the model on the testing split
    eval_model = evaluate_model_op(
```

```
        dataset_path=str(create_testing_split.outputs['output_gcs_path']),
        model_path=str(train_model.outputs['job_dir']),
        metric_name=evaluation_metric_name)

    # Deploy the model if the primary metric is better than threshold
    with kfp.dsl.Condition(eval_model.outputs['metric_value'] > evaluation_me
        deploy_model = mlengine_deploy_op(
        model_uri=train_model.outputs['job_dir'],
        project_id=project_id,
        model_id=model_id,
        version_id=version_id,
        runtime_version=RUNTIME_VERSION,
        python_version=PYTHON_VERSION,
        replace_existing_version=replace_existing_version)

    # Configure the pipeline to run using the service account defined
    # in the user-gcp-sa k8s secret
    if USE_KFP_SA == 'True':
        kfp.dsl.get_pipeline_conf().add_op_transformer(
              use_gcp_secret('user-gcp-sa'))
```

The custom components execute in a container image defined in `base_image/Dockerfile`.

In [ ]:
```
!cat base_image/Dockerfile
```

The training step in the pipeline employes the AI Platform Training component to schedule a AI Platform Training job in a custom training container. The custom training image is defined in `trainer_image/Dockerfile`.

In [ ]:
```
!cat trainer_image/Dockerfile
```

Building and deploying the pipeline

Before deploying to AI Platform Pipelines, the pipeline DSL has to be compiled into a pipeline runtime format, also refered to as a pipeline package. The runtime format is based on Argo Workflow, which is expressed in YAML.

Configure environment settings

Update the below constants with the settings reflecting your lab environment.

- `REGION` - the compute region for AI Platform Training and Prediction
- `ARTIFACT_STORE` - the GCS bucket created during installation of AI Platform Pipelines. The bucket name will be similar to `qwiklabs-gcp-xx-xxxxxxx-kubeflowpipelines-default`.
- `ENDPOINT` - set the `ENDPOINT` constant to the endpoint to your AI Platform Pipelines instance. Then endpoint to the AI Platform Pipelines instance can be found on the AI Platform Pipelines page in the Google Cloud Console.

1. Open the **SETTINGS** for your instance
2. Use the value of the `host` variable in the **Connect to this Kubeflow Pipelines instance from a Python client via Kubeflow Pipelines SKD** section of the **SETTINGS** window.

In [ ]:
```
REGION = 'us-central1'
ENDPOINT = '337dd39580cbcbd2-dot-us-central2.pipelines.googleusercontent.com'
ARTIFACT_STORE_URI = 'gs://qwiklabs-gcp-xx-xxxxxxx-kubeflowpipelines-default'
PROJECT_ID = !(gcloud config get-value core/project)
PROJECT_ID = PROJECT_ID[0]
```

Build the trainer image

In [ ]:
```
IMAGE_NAME='trainer_image'
TAG='latest'
TRAINER_IMAGE='gcr.io/{}/{}:{}'.format(PROJECT_ID, IMAGE_NAME, TAG)
```

Note: Please ignore any **incompatibility ERROR** that may appear for the packages visions as it will not affect the lab's functionality.

In [ ]:
```
!gcloud builds submit --timeout 15m --tag $TRAINER_IMAGE trainer_image
```

Build the base image for custom components

In [ ]:
```
IMAGE_NAME='base_image'
TAG='latest'
BASE_IMAGE='gcr.io/{}/{}:{}'.format(PROJECT_ID, IMAGE_NAME, TAG)
```

In [ ]:
```
!gcloud builds submit --timeout 15m --tag $BASE_IMAGE base_image
```

Compile the pipeline

You can compile the DSL using an API from the **KFP SDK** or using the **KFP** compiler.

To compile the pipeline DSL using the **KFP** compiler.

Set the pipeline's compile time settings

The pipeline can run using a security context of the GKE default node pool's service account or the service account defined in the `user-gcp-sa` secret of the Kubernetes namespace hosting Kubeflow Pipelines. If you want to use the `user-gcp-sa` service account you change the value of `USE_KFP_SA` to `True` .

Note that the default AI Platform Pipelines configuration does not define the `user-gcp-sa` secret.

In [ ]:
```
USE_KFP_SA = False

COMPONENT_URL_SEARCH_PREFIX = 'https://raw.githubusercontent.com/kubeflow/pipe
RUNTIME_VERSION = '1.15'
PYTHON_VERSION = '3.7'

%env USE_KFP_SA={USE_KFP_SA}
%env BASE_IMAGE={BASE_IMAGE}
%env TRAINER_IMAGE={TRAINER_IMAGE}
%env COMPONENT_URL_SEARCH_PREFIX={COMPONENT_URL_SEARCH_PREFIX}
%env RUNTIME_VERSION={RUNTIME_VERSION}
%env PYTHON_VERSION={PYTHON_VERSION}
```

Use the CLI compiler to compile the pipeline

In [ ]:

```
!dsl-compile --py pipeline/covertype_training_pipeline.py --output covertype_
```

The result is the `covertype_training_pipeline.yaml` file.

In [ ]:

```
!head covertype_training_pipeline.yaml
```

Deploy the pipeline package

In [ ]:

```
PIPELINE_NAME='covertype_continuous_training'

!kfp --endpoint $ENDPOINT pipeline upload \
-p $PIPELINE_NAME \
covertype_training_pipeline.yaml
```

Submitting pipeline runs

You can trigger pipeline runs using an API from the KFP SDK or using KFP CLI. To submit the run using KFP CLI, execute the following commands. Notice how the pipeline's parameters are passed to the pipeline run.

List the pipelines in AI Platform Pipelines

In [ ]:

```
!kfp --endpoint $ENDPOINT pipeline list
```

Submit a run

Find the ID of the `covertype_continuous_training` pipeline you uploaded in the previous step and update the value of `PIPELINE_ID` .

In [ ]:

```
PIPELINE_ID='0918568d-758c-46cf-9752-e04a4403cd84' #Change
```

In [ ]:

```
EXPERIMENT_NAME = 'Covertype_Classifier_Training'
RUN_ID = 'Run_001'
SOURCE_TABLE = 'covertype_dataset.covertype'
DATASET_ID = 'splits'
EVALUATION_METRIC = 'accuracy'
EVALUATION_METRIC_THRESHOLD = '0.69'
MODEL_ID = 'covertype_classifier'
VERSION_ID = 'v01'
REPLACE_EXISTING_VERSION = 'True'

GCS_STAGING_PATH = '{}/staging'.format(ARTIFACT_STORE_URI)
```

In [ ]:

```
!kfp --endpoint $ENDPOINT run submit \
-e $EXPERIMENT_NAME \
-r $RUN_ID \
-p $PIPELINE_ID \
project_id=$PROJECT_ID \
```

```
gcs_root=$GCS_STAGING_PATH \
region=$REGION \
source_table_name=$SOURCE_TABLE \
dataset_id=$DATASET_ID \
evaluation_metric_name=$EVALUATION_METRIC \
evaluation_metric_threshold=$EVALUATION_METRIC_THRESHOLD \
model_id=$MODEL_ID \
version_id=$VERSION_ID \
replace_existing_version=$REPLACE_EXISTING_VERSION
```

where

- EXPERIMENT_NAME is set to the experiment used to run the pipeline. You can choose any name you want. If the experiment does not exist it will be created by the command
- RUN_ID is the name of the run. You can use an arbitrary name
- PIPELINE_ID is the id of your pipeline. Use the value retrieved by the `kfp pipeline list` command
- GCS_STAGING_PATH is the URI to the GCS location used by the pipeline to store intermediate files. By default, it is set to the `staging` folder in your artifact store.
- REGION is a compute region for AI Platform Training and Prediction.

You should be already familiar with these and other parameters passed to the command. If not go back and review the pipeline code.

Monitoring the run

You can monitor the run using KFP UI. Follow the instructor who will walk you through the KFP UI and monitoring techniques.

To access the KFP UI in your environment use the following URI:

https://[ENDPOINT]

NOTE that your pipeline run may fail due to the bug in a BigQuery component that does not handle certain race conditions. If you observe the pipeline failure, re-run the last cell of the notebook to submit another pipeline run or retry the run from the KFP UI