Kubeflow: End to End

2 hoursFree

Rate Lab

Introduction

Kubeflow

Kubeflow is a machine learning toolkit for Kubernetes. The project is dedicated to making **deployments** of machine learning (ML) workflows on Kubernetes simple, portable, and scalable. The goal is to provide a straightforward way to deploy best-of-breed open-source systems for ML to diverse infrastructures.
A Kubeflow deployment is:

- **Portable** - Works on any Kubernetes cluster, whether it lives on Google Cloud Platform (GCP), on-premise, or across providers.
- **Scalable** - Can utilize fluctuating resources and is only constrained by the number of resources allocated to the Kubernetes cluster.
- **Composable** - Enhanced with service workers to work offline or on low-quality networks

Kubeflow will let you organize loosely-coupled microservices as a single unit and deploy them to a variety of locations, whether that's a laptop or the cloud. This codelab will walk you through creating your own Kubeflow deployment.

What you'll build

In this lab, you're going to build a web app that summarizes GitHub issues using a trained model. Upon completion, your infrastructure will contain:

- A GKE cluster with standard Kubeflow and Seldon Core installations
- A training job that uses Tensorflow to generate a Keras model
- A serving container that provides predictions
- A UI that uses the trained model to provide summarizations for GitHub issues

What you'll learn

- How to install Kubeflow
- How to run training using the Tensorflow job server to generate a Keras model
- How to serve a trained model with Seldon Core
- How to generate and use predictions from a trained model

What you'll need

- A basic understanding of Kubernetes
- A GitHub account

Setup the environment

Qwiklabs setup

For each lab, you get a new Google Cloud project and set of resources for a fixed time at no cost.

1. Make sure you signed into Qwiklabs using an **incognito window**.

2. Note the lab's access time (for example,

and make sure you can finish in that time block.

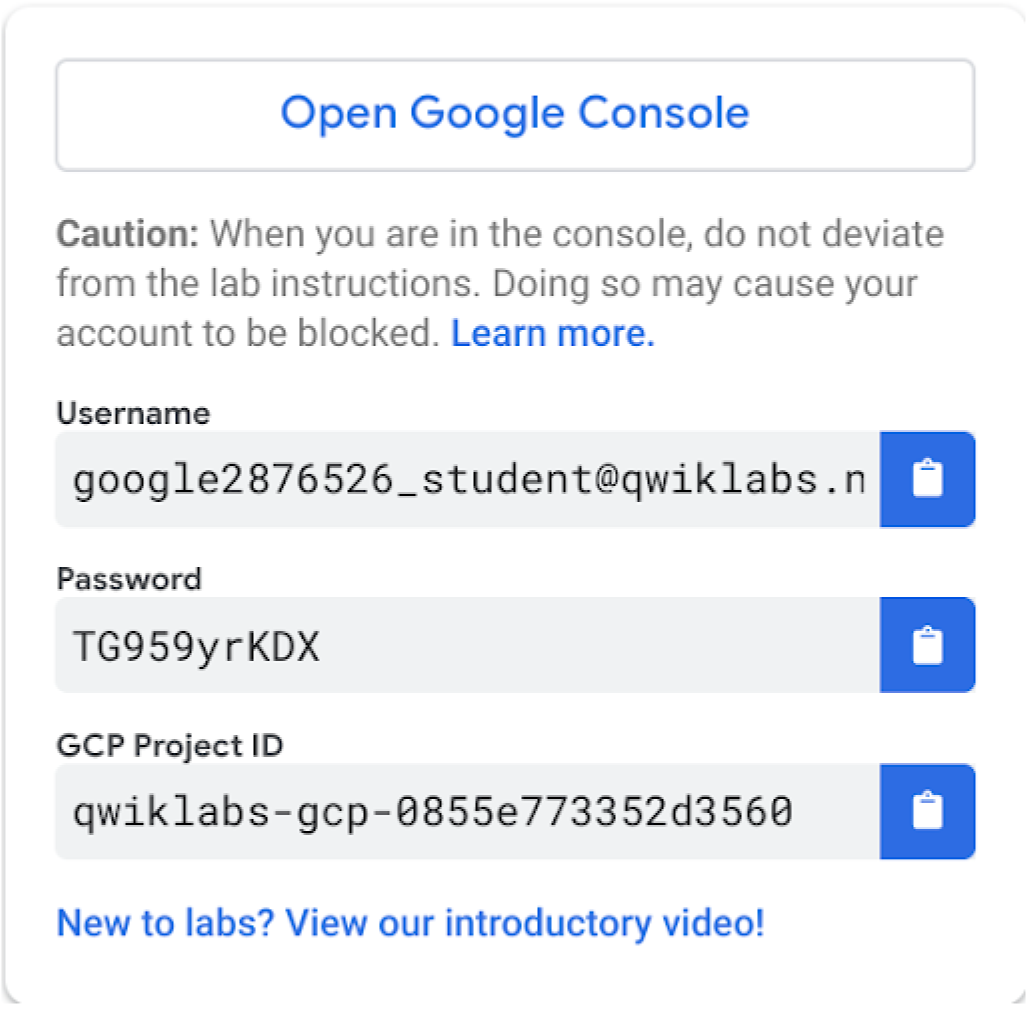
There is no pause feature. You can restart if needed, but you have to start at the beginning.

3. When ready, click

.

4. Note your lab credentials. You will use them to sign in to the Google Cloud

Console.


Open Google Console

Caution: When you are in the console, do not deviate from the lab instructions. Doing so may cause your account to be blocked. Learn more.

Username

google2876526_student@qwiklabs.n

Password

TG959yrKDX

GCP Project ID

qwiklabs-gcp-0855e773352d3560

New to labs? View our introductory video!

5. Click **Open Google Console**.

6. Click **Use another account** and copy/paste credentials for **this** lab into the prompts.

If you use other credentials, you'll get errors or **incur charges**.

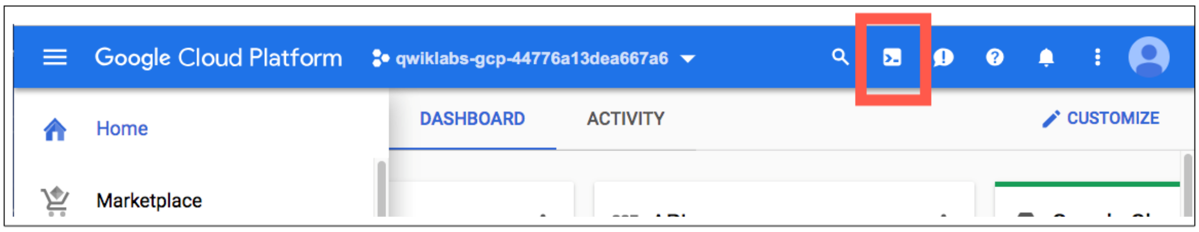
7. Accept the terms and skip the recovery resource page.

Do not click **End Lab** unless you are finished with the lab or want to restart it. This clears your work and removes the project.

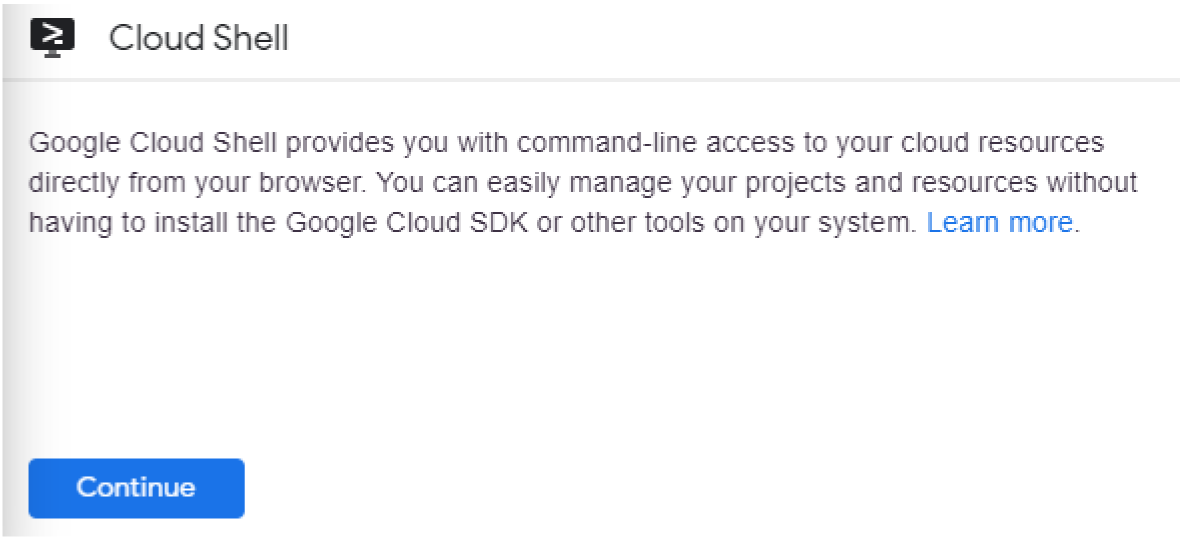
Activate Cloud Shell

Cloud Shell is a virtual machine that is loaded with development tools. It offers a persistent 5GB home directory and runs on the Google Cloud. Cloud Shell provides command-line access to your Google Cloud resources.

In the Cloud Console, in the top right toolbar, click the **Activate Cloud Shell** button.



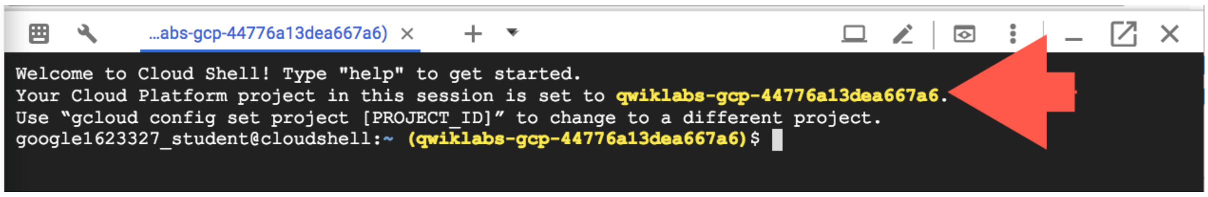
Click **Continue**.



Cloud Shell

Google Cloud Shell provides you with command-line access to your cloud resources directly from your browser. You can easily manage your projects and resources without having to install the Google Cloud SDK or other tools on your system. Learn more.

It takes a few moments to provision and connect to the environment. When you are connected, you are already authenticated, and the project is set to your *PROJECT_ID*. For example:



```
Welcome to Cloud Shell! Type "help" to get started.
Your Cloud Platform project in this session is set to qwiklabs-gcp-44776a13dea667a6.
Use "gcloud config set project [PROJECT_ID]" to change to a different project.
google1623327_student@cloudshell:~ (qwiklabs-gcp-44776a13dea667a6)$
```

`gcloud` is the command-line tool for Google Cloud. It comes pre-installed on Cloud Shell and supports tab-completion.

You can list the active account name with this command:

```
gcloud auth list
content_copy
```

(Output)

```
Credentialed accounts:
 - <myaccount>@<mydomain>.com (active)content_copy
```

(Example output)

```
Credentialed accounts:
 - google1623327_student@qwiklabs.netcontent_copy
```

You can list the project ID with this command:

```
gcloud config list project
content_copy
```

(Output)

```
[core]
project = <project_ID>content_copy
```

(Example output)

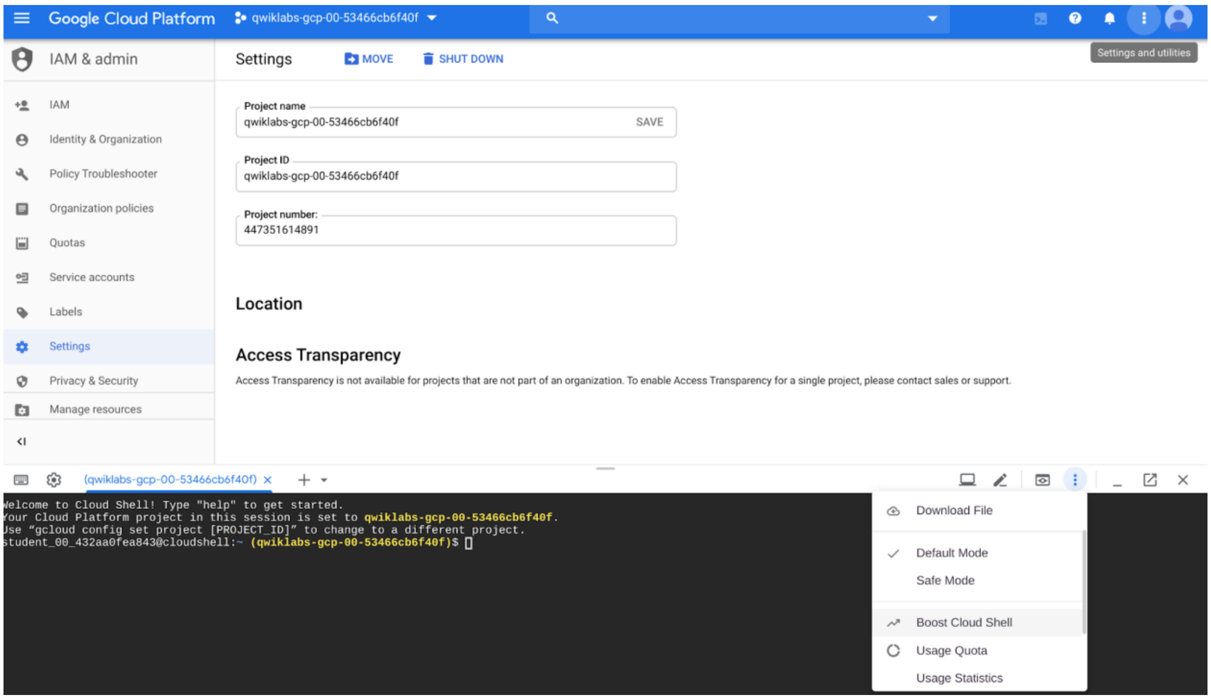
```
[core]
project = qwiklabs-gcp-44776a13dea667a6content_copy
```

For full documentation of `gcloud` see the gcloud command-line tool overview.

Close the main **Navigation Menu** by clicking the three lines at the top left of the screen (hamburger), next to the Google Cloud Platform logo.

Enable Boost Mode

In the Cloud Shell window, click on the three dots icon at the far right. Select **Boost Cloud Shell**, then **Restart Cloud Shell in Boost Mode**. This will provision a larger instance for your Cloud Shell session, resulting in speedier Docker builds.



Project name: qwiklabs-gcp-00-53466cb6f40f

Project ID: qwiklabs-gcp-00-53466cb6f40f

Project number: 447351614891

Access Transparency is not available for projects that are not part of an organization. To enable Access Transparency for a single project, please contact sales or support.

```
Welcome to Cloud Shell! Type "help" to get started.
Your Cloud Platform project in this session is set to qwiklabs-gcp-00-53466cb6f40f.
Use “gcloud config set project [PROJECT_ID]” to change to a different project.
student_00_432aa0fea843@cloudshell:~ (qwiklabs-gcp-00-53466cb6f40f)$
```

Download the project files

The following commands in Cloud Shell to download and unpack an archive of the [Kubeflow examples repo](), which contains all of the official Kubeflow examples:

```
wget https://github.com/kubeflow/examples/archive/v0.2.zip
unzip v0.2.zip
mv examples-0.2 ${HOME}/examplescontent_copy
```

Set your GitHub token

This lab involves the use of many different files obtained from public repos on GitHub. To prevent rate-limiting, setup an access token with no permissions. This is simply to authorize you as an individual rather than anonymous user.

1. Navigate to https://github.com/settings/tokens and generate a new token with no permissions.
2. Save it somewhere safe. If you lose it, you will need to delete and create a new one.

3. Set the GITHUB_TOKEN environment variable:

```
export \
  GITHUB_TOKEN=<token>content_copy
```

Install Ksonnet

Set the correct version and an environment variable:

```
export KS_VER=ks_0.13.1_linux_amd64content_copy
```

Install the binary

Download and unpack the appropriate binary, then add it to your $PATH:

```
wget -O /tmp/$KS_VER.tar.gz
https://github.com/ksonnet/ksonnet/releases/download/v0.13.1/$KS_VER.tar.gz
content_copy
mkdir -p ${HOME}/bin
tar -xvf /tmp/$KS_VER.tar.gz -C ${HOME}/bincontent_copy
export PATH=$PATH:${HOME}/bin/$KS_VERcontent_copy
```

To familiarize yourself with Ksonnet concepts, see this diagram.

Retrieve the project ID

Store the project ID:

```
export PROJECT_ID=$(gcloud config get-value project)content_copy
```

Create a service account

Create a service account with read/write access to storage buckets:

```
export SERVICE_ACCOUNT=github-issue-summarization
export
SERVICE_ACCOUNT_EMAIL=${SERVICE_ACCOUNT}@${PROJECT_ID}.iam.gserviceaccount.
com
gcloud iam service-accounts create ${SERVICE_ACCOUNT} \
  --display-name "GCP Service Account for use with kubeflow examples"

gcloud projects add-iam-policy-binding ${PROJECT_ID} --member \
  serviceAccount:${SERVICE_ACCOUNT_EMAIL} \
  --role=roles/storage.admincontent_copy
```

Generate a credentials file for upload to the cluster:

```
export KEY_FILE=${HOME}/secrets/${SERVICE_ACCOUNT_EMAIL}.json
gcloud iam service-accounts keys create ${KEY_FILE} \
  --iam-account ${SERVICE_ACCOUNT_EMAIL}content_copy
```

Create a storage bucket

Create a Cloud Storage bucket for storing your trained model and issue the “mb” (make bucket) command:

```
export BUCKET=kubeflow-${PROJECT_ID}
gsutil mb -c regional -l us-central1 gs://${BUCKET}content_copy
```

Create a cluster

Create a managed Kubernetes cluster on Kubernetes Engine by running:

```
gcloud container clusters create kubeflow-qwiklab \
  --machine-type n1-standard-4 \
  --zone us-central1-a  \
  --scopes=compute-rw,storage-rw \
  --enable-autorepair \
  --cluster-version=1.14.10content_copy
```

Cluster creation will take a few minutes to complete.

Connect your local environment to the Google Kubernetes Engine (GKE) cluster:

```
gcloud container clusters get-credentials kubeflow-qwiklab --zone us-central1-acontent_copy
```

This configures your `kubectl` context so that you can interact with your cluster. To verify the connection, run the following command:

```
kubectl cluster-infocontent_copy
```

Verify that this IP address matches the IP address corresponding to the Endpoint in your Google Cloud Platform Console or by comparing the Kubernetes master IP is the same as the Master_IP address in the previous step.
To enable the installation of Kubeflow and Seldon components, run the following to create two ClusterRoleBindings, which allows the creation of objects:

```
kubectl create clusterrolebinding default-admin \
  --clusterrole=cluster-admin \
  --user=$(gcloud config get-value account)content_copy
kubectl create clusterrolebinding seldon-admin \
  --clusterrole=cluster-admin \
  --serviceaccount=default:defaultcontent_copy
```

Upload service account credentials:

```
kubectl create secret generic user-gcp-sa \
  --from-file=user-gcp-sa.json="${KEY_FILE}"content_copy
```

Install Kubeflow with Seldon

Ksonnet is a templating framework, which allows us to utilize common object definitions and customize them to our environment. We begin by referencing

Kubeflow templates and apply environment-specific parameters. Once manifests have been generated specifically for our cluster, they can be applied like any other kubernetes object using `kubectl`.

Initialize a ksonnet app

Run these commands to go inside the `github_issue_summarization` directory; then create an new ksonnet app directory, fill it with boilerplate code, and retrieve component files:

```
cd ${HOME}/examples/github_issue_summarization
ks init kubeflow
cd kubeflow
cp ../ks-kubeflow/components/kubeflow-core.jsonnet components
cp ../ks-kubeflow/components/params.libsonnet components
cp ../ks-kubeflow/components/seldon.jsonnet components
cp ../ks-kubeflow/components/tfjob-v1alpha2.* components
cp ../ks-kubeflow/components/ui.* componentscontent_copy
```

Install packages and generate core components

Register the Kubeflow template repository:

```
export VERSION=v0.2.0-rc.1
ks registry add kubeflow
github.com/kubeflow/kubeflow/tree/${VERSION}/kubeflowcontent_copy
```

Install Kubeflow core and Seldon components:

```
ks pkg install kubeflow/core@${VERSION}
ks pkg install kubeflow/tf-serving@${VERSION}
ks pkg install kubeflow/tf-job@${VERSION}
ks pkg install kubeflow/seldon@${VERSION}content_copy
```

Note: If you run into rate-limit errors, be sure your GITHUB_TOKEN environment variable is set properly. See the **Set Your GitHub Token** section above for more details.

Create the environment

Define an environment that references our specific cluster:

```
ks env add gke
ks param set --env gke kubeflow-core \
  cloud "gke"
ks param set --env gke kubeflow-core \
  tfAmbassadorServiceType "LoadBalancer"content_copy
```

Apply the generated manifests to the cluster to create the Kubeflow and Seldon components:

```
ks apply gke -c kubeflow-core -c seldoncontent_copy
```

Your cluster now contains a Kubeflow installation with Seldon with the following components:

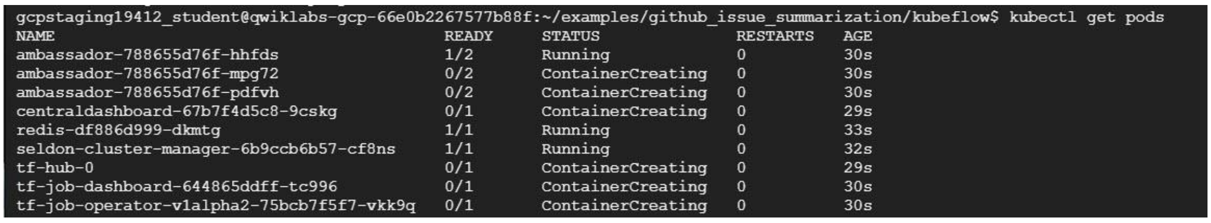
- Reverse HTTP proxy (Ambassador)
- Central dashboard
- Jupyterhub
- TF job dashboard
- TF job operator
- Seldon cluster manager
- Seldon cache

You can view the components by running:

```
kubectl get podscontent_copy
```

Note: Image pulls can take a while. You can expect pods to remain in ContainerCreating status for a few minutes.

Also, If the status of `tf-hub` is showing as `ImagePullBackoff` you can safely ignore it and move forward in the lab.
You should see output similar to this:



```
gcpstaging19412_student@qwiklabs-gcp-66e0b2267577b88f:~/examples/github_issue_summarization/kubeflow$ kubectl get pods
NAME                                          READY     STATUS              RESTARTS   AGE
ambassador-788655d76f-hhfds                   1/2       Running             0          30s
ambassador-788655d76f-mpg72                   0/2       ContainerCreating   0          30s
ambassador-788655d76f-pdfvh                   0/2       ContainerCreating   0          30s
centraldashboard-67b7f4d5c8-9cskg             0/1       ContainerCreating   0          29s
redis-df886d999-dkmtg                         1/1       Running             0          33s
seldon-cluster-manager-6b9ccb6b57-cf8ns       1/1       Running             0          32s
tf-hub-0                                      0/1       ContainerCreating   0          29s
tf-job-dashboard-644865ddff-tc996             0/1       ContainerCreating   0          30s
tf-job-operator-v1alpha2-75bcb7f5f7-vkk9q     0/1       ContainerCreating   0          30s
```

Train a model

In this section, you will create a component that trains a model.

Set the component parameters:

```
cd ${HOME}/examples/github_issue_summarization/kubeflow
ks param set --env gke tfjob-v1alpha2 image "gcr.io/kubeflow-examples/tf-job-issue-summarization:v20180629-v0.1-2-g98ed4b4-dirty-182929"
ks param set --env gke tfjob-v1alpha2 output_model_gcs_bucket "${BUCKET}"content_copy
```

The training component `tfjob-v1alpha2` is now configured to use a pre-built image. If you would prefer to generate your own instead, continue with the Optional create the training image step.

(Optional) Create the training image

Image creation can take 5-10 minutes.

In the `github_issue_summarization` directory, navigate to the folder containing the training code (`notebooks`). From there, issue a `make` command that builds the image and stores it in Google Container Registry (GCR). This places it in a location accessible from inside the cluster.

```
cd ${HOME}/examples/github_issue_summarization/notebooks

make PROJECT=${PROJECT_ID} push
```

Once the image has been built and stored in GCR, update the component parameter with a link that points to the custom image:

```
export TAG=$(gcloud container images list-tags \

gcr.io/${PROJECT_ID}/tf-job-issue-summarization \

--limit=1 \
--format='get(tags)')

cd ${HOME}/examples/github_issue_summarization/kubeflow

ks param set --env gke tfjob-v1alpha2 image
"gcr.io/${PROJECT_ID}/tf-job-issue-summarization:${TAG}"
```

Launch training

Apply the component manifests to the cluster:

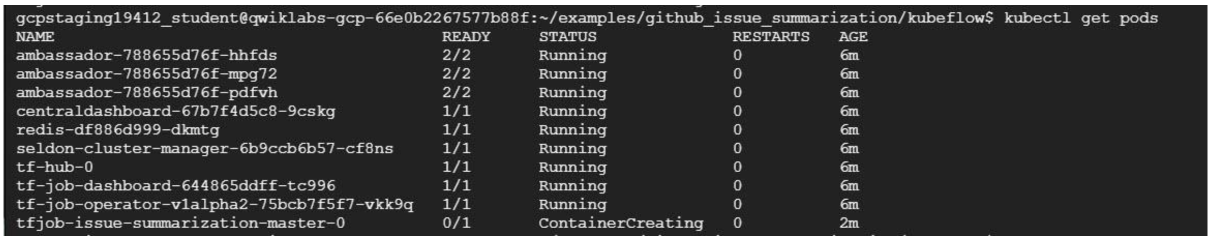
```
ks apply gke -c tfjob-v1alpha2content_copy
```

View the running job

View the resulting pods:

```
kubectl get podscontent_copy
```

Your cluster state should look similar to this:



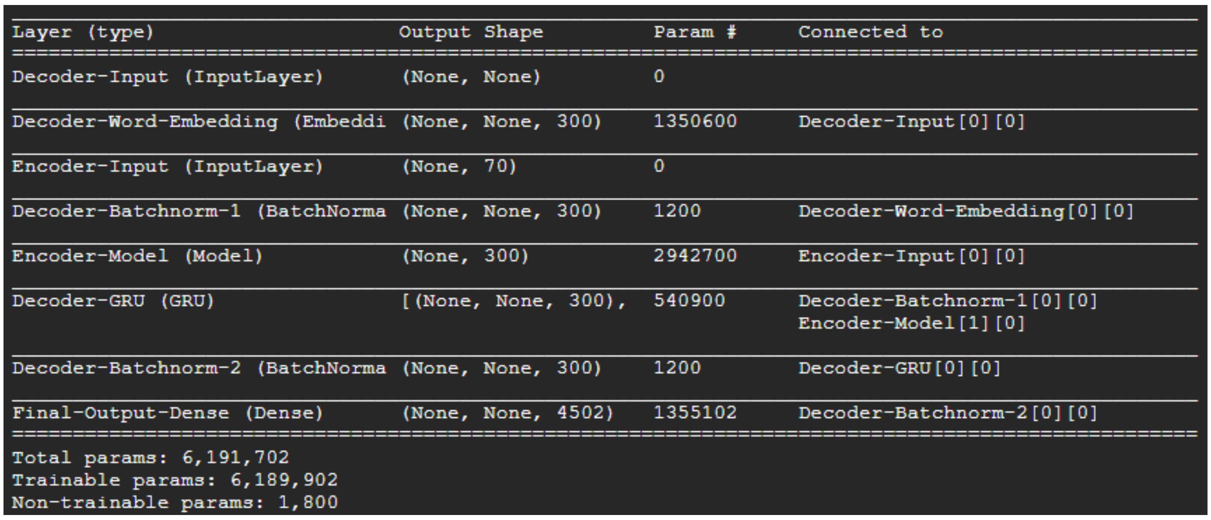
```
gcpstaging19412_student@qwiklabs-gcp-66e0b2267577b88f:~/examples/github_issue_summarization/kubeflow$ kubectl get pods
NAME                                        READY     STATUS              RESTARTS   AGE
ambassador-788655d76f-hhfds                 2/2       Running             0          6m
ambassador-788655d76f-mpg72                 2/2       Running             0          6m
ambassador-788655d76f-pdfvh                 2/2       Running             0          6m
centraldashboard-67b7f4d5c8-9cskg           1/1       Running             0          6m
redis-df886d999-dkmtg                       1/1       Running             0          6m
seldon-cluster-manager-6b9ccb6b57-cf8ns     1/1       Running             0          6m
tf-hub-0                                    1/1       Running             0          6m
tf-job-dashboard-644865ddff-tc996           1/1       Running             0          6m
tf-job-operator-v1alpha2-75bcb7f5f7-vkk9q   1/1       Running             0          6m
tfjob-issue-summarization-master-0          0/1       ContainerCreating   0          2m
```

It can take a few minutes to pull the image and start the container.

Once the new pod is running, tail the logs:

```
kubectl logs -f \
  $(kubectl get pods -ltf_job_key=tfjob-issue-summarization -
o=jsonpath='{.items[0].metadata.name}')content_copy
```

Inside the pod, you will see the download of source data (`github-issues.zip`) before training begins. Continue tailing the logs until the pod exits on its own and you find yourself back at the command prompt. When you see the command prompt, continue with the next step.

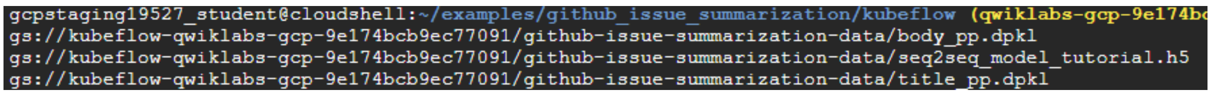


```
Layer (type)                    Output Shape          Param #     Connected to
==================================================================================================
Decoder-Input (InputLayer)      (None, None)          0

Decoder-Word-Embedding (Embeddi (None, None, 300)     1350600     Decoder-Input[0][0]

Encoder-Input (InputLayer)      (None, 70)            0

Decoder-Batchnorm-1 (BatchNorma (None, None, 300)     1200        Decoder-Word-Embedding[0][0]

Encoder-Model (Model)           (None, 300)           2942700     Encoder-Input[0][0]

Decoder-GRU (GRU)               [(None, None, 300),   540900      Decoder-Batchnorm-1[0][0]
                                                                  Encoder-Model[1][0]

Decoder-Batchnorm-2 (BatchNorma (None, None, 300)     1200        Decoder-GRU[0][0]

Final-Output-Dense (Dense)      (None, None, 4502)    1355102     Decoder-Batchnorm-2[0][0]
==================================================================================================
Total params: 6,191,702
Trainable params: 6,189,902
Non-trainable params: 1,800
```

To verify that training completed successfully, check to make sure all three model files were uploaded to your GCS bucket:

```
gsutil ls gs://${BUCKET}/github-issue-summarization-datacontent_copy
```

You should see something like this:



```
gcpstaging19527_student@cloudshell:~/examples/github_issue_summarization/kubeflow (qwiklabs-gcp-9e174bc
gs://kubeflow-qwiklabs-gcp-9e174bcb9ec77091/github-issue-summarization-data/body_pp.dpkl
gs://kubeflow-qwiklabs-gcp-9e174bcb9ec77091/github-issue-summarization-data/seq2seq_model_tutorial.h5
gs://kubeflow-qwiklabs-gcp-9e174bcb9ec77091/github-issue-summarization-data/title_pp.dpkl
```

Serve the trained model

In this section, you will create a component that serves a trained model.

Set component parameters:

```
export SERVING_IMAGE=gcr.io/kubeflow-examples/issue-summarization-model:v20180629-v0.1-2-g98ed4b4-dirty-182929content_copy
```

Create the serving image

The serving component is configured to run a pre-built image, to save you some time. If you would prefer to serve the model you created in the previous step, you can generate your own by continuing with *Optional image creation* step. Otherwise, continue with the *Create the serving component* step.

(Optional) image creation

Download the trained model files

Retrieve the trained model files that were generated in the previous step:

```
cd ${HOME}/examples/github_issue_summarization/notebooksgsutil cp
gs://${BUCKET}/github-issue-summarization-data/* .
```

Generate image build files

Using a Seldon wrapper, generate files for building a serving image. This command creates a build directory and image creation script:

```
docker run -v $(pwd):/my_model seldonio/core-python-wrapper:0.7 \

/my_model IssueSummarization 0.1 gcr.io \

--base-image=python:3.6 \

--image-name=${PROJECT_ID}/issue-summarization-model
```

Generate a serving image

Using the files created by the wrapper, generate a serving image and store it in GCR:

```
cd ${HOME}/examples/github_issue_summarization/notebooks/build

./build_image.sh

gcloud docker -- push gcr.io/${PROJECT_ID}/issue-summarization-
model:0.1

export SERVING_IMAGE=gcr.io/${PROJECT_ID}/issue-summarization-
model:0.1
```

Create the serving component

This serving component is configured to run a pre-built image. Using a Seldon ksonnet template, generate the serving component.

Navigate back to the ksonnet app directory and issue the following command:

```
cd ${HOME}/examples/github_issue_summarization/kubeflow
ks generate seldon-serve-simple issue-summarization-model \
  --name=issue-summarization \
  --image=${SERVING_IMAGE} \
  --replicas=2content_copy
```

Launch serving

Apply the component manifests to the cluster:

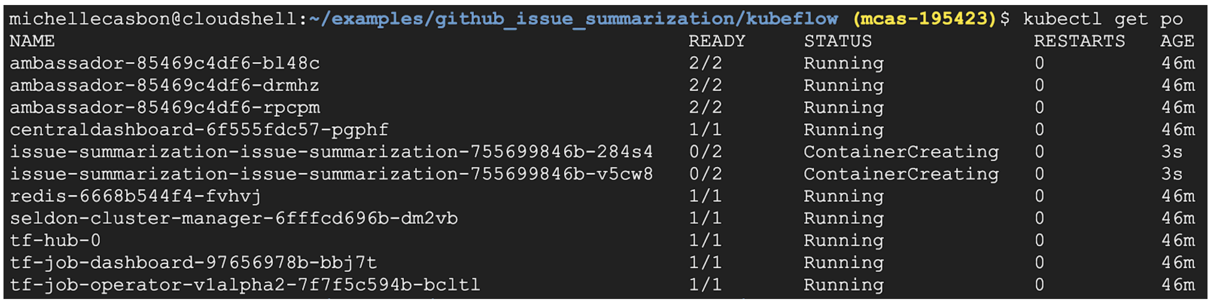
```
ks apply gke -c issue-summarization-modelcontent_copy
```

View the running pods

You will see several new pods appear:

```
kubectl get podscontent_copy
```

Your cluster state should look similar to this:



```
michellecasbon@cloudshell:~/examples/github_issue_summarization/kubeflow (mcas-195423)$ kubectl get po
NAME                                                     READY     STATUS              RESTARTS   AGE
ambassador-85469c4df6-bl48c                              2/2       Running             0          46m
ambassador-85469c4df6-drmhz                              2/2       Running             0          46m
ambassador-85469c4df6-rpcpm                              2/2       Running             0          46m
centraldashboard-6f555fdc57-pgphf                        1/1       Running             0          46m
issue-summarization-issue-summarization-755699846b-284s4  0/2       ContainerCreating   0          3s
issue-summarization-issue-summarization-755699846b-v5cw8  0/2       ContainerCreating   0          3s
redis-6668b544f4-fvhvj                                   1/1       Running             0          46m
seldon-cluster-manager-6fffcd696b-dm2vb                  1/1       Running             0          46m
tf-hub-0                                                 1/1       Running             0          46m
tf-job-dashboard-97656978b-bbj7t                         1/1       Running             0          46m
tf-job-operator-v1alpha2-7f7f5c594b-bcltl                1/1       Running             0          46m
```

Wait a minute or two and re-run the previous command. Once the pods are running, tail the logs for one of the serving containers to verify that it is running on port 9000:

```
kubectl logs \
  $(kubectl get pods \
    -lseldon-app=issue-summarization \
    -o=jsonpath='{.items[0].metadata.name}') \
  issue-summarizationcontent_copy
```
Press **Ctrl** + **C** to return to the command line.

Add a UI

Set parameter values

```
cd ${HOME}/examples/github_issue_summarization/kubeflow
ks param set --env gke ui image "gcr.io/kubeflow-examples/issue-
summarization-ui:v20180629-v0.1-2-g98ed4b4-dirty-182929"
ks param set --env gke ui githubToken ${GITHUB_TOKEN}
ks param set --env gke ui modelUrl "http://issue-
summarization.default.svc.cluster.local:8000/api/v0.1/predictions"
ks param set --env gke ui serviceType "LoadBalancer"content_copy
```

(Optional) Create the UI image

The UI component is now configured to use a pre-built image. If you would prefer to generate your own instead, continue with this step.

Note: Image creation can take 5-10 minutes. This step is optional. Alternatively, skip directly to the **Launch the UI** section below.
Switch to the docker directory and build the image for the UI:

```
cd ${HOME}/examples/github_issue_summarization/docker
docker build -t gcr.io/${PROJECT_ID}/issue-summarization-ui:latest
.content_copy
```
After it has been successfully built, store it in GCR:

```
gcloud docker -- push gcr.io/${PROJECT_ID}/issue-summarization-
ui:latestcontent_copy
```
Update the component parameter with a link that points to the custom image:

```
cd ${HOME}/examples/github_issue_summarization/kubeflow
```

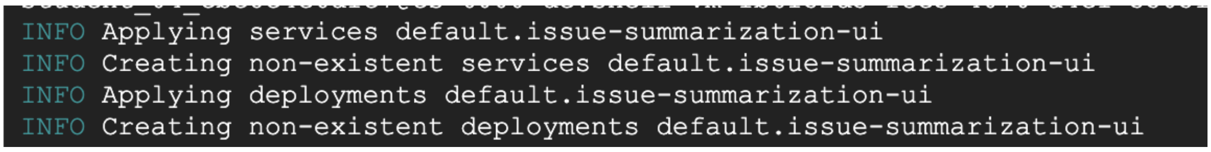
```
ks param set --env gke ui image gcr.io/${PROJECT_ID}/issue-summarization-ui:latestcontent_copy
```

Launch the UI

Apply the component manifests to the cluster:

```
ks apply gke -c uicontent_copy
```

You should see something like this:



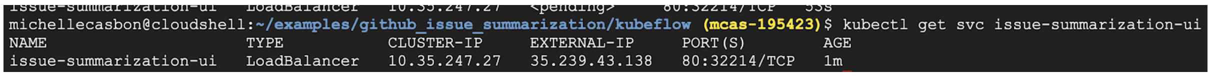
```
INFO Applying services default.issue-summarization-ui
INFO Creating non-existent services default.issue-summarization-ui
INFO Applying deployments default.issue-summarization-ui
INFO Creating non-existent deployments default.issue-summarization-ui
```

View the UI

To view the UI, get the external IP address:

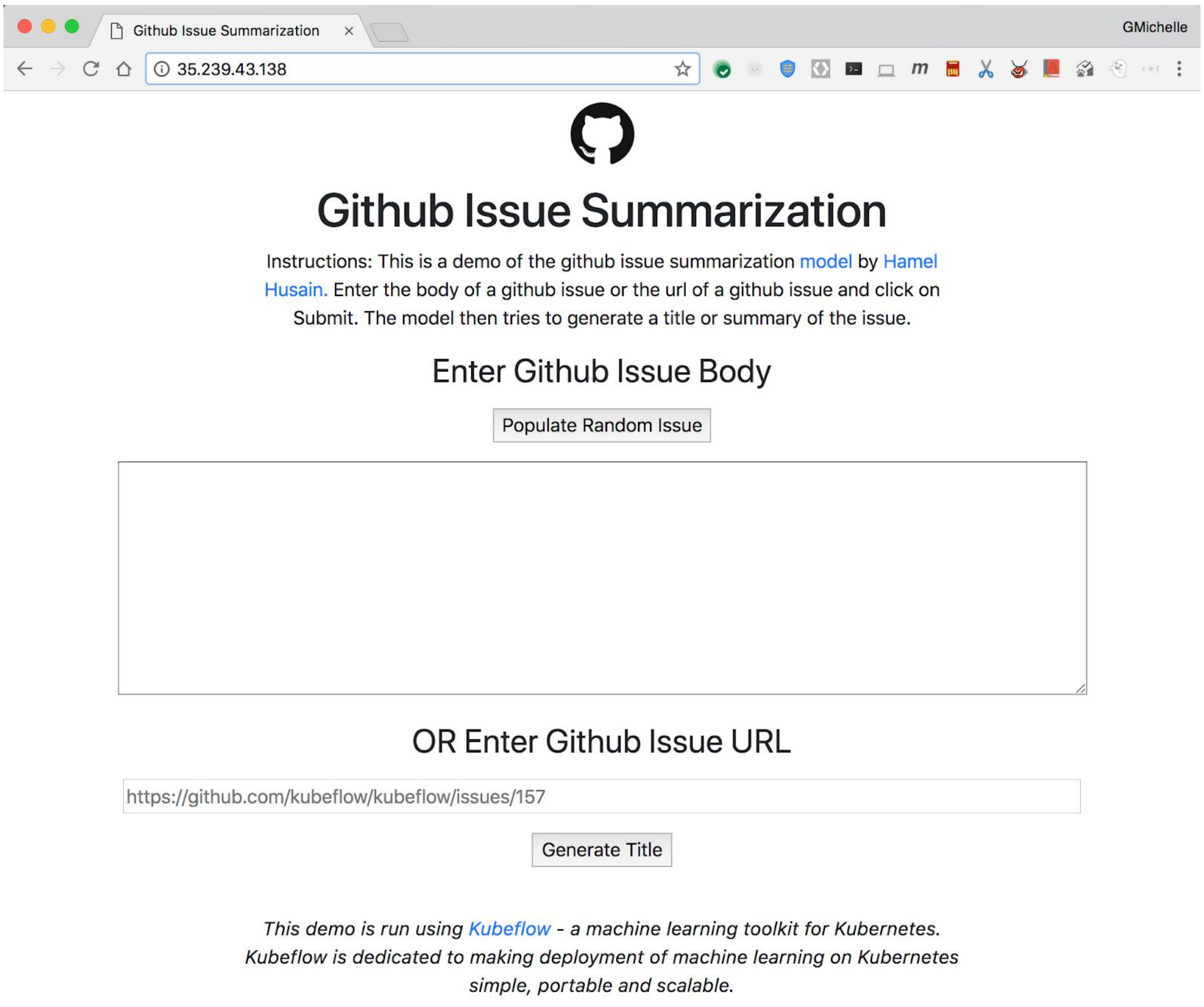
```
kubectl get svc issue-summarization-uicontent_copy
```

Wait until the external IP address has been populated. Re-run the command until it appears. Copy the External-IP address.



```
michellecasbon@cloudshell:~/examples/github_issue_summarization/kubeflow (mcas-195423)$ kubectl get svc issue-summarization-ui
NAME                     TYPE           CLUSTER-IP     EXTERNAL-IP     PORT(S)        AGE
issue-summarization-ui   LoadBalancer   10.35.247.27   35.239.43.138   80:32214/TCP   1m
```

In a browser, paste the `EXTERNAL-IP` to view the results. You should see something like this:


Github Issue Summarization

35.239.43.138

Github Issue Summarization

Instructions: This is a demo of the github issue summarization model by Hamel Husain. Enter the body of a github issue or the url of a github issue and click on Submit. The model then tries to generate a title or summary of the issue.

Enter Github Issue Body

OR Enter Github Issue URL

https://github.com/kubeflow/kubeflow/issues/157

This demo is run using Kubeflow - a machine learning toolkit for Kubernetes. Kubeflow is dedicated to making deployment of machine learning on Kubernetes simple, portable and scalable.

Click the **Populate Random Issue** button to fill in the large text box with a random issue summary. Then click the **Generate Title** button to view the machine generated title produced by your trained model. Click the button a couple of times to give yourself some more data to look at in the next step.

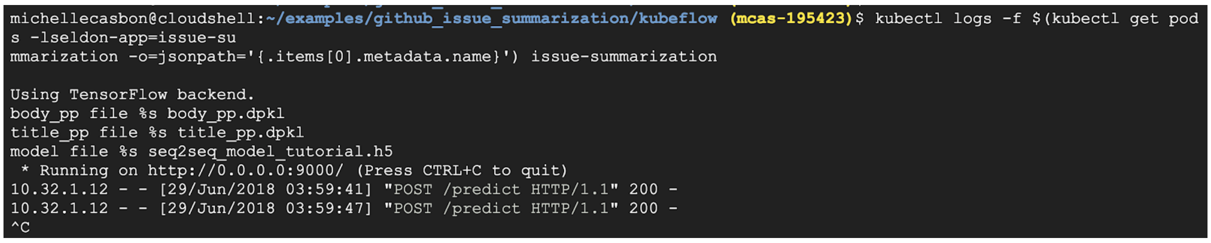
View serving container logs

In Cloud Shell, tail the logs of one of the serving containers to verify that it is receiving a request from the UI and providing a prediction in response:

```
kubectl logs -f \
  $(kubectl get pods \
    -lseldon-app=issue-summarization \
    -o=jsonpath='{.items[0].metadata.name}') \
  issue-summarizationcontent_copy
```

Back in the UI, press the **Generate Title** button a few times to view the POST request in Cloud Shell. Since there are two serving containers, you might need to try a few times before you see the log entry.

Press **Ctrl+C** to return to the command prompt.



```
michellecasbon@cloudshell:~/examples/github_issue_summarization/kubeflow (mcas-195423)$ kubectl logs -f $(kubectl get pod
s -lseldon-app=issue-su
mmarization -o=jsonpath='{.items[0].metadata.name}') issue-summarization

Using TensorFlow backend.
body_pp file %s body_pp.dpkl
title_pp file %s title_pp.dpkl
model file %s seq2seq_model_tutorial.h5
 * Running on http://0.0.0.0:9000/ (Press CTRL+C to quit)
10.32.1.12 - - [29/Jun/2018 03:59:41] "POST /predict HTTP/1.1" 200 -
10.32.1.12 - - [29/Jun/2018 03:59:47] "POST /predict HTTP/1.1" 200 -
^C
```

Clean up

Remove GitHub token

Navigate to https://github.com/settings/tokens and remove the generated token.

End your lab

When you have completed your lab, click **End Lab**. Qwiklabs removes the resources you’ve used and cleans the account for you.

You will be given an opportunity to rate the lab experience. Select the applicable number of stars, type a comment, and then click **Submit**.

The number of stars indicates the following:

- 1 star = Very dissatisfied
- 2 stars = Dissatisfied
- 3 stars = Neutral
- 4 stars = Satisfied
- 5 stars = Very satisfied

You can close the dialog box if you don't want to provide feedback.

For feedback, suggestions, or corrections, please use the **Support** tab.