School of Information Technology and Engineering

Cloud Application Development
Assignment 4
Title: **Application Security Best Practices and Scaling Applications on Google Cloud**

Done by: **Altair Kabdrakhmanov**
4th year student, ID: 21B030829,
Information Systems
Date of submission: 24th November

Almaty, Fall 2024

Executive Summary

This report shows how to improve security and scalability for apps on Google Cloud. For security was used IAM to control access, encrypted data with Google Cloud KMS, and added HTTPS for safe data transfer. Also it was set up monitoring, tested the app for security issues, and made a plan to handle security problems.

For scalability, it was used horizontal and vertical scaling to handle more users and traffic. Tools like Kubernetes auto-scaling and load balancing helped keep the app running smoothly. To save costs, we chose the right storage, used serverless options, and set alerts to track usage and spending.

Table of Contents

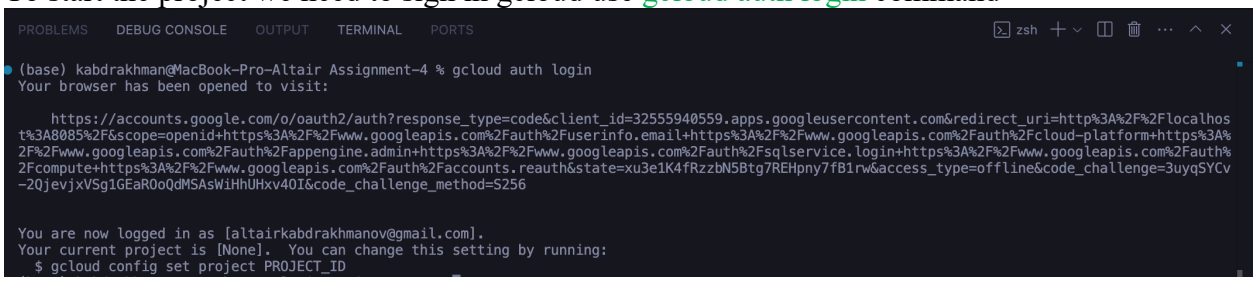
1. Introduction

Google Cloud is a powerful platform that helps businesses run applications and store data on cloud. It offers many services, like computing power, storage, and tools for managing apps. Google Cloud is used by companies around the world because it is reliable, flexible, and easy to scale.

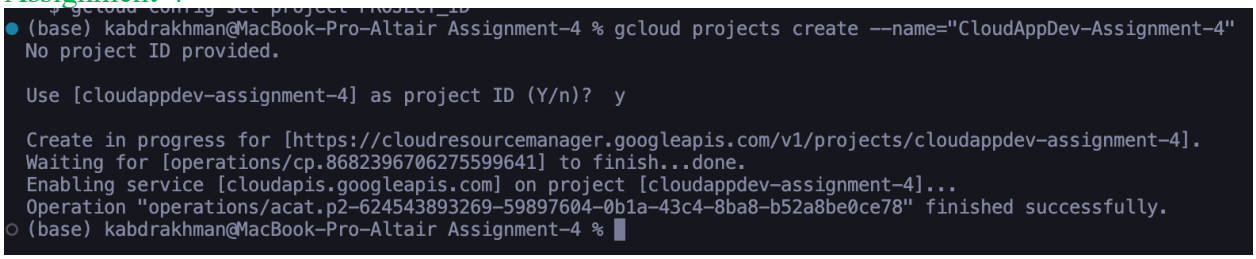
In cloud applications, security and scalability are very important. Security keeps data safe from unauthorized access, while scalability ensures the app can handle more users and work as needed. Without good security, sensitive data can be exposed, and without scalability, the app can become slow or stop working during busy times.

The purpose of this report is to explore the best practices for security and scaling applications on Google Cloud. It will explain how to secure an app using Google Cloud's tools and how to make the app scale easily to handle growing demand. The scope of this report includes setting up security, testing for vulnerabilities, monitoring, handling incidents, and optimizing costs while scaling the app on Google Cloud.

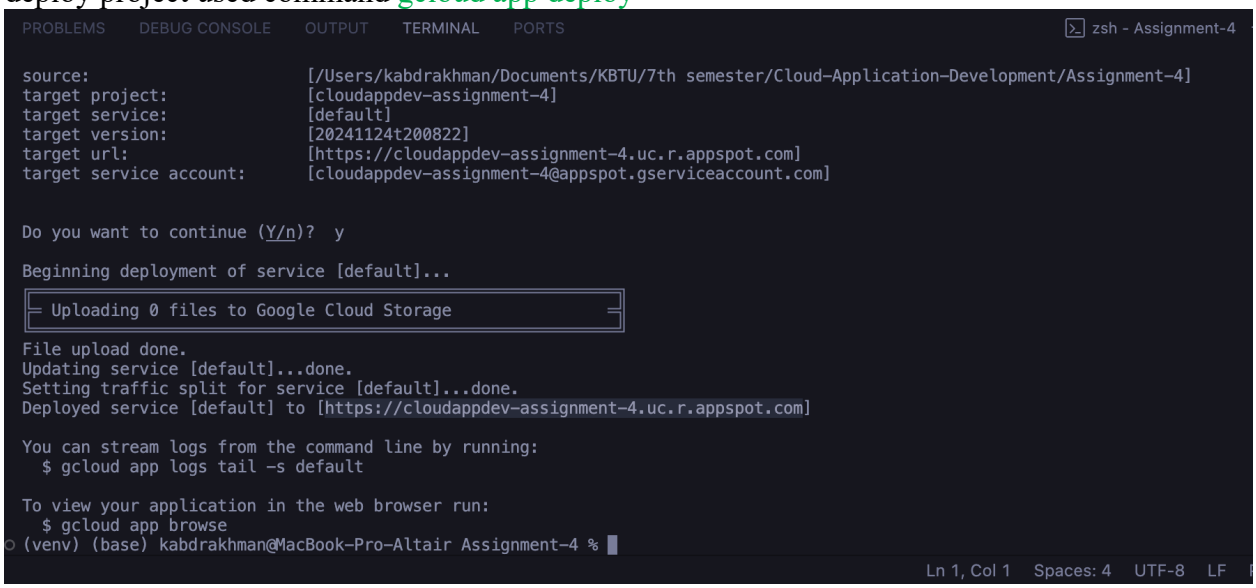
To start the project we need to sign in gcloud use gcloud auth login command



and to create a project with this command: gcloud projects create --name="CloudAppDev-Assignment-4"



For this assiggmnet was used simple Flask application “Phonebook” with CRUD Operations. To deploy project used command gcloud app deploy



2. Application Security Best Practices

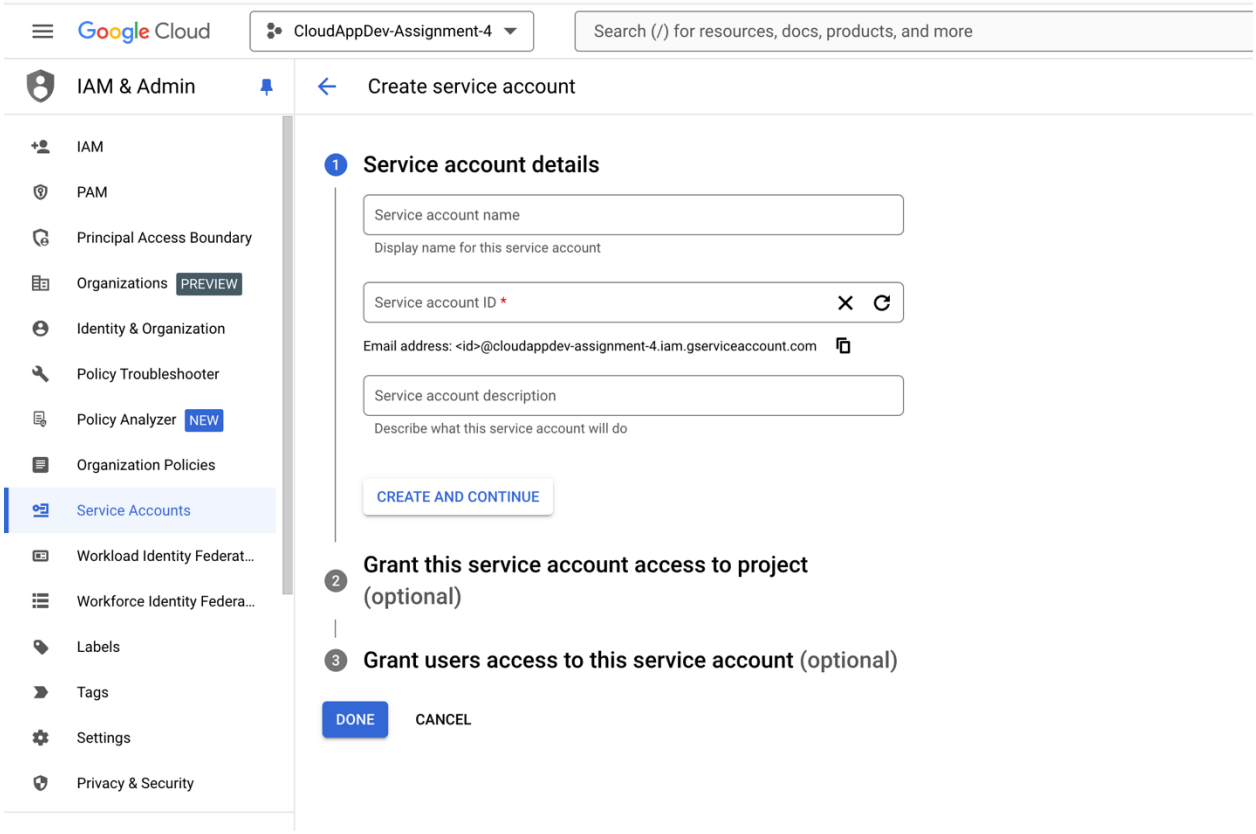
2.1. Overview of Cloud Security

Cloud security is set of practices and technologies used to protect data, applications, and infrastructure in cloud environments. It focuses on data confidentiality, integrity, and availability. Google Cloud provides built-in tools like Identity and Access Management (IAM), encryption, monitoring, and logging to implement security.

2.2. IAM Configuration

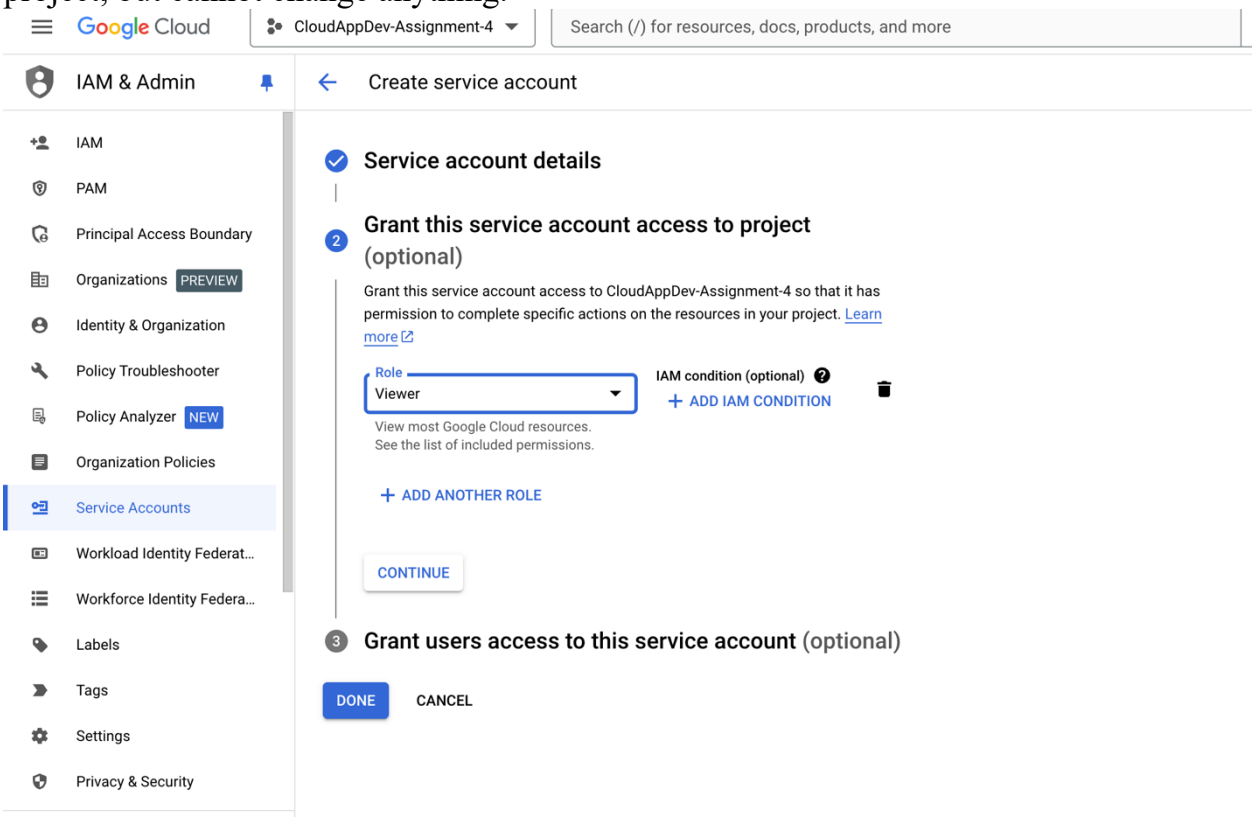
Identity and Access Management (IAM) is a Google Cloud tool that helps control who can access your resources and what actions they can perform. It allows you to create roles with specific permissions, follow the principle of least privilege to give only the necessary access, and set conditions like time or location to limit access further.

Now move to creation of a service account for my application and assign the principle of least privileage we need to go to the following link https://console.cloud.google.com/iam-admin. Select “Service Accounts” option on the left table. Click on “CREATE SERVICE ACCOUNT”.

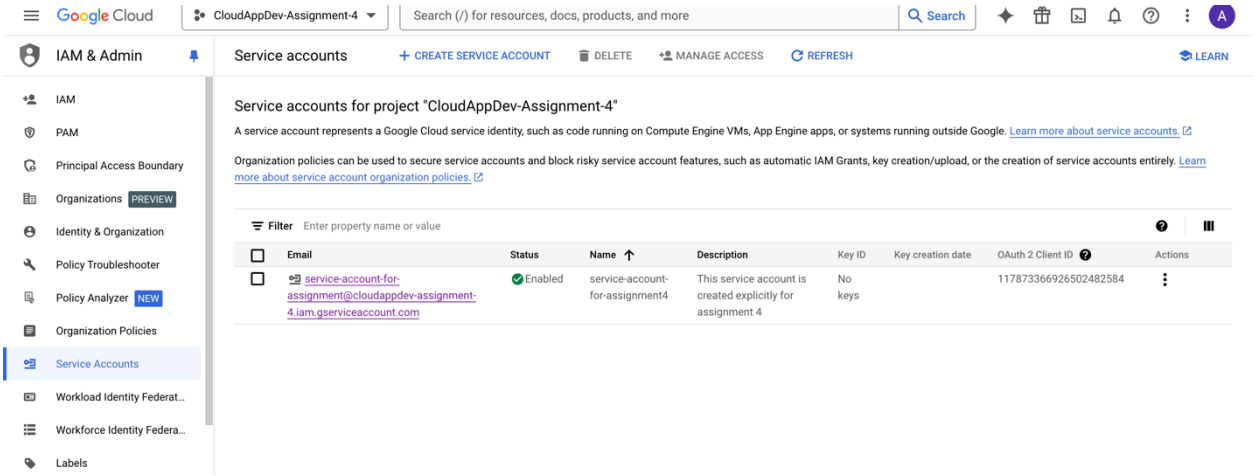


You will see this window. Write name and go the second option.

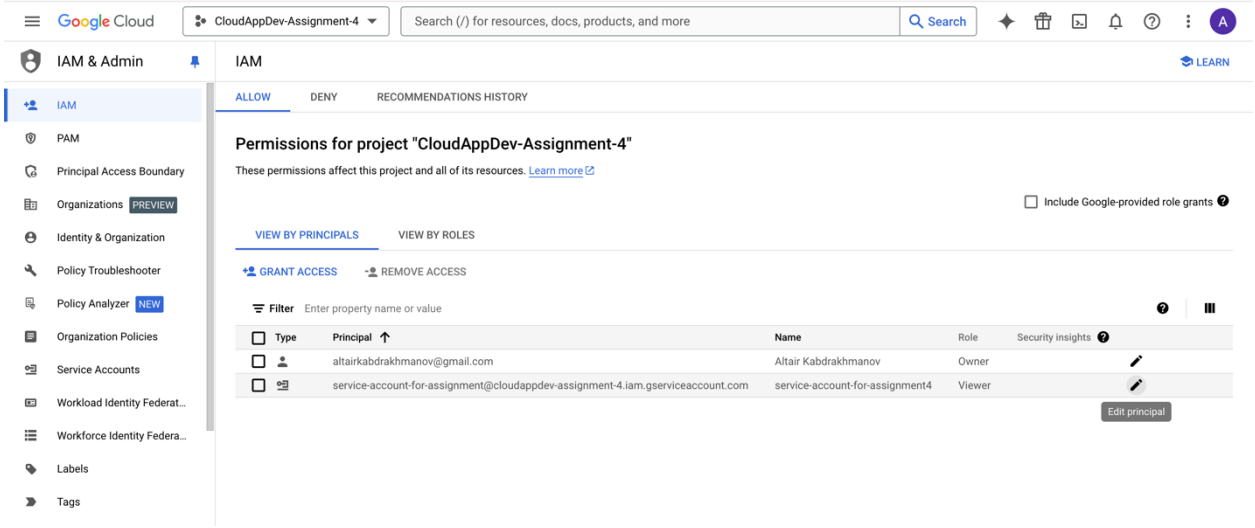
I provide Role Viewere for this service account. That means it can view everythin in my project, but cannot change anything.



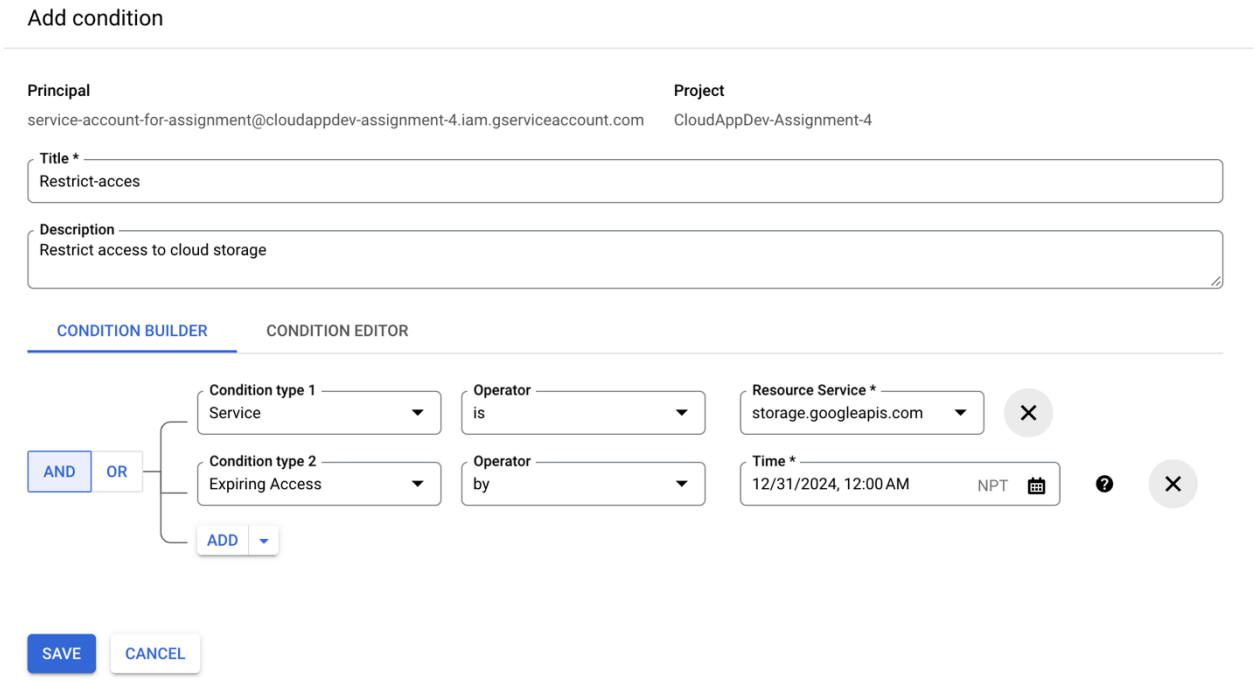
As result we can see list of available service accounts in our project.



Now we can configure IAM Conditions to Restrict Access Based on Attributes (ABAC). Select “IAM” option and click edit button on just created service account name.

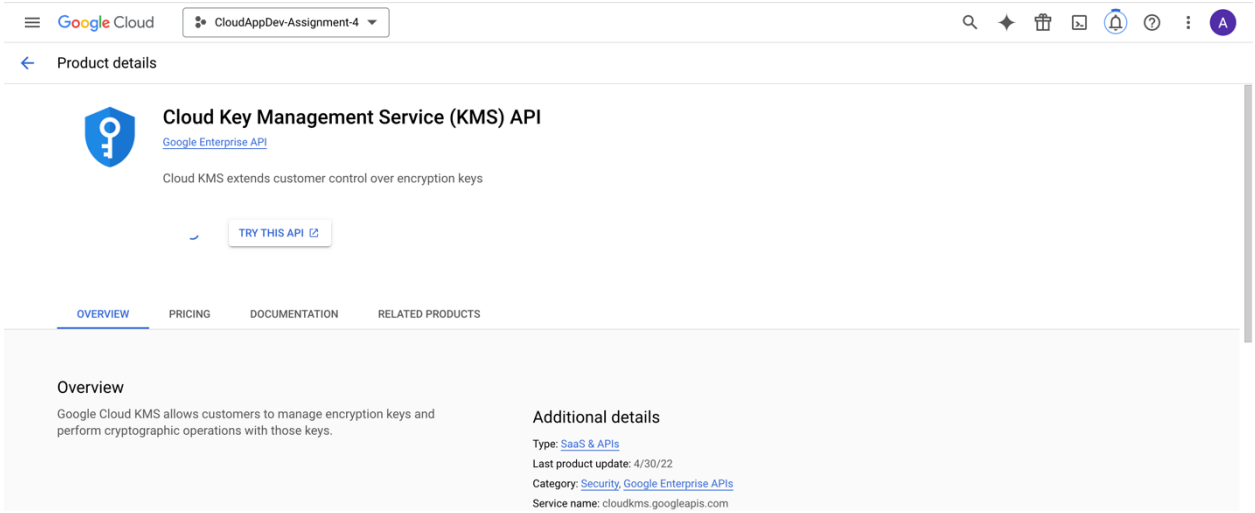


Click on IAM condition (optional) button and create own logic to restict access. I provide acces to service account until 31th December, then it will not available. Click save.

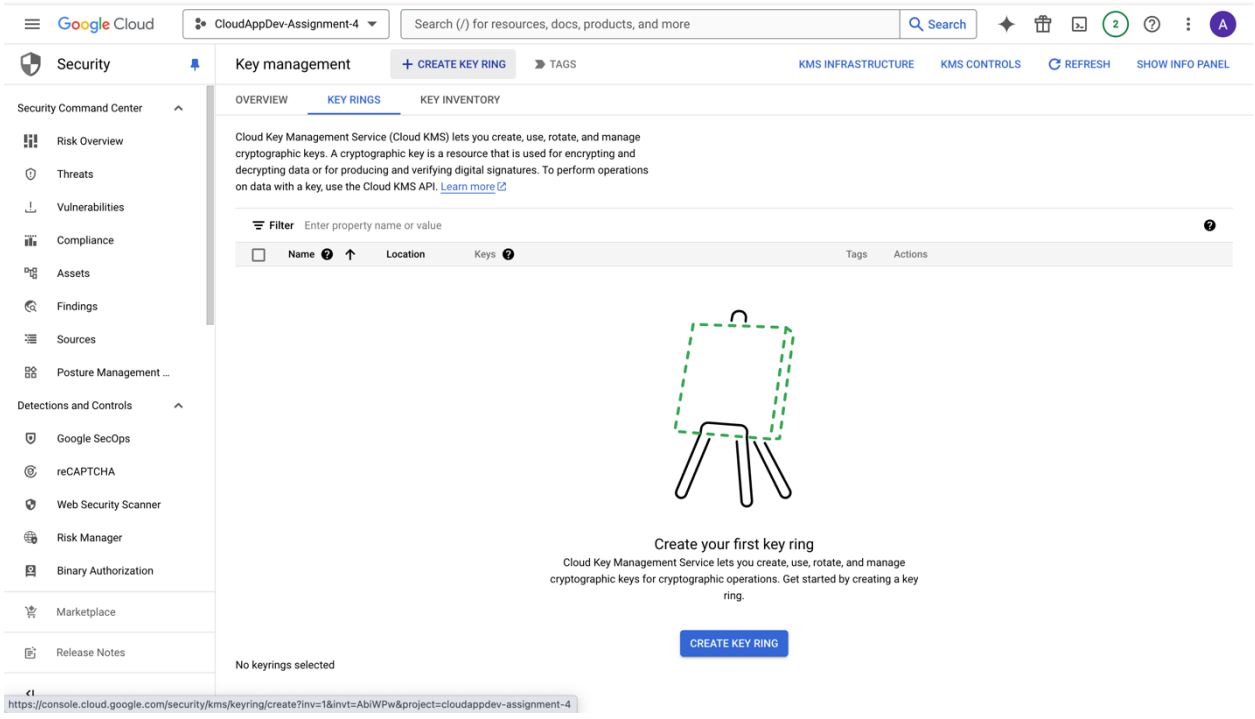


2.3. Data Protection

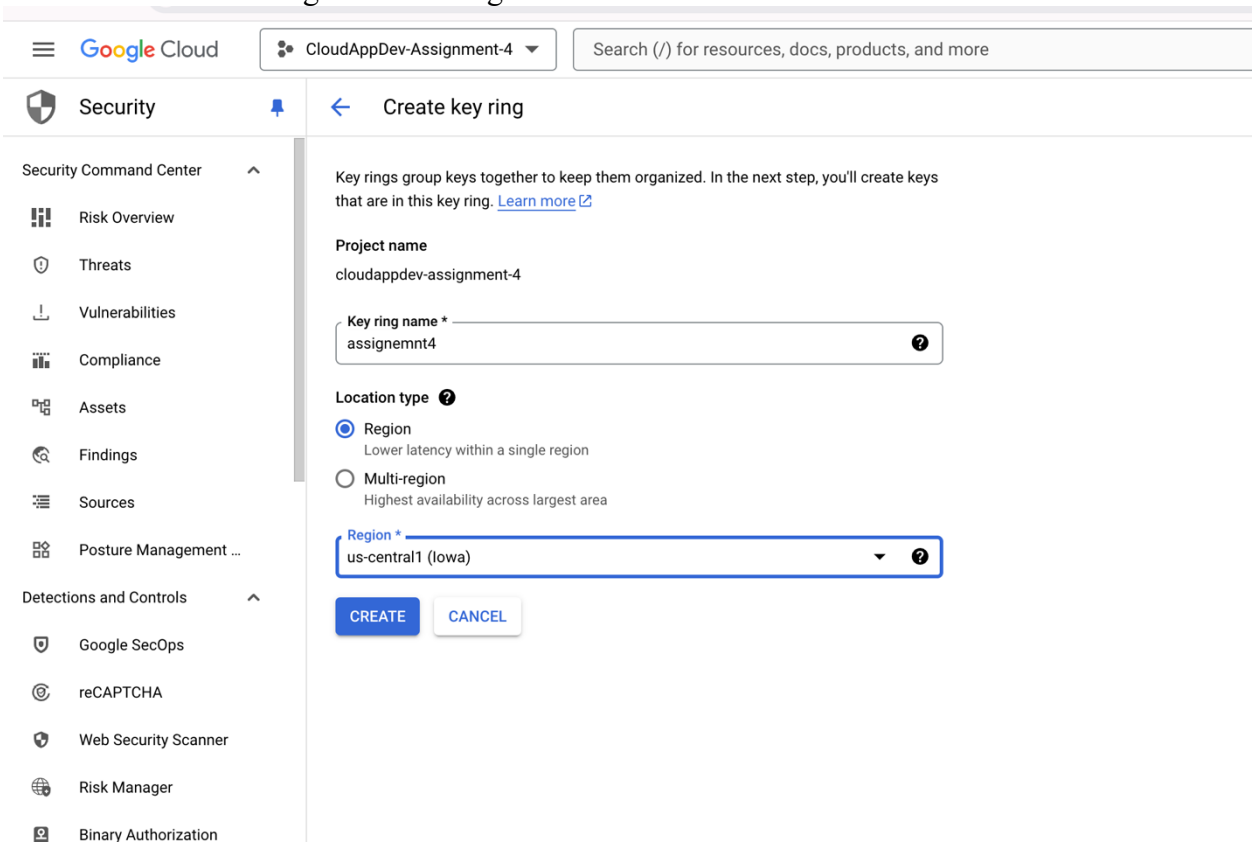
We will use **Google Cloud KMS to set up encryption for data at rest.** First we need to enable Cloud KMS API.



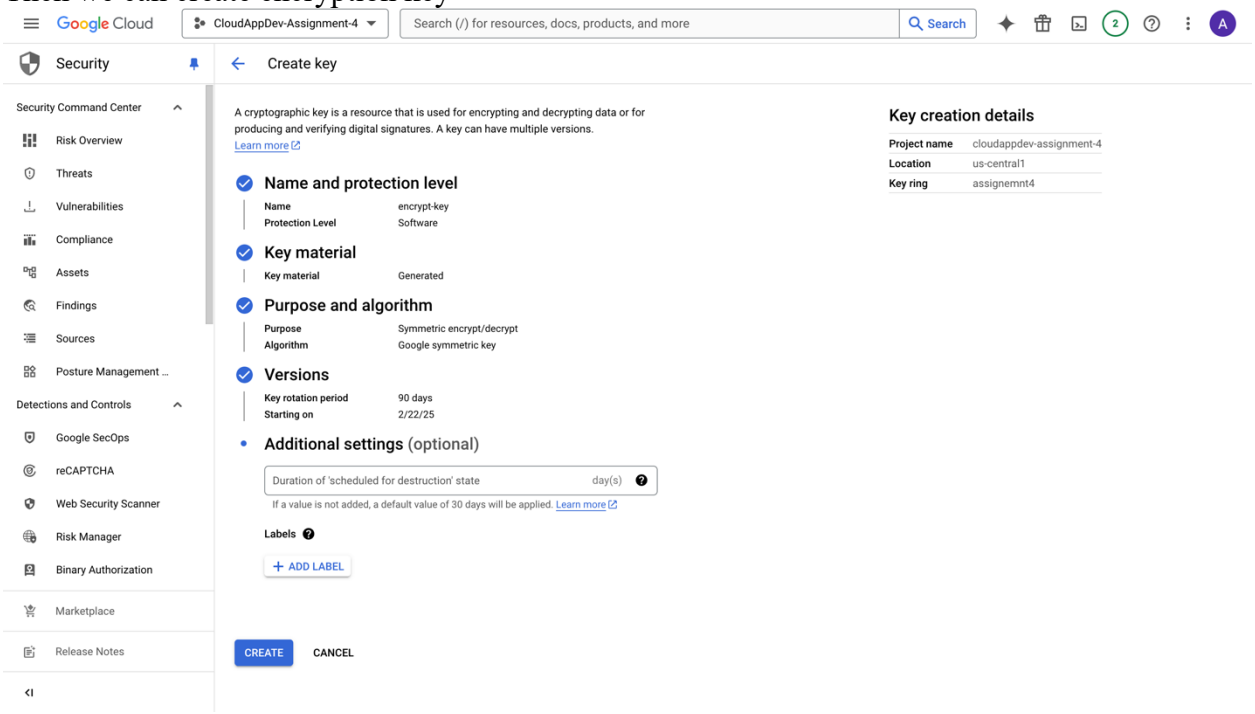
Select here “CREATE KEY RING”



Here write name of ring and select region.

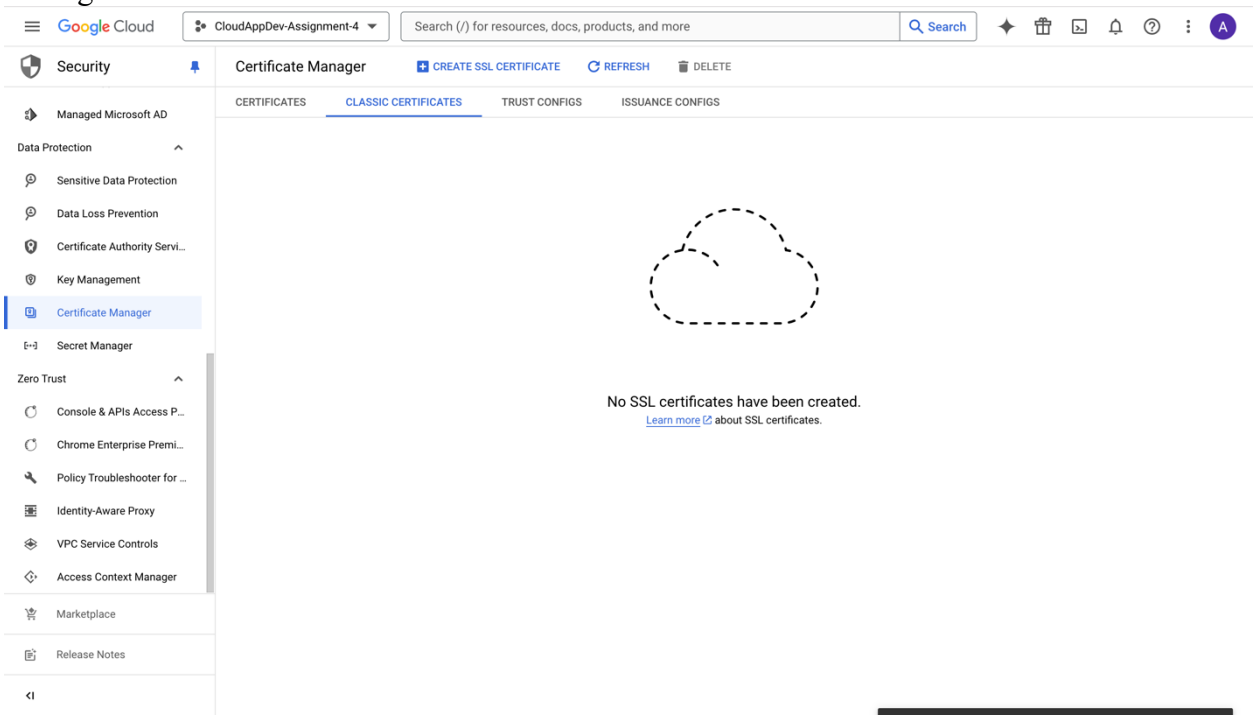


Then we can create encryption key

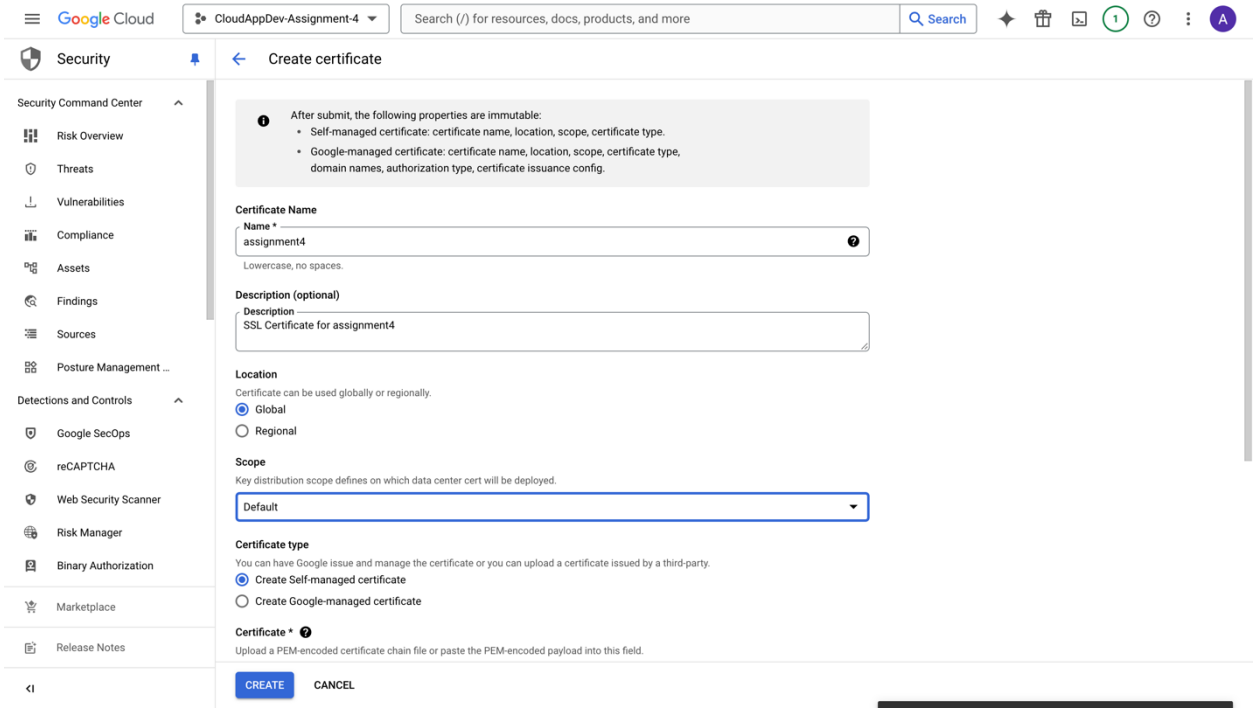


Use HTTPS for data in transit and configure load balancers to enforce SSL.
To use https is necessary to create SSL certificate. Go to security page and “Certificate

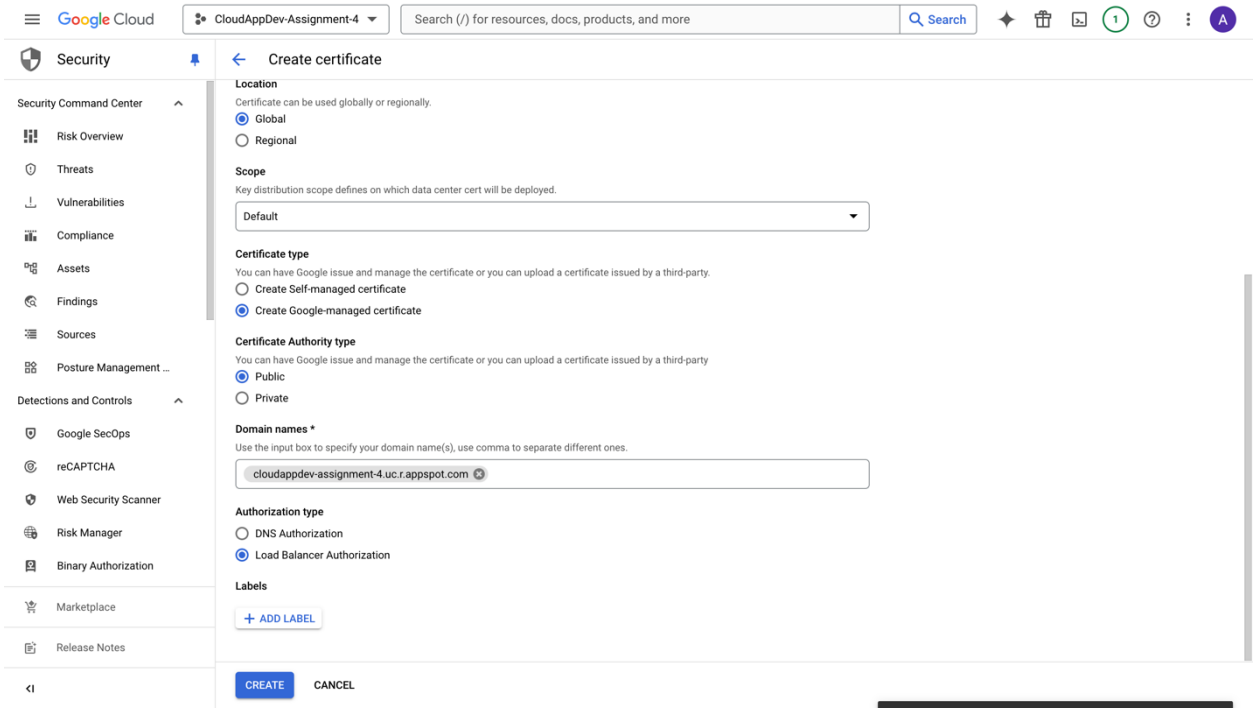
Manager" section and click on button "CREATE SSL CERTIFICATE".



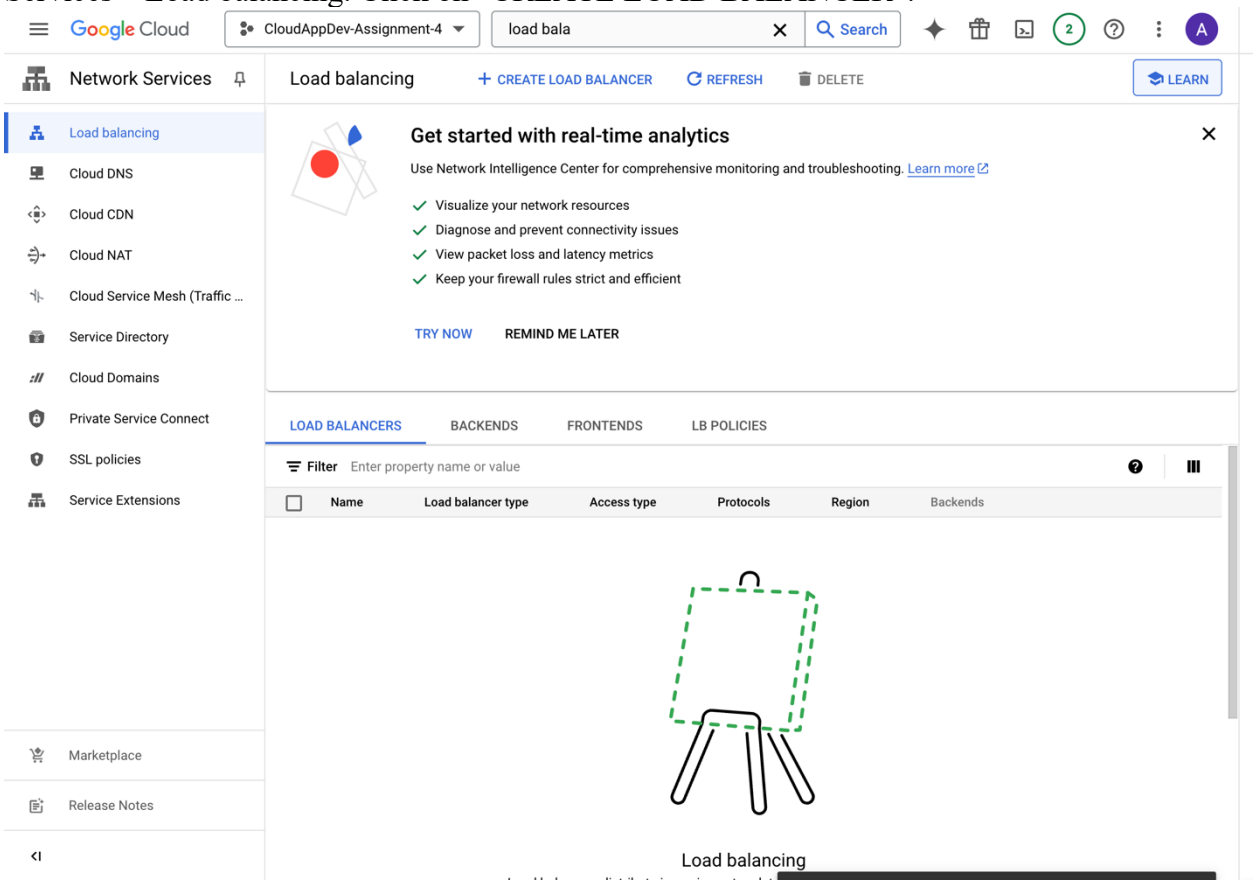
Fill the necessary fields.



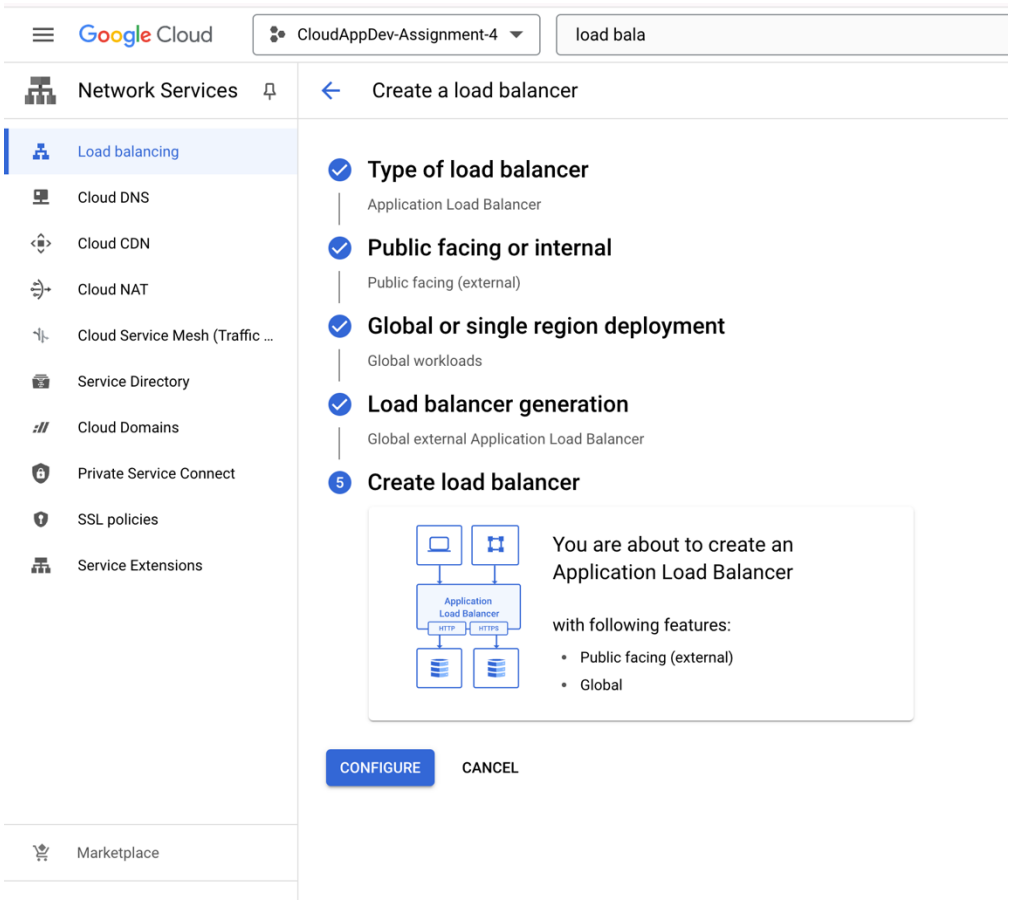
Where Domain name write the application domain name. For me it is cloudappdev-assignment-4.uc.r.appspot.com



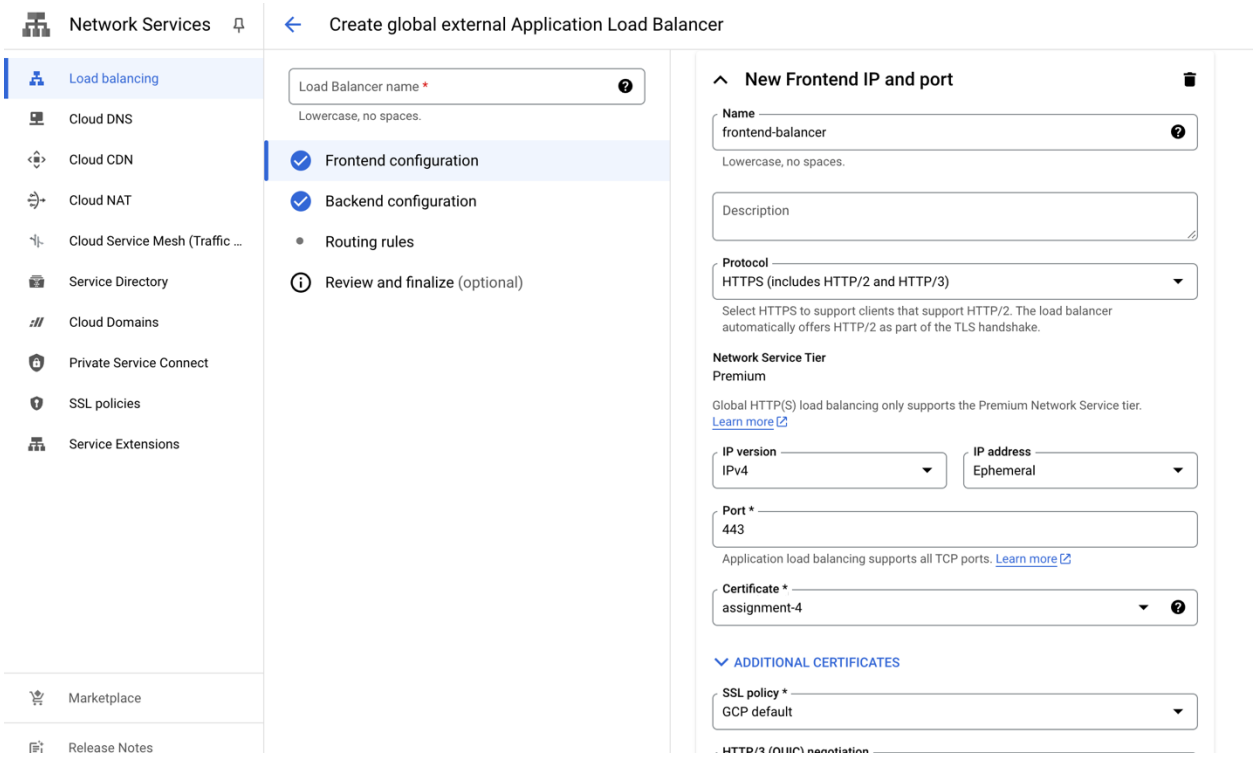
The next is **configure Load Balancer** for my application. Go to search and find Network Services – Load balancing. Click on “CREATE LOAD BALANCER”.



The configuration for our load balancer.



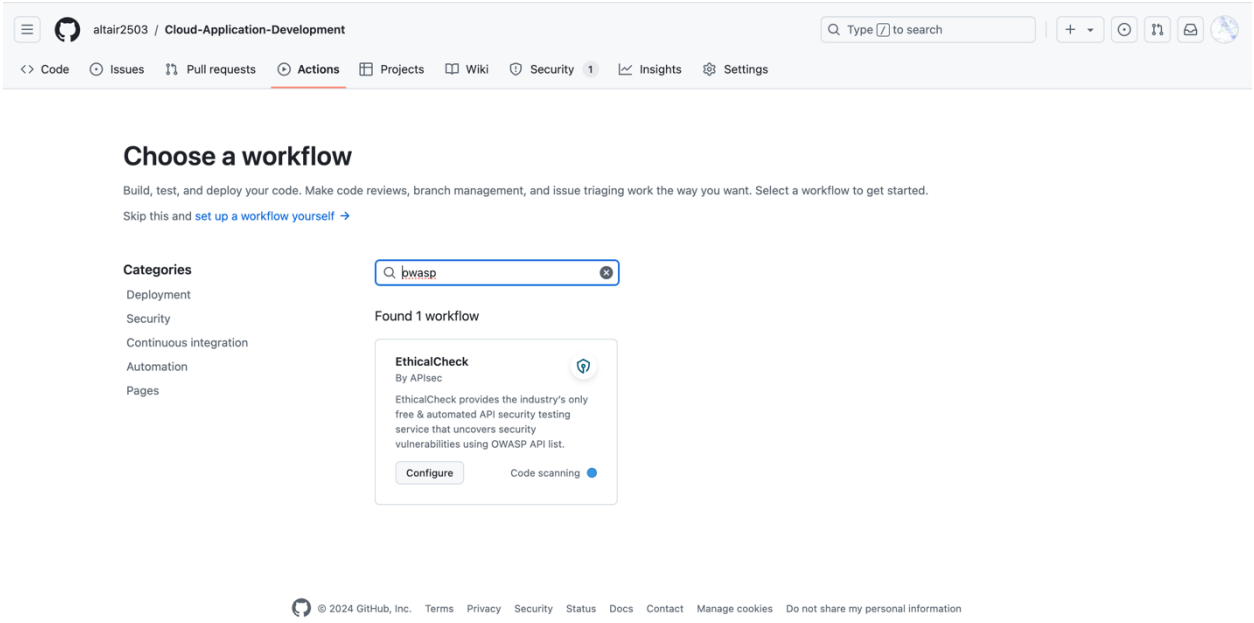
On frontend write name and select before creted SSL certificate



2.4. Security Testing

We need to add a security scanning tool to the CI/CD pipeline. For that I will use GitHub Actions with OWASP ZAP tool. When we make commits, the tool will analyze the code and create a report for us.

For that go to your git repository. Select actions and find by search OWASP. Click configure and continue

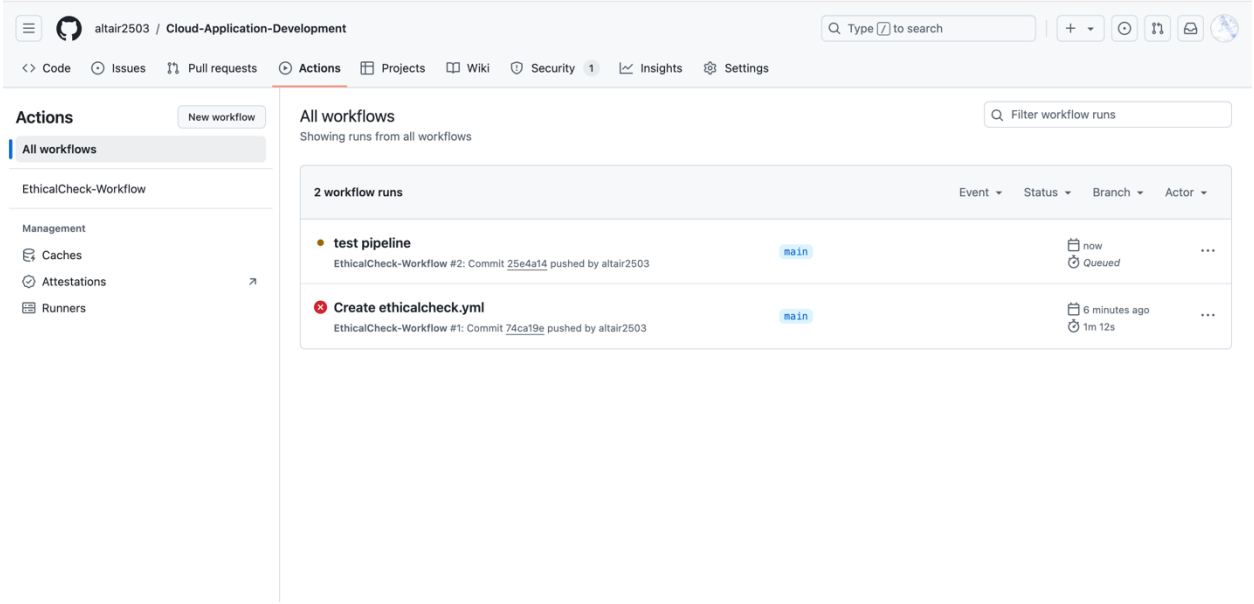


Used configuration for that:

Assignment-4 > owasp-zap.yml

```yaml
name: OWASP ZAP

on:
  push:
    branches:
      - master

jobs:
  zap_scan:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout code
        uses: actions/checkout@v2

      - name: Set up OWASP ZAP
        uses: zaproxy/action-full-scan@v0.1.0
        with:
          target: "https://cloudappdev-assignment-4.uc.r.appspot.com"
          docker_name: "owasp/zap2docker-stable"
          cmd_options: "-r zap_report.html"

      - name: Upload OWASP ZAP Report
        uses: actions/upload-artifact@v3
        with:
          name: zap-report
          path: zap_report.html
```

With new commit it shows github workflow that scan our application.



Now when we do commit it generates report in github actions. The report itself:

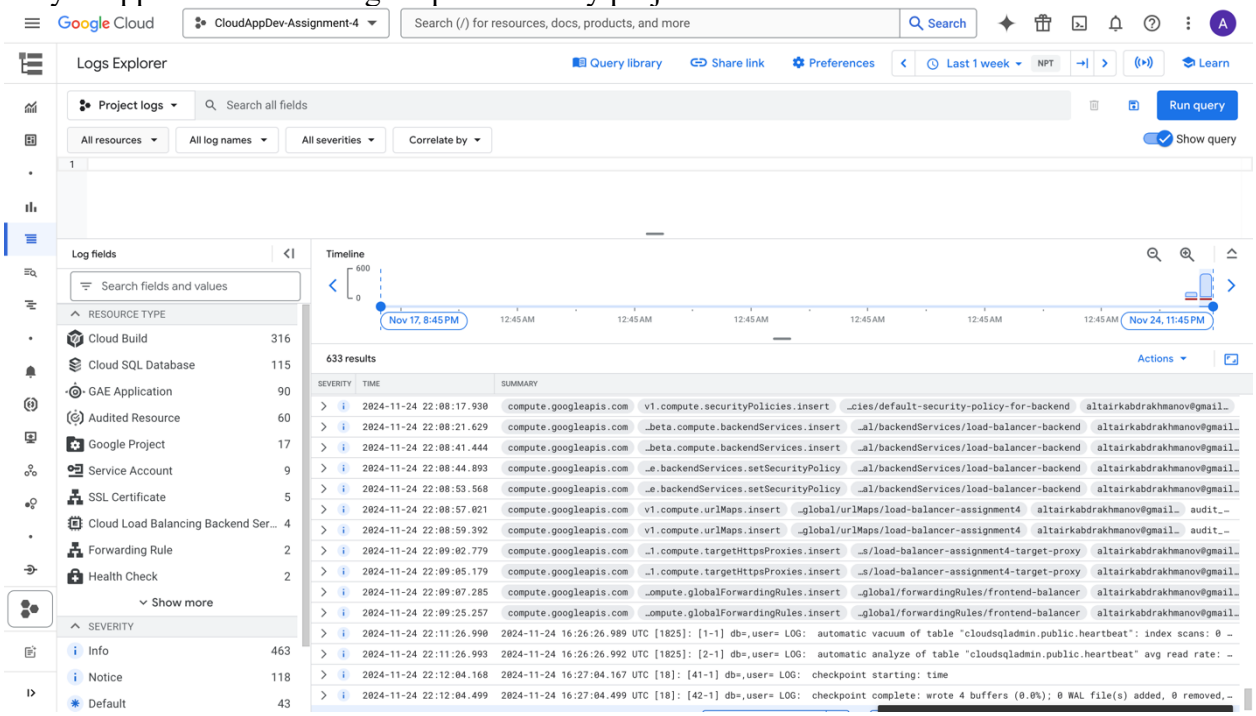
Risk Level	Number of Alerts
High	0
Medium	0
Low	2
Informational	4
False Positives:	0

Alerts

Name	Risk Level	Number of Instances
Content Security Policy (CSP) Header Not Set	Medium	0
Missing Anti-clickjacking Header	Medium	0
Permissions Policy Header Not Set	Low	0
Server Leaks Version Information via "Server" HTTP Response Header Field	Low	0
X-Content-Type-Options Header Missing	Low	2
Storable and Cacheable Content	Informational	4

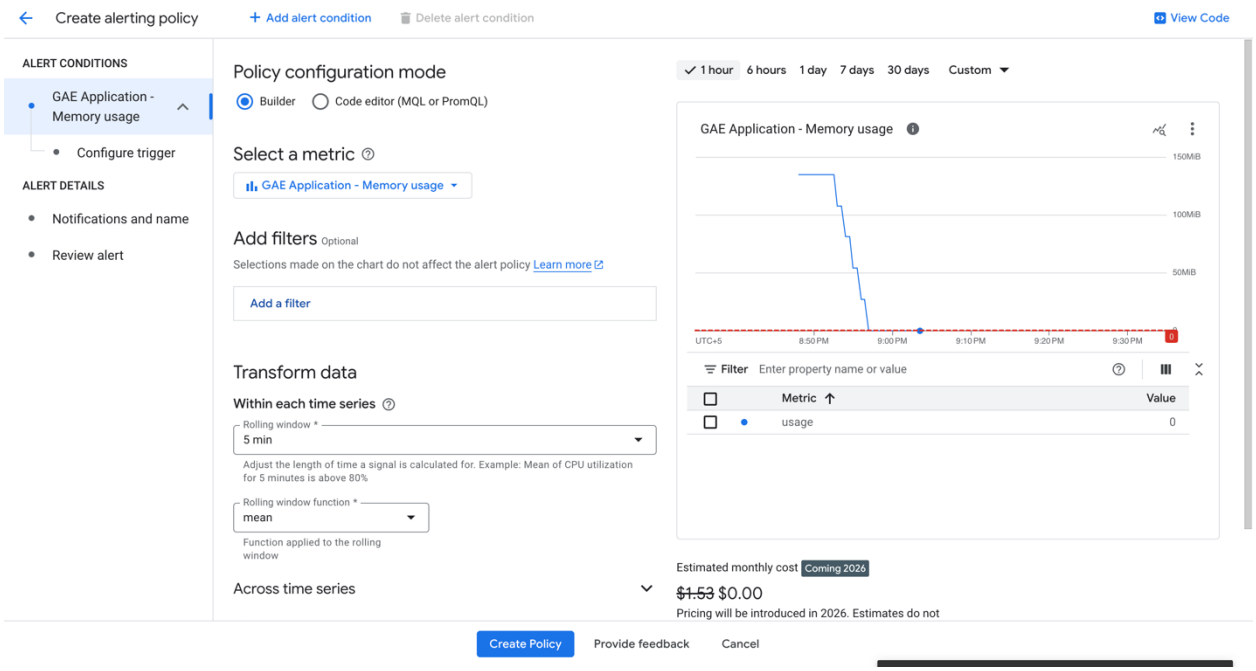
2.5. Monitoring and Logging

First we need to enable Google Cloud Audit Logs for my project. To do that search logs explorer and activate it. If you did gcloud app deploy (as I did before) it will automatically enable logging for you applicatin. Below logs explorer for my project.

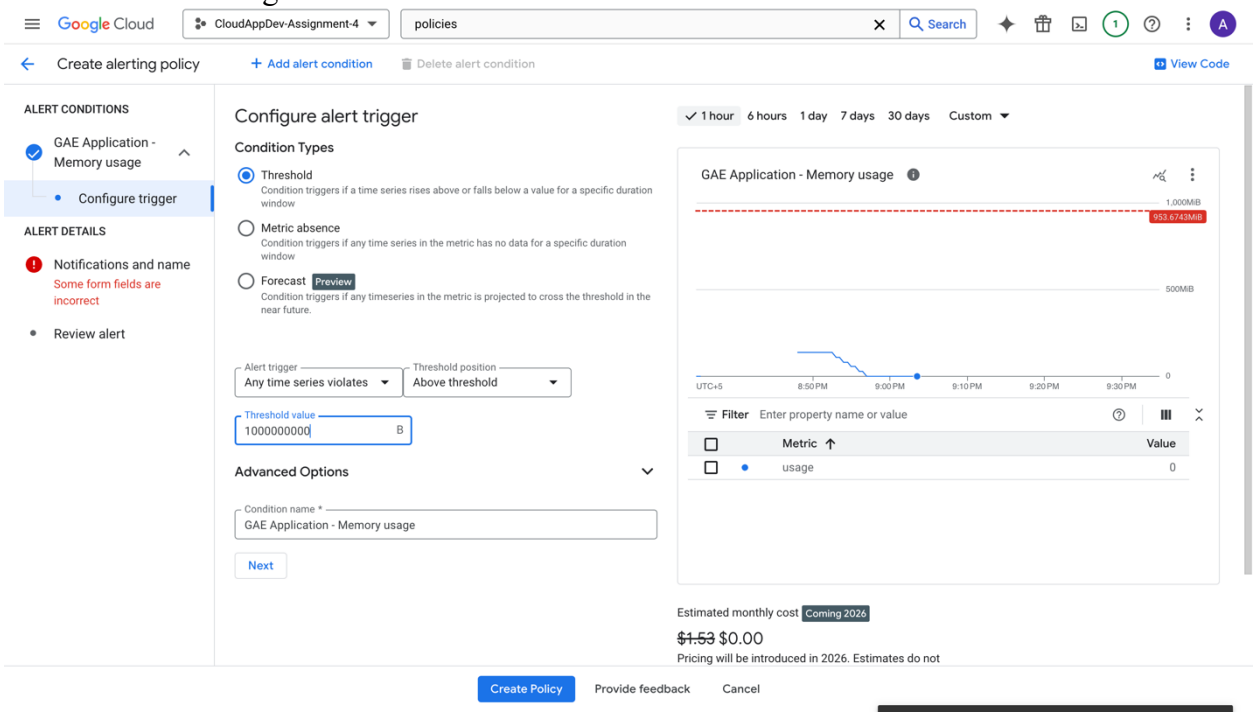


The next is to set up alerts using Google Cloud Monitoring based on specific security events. To do that search policies minotring and create alert. For my case I create an alert for memory

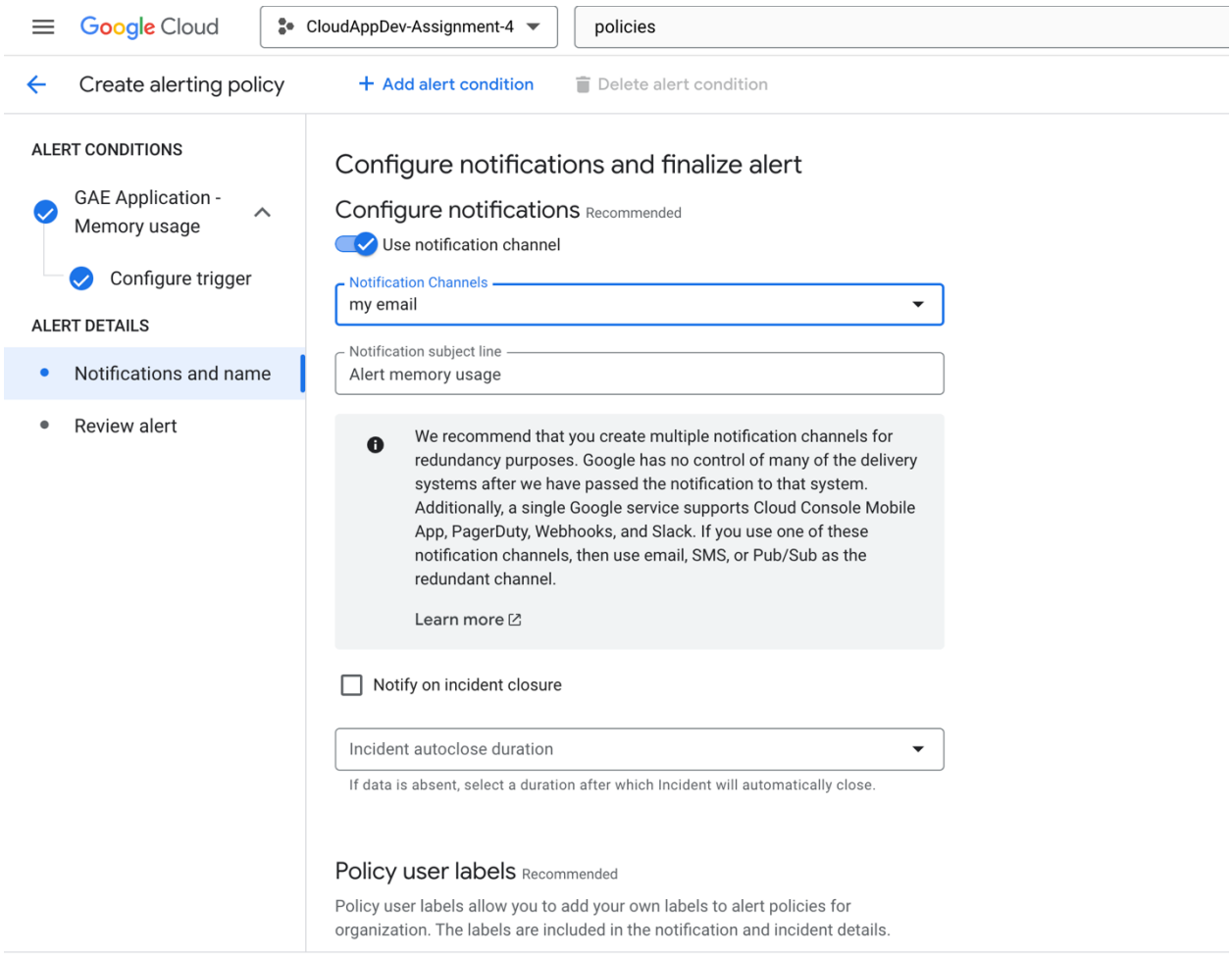
usage.



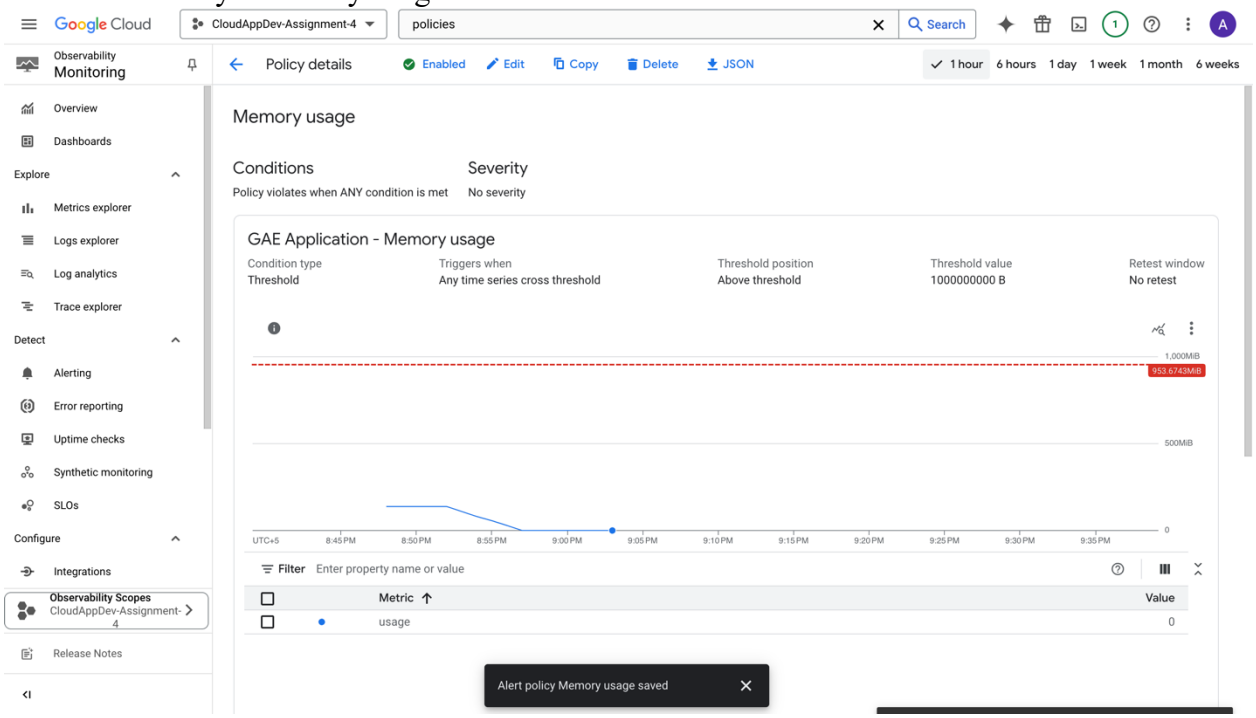
Set alert to 1GB usage.



As notification channel I chose my email.



As result here my “Memory usage” alert.



2.6. Incident Response

Incident response is the common practice for organiation to respond to critical failures of the system. It can be referred as part of Business Continuity Plan (BCP). The process of identifying, managing, and resolving security breaches is very important to manage system availabilty. The key components of a response plan include:

1. **Preparation:** Document response procedures and ensure all team members are trained.
2. **Detection and Analysis:** Use tools like Google Cloud Monitoring and Audit Logs to identify unusual activities.
3. **Containment:** Isolate the affected systems to prevent the spread of the breach.
4. **Eradication:** Identify and remove the cause of the incident, such as malicious code or compromised accounts.
5. **Recovery:** Restore normal operations by patching vulnerabilities and verifying system integrity.
6. **Post-Incident Review:** Conduct a review to analyze the cause of the incident, evaluate the response, and implement improvements.

To demonstrate incident response, simulate a security incident by triggering a fake vulnerability. After that make all steps described above as training of incedent response.

3. Scaling Applications on Google Cloud

3.1. Overview of Scalability

Scalability is the ability of a system to handle more work or support more users when needed. It is important because it helps applications run smoothly even when there are more requests or higher demand.

There are two main types of scalability:

1. Horizontal Scalability means adding more machines or instances to share the workload.It is good for applications that do not keep data on the instance (stateless), so any instance can handle any request. Benefits: Better performance, higher availability, and less chance of failure.
2. Vertical Scalability means increasing the power of one machine, like adding more CPU or memory. It works well for applications that keep important data in the instance (stateful). Benefits: Simpler to set up, but there is a limit to how much you can upgrade one machine.

Both methods can be used together depending on the needs of the application. In cloud systems like Google Cloud, tools like auto-scaling can make scaling automatic, saving both time and effort.

3.2. Application Design

For this purpose it is used simple phonebook application written in Flask framework. With CRUD operations:

```python
# Create a new contact
@app.route('/contacts', methods=['POST'])
def create_contact():
    global next_contact_id
    contact_data = request.json

    # Proceed to create the contact if validation succeeds
    contact = {
        'id': next_contact_id,
        'name': contact_data['name'],
        'phone': contact_data['phone']
    }
    phonebook[next_contact_id] = contact
    next_contact_id += 1  # Increment the ID for the next contact


    return jsonify(contact), 201

# Read all contacts
@app.route('/contacts', methods=['GET'])
def get_contacts():
    return jsonify(list(phonebook.values())), 200

# Read a specific contact by ID
@app.route('/contacts/<int:contact_id>', methods=['GET'])
def get_contact(contact_id):
    contact = phonebook.get(contact_id)
    if not contact:
        return jsonify({'message': 'Contact not found'}), 404
    return jsonify(contact), 200

# Update a contact
@app.route('/contacts/<int:contact_id>', methods=['PUT'])
def update_contact(contact_id):
    contact = phonebook.get(contact_id)
    if not contact:
        return jsonify({'message': 'Contact not found'}), 404
    updated_data = request.json
    contact['name'] = updated_data.get('name', contact['name'])
    contact['phone'] = updated_data.get('phone', contact['phone'])

    return jsonify(contact), 200

# Delete a contact
@app.route('/contacts/<int:contact_id>', methods=['DELETE'])
def delete_contact(contact_id):
    contact = phonebook.pop(contact_id, None)
    if not contact:
        return jsonify({'message': 'Contact not found'}), 404

    return jsonify({'message': 'Contact deleted'}), 200
```

Also it is used Google Kubernetes Engine that supports containerizing and autoscaling. But for we need to create image of application and enable GKE. So to do that, I create

Dockerfile:

Assignment-4 > Dockerfile

```dockerfile
# Use the official Python image as a base
FROM python:3.9-slim

# Set the working directory inside the container
WORKDIR /app

# Copy the requirements file into the container
COPY requirements.txt .

# Install the required packages
RUN pip install --no-cache-dir -r requirements.txt

# Copy the rest of the application code into the container
COPY main.py .

# Expose the port the app runs on
EXPOSE 8080

# Command to run the application using gunicorn
CMD ["gunicorn", "--bind", "0.0.0.0:8080", "main:app"]

```

Build the image: docker build -t phonebook-app .

PROBLEMS DEBUG CONSOLE OUTPUT TERMINAL PORTS zsh - As

```
Compressing objects: 100% (3/3), done.
Writing objects: 100% (5/5), 421 bytes | 421.00 KiB/s, done.
Total 5 (delta 2), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/altair2503/Cloud-Application-Development
   25e4a14..ae44a5f  main -> main
(base) kabdrakhman@MacBook-Pro-Altair Cloud-Application-Development % cd Assignment-4
(base) kabdrakhman@MacBook-Pro-Altair Assignment-4 % docker build -t phonebook-app .
[+] Building 20.2s (10/10) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 548B
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load metadata for docker.io/library/python:3.9-slim
 => [1/5] FROM docker.io/library/python:3.9-slim@sha256:6250eb7983c08b3cf5a7db9309f8630d3ca03dd152158fa37a3f8daaf397
 => => resolve docker.io/library/python:3.9-slim@sha256:6250eb7983c08b3cf5a7db9309f8630d3ca03dd152158fa37a3f8daaf397
 => => sha256:193e8340ba1de799bd9476ee14a9c4e24d0f51ccfe4c57c82bb7e6f7f807526e 14.83MB / 14.83MB
 => => sha256:6250eb7983c08b3cf5a7db9309f8630d3ca03dd152158fa37a3f8daaf397085d 10.41kB / 10.41kB
 => => sha256:da365f61478902de7be5ff08369c8c38be5a9cc924b1c6504bdbcd497d9e60c3 1.75kB / 1.75kB
 => => sha256:d5a37ad066668ba1afd69691600454b12a51f1c92f19858dfda2b00c48d579c2 5.43kB / 5.43kB
 => => sha256:6d29a096dd42e5e003949f934fa6b1a3ec8e076dd8cfc2a85a4e750a3639bf7a 29.16MB / 29.16MB
 => => sha256:6fab32a80202b33cfda04522829fe2e08e9c3a1a4a548a71a09d35720709a5ba 3.33MB / 3.33MB
 => => sha256:7e9a72ce45a2f47cd2b49aa49edba5d1d4c3b2fe18309912991d48b62ebccaa8 249B / 249B
 => => extracting sha256:6d29a096dd42e5e003949f934fa6b1a3ec8e076dd8cfc2a85a4e750a3639bf7a
 => => extracting sha256:6fab32a80202b33cfda04522829fe2e08e9c3a1a4a548a71a09d35720709a5ba
 => => extracting sha256:193e8340ba1de799bd9476ee14a9c4e24d0f51ccfe4c57c82bb7e6f7f807526e
 => => extracting sha256:7e9a72ce45a2f47cd2b49aa49edba5d1d4c3b2fe18309912991d48b62ebccaa8
 => [internal] load build context
 => => transferring context: 2.15kB
 => [2/5] WORKDIR /app
 => [3/5] COPY requirements.txt .
 => [4/5] RUN pip install --no-cache-dir -r requirements.txt
 => [5/5] COPY main.py .
 => exporting to image
 => => exporting layers
 => => writing image sha256:224a2aa191967be15ded94ca38f248f45393d5ce57542ec8e37202838f4584c2
 => => naming to docker.io/library/phonebook-app

What's Next?
  View a summary of image vulnerabilities and recommendations → docker scout quickview
(base) kabdrakhman@MacBook-Pro-Altair Assignment-4 %
```

Ln 7, Col 44 Spaces: 4

Prior pushing image to GCR, I tested locally with this command: docker run -p 8080:8080 phonebook-app

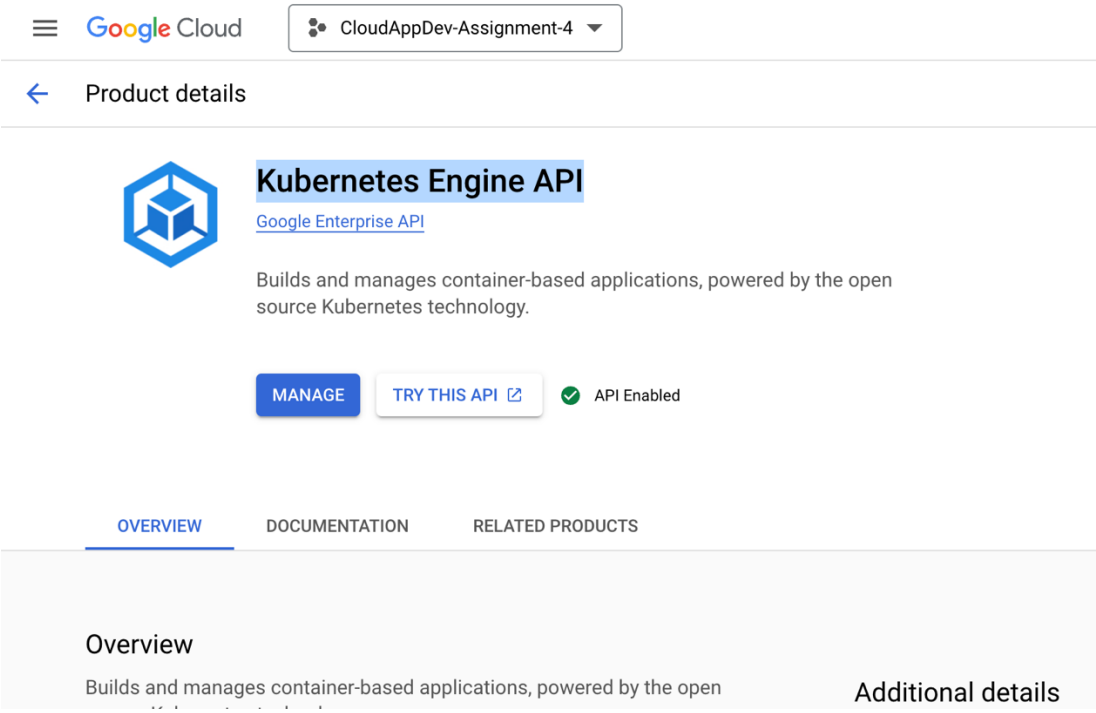
```
(base) kabdrakhman@MacBook-Pro-Altair Assignment-4 % docker run -p 8080:8080 phonebook-app
[2024-11-24 16:50:55 +0000] [1] [INFO] Starting gunicorn 20.1.0
[2024-11-24 16:50:55 +0000] [1] [INFO] Listening at: http://0.0.0.0:8080 (1)
[2024-11-24 16:50:55 +0000] [1] [INFO] Using worker: sync
[2024-11-24 16:50:55 +0000] [7] [INFO] Booting worker with pid: 7
```

To have permission to add images into your remote docker registry on google cloud you need to execute this command: gcloud auth configure-docker

To push my docker images I used this command to tag phonebook with my google cloud registry (GCR): docker tag phonebook-app gcr.io/cloudappdev-assignment-4/phonebook-app:latest
Only then I could push image to GCR docker push gcr.io/ cloudappdev-assignment-4/phonebook-app:latest

```
(base) kabdrakhman@MacBook-Pro-Altair Assignment-4 % docker tag phonebook-app gcr.io/cloudappdev-assignment-4/phonebook-app:latest
(base) kabdrakhman@MacBook-Pro-Altair Assignment-4 % docker push gcr.io/cloudappdev-assignment-4/phonebook-app:latest
The push refers to repository [gcr.io/cloudappdev-assignment-4/phonebook-app]
94b63cbd4acd: Pushed
6f4c54009c8c: Pushed
33b038fbdcc5: Pushed
2d032a168907: Pushed
00490679d83e: Layer already exists
8dc6549c207d: Layer already exists
678d187bf5a8: Layer already exists
077584c0c75a: Layer already exists
latest: digest: sha256:54ad10b9f070e309d4cc5403cea9a0903cb38808232883cace3b8d53761f0a67 size: 1990
(base) kabdrakhman@MacBook-Pro-Altair Assignment-4 %
```

To create Kubernetes enable api - Kubernetes Engine API



and to create cluster used code:

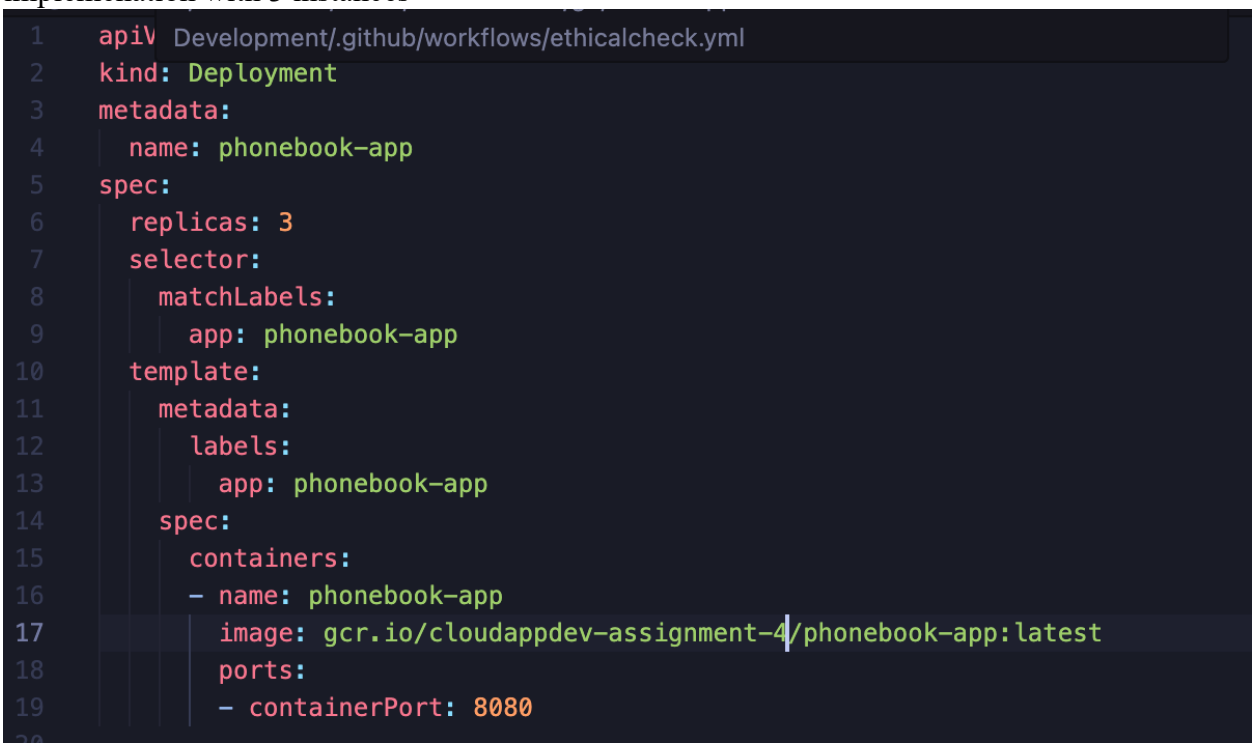
```
gcloud container clusters create assignment4-cluster \
--zone us-central1-a \
--num-nodes 1
```

To enable kubectl for my cluster used command:
gcloud container clusters get-credentials assignment4-cluster --zone us-central1-a

3.3. Scaling Methods

We have two types of scaling methods is to use horizontal or vertical scaling. The difference is that:

- Horizontal scaling is used when we increase number of instance. It is preferred to use when the application is stateless and requires high availability or load distribution across multiple instances. That means it doesn't share its states with other apps. The implemenation with 3 instances



- Vertical scalling is used when we increase capacity of our machine. It is preferred to use when the application is stateful or requires more resources than a single instance can handle. That means when instance saves it state. The implemenation with high number of memory and CPUs.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: phonebook-app
spec:
  replicas: 1
  selector:
    matchLabels:
      app: phonebook-app
  template:
    metadata:
      labels:
        app: phonebook-app
    spec:
      containers:
      - name: phonebook-app
        image: gcr.io/cloudappdev-assignment-4/phonebook-app:latest
        ports:
        - containerPort: 8080
        resources:
          requests:
            cpu: "500m"
            memory: "256Mi"
---
```

3.4. Load Balancing

To enable load balancing it was created service instance attached to my pod with load balancing method. Here is my service:

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

Also it is important to load traffic to working pods. For that it is necessary to have liveness and readiness implementation. To app was added:

```python
next_contact_id = 1  # Start with ID 1

# Health Check Endpoints

# Liveness check endpoint
@app.route('/health', methods=['GET'])
def healthz():
    # Just returns a 200 OK if the app is alive
    return jsonify({"status": "ok"}), 200

# Readiness check endpoint
@app.route('/readiness', methods=['GET'])
def readiness():
    # Check if phonebook is empty, which might be a simple readiness check (or use other logic)
    if len(phonebook) >= 0:  # You could make this check more complex if needed
        return jsonify({"status": "ready"}), 200
    else:
        return jsonify({"status": "not ready"}), 503

```

And to main pod was added liveness and readiness fields:

```yaml
        image: gcr.io/cloudappdev-assignment-4/phonebook-app:latest
        ports:
        - containerPort: 8080
        resources:
          requests:
            cpu: "500m"  # Request 500m CPU for the container
            memory: "256Mi"  # Request 256Mi memory for the container
        # Health checks (probes)
        livenessProbe:
          httpGet:
            path: /health  # Adjust the path based on your app's health endpoint
            port: 8080
          initialDelaySeconds: 10  # Time to wait before the first probe
          periodSeconds: 5  # Time between subsequent probes
          failureThreshold: 3  # Number of consecutive failures before marking the pod as unhealthy

        readinessProbe:
          httpGet:
            path: /readiness  # Adjust the path based on your app's readiness endpoint
            port: 8080
          initialDelaySeconds: 5  # Time to wait before starting readiness checks
          periodSeconds: 5  # Time between subsequent probes
          failureThreshold: 3  # Number of consecutive failures before stopping traffic to the pod

```

3.5. Auto-Scaling Implementation

To enable horizonatl auto scalling it was defined HorizontalPodAutoscaler that will increase the number of instances if cpu usage exceeds more than 50 percents.

```yaml
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: phonebook-app-hpa
  namespace: default
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: phonebook-app
  minReplicas: 1  # Minimum number of replicas
  maxReplicas: 10  # Maximum number of replicas
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 50  # Scale when average CPU usage exceeds 50%
```

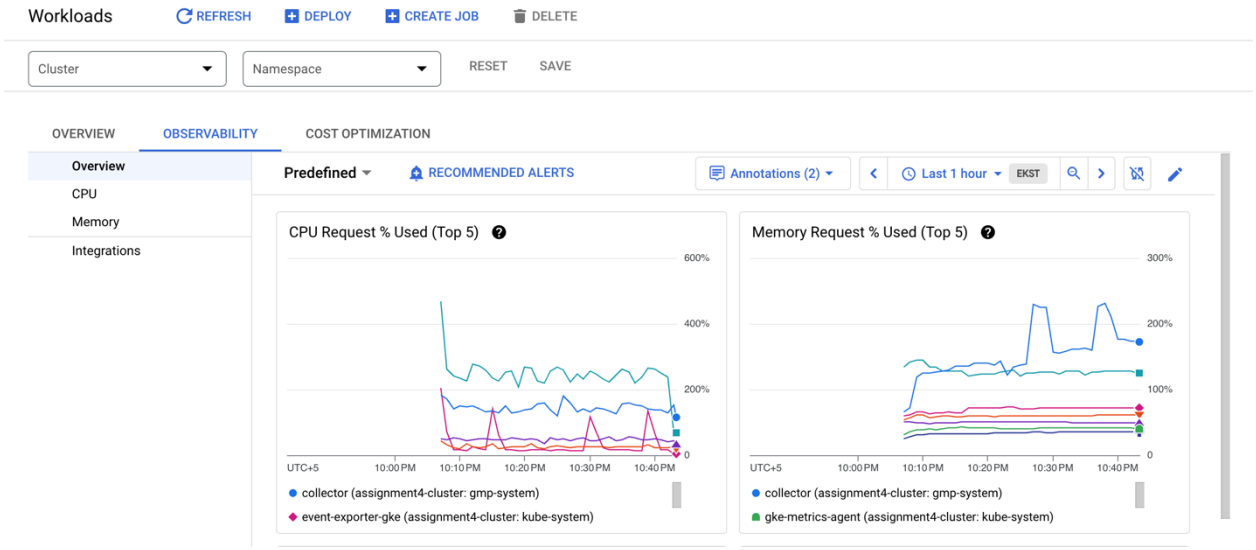
For vertical autoscaling it was added limits field to enable scaling memory and CPUs for my instance. It means it will use more than requested cpu and memory

Assignment-4 > phonebook.yaml

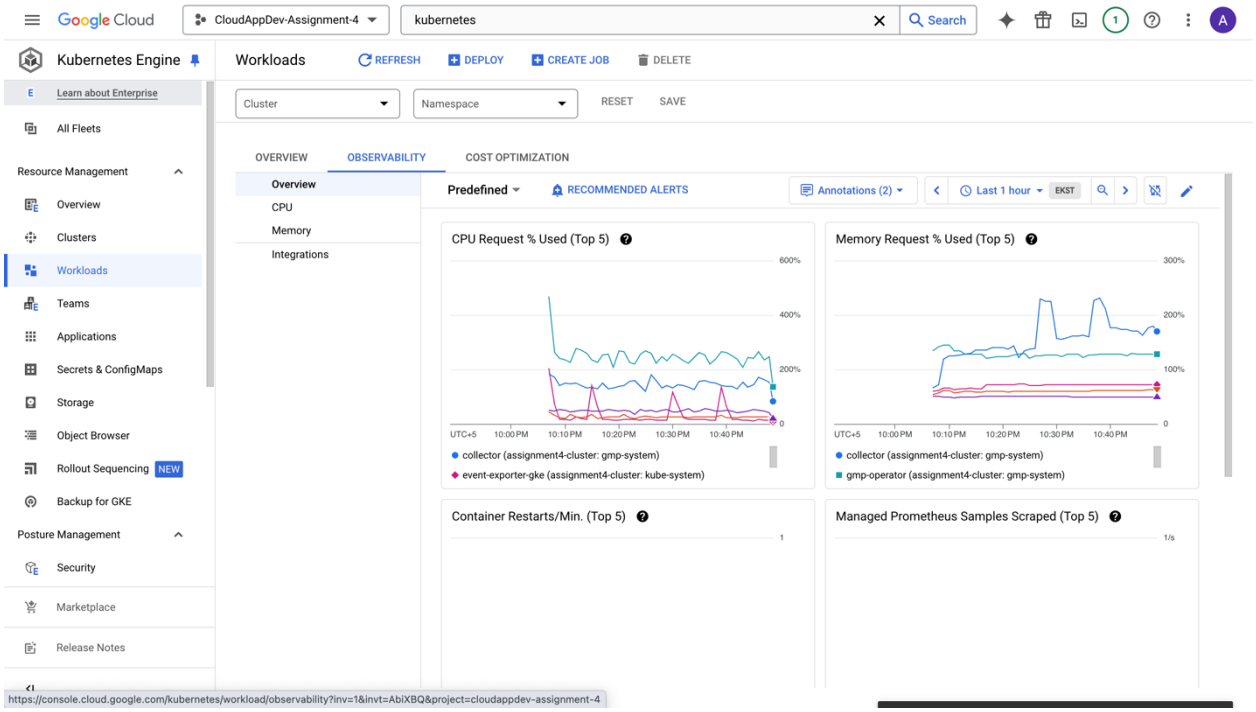
```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: phonebook-app
spec:
  replicas: 1
  selector:
    matchLabels:
      app: phonebook-app
  template:
    metadata:
      labels:
        app: phonebook-app
    spec:
      containers:
      - name: phonebook-app
        image: gcr.io/cloudappdev-assignment-4/phonebook-app:latest
        ports:
        - containerPort: 8080
        resources:
          requests:
            cpu: "500m"  # Request 500m CPU for the container
            memory: "256Mi"  # Request 256Mi memory for the container
          limits:
            cpu: "1"  # Limit the container to 1 CPU
            memory: "512Mi"  # Limit the container to 512Mi memory
---
```

3.6. Performance Monitoring

GKE automatically enable Google Cloud Monirong tool for my cluster that monitor CPU and memory usage:



It can be found in workloads tab, observability option.



3.7. Cost Optimization Strategies

Cost optimization means using resources in a smart way to reduce expenses while keeping good performance. Here are some tips to save money on Google Cloud:

1. **Use the Right Resources:** Check how much CPU, memory, and storage your app uses and adjust instance size to avoid oversized resources.
2. **Auto-Scaling:** Set up auto-scaling to adjust resources based on demand and avoid wasting money during low-usage times.
3. **Storage Optimization:** Choose the right storage class—Standard for frequent use, Coldline for rare use, and Archive for long-term—and automate moving old data to cheaper storage.
4. **Serverless and Managed Services:** Use serverless options like Cloud Functions to pay only for usage and managed services like BigQuery for automatic scaling.
5. **Monitor Usage:** Use Google Cloud Monitoring to track costs and resource usage, and set alerts for unexpected cost increases.
6. **Optimize Networking:** Keep resources in the same region to reduce data transfer costs and use Cloud CDN to cache content closer to users.
7. **Remove Unused Resources:** Identify and delete unused instances, disks, or IP addresses, and turn off non-essential resources during off-hours.
8. **Review and Automate:** Regularly review your cloud costs and usage, and use tools like Google Cloud Recommender to automate savings.

We can reduce costs while keeping our application scalable and reliable. It helps make cloud resources more efficient and avoids spending money on unnecessary items.

4. Conclusion

In this report, we have discussed the best practices for improving security and scalability of applications on Google Cloud. We learned that using tools like IAM for access control, Google Cloud KMS for data encryption, and HTTPS for secure data transfer are important for

keeping applications safe. Security testing and continuous monitoring also help identify and fix issues quickly.

For scalability, we learnt horizontal and vertical scaling methods to manage user traffic and ensure the app works smoothly. Using load balancing and auto-scaling, we can make the app performs well even during high traffic. Cost optimization strategies also help reduce unnecessary spending while maintaining performance.

5. Recommendations

After completing the assignment I come up to the following recommendations:

1. **IAM Roles**: By using the principle of least privilege it makes only the necessary permissions are granted to users and service accounts. Reducing the risk of vulnearability
2. **Data Protection**: Data encryption is important for maintaining confidentiality and integrity. Always encrypt sensitive data at rest using tools like Google Cloud Key Management Service (KMS).
3. **Secure Network Protocols**: Implement SSL certificates and configure the load balancer make possible that all data intransition between clients and the server is securely encrypted.
4. **Load Balancers**: Load balancing is essential for distributing network traffic efficiently across application, makes the application's availability and reliability.
5. **Auto-Scaling**: Both horizontal and vertical scaling strategies should be used depending on the application's architecture ensuring zero-down time for our app.
6. **Monitoring and Alerting**: Use Google Cloud's monitoring and logging tools to continuously track application performance and security events.
7. **Security and Performance Reviews**: Using tools like OWASP ZAP for security testing and Google Cloud Monitoring for tracking resource usage makes avoiding of issues in production

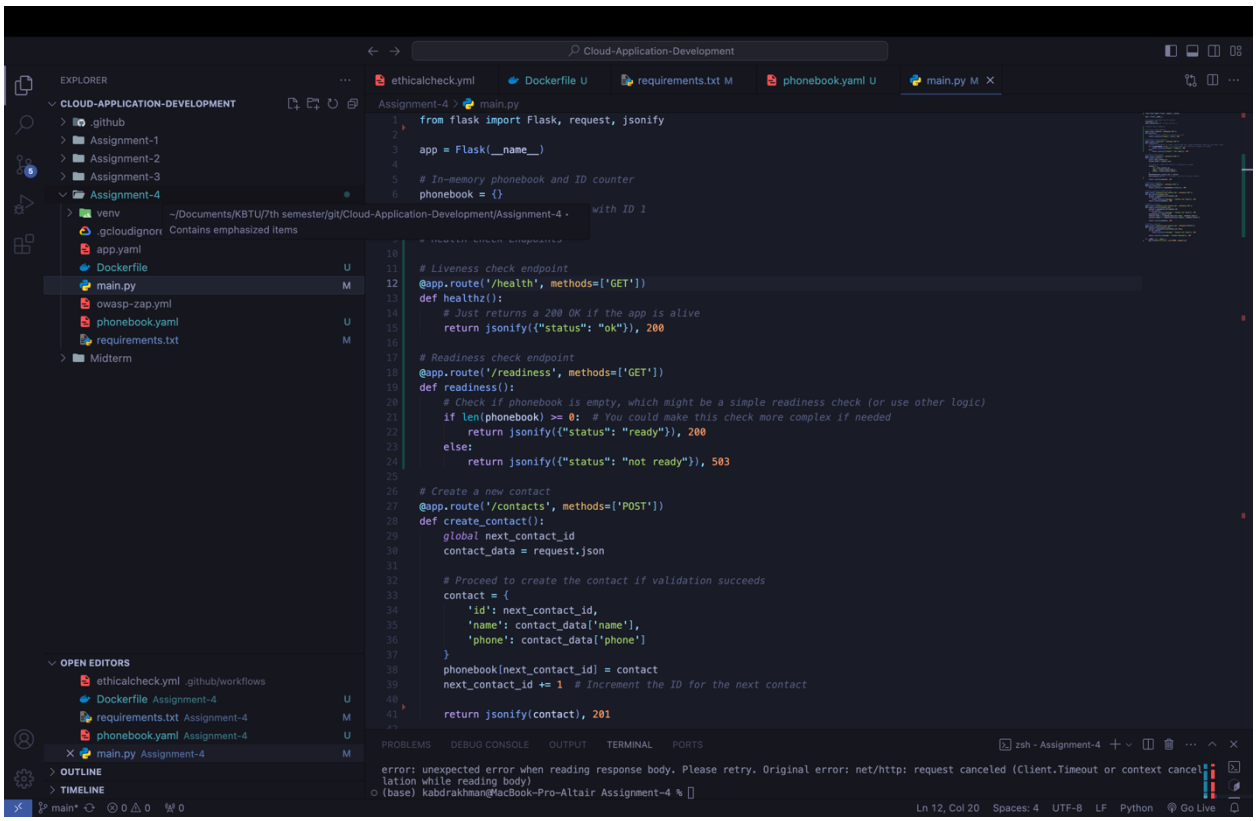
With these recommendations, the application will have enhanced security, scalability, and cost optimization, ensuring it is resilient to both current and future demands on Google Cloud.

6. References

- IAM documenation – https://cloud.google.com/iam/docs/
- Google KMS - https://cloud.google.com/kms/docs
- Load Balancer - https://cloud.google.com/load-balancing/docs
- Kubernetes - https://cloud.google.com/kubernetes-engine/docs
- Horizontal Autoscalling - https://cloud.google.com/kubernetes-engine/docs/how-to/scaling-apps
- Alerting - https://cloud.google.com/retail/docs/monitor
- OWASP ZAP - https://github.com/zaproxy/zaproxy

7. Appendices

Project structure


Cloud-Application-Development
EXPLORER
CLOUD-APPLICATION-DEVELOPMENT
.github
Assignment-1
Assignment-2
Assignment-3
Assignment-4
venv
~/Documents/KBTU/7th semester/git/Cloud-Application-Development/Assignment-4 •
Contains emphasized items
app.yaml
Dockerfile
main.py
owasp-zap.yml
phonebook.yaml
requirements.txt
Midterm
OPEN EDITORS
ethicalcheck.yml .github/workflows
Dockerfile Assignment-4
requirements.txt Assignment-4
phonebook.yaml Assignment-4
main.py Assignment-4
OUTLINE
TIMELINE
ethicalcheck.yml
Dockerfile U
requirements.txt M
phonebook.yaml U
main.py M
Assignment-4 > main.py
from flask import Flask, request, jsonify
app = Flask(__name__)
# In-memory phonebook and ID counter
phonebook = {}
with ID 1
Liveness check endpoint
@app.route('/health', methods=['GET'])
def healthz():
# Just returns a 200 OK if the app is alive
return jsonify({"status": "ok"}), 200
Readiness check endpoint
@app.route('/readiness', methods=['GET'])
def readiness():
# Check if phonebook is empty, which might be a simple readiness check (or use other logic)
if len(phonebook) >= 0: # You could make this check more complex if needed
return jsonify({"status": "ready"}), 200
else:
return jsonify({"status": "not ready"}), 503
Create a new contact
@app.route('/contacts', methods=['POST'])
def create_contact():
global next_contact_id
contact_data = request.json
# Proceed to create the contact if validation succeeds
contact = {
'id': next_contact_id,
'name': contact_data['name'],
'phone': contact_data['phone']
}
phonebook[next_contact_id] = contact
next_contact_id += 1 # Increment the ID for the next contact
return jsonify(contact), 201
PROBLEMS
DEBUG CONSOLE
OUTPUT
TERMINAL
PORTS
zsh - Assignment-4
error: unexpected error when reading response body. Please retry. Original error: net/http: request canceled (Client.Timeout or context cancel
lation while reading body)
(base) kabdrakhman@MacBook-Pro-Altair Assignment-4 %
main*
Ln 12, Col 20
Spaces: 4
UTF-8
LF
Python
Go Live