**Building a Kubeflow Pipeline.**

For this example, we would be working on a supervised learning classification problem using an Artificial Neural network. The goal is to determine if a customer would churn/leave or not.

**Step 1:**

The first step is to make sure you have successfully deployed kubeflow, all its dependencies and you have access to the dashboard. There are many ways to deploy kubeflow. It can be deployed on your system(MacOs, Windows, Linux) and on the cloud(GCP, AWS, IBM, Azure). The following links should guide you on how to deploy kubeflow on different platforms::

1. [Deploy Kubeflow on Windows](https://www.kubeflow.org/docs/started/workstation/getting-started-windows/)
2. [Deploy Kubeflow on MacOs](https://www.kubeflow.org/docs/started/workstation/getting-started-macos/)
3. [Deploy Kubeflow on Linux](https://www.kubeflow.org/docs/started/workstation/getting-started-linux/)
4. [Deploy Kubeflow on GCP](https://www.kubeflow.org/docs/gke/)
5. [Deploy Kubeflow on AWS](https://www.kubeflow.org/docs/aws/)
6. [Deploy Kubeflow on IBM](https://www.kubeflow.org/docs/ibm/iks-e2e/)
7. [Deploy Kubeflow on Azure](https://www.kubeflow.org/docs/azure/)

For this example we would be using GCP to build the pipeline. To hasten the process we have put together the following steps for deploying Kubeflow on GCP.

1. Create a new project
2. Enable boost shell on cloud shell
3. Enable deployment manager API
4. Setup OAuth credentials
5. Setup environmental files and enable other api’s

*1. export DEPLOYMENT\_NAME=kf-codelab*

*2. export PROJECT\_ID=your-project-id*

*3. export ZONE=us-central1-c or us-east1-c*

*4. gcloud config set project ${PROJECT\_ID}*

*5. gcloud config set compute/zone ${ZONE}*

* 1. *gcloud services enable cloudresourcemanager.googleapis.com iam.googleapis.com file.googleapis.com ml.googleapis.com*

6. Download kfctl v1.0 on the cloud shell with

wget <https://github.com/kubeflow/kfctl/releases/download/v1.1.0/kfctl_v1.1.0-0-g9a3621e_linux.tar.gz>

7. Extract the downloaded package with tar -xvf kfctl\_v1.1.0-0-g9a3621e\_linux.tar.gz

8. Export kfctl path using export PATH=$PATH:/home/home-name/

1. go to home directory using cd
2. Get the current directory with pwd

9. Authorize gcloud with gcloud auth login **and** gcloud auth application-default login

10.set config\_uri

*export CONFIG\_URI="*[*https://raw.githubusercontent.com/kubeflow/manifests/v1.0-branch/kfdef/kfctl\_gcp\_iap.v1.0.2.yaml*](https://raw.githubusercontent.com/kubeflow/manifests/v1.0-branch/kfdef/kfctl_gcp_iap.v1.0.2.yaml)*"*

11. Set up oauth secret

export CLIENT\_ID=your-auth-client-id

export CLIENT\_SECRET=your-oauth-client-secret

12. Setup kubeflow environment variables

KF\_NAME=kf-codelab

BASE\_DIR=${HOME}/kf\_deployments

export KF\_DIR=${BASE\_DIR}/${KF\_NAME}

13. Apply Kubeflow

mkdir -p ${KF\_DIR}

cd ${KF\_DIR}

kfctl apply -V -f ${CONFIG\_URI}

14. Create Cluster

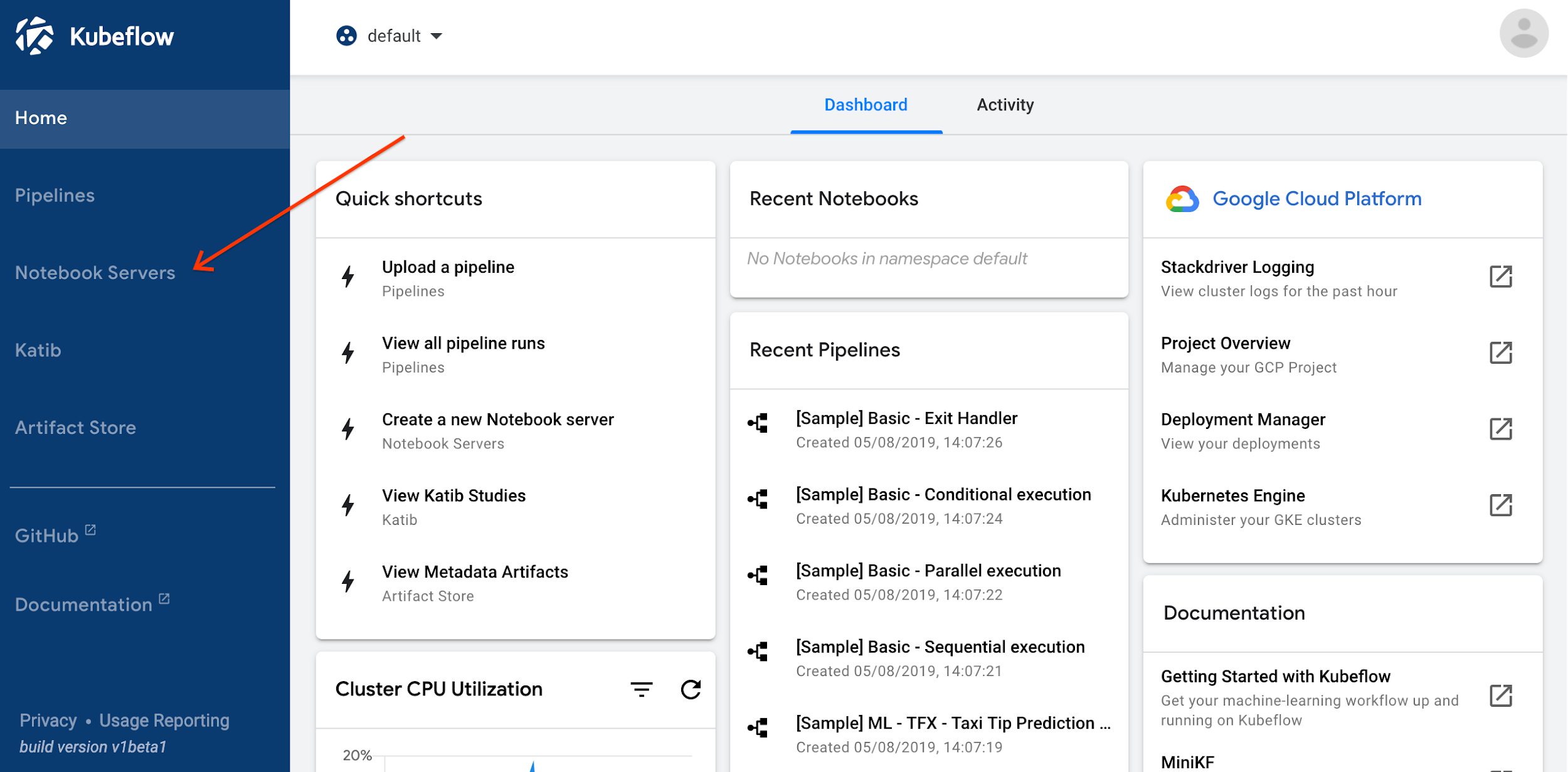
gcloud container clusters get-credentials ${KF\_NAME} --zone ${ZONE} --project ${PROJECT\_ID}

kubectl -n kubeflow get all

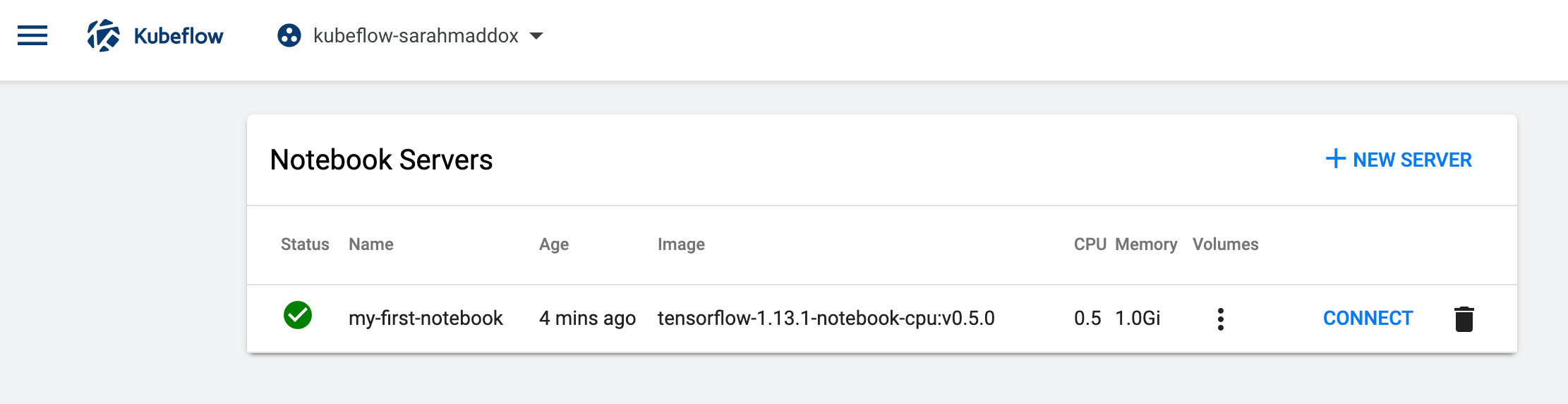
kubectl -n istio-system get ingress

**Step 2:**

Once you have followed the steps and deployed kubeflow on GCP, paste the link given in your browser to get access to the Kubeflow dashboard.

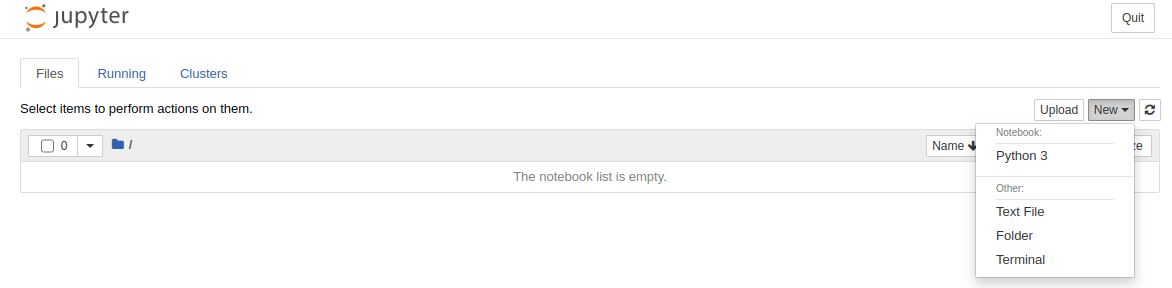


Start by setting up a jupyter notebook through the Notebook Servers tab. Follow the steps here to set up your notebook [server](https://www.kubeflow.org/docs/notebooks/setup/).



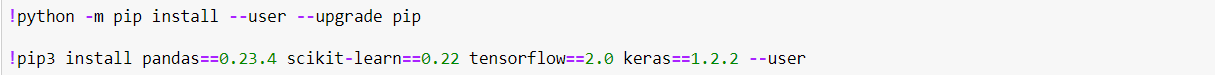
**Step 3:**

Once the server is set up, click **connect** for access to your jupyter notebook. Open up the terminal and clone the following repository from GitHub.



|  |
| --- |
| $ git clone <https://github.com/AdeloreSimiloluwa/Artificial-Neural-Network> |

Open the file “[Artificial Neural Networks Pipeline](https://github.com/AdeloreSimiloluwa/Artificial-Neural-Network/blob/master/Artificial%20Neural%20Networks%20Pipeline..ipynb)”. The data needed for this project has already been uploaded on git so you can proceed to run each cell in the notebook. Start by installing the needed libraries and restart the kernel. The next step is to import the data and preprocess.



**Step 4:**

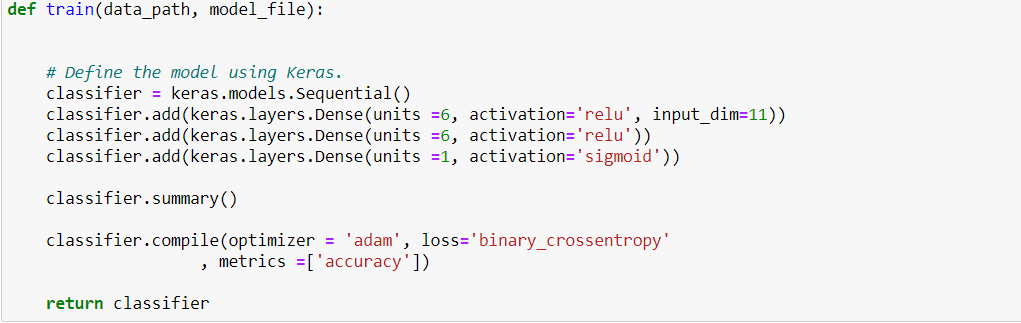
Now installing the kubeflow pipeline SDK. The [kubeflow pipeline SDK](https://www.kubeflow.org/docs/pipelines/sdk/sdk-overview/) offers python packages that you can use to run your machine learning workflows. To install SDK, run this cell and restart your kernel.

|  |
| --- |
| !pip install -q kfp --upgrade --user |

**Step 5:**

Create python functions for training, testing and prediction and convert them into containers with this function, [func to container op](https://kubeflow-pipelines.readthedocs.io/en/latest/source/kfp.components.html#kfp.components.func_to_container_op).

Training function:



Predict function:



Now wrap the up into container components with:

|  |
| --- |
| name = comp.func\_to\_container\_op(func) |

Parameters : func is the python function you intend to convert

**Step 6:**

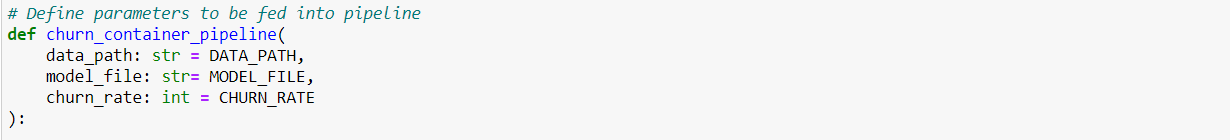
Usually, when defining the kubflow pipeline, one would have to manually manipulate a YAML file, but with Kubeflow Pipelines SDK you can define your pipeline with :

|  |
| --- |
| @dsl.pipeline(name= “”, description = “”) |

Before this, you need to initialize a kubeflow client that would enable communication with the Pipelines API server so you can create runs and experiment from your jupyter notebook.

|  |
| --- |
| client = kfp.Client() |

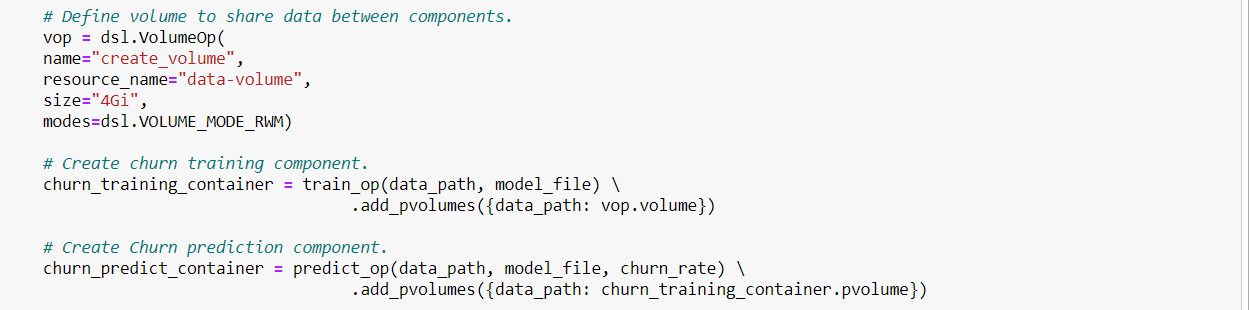
After this, you define the parameters that should go into the pipeline. In this case we first define the data path, the model path(where the model is stored), and the index of the test data you want predicted.



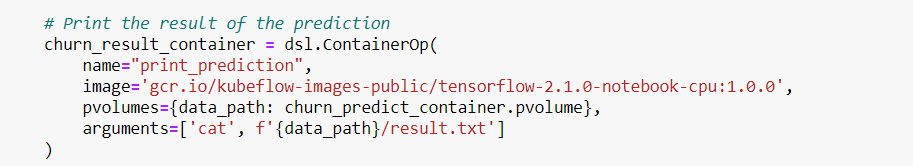
**Step 7:**

Now we can define the pipeline components, compile and run it on the dashboard. With [ContainerOp](https://kubeflow-pipelines.readthedocs.io/en/latest/source/kfp.dsl.html#kfp.dsl.ContainerOp) func we would define the components, order of operations and dependencies of the pipeline.

Here we create the training components and attach [persistent volumes](https://kubernetes.io/docs/concepts/storage/persistent-volumes/) to be mounted to the container.



Now print your results:



ContainerOp parameters include:

**name** - the name displayed for the component execution during runtime.

**image** - image tag for the Docker container to be used.

**pvolumes** - dictionary of paths and associated Persistent Volumes to be mounted to the container before execution.

**arguments** - command to be run by the container at runtime.

Run the next two cells to compile the pipeline and run it within an experiment. Click **run** to view your pipeline on the Kubeflow pipeline UI.



The components you defined in the notebook should be displayed on the UI.

Once the components are done running you can check the logs for your prediction result. It would display the Churn rate for the index you provide, the level of confidence and the actual label for that index.