Start Lab 01:30:00

Implementing Canary Releases of TensorFlow Model Deployments with Kubernetes and Istio

–/100

1 hour 30 minutes Free

GSP778

Google Cloud Self-Paced Labs

Overview

Istio is an open source framework for connecting, securing, and managing microservices, including services running on Kubernetes Engine. It lets you create a mesh of deployed services with load balancing, service-to-service authentication, monitoring, and more, without requiring any changes in service code.

This lab shows you how to use Istio on Google Kubernetes Engine (GKE) and TensorFlow Serving to create canary deployments of TensorFlow machine learning models.

Objectives

In this lab, you will learn how to:

- Prepare a GKE cluster with the Istio add-on for TensorFlow Serving.
- Create a canary release of a TensorFlow model deployment.
- Configure various traffic splitting strategies.

Prerequisites

To successfully complete the lab you need to have a solid understanding of how to save and load TensorFlow models and a basic familiarity with Kubernetes and Istio concepts and architecture. Before proceeding with the lab we recommend reviewing the following resources:

- Using the SavedModel format
- Kubernetes Overview
- Istio Concepts

Lab scenario

In the lab, you will walk through a canary deployment of two versions of the **ResNet** model. The idea behind a canary deployment is to introduce a new version of a service (model deployment) by first testing it using a small percentage of user traffic, and then if the new model meets the set requirements, redirect, possibly gradually in increments, the traffic from the old version to the new one.

In its simplest form, the traffic sent to the canary version is a randomly selected percentage of requests send to a common endpoint that exposes both models. The more sophisticated traffic splitting schemas can also be used. For example, the traffic can be split based on the orginating region, user or user group, or other properties of the request. When the traffic is split based on well defined groups of originators, the canary deployment can be used as a foundation of A/B testing.

You will use TensorFlow Serving to deploy two versions of **ResNet**: **ResNet50** and **ResNet101**. Both models expose the same interface (inputs and outputs). **ResNet50** will be a simulated production model. **ResNet101** will be a new, canary release.

TensorFlow Serving is a flexible, high-performance serving system for machine learning models, designed for production environments. TensorFlow Serving makes it easy to

deploy new algorithms and experiments, while keeping the same server architecture and APIs. TensorFlow Serving provides out-of-the-box integration with TensorFlow models, but can be easily extended to serve other types of models and data. TensorFlow Serving can be run in a docker container and deployed and managed by Kubernetes. In the lab, you will deploy TensorFlow Serving as a Kubernetes Deployment on Google Cloud Kubernetes Engine (GKE).

Istio will be used to configure transparent traffic splitting between both deployments. Both models will be exposed through the same external endpoint. You will use Istio's traffic management features to experiment with various traffic splitting strategies.

Summary of the tasks performed during the lab:

- Creating a GKE cluster with Istio add-on
- Deploying ResNet models using TensorFlow Serving
- Configuring Istio Ingress gateway
- Configuring Istio virtual services and destination rules
- Configuring weight based routing
- Configuring content based routing

Setup and requirements

Before you click the Start Lab button

Read these instructions. Labs are timed and you cannot pause them. The timer, which starts when you click **Start Lab**, shows how long Google Cloud resources will be made available to you.

This hands-on lab lets you do the lab activities yourself in a real cloud environment, not in a simulation or demo environment. It does so by giving you new, temporary credentials that you use to sign in and access Google Cloud for the duration of the lab.

What you need

To complete this lab, you need:

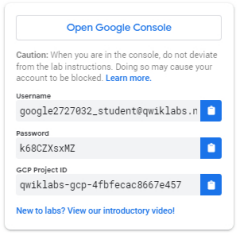
- Access to a standard internet browser (Chrome browser recommended).
- Time to complete the lab.

Note: If you already have your own personal Google Cloud account or project, do not use it for this lab.

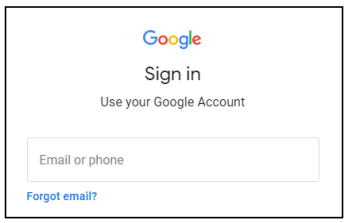
Note: If you are using a Chrome OS device, open an Incognito window to run this lab.

How to start your lab and sign in to the Google Cloud Console

1. Click the **Start Lab** button. If you need to pay for the lab, a pop-up opens for you to select your payment method. On the left is a panel populated with the temporary credentials that you must use for this lab.

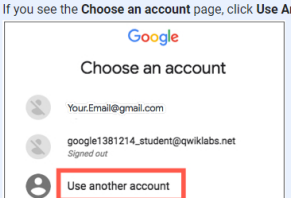


2. Copy the username, and then click **Open Google Console**. The lab spins up resources, and then opens another tab that shows the **Sign in** page.



Tip: Open the tabs in separate windows, side-by-side.

If you see the **Choose an account** page, click **Use Another Account**.



3. In the **Sign in** page, paste the username that you copied from the left panel. Then copy and paste the password.

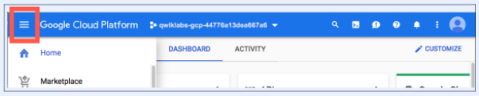
   Important: You must use the credentials from the left panel. Do not use your Google Cloud Training credentials. If you have your own Google Cloud account, do not use it for this lab (avoids incurring charges).

4. Click through the subsequent pages:

   - Accept the terms and conditions.
   - Do not add recovery options or two-factor authentication (because this is a temporary account).
   - Do not sign up for free trials.

After a few moments, the Cloud Console opens in this tab.

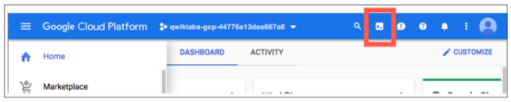
Note: You can view the menu with a list of Google Cloud Products and Services by clicking the **Navigation menu** at the top-left.



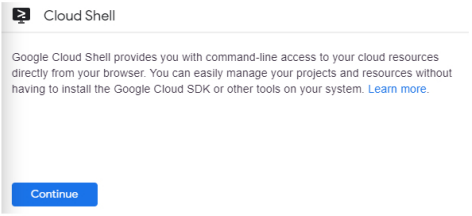
Activate Cloud Shell

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.

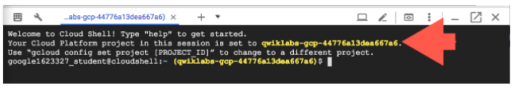
In the Cloud Console, in the top right toolbar, click the **Activate Cloud Shell** button.



Click **Continue**.



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:



`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
```

(Output)

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

(Example output)

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

You can list the project ID with this command:

```
gcloud config list project
```

(Output)

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

(Example output)

```
[core]
project = qwiklabs-gcp-44776a13dea667a6
```

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

Lab tasks

You will use **Cloud Shell** and **Cloud Console** for all of the tasks in the lab. Some tasks require you to edit text files. You can use any of the classic command line text editors pre-installed in **Cloud Shell**, including *vim*, *emacs*, or *nano*. You can also use the built-in Cloud Shell Editor.

Before proceeding, make sure that you completed the **Activate Cloud Shell** step in the **Qwiklabs setup** instructions and your **Cloud Shell** is open and ready.

Getting lab files

Start by getting the lab files from GitHub:

```
cd
SRC_REPO=https://github.com/GoogleCloudPlatform/mlops-on-gcp
kpt pkg get $SRC_REPO/workshops/mlep-qwiklabs/tfserving-canary-gke tfserving-canary
cd tfserving-canary
```

Creating a GKE cluster

Set the default compute zone and a couple of environment variables to store your project id and cluster name:

```
gcloud config set compute/zone us-central1-f
PROJECT_ID=$(gcloud config get-value project)
CLUSTER_NAME=cluster-1
```

To create a GKE cluster with Istio add-on, execute the below command. It may take a few minutes to complete.

```
gcloud beta container clusters create $CLUSTER_NAME \
  --project=$PROJECT_ID \
  --addons=Istio \
  --istio-config=auth=MTLS_PERMISSIVE \
  --cluster-version=latest \
  --machine-type=n1-standard-4 \
  --num-nodes=3
```

After the command completes you should see the ouput similar to one below:

Get the credentials for you new cluster so you can interact with it using `kubectl`.

```
gcloud container clusters get-credentials $CLUSTER_NAME
```

Ensure the following Kubernetes services are deployed: `istio-citadel`, `istio-galley`, `istio-pilot`, `istio-ingressgateway`, `istio-policy`, `istio-sidecar-injector`, and `istio-telemetry`

```
kubectl get service -n istio-system
```

Ensure that the corresponding Kubernetes Pods are deployed and all containers are up and running: `istio-pilot-*`, `istio-policy-*`, `istio-telemetry-*`, `istio-galley-*`, `istio-ingressgateway-*`, `istio-sidecar-injector-*`, and `istio-citadel-*`

```
kubectl get pods -n istio-system
```

Configuring automatic sidecar injection

In order to take advantage of all of Istio's features, pods in the Istio mesh must be running an Istio sidecar proxy.

There are two ways of injecting the Istio sidecar into a pod: manually using the istioctl command or by enabling automatic Istio sidecar injection in the pod's namespace.

Manual injection directly modifies configuration, like deployments, and injects the proxy configuration into it. When enabled in a pod's namespace, automatic injection injects the

proxy configuration at pod creation time using an admission controller.

In this lab, the automatic sidecar injection is used. To configure automatics sidecar injection execute the following command.

```
kubectl label namespace default istio-injection=enabled
```

Click *Check my progress* to verify the objective

Create a GKE cluster and configure sidecar injection

Check my progress

Deploying ResNet50

As described in the lab scenario overview section, the **ResNet50** model will represent a current production model and the **ResNet101** will be a canary release.

The pretrained models in the `SavedModel` format have been uploaded to a public Cloud Storage location.

You will first download the model files to a storage bucket in your project. Since storage buckets are a global resource in Google Cloud you have to use a unique bucket name. For the purpose of this lab, you can use your project id as a name prefix.

To create a storage bucket in your project:

```
export MODEL_BUCKET=${PROJECT_ID}-bucket
gsutil mb gs://${MODEL_BUCKET}
```

After the bucket has been created, copy the model files:

```
gsutil cp -r gs://workshop-datasets/models/resnet_101
gs://${MODEL_BUCKET}
gsutil cp -r gs://workshop-datasets/models/resnet_50
gs://${MODEL_BUCKET}
```

Click *Check my progress* to verify the objective

Deploy ResNet50

Check my progress

You are now ready to create a TensorFlow Serving deployment and configure it to serve the ResNet50 model. You will deploy TF Serving in three steps:

1. First you will create a Kubernetes ConfigMap that points to the location of the ResNet50 model in your storage bucket
2. Then, you will create a Kubernetes Deployment using a standard TensorFlow Serving image from **Docker Hub**.
3. When the deployment is ready, you will create a Kubernetes Service to provide an interface to the model deployment.

Creating ConfigMap

Use your preferred command line editor or **Cloud Shell Editor** to update the `MODEL_NAME` field in the `tf-serving/configmap-resnet50.yaml` file to reference your bucket. Recall that the bucket name was stored in the $MODEL_BUCKET environment variable:

```
echo $MODEL_BUCKET
```

After the update the `configmap-resnet50.yaml` should look similar to the one below:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: tfserving-configs
data:
  MODEL_NAME: image_classifier
  MODEL_PATH: gs://qwiklabs-gcp-03-4b91a600a7a2-bucket/resnet_50
```

Using `kubectl` create the *ConfigMap*:

```
kubectl apply -f tf-serving/configmap-resnet50.yaml
```

Click *Check my progress* to verify the objective

Create ConfigMap

Check my progress

Creating TensorFlow Serving deployment of the ResNet50 model.

Inspect the manifest for the TensorFlow Serving deployment.

```
cat tf-serving/deployment-resnet50.yaml
```

Notice that the deployment is annotated with two labels: `app: image-classifier` and `version: resnet50`. These labels will be the key when configuring Istio traffic routing.

```
...
apiVersion: apps/v1
kind: Deployment
metadata:
  name: image-classifier-resnet50
  namespace: default
  labels:
    app: image-classifier
    version: resnet50
...
```

Create the deployment

```
kubectl apply -f tf-serving/deployment-resnet50.yaml
```

It may take a few minutes before the deployment is ready.

To check the status of the deployment:

```
kubectl get deployments
```

Wait till the *READY* column in the output of the previous command changes to `1/1`.

```
student_01_b5243c1b8ed3@cloudshell:~/tfserving-canary (qwiklabs-gcp-01-c1aec271a548)$ kubectl get deployments
NAME                          READY   UP-TO-DATE   AVAILABLE   AGE
image-classifier-resnet50     1/1     1            1           83s
```

You will now configure a Kubernetes service that exposes a stable IP address and DNS name.

Inspect the service manifest is in the `tf-serving/service.yaml` file.

```
apiVersion: v1
kind: Service
metadata:
  name: image-classifier
  namespace: default
  labels:
    app: image-classifier
    service: image-classifier
spec:
  type: ClusterIP
  ports:
  - port: 8500
    protocol: TCP
    name: tf-serving-grpc
  - port: 8501
    protocol: TCP
    name: tf-serving-http
  selector:
    app: image-classifier
```

The `selector` field refers to the `app: image-classifier` label. What it means is that the service will load balance across all pods annotated with this label. At this point these are the pods comprising the ResNet50 deployment. The service type is `ClusterIP`. The IP address exposed by the service is only visible within the cluster.

To create the service execute the following command:

```
kubectl apply -f tf-serving/service.yaml
```

Click *Check my progress* to verify the objective

Create TensorFlow Serving deployment for ResNet50 model

Check my progress

Configuring Istio Ingress gateway

You use an Istio Ingress gateway to manage inbound and outbound traffic for your mesh, letting you specify which traffic you want to enter and leave the mesh. Unlike other mechanisms for controlling traffic entering your systems, such as the Kubernetes Ingress APIs, Istio gateways let you use the full power and flexibility of Istio's traffic routing.

The gateway manifest in the `tf-serving/gateway.yaml` file configures the gateway to open port 80 for the `HTTP` traffic.

```
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: image-classifier-gateway
spec:
  selector:
    istio: ingressgateway
  servers:
  - port:
      number: 80
      name: http
      protocol: HTTP
    hosts:
    - "*"
```

To create the gateway execute the below command:

To create the gateway execute the below command:

```
kubectl apply -f tf-serving/gateway.yaml
```

At this point the gateway is running but your ResNet50 deployment is still not accessible from the outside of the cluster. You need to configure a virtual service.

Virtual services, along with destination rules are the key building blocks of Istio's traffic routing functionality. A virtual service lets you configure how requests are routed to a service within an Istio service mesh. Each virtual service consists of a set of routing rules that are evaluated in order, letting Istio match each given request to the virtual service to a specific real destination within the mesh.

You will start by configuring a virtual service that forwards all requests sent through the gateway on port 80 to the `image-classifier` service on port 8501. Recall that the `image-classifier` service is configured to load balance between pods annotated with the `app: image-classifer` label.

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: image-classifier
spec:
  hosts:
  - "*"
  gateways:
  - image-classifier-gateway
  http:
  - route:
    - destination:
        host: image-classifier
        port:
          number: 8501
```

As a result all requests send to the gateway will be forwarded to the ResNet50 deployment.

To create the virtual service execute the below command:

```
kubectl apply -f tf-serving/virtualservice.yaml
```

Click *Check my progress* to verify the objective

Configure Istio Ingress gateway

Check my progress

Testing access to the ResNet50 model

The ResNet50 deployment can now be accessed from outside of the cluster. You can test it by submitting a request using the `curl` command.

In the `payloads` folder you can find a sample request body (`request-body.json`) formatted to conform to the TensorFlow Serving REST API. It contains the picture of Grace Hopper.

To send the request you need an external IP address and a port exposed by the Istio gateway.

Set the ingress IP and port:

```
export INGRESS_HOST=$(kubectl -n istio-system get service istio-
ingressgateway -o
jsonpath='{.status.loadBalancer.ingress[0].ip}')
export INGRESS_PORT=$(kubectl -n istio-system get service istio-
ingressgateway -o jsonpath='{.spec.ports[?
(@.name=="http2")].port}')
```

Set the gateway URL

```
export GATEWAY_URL=$INGRESS_HOST:$INGRESS_PORT
```

```
export GATEWAY_URL=$INGRESS_HOST:$INGRESS_PORT
echo $GATEWAY_URL
```

The TensorFlow Serving REST prediction endpoint is accessible at:

```
http://host:port/v1/models/${MODEL_NAME}
[/versions/${VERSION}|/labels/${LABEL}]:predict
```

where /versions/$${VERSION} or /labels/$${LABEL} are optional. If omitted the latest version is used.

In our deployment the endpoint is at:

```
http://$GATEWAY_URL/v1/models/image_classifier:predict.
```

To send the request to the model invoke the below command.

```
curl -d @payloads/request-body.json -X POST
http://$GATEWAY_URL/v1/models/image_classifier:predict
```

The response returned by the model includes the list of 5 most likely labels with the associated probabilities. The response should look similar to the one below:

Note that the model ranked the `military uniform` label with the probability of around 45%.

Deploying ResNet101 as a canary release

You will now deploy the ResNet101 model as a canary release. As in the case of the ResNet50 model, ResNet101 will be deployed as a Kubernetes Deployment with annotated with the `app` label set to `image-classifier`. Recall that the `image-classifier` Kubernetes service is configured to load balance between pods that contain the `app` label set to this value and that the Istio virtual service is configured to forward all the traffic from the Istio Ingress gateway to the `image-classifier` service.

If you deployed ResNet101 without any changes to the Istio virtual service the incoming traffic would be load balanced across pods from both deployments using a round-robin strategy. To have a better control over which requests go to which deployment you need to configure a destination rule and modify the virtual service.

Along with virtual services, destination rules are a key part of Istio's traffic routing functionality. Destinations rules allow much finer control over traffic destinations. In particular, you can use destination rules to specify name service subsets, such as grouping all a given service's instances by version. You can then use these service subsets in routing rules of virtual services to control the traffic to different instances of you service. In our case, the destination rule configures two different subsets of the `image-classifier` service using the `version` label as a selector, in order to differentiate between ResNet50 and ResNet101 deployments.

```
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: image-classifier
spec:
  host: image-classifier
  subsets:
  - name: resnet101
    labels:
      version: resnet101
  - name: resnet50
    labels:
      version: resnet50
```

Recall that the ResNet50 deployment is annotated with two labels: `app: image-classifier` and `version: resnet50`. The ResNet101 deployment is also annotated with the same labels. The value of the `app` label is the same but the value of the `version` label is different.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: image-classifier-resnet101
  namespace: default
  labels:
    app: image-classifier
    version: resnet101
```

Our destination rule allows the virtual service to route traffic between two deployments behind the `image-classifier` service.

To create the destination rule:

```
kubectl apply -f tf-serving/destinationrule.yaml
```

The final step is to reconfigure the virtual service to use the destination rule.

You will start by modifying the virtual service to route all requests from the Ingress

gateway to the `resnet50` service subset.

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: image-classifier
spec:
  hosts:
  - "*"
  gateways:
  - image-classifier-gateway
  http:
  - route:
    - destination:
        host: image-classifier
        subset: resnet50
        port:
          number: 8501
      weight: 100
    - destination:
        host: image-classifier
        subset: resnet101
        port:
          number: 8501
      weight: 0
```

Notice how the `weight` field is used to redirect 100% of traffic to the `resnet50` subset of the `image-classifier` service.

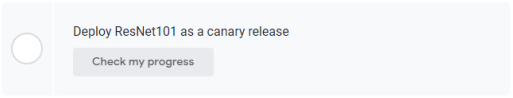
To apply the changes:

```
kubectl apply -f tf-serving/virtualservice-weight-100.yaml
```

Let's now deploy the ResNet101 model using the same process as for the ResNet50 model. Refer back to the steps above to the `kubectl` commands for applying updates to the configmaps and deployment configurations.

1. Update the `tf-serving/configmap-resnet101.yaml` to point to the `gs://[YOUR_BUCKET]/resnet_101` location.
2. Apply the `tf-serving/configmap-resnet101.yaml` manifest.
3. Apply the `tf-serving/deployment-resnet101.yaml` manifest.
4. Wait for the deployment to start successfully.

Click *Check my progress* to verify the objective



At this point the ResNet101 deployment is ready but the virtual service is configured to route all requests to ResNet50.

Verify this by sending a few prediction requests.

```
curl -d @payloads/request-body.json -X POST
http://$GATEWAY_URL/v1/models/image_classifier:predict
```

All requests return the same result.

Configuring weighted load balancing

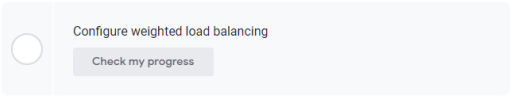
You will now reconfigure the Istio virtual service to split the traffic between the ResNet50 and ResNet101 models using weighted load balancing - 70% requests will go to ResNet50 and 30% to ResNet101. The manifest for the new configuration is in the `tf-serving/virtualservice-weight-70.yaml` file.

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: image-classifier
spec:
  hosts:
  - "*"
  gateways:
  - image-classifier-gateway
  http:
  - route:
    - destination:
        host: image-classifier
        subset: resnet50
        port:
          number: 8501
      weight: 70
    - destination:
        host: image-classifier
        subset: resnet101
        port:
          number: 8501
      weight: 30
```

To apply the manifest:

```
kubectl apply -f tf-serving/virtualservice-weight-70.yaml
```

Click *Check my progress* to verify the objective



Send a few more requests - more than 10.

```
curl -d @payloads/request-body.json -X POST
http://$GATEWAY_URL/v1/models/image_classifier:predict
```

Notice that responses are now different. The probability assigned to the `military uniform` label is around 94% for some reponses and %45 for the others. The 94% responses are from the ResNet101 model.

As an optional task, reconfigure the virtual service to route 100% traffic to the ResNet101 model.

Configuring focused canary testing

In the previous steps you learned how to control fine-grained traffic percentages. Istio routing rules allow for much more sophisticated canary testing scenarios.

In this section, you will reconfigure the virtual service to route traffic to the canary deployment based on request headers. This approach allows a variety of scenarios, including routing requests from a specific group of users to the canary release. Let's assume that the request from the canary users will carry a custom header `user-group`. If this header is set to `canary` the requests will be routed to ResNet101.

The `tf-serving/virtualservice-focused-routing.yaml` manifest defines this configuration:

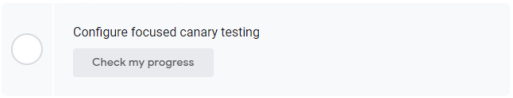
```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: image-classifier
spec:
  hosts:
  - "*"
  gateways:
  - image-classifier-gateway
  http:
  - match:
    - headers:
        user-group:
          exact: canary
    route:
      - destination:
          host: image-classifier
          subset: resnet101
          port:
            number: 8501
  - route:
    - destination:
        host: image-classifier
        subset: resnet50
        port:
          number: 8501
```

The `match` field defines a matching pattern and the route that is used for the requests with the matching header.

Reconfigure the virtual service:

```
kubectl apply -f tf-serving/virtualservice-focused-routing.yaml
```

Click *Check my progress* to verify the objective



Send a few requests without the `user-group` header.

```
curl -d @payloads/request-body.json -X POST
http://$GATEWAY_URL/v1/models/image_classifier:predict
```

Notice that all of the responses are coming from the ResNet50 model - the probablity of the `military uniform` label is around 45%.

Now, send a few requests with the `user-group` header set to `canary`.

```
curl -d @payloads/request-body.json -H "user-group: canary" -X
POST http://$GATEWAY_URL/v1/models/image_classifier:predict
```

All of the response are now sent by the ResNet101 model - the probability of the `military uniform` label is around 94%.

This deployment configuration can be used for A/B testing as you can easily monitor the

performance of the canary model on a specific user group.

Congratulations

In this lab, you learned how to setup canary model releases to test for early issues and measure new model release performance through A/B tests with users in a live production environment. You setup a GKE cluster with a high-performance model server, TF Serving, with an Isitio add-on for creating a service mesh to load balance requests and route traffic between models. You deployed ResNet50 as a simulated production model, ResNet101 as a new canary release, and then configured traffic splitting strategies between models based on weighted load balancing and on request headers. You can now utilize canary releases in your own model deployment workflows to help you operate reliable and continuously improving production machine learning services.

Finish your Quest

This self-paced lab is part of Qwiklabs Advanced ML: ML Infrastructure Quest. A Quest is a series of related labs that form a learning path. Completing a Quest earns you a badge to recognize your achievement. You can make your badge (or badges) public and link to them in your online resume or social media account. Enroll in this Quest and get immediate completion credit for taking this lab. See other available Qwiklabs Quests.

Next steps / learn more

This lab is apart of the DeepLearning.AI Machine Learning for Engineering (MLOps) specialization available on Coursera. Continue your learning by enrolling in and completing the specialization to earn a certificate to showcase your MLOps expertise.

Google Cloud also offers a managed model prediction service through its Vertex AI platform. The Vertex Prediction makes it easy to deploy models into production, for online serving via HTTP or batch prediction for bulk scoring. You can deploy custom models built on any framework (e.g. TensorFlow, PyTorch, Scikit-learn, XGBoost) to Vertex Prediction, with built-in explainability, metadata, and monitoring tooling to track your models' performance. Learn more by visiting the Vertex Prediction documentation. Lastly, review Google Cloud's recommended best practices for model deployment and serving in this architecture guide.

Google Cloud Training & Certification

...helps you make the most of Google Cloud technologies. Our classes include technical skills and best practices to help you get up to speed quickly and continue your learning journey. We offer fundamental to advanced level training, with on-demand, live, and virtual options to suit your busy schedule. Certifications help you validate and prove your skill and expertise in Google Cloud technologies.

Manual Last Updated August 16, 2021
Lab Last Tested August 16, 2021