

# Production Machine Learning Systems

[Professional Machine Learning Engineer Certification Learning Path](#) navigate\_next [Production Machine Learning Systems](#) navigate\_next Designing Adaptable ML Systems

## Vertex AI: Training and Serving a Custom Model

2 hours Free

### Overview

In this lab, you learn how to use [Vertex AI](#) to train and serve a TensorFlow model using code in a custom container.

While you're using TensorFlow for the model code here, you could easily replace it with another framework.

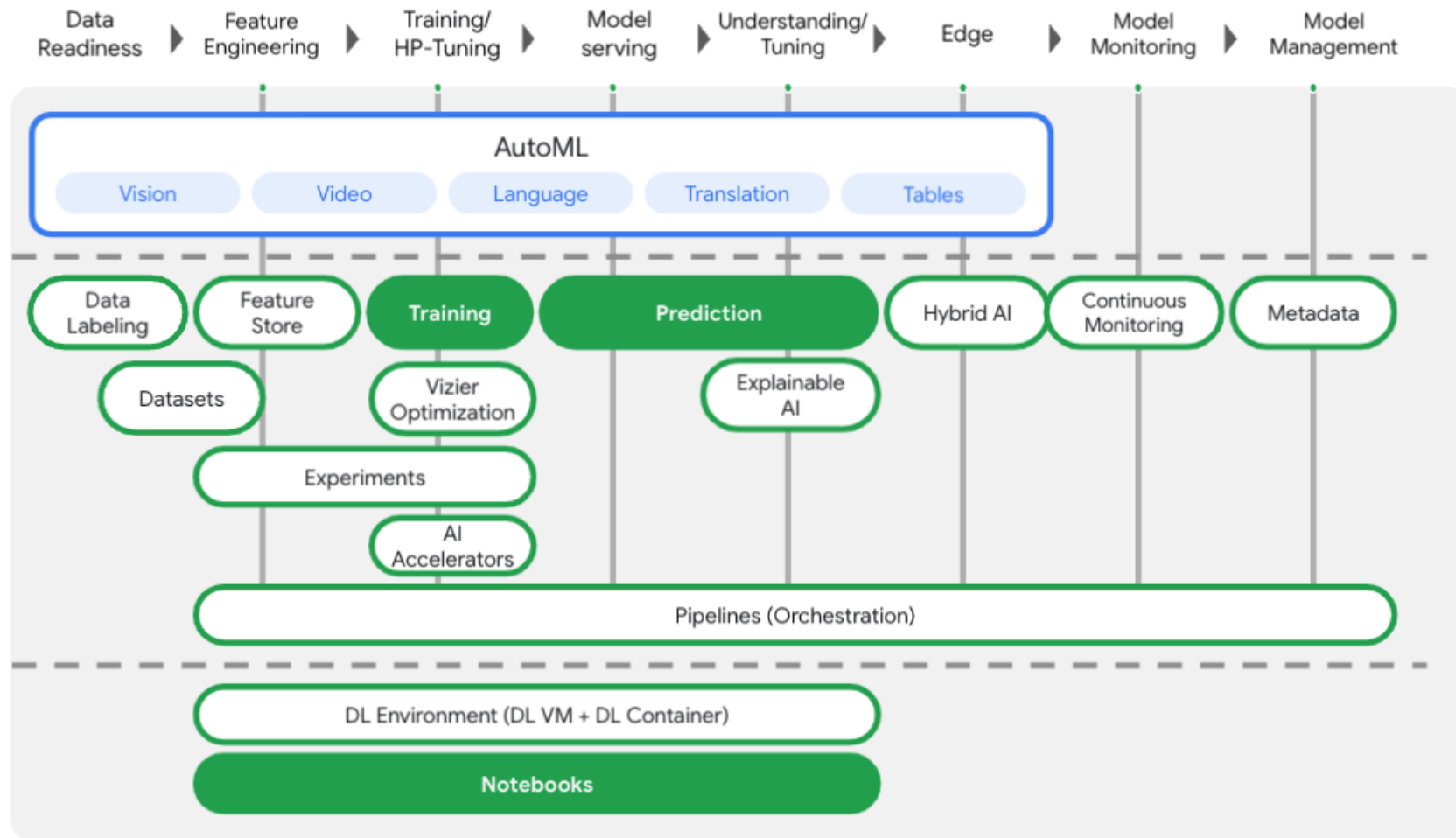
### Learning objectives

- Build and containerize model training code in Vertex Notebooks.
- Submit a custom model training job to Vertex AI.
- Deploy your trained model to an endpoint, and use that endpoint to get predictions.

### Introduction to Vertex AI

This lab uses the newest AI product offering available on Google Cloud. [Vertex AI](#) integrates the ML offerings across Google Cloud into a seamless development experience. Previously, models trained with AutoML and custom models were accessible via separate services. The new offering combines both into a single API, along with other new products. You can also migrate existing projects to Vertex AI. If you have any feedback, please see the [support page](#).

Vertex AI includes many different products to support end-to-end ML workflows. This lab will focus on the products highlighted below: Training, Prediction, and Notebooks.



## Task 1. Set up your environment

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

1. Sign in to Qwiklabs using an **incognito window**.
2. Note the lab's access time (for example, 1:15:00), and make sure you can finish within that time. There is no pause feature. You can restart if needed, but you have to start at the beginning.
3. When ready, click **Start lab**.
4. Note your lab credentials (**Username** and **Password**). You will use them to sign in to the Google Cloud Console.

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 receive errors or **incur charges**.
7. Accept the terms and skip the recovery resource page.

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

## Enable the Compute Engine API

Navigate to [Compute Engine API](#) and select **Enable** if it isn't already enabled. You'll need this to create your notebook instance.

## Enable the Vertex AI API

Navigate to the [Vertex AI section of your Cloud Console](#) and click **Enable Vertex AI API**.

## Enable the Container Registry API

Navigate to the [Container Registry](#) and select **Enable** if it isn't already enabled. You'll use this to create a container for your custom training job.

## Enable NoteBooks API

## Launch Vertex AI Notebooks instance

1. In the Google Cloud Console, on the **Navigation Menu**, click **Vertex AI > Workbench**. Select **User-Managed Notebooks**.
2. On the Notebook instances page, click **New Notebook > TensorFlow Enterprise > TensorFlow Enterprise 2.6 (with LTS) > Without GPUs**.
3. In the **New notebook** instance dialog, confirm the name of the deep learning VM, if you don't want to change the region and zone, leave all settings as they are and then click **Create**. The new VM will take 2-3 minutes to start.
4. Click **Open JupyterLab**.  
A JupyterLab window will open in a new tab.
5. You will see "Build recommended" pop up, click **Build**. If you see the build failed, ignore it.

## Task 2. Containerize training code

You are going to submit this training job to Vertex by putting your training code in a [Docker container](#) and pushing this container to [Google Container Registry](#). Using this approach, you can train a model built with any framework.

To start, from the Launcher menu, open a terminal window in your notebook instance.

Create a new directory called `mpg` and `cd` into it:

```
mkdir mpg cd mpg
```

## Create a Dockerfile

Your first step in containerizing your code is to create a Dockerfile. In your Dockerfile you'll include all the commands needed to run your image. It'll install all the libraries you're using and set up the entry point for your training code. From your terminal, create an empty Dockerfile:

```
touch Dockerfile
```

Open the Dockerfile and copy the following into it:

```
FROM gcr.io/deeplearning-platform-release/tf2-cpu.2-3 WORKDIR /root WORKDIR / # Copies the trainer code to the docker image. COPY trainer /trainer # Sets up the entry point to invoke the trainer. ENTRYPOINT ["python", "-m", "trainer.train"]
```

This Dockerfile uses the [Deep Learning Container TensorFlow Enterprise 2.3 Docker image](#). The Deep Learning Containers on Google Cloud come with many common ML and data science frameworks pre-installed. The one you're using includes TF Enterprise 2.3, Pandas, Scikit-learn, and others. After downloading that image, this Dockerfile sets up the entrypoint for your training code. You haven't created these files yet--in the next step, you'll add the code for training and exporting your model.

## Create a Cloud Storage bucket

In your training job, you'll export your trained TensorFlow model to a Cloud Storage Bucket. Vertex will use this to read your exported model assets and deploy the model. From your terminal, run the following to define an env variable for your project, making sure to replace `your-cloud-project` with the ID of your project:

**Note:** You can get your project ID by running `gcloud config list --format 'value(core.project)'` in your terminal. `PROJECT_ID=`

Next, run the following in your terminal to create a new bucket in your project. The `-l` (location) flag is important since this needs to be in the same region where you'll deploy a model endpoint later in the tutorial:

```
BUCKET_NAME="gs://${PROJECT_ID}-bucket" gsutil mb -l us-central1 $BUCKET_NAME
```

## Add model training code

From your terminal, run the following to create a directory for your training code and a Python file where you'll add the code:

```
mkdir trainer touch trainer/train.py
```

You should now have the following in your mpg/ directory:

```
+ Dockerfile + trainer/ + train.py
```

Next, open the `train.py` file you just created and copy the code below (this is adapted from the [tutorial](#) in the TensorFlow docs).

At the beginning of the file, update the `BUCKET` variable with the name of the Storage Bucket you created in the previous step:

```
import numpy as np import pandas as pd import pathlib import tensorflow as tf from tensorflow import keras from tensorflow.keras import layers
print(tf.__version__) """## The Auto MPG dataset The dataset is available from the [UCI Machine Learning
Repository](https://archive.ics.uci.edu/ml/). ### Get the data First download the dataset. """ dataset_path = keras.utils.get_file("auto-mpg.data",
"http://archive.ics.uci.edu/ml/machine-learning-databases/auto-mpg/auto-mpg.data") dataset_path """Import it using pandas""" column_names =
['MPG','Cylinders','Displacement','Horsepower','Weight', 'Acceleration', 'Model Year', 'Origin'] dataset = pd.read_csv(dataset_path,
names=column_names, na_values = "?", comment='#t', sep=" ", skipinitialspace=True) dataset.tail() # TODO: replace `your-gcs-bucket` with the
name of the Storage bucket you created earlier BUCKET = 'gs://your-gcs-bucket' """### Clean the data The dataset contains a few unknown values.
""" dataset.isna().sum() """To keep this initial tutorial simple, drop those rows.""" dataset = dataset.dropna() """The `Origin` column is really
categorical, not numeric. So convert that to a one-hot: """ dataset['Origin'] = dataset['Origin'].map({1: 'USA', 2: 'Europe', 3: 'Japan'}) dataset =
pd.get_dummies(dataset, prefix=" ", prefix_sep="_") dataset.tail() """### Split the data into train and test Now split the dataset into a training set and a
test set. You will use the test set in the final evaluation of your model. """ train_dataset = dataset.sample(frac=0.8,random_state=0) test_dataset =
dataset.drop(train_dataset.index) """### Inspect the data Have a quick look at the joint distribution of a few pairs of columns from the training set.
Also look at the overall statistics: """ train_stats = train_dataset.describe() train_stats.pop("MPG") train_stats = train_stats.transpose() train_stats
"""### Split features from labels Separate the target value, or "label", from the features. This label is the value that you will train the model to
predict. """ train_labels = train_dataset.pop('MPG') test_labels = test_dataset.pop('MPG') """### Normalize the data Look again at the `train_stats`
block above and note how different the ranges of each feature are. It is good practice to normalize features that use different scales and ranges.
Although the model *might* converge without feature normalization, it makes training more difficult, and it makes the resulting model dependent on
the choice of units used in the input. Note: Although we intentionally generate these statistics from only the training dataset, these statistics will also
be used to normalize the test dataset. We need to do that to project the test dataset into the same distribution that the model has been trained on. """
def norm(x): return (x - train_stats['mean']) / train_stats['std'] normed_train_data = norm(train_dataset) normed_test_data = norm(test_dataset)
"""This normalized data is what we will use to train the model. Caution: The statistics used to normalize the inputs here (mean and standard
deviation) need to be applied to any other data that is fed to the model, along with the one-hot encoding that we did earlier. That includes the test set
as well as live data when the model is used in production. ## The model ### Build the model Let's build our model. Here, we'll use a `Sequential`
model with two densely connected hidden layers, and an output layer that returns a single, continuous value. The model building steps are wrapped in
a function, `build_model`, since we'll create a second model later on. """ def build_model(): model = keras.Sequential([ layers.Dense(64,
activation='relu', input_shape=[len(train_dataset.keys())]), layers.Dense(64, activation='relu'), layers.Dense(1) ]) optimizer =
```

```
tf.keras.optimizers.RMSprop(0.001) model.compile(loss='mse', optimizer=optimizer, metrics=['mae', 'mse']) return model
model = build_model()
"""### Inspect the model Use the `.summary` method to print a simple description of the model """ model.summary()
"""Now try out the model. Take a batch of `10` examples from the training data and call `model.predict` on it. It seems to be working, and it produces a result of the expected shape and type. ### Train the model Train the model for 1000 epochs, and record the training and validation accuracy in the `history` object. Visualize the model's training progress using the stats stored in the `history` object. This graph shows little improvement, or even degradation, in the validation error after about 100 epochs. Let's update the `model.fit` call to automatically stop training when the validation score doesn't improve. We'll use an *EarlyStopping callback* that tests a training condition for every epoch. If a set amount of epochs elapses without showing improvement, then it will automatically stop the training. You can learn more about this callback [here](https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/EarlyStopping). """
model = build_model() EPOCHS = 1000 # The patience parameter is the amount of epochs to check for improvement
early_stop = keras.callbacks.EarlyStopping(monitor='val_loss', patience=10)
early_history = model.fit(normed_train_data, train_labels, epochs=EPOCHS, validation_split = 0.2, callbacks=[early_stop]) # Export model and save to GCS
model.save(BUCKET + '/mpg/model')
```

## Build and test the container locally

From your terminal, define a variable with the URI of your container image in Google Container Registry:

```
IMAGE_URI="gcr.io/$PROJECT_ID/mpg:v1"
```

Then, build the container by running the following from the root of your `mpg` directory:

```
docker build ./ -t $IMAGE_URI
```

Run the container within your notebook instance to ensure it's working correctly:

```
docker run $IMAGE_URI
```

The model should finish training in 1-2 minutes with a validation accuracy around 72% (exact accuracy may vary). When you've finished running the container locally, push it to Google Container Registry:

```
docker push $IMAGE_URI
```

With our container pushed to Container Registry, you're now ready to kick off a custom model training job.

## Task 3. Run a training job on Vertex AI

Vertex AI gives you two options for training models:

- **AutoML:** Train high-quality models with minimal effort and ML expertise.
- **Custom training:** Run your custom training applications in the cloud using one of Google Cloud's pre-built containers, or use your own.

In this lab, you're using custom training via our own custom container on Google Container Registry. To start, navigate to the **Models** section in the Vertex section of your Cloud console:



Dashboard



Data sets



Features



Labelling tasks



Notebooks



Pipelines



Training



Models



Endpoints



Batch predictions

**Kick off the training job**



Click **Create** to enter the parameters for your training job and deployed model:

- Under **Dataset**, select **No managed dataset**.
- Then select **Custom training (advanced)** as your training method and click **Continue**.
- Enter `mpg` (or whatever you'd like to call your model) for **Model name**.
- Click **Continue**.

In the **Container settings** step, select **Custom container**:

Select a pre-built container or build a custom container using ML frameworks (as well as non-ML dependencies, libraries and binaries) that are not otherwise supported.

[Learn more](#)

☐ **Pre-built container**  
View the list of [supported runtimes](#) including TensorFlow, scikit-learn and PyTorch versions

☒ **Custom container**  
Build a custom Docker container. Must be stored in [Container Registry](#)

In the first box (**Container image on GCR**), enter the value of your `IMAGE_URI` variable above. It should be: `gcr.io/your-cloud-project/mpg:v1`, with your own project name. Leave the rest of the fields blank and click **Continue**.

You won't use hyperparameter tuning in this tutorial, so leave the **Enable hyperparameter tuning** box **unchecked** and click **Continue**.

In **Compute and pricing**, leave the selected region as is and select the following machine type:

**Machine type \***

☰ Filter Type to filter

Standard	<b>n1-standard-4</b> 4 vCPUs, 15 GiB memory
High-memory	
High-cpu	<b>n1-standard-8</b> 8 vCPUs, 30 GiB memory
Custom	
High-gpu	<b>n1-standard-16</b> 16 vCPUs, 60 GiB memory
Mega-gpu	<b>n1-standard-32</b> 32 vCPUs, 120 GiB memory

[CLEAR SELECTION](#)

Because the model in this demo trains quickly, you're using a smaller machine type.

**Note:** You are welcome to experiment with larger machine types and GPUs if you'd like. If you use GPUs, you'll need to use a GPU-enabled base container image.

Under the **Prediction container** step, select **Pre-built container** and select **2.6** as the **Model framework version**.

## Train new model

- ✓ Training method
- ✓ Model details
- ✓ Training container
- ✓ Hyperparameters (optional)
- ✓ Compute and pricing
- 6 Prediction container (optional)**

START TRAINING

CANCEL

You can associate your custom-trained model with a container in order to serve prediction requests using Vertex AI. [Learn more about getting predictions.](#)

- ☐ No prediction container  
You can always import your model artifact later to serve prediction requests
- ☒ Pre-built container  
View the list of [supported runtimes](#) including TensorFlow, scikit-learn and PyTorch versions
- ☐ Custom container  
Build a custom Docker container. Must be stored in [Container Registry or Artifact Registry](#)

### Pre-built container settings

Vertex AI provides Docker container images for serving predictions. To use a pre-built container, your trained model code must be in Python 3.7. [Learn more about pre-built containers](#)

In order to run in a pre-built container, your code needs to be in Python 3.7

Model framework \*  
TensorFlow ▼

Model framework version \*  
2.6 ▼

Accelerator type \*  
None ▼

gs:// Model directory \* BROWSE

Cloud Storage location containing the model artifact and any supporting files

Leave the default settings for the pre-built container as is. Under **model directory**, enter your GCS bucket with the mpg subdirectory. This is the path in your model training script where you export your trained model. It should look like this:

### Model directory \*

✓ gs:// your-gcs-bucket/mpg

BROWSE

The location on Cloud Storage where your Python module outputted your model artifacts

Vertex will look in this location when it deploys your model. Now you're ready for training! Click **Start training** to kick off the training job. In the **Training** section of your console, you'll see something like this:

#### TRAINING PIPELINE

#### CUSTOM JOB

#### HYPERPARAMETER TUNING

Training pipelines are the primary model training workflow in AI Platform (Unified). You can use training pipelines to create an AutoML-trained model or a custom-trained model. For custom-trained models, training pipelines orchestrate custom training jobs and hyperparameter tuning with additional steps like adding a dataset or uploading the model to AI Platform for prediction serving. [Learn More](#)

Region

us-central1 (Iowa)



Filter training pipelines...

Name	ID	Job type	Model type
mpg	2191265101506412544	Training pipeline	Custom

**Note:** The training job will take about 10-15 minutes to complete.

## Task 4. Deploy a model endpoint

When you set up your training job, you specified where Vertex AI should look for your exported model assets. As part of our training pipeline, Vertex will create a model resource based on this asset path. The model resource itself isn't a deployed model, but once you have a model you're ready to deploy it to an endpoint. To learn more about models and endpoints in Vertex AI, check out the [Get started documentation](#) for Vertex AI.

In this step you'll create an endpoint for our trained model. You can use this to get predictions on our model via the Vertex AI API.

## Deploy endpoint

When your training job completes, you should see a model named **mpg** (or whatever you named it) in the **Models** section of your console:

Models are built from your data sets or unmanaged data sources. There are many different types of machine-learning models available on AI Platform, depending on your use case and level of experience with machine learning. [Learn more](#)

Region

us-central1 (Iowa) ▼ ?

☰ Filter models...

	Name	ID	Data	Endpoints	Region	Type
✓	mpg	7268545915785314304	—	0	us-central1	👤 Custom trained

When your training job ran, Vertex created a model resource for you. In order to use this model, you need to deploy an endpoint. You can have many endpoints per model. Click on the model and then click **Deploy to endpoint**.




Select **Create new endpoint** and give it a name, like `v1`, and click **Continue**. Leave **Traffic split** at 100 and enter 1 for **Minimum number of compute nodes**. Under **Machine type**, select **n1-standard-2** (or any machine type you'd like). Then click **Done** and then **Deploy**.

Deploying the endpoint will take 10-15 minutes. When the endpoint has finished deploying, you'll see the following, which shows one endpoint deployed under your model resource:

## Deploy your model

Endpoints are machine learning models made available for online prediction requests. Endpoints are useful for timely predictions from many users (for example, in response to an application request). You can also request batch predictions if you don't need immediate results.

DEPLOY TO ENDPOINT

	Endpoint	ID	Models	Region	Last updated	API	Notification	Metadata	
	<a href="#">v1</a>	6681723365901729792	1	us-central1	Sep 21, 2020, 3:24:21 PM	<a href="#">Sample request</a>			

**Note:** Bucket region and the region used to deploy the model endpoint need to be the same.

## Get predictions on the deployed model

You'll get predictions on our trained model from a Python notebook, using the Vertex Python API. Go back to your notebook instance, and create a Python 3 notebook from the Launcher:



## Notebook

---



Python 3



Python 2



## Console

---



Python 3



Python 2



## Other

---



Terminal



Text File



Markdown File



Show  
Contextual Help

In your notebook, run the following in a cell to install the Vertex AI SDK:

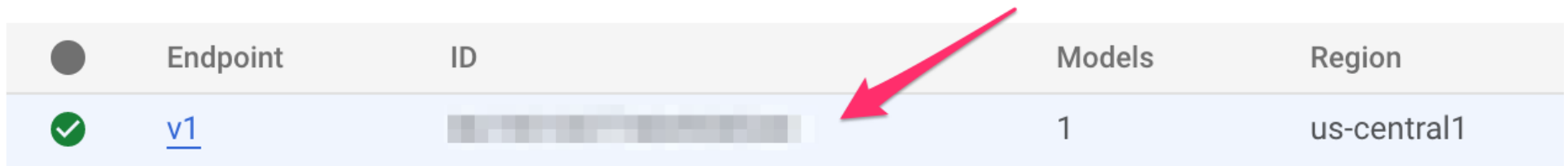
```
!pip3 install google-cloud-aiplatform --upgrade --user
```

Then add a cell in your notebook to import the SDK and create a reference to the endpoint you just deployed:

```
from google.cloud import aiplatform endpoint = aiplatform.Endpoint( endpoint_name="projects/YOUR-PROJECT-NUMBER/locations/us-central1/endpoints/YOUR-ENDPOINT-ID" )
```

You'll need to replace two values in the endpoint\_name string above with your project number and endpoint. You can find your project number by navigating to your [project dashboard](#) and getting the Project Number value.

You can find your endpoint ID in the endpoints section of the console here:



	Endpoint	ID	Models	Region
✓	<a href="#">v1</a>	[REDACTED]	1	us-central1

Finally, make a prediction to your endpoint by copying and running the code below in a new cell:

```
test_mpg = [1.4838871833555929, 1.8659883497083019, 2.234620276849616, 1.0187816540094903, -2.530890710602246, -1.6046416850441676, -0.4651483719733302, -0.4952254087173721, 0.7746763768735953] response = endpoint.predict([test_mpg]) print('API response: ', response) print('Predicted MPG: ', response.predictions[0][0])
```


This example already has normalized values, which is the format our model is expecting.

Run this cell, and you should see a prediction output around 16 miles per gallon.

## Task 5. Cleanup

- Do one of the following:
  - To continue using the notebook you created in this lab, it is recommended that you turn it off when not in use. From the Notebooks UI in the Cloud Console, select the notebook and then click **Stop**.
  - To delete the notebook entirely, click **Delete**.
- To delete the endpoint you deployed, in the **Endpoints** section of your Vertex AI console, click **Undeploy model from endpoint**, and then click **Undeploy**.



3. To remove the endpoint, click the overflow menu , and then click **Remove endpoint**.
4. To delete the Cloud Storage bucket, on the **Cloud Storage** page, select your bucket, and then click **Delete**.

## Congratulations!

You've learned how to use Vertex AI to:

- Train a model by providing the training code in a custom container. You used a TensorFlow model in this example, but you can train a model built with any framework using custom containers.
- Deploy a TensorFlow model using a pre-built container as part of the same workflow you used for training.
- Create a model endpoint and generate a prediction.

## 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.

Copyright 2022 Google LLC All rights reserved. Google and the Google logo are trademarks of Google LLC. All other company and product names may be trademarks of the respective companies with which they are associated.

- [Overview](#)
- [Introduction to Vertex AI](#)
- [Task 1. Set up your environment](#)
- [Task 2. Containerize training code](#)

- [Task 3. Run a training job on Vertex AI](#)
- [Task 4. Deploy a model endpoint](#)
- [Task 5. Cleanup](#)
- [Congratulations!](#)
- [End your lab](#)