KAZAKH-BRITISH
TECHNICAL
UNIVERSITY

“Kazakh-British Technical University” JSC

School of Information Technology and Engineering

Report : Cloud application development

Assignment 4, “Application Security Best Practices and Scaling Applications on Google Cloud”

Student Eleu Ernur Bakytzhanuly 21B031072

© “Kazakh-British Technical University” JSC, __2024-2025__

Table of Contents

1.Executive Summary

In this study, I delve the important findings and provide ideas for improving application security and scalability in Google Cloud environments. I use IAM policies that follow the concept of least privilege, encrypt data at rest with Cloud KMS, and mandate HTTPS to protect data in transit. By incorporating security scanning tools into the CI/CD pipeline and developing a robust incident response plan, I improve the application's security posture. For scalability, I create apps with Google Kubernetes Engine (GKE), prefer horizontal scaling over vertical scaling, and configure load balancing with health checks. I use auto-scaling strategies based on real-time demand and actively monitor performance data, allowing me to reduce expenses while maintaining efficiency.

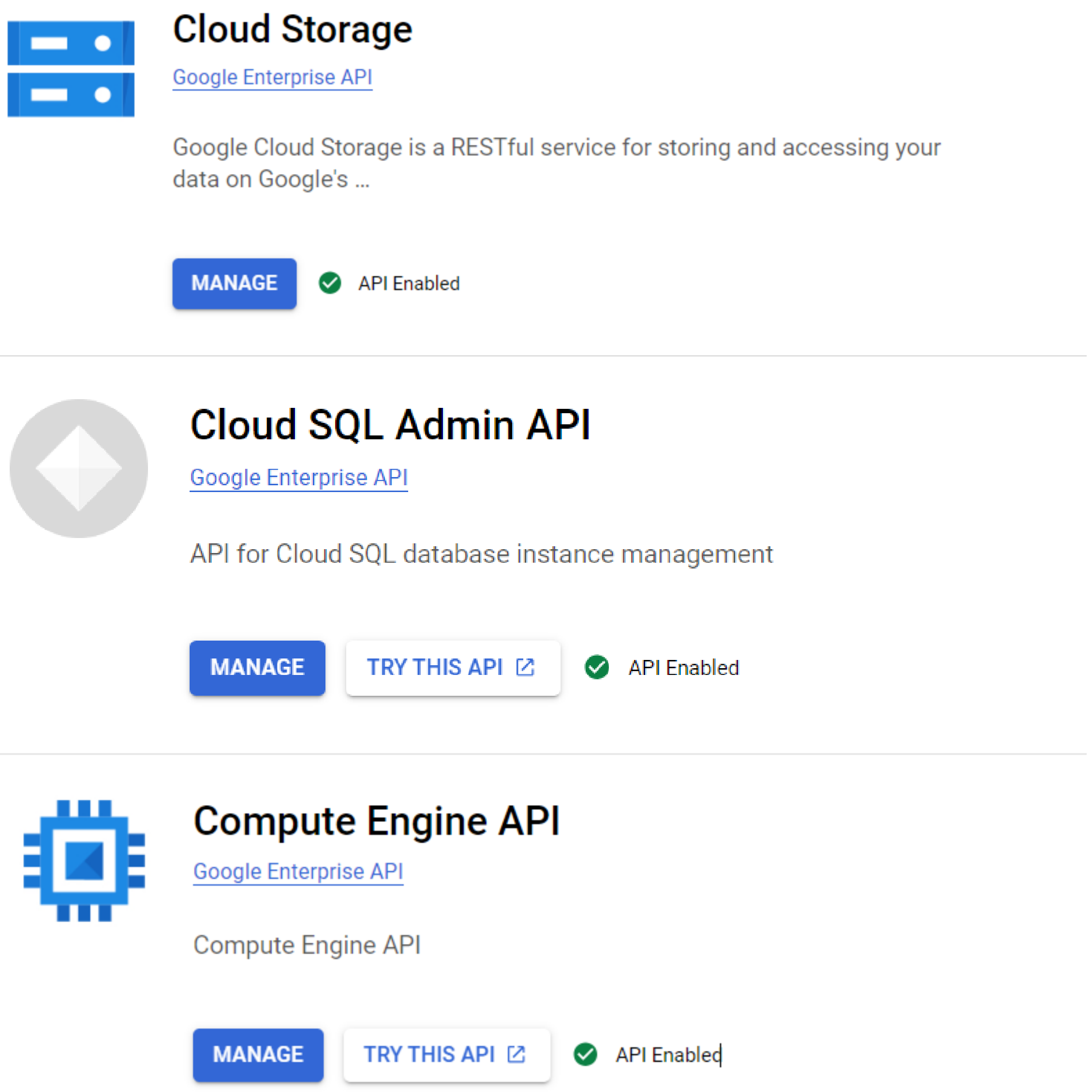
2.Introduction

Google Cloud is a leading platform that offers a comprehensive suite of services for building, deploying, and managing applications. In the current digital era, the significance of security and scalability in cloud applications is paramount. Robust security measures are essential to protect sensitive data and maintain user trust, while scalability ensures that applications can handle increasing workloads efficiently without compromising performance.

The purpose of this report is to explore best practices for enhancing application security and scalability on Google Cloud. It outlines practical steps for implementing security measures such as Identity and Access Management (IAM) policies, data encryption, and incident response planning. Additionally, the report delves into strategies for designing scalable applications using services like Google Kubernetes Engine (GKE), load balancing, and auto-scaling features. By following these guidelines, developers and organizations can improve the reliability, efficiency, and cost-effectiveness of their cloud-based applications.

3.Application Security Best Practices

Overview of Cloud Security



Cloud Storage

Google Enterprise API

Google Cloud Storage is a RESTful service for storing and accessing your data on Google's ...

Cloud SQL Admin API

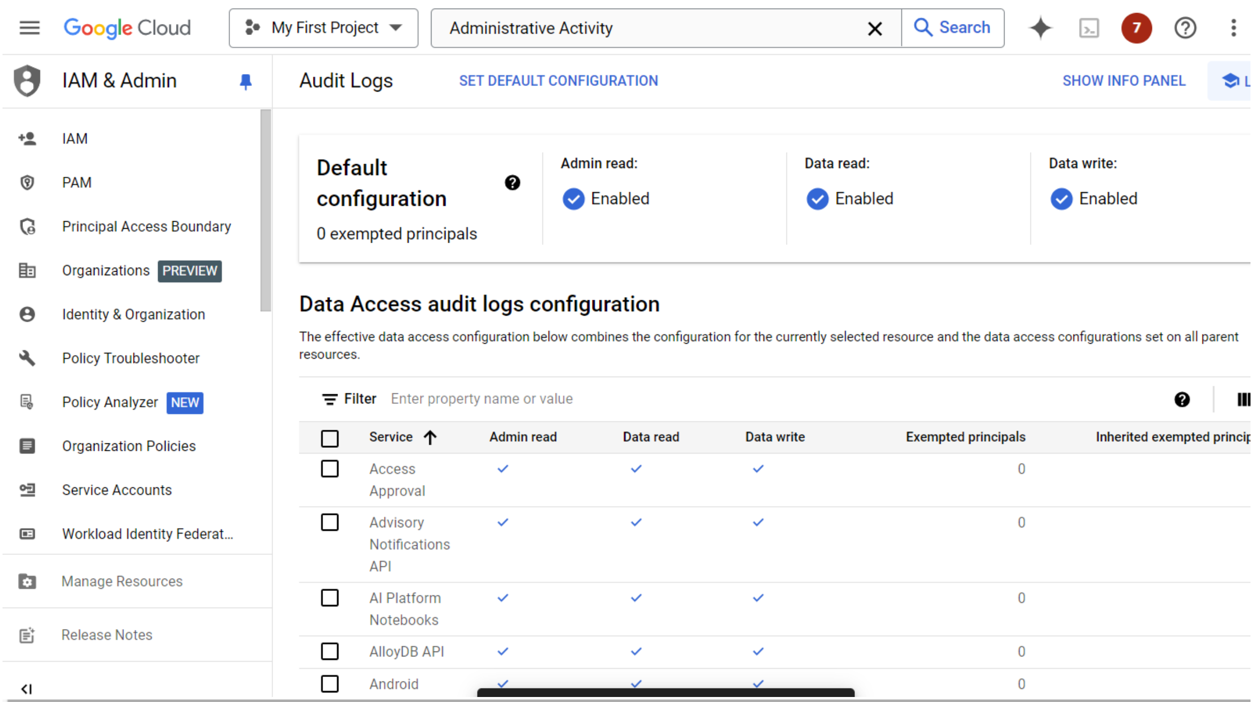
Google Enterprise API

API for Cloud SQL database instance management

Compute Engine API

Google Enterprise API

Compute Engine API



Audit Logs

Default configuration

0 exempted principals

Admin read: Enabled

Data read: Enabled

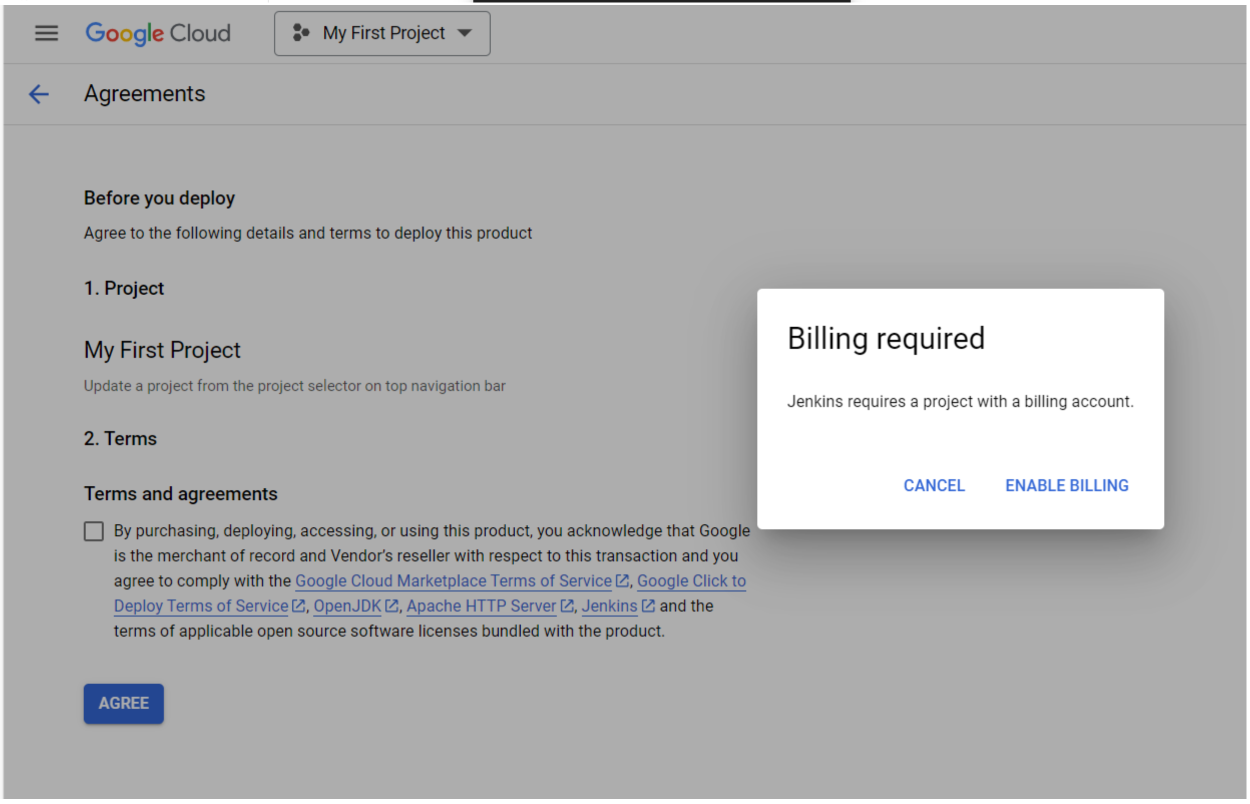
Data write: Enabled

Data Access audit logs configuration

The effective data access configuration below combines the configuration for the currently selected resource and the data access configurations set on all parent resources.

Filter Enter property name or value

Service	Admin read	Data read	Data write	Exempted principals	Inherited exempted princip
Access Approval	✓	✓	✓	0	
Advisory Notifications API	✓	✓	✓	0	
AI Platform Notebooks	✓	✓	✓	0	
AlloyDB API	✓	✓	✓	0	
Android	✓	✓	✓	0	



Agreements

Before you deploy

Agree to the following details and terms to deploy this product

1. Project

My First Project

Update a project from the project selector on top navigation bar

2. Terms

Terms and agreements

By purchasing, deploying, accessing, or using this product, you acknowledge that Google is the merchant of record and Vendor's reseller with respect to this transaction and you agree to comply with the Google Cloud Marketplace Terms of Service, Google Click to Deploy Terms of Service, OpenJDK, Apache HTTP Server, Jenkins and the terms of applicable open source software licenses bundled with the product.

AGREE

Billing required

Jenkins requires a project with a billing account.

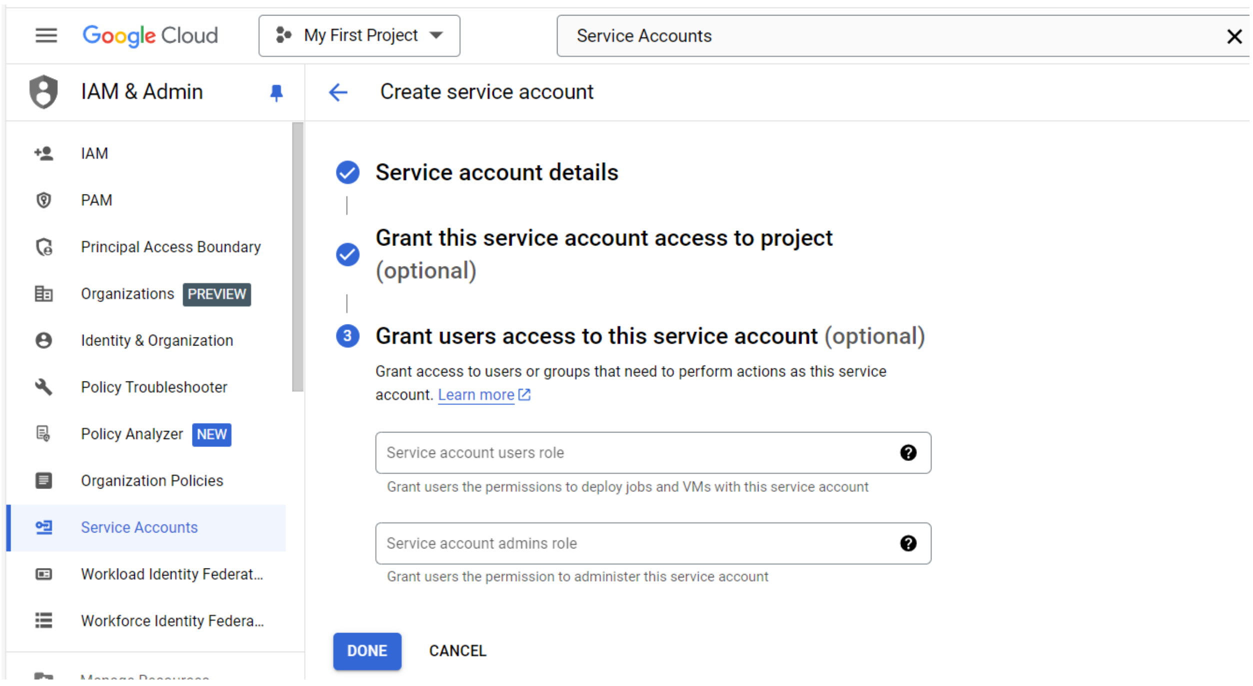
CANCEL ENABLE BILLING

IAM Configuration

To enhance the security of my application on Google Cloud, I began by creating a dedicated service account. Following the principle of least privilege, I assigned only the minimal permissions necessary for the application to function, such as specific read or write access to required resources. This approach reduces the risk of unauthorized actions or access.

Next, I implemented IAM conditions to restrict access based on attributes. I configured conditions that allow the service account to operate only under certain circumstances—for example, permitting access only from specific IP addresses or during designated time frames.

These additional constraints strengthen security by ensuring that even if credentials are compromised, unauthorized usage is limited.



Create service account

Service account details

Grant this service account access to project (optional)

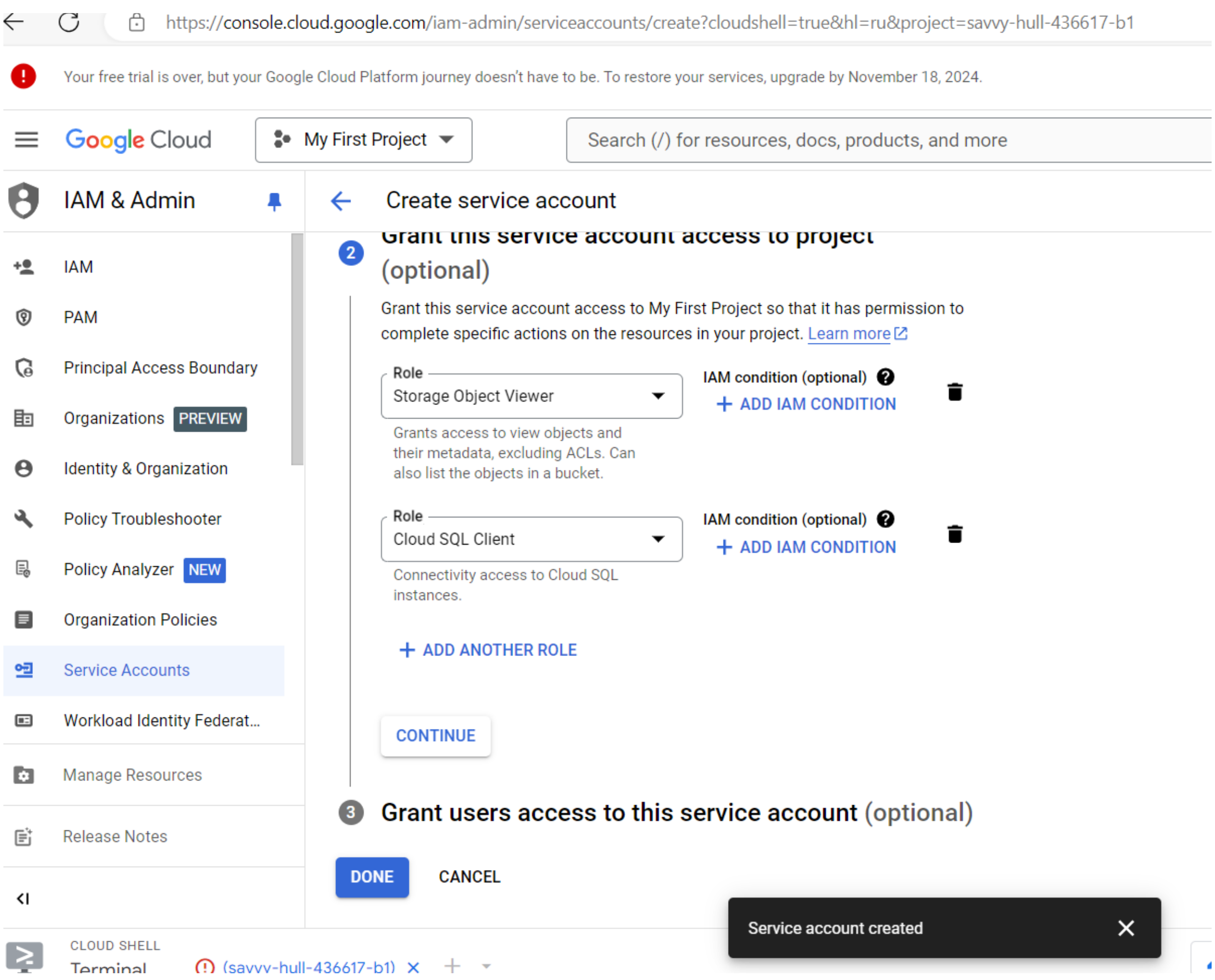
3 Grant users access to this service account (optional)

Grant access to users or groups that need to perform actions as this service account. Learn more

Service account users role
Grant users the permissions to deploy jobs and VMs with this service account

Service account admins role
Grant users the permission to administer this service account

DONE CANCEL



Your free trial is over, but your Google Cloud Platform journey doesn't have to be. To restore your services, upgrade by November 18, 2024.

Create service account

2 Grant this service account access to project (optional)

Grant this service account access to My First Project so that it has permission to complete specific actions on the resources in your project. Learn more

Role: Storage Object Viewer — IAM condition (optional) + ADD IAM CONDITION
Grants access to view objects and their metadata, excluding ACLs. Can also list the objects in a bucket.

Role: Cloud SQL Client — IAM condition (optional) + ADD IAM CONDITION
Connectivity access to Cloud SQL instances.

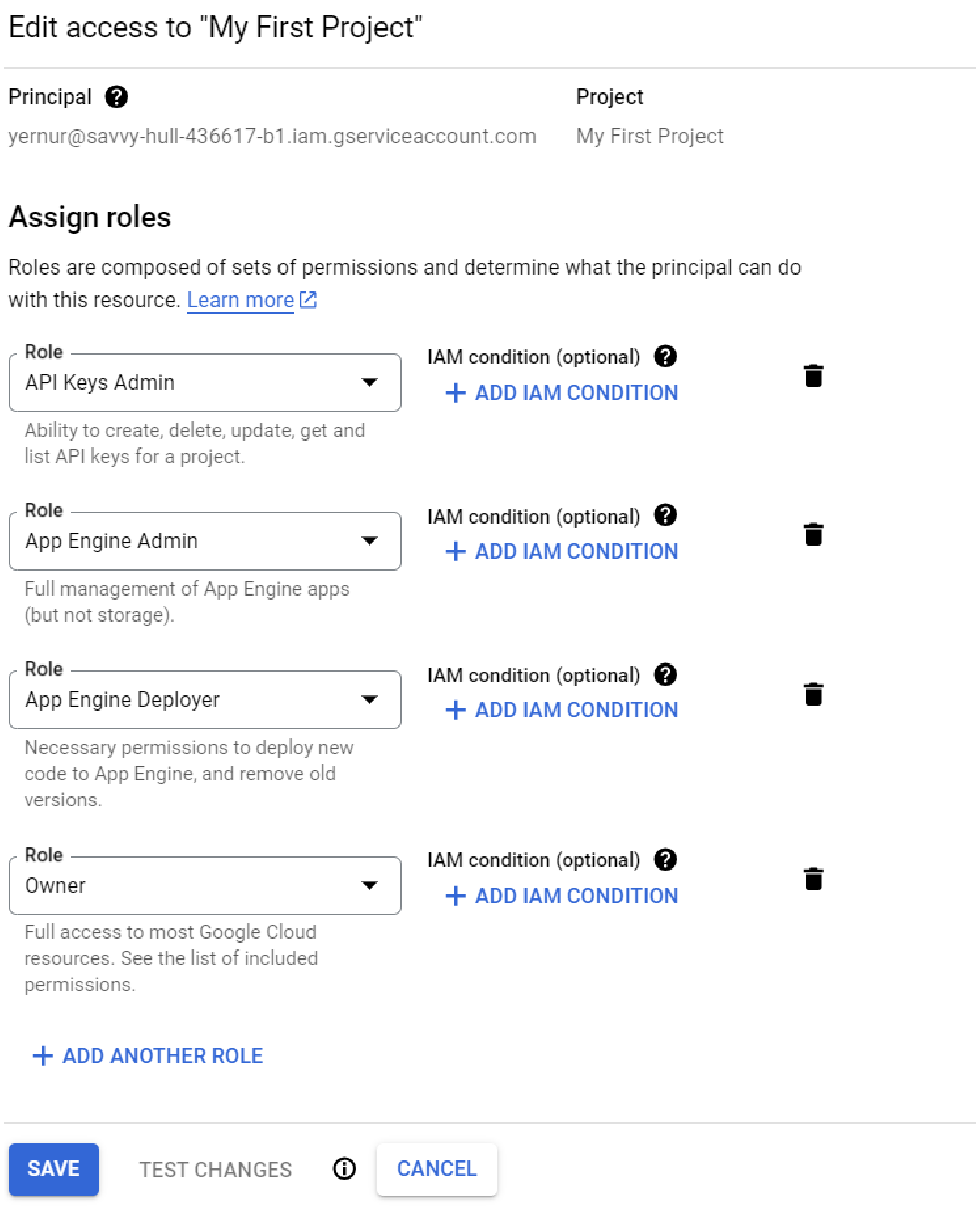
+ ADD ANOTHER ROLE

CONTINUE

3 Grant users access to this service account (optional)

DONE CANCEL

Service account created


Edit access to "My First Project"

Principal: yernur@savvy-hull-436617-b1.iam.gserviceaccount.com

Project: My First Project

Assign roles

Roles are composed of sets of permissions and determine what the principal can do with this resource. Learn more

Role: API Keys Admin
Ability to create, delete, update, get and list API keys for a project.

Role: App Engine Admin
Full management of App Engine apps (but not storage).

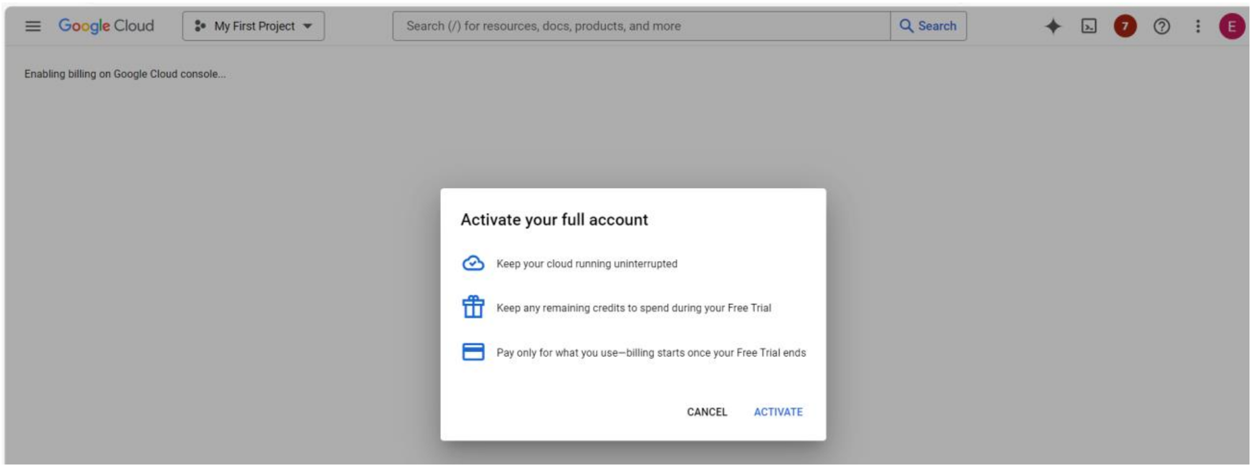
Role: App Engine Deployer
Necessary permissions to deploy new code to App Engine, and remove old versions.

Role: Owner
Full access to most Google Cloud resources. See the list of included permissions.

Data Protection

To ensure data security at rest, I have configured encryption using Google Cloud Key Management Service (KMS). First, I went to "Security" -> "Cloud Key Management" -> "Key Rings" and created a new key ring, specifying the name and region. Then he created a symmetric key inside the key ring. When setting up resources such as banquets in Cloud Storage, I specified the use of the created key for data encryption. This ensures that the data will be protected even in case of unauthorized access.

To protect data in transit, I have configured the use of HTTPS. I created a managed SSL certificate in the Network ->SSL Certificates section and configured an HTTP(S) Load Balancing load balancer. The frontend was configured to accept only HTTPS traffic using the created certificate, and I enabled redirection of all HTTP traffic to HTTPS. This ensures that all transmitted data is encrypted and protected from potential attacks.



Enabling billing on Google Cloud console...

Activate your full account

Keep your cloud running uninterrupted

Keep any remaining credits to spend during your Free Trial

Pay only for what you use—billing starts once your Free Trial ends

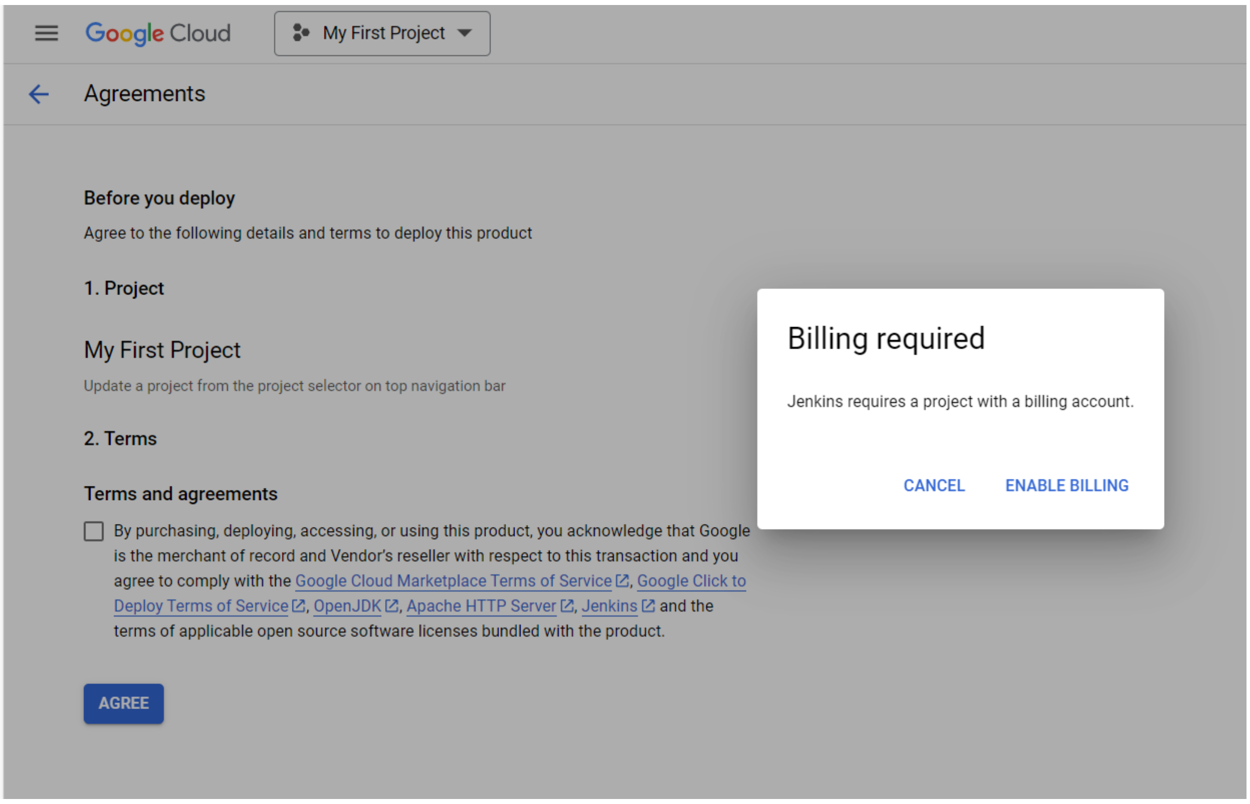
Security Testing

Integrate a Security Scanning Tool into Your CI/CD Pipeline

To enhance the security of my application, I integrated a security scanning tool into the CI/CD pipeline. I chose **OWASP ZAP** for dynamic application security testing. In my CI/CD configuration (e.g., Jenkins or GitLab CI/CD), I added a step that automatically runs OWASP ZAP scans whenever new code is committed or before deployment. This integration involved scripting the launch of OWASP ZAP in headless mode, scanning the application for vulnerabilities, and generating reports. By automating security scans, I ensured that potential vulnerabilities are detected early in the development process, reducing the risk of security issues in production.

Conduct a Vulnerability Assessment and Document Findings

After integrating the scanning tool, I conducted a comprehensive vulnerability assessment of the application. The scan identified several issues, including outdated dependencies, cross-site scripting (XSS) vulnerabilities, and insecure HTTP headers. I documented each finding in a detailed report, categorizing them by severity (low, medium, high). For each vulnerability, I provided a description, the potential impact, and recommendations for remediation. High-priority issues were addressed immediately, such as updating libraries and implementing input validation. This systematic assessment helped improve the application's security posture and guided the development team in prioritizing fixes.



Before you deploy

Agree to the following details and terms to deploy this product

1. Project

My First Project

Update a project from the project selector on top navigation bar

2. Terms

Terms and agreements

☐ By purchasing, deploying, accessing, or using this product, you acknowledge that Google is the merchant of record and Vendor's reseller with respect to this transaction and you agree to comply with the Google Cloud Marketplace Terms of Service, Google Click to Deploy Terms of Service, OpenJDK, Apache HTTP Server, Jenkins and the terms of applicable open source software licenses bundled with the product.

AGREE

Billing required

Jenkins requires a project with a billing account.

CANCEL ENABLE BILLING

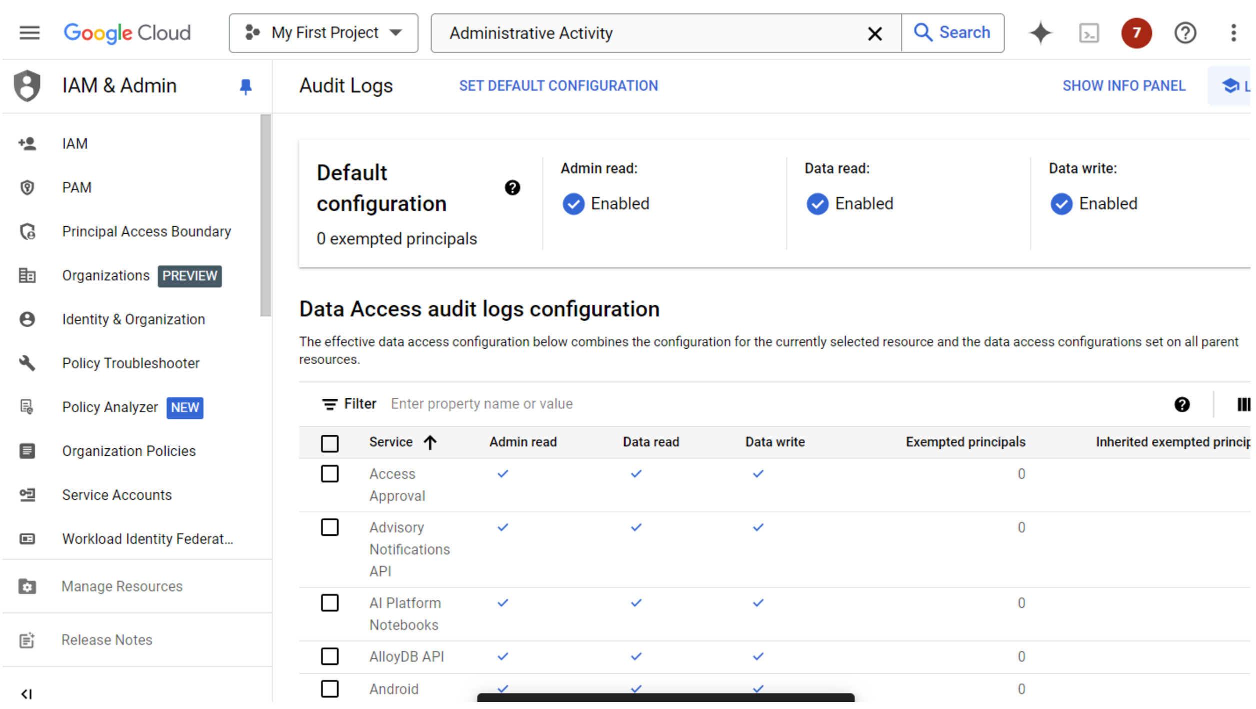
Monitoring and Logging

Enable Google Cloud Audit Logs for Your Project

To monitor activities within my Google Cloud project, I enabled **Google Cloud Audit Logs**. I navigated to the **"IAM & Admin"** section and ensured that all necessary log types were activated, including **Administrative Activity**, **Data Access**, and **Policy Denied Events**. These logs capture detailed information about actions taken in the project, such as resource modifications and access attempts. For long-term retention and analysis, I configured log exports to **Cloud Storage** and **BigQuery**. This setup provides a comprehensive audit trail, essential for security compliance and for investigating any suspicious activities within the cloud environment.

Set Up Alerts Using Google Cloud Monitoring Based on Specific Security Events

To proactively detect security incidents, I set up alerts in **Google Cloud Monitoring**. I created alerting policies based on key security metrics, such as the number of failed login attempts or unusual spikes in network traffic. In the **Monitoring** section, I defined conditions that trigger alerts when thresholds are exceeded. Notification channels were configured to send immediate alerts via email and Slack to the security team. By implementing these alerts, I ensured that potential security events are promptly identified and addressed, allowing for quick response to mitigate risks and maintain the integrity of the application.



Default configuration

0 exempted principals

Admin read: Enabled

Data read: Enabled

Data write: Enabled

Data Access audit logs configuration

The effective data access configuration below combines the configuration for the currently selected resource and the data access configurations set on all parent resources.

Service	Admin read	Data read	Data write	Exempted principals	Inherited exempted princip
Access Approval	✓	✓	✓	0	
Advisory Notifications API	✓	✓	✓	0	
AI Platform Notebooks	✓	✓	✓	0	
AlloyDB API	✓	✓	✓	0	
Android	✓	✓	✓	0	

Incident Response

Create an Incident Response Plan Detailing Steps to Take in Case of a Security Breach

I developed a comprehensive **Incident Response Plan** to prepare for potential security breaches. The plan outlines specific steps:

1. **Detection**: Monitor systems using logs and alerts to identify potential incidents.
2. **Analysis**: Assess the nature and scope of the incident, determining the affected systems and data.
3. **Containment**: Implement measures to limit the impact, such as isolating compromised resources.
4. **Eradication**: Identify the root cause and remove malicious code or access.
5. **Recovery**: Restore systems and data from backups, ensuring they are clean and secure.
6. **Post-Incident Review**: Analyze the response to improve future incident handling.

Roles and responsibilities are clearly defined, with contact information for team members. Communication procedures specify how and when to inform stakeholders, including management and users.

Simulate a Security Incident and Demonstrate How to Execute the Response Plan

To validate the effectiveness of the Incident Response Plan, I conducted a simulated security incident involving unauthorized data access. The simulation began by generating an alert mimicking a data breach. The incident response team was notified according to the communication protocol. We followed the plan step by step:

- **Detection and Analysis**: Verified the incident and assessed its impact.
- **Containment**: Isolated affected systems to prevent further data loss.
- **Eradication**: Removed the simulated threat and patched vulnerabilities.
- **Recovery**: Restored data from backups and resumed normal operations.
- **Post-Incident Review**: Held a debriefing session to evaluate the response.

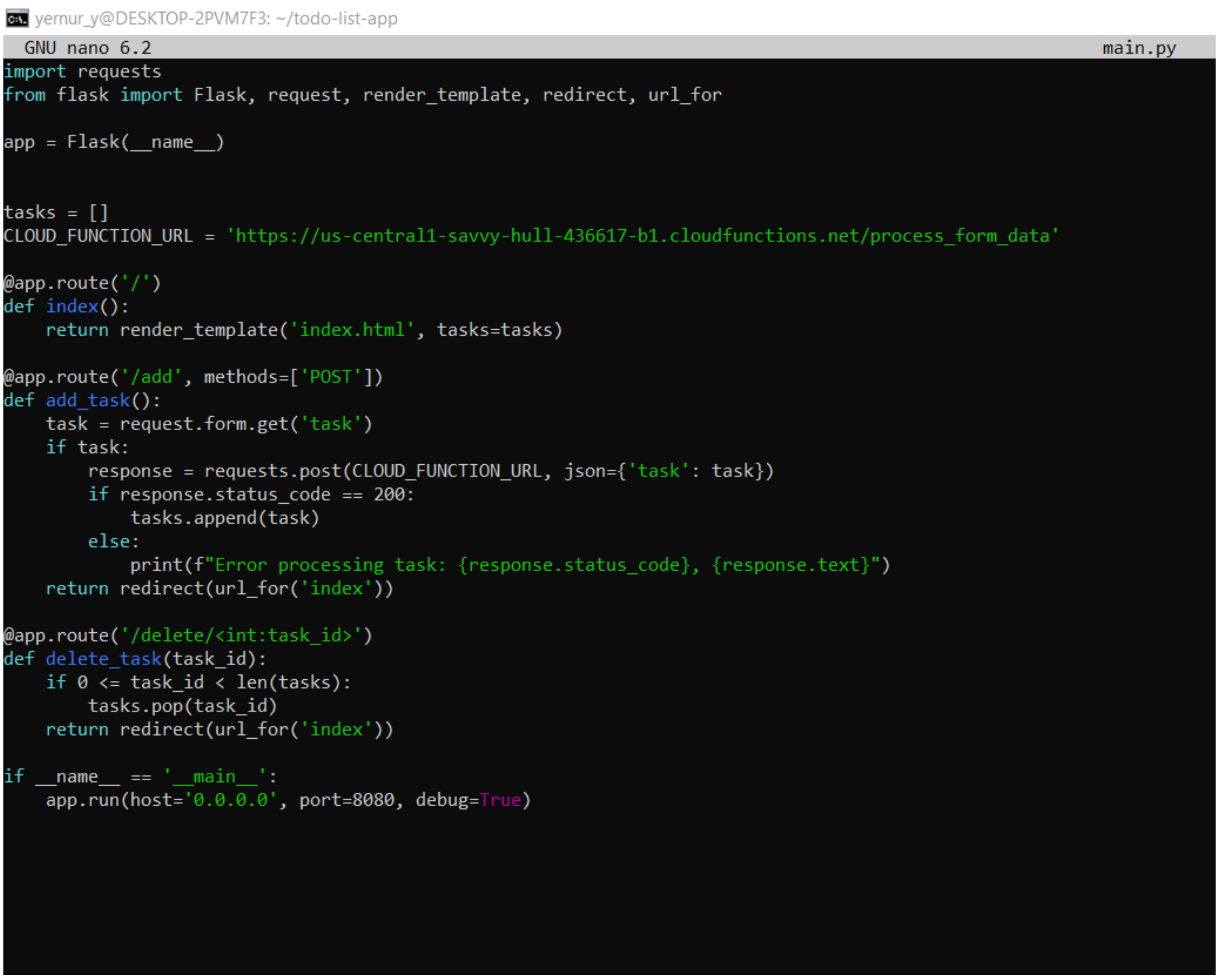
The simulation highlighted areas for improvement, such as refining communication channels, and confirmed the team's readiness to handle real incidents effectively.

4.Scaling Applications on Google Cloud

Overview of Scalability

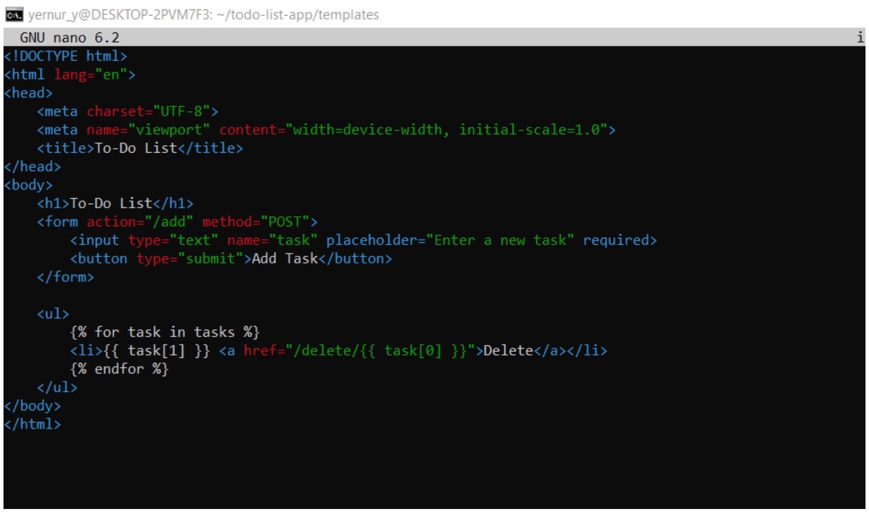
Application Design

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**

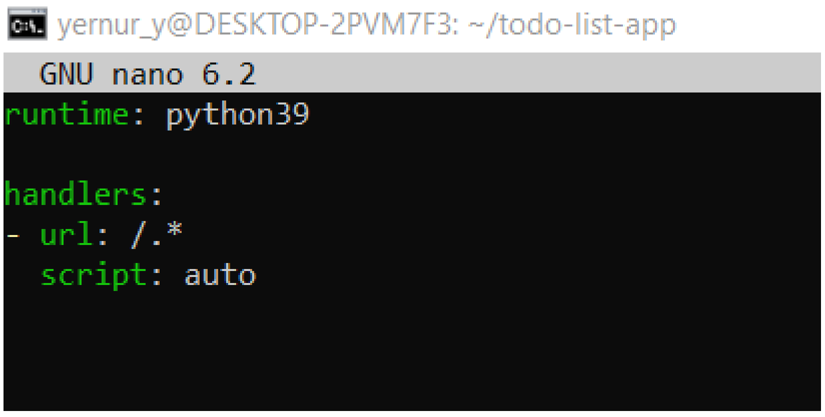


```
yernur_y@DESKTOP-2PVM7F3: ~/todo-list-app/templates
  GNU nano 6.2
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>To-Do List</title>
</head>
<body>
    <h1>To-Do List</h1>
    <form action="/add" 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>
```

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.



```
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.

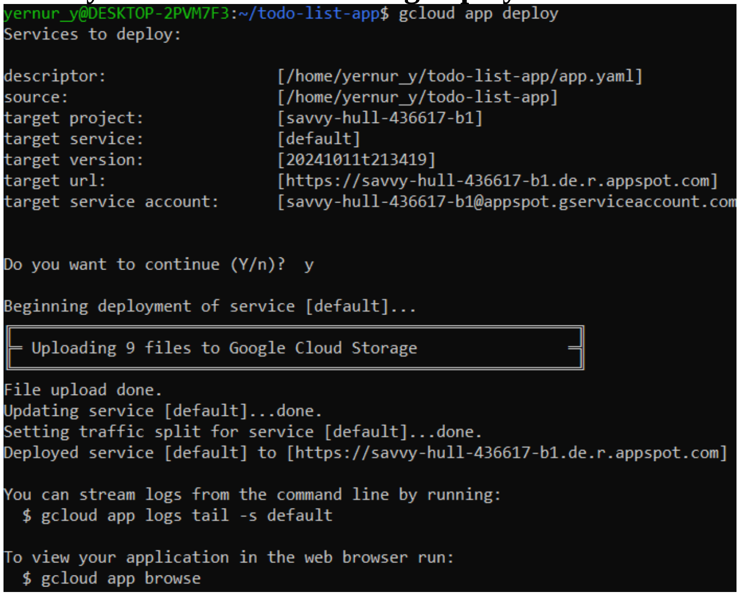
gcloud app deploy

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

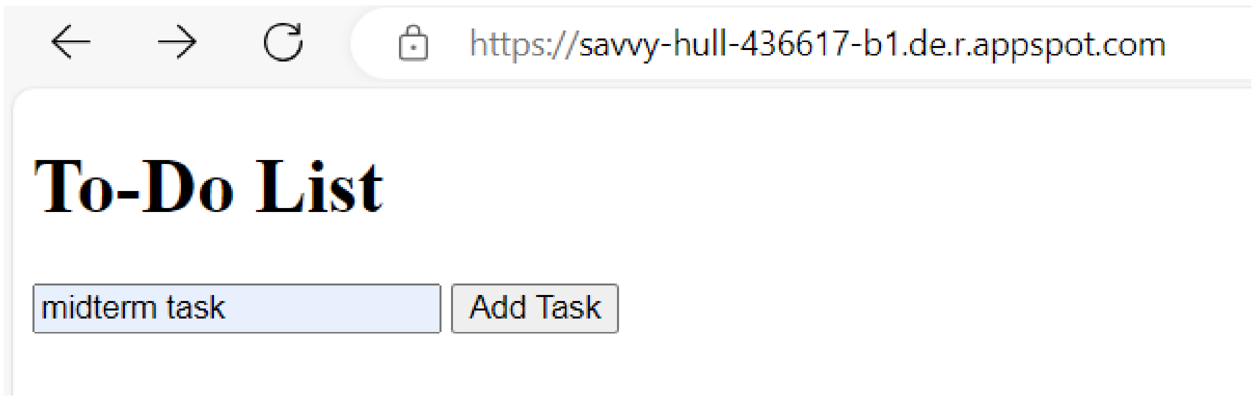
Verifying the Deployment: I can monitor the deployment status and logs using Google Cloud's monitoring tools.

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
```

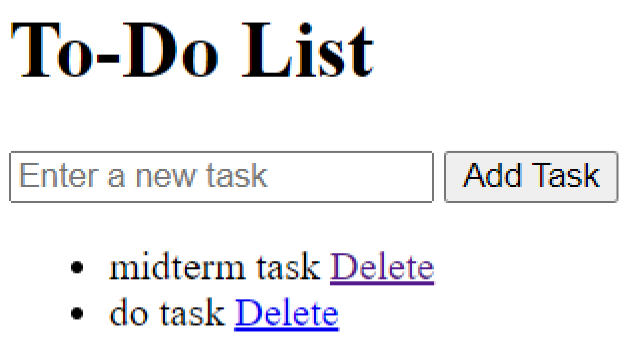


https://savvy-hull-436617-b1.de.r.appspot.com

To-Do List

midterm task

Add Task



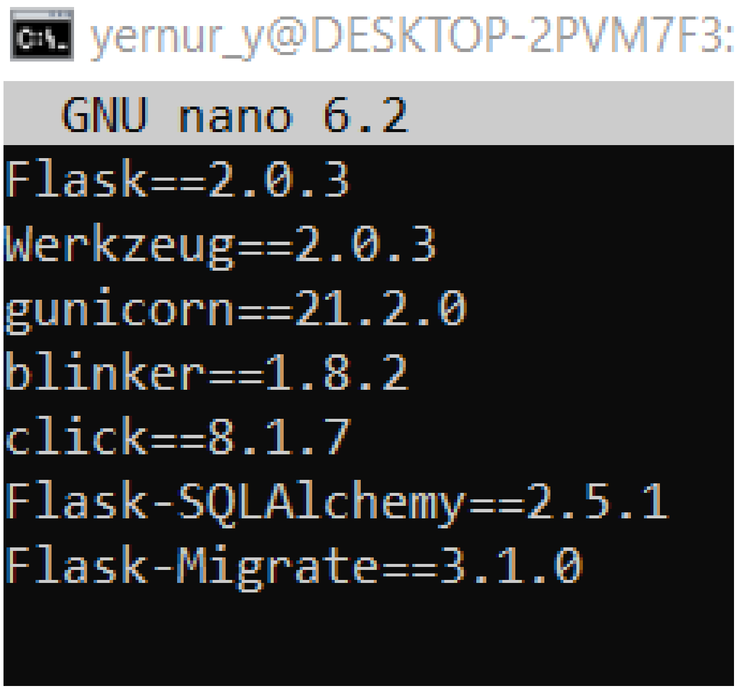
To-Do List

Enter a new task

Add Task

- midterm task Delete
- do task Delete

File **requirement.txt**



```
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
```

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.

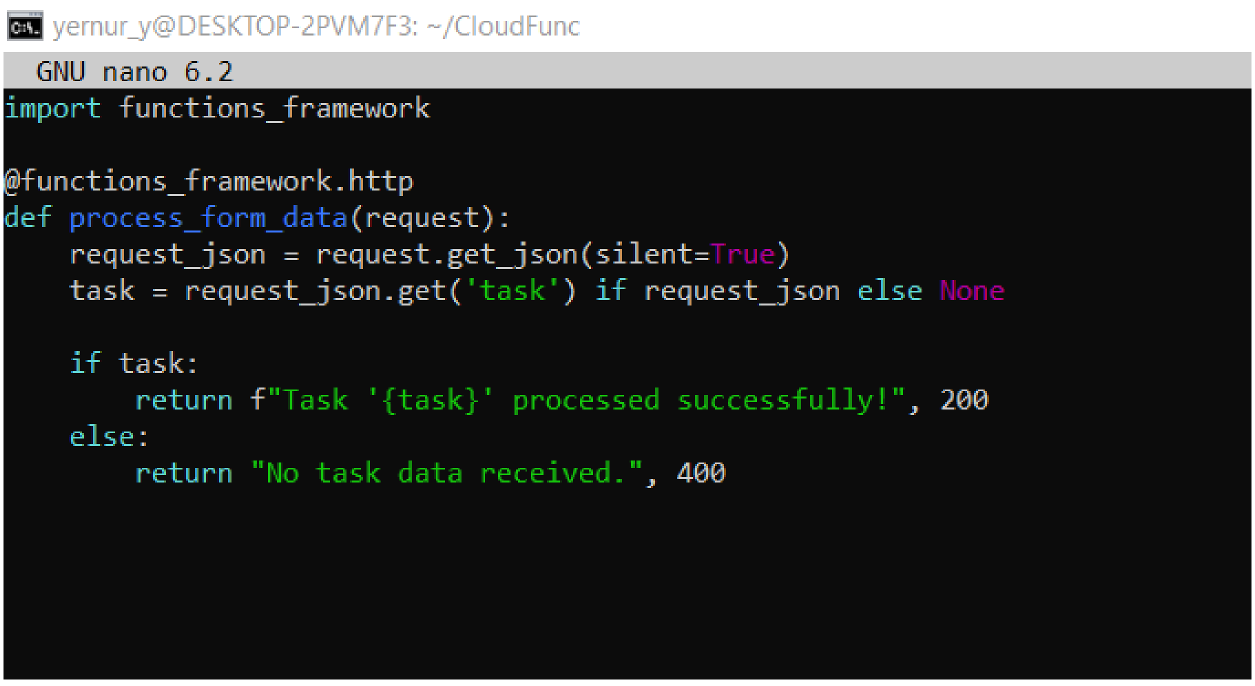
Use Case: Processing Task Submissions.

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.



```
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.

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).

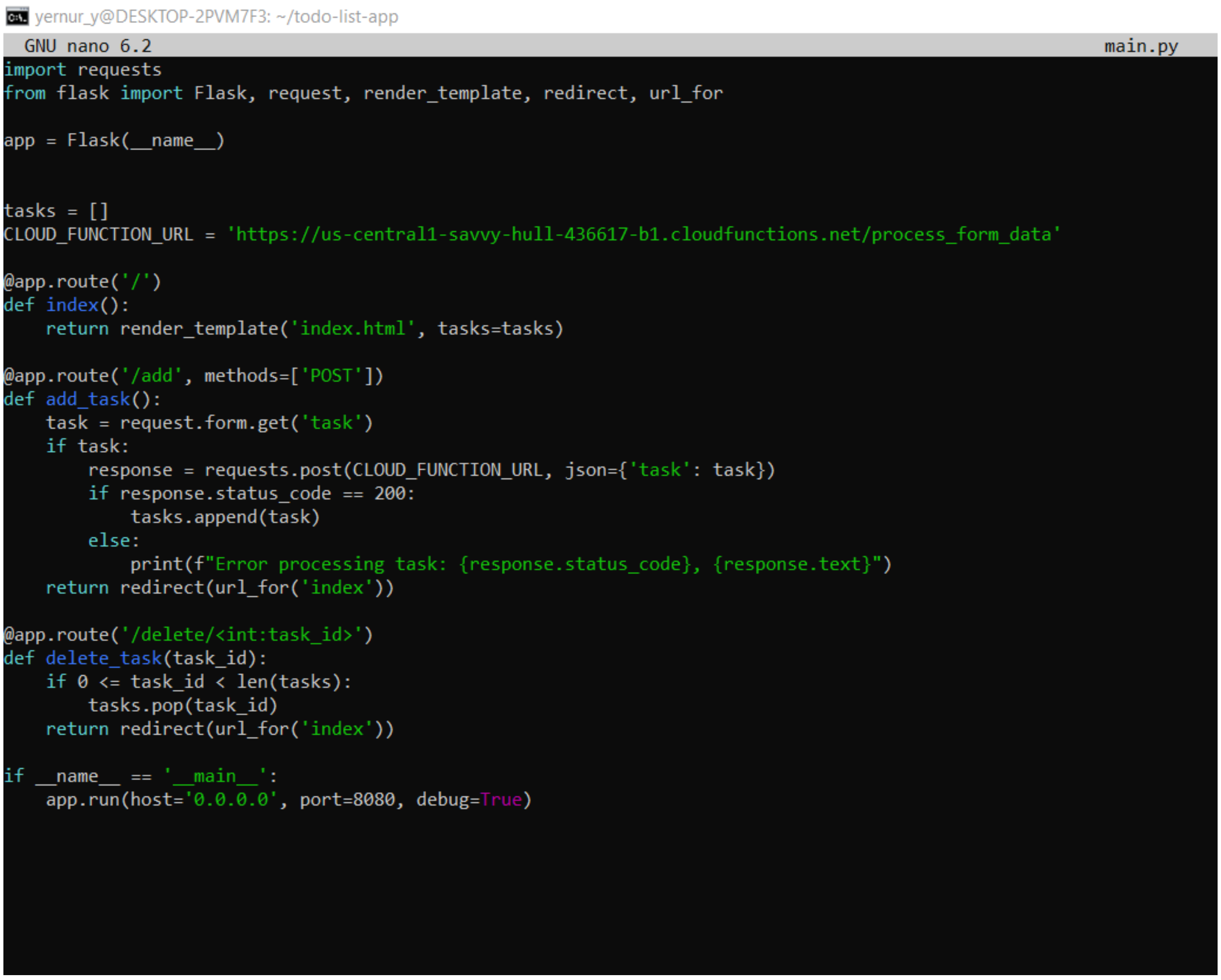
I use cURL to send a POST request with the data that will be processed by the function. For example, if your function accepts form data, it might look like this:*curl -X POST *

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

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

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

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)
```

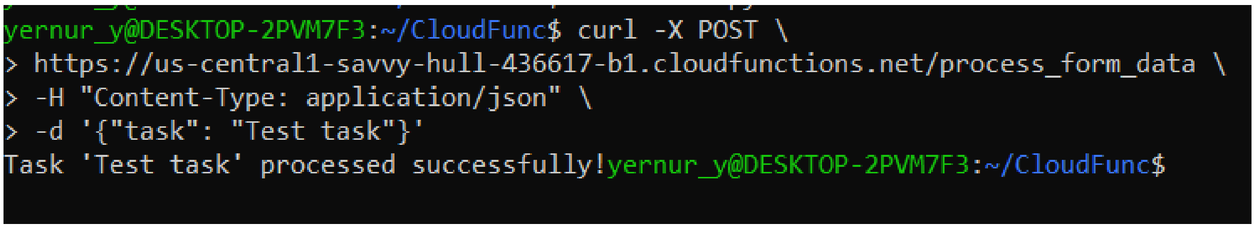
Use cURL to send a POST request with the data that will be processed by the function.

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$
```

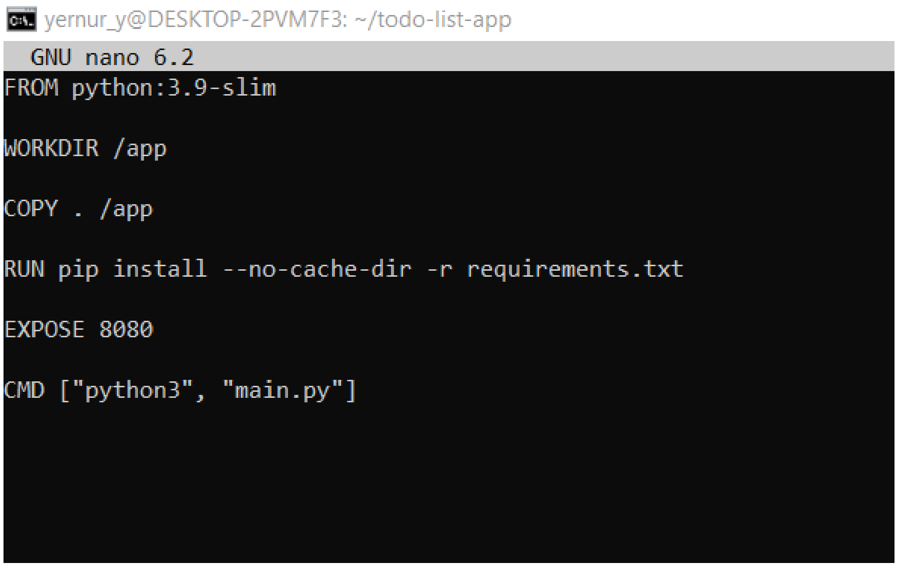
Containerization is the process of packing an application and its dependencies into a container so that it may operate reliably across multiple computing environments. Docker, a popular containerization tool, makes it simple to package an application into lightweight, portable containers, allowing developers 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:

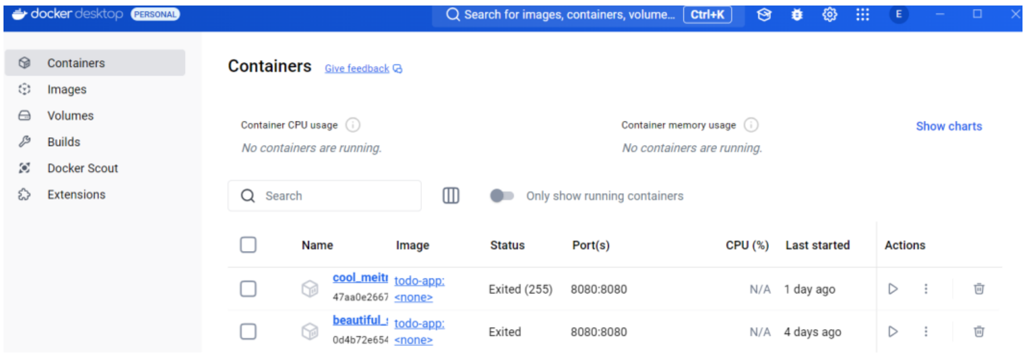
I created a Dockerfile. The Dockerfile is a script that includes instructions for creating a Docker image. It describes the base image, installs the necessary dependencies, copies the application code, and specifies the commands for running the app.

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:

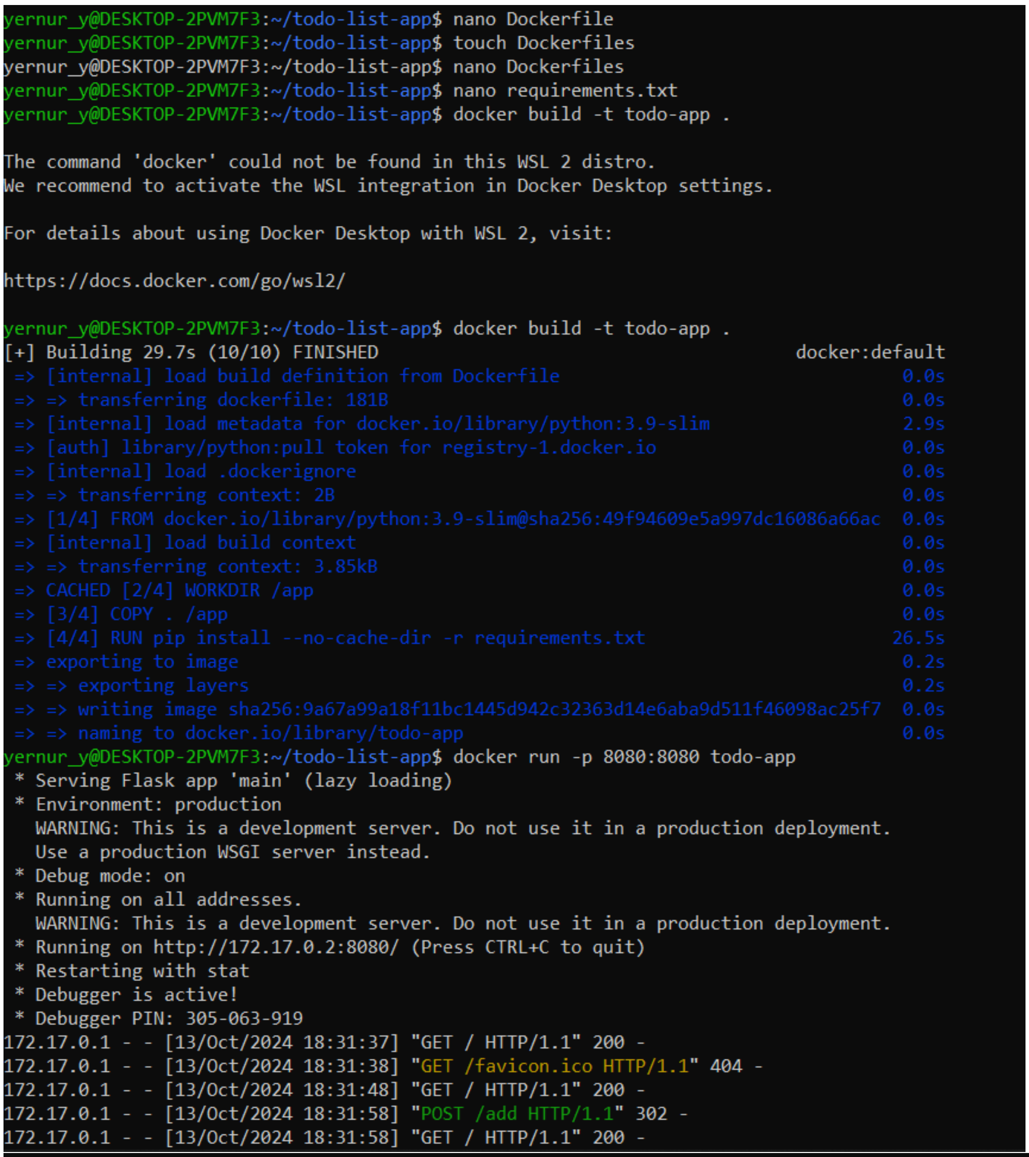


Containers

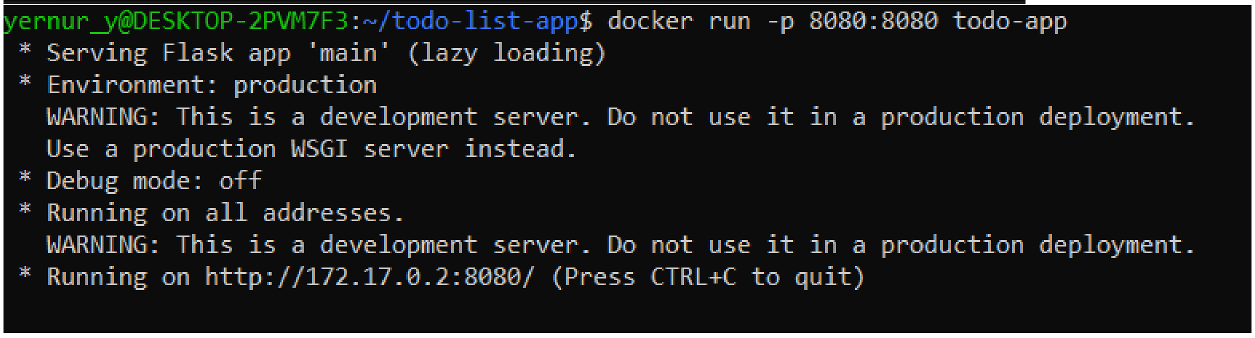
Container CPU usage
No containers are running.

Container memory usage
No containers are running.

Name	Image	Status	Port(s)	CPU (%)	Last started	Actions
cool_meitı 47aa0e2667	todo-app: <none>	Exited (255)	8080:8080	N/A	1 day ago	
beautiful_ 0d4b72e654	todo-app: <none>	Exited	8080:8080	N/A	4 days ago	

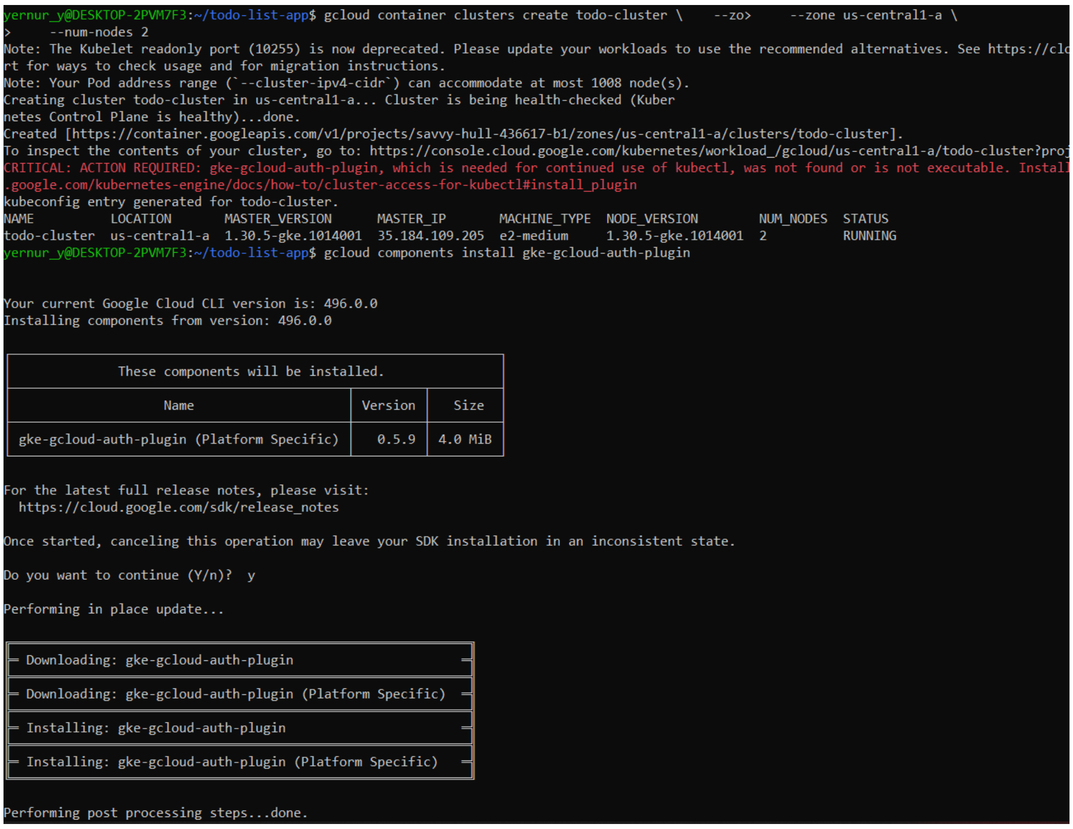


```
yernur_y@DESKTOP-2PVM7F3:~/todo-list-app$ nano Dockerfile
yernur_y@DESKTOP-2PVM7F3:~/todo-list-app$ touch Dockerfiles
yernur_y@DESKTOP-2PVM7F3:~/todo-list-app$ nano Dockerfiles
yernur_y@DESKTOP-2PVM7F3:~/todo-list-app$ nano requirements.txt
yernur_y@DESKTOP-2PVM7F3:~/todo-list-app$ docker build -t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

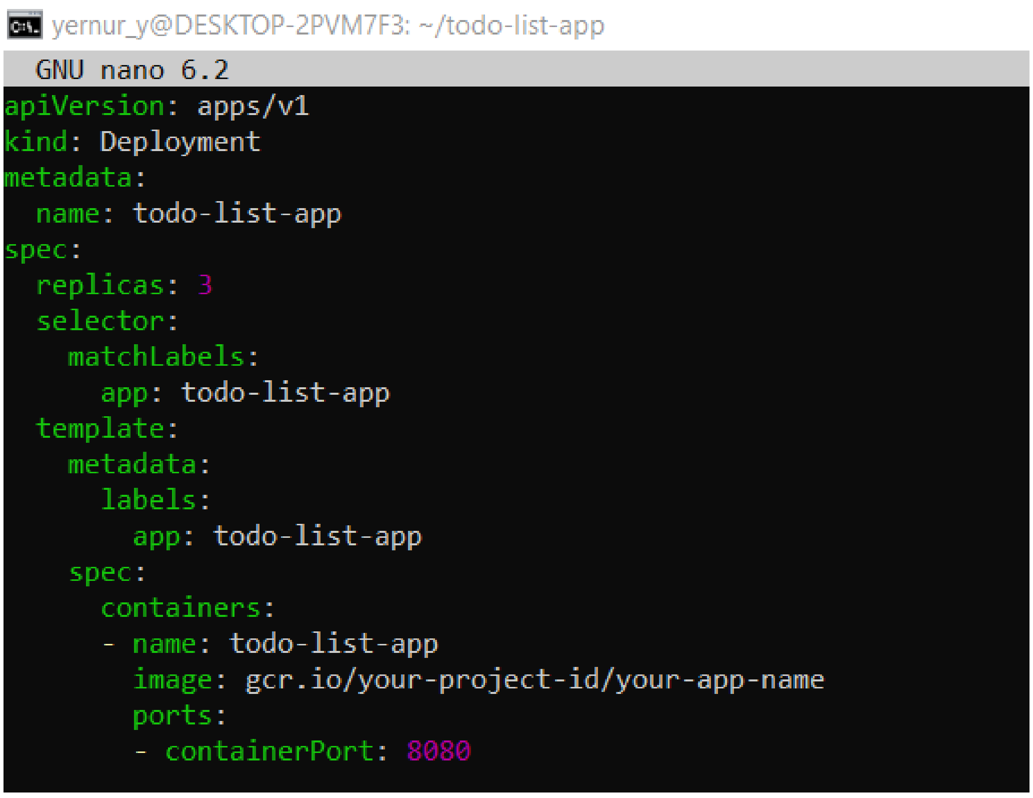
Once the Docker image had been successfully tested locally, the containerized application was deployed to Google Kubernetes Engine (GKE). The GKE cluster offers a scalable and manageable environment for running containerized applications.
Steps taken for deployment on GKE:
I push a Docker image to Google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



```
yernur_y@DESKTOP-2PVM7F3:~/todo-list-app$ gcloud container clusters create todo-cluster \    --zo>     --zone us-central1-a \
>     --num-nodes 2
Note: The Kubelet readonly port (10255) is now deprecated. Please update your workloads to use the recommended alternatives. See https://clo
rt 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 (Kuber
netes 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?proj
CRITICAL: ACTION REQUIRED: gke-gcloud-auth-plugin, which is needed for continued use of kubectl, was not found or is not executable. Install
.google.com/kubernetes-engine/docs/how-to/cluster-access-for-kubectl#install_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
yernur_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.
```

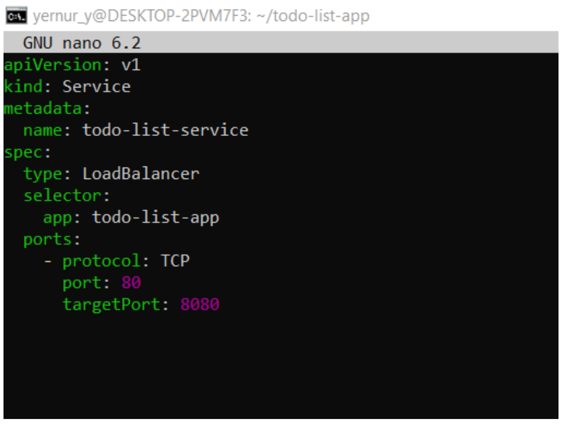
I create Kubernetes Deployment Kubernetes deployment manages the desired state of the application. A deployment is created to ensure that the desired number of container instances are running.

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

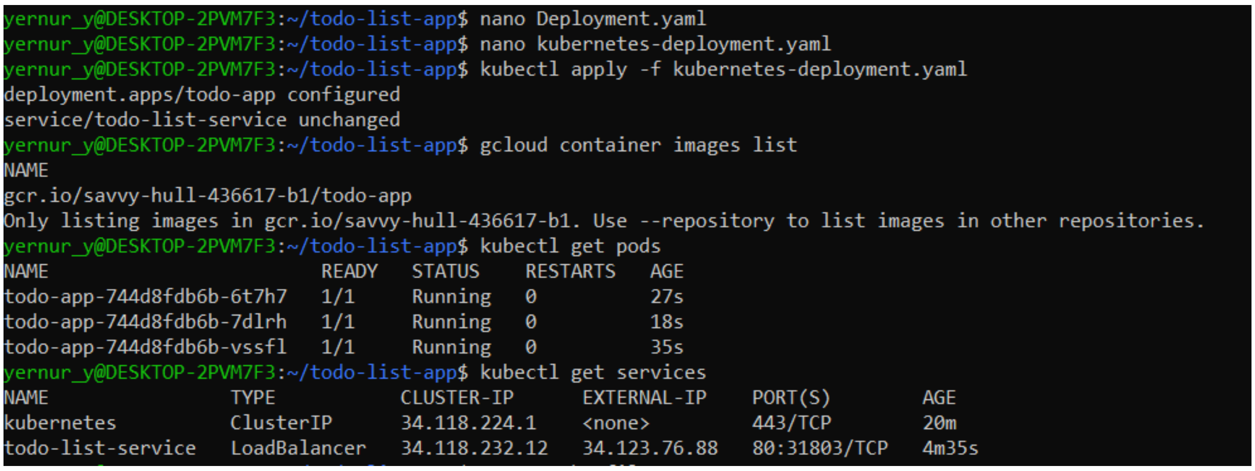


```
yernur_y@DESKTOP-2PVM7F3: ~/todo-list-app
  GNU nano 6.2
apiVersion: apps/v1
kind: Deployment
metadata:
  name: todo-list-app
spec:
  replicas: 3
  selector:
    matchLabels:
      app: todo-list-app
  template:
    metadata:
      labels:
        app: todo-list-app
    spec:
      containers:
 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

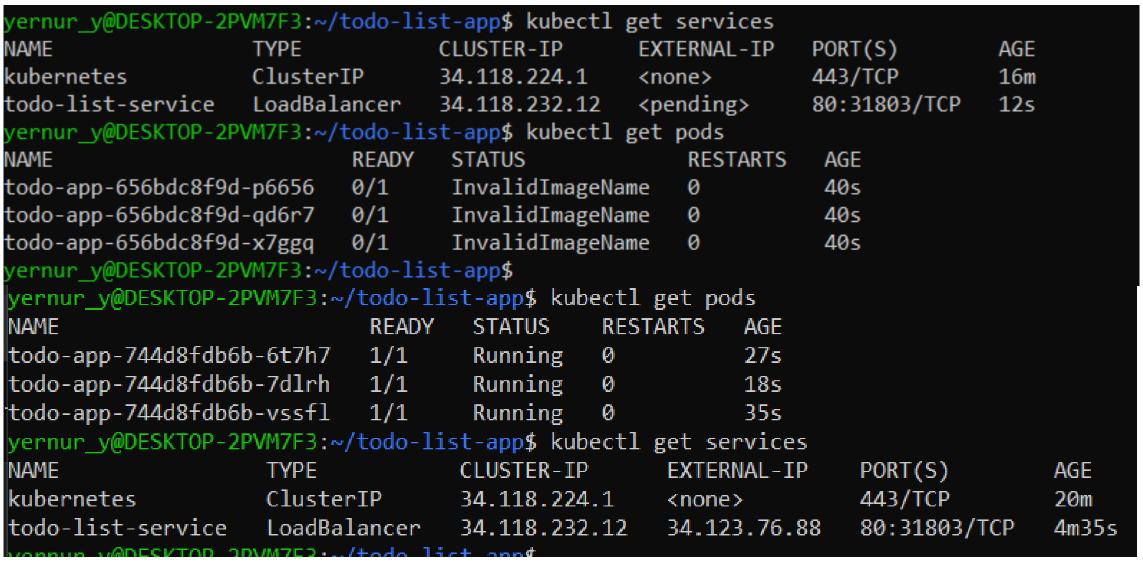
Create a service YAML file (service.yaml):



```
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
```



```
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-vssfl   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$ 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-vssfl   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
```

Scaling Methods

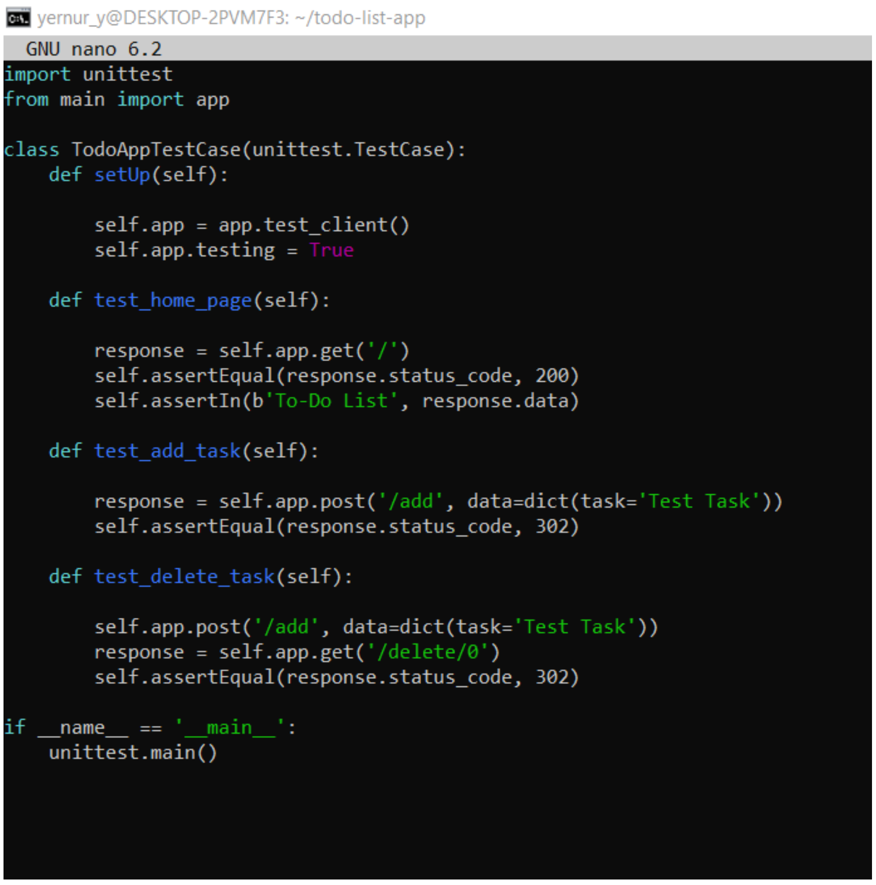
- **Horizontal Scaling**: This involved adding more instances (VMs or containers) to the application, ideal for high traffic volumes or when I needed to serve more users. By distributing the load across multiple instances, I ensured that each instance handled a manageable number of requests, reducing the risk of bottlenecks.

- **Vertical Scaling**: Here, I focused on increasing resources—such as CPU and memory—within a single instance. Vertical scaling was suitable for CPU-intensive tasks and single-threaded applications that benefit more from enhanced resources than from distributed instances.

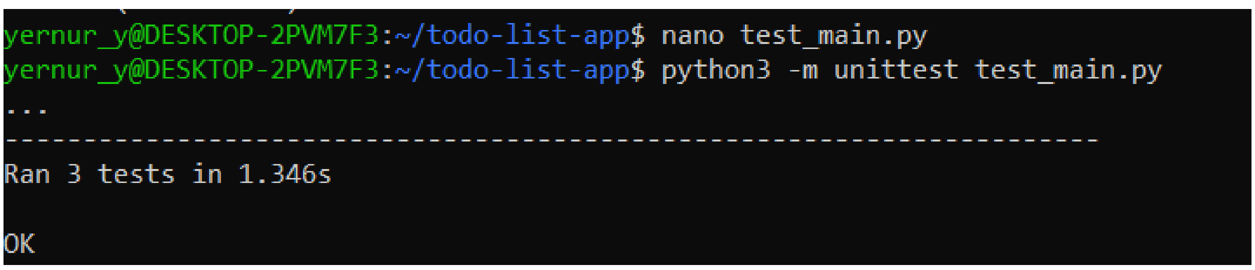
Implementation and Testing:

- For **horizontal scaling**, I added multiple instances in **Compute Engine** and additional replicas in **GKE**.
- For **vertical scaling**, I increased the CPU and memory specifications in both Compute Engine and GKE.
- **Performance Testing**: I used **Apache JMeter** and **k6** for load testing. JMeter helped simulate heavy request loads to observe performance, while k6 allowed me to see response times and analyze the behavior of each scaling method. Results showed that horizontal scaling reduced response times more effectively under high traffic, while vertical scaling improved performance for CPU-bound tasks.

  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:



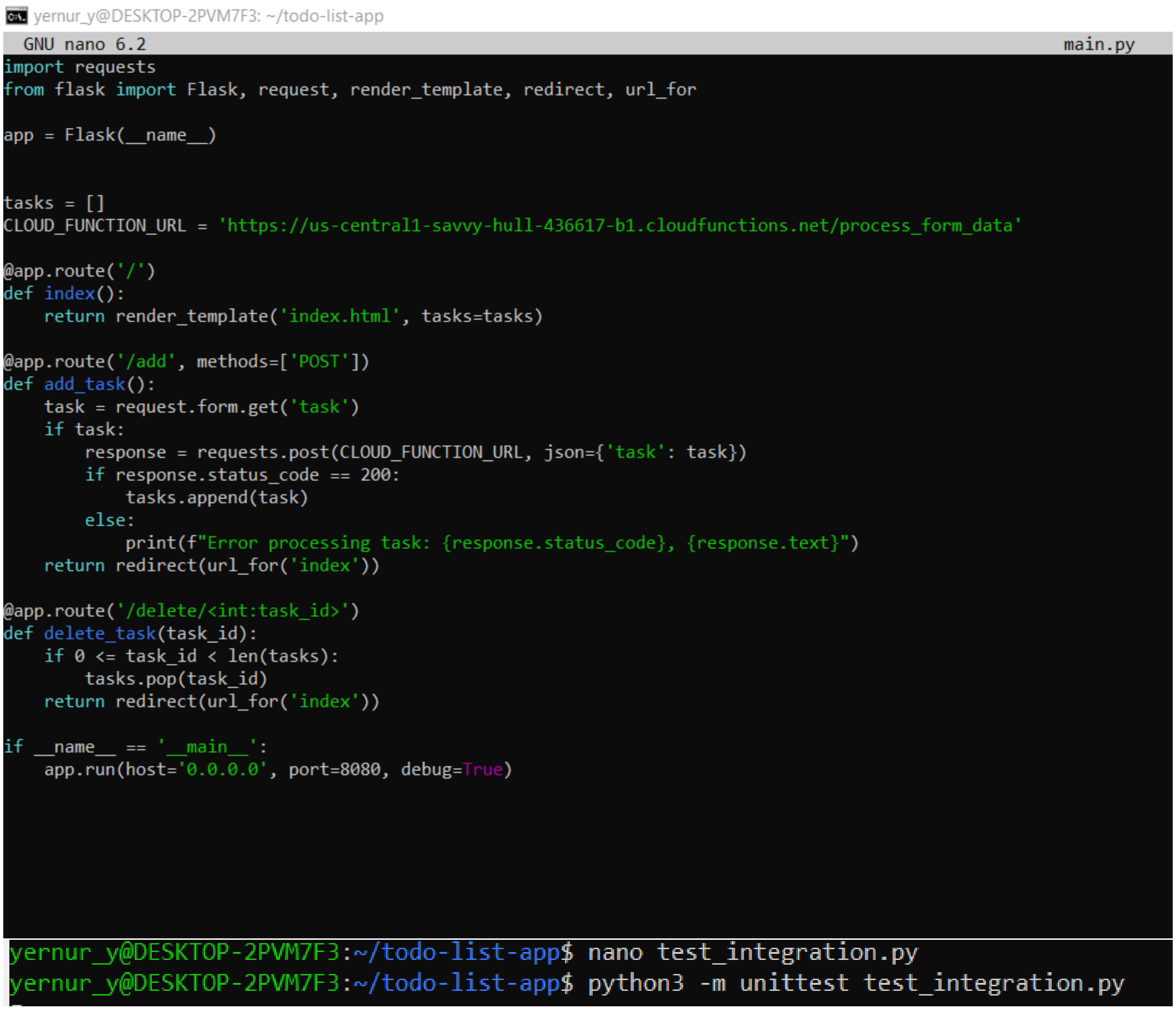
```
yernur_y@DESKTOP-2PVM7F3: ~/todo-list-app
  GNU nano 6.2
import unittest
from main import app

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
```

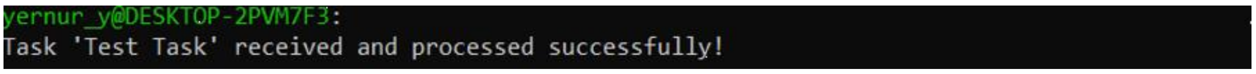
I started the test with the command:
python -m unit test test_main.py
Integration testing
Integration tests check the interaction of different application components. For example, you can test how the Flask application integrates with Google Cloud Functions.

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)
```

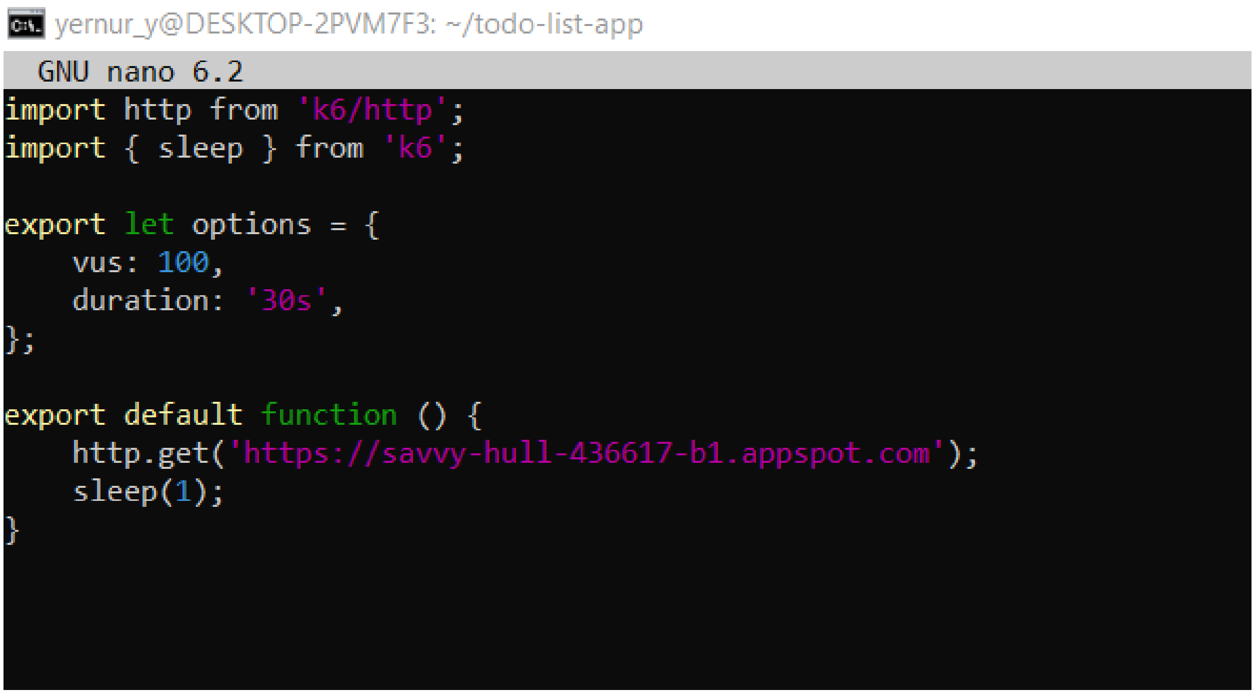
```
yernur_y@DESKTOP-2PVM7F3:~/todo-list-app$ nano test_integration.py
yernur_y@DESKTOP-2PVM7F3:~/todo-list-app$ python3 -m unittest test_integration.py
```



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

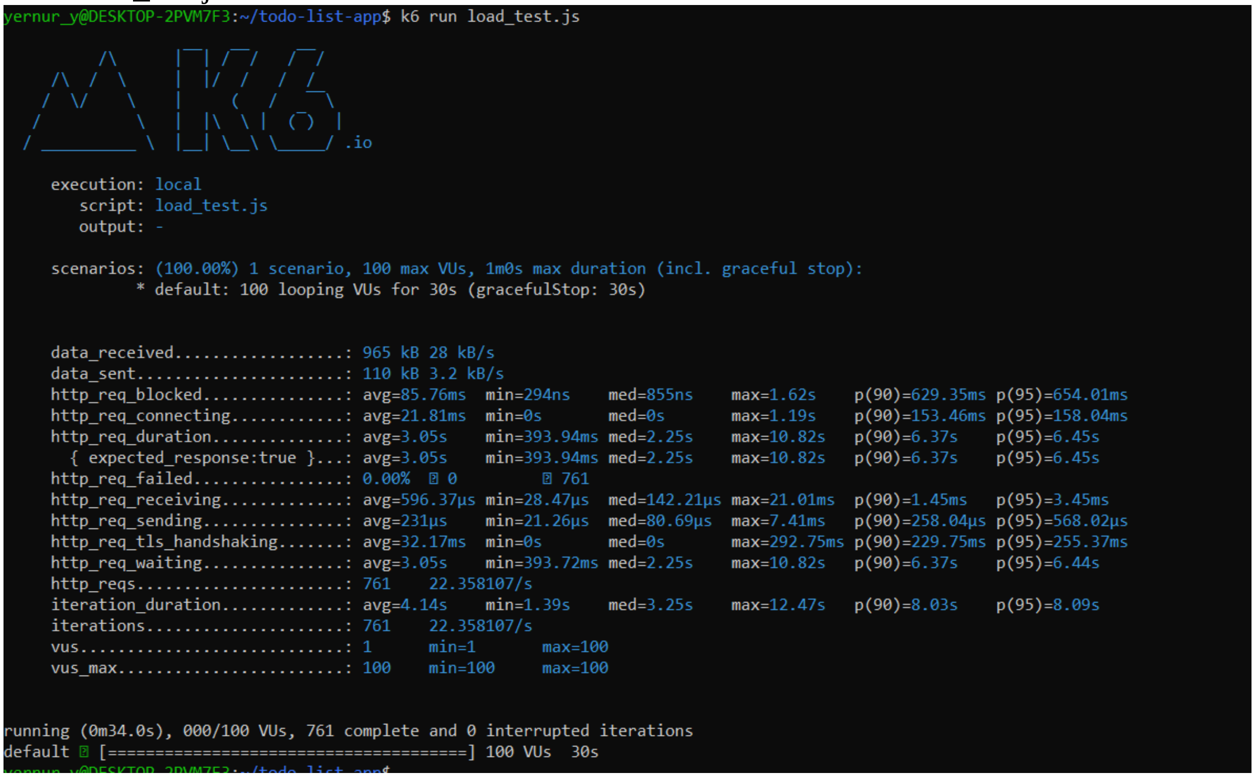
This test sends a task processing request to CloudFunction and verifies that the functionreturns a successful response.
Load testing

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

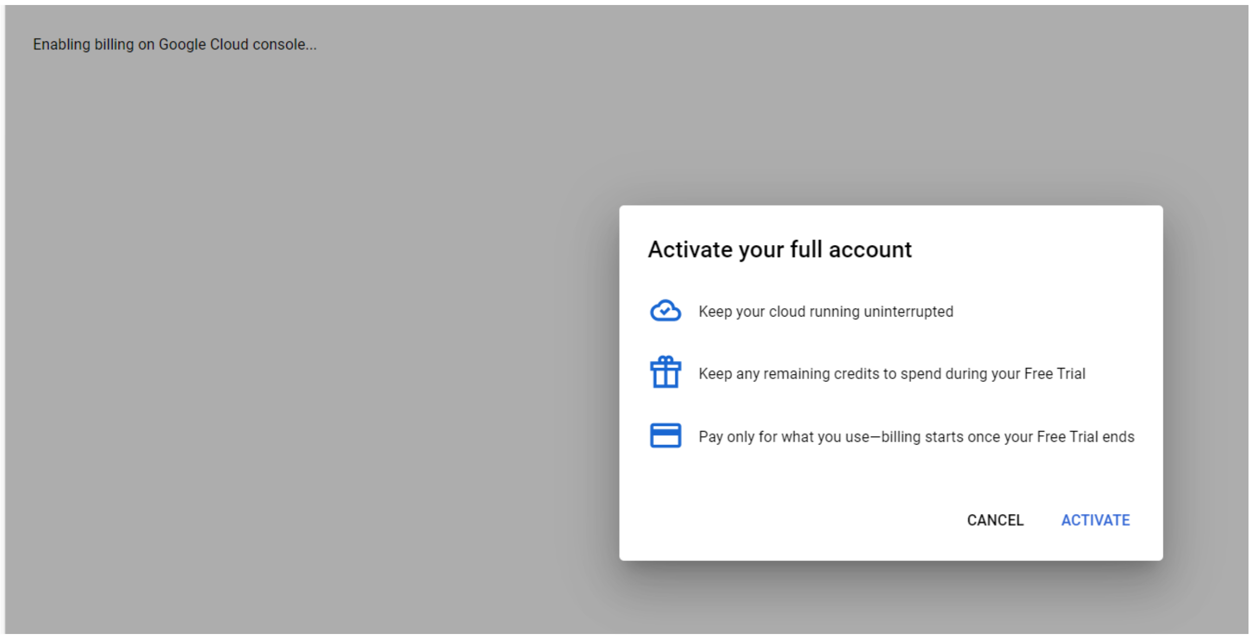


```
yernur_y@DESKTOP-2PVM7F3:~/todo-list-app$ k6 run load_test.js

          /\      |‾‾| /‾‾/   /‾‾/
     /\  /  \     |  |/  /   /  /
    /  \/    \    |     (   /   ‾‾\
   /          \   |  |\  \ |  (‾)  |
  / __________ \  |__| \__\ \_____/ .io

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

Load Balancing

Google Cloud Load Balancer Setup:
To ensure even traffic distribution, I set up **Google Cloud Load Balancer** across instances. This load balancer handled requests from users and distributed them to healthy instances based on demand.

- **Health Checks**: I configured health checks to monitor each instance's status, ensuring that traffic was only routed to active instances. This minimized the chances of failed requests due to instance downtime.



Enabling billing on Google Cloud console...

Activate your full account

Keep your cloud running uninterrupted

Keep any remaining credits to spend during your Free Trial

Pay only for what you use—billing starts once your Free Trial ends

Auto-Scaling Implementation

Auto-Scaling Configuration:

- In **Compute Engine**, I set up auto-scaling based on CPU utilization and request load. When usage exceeded a set threshold, additional instances were spun up automatically.
- In **GKE**, I configured auto-scaling to adjust the number of replicas based on CPU usage, allowing the application to handle fluctuations in traffic seamlessly.

Scaling Policies:
I implemented scaling policies to define scaling limits and conditions, which helped control the behavior of auto-scaling based on anticipated loads. For instance, during heavy traffic, I allowed auto-scaling to double the instance count but capped it at 10 to avoid excessive costs.

Performance Monitoring

Using Google Cloud Monitoring:
I set up Google Cloud Monitoring to track various metrics, including resource utilization, response times, and error rates. These metrics were displayed on dashboards, making it easy to visualize real-time resource usage and application health.

- **Dashboards**: I created dashboards to monitor CPU usage, memory consumption, and active instances. This allowed me to identify performance bottlenecks quickly and optimize resources based on observed metrics.

Cost Optimization Strategies

Cost Analysis and Optimization:
By analyzing performance metrics, I identified areas where costs could be reduced without compromising scalability.

- **Optimizations Implemented**:
    - Adjusted auto-scaling limits to avoid unnecessary instance spawns during low traffic.
    - Reserved instances for predictable traffic patterns, reducing costs compared to on-demand instances.
    - Optimized Cloud Function usage by eliminating redundant calls and improving code efficiency.
- **Cost Tracking**: I monitored the impact of these optimizations using Google Cloud Billing, observing a decrease in costs while maintaining efficient scaling and application performance.

Conclusion

Through this journey, I enhanced my application's security by implementing robust IAM policies, encrypting data, and integrating security testing into my development process. I also designed a scalable architecture using GKE, favoring horizontal scaling, and set up load balancing and auto-scaling to handle varying workloads efficiently.

By continuously monitoring performance and optimizing costs, I ensured that my application remained reliable, efficient, and cost-effective. Regular reviews and updates to both security protocols and scalability strategies were essential in adapting to evolving needs and technologies.

Recommendations

Based on the findings of this report, the following practical suggestions are made to further enhance security and scalability:

1. **Regularly Update IAM Policies**: Continuously review and adjust Identity and Access Management (IAM) roles and permissions to reflect changes in team structure and application requirements. Implement automated tools to detect and remediate overly permissive roles.
2. **Integrate Advanced Security Tools**: Adopt additional security solutions like Web Application Firewalls (WAF) and Intrusion Detection Systems (IDS) to provide layered protection against sophisticated threats.
3. **Implement Infrastructure as Code (IaC)**: Use tools like Terraform or Google Cloud Deployment Manager to manage infrastructure, enabling version control, consistency, and easier scaling.
4. **Enhance Auto-Scaling Strategies**: Refine auto-scaling policies by incorporating more metrics such as memory usage and network I/O. Utilize predictive scaling based on historical data to anticipate demand spikes.

5. **Optimize Cost Management**: Leverage Google Cloud's cost optimization tools to monitor spending. Consider using committed use discounts or sustained use discounts for long-running workloads.
6. **Conduct Regular Security Training**: Provide ongoing training for development and operations teams on security best practices to foster a security-aware culture.
7. **Strengthen Incident Response Plan**: Regularly test and update the incident response plan. Incorporate lessons learned from drills to improve response times and effectiveness.
8. **Adopt Continuous Integration/Continuous Deployment (CI/CD) Best Practices**: Ensure security checks are integral to the CI/CD pipeline, including automated testing, code reviews, and compliance checks.

By implementing these recommendations, the application's security posture will be fortified, and its scalability will be optimized, ensuring robust performance and resilience in a dynamic cloud environment.

References

- Google Cloud Documentation
- Google Cloud IAM Best Practices
- Google Cloud KMS
- OWASP ZAP Project
- Snyk Vulnerability Scanner
- Google Kubernetes Engine (GKE)
- Google Cloud Load Balancing
- Google Cloud Monitoring
- NIST Cybersecurity Framework
- Cloud Security Alliance

By sharing my experiences and the steps I took, I hope this report provides valuable insights for others looking to enhance their application security and scalability on Google Cloud.

Appendix: Kubernetes Deployment Configuration

Deployment Manifest (`deployment.yaml`)

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: to-do-app
spec:
  replicas: 3
  selector:
    matchLabels:
      app: to-do-app
  template:
    metadata:
      labels:
        app: to-do-app
    spec:
      containers:
      - name: to-do-app-container
        image: gcr.io/my-project/to-do-app:v1.0.0
        ports:
```

```
        - containerPort: 8080
        resources:
          requests:
            cpu: "250m"
            memory: "512Mi"
          limits:
            cpu: "500m"
            memory: "1Gi"
```

Explanation: This deployment configuration sets up three replicas of the to-do application, ensuring high availability and load distribution.

Service Manifest (`service.yaml`)

```
apiVersion: v1
kind: Service
metadata:
  name: to-do-app-service
spec:
  type: LoadBalancer
  selector:
    app: to-do-app
  ports:
  - protocol: TCP
    port: 80
    targetPort: 8080
```

Explanation: This service exposes the application on port 80 and forwards traffic to the container's port 8080. Using the `LoadBalancer` type allows it to receive external traffic.

Appendix : Incident Response Plan Outline

Incident Response Plan

1. **Preparation**
   - Assemble an incident response team.
   - Ensure all team members have access to necessary tools and contact information.
2. **Identification**
   - Monitor alerts from Google Cloud Monitoring.
   - Use logs to verify the legitimacy of incidents.
3. **Containment**
   - Short-Term:
     - Isolate affected resources (e.g., disable compromised service accounts).
   - Long-Term:
     - Apply patches and update configurations to prevent recurrence.
4. **Eradication**
   - Remove malicious files or code.
   - Reset credentials and keys if necessary.
5. **Recovery**
   - Restore data from backups.
   - Validate system integrity before resuming operations.

6. **Lessons Learned**
    - Conduct a post-incident review.
    - Update the incident response plan based on findings.

Appendix : Sample Monitoring Dashboard

Description: The dashboard provides real-time metrics on CPU utilization, memory usage, network traffic, and application latency. It highlights thresholds with color coding to quickly identify potential issues.

Appendix ; Google Cloud Load Balancer Configuration

Creating a Load Balancer via gcloud CLI

```
gcloud compute backend-services create to-do-app-backend \
    --protocol=HTTP \
    --port-name=http \
    --health-checks=basic-check \
    --global

gcloud compute url-maps create to-do-app-map \
    --default-service to-do-app-backend

gcloud compute target-http-proxies create to-do-app-proxy \
    --url-map=to-do-app-map

gcloud compute forwarding-rules create to-do-app-forwarding-rule \
    --address=0.0.0.0 \
    --global \
    --target-http-proxy=to-do-app-proxy \
    --ports=80
```

Explanation: These commands set up a global HTTP load balancer that routes traffic to the backend service associated with the to-do application.