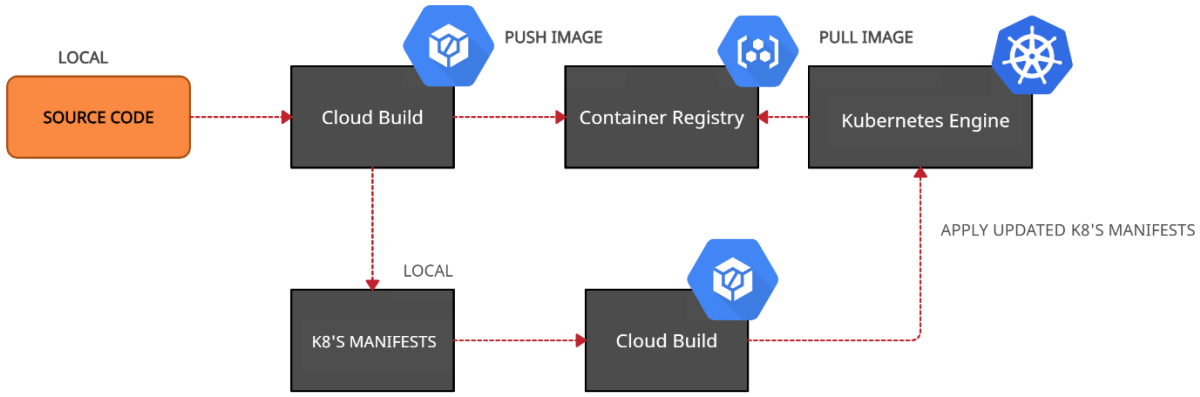
GCP DEVOPS PROJECT

Problem Statement: Create a sample application using any technology along with microservices to demonstrate implementation of the DevOps principle using GCP services. Establish a pipeline for continuous integration, continuous testing, and continuous deployment.



Motivation: As with growth of Machine Learning, most of the organization are shifting their whole ML workloads to cloud platform so that they can achieve High Availability and utilize high performance, As there are a lot of cloud platforms are available to deploy our whole ML models. Some are AWS, GCP, Azure. For this Project I have selected GCP as one of the cloud platforms.

Application: A simple web app with minimal framework using PyTorch and Streamlit to showcase an image classification model, where user will input any kind of image and just wait for the results which can be use as fun quizzes .

5 predictions should be outputted from highest probability to lowest probability.

We can input any image from our local system and it will give us the 5 closest answers to the image ranking from most matching probability to least probability.

```python
"""Create an Image Classification Web App using PyTorch and Streamlit."""
# import libraries
from PIL import Image
from torchvision import models, transforms
import torch
import streamlit as st

# set title of app
st.title("Simple Image Classification Application")
st.write("")

# enable users to upload images for the model to make predictions
file_up = st.file_uploader("Upload an image", type = "jpg")


def predict(image):
    """Return top 5 predictions ranked by highest probability.

    Parameters
    ----------
    :param image: uploaded image
    :type image: jpg
    :rtype: List
    :return: top 5 predictions ranked by highest probability
    """
    # create a ResNet model
    resnet = models.resnet101(pretrained = True)

    # transform the input image through resizing, normalization
    transform = transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize(
            mean = [0.485, 0.456, 0.406],
            std = [0.229, 0.224, 0.225]
            )])

    # load the image, pre-process it, and make predictions
    img = Image.open(image)
    batch_t = torch.unsqueeze(transform(img), 0)
    resnet.eval()
    out = resnet(batch_t)

    with open('imagenet_classes.txt') as f:
        classes = [line.strip() for line in f.readlines()]

    # return the top 5 predictions ranked by highest probabilities
    prob = torch.nn.functional.softmax(out, dim = 1)[0] * 100
    _, indices = torch.sort(out, descending = True)
    return [(classes[idx], prob[idx].item()) for idx in indices[0][:5]]


if file_up is not None:
    # display image that user uploaded
    image = Image.open(file_up)
    st.image(image, caption = 'Uploaded Image.', use_column_width = True)
    st.write("")
    st.write("Just a second ...")
    labels = predict(file_up)

    # print out the top 5 prediction labels with scores
    for i in labels:
        st.write("Prediction (index, name)", i[0], ",   Score: ", i[1])

```

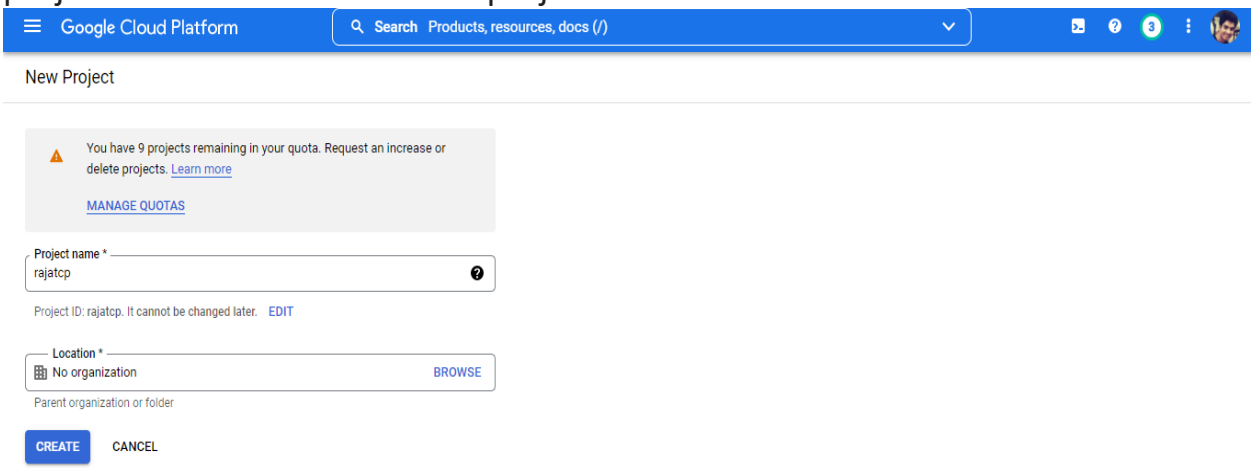
Introduction

Once we have developed and deployed the application/product, It needs to be continuously updated based on user feedback or the addition of new features. This process should be automated, as without automation we have to run the same development and deployment steps/commands again and again for every change to the application.

With Continuous Integration and Continuous Delivery Pipeline, we can automate the complete workflow from building, testing, packaging, and deploying, which will be triggered when there are any changes to an existing application or we can say if there is any new commit to an existing code repository.

Project Setup

Once the account is created, on the home page there is an option to create a project. Go ahead create a new project.



I have created a project, **rajatcp** so I will use that for setting up a complete pipeline.

Container Registry

It is used to store Docker Container images, Similar to DockerHub, AWS ECS, and other private cloud container registries.

Benefits of Container Registry

- Secure, Private Docker Registry
- Build and deploy automatically
- In-depth vulnerability scanning
- Lockdown risky images
- Native Docker support
- Fast, high-availability access

To use the service, we need to activate the API, search for **Container registry** in the search bar, and on the next page click on **Enable Container Registry API** button.

Google Kubernetes Engine (GKE)

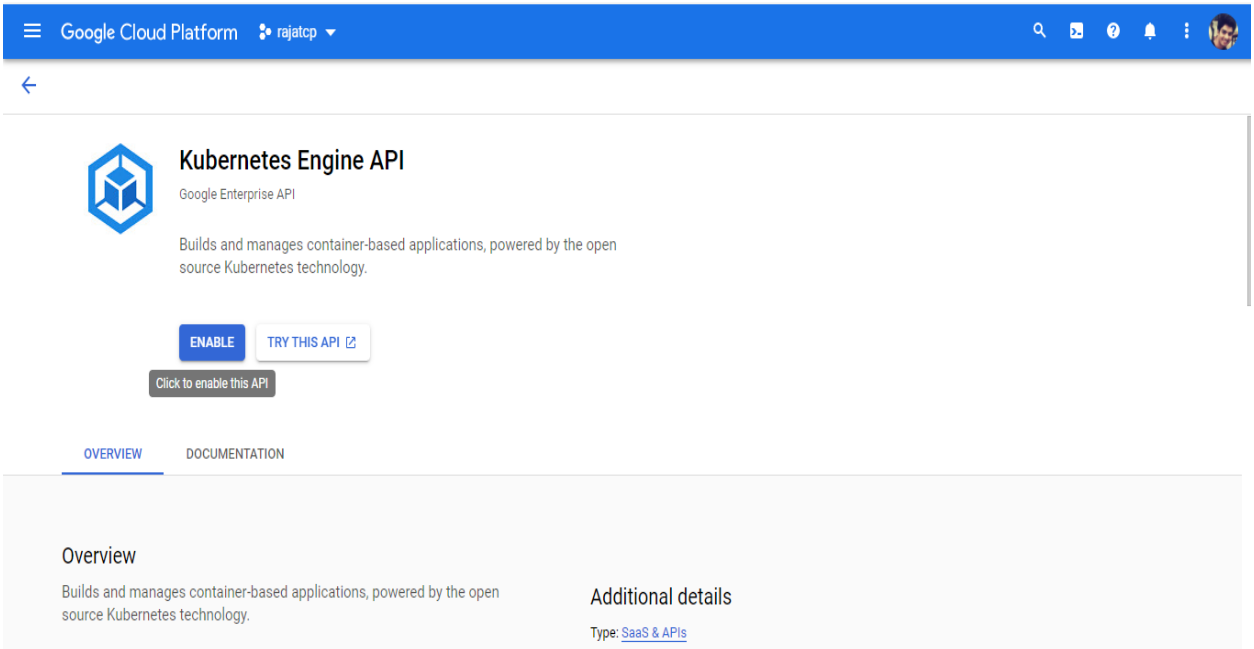
Kubernetes, also known as K8s, is an open-source system for automating deployment, scaling, and management of containerized applications.

GKE is Kubernetes managed by Google infrastructure. Some of the benefits are:

- Auto Scale
- Auto Upgrade
- Auto Repair
- Logging and Monitoring
- Load Balancing

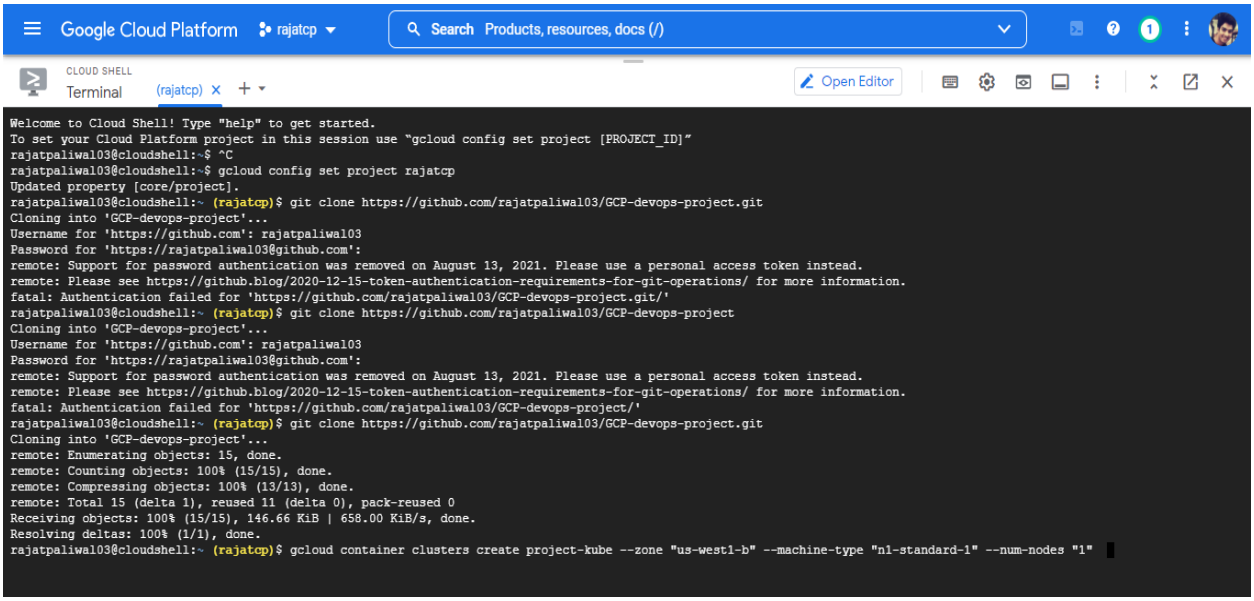
Now to use the service, we need to enable the API. Search for Kubernetes Engine API in the search bar and enable the API.

Once the API is enabled, we need to create a cluster. For that open cloud shell, you will find an icon on the top right side for Activate Cloud Shell.



Now to create a cluster, enter the following command in the cloud shell :

```
 gcloud container clusters create project-kube --zone "us-west1-b" --
machine-type "n1-standard-1" --num-nodes "1"
```



It will create a Kubernetes cluster, '**project-kube** 'with 1 compute node of n1-standard-1 machine type.

Now we have to create the following configuration files for our pipeline:

Dockerfile: A simple file that consists of instructions to build a Docker Image. Each instruction in a docker file is a command/operation, for example, what operating system to use, what dependencies to install or how to compile the code, and many such instructions which act as a layer.

```
# python
FROM python:3.8.5

RUN apt-get update

# Copy local code to the container image.
ENV APP_HOME /app
WORKDIR $APP_HOME
COPY . ./

RUN ls -la $APP_HOME/

# Install dependencies
RUN pip install -r requirements.txt

# Run the streamlit on container startup
CMD [ "streamlit", "run","--server.enableCORS","false","streamlit_ui.py" ]

```

Deployment YAML: To run an application we need to create a Deployment object and we can do that using a YAML

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: imgclass
spec:
  replicas: 2
  selector:
    matchLabels:
      app: imageclassifier
  template:
    metadata:
      labels:
        app: imageclassifier
    spec:
      containers:
      - name: cv-app
        image: gcr.io/rajatcp/streamlit_ui:v1
        ports:
        - containerPort: 8501
```

- apiVersion: Version of Kubernetes API
- kind: Kind of object to create, here its Deployment
- metadata: Data about objects to identify them
- spec: Specifications of the object, it includes replicas(no. of pods), labels, container image

Service YAML: To expose an application running on a set of Pods as a network service we need a Service YAML file.

In this file, we have specified kind: Service, in spec type: LoadBalancer to automatically distribute the load, and app name is same as in Deployment YAML file. It has port mapping which targets container port 8501.

CloudBuild

Cloud Build is a service that executes your builds on Google Cloud Platform's infrastructure.

Cloud Build can import source code from a variety of repositories or cloud storage spaces, execute a build to your specifications, and produce artifacts such as Docker containers or Java archives.

It executes the commands in steps and is similar to executing commands in a script.

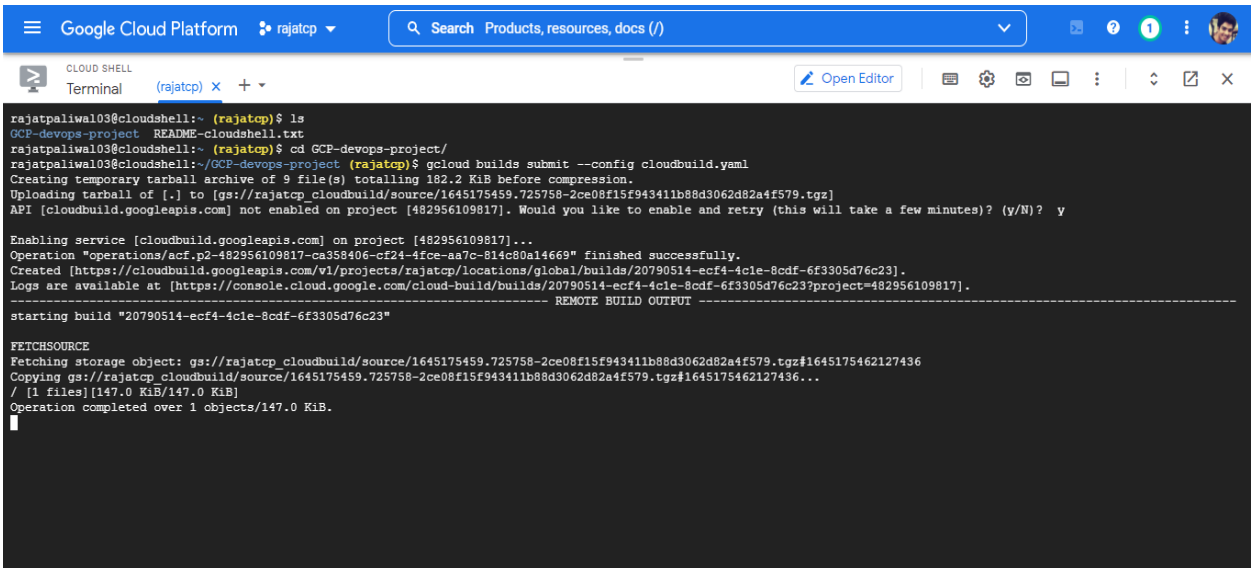
```yaml
steps:
- name: 'gcr.io/cloud-builders/docker'
  args: ['build', '-t', 'gcr.io/$PROJECT_ID/streamlit_ui:v1', '.']
  timeout: 180s
- name: 'gcr.io/cloud-builders/docker'
  args: ['push', 'gcr.io/$PROJECT_ID/streamlit_ui:v1']
- name: 'gcr.io/cloud-builders/gcloud'
  entrypoint: "bash"
  args:
  - "-c"
  - |
    echo "Docker Container Built"

- name: "gcr.io/cloud-builders/gke-deploy"
  args:
  - run
  - --filename=k8s/
  - --location=us-west1-b
  - --cluster=project-kube
  - --timeout=500s
```

In the first step, we will build the Docker image and in the next step, we will push the image to Google Container Registry. The final step is to deploy the application on the Kubernetes cluster, filename is the folder directory that will have

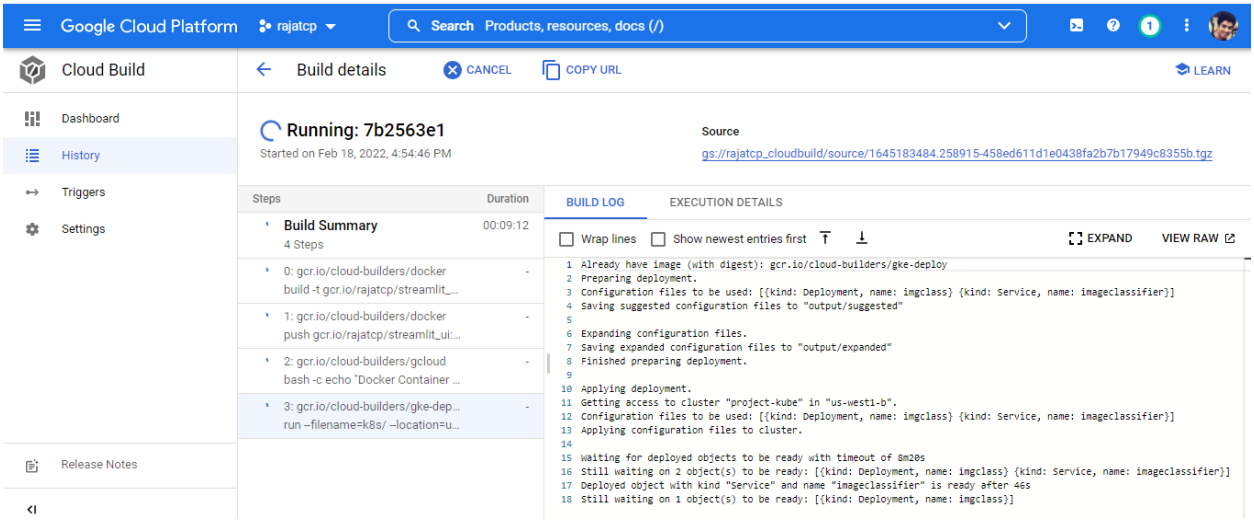
Deployment and Service YAML files, specify the image and cluster name that we have created earlier.

KickStart Pipeline:

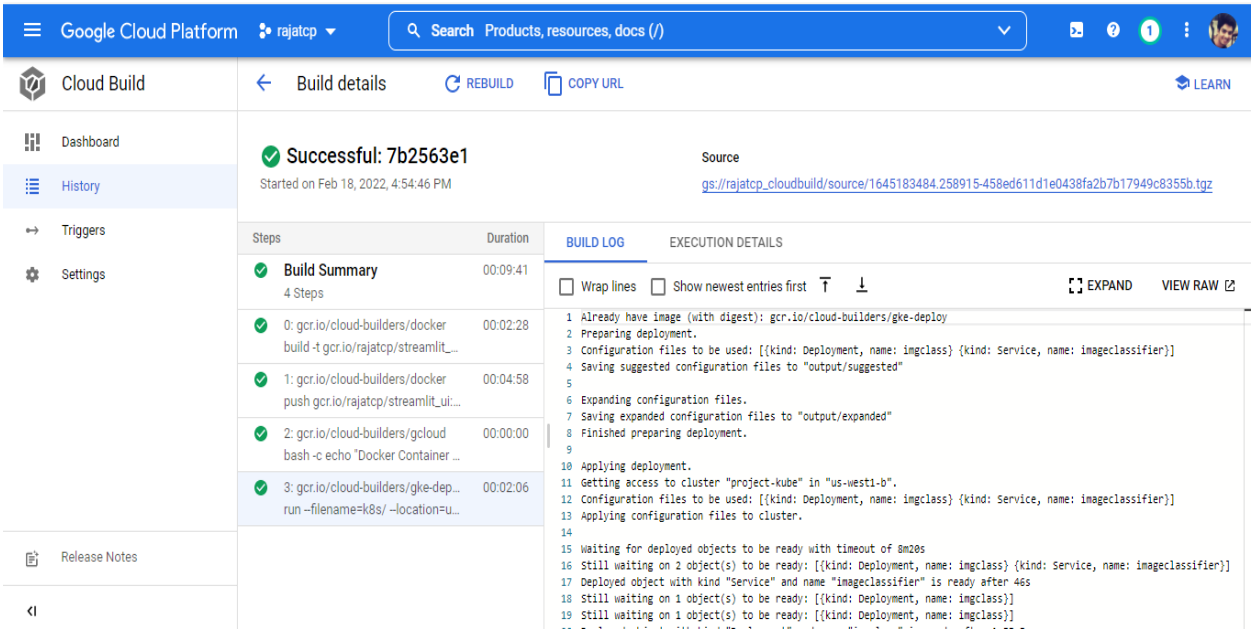
```
gcloud builds submit –config cloudbuild.yaml
```

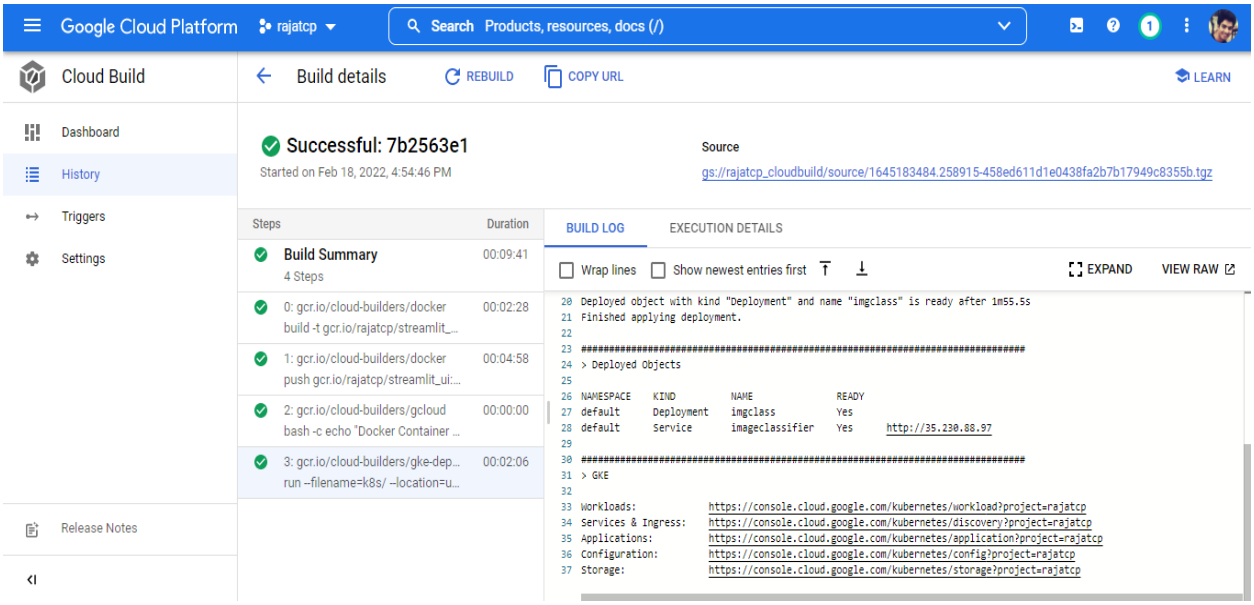


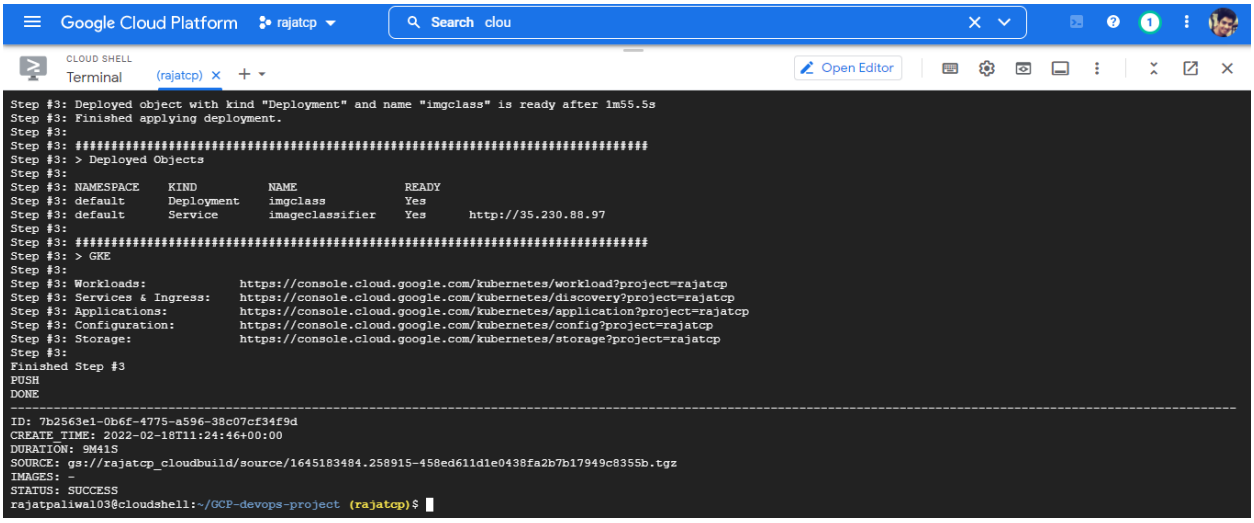
Building start:

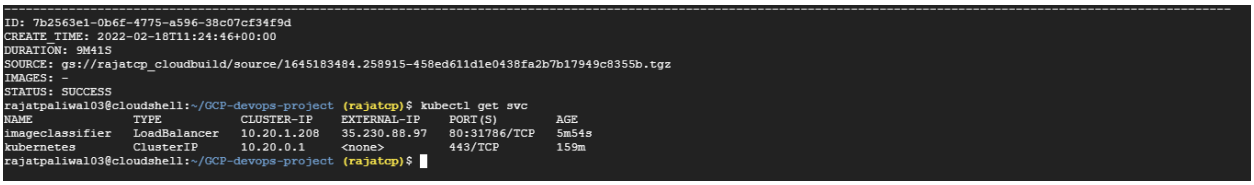


Building successful:

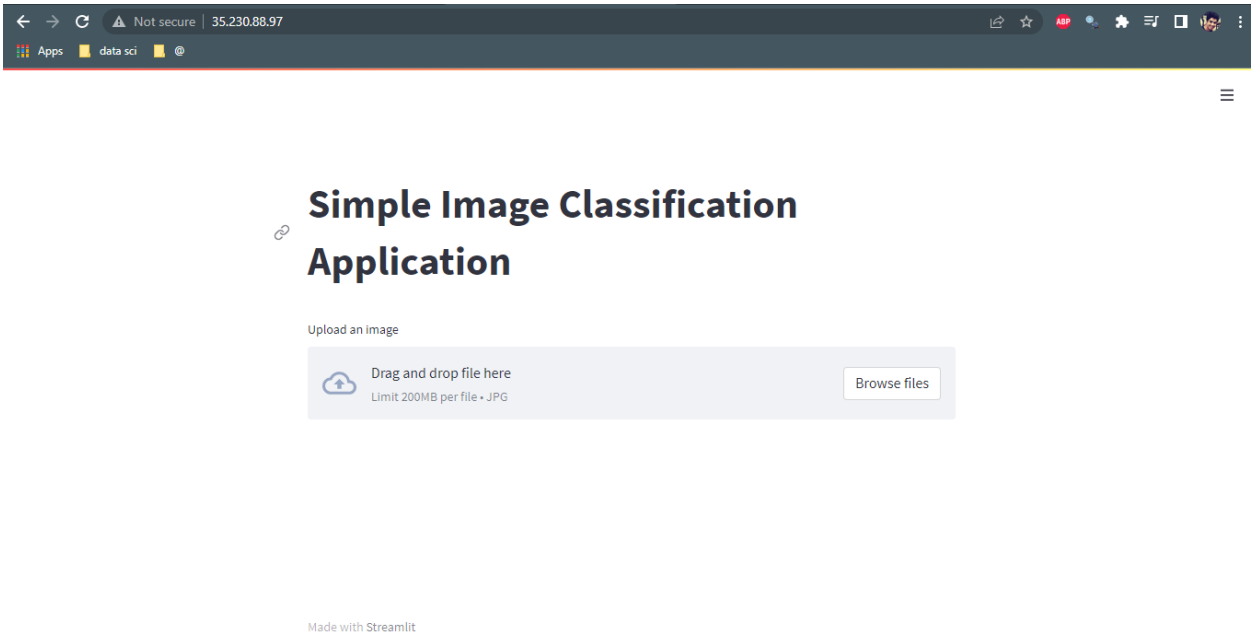




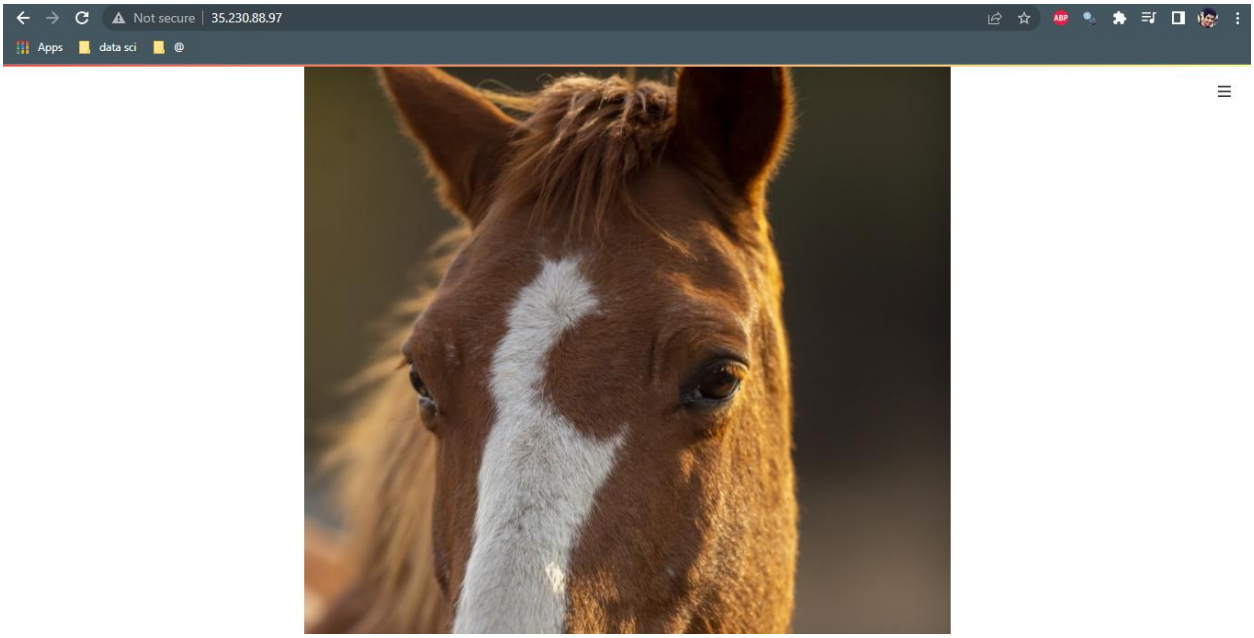




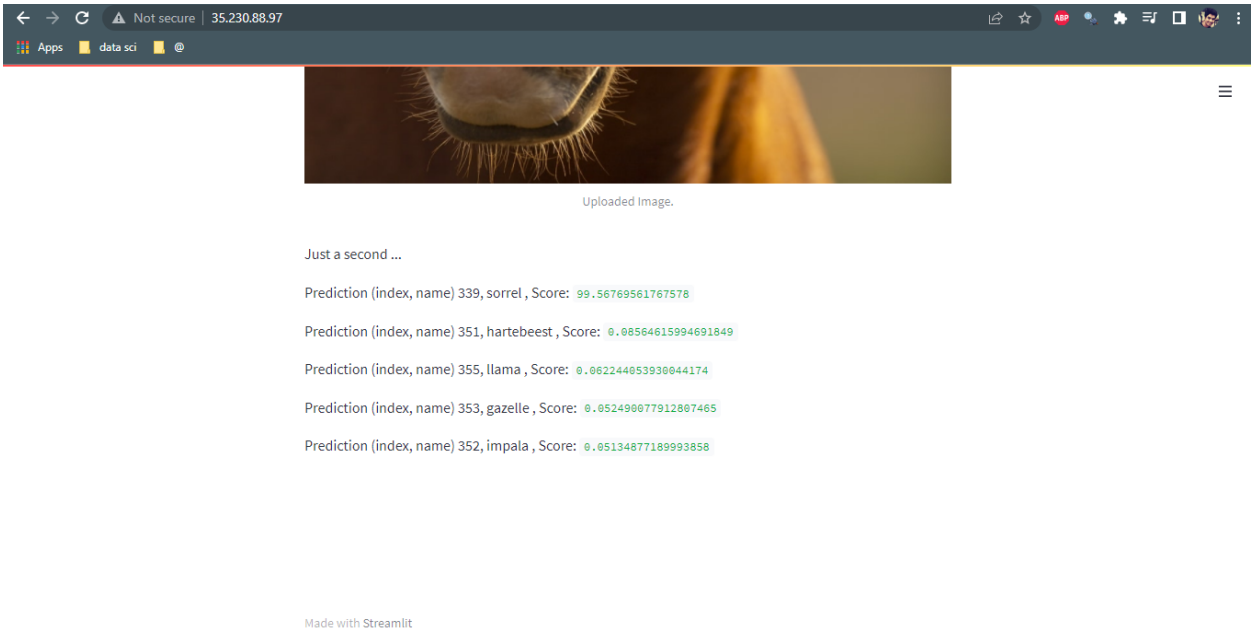
Now we can go to the endpoint and check our application.



Let’s try to input any image in that:

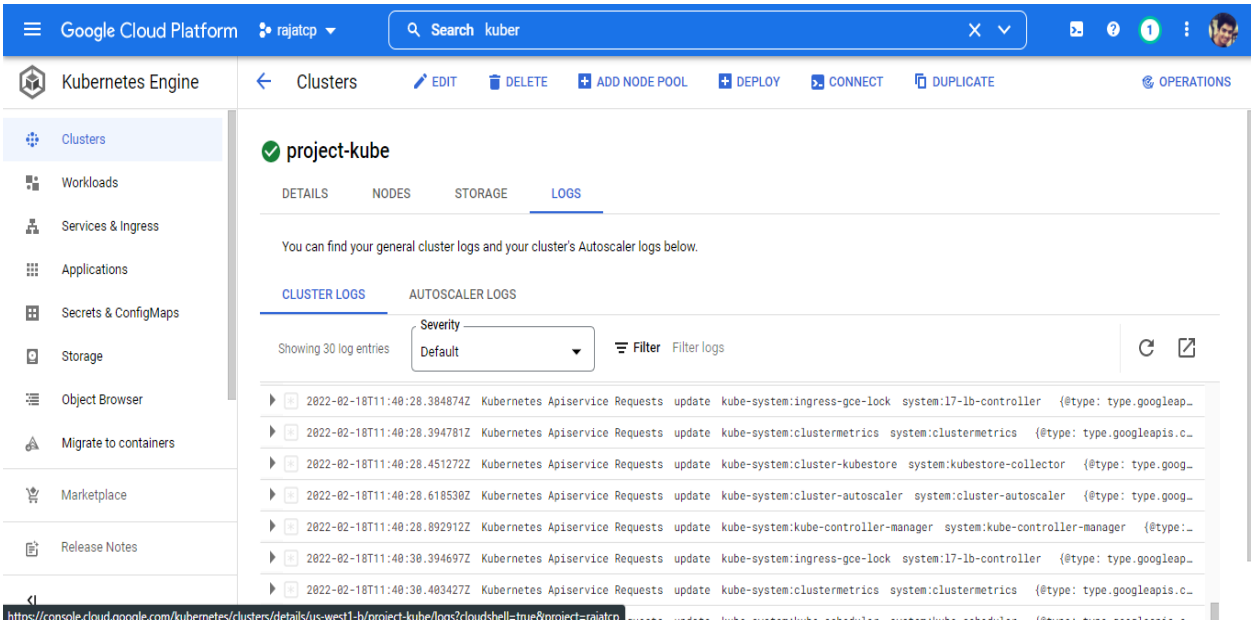


Results:



Top result give us the horse type as “sorrel” with matching probability of 99.56%.

For checking cluster logs :

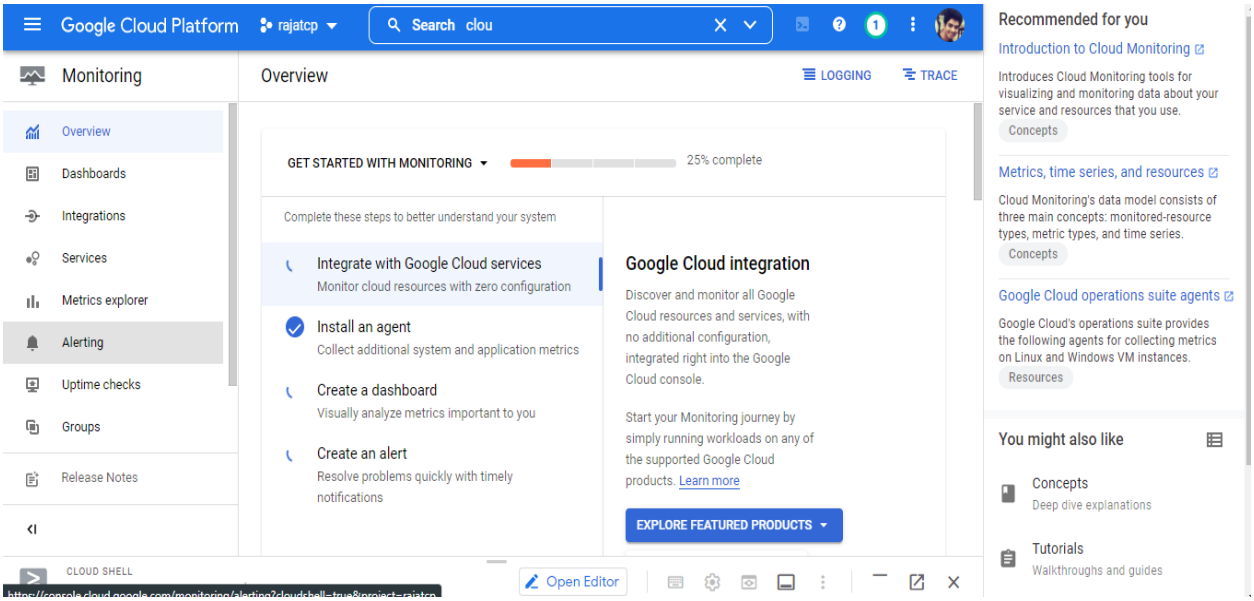


Monitoring:

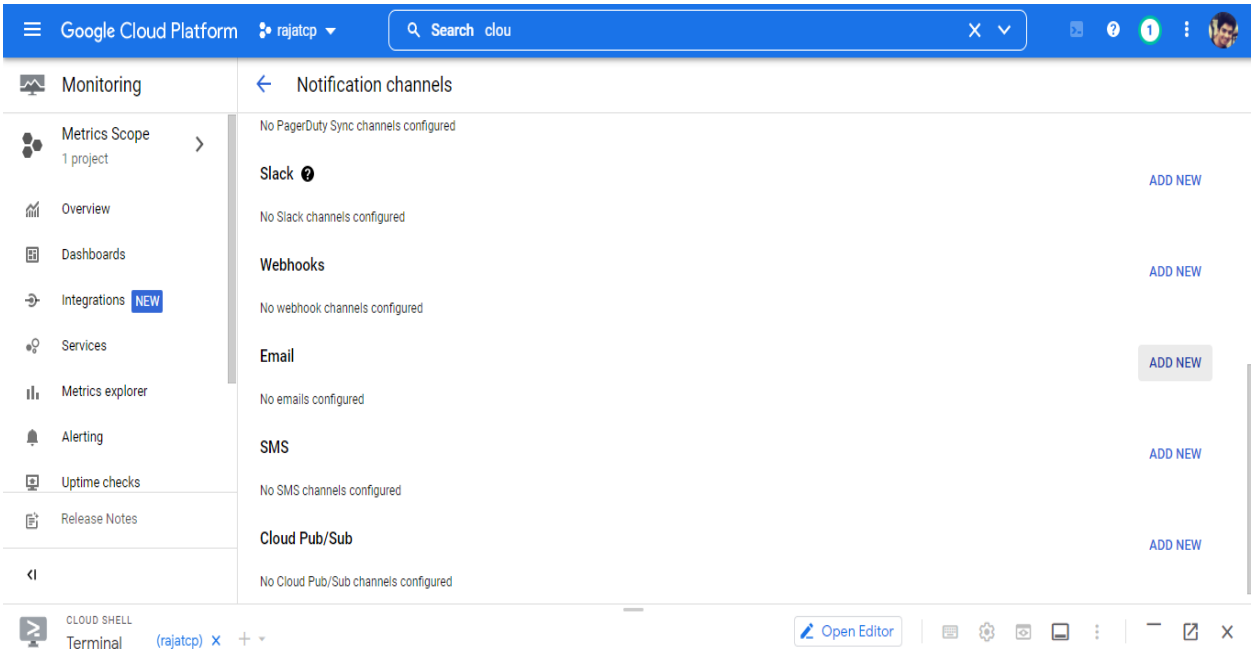
To add an email notification channel, do the following:

1) In the Cloud Console, select **Monitoring**
2) Click **Alerting** and then click **Edit notification channels**.
3) In the **Email** section, click **Add new**.
4) Complete the dialog and click **Save**.

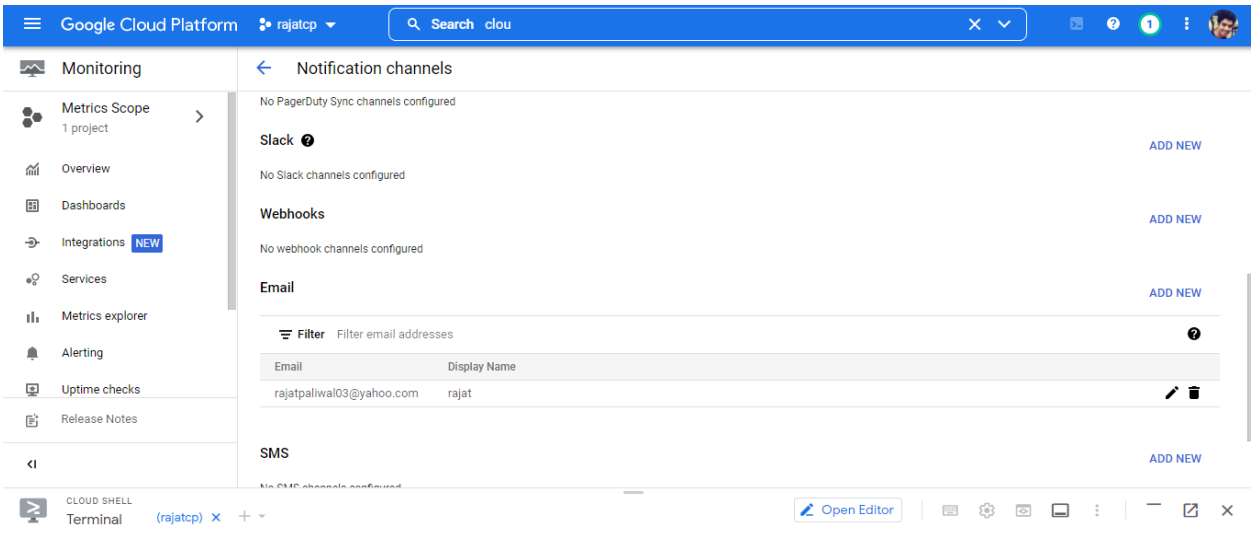
Step 1:


Google Cloud Platform
rajatcp
Search clou
Monitoring
Overview
LOGGING
TRACE
Overview
Dashboards
Integrations
Services
Metrics explorer
Alerting
Uptime checks
Groups
Release Notes
GET STARTED WITH MONITORING
25% complete
Complete these steps to better understand your system
Integrate with Google Cloud services
Monitor cloud resources with zero configuration
Install an agent
Collect additional system and application metrics
Create a dashboard
Visually analyze metrics important to you
Create an alert
Resolve problems quickly with timely notifications
Google Cloud integration
Discover and monitor all Google Cloud resources and services, with no additional configuration, integrated right into the Google Cloud console.
Start your Monitoring journey by simply running workloads on any of the supported Google Cloud products. Learn more
EXPLORE FEATURED PRODUCTS
Recommended for you
Introduction to Cloud Monitoring
Introduces Cloud Monitoring tools for visualizing and monitoring data about your service and resources that you use.
Concepts
Metrics, time series, and resources
Cloud Monitoring's data model consists of three main concepts: monitored-resource types, metric types, and time series.
Concepts
Google Cloud operations suite agents
Google Cloud's operations suite provides the following agents for collecting metrics on Linux and Windows VM instances.
Resources
You might also like
Concepts
Deep dive explanations
Tutorials
Walkthroughs and guides
CLOUD SHELL
Open Editor

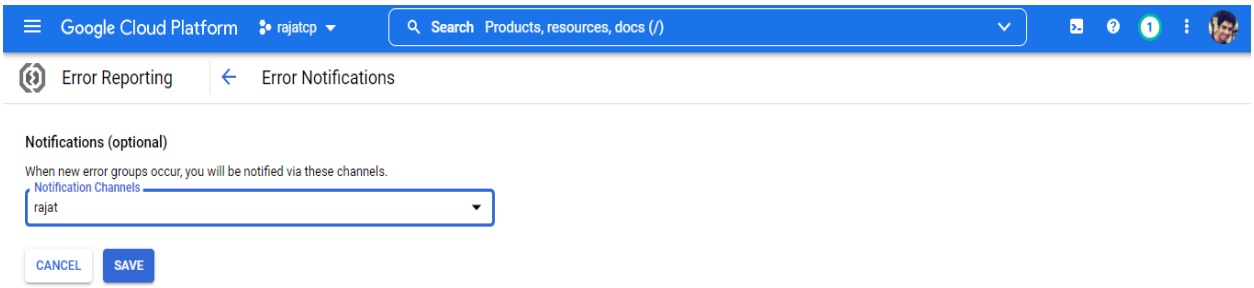
Step 2:


Google Cloud Platform
rajatcp
Search clou
Monitoring
Notification channels
Metrics Scope
1 project
Overview
Dashboards
Integrations NEW
Services
Metrics explorer
Alerting
Uptime checks
Release Notes
No PagerDuty Sync channels configured
Slack
ADD NEW
No Slack channels configured
Webhooks
ADD NEW
No webhook channels configured
Email
ADD NEW
No emails configured
SMS
ADD NEW
No SMS channels configured
Cloud Pub/Sub
ADD NEW
No Cloud Pub/Sub channels configured
CLOUD SHELL
Terminal
(rajatcp)
Open Editor

Step 3:



Step 4:



Complete the dialog and click **Save**.