

[Open in app](#)[Follow](#)

552K Followers



HANDS-ON TUTORIALS

A Hands-on Tutorial to Continuous Deployment Pipelines for ML-based Web Apps on Google Cloud

A comprehensive guide to ML App Deployment using Flask



Dr. Sohini Roychowdhury Feb 17 · 7 min read

Machine Learning (ML) models typically leverage the capability to learn patterns from previously seen (training) data and apply them to predict the outcome for new (test) data. Deploying ML models as web apps can help test the efficacy of trained ML models by allowing access to multiple users and testing environments, thereby gathering test performance metrics. However, ML web app deployment in production can be a highly complex process that may need to ensure minimal down-time for users in the event of app update processes. Cloud-based deployment solutions such as Google Cloud Platform (GCP) have highly simplified the process of continuous integration and continuous deployment (CI/CD) through pipelines and triggers that can be designed to ensure sanity checks as well as the integrity of the integrated code base for updated application runs.

In this tutorial, we will review a detailed example wherein we deploy a ML model web app on GCP through a CD pipeline. To follow the steps from data analysis to final app deployment, fork the github repository at [1] and follow the steps below.

[Open in app](#)

The first step is understanding the data followed by ML model creation and subsequent deployment. For this step we refer to the file *model.py* under the folder *app_files*.

For this tutorial, we will be using the 50_Startup.csv data from [2] as shown in Fig. 1, where the goal is to train a regression model ($f(X)$) using the three features (x_1, x_2, x_3) as (R&D Spend, Administration Cost, Marketing Cost), and predict the company 'Profit' (Y).

	R&D Spend	Administration	Marketing	Profit
1	165349.2	136897.8	471784.1	192261.83
2	162597.7	151377.59	443898.53	191792.06
3	153441.51	101145.55	407934.54	191050.39
4	144372.41	118671.85	383199.62	182901.99

Fig 1: A snapshot of the training data set for regression data modeling.

Our goal is to estimate Profit as a function of its features as $f(x) = g(x_1, x_2, x_3)$, where $g(\cdot)$ represents linear or non-linear combinations of the features x_1, x_2, x_3 , respectively, that can be estimated from the training data.

While Linear Regression is the simplest solution to the above data set, one of the most scalable data models for large data sets continues to be ensemble models that are based on Decision Trees, since they enable data aggregation and expendability. In this work, we implement a non-linear Decision Tree-based Regression model as shown in Fig 2. To train and visualize the regression model, run the file *model.py* with code snippet shown below.

```
1 from sklearn.tree import DecisionTreeRegressor
2 reg=DecisionTreeRegressor(max_depth=5)
3 reg.fit(X,y)
4 y_pred=reg.predict(X)
5 print(reg.score)
```

model.py hosted with ❤ by GitHub

[view raw](#)

A Decision Tree Regressor proceeds training by partitioning sub-regions or blocks in the feature subspace such that similar samples flock together to minimize the residual sum

[Open in app](#)

respective sub-region as the estimated outcome [3]. The non-linear nature of the trained decision tree regression model with a maximum depth of 5 is shown in Fig. 2. The code to visualize the trained model across each feature is shown below.

```

1  #To generate a best fit model
2  X_range=np.zeros((50,3))
3  y_range=np.zeros((50,))
4  for i in range(3):
5      Xi=X[:,i]
6      vals=plt.hist(Xi,49)
7      plt.xlabel("Feature")
8      plt.ylabel("Frequency")
9      X_range[:,i]=np.transpose(vals[1])
10 y_range=model.predict(X_range)
11
12
13 # Plot the results
14 plt.figure()
15 plt.scatter(X[:,0], y, s=20, edgecolor="black", c="darkorange", label="train data")
16 plt.scatter(x_test[:,0], model.predict(x_test), s=30, color="yellowgreen", label="test data")
17 plt.plot(X_range[:,0], y_range, color="cornflowerblue",
18         label="Regression_model", linewidth=2)
19 plt.xlabel("R&D Cost")
20 plt.ylabel("Profit")
21 plt.title("Decision Tree Regression")
22 plt.legend()
23 plt.show()

```

model.py hosted with ❤ by GitHub

[view raw](#)

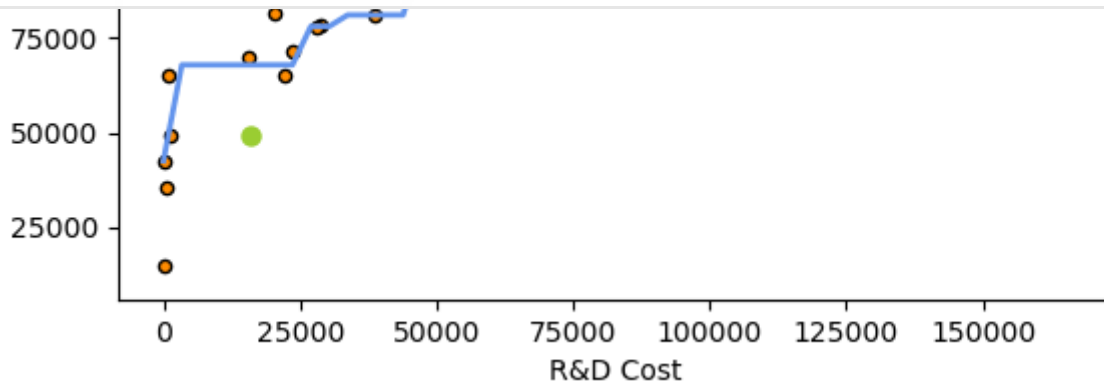
[Open in app](#)

Fig 2: Example of Decision Tree Regression model fit on the data set visualized on the single feature of R&D cost. The test data set is [16000, 135000, 450000].

Step 1 Command at terminal: python model.py

Step 2: ML Web App using Flask

Having built the ML model, the next step is to package it and serve as a Web App. Here, we refer to the file *app.py* and html files under the *templates* folder contained in *app_files*.

We utilize Flask-based RESTful api from [4]. REpresentational State Transfer (REST) is a standard architectural design for web APIs and Flask has been the choice for python web API serves since its inception in 2010. The two major components for designing the Flask-API are [5]:

1. Flask library and POST HTTP method to serve the predictions.
2. Front end HTML files that are useful for users to interact with the ML model.

One significant change in the *app.py* file from general Flask-based python deployments in [4–5] is that the running *app.py* launches an API that runs at <http://0.0.0.0:8000> instead of its default settings.

```
1 if __name__ == '__main__':
2     app.run( debug=True, host='0.0.0.0', port=int(os.environ.get('PORT', 8080)) )
3     label="Regression_model", linewidth=2)
```

app.py hosted with ❤ by GitHub

[view raw](#)

[Open in app](#)

port 8080. For details on ways to debug a GCR app see [6].

Step 2 Command at terminal: `python app.py`

Step 3: Create a Docker Container for Google Cloud

Now that we have a functional web app, we need to package it for a Google Cloud Run. For this purpose, we will create a Docker container, as in [7] that is a standalone software and can be installed on any physical or virtual machine to run containerized applications. The Docker file for this tutorial, as shown below, must include 4 key components.

```
1 FROM python:3.6-slim-buster
2
3 RUN python -m pip install --upgrade pip
4 COPY requirements.txt .
5 RUN pip install -r requirements.txt
6
7 COPY . /app
8 WORKDIR /app/app_files
9
10 CMD ["python", "app.py"]
```

Dockerfile hosted with ❤️ by GitHub

[view raw](#)

1. The operating system needed by the application;
2. List of dependencies such as libraries and packages in the *requirements.txt* file;
3. Application files;
4. Command to launch the application.

Step 4: Create a YAML file for the Cloud Run

The final component required to run the CD pipeline is a **YAML** Ain't Markup Language file based on the application in [7]. YAML files follow the data serialization standard to build configuration settings for a CD pipeline. Details on building a YAML file for your

[Open in app](#)

```
1  steps:
2    # Step 1: pull the container image if it is already built, if fails do not exit
3    - name: 'gcr.io/cloud-builders/docker'
4      entrypoint: 'bash'
5      args:
6        - '-c'
7        - 'docker pull gcr.io/$PROJECT_ID/appcid:latest || exit 0'
8    # Step 2: Create a docker image if none exists, esle load existing one
9    - name: 'gcr.io/cloud-builders/docker'
10     args:
11       - 'build'
12       - '-t'
13       - 'gcr.io/$PROJECT_ID/appcid:latest'
14       - '-t'
15       - 'gcr.io/$PROJECT_ID/appcid:$COMMIT_SHA'
16       - '--cache-from'
17       - 'gcr.io/$PROJECT_ID/appcid:latest'
18       - '.'
19    # Step 3: Push the docker image to the run
20    - name: 'gcr.io/cloud-builders/docker'
21     args:
22       - 'push'
23       - 'gcr.io/$PROJECT_ID/appcid'
24    # Step 4: Deploy container image to Cloud Run
25    - name: 'gcr.io/cloud-builders/gcloud'
26     args:
27       - 'run'
28       - 'deploy'
29       - 'flaskappml'
30       - '--image'
31       - 'gcr.io/$PROJECT_ID/appcid:latest'
32       - '--region'
33       - 'us-central1'
34       - '--platform'
35       - 'managed'
36       - '--allow-unauthenticated'
```

cloudbuild.yaml hosted with ❤ by GitHub

[view raw](#)

[Open in app](#)

2. Create a build if none exists so far, otherwise use the cache build.
3. Push the docker image to GCR
4. Deploy the application to GCR.

Now that we have all the components to run and deploy the ML web app, lets create a trigger to deploy the app on *Google Cloud*.

Step 5: Create the CD Pipeline and Run

Now that we have an app that runs locally, the next task is to set up a project in GCP that will generate triggers and the cloud run. Follow the steps below to ensure you have a viable Google cloud project to begin the CD process.

1. Sign up at Google Console: <https://console.cloud.google.com/home/>
2. Create a new Project as shown below:

Google Cloud Platform

New Project

Project name *
My Project 90035 ?

Project ID: hopeful-dolphin-304723. It cannot be changed later. [EDIT](#)

Organization *
fourthbrain.ai ?

Select an organization to attach it to a project. This selection can't be changed later.

Location *
fourthbrain.ai [BROWSE](#)

Parent organization or folder

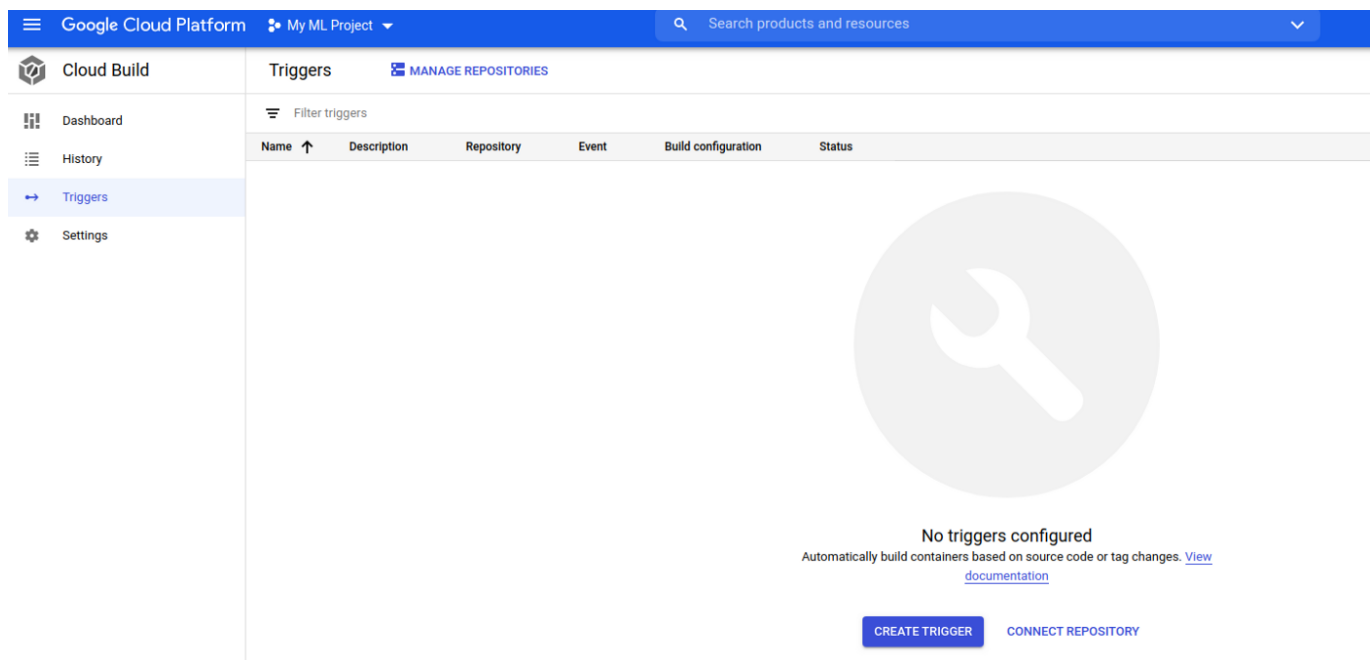
[CREATE](#) [CANCEL](#)

[Open in app](#)


3. Ensure Billing is activated for the project.

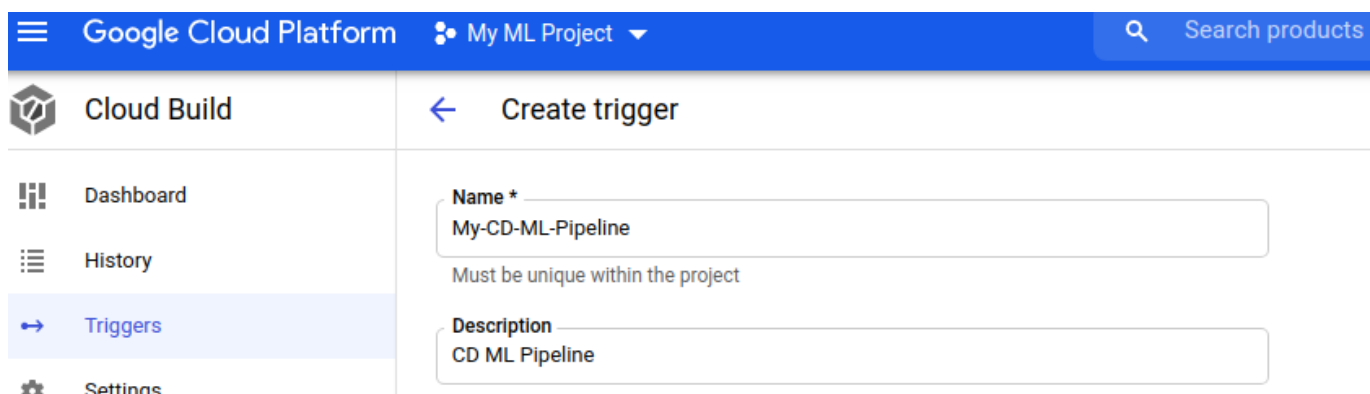
4. Start from Github code to create a CD pipeline. Use the forked Github repository from your Github account for this purpose.

5. Go to the page: <https://console.cloud.google.com/cloud-build/triggers>. Ensure your project is selected on the top panel. Next click 'Create Trigger' button at the bottom as shown below.



Build Triggers in GCP

6. This will take you to the following page where you will need to create a 'Push to a branch' event for your forked Github repository. To deploy the app, select the 'Cloud Build' configuration file option as shown below and hit 'Create' at the bottom.



Open in app



- ☒ Push to a branch
☐ Push new tag
☐ Pull request (GitHub App only)

Or in response to

- ☐ Manual invocation
☐ Webhook event **PREVIEW**

Source

Repository *

Type to filter

- sohiniroych/Azure-CI-CD_Example-ML (GitHub App)
- sohiniroych/Flask-ML-Pipeline_GCP-Tutorial (GitHub App)
- sohiniroych/ml-deployment-on-gcloud (GitHub App)
- sohiniroych/ML-pipeline-Flask-Tutorial (GitHub App)

CONNECT NEW REPOSITORY

Build configuration

File type

- ☒ Cloud Build configuration file (yaml or json)
☐ Dockerfile

Steps to Create a Push Trigger using Github code

7. Now a trigger gets created. Hit RUN option next to the trigger and look at the logs that get generated. This would take a few minutes to run and deploy. At the end you should have something that looks like below.

Google Cloud Platform My ML Project Search products and resources

Cloud Build Build details REBUILD COPY URL

Dashboard History Triggers Settings

Successful: 224506cc
Started on Feb 12, 2021, 3:39:18 PM

Trigger: ML-CD-Pipeline-Deploy Source: sohiniroych/Flask-ML-Pipeline_GCP-Tutorial Branch: main Commit: 13fbc3d

Steps	Duration	BUILD LOG	EXECUTION DETAILS	BUILD ARTIFACTS
Build Summary 4 Steps	00:01:40			
0: gcr.io/cloud-builders/docker bash -c docker pull gcr.io/my-ml-project-303018/appcid...	00:00:01	124 Step #2: 93714065b437: Preparing		
1: gcr.io/cloud-builders/docker build -t gcr.io/my-ml-project-303018/appcid:latest -t gcr...	00:00:36	125 Step #2: 8711af342595: Preparing		
2: gcr.io/cloud-builders/docker push gcr.io/my-ml-project-303018/appcid	00:00:22	126 Step #2: 9e82f04c782: Preparing		
3: gcr.io/cloud-builders/gcloud run deploy flaskappml --image gcr.io/my-ml-project-3030...	00:00:35	127 Step #2: d664a732356a: Waiting		
		128 Step #2: 93714065b437: Waiting		
		129 Step #2: 8711af342595: Waiting		
		130 Step #2: 9e82f04c782: Waiting		
		131 Step #2: c9b5895567e: Layer already exists		
		132 Step #2: d664a732356a: Layer already exists		
		133 Step #2: 93714065b437: Layer already exists		
		134 Step #2: 8711af342595: Layer already exists		
		135 Step #2: 9e82f04c782: Layer already exists		
		136 Step #2: 17a8f15e248: Pushed		
		137 Step #2: a190832acbd8: Pushed		
		138 Step #2: a4409debb592: Pushed		
		139 Step #2: 0744d68555a5: Pushed		
		140 Step #2: 13fbc3d7126327113f3a32a059385d66cfa3: digest: sha256:37fa4502ec7478b4852cc067d10954888559f8f884f52b083238449f2a9822c size: 2289		
		141 Step #2: 17a8f15e248: Preparing		
		142 Step #2: 0744d68555a5: Preparing		
		143 Step #2: a4409debb592: Preparing		
		144 Step #2: a190832acbd8: Preparing		
		145 Step #2: c9b5895567e: Preparing		
		146 Step #2: d664a732356a: Preparing		
		147 Step #2: 17a8f15e248: Layer already exists		
		148 Step #2: 0744d68555a5: Layer already exists		
		149 Step #2: 93714065b437: Preparing		
		150 Step #2: 8711af342595: Preparing		
		151 Step #2: 9e82f04c782: Preparing		
		152 Step #2: a4409debb592: Layer already exists		

[Open in app](#)

Outcome of the Push Trigger Run

8. Now go to the GCR console at: <https://console.cloud.google.com/run> and notice your app (flaskappml) running as shown below:

Google Cloud Platform

My ML Project

Search products and resources

Cloud Run

Services

+

CREATE SERVICE

MANAGE CUSTOM DOMAINS

COPY

DELETE

Each Cloud Run service has a unique endpoint and autoscales deployed containers. [Learn more](#)

Filter

Filter services

<div><input type="checkbox"/></div>	<div><div></div><div>Name</div><div>↑</div></div>	<div>Req/sec</div> <div>?</div>	<div>Region</div>	<div>Authentication</div> <div>?</div>	<div>Ingress</div> <div>?</div>	<div>Last deployed</div>	<div>Deployed by</div>
<div><input checked="" type="checkbox"/></div>	<div>flaskappml</div>	<div>0</div>	<div>us-central1</div>	<div>Allow unauthenticated</div>	<div>All</div>	<div>Feb 12, 2021, 3:40:55 PM</div>	<div>Cloud Build</div>

Deployed App in GCR

Clicking on 'flaskappml' takes you to the GCR page with the URL displayed on top right. This is the URL to your deployed webapp that you can now test on any device.

Don't forget to clean up the triggers and the GCR before your leave. This is crucial to ensure appropriate resource usage.

Conclusion

Building ML-models and deploying them on cloud platforms not only enhances usability, but it also helps gather insights into the limiting conditions for the model. To ensure robust deployed web apps, CI pipelines enable code to be committed to a repository in a systematic manner while CD pipelines enable code sanity triggers and checks to ensure successful app deployment. Extending this tutorial to other web apps is a relatively simple process. This will require the following 4 major changes:

1. Updating the app_files to reflect your application of interest.
2. Updating the Dockerfile and requirements.txt files to represent the libraries needed for your application.

Open in app



repo.

Now you have the resources needed to build CI/CD pipelines for your own ML-based use cases.

References:

[1] S. Roychowdhury. Flask ML Model CD Pipeline Tutorial [Online]:

https://github.com/sohiniroych/Flask-ML-Pipeline_GCP-Tutorial/

[2] K. Veerakumar. Startup — Multiple Linear Regression.

[Online]: <https://www.kaggle.com/karthickveerakumar/startup-logistic-regression>

[3] L. Li. Classification and Regression Analysis with Decision Trees [Online]:

[Classification and Regression Analysis with Decision Trees | by Lorraine Li | Towards Data Science](#)

[4] A. Jaleel. Startup:REST API with Flask [Online]:

<https://www.kaggle.com/ahammedjaleel/startup-rest-api-with-flask>

[5] M. Grinberg. Designing a RESTful API with Python and Flask [Online]:

<https://blog.miguelgrinberg.com/post/designing-a-restful-api-with-python-and-flask>

[6] Google Cloud. Trouble shooting. [Online]

<https://cloud.google.com/run/docs/troubleshooting>

[7] J. Araujo. Machine Learning deployment pipeline on Google Cloud Run [Online]:

<https://github.com/jgvaraujo/ml-deployment-on-gcloud>

[8] Google Cloud. app.yaml Reference [Online]:

<https://cloud.google.com/appengine/docs/standard/python/config/appref>

Thanks to Anne Bonner.

Sign up for The Variable

Open in app



and cutting-edge research to original features you don't want to miss. [Take a look.](#)

Get this newsletter

Emails will be sent to alessio.tamburro@gmail.com.

[Not you?](#)

Google Cloud Run

Continuous Deployment

Editors Pick

ML Deployment On Gcp

Hands On Tutorials

[About](#) [Help](#) [Legal](#)

Get the Medium app

