Overview of a Simple End-to-End Solution Pipeline

In this simple example, we will implement a deep neural regressor network to predict the closing prices of Bitcoin crypto-currency. The machine learning code itself is pretty basic as it is not the focus of this article. The goal here is to orchestrate a machine learning engineering solution using microservice architectures on Kubernetes with Kubeflow Pipelines. The code for this chapter is in the book code repository. Clone the repository from the GCP Cloud Shell.

The pipeline consists of the following components:

1. Move raw data hosted on GitHub to a storage bucket.
2. Transform the dataset using Google Dataflow.
3. Carry out hyper-parameter training on Cloud Machine Learning Engine.
4. Train the model with the optimized hyper-parameters.
5. Deploy the model for serving on Cloud MLE.

Create a Container Image for Each Component

First, we'll package the client and runtime code into a Docker image. This image also contains the secure service account key to authenticate against GCP. For example, the component to transform the dataset using Dataflow has the following files built into its image:

- `__ Dockerfile`: Dockerfile to build the Docker image.
- `__ build.sh`: Script to initiate the container build and upload to Google Container Registry.
- `__ dataflow_transform.py`: Code to run the beam pipeline on Cloud Dataflow.
- `__ service_account.json`: Secure key to authenticate container on GCP.
- `__ local_test.sh`: Script to run the image pipeline component locally.

Build Containers Before Uploading to Kubeflow Pipelines

Before uploading the pipeline to Kubeflow Pipelines, be sure to build the component containers so that the latest version of the code is packaged and uploaded as images to the container registry. The code provides a handy `bash` script to build all containers.

Compile the Pipeline Using the Kubeflow Pipelines DSL Language

The pipeline code contains a specification on how the components interact with one another. Each component has an output that serves as an input to the next component in the pipeline. The Kubeflow pipeline DSL language `dsl-compile` from the Kubeflow Pipelines SDK is used to compile the pipeline code in Python for upload to Kubeflow Pipelines.

Ensure the Kubeflow Pipelines SDK is installed on the local machine by running

```
# install kubeflow pipeline sdk
pip install https://storage.googleapis.com/ml-pipeline/release/0.1.12/kfp.
tar.gz --upgrade

# verify the install
which dsl-compile
```

Compile the pipeline by running

```
# compile the pipeline
python3 [path/to/python/file.py] [path/to/output/tar.gz]
```

For the sample code, we used

```
python3 crypto_pipeline.py crypto_pipeline.tar.gz
```