

“Kazakh-British Technical University” JSC

School of Information Technology and Engineering

Report : Cloud application development

**Midterm Project: Deploying a Scalable Web Application
on Google Cloud Platform**

Student Eleu Ernur Bakytzhanuly 21B031072

Table of Contents

1. **Executive Summary**
2. **Introduction**
3. **Project Objectives**
4. **Google Cloud Platform Overview**
5. **Google Cloud SDK and Cloud Shell**
6. **Google App Engine**
7. **Building with Google Cloud Functions**
8. **Containerizing Applications**
9. **Managing APIs with Google Cloud Endpoints**
10. **Testing and Quality Assurance**
11. **Monitoring and Maintenance**
12. **Challenges and Solutions**
13. **Conclusion**
14. **References**
15. **Appendices**

1.Executive Summary

The goal of this project was to create a scalable and resilient web-based To-Do List application that utilized modern cloud technology. I built the app with Flask and placed it on Google Cloud Platform (GCP), specifically App Engine, to ensure that it could handle large traffic seamlessly. Additionally, I used Google Cloud Functions to automate background chores like as form submissions and notifications, ensuring that backend processes run smoothly.

To increase the app's scalability, I Dockerized it and placed it on Google Kubernetes Engine (GKE), allowing it to handle heavy traffic without interruption. Security was another key concern. I secured the API with Google Cloud Endpoints and implemented access control with API keys. I also configured Google Cloud Monitoring and Logging to track performance and issues in real time, ensuring that the app operates smoothly.

Key project achievements include:

- Launched a scalable and reliable To-Do List app.
- Automating background operations with serverless functions.
- Efficient resource management through full containerization.
- Enhancing monitoring to ensure excellent performance.

2.Introduction

For this project, I used GCP as the cloud infrastructure to launch the scalable To-Do List app. GCP's scalability and auto-scaling technologies were critical in ensuring that the app could handle variable traffic levels without experiencing performance difficulties. I was also able to keep prices low by employing a pay-as-you-go strategy, paying for resources only when they were needed.

The flexibility of GCP enabled me to deploy updates rapidly and grow resources as needed. Features such as Cloud Functions for serverless computing and GKE for container orchestration enabled me to explore and optimize performance.

Security was a primary issue, and GCP's built-in capabilities such as encryption and identity management gave the tools required to keep sensitive data secure.

3.Project Objectives

The purpose of this project was to leverage GCP to create and deploy a scalable, secure, and user-friendly online application. Here are the primary objectives I wanted to achieve:

- User-Friendly To-Do List App: I created the app with Flask, focusing on developing a responsive interface that works on all devices (desktop, tablet, and smartphone).
- Used Google Cloud Functions to automate backend activities such as form submissions and alerts, reducing human infrastructure maintenance.
- High Availability and Scalability: By deploying on Google App Engine and using GKE for containerized deployments, the app can automatically scale to accommodate high traffic without performance issues.

- I secured the app's APIs with Google Cloud Endpoints and enabled API key authentication for secure external communication.
- I used Google Cloud Monitoring and Logging to monitor the app's health and performance, creating custom dashboards and alert policies to address possible issues quickly.

4. Google Cloud Platform. Overview

GCP provided the tools and infrastructure I needed to develop and launch the To-Do List app. I used numerous critical GCP components, including:

- Used App Engine for automatic scaling and GKE to manage containerized apps in the compute services area.
- Utilized Cloud Storage and SQL to efficiently manage huge datasets and transactional data.
- Implemented Cloud Load Balancing and VPC for secure and efficient traffic delivery.
- Implemented DevOps tools, including Cloud Build for CI/CD and Cloud Functions for automation of background chores.

5. Set up Google Cloud SDK and Cloud Shell.

To manage cloud resources more efficiently, I installed the Google Cloud SDK, which allows me to connect with GCP services directly from the command line. I also utilized Cloud Shell extensively to manage and deploy the application, which made chores like API configuration and Kubernetes cluster management easier.

Cloud Shell's built-in Git feature simplified version management, allowing me to grab and push code directly from the cloud. This configuration was critical for sustaining seamless development workflows and monitoring app performance in real time.

6. Google App Engine

Application Development

The project entailed creating a To-Do List web application with the Flask framework (Python). The application allows users to easily create, manage, and delete tasks. The application's key functions are task management, which allows users to add new tasks, browse their list, and delete completed tasks.

- The application interface is responsive and optimized for desktop and mobile devices, providing a uniform experience across all platforms.
- The application's serverless architecture optimizes resource management by using Google Cloud Functions for background operations like form submissions and notifications.

Flask was chosen for its simplicity, lightweight structure, and compatibility with cloud systems. Here is an excerpt of the main application code:

```
yernur_y@DESKTOP-2PVM7F3: ~/todo-list-app
GNU nano 6.2 main.py
import requests
from flask import Flask, request, render_template, redirect, url_for

app = Flask(__name__)

tasks = []
CLOUD_FUNCTION_URL = 'https://us-central1-savvy-hull-436617-b1.cloudfunctions.net/process_form_data'

@app.route('/')
def index():
    return render_template('index.html', tasks=tasks)

@app.route('/add', methods=['POST'])
def add_task():
    task = request.form.get('task')
    if task:
        response = requests.post(CLOUD_FUNCTION_URL, json={'task': task})
        if response.status_code == 200:
            tasks.append(task)
        else:
            print(f"Error processing task: {response.status_code}, {response.text}")
    return redirect(url_for('index'))

@app.route('/delete/<int:task_id>')
def delete_task(task_id):
    if 0 <= task_id < len(tasks):
        tasks.pop(task_id)
    return redirect(url_for('index'))

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=8080, debug=True)
```

File index.html

```
yernur_y@DESKTOP-2PVM7F3: ~/todo-list-app/templates
GNU nano 6.2
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>To-Do List</title>
</head>
<body>
  <h1>To-Do List</h1>
  <form action="/add" method="POST">
    <input type="text" name="task" placeholder="Enter a new task" required>
    <button type="submit">Add Task</button>
  </form>

  <ul>
    {% for task in tasks %}
    <li>{{ task[1] }} <a href="/delete/{{ task[0] }}">Delete</a></li>
    {% endfor %}
  </ul>
</body>
</html>
```

Deployment to Google App Engine

Deploying the application on **Google App Engine** ensures that it can automatically scale based on the traffic load, maintaining high availability and performance without manual intervention.

Here is a step-by-step guide for deploying the Flask application to App Engine:

Preparing the App Engine configuration file (app.yaml): Create an app.yaml file in the root directory of your project. This file defines the runtime and entry point for the application.

```
C:\> yernur_y@DESKTOP-2PVM7F3: ~/todo-list-app
GNU nano 6.2
runtime: python39

handlers:
- url: /*
  script: auto
```

Deploying the Application: I deployed the application to Google App Engine.

gcloud app deploy

This command will package the application and deploy it to App Engine.

Verifying the Deployment: I can monitor the deployment status and logs using Google Cloud's monitoring tools.

gcloud app logs tail -s default

This command streams real-time logs from the deployed application, allowing I to check for any errors or issues during deployment or runtime.

```
yernur_y@DESKTOP-2PVM7F3:~/todo-list-app$ gcloud app deploy
Services to deploy:

descriptor:      [/home/yernur_y/todo-list-app/app.yaml]
source:          [/home/yernur_y/todo-list-app]
target project:  [savvy-hull-436617-b1]
target service:  [default]
target version:  [20241011t213419]
target url:      [https://savvy-hull-436617-b1.de.r.appspot.com]
target service account: [savvy-hull-436617-b1@appspot.gserviceaccount.com]

Do you want to continue (Y/n)? y

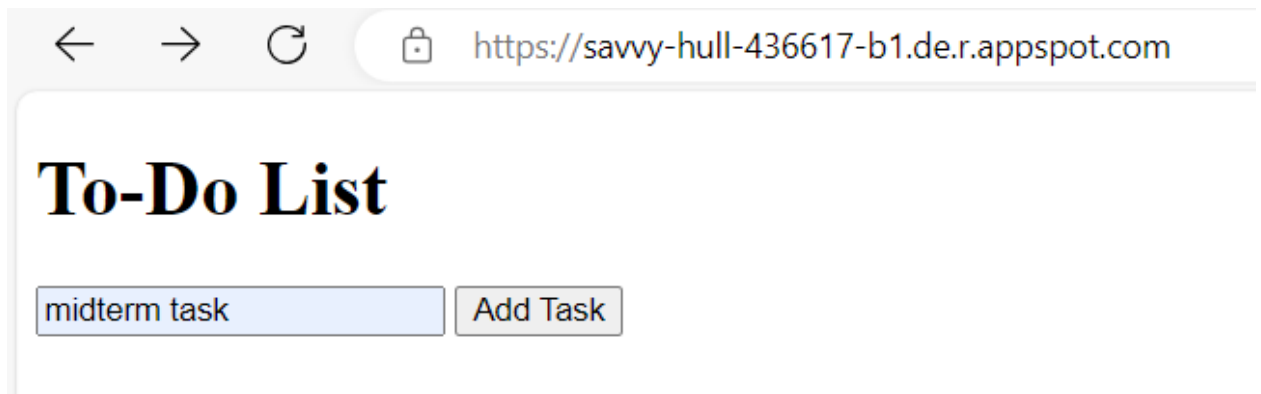
Beginning deployment of service [default]...

[= Uploading 9 files to Google Cloud Storage =]

File upload done.
Updating service [default]...done.
Setting traffic split for service [default]...done.
Deployed service [default] to [https://savvy-hull-436617-b1.de.r.appspot.com]

You can stream logs from the command line by running:
$ gcloud app logs tail -s default

To view your application in the web browser run:
$ gcloud app browse
```



To-Do List

- midterm task [Delete](#)
- do task [Delete](#)

File **requirement.txt**

 yernur_y@DESKTOP-2PVM7F3:

```
GNU nano 6.2
Flask==2.0.3
Werkzeug==2.0.3
gunicorn==21.2.0
blinker==1.8.2
click==8.1.7
Flask-SQLAlchemy==2.5.1
Flask-Migrate==3.1.0
```

7. Building with Google Cloud Functions

Google Cloud Functions is a fully managed serverless computing solution that enables developers to run event-driven code without concern for the underlying infrastructure. Cloud

Functions were used for a variety of activities in the To-Do List online application, including form submission handling, notification sending, and background processing. This resulted in a lightweight and efficient architecture, minimizing the need for manual server management.

Use Case: Processing Task Submissions.

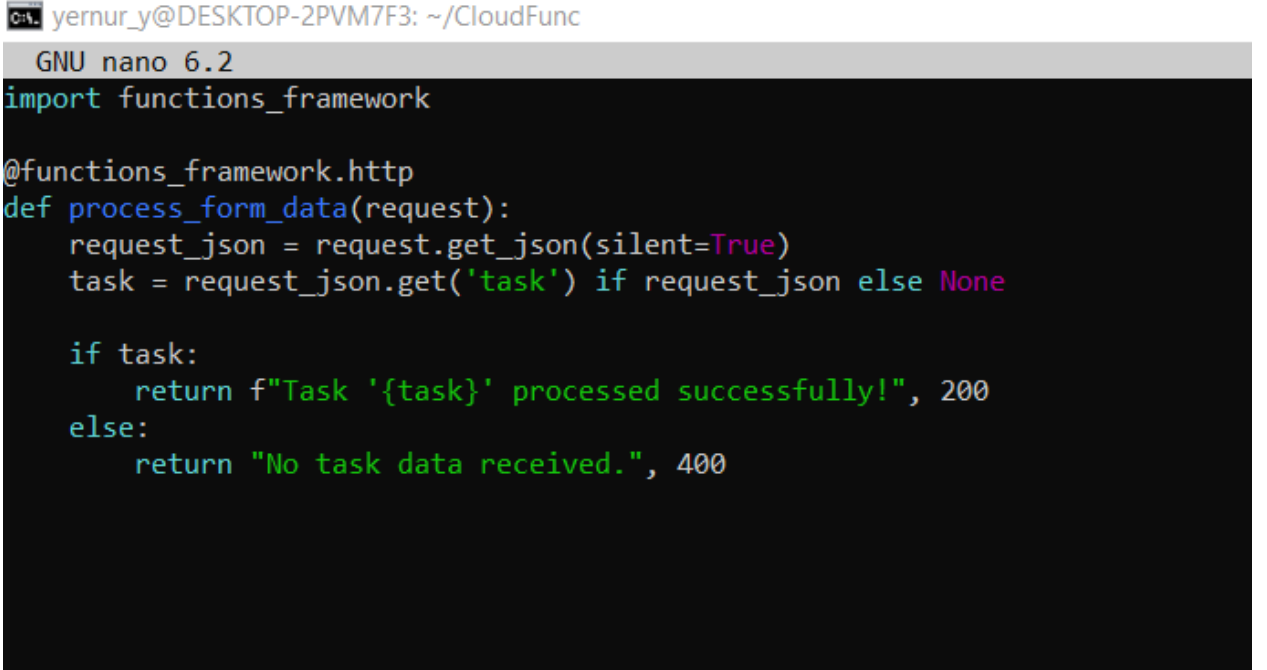
The project's major use case for Cloud Functions was to process task submissions from the To-Do List application. Offloading these duties to Cloud Functions improved the application's scalability and efficiency, allowing background actions to be handled asynchronously.

Creating A Cloud Function

Here's a step-by-step breakdown of how the Cloud Functions were developed and incorporated into the application:

Writing the Cloud Function.

The Cloud Function shown below processes task submissions. This function accepts data, processes it (e.g., validation or formatting), and sends a confirmation message to the user.



```
yernur_y@DESKTOP-2PVM7F3: ~/CloudFunc
GNU nano 6.2
import functions_framework

@functions_framework.http
def process_form_data(request):
    request_json = request.get_json(silent=True)
    task = request_json.get('task') if request_json else None

    if task:
        return f"Task '{task}' processed successfully!", 200
    else:
        return "No task data received.", 400
```

This Cloud Function takes in a task submission via an HTTP POST request.

2. Deploying the Cloud Function

To deploy this Cloud Function, use the Google Cloud SDK. Run the following command:

```
gcloud functions deploy process_form_data \

--runtime python39 \

--trigger-http \

--allow-unauthenticated
```


--runtime python39 : Specifies the Python 3.9 runtime environment for the function.

--trigger-http: Defines the function as being triggered via HTTP requests.

--allow-unauthenticated: Allows unauthenticated access to the function, which is useful in public-facing web apps (can be restricted based on your app's requirements).

I use cURL to send a POST request with the data that will be processed by the function. For example, if your function accepts form data, it might look like this: *curl -X POST *

*https://REGION-PROJECT_ID.cloudfunctions.net/process_form_data *

*-H "Content-Type: application/json" *

-d '{"task": "Test task"}'

3. Integrating Cloud Function with the Main Application

In the main Flask web application, the Cloud Function is called to handle the task processing. Here's an example of how the Cloud Function is integrated:

```
yernur_y@DESKTOP-2PVM7F3: ~/todo-list-app
GNU nano 6.2 main.py
import requests
from flask import Flask, request, render_template, redirect, url_for

app = Flask(__name__)

tasks = []
CLOUD_FUNCTION_URL = 'https://us-central1-savvy-hull-436617-b1.cloudfunctions.net/process_form_data'

@app.route('/')
def index():
    return render_template('index.html', tasks=tasks)

@app.route('/add', methods=['POST'])
def add_task():
    task = request.form.get('task')
    if task:
        response = requests.post(CLOUD_FUNCTION_URL, json={'task': task})
        if response.status_code == 200:
            tasks.append(task)
        else:
            print(f"Error processing task: {response.status_code}, {response.text}")
    return redirect(url_for('index'))

@app.route('/delete/<int:task_id>')
def delete_task(task_id):
    if 0 <= task_id < len(tasks):
        tasks.pop(task_id)
    return redirect(url_for('index'))

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=8080, debug=True)
```

Use cURL to send a POST request with the data that will be processed by the function.

```
curl -X POST \
https://REGION-PROJECT_ID.cloudfunctions.net/process_form_data \
-H "Content-Type: application/json" \
-d '{"task": "Test task"}'
```

```
yernur_y@DESKTOP-2PVM7F3:~/CloudFunc$ curl -X POST \
> https://us-central1-savvy-hull-436617-b1.cloudfunctions.net/process_form_data \
> -H "Content-Type: application/json" \
> -d '{"task": "Test task"}'
Task 'Test task' processed successfully!yernur_y@DESKTOP-2PVM7F3:~/CloudFunc$
```

8. Containerizing Applications

Containerization is the process of packing an application and its dependencies into a container so that it may operate reliably across multiple computing environments. Docker, a popular containerization tool, makes it simple to package an application into lightweight, portable containers, allowing developers to focus on code while assuring that the program acts consistently regardless of where it runs.

In this project, Docker was used to containerize a Flask-based web application and deploy it to Google Kubernetes Engine (GKE), a managed Kubernetes service on Google Cloud that automates container orchestration and scaling.

Docker Overview

To containerize the application, the following stages were completed:

I created a Dockerfile. The Dockerfile is a script that includes instructions for creating a Docker image. It describes the base image, installs the necessary dependencies, copies the application code, and specifies the commands for running the app.

Here's the Dockerfile for the Flask web application:

```
yernur_y@DESKTOP-2PVM7F3: ~/todo-list-app
GNU nano 6.2
FROM python:3.9-slim

WORKDIR /app

COPY . /app

RUN pip install --no-cache-dir -r requirements.txt

EXPOSE 8080

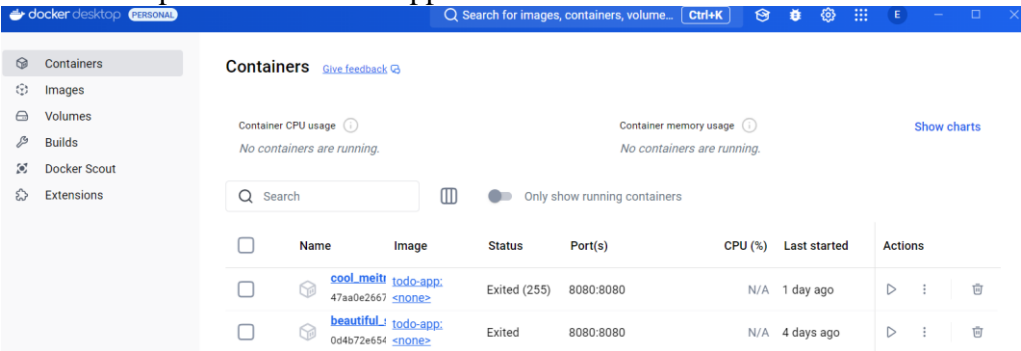
CMD ["python3", "main.py"]
```

I built the Docker image. After completing the Dockerfile, the Docker image was built. The image contains all of the dependencies and code necessary to run the application. To create the Docker image, run the following command:

```
docker build -t todo-app
```

Run the Container locally. To test the containerized application locally, the Docker image was executed using the following command:

```
docker run -p 8080:8080 todo-app
```



```

yernur_y@DESKTOP-2PVM7F3:~/todo-list-app$ nano Dockerfile
yernur_y@DESKTOP-2PVM7F3:~/todo-list-app$ touch Dockerfiles
yernur_y@DESKTOP-2PVM7F3:~/todo-list-app$ nano Dockerfiles
yernur_y@DESKTOP-2PVM7F3:~/todo-list-app$ nano requirements.txt
yernur_y@DESKTOP-2PVM7F3:~/todo-list-app$ docker build -t todo-app .

The command 'docker' could not be found in this WSL 2 distro.
We recommend to activate the WSL integration in Docker Desktop settings.

For details about using Docker Desktop with WSL 2, visit:
https://docs.docker.com/go/wsl2/

yernur_y@DESKTOP-2PVM7F3:~/todo-list-app$ docker build -t todo-app .
[+] Building 29.7s (10/10) FINISHED                                docker:default
=> [internal] load build definition from Dockerfile                0.0s
=> => transferring dockerfile: 181B                                0.0s
=> [internal] load metadata for docker.io/library/python:3.9-slim 2.9s
=> [auth] library/python:pull token for registry-1.docker.io      0.0s
=> [internal] load .dockerignore                                  0.0s
=> => transferring context: 2B                                       0.0s
=> [1/4] FROM docker.io/library/python:3.9-slim@sha256:49f94609e5a997dc16086a66ac 0.0s
=> [internal] load build context                                  0.0s
=> => transferring context: 3.85kB                                   0.0s
=> CACHED [2/4] WORKDIR /app                                       0.0s
=> [3/4] COPY . /app                                              0.0s
=> [4/4] RUN pip install --no-cache-dir -r requirements.txt       26.5s
=> exporting to image                                              0.2s
=> => exporting layers                                              0.2s
=> => writing image sha256:9a67a99a18f11bc1445d942c32363d14e6aba9d511f46098ac25f7 0.0s
=> => naming to docker.io/library/todo-app                        0.0s
yernur_y@DESKTOP-2PVM7F3:~/todo-list-app$ docker run -p 8080:8080 todo-app
* Serving Flask app 'main' (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: on
* Running on all addresses.
  WARNING: This is a development server. Do not use it in a production deployment.
* Running on http://172.17.0.2:8080/ (Press CTRL+C to quit)
* Restarting with stat
* Debugger is active!
* Debugger PIN: 305-063-919
172.17.0.1 - - [13/Oct/2024 18:31:37] "GET / HTTP/1.1" 200 -
172.17.0.1 - - [13/Oct/2024 18:31:38] "GET /favicon.ico HTTP/1.1" 404 -
172.17.0.1 - - [13/Oct/2024 18:31:48] "GET / HTTP/1.1" 200 -
172.17.0.1 - - [13/Oct/2024 18:31:58] "POST /add HTTP/1.1" 302 -
172.17.0.1 - - [13/Oct/2024 18:31:58] "GET / HTTP/1.1" 200 -

yernur_y@DESKTOP-2PVM7F3:~/todo-list-app$ docker run -p 8080:8080 todo-app
* Serving Flask app 'main' (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on all addresses.
  WARNING: This is a development server. Do not use it in a production deployment.
* Running on http://172.17.0.2:8080/ (Press CTRL+C to quit)

```

GKE Deployment

Once the Docker image had been successfully tested locally, the containerized application was deployed to Google Kubernetes Engine (GKE). The GKE cluster offers a scalable and manageable environment for running containerized applications.

Steps taken for deployment on GKE:

I push a Docker image to Google Container Registry. Before deploying to GKE, the Docker image must be saved in a container registry. Docker images on GCP are stored using Google Container Registry (GCR).

First, tag the Docker image for GCR.

```
docker tag todo-app gcr.io/savvy-hull-436617-b1/todo-app
```

Push the image to GCR:

```
docker push gcr.io/savvy-hull-436617-b1/todo-app
```

I create a GKE Cluster A GKE cluster is required to deploy and manage the containerized application. The following command creates a new GKE cluster with three nodes:

```
gcloud container clusters create todo-cluster \
  --zone us-central1-a \
  --num-nodes 3
```

After the cluster is created, configure kubectl to interact with the cluster:

```
gcloud container clusters get-credentials todo-cluster --zone us-central1-a
```

```
vernur_y@DESKTOP-2PVM7F3:~/todo-list-app$ gcloud container clusters create todo-cluster \
> --num-nodes 2
Note: The Kubelet readonly port (10255) is now deprecated. Please update your workloads to use the recommended alternatives. See https://cloud.google.com/kubernetes-engine/docs/how-to/kubelet-access for ways to check usage and for migration instructions.
Note: Your Pod address range ('--cluster-ipv4-cidr') can accommodate at most 1008 node(s).
Creating cluster todo-cluster in us-central1-a... Cluster is being health-checked (Kubernetes Control Plane is healthy)...done.
Created [https://container.googleapis.com/v1/projects/savvy-hull-436617-b1/zones/us-central1-a/clusters/todo-cluster].
To inspect the contents of your cluster, go to: https://console.cloud.google.com/kubernetes/workload/_gcloud/us-central1-a/todo-cluster?project=savvy-hull-436617-b1
CRITICAL: ACTION REQUIRED: gke-gcloud-auth-plugin, which is needed for continued use of kubectl, was not found or is not executable. Install it with: gcloud components install gke-gcloud-auth-plugin
kubeconfig entry generated for todo-cluster.
NAME          LOCATION    MASTER_VERSION  MASTER_IP      MACHINE_TYPE  NODE_VERSION    NUM_NODES  STATUS
todo-cluster  us-central1-a  1.30.5-gke.1014001  35.184.109.205  e2-medium    1.30.5-gke.1014001  2          RUNNING
vernur_y@DESKTOP-2PVM7F3:~/todo-list-app$ gcloud components install gke-gcloud-auth-plugin

Your current Google Cloud CLI version is: 496.0.0
Installing components from version: 496.0.0

These components will be installed.



| Name                                       | Version | Size    |
|--------------------------------------------|---------|---------|
| gke-gcloud-auth-plugin (Platform Specific) | 0.5.9   | 4.0 MiB |



For the latest full release notes, please visit:
https://cloud.google.com/sdk/release_notes

Once started, canceling this operation may leave your SDK installation in an inconsistent state.

Do you want to continue (Y/n)? y

Performing in place update...



|                                                         |  |
|---------------------------------------------------------|--|
| Downloading: gke-gcloud-auth-plugin                     |  |
| Downloading: gke-gcloud-auth-plugin (Platform Specific) |  |
| Installing: gke-gcloud-auth-plugin                      |  |
| Installing: gke-gcloud-auth-plugin (Platform Specific)  |  |



Performing post processing steps...done.
```

1. **I create Kubernetes Deployment** Kubernetes deployment manages the desired state of the application. A deployment is created to ensure that the desired number of container instances are running.

Create a YAML file (deployment.yaml) for the deployment:

```
yernur_y@DESKTOP-2PVM7F3: ~/todo-list-app
GNU nano 6.2
apiVersion: apps/v1
kind: Deployment
metadata:
  name: todo-list-app
spec:
  replicas: 3
  selector:
    matchLabels:
      app: todo-list-app
  template:
    metadata:
      labels:
        app: todo-list-app
    spec:
      containers:
      - name: todo-list-app
        image: gcr.io/your-project-id/your-app-name
        ports:
        - containerPort: 8080
```

Apply the deployment using kubectl:

```
kubectl apply -f deployment.yaml
```

Expose the Application The application needs to be exposed to the internet to allow external access. A Kubernetes service is created to expose the deployment to an external IP.

Create a service YAML file (service.yaml):

```
yernur_y@DESKTOP-2PVM7F3: ~/todo-list-app
GNU nano 6.2
apiVersion: v1
kind: Service
metadata:
  name: todo-list-service
spec:
  type: LoadBalancer
  selector:
    app: todo-list-app
  ports:
  - protocol: TCP
    port: 80
    targetPort: 8080
```

Apply the service:

```
kubectl apply -f service.yaml
```

Once the service is created, it provisions a public IP address to access the web application.

```

yernur_y@DESKTOP-2PVM7F3:~/todo-list-app$ nano Deployment.yaml
yernur_y@DESKTOP-2PVM7F3:~/todo-list-app$ nano kubernetes-deployment.yaml
yernur_y@DESKTOP-2PVM7F3:~/todo-list-app$ kubectl apply -f kubernetes-deployment.yaml
deployment.apps/todo-app configured
service/todo-list-service unchanged
yernur_y@DESKTOP-2PVM7F3:~/todo-list-app$ gcloud container images list
NAME
gcr.io/savvy-hull-436617-b1/todo-app
Only listing images in gcr.io/savvy-hull-436617-b1. Use --repository to list images in other repositories.
yernur_y@DESKTOP-2PVM7F3:~/todo-list-app$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
todo-app-744d8fdb6b-6t7h7           1/1     Running   0           27s
todo-app-744d8fdb6b-7dlrh           1/1     Running   0           18s
todo-app-744d8fdb6b-vssf1           1/1     Running   0           35s
yernur_y@DESKTOP-2PVM7F3:~/todo-list-app$ kubectl get services
NAME              TYPE          CLUSTER-IP    EXTERNAL-IP    PORT(S)          AGE
kubernetes        ClusterIP      34.118.224.1   <none>          443/TCP           20m
todo-list-service  LoadBalancer  34.118.232.12  34.123.76.88   80:31803/TCP      4m35s

```

5.Checked the deployment status

```

yernur_y@DESKTOP-2PVM7F3:~/todo-list-app$ kubectl get services
NAME              TYPE          CLUSTER-IP    EXTERNAL-IP    PORT(S)          AGE
kubernetes        ClusterIP      34.118.224.1   <none>          443/TCP           16m
todo-list-service  LoadBalancer  34.118.232.12  <pending>       80:31803/TCP      12s
yernur_y@DESKTOP-2PVM7F3:~/todo-list-app$ kubectl get pods
NAME                                READY   STATUS             RESTARTS   AGE
todo-app-656bdc8f9d-p6656           0/1     InvalidImageName    0           40s
todo-app-656bdc8f9d-qd6r7           0/1     InvalidImageName    0           40s
todo-app-656bdc8f9d-x7ggq           0/1     InvalidImageName    0           40s
yernur_y@DESKTOP-2PVM7F3:~/todo-list-app$
yernur_y@DESKTOP-2PVM7F3:~/todo-list-app$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
todo-app-744d8fdb6b-6t7h7           1/1     Running   0           27s
todo-app-744d8fdb6b-7dlrh           1/1     Running   0           18s
todo-app-744d8fdb6b-vssf1           1/1     Running   0           35s
yernur_y@DESKTOP-2PVM7F3:~/todo-list-app$ kubectl get services
NAME              TYPE          CLUSTER-IP    EXTERNAL-IP    PORT(S)          AGE
kubernetes        ClusterIP      34.118.224.1   <none>          443/TCP           20m
todo-list-service  LoadBalancer  34.118.232.12  34.123.76.88   80:31803/TCP      4m35s
yernur_y@DESKTOP-2PVM7F3:~/todo-list-app$

```

9.Managing APIs with Google Cloud Endpoints

Creating an OpenAPI configuration file

First,I create an Open API configuration file (for example, openapi.yaml) that will describe the API of your Flash application.

```
yernur_y@DESKTOP-2PVM7F3: ~/todo-list-app
GNU nano 6.2
openapi: 3.0.0
info:
  title: Todo API
  description: API for managing tasks
  version: 1.0.0
servers:
  - url: https://savvy-hull-436617-b1.cloudfunctions.net
paths:
  /tasks:
    get:
      summary: Get all tasks
      operationId: getTasks
      responses:
        '200':
          description: Successful response
  /tasks/{task_id}:
    delete:
      summary: Delete a task
      operationId: deleteTask
      parameters:
        - name: task_id
          in: path
          required: true
          schema:
            type: integer
      responses:
        '200':
          description: Task successfully deleted
components:
  securitySchemes:
    google_id_token:
      type: oauth2
      flows:
        implicit:
          authorizationUrl: ''
          scopes: []
security:
  - google_id_token: []
```


YAML Lint

Paste in your YAML and click "Go" - we'll tell you if it's valid or not, and give you a nice clean UTF-8 v it.

```
1  ---
2  openapi: 3.0.0
3  info:
4    title: Hello World API
5    description: A simple API to return a greeting message
6    version: 1.0.0
7  paths:
8    /api/hello:
9      get:
10        summary: Returns a greeting message
11        responses:
12          "200":
13            description: A greeting message
14            content:
15              application/json:
16                schema:
17                  type: object
18                  properties:
19                    message:
20                      type: string
```

Go ☒ Reformat (strips comments) ☒ Resolve aliases

Valid YAML!

Endpoint Deployment

I have deployed the configuration in Google Cloud Endpoints:

```
gcloud endpoints services deploy openapi.yaml
```

Step 2: Implementing Authentication

To protect access to my API, I will set up authentication using Google Identity Token. This step allows you to make sure that only authorized users access the API.

Creating Authentication using Google ID Token

To do this, I need to set up authentication in the API configuration using the open api.yaml file. I am adding a section for Google ID Token:

components:

security Schemes:

google_id_token:

type: oauth2

flows:

implicit:

authorization Url: "

scopes: []

security:

- google_id_token: []

Protecting endpoints

Once the authentication system is set up, I can protect specific API routes. For example, to get a list of tasks, I add authentication to the route /tasks:

/tasks:

get:

summary: Get all tasks

security:

- google_id_token: []

responses:

'200':

description: Todo-list

Processing authentication tokens in the code

In my Flask application, I implement Google ID Token verification using the google-auth library. Sample code:

yernur_y@DESKTOP-2PVM7F3: ~/todo-list-app

GNU nano 6.2

```
from google.auth.transport import requests
from google.oauth2 import id_token
from flask import Flask, request, jsonify

app = Flask(__name__)

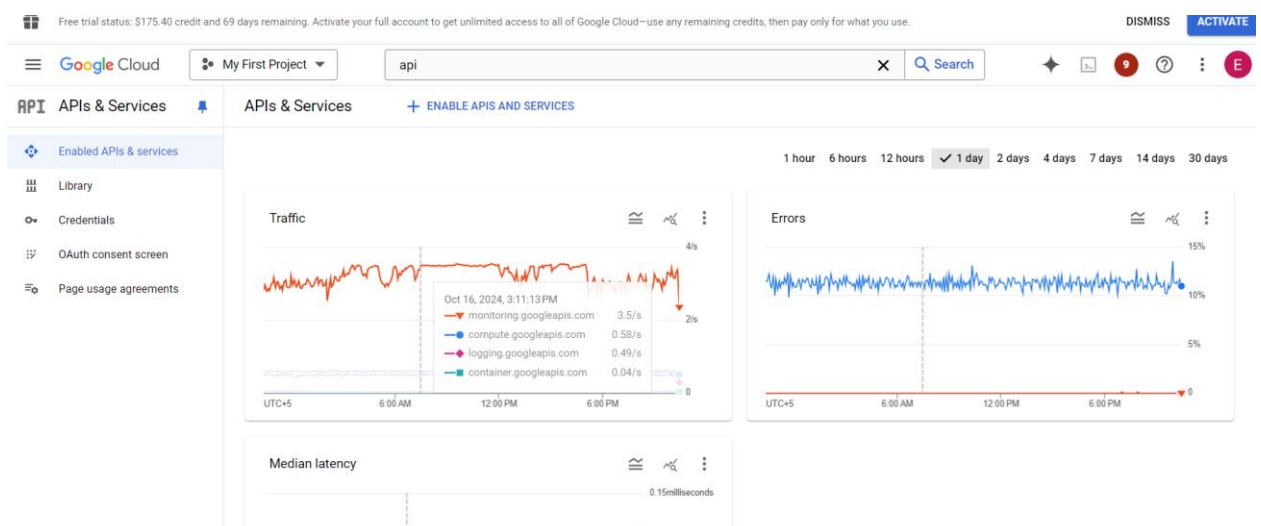
CLIENT_ID = '<savvy-hull-436617-b1>.apps.googleusercontent.com'

def verify_token(token):
    try:
        id_info = id_token.verify_oauth2_token(token, requests.Request(), CLIENT_ID)
        return id_info
    except ValueError:
        return None

@app.route('/tasks')
def get_tasks():
    token = request.headers.get('Authorization').split(' ').pop()
    id_info = verify_token(token)
    if id_info:
        return jsonify(tasks=tasks)
    else:
        return jsonify(error='Invalid token'), 401
```

This code checks the transferred token and makes sure that it is valid. If the token is invalid or missing, the server returns a 401 (Unauthorized) error. Otherwise, the task list is returned.

Step 3: API Monitoring



After configuring API security, it is important to monitor its performance and serviceability.

Enabling monitoring

To start tracking API requests, I turn on Google Cloud Monitoring:

gcloud services enable code monitoring.googleapis.com

Google Cloud now automatically collects data such as the number of requests, errors, and processing time.

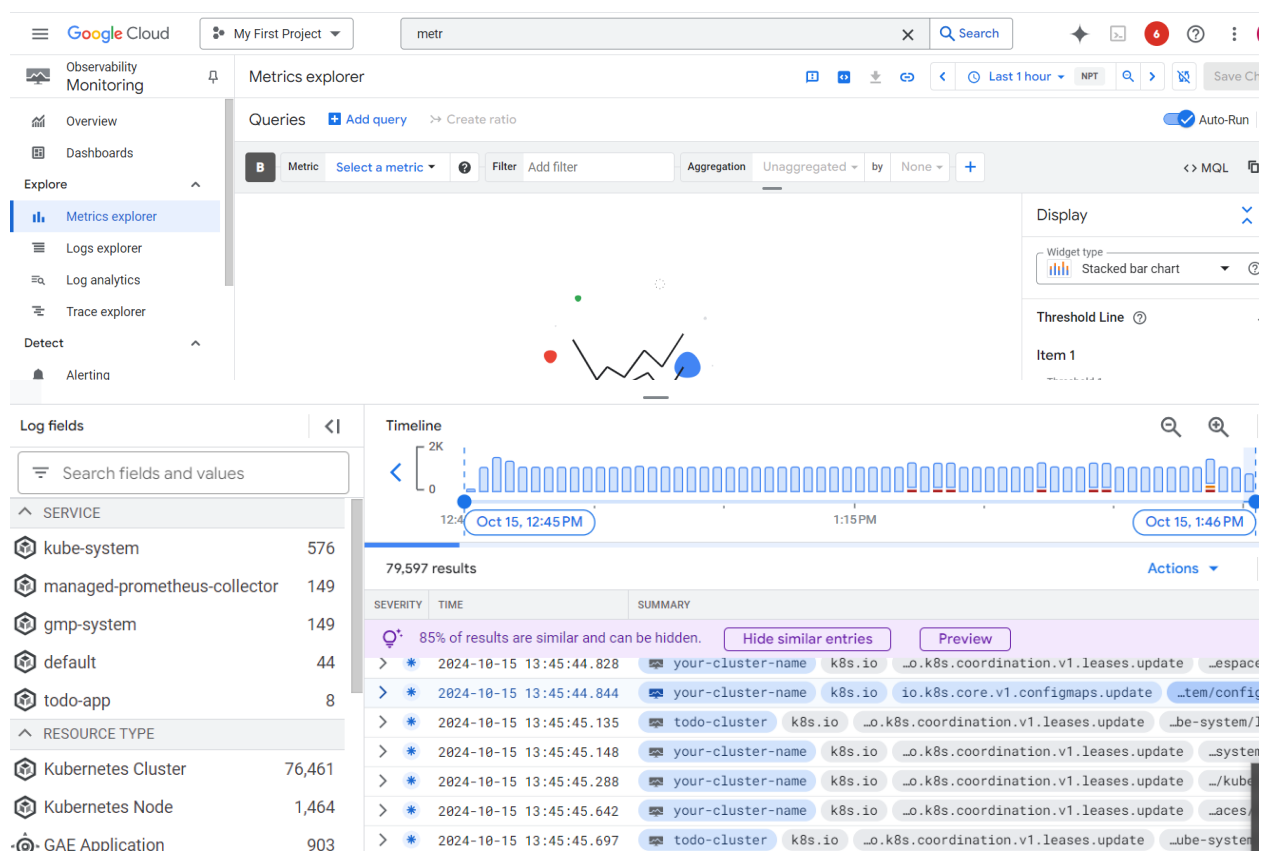
Viewing metrics

To view metrics, I log into the Google Cloud Console, in the Endpoints → Metrics section. Here I see information about:

Productivity: how long it takes to process each request.

Errors: how many errors occur when the API is running.

Use: how often and by which users the API is called.



10. Testing and Quality Assurance Plan

Unit testing

Unit tests allow you to check the operation of individual functions or methods of an application. This is the first level of testing, which aims to identify errors within individual components.

Example of a unit test for a Flask application:

```
yernur_y@DESKTOP-2PVM7F3: ~/todo-list-app
```

```
GNU nano 6.2
```

```
import unittest
from main import app

class TodoAppTestCase(unittest.TestCase):
    def setUp(self):

        self.app = app.test_client()
        self.app.testing = True

    def test_home_page(self):

        response = self.app.get('/')
        self.assertEqual(response.status_code, 200)
        self.assertIn(b'To-Do List', response.data)

    def test_add_task(self):

        response = self.app.post('/add', data=dict(task='Test Task'))
        self.assertEqual(response.status_code, 302)

    def test_delete_task(self):

        self.app.post('/add', data=dict(task='Test Task'))
        response = self.app.get('/delete/0')
        self.assertEqual(response.status_code, 302)

if __name__ == '__main__':
    unittest.main()
```

```
yernur_y@DESKTOP-2PVM7F3:~/todo-list-app$ nano test_main.py
yernur_y@DESKTOP-2PVM7F3:~/todo-list-app$ python3 -m unittest test_main.py
...
-----
Ran 3 tests in 1.346s

OK
```

I started the test with the command:

```
python -m unit test test_main.py
```

Integration testing

Integration tests check the interaction of different application components. For example, you can test how the Flask application integrates with Google Cloud Functions.

An example of an integration test for Cloud Function:

```
yernur_y@DESKTOP-2PVM7F3: ~/todo-list-app
GNU nano 6.2 main.py
import requests
from flask import Flask, request, render_template, redirect, url_for

app = Flask(__name__)

tasks = []
CLOUD_FUNCTION_URL = 'https://us-central1-savvy-hull-436617-b1.cloudfunctions.net/process_form_data'

@app.route('/')
def index():
    return render_template('index.html', tasks=tasks)

@app.route('/add', methods=['POST'])
def add_task():
    task = request.form.get('task')
    if task:
        response = requests.post(CLOUD_FUNCTION_URL, json={'task': task})
        if response.status_code == 200:
            tasks.append(task)
        else:
            print(f"Error processing task: {response.status_code}, {response.text}")
    return redirect(url_for('index'))

@app.route('/delete/<int:task_id>')
def delete_task(task_id):
    if 0 <= task_id < len(tasks):
        tasks.pop(task_id)
    return redirect(url_for('index'))

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=8080, debug=True)
```

```
yernur_y@DESKTOP-2PVM7F3:~/todo-list-app$ nano test_integration.py
yernur_y@DESKTOP-2PVM7F3:~/todo-list-app$ python3 -m unittest test_integration.py
```

```
yernur_y@DESKTOP-2PVM7F3:
Task 'Test Task' received and processed successfully!
```

This test sends a task processing request to CloudFunction and verifies that the function returns a successful response.

Load testing

Load testing helps to assess how resilient an application is to high load — for example, when dozens or hundreds of users are accessing at the same time. You can use the k6 tool for such tests.

An example of a load test using k6:

yernur_y@DESKTOP-2PVM7F3: ~/todo-list-app

GNU nano 6.2

```
import http from 'k6/http';
import { sleep } from 'k6';

export let options = {
  vus: 100,
  duration: '30s',
};

export default function () {
  http.get('https://savvy-hull-436617-b1.appspot.com');
  sleep(1);
}
```

Running a load test:

k6 ru load_test.js

yernur_y@DESKTOP-2PVM7F3:~/todo-list-app\$ k6 run load_test.js



execution: local
script: load_test.js
output: -

scenarios: (100.00%) 1 scenario, 100 max VUs, 1m0s max duration (incl. graceful stop):
* default: 100 looping VUs for 30s (gracefulStop: 30s)

data_received.....	965 kB	28 kB/s				
data_sent.....	110 kB	3.2 kB/s				
http_req_blocked.....	avg=85.76ms	min=294ns	med=855ns	max=1.62s	p(90)=629.35ms	p(95)=654.01ms
http_req_connecting.....	avg=21.81ms	min=0s	med=0s	max=1.19s	p(90)=153.46ms	p(95)=158.04ms
http_req_duration.....	avg=3.05s	min=393.94ms	med=2.25s	max=10.82s	p(90)=6.37s	p(95)=6.45s
{ expected_response:true }...	avg=3.05s	min=393.94ms	med=2.25s	max=10.82s	p(90)=6.37s	p(95)=6.45s
http_req_failed.....	0.00%	0	761			
http_req_receiving.....	avg=596.37µs	min=28.47µs	med=142.21µs	max=21.01ms	p(90)=1.45ms	p(95)=3.45ms
http_req_sending.....	avg=231µs	min=21.26µs	med=80.69µs	max=7.41ms	p(90)=258.04µs	p(95)=568.02µs
http_req_tls_handshaking.....	avg=32.17ms	min=0s	med=0s	max=292.75ms	p(90)=229.75ms	p(95)=255.37ms
http_req_waiting.....	avg=3.05s	min=393.72ms	med=2.25s	max=10.82s	p(90)=6.37s	p(95)=6.44s
http_reqs.....	761	22.358107/s				
iteration_duration.....	avg=4.14s	min=1.39s	med=3.25s	max=12.47s	p(90)=8.03s	p(95)=8.09s
iterations.....	761	22.358107/s				
vus.....	1	min=1	max=100			
vus_max.....	100	min=100	max=100			

running (0m34.0s), 000/100 VUs, 761 complete and 0 interrupted iterations

default [=====] 100 VUs 30s

This test simulates 100 users who will access your application within 30 seconds, helping to assess its scalability and load tolerance.

Test results:

Unit testing enables you to ensure that each individual component of the application (function, method) is functioning properly.

Integration tests: Checks the interaction between multiple application components, such as your Flask application and Google Cloud Functions.

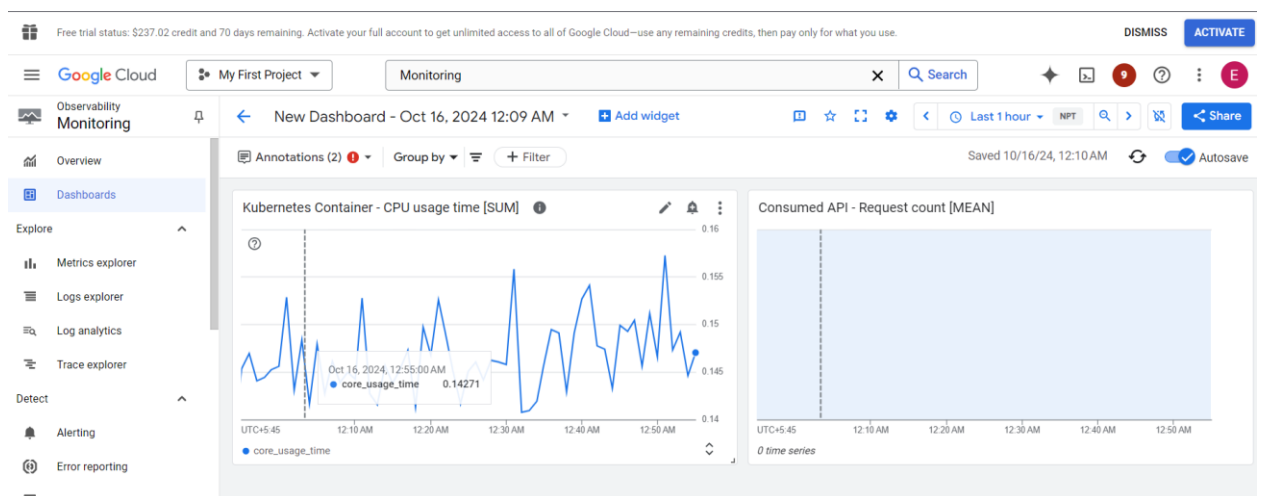
Load testing evaluates how an application handles many simultaneous requests and aids in the identification of scalability concerns.

11. Monitoring and Maintenance

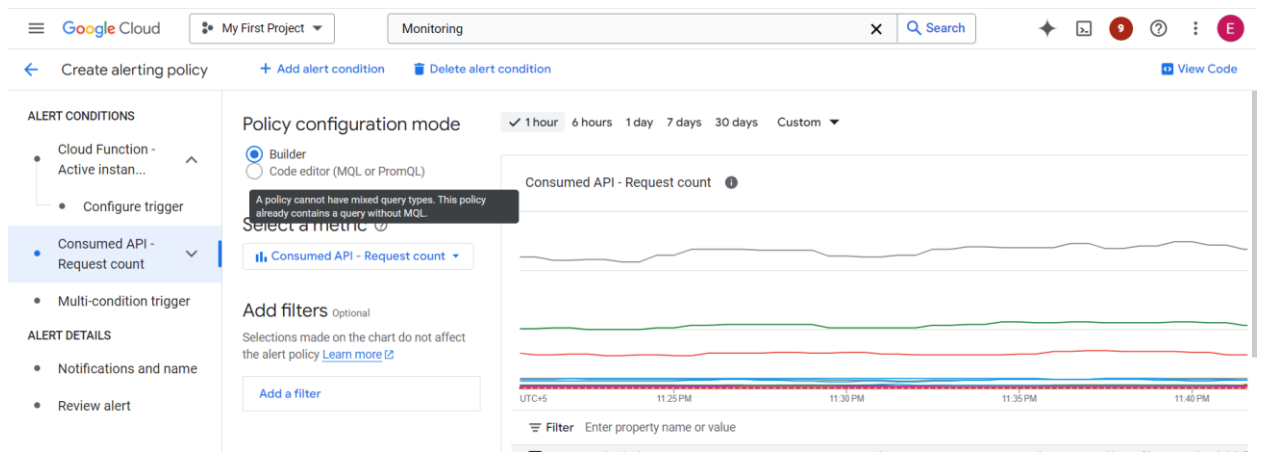
Monitoring with Google Cloud

To ensure that the application remains performant and reliable, I utilized Google Cloud's monitoring tools, such as Cloud Monitoring and Cloud Logging. These tools provide insights into real-time metrics, system health, and logs of the application.

- Cloud Monitoring helps track key performance indicators like CPU usage, memory consumption, and request latency. By setting up dashboards, I can continuously monitor application performance and detect anomalies early on.
- Cloud Logging allows me to capture all API requests and error messages. This is invaluable for troubleshooting, as it provides detailed information on what went wrong and when.



I also set up alerting policies to notify me via email whenever certain thresholds are met, such as high error rates or prolonged response times. This proactive approach ensures that I can address issues before they impact users.



Maintenance Practices

Regular maintenance is crucial to maintaining uptime and reliability. My maintenance practices include:

- **Automated Backups:** I scheduled automatic backups of the database and other critical resources to prevent data loss in case of system failure.
- **System Updates:** Keeping the system's dependencies and libraries updated ensures the latest security patches and performance improvements are applied.
- **Scaling Strategy:** The application is designed to scale with increasing user demand by utilizing Google Cloud's auto-scaling features. This ensures the application can handle sudden spikes in traffic without crashing.
- **Error Monitoring:** Cloud Monitoring tracks error rates, and I regularly review logs to identify and resolve recurring issues.

By combining monitoring tools with regular maintenance practices, I can ensure the application remains reliable, responsive, and secure. This strategy helps minimize downtime and maintains a seamless user experience.

12. Challenges and solutions

Throughout the project, I met a number of obstacles that demanded innovative solutions:

API Authentication: Initially, implementing Google Identity credentials for secure API access was challenging. I fixed this problem by reading Google's instructions and using the google-auth module, which allows for seamless authentication in my Flask project.

Cloud Function Integration: I had difficulty guaranteeing effective connectivity between my Flask app and Google Cloud Functions. I debugged the integration by sending HTTP requests and checking responses to ensure that the Cloud Function worked as intended.

Load Testing Issues: During load testing, the application struggled to handle excessive traffic. I optimized the app by enabling auto-scaling on Google Kubernetes Engine (GKE), ensuring that it scaled in response to demand.

By tackling these issues, I was able to create a robust and scalable application.

13. Conclusion.

Overall, the initiative was successful. I built an API-driven task management application that uses Google Cloud Endpoints and Google Cloud Functions to perform serverless tasks. Technologies like Flask and Google Cloud have shown to be quite useful in developing scalable and secure apps.

Moving ahead, I recommend improving the app's performance using sophisticated caching techniques and implementing machine learning features for task prioritization. The initiative has established a solid platform for future enhancements.

14. References

<https://stackoverflow.com/>

https://cloud.google.com/endpoints/docs/openapi/openapi-overview#basic_structure_of_an_openapi_document

Google Cloud Documentation: Google Cloud Endpoints

Flask Documentation: Flask Official Docs

Google OAuth 2.0 Authentication: Google OAuth 2.0 Guide

15. Appendices

Appendix A: Dockerfile for Flask Application

The Dockerfile is a script that contains a series of commands to build a Docker image for the Flask application. The goal is to containerize the application so that it can be deployed in a consistent environment across different platforms, such as local machines, cloud services, or Kubernetes clusters.

Dockerfile Explanation:

- **FROM python:3.10-slim:** This specifies the base image, which is a lightweight version of Python 3.10. It ensures the application runs in a Python environment.
- **WORKDIR /app:** This sets the working directory inside the Docker container to `/app`. All subsequent commands will be run in this directory.
- **COPY ./app:** This command copies the contents of the current directory (where the Dockerfile is located) into the `/app` directory inside the container.
- **RUN pip install --no-cache-dir -r requirements.txt:** This installs the Python dependencies listed in the `requirements.txt` file using the `pip` package manager, without caching to reduce the image size.
- **EXPOSE 8080:** This makes port 8080 available for the application, allowing it to accept requests from outside the container.
- **ENV FLASK_ENV=production:** This sets an environment variable to specify that the Flask application should run in production mode, ensuring performance optimization and security features.
- **CMD ["python", "main.py"]:** This tells Docker to run `main.py` using Python when the container is started. This is the entry point for running the Flask application.

This Dockerfile allows you to create a portable and reproducible environment for the Flask application, ensuring consistency in deployment.

```
FROM python:3.10-slim
```

```
WORKDIR /app
```

```
COPY . /app
```

```
RUN pip install --no-cache-dir -r requirements.txt
```

```
EXPOSE 8080
```

```
ENV FLASK_ENV=production
```

```
CMD ["python", "main.py"]
```

Appendix B: OpenAPI Specification for Task Management API

The OpenAPI specification defines the structure and behavior of the API for the task management system. OpenAPI is a standard format for describing REST APIs, making it easier to develop, consume, and document APIs.

yernur_y@DESKTOP-2PVM7F3: ~/todo-list-app

```
GNU nano 6.2
openapi: 3.0.0
info:
  title: Todo API
  description: API for managing tasks
  version: 1.0.0
servers:
  - url: https://savvy-hull-436617-b1.cloudfunctions.net
paths:
  /tasks:
    get:
      summary: Get all tasks
      operationId: getTasks
      responses:
        '200':
          description: Successful response
    /tasks/{task_id}:
      delete:
        summary: Delete a task
        operationId: deleteTask
        parameters:
          - name: task_id
            in: path
            required: true
            schema:
              type: integer
        responses:
          '200':
            description: Task successfully deleted
components:
  securitySchemes:
    google_id_token:
      type: oauth2
      flows:
        implicit:
          authorizationUrl: ''
          scopes: []
security:
  - google_id_token: []
```

openapi: 3.0.0: This specifies that the OpenAPI version being used is 3.0.0.

info: This section contains metadata about the API.

- title: The title of the API is "Task Management API", which describes its purpose.

- **description:** A brief description of the API, noting that it is used for managing tasks in a to-do list application.
- **version:** This indicates the version of the API, which in this case is 1.0.0.

servers: This section lists the available servers for the API.

- **url:** The base URL for the API, where the placeholder "your-project-id" should be replaced with your actual Google Cloud project ID. This URL points to the Cloud Functions endpoint that handles the API requests.

paths: This section defines the available endpoints.

/tasks: This endpoint is used to manage tasks.

get: The HTTP GET method retrieves all tasks.

summary: A brief description of what the endpoint does (retrieves all tasks).

operationId: A unique identifier for this operation, used for documentation or API generation tools.

responses: This section describes the possible responses.

'200': A successful response will return a list of tasks.

/tasks/{task_id}: This endpoint is used to delete a specific task based on the task ID.

delete: The HTTP DELETE method deletes a specific task.

summary: Describes the operation, which deletes a task.

parameters: Specifies the task ID parameter that needs to be passed in the URL path.

name: The name of the parameter, which is task_id.

in: Indicates that the parameter is in the path.

required: Specifies that the parameter is mandatory.

schema: Describes the data type of the parameter (an integer in this case).

responses: This section describes the possible responses.

'200': A successful response will confirm that the task was deleted.

components: This section defines reusable security schemes.

securitySchemes: Defines the security protocols used in the API.

google_id_token: Specifies that OAuth 2.0 tokens will be used for authentication. This ensures that users must provide a valid Google ID token to access certain endpoints.

security: This section applies security settings globally to the API.

google_id_token: Requires authentication via Google ID tokens for the entire API.