Application Security Best Practices and Scaling Applications on Google Cloud

Date: 22.11.2024

Made by:
Alimov Yedige

Content

Introduction

Overview of Google Cloud

Google Cloud Platform (GCP) is a suite of cloud computing services provided by Google. It offers a range of tools and services for computing, storage, data analytics, machine learning, networking, and more. GCP enables organizations to build, deploy, and scale applications, websites, and services on the same infrastructure that powers Google's own products, such as Google Search, Gmail, and YouTube.

Key Components of Google Cloud:

1. **Compute Engine**: Virtual machines (VMs) running on Google's infrastructure.
2. **App Engine**: A platform-as-a-service (PaaS) for building scalable web applications and mobile backends.
3. **Kubernetes Engine**: Managed Kubernetes service for containerized applications.
4. **Cloud Functions**: Event-driven serverless compute platform.
5. **Cloud Storage**: Object storage service for storing and retrieving large amounts of unstructured data.
6. **BigQuery**: A fully-managed, serverless data warehouse for large-scale data analytics.
7. **Cloud SQL**: Managed relational database service for MySQL, PostgreSQL, and SQL Server.
8. **Cloud Pub/Sub**: Messaging service for building event-driven systems and real-time analytics.

Significance of Security in Cloud Applications

Security is a critical aspect of any cloud application. With the increasing reliance on cloud services, ensuring the confidentiality, integrity, and availability of data and applications is paramount. Security in the cloud encompasses various practices and technologies to protect cloud-based systems, data, and infrastructure.

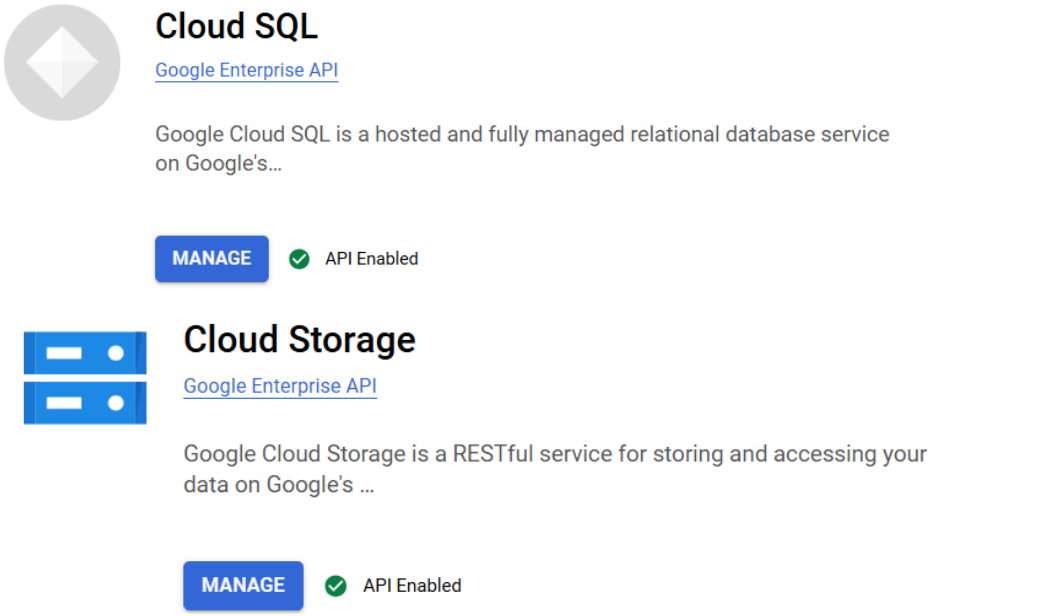
Google Cloud provides a robust set of tools and services that empower organizations to build, deploy, and scale applications securely and efficiently. Security ensures the protection of data and applications, while scalability ensures that applications can handle varying loads and growth. Together, these elements are critical for the success and reliability of cloud-based applications.

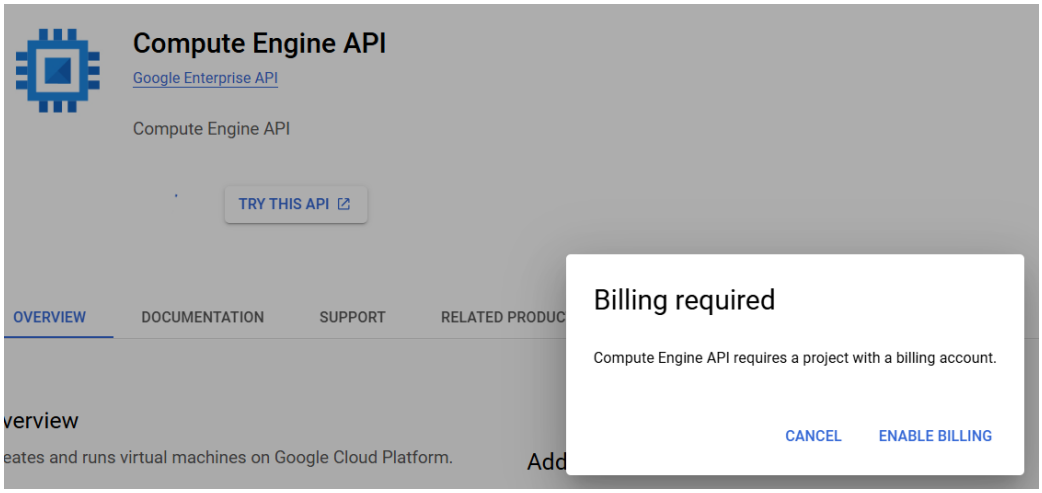
Exercise 1

Application Security Best Practices

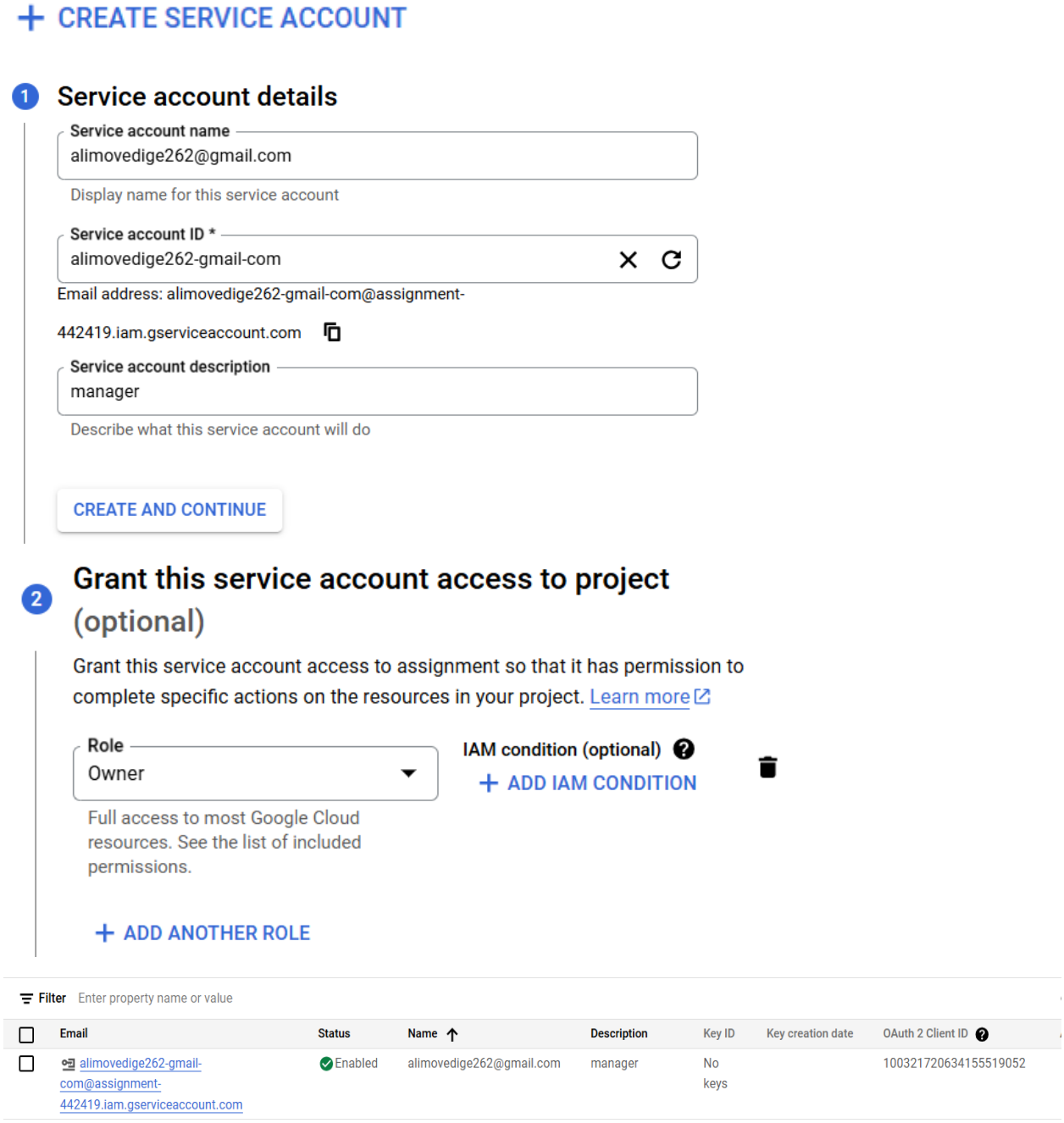
- **Set Up a Google Cloud Project:**
    - Create a new Google Cloud project.

- Enable necessary APIs (e.g., Cloud Storage, Cloud SQL, Compute Engine).

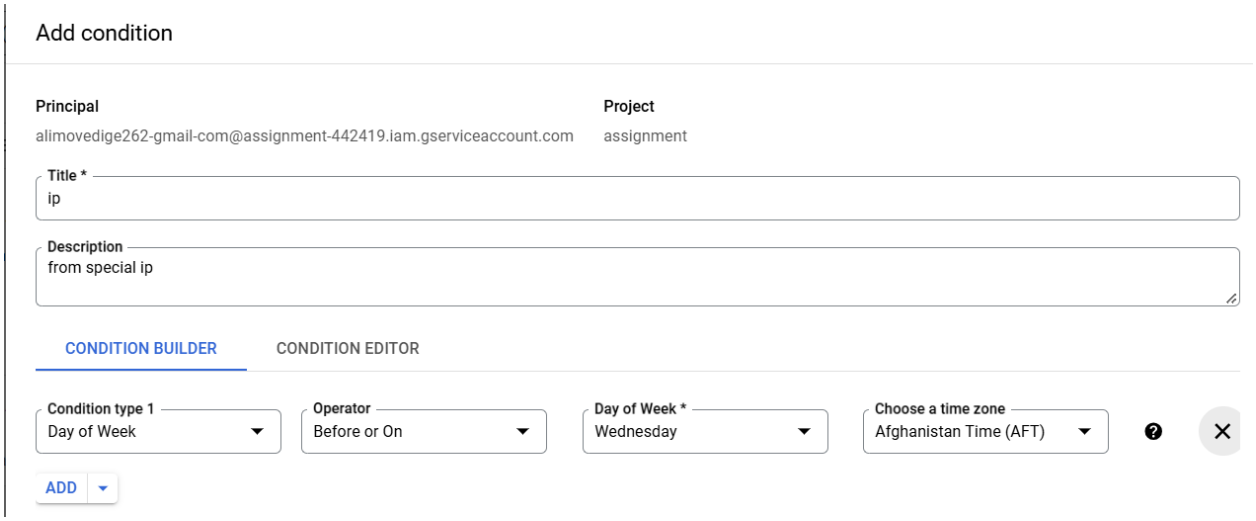




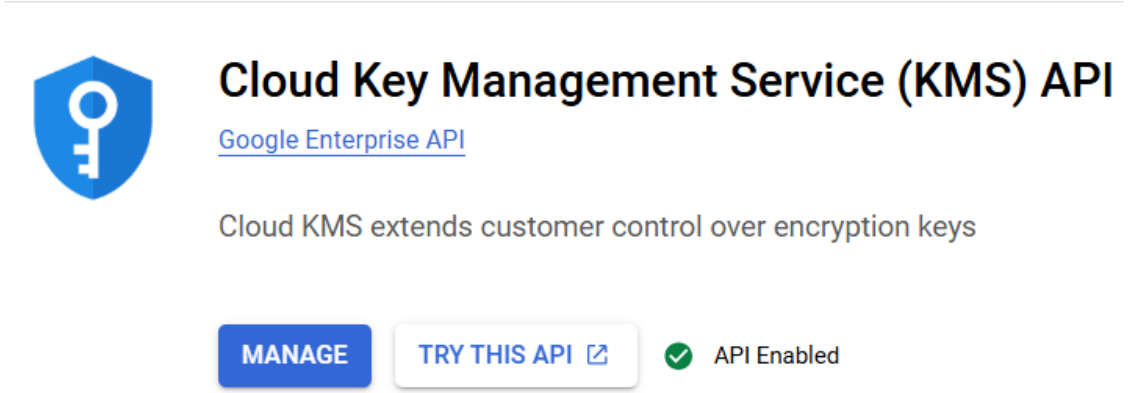
1. **Identity and Access Management (IAM):**
   - Create a service account for your application and assign the principle of least privilege.



- Implement IAM conditions to restrict access based on attributes



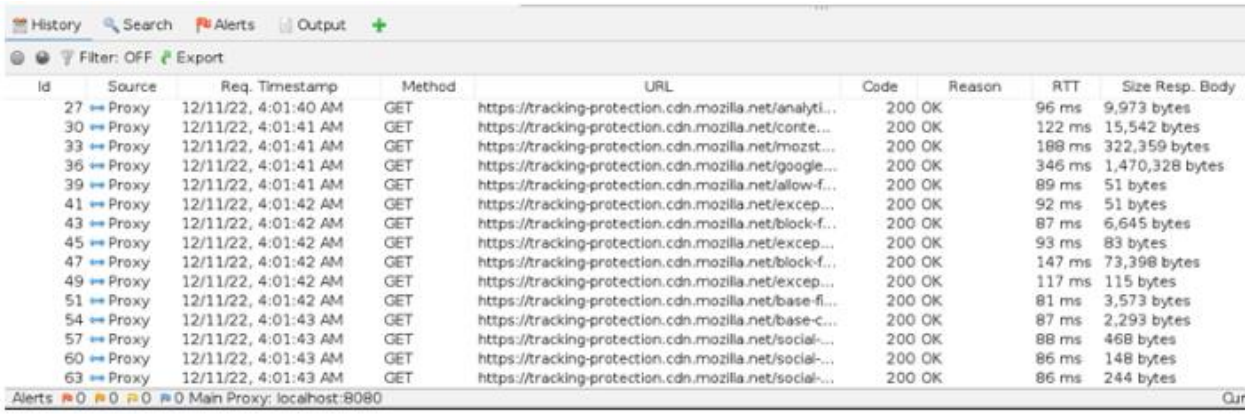
2. **Data Protection:**
    - Set up encryption for data at rest using Google Cloud KMS.

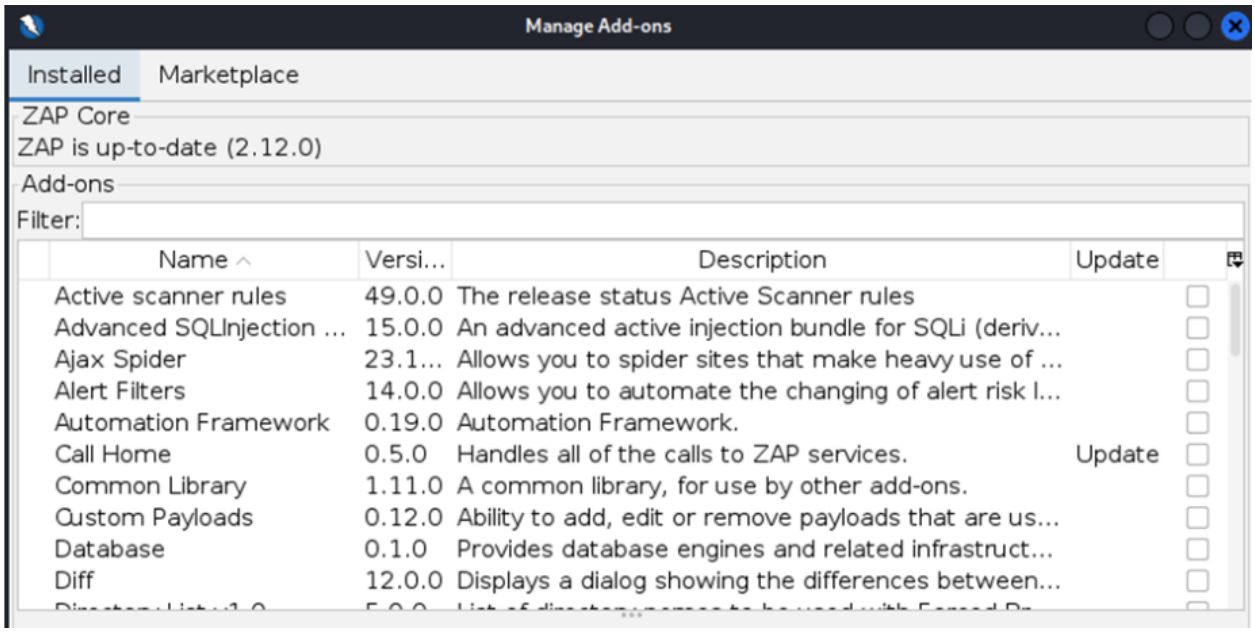


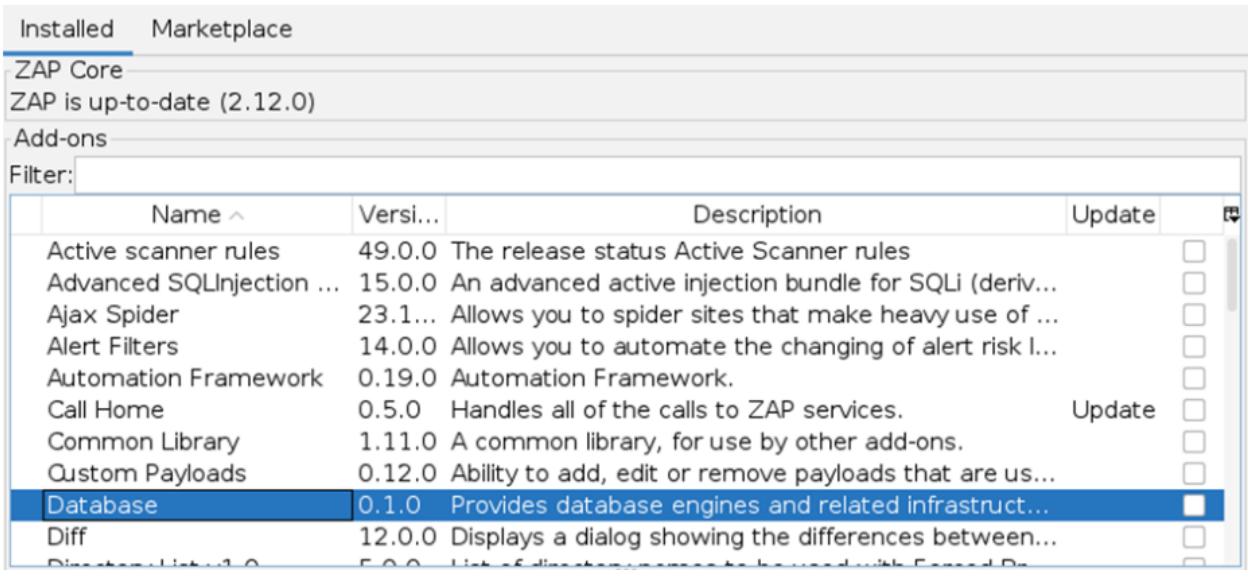
3. **Application Security Testing:**
    - Integrate a security scanning tool (e.g., Snyk, OWASP ZAP) into your CI/CD pipeline.

The Zed Attack Proxy (ZAP) by Checkmarx is the world's most widely used web app scanner. Free and open source. A community based GitHub Top 1000 project that anyone can contribute to. It can help you automatically find security vulnerabilities in your web applications while you are developing and testing your applications. It's also a great tool for experienced pentesters to use for manual security testing.

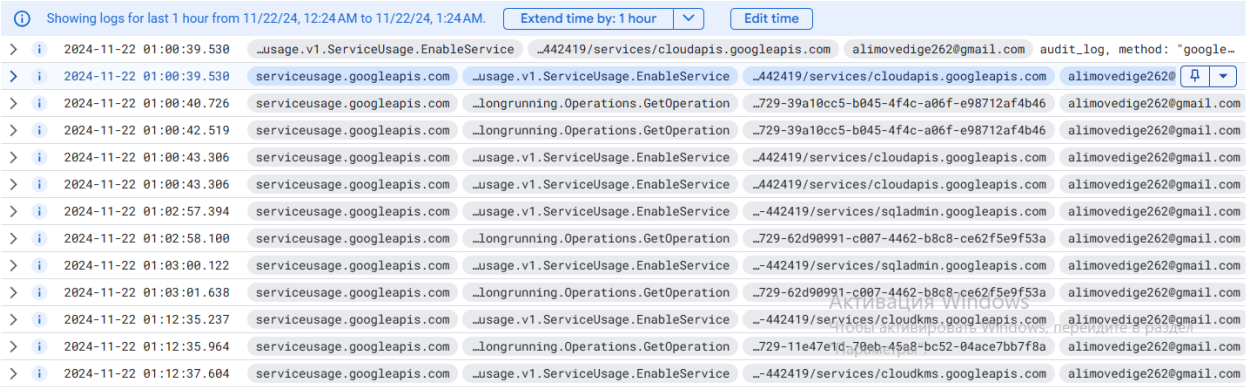
- Conduct a vulnerability assessment of your application and document findings.



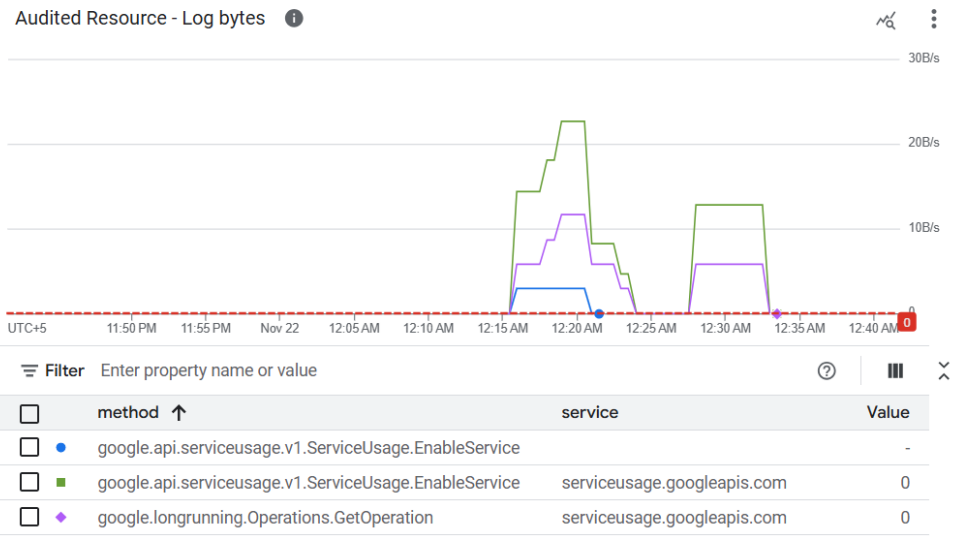


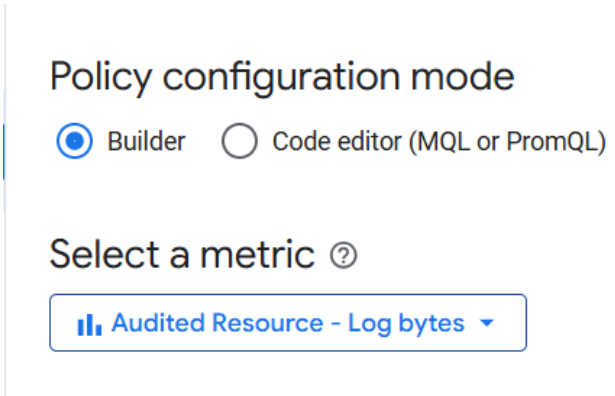


4. **Monitoring and Logging:**
   - Enable Google Cloud Audit Logs for your project.



   - Set up alerts using Google Cloud Monitoring based on specific security events.





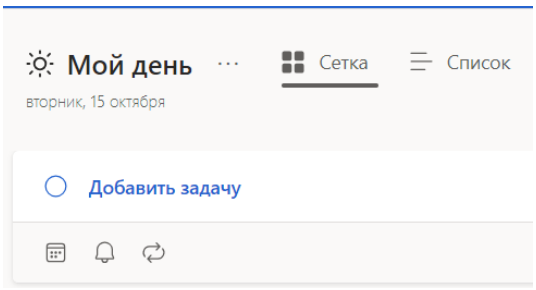
Exercise 2

Scaling Applications on Google Cloud

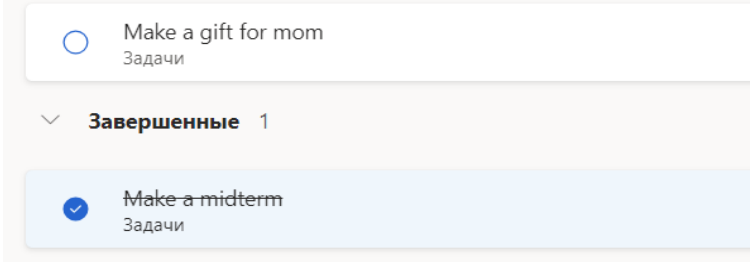
The To-Do List is one of the simpler, practical examples for deploying a web application on Google Cloud Platform. GCP provides access to an infrastructure that provides high availability and scalability. The project thus demonstrates how several GCP services can be used together in order to solve a more general task management problem.

Features:

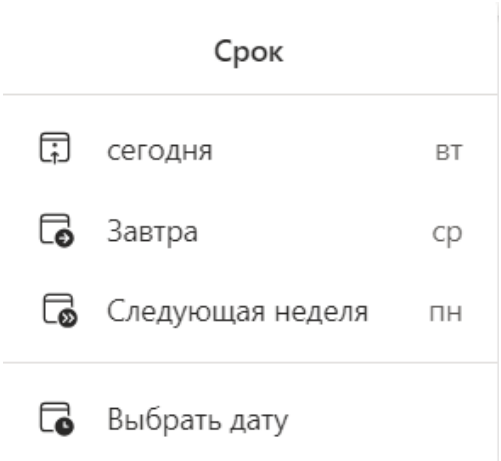
- Create and edit tasks.



- Setting the task status (completed/not completed).



- Notifications about the approaching deadline.



Google Cloud Platform Overview:

- App Engine is a serverless, fully managed platform for creating and deploying web apps on a large scale. You may create your apps using a variety of widely used languages,

libraries, and frameworks, and App Engine will handle server provisioning and demand-driven scalability of your app instances.

- Cloud Functions is a pay-as-you-go, scalable Functions-as-a-Service (FaaS) platform that allows you to run your code with no server management. The developer experience with Google Cloud Functions is straightforward and easy to use. Simply write your code and leave the operational infrastructure to Google Cloud.
- Cloud Endpoints employs a distributed Extensible Service Proxy, a service proxy that operates in a separate Docker container, as part of its distributed API management system. 00:23 Helping you develop and manage even the most demanding APIs with excellent performance and minimal latency is the aim.

Google Cloud SDK and Cloud Shell:

1) Choose and set new project called “midterm”

```
alimovedige262@cloudshell:~ (midterm-438617)$ gcloud config set project midterm
```

2) Create virtual environment and activate it
3) Install Flask

```
alimovedige262@cloudshell:~ (midterm)$ python3 -m venv virt
alimovedige262@cloudshell:~ (midterm)$ source virt/bin/activate
(virt) alimovedige262@cloudshell:~ (midterm)$ pip install Flask
```

Google App Engine:

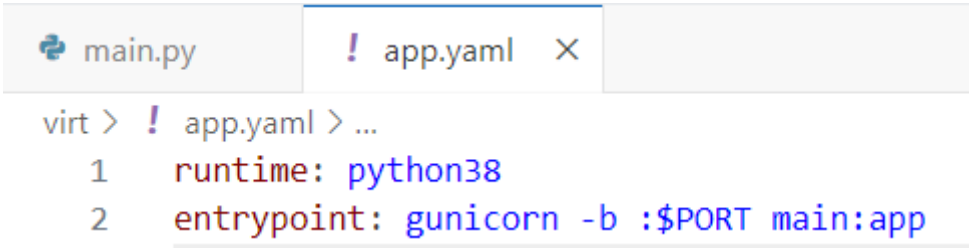
4) Create main.py code for to do list
   Line 5 is a simple in-memory list called “tasks” which is used to store tasks. A real-world application would most likely replace this with a database.
   Line 7 is decorator which defines a route for handling HTTP requests to the tasks endpoint. It accepts both GET and POST methods
   Line 7, 8, 9,10, 11, 12, 13 checks if the request method is POST, retrieves the task data from the JSON body of the request, adds the new task to the tasks list and returns a JSON response with the newly added task and an HTTP status code of 201.

main.py app.yaml

virt > main.py > ...

```python
from flask import Flask, request, jsonify

app = Flask(__name__)

tasks = []

@app.route('/tasks', methods=['GET', 'POST'])
def handle_tasks():
    if request.method == 'POST':
        task = request.json.get('task')
        tasks.append(task)
        return jsonify({'task': task}), 201
    return jsonify({'tasks': tasks})

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

5) Create app.yaml for deploying App Engine



Building with Google Cloud Functions:

6) Create function code for notification. This function will serve HTTP requests to extract a task from the request's JSON body and perform an action on it-specifically, notifying of the said task. The function will return a confirmation message that a notification has been sent out for the particular task.

Function

```
import functions_framewor

@functions_framework.http

def notify(request):

    request_json = request.get_json(silent=True)

    task = request_json.get('task')

    return f'Notification sent for task: {task}'
```

Containerizing Applications:

7) Create a dockerfile to containerize the application and then build the docker image using docker build -t app

```
alimovedige262@cloudshell:~ (myassignment-437807)$ mkdir mydocker
alimovedige262@cloudshell:~ (myassignment-437807)$ cd mydocker
alimovedige262@cloudshell:~/mydocker (myassignment-437807)$ touch app.py
alimovedige262@cloudshell:~/mydocker (myassignment-437807)$ docker build -t hello-world-app .
[+] Building 6.7s (8/8) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 465B
 => [internal] load metadata for docker.io/library/python:3.9-slim
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [1/3] FROM docker.io/library/python:3.9-slim@sha256:49f94609e5a997dc16086a66ac9664591854031d48e375945a9dbf4d1d53abbc
 => => resolve docker.io/library/python:3.9-slim@sha256:49f94609e5a997dc16086a66ac9664591854031d48e375945a9dbf4d1d53abbc
 => => sha256:fdeeec85abbad3878f2008f9445f15a19a5a224d1b7e7715ac6b923072333e57 14.74MB / 14.74MB
 => => sha256:49f94609e5a997dc16086a66ac9664591854031d48e375945a9dbf4d1d53abbc 10.41kB / 10.41kB
 => => sha256:93ab151da4e5310ea79c4ecf306ece628262b86a4d7a49cc601664f19fe44e36 1.75kB / 1.75kB
 => => sha256:9d8cb7037cd8e90893e5f430ce4c048a872511e414580c7641675f2dad0a0351 5.20kB / 5.20kB
 => => sha256:302e3ee498053a7b5332ac79e8efebec16e900289fc1ecd1c754ce8fa047fcab 29.13MB / 29.13MB
 => => sha256:4c0965d3919510b506d8856ebc050a96e996c7dae96e4fb420882dbe7e037e67 3.51MB / 3.51MB
 => => sha256:62a08b8dd4f53ad5493dabf2af00ccde91abb3771fb2187040bcf2fe94a7ced7 248B / 248B
 => => extracting sha256:302e3ee498053a7b5332ac79e8efebec16e900289fc1ecd1c754ce8fa047fcab
 => => extracting sha256:4c0965d3919510b506d8856ebc050a96e996c7dae96e4fb420882dbe7e037e67
 => => extracting sha256:fdeeec85abbad3878f2008f9445f15a19a5a224d1b7e7715ac6b923072333e57
 => => extracting sha256:62a08b8dd4f53ad5493dabf2af00ccde91abb3771fb2187040bcf2fe94a7ced7
 => [internal] load build context
 => => transferring context: 542B
 => [2/3] WORKDIR /app
 => [3/3] COPY . /app
 => exporting to image
 => => exporting layers
 => => writing image sha256:3f8c7ac9d00287b93487b74fd24467da6de5c195e08a6c26646d0e455c54c9b9
 => => naming to docker.io/library/hello-world-app
alimovedige262@cloudshell:~/mydocker (myassignment-437807)$ docker run --rm hello-world-app
Hello from inside the container!
alimovedige262@cloudshell:~/mydocker (myassignment-437807)$
```

8) Run the command to deploy on GKE:

deployment.yaml | service.yaml

docker > deployment.yaml > {} spec > {} template > {} spec

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: todo-app
spec:
  replicas: 3
  selector:
    matchLabels:
      app: todo-app
  template:
    metadata:
      labels:
        app: todo-app
    spec:
      containers:
      - name: todo-app
        image: gcr.io/midterm/todo-app
        ports:
        - containerPort: 8080
```

9) Creating a service to access the application

deployment.yaml | service.yaml

docker > service.yaml > {} spec > [ ] ports

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

Managing APIs with Google Cloud Endpoints:

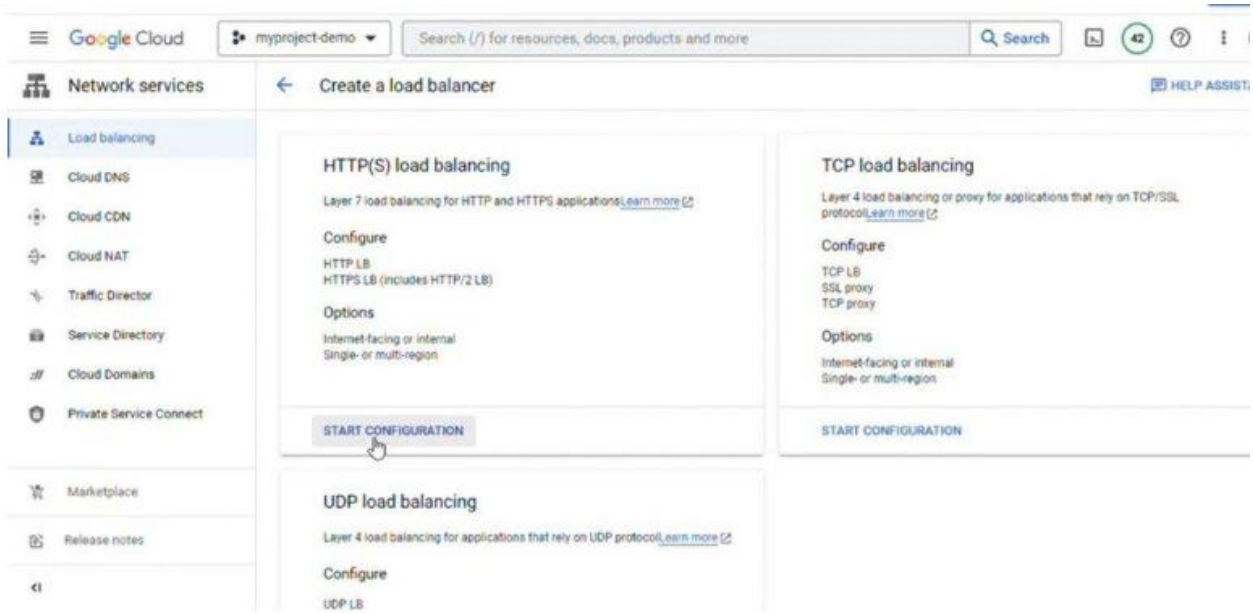
10) Configuring the API using Google Cloud Endpoints

deployment.yaml | api.yaml | service.yaml

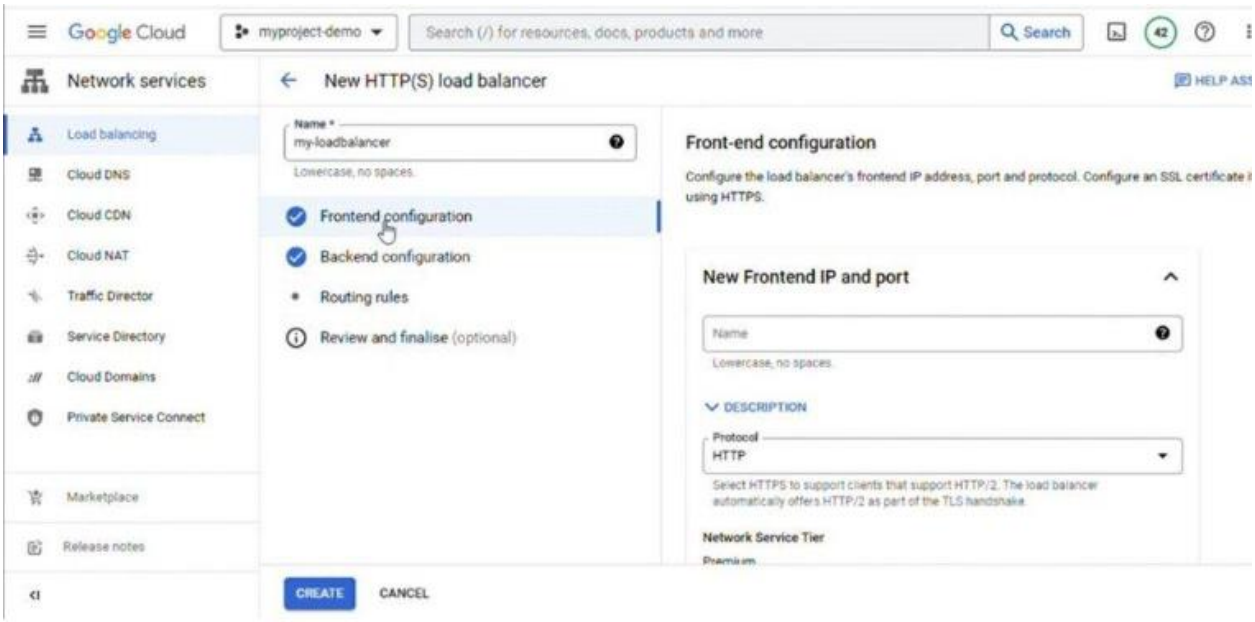
docker > api.yaml > ...

```yaml
swagger: "2.0"
info:
  title: "midterm"
  description: "API for todo app"
  version: "1.0.0"
host: "todolist.appspot.com"
schemes:
  - "https"
paths:
  /tasks:
    get:
      summary: "List tasks"
      responses:
        200:
          description: "A list of tasks"
```

1. **Load Balancing:**
    - Set up Google Cloud Load Balancing to distribute traffic across multiple instances.
    - Configure health checks to ensure traffic is routed only to healthy instances.



Choose the type of load balancer



Conclusion

- Summary of the importance of applying security best practices and effective scaling strategies on Google Cloud.

  Implementing security best practices and effective scaling strategies on Google Cloud is crucial for ensuring the protection, performance, and growth of applications and services. Security best practices are essential for protecting sensitive data, such as personal information, financial records, and intellectual property. By employing encryption both at rest and in transit, organizations can safeguard data from unauthorized access and breaches. This is particularly important in industries with stringent regulatory requirements, such as GDPR and HIPAA. Adhering to these regulations helps organizations avoid legal penalties and maintain customer trust. Effective scaling strategies are equally important for managing increased traffic and maintaining application performance. These strategies ensure that applications can handle peak times without compromising user experience. By optimizing resource utilization, organizations

can adjust the number and size of instances based on current demand, preventing over-provisioning and under-utilization, which leads to significant cost savings.

Applying security best practices and effective scaling strategies on Google Cloud is vital for protecting data, ensuring compliance, preventing breaches, optimizing resources, enhancing performance, and supporting business growth. These practices not only safeguard the organization's assets but also enable sustainable and cost-effective operations in the cloud environment.

Recommendations

1. Practical suggestions for further improving security and scalability based on the report's findings.

Improving security involves conducting regular security audits and compliance checks to ensure all systems and practices adhere to the latest security standards and regulations. This includes periodic vulnerability assessments and penetration testing to identify and address potential security weaknesses. Strengthening Identity and Access Management (IAM) policies by implementing multi-factor authentication (MFA) for all users, especially those with elevated privileges, is crucial. Utilizing IAM conditions to enforce context-aware access, restricting access based on user location, device, or time of day, can further enhance security. Data encryption enhancements, such as implementing envelope encryption, provide additional layers of protection. Integrating automated security response mechanisms using Google Cloud's Security Command Center and Chronicle helps detect and respond to security threats in real time, minimizing response times and mitigating potential damage quickly. Regular security awareness training for all employees is also essential, as educating staff about the latest phishing techniques, social engineering attacks, and best practices for data security can significantly reduce the risk of security incidents caused by human error.

References

- List of all sources and documentation referenced in the report.

1. Google Cloud. (n.d.). **Identity and Access Management (IAM)**. Retrieved from Google Cloud IAM Documentation
2. Google Cloud. (n.d.). **Cloud Key Management Service (KMS)**. Retrieved from Google Cloud KMS Documentation
3. Google Cloud. (n.d.). **Encryption at Rest**. Retrieved from Google Cloud Encryption at Rest Documentation
4. Google Cloud. (n.d.). **Encryption in Transit**. Retrieved from Google Cloud Encryption in Transit Documentation
5. Google Cloud. (n.d.). **Cloud Audit Logs**. Retrieved from Google Cloud Audit Logs Documentation
6. Google Cloud. (n.d.). **Cloud Monitoring**. Retrieved from Google Cloud Monitoring Documentation
7. Google Cloud. (n.d.). **Security Command Center**. Retrieved from Google Cloud Security Command Center Documentation
8. Google Cloud. (n.d.). **Google Kubernetes Engine (GKE)**. Retrieved from Google Kubernetes Engine Documentation
9. Google Cloud. (n.d.). **Auto-Scaling**. Retrieved from Google Cloud Auto-Scaling Documentation
10. Google Cloud. (n.d.). **Load Balancing**. Retrieved from Google Cloud Load Balancing Documentation
11. Google Cloud. (n.d.). **Google Cloud Functions**. Retrieved from Google Cloud Functions Documentation

12. Google Cloud. (n.d.). **Cost Management**. Retrieved from Google Cloud Cost Management Documentation
13. Snyk. (n.d.). **Snyk Vulnerability Scanner**. Retrieved from Snyk Documentation
14. OWASP. (n.d.). **OWASP ZAP**. Retrieved from OWASP ZAP Documentation
15. TensorFlow. (n.d.). **TensorFlow Documentation**. Retrieved from TensorFlow Documentation
16. Chronicle. (n.d.). **Chronicle Documentation**. Retrieved from Chronicle Documentation

Appendices

- Additional material, such as code snippets, configuration examples, or detailed charts and graphs.

A.1. IAM Configuration Example

```
# Create a service account
gcloud iam service-accounts create my-service-account --display-name "My Service Account"

# Assign roles to the service account
gcloud projects add-iam-policy-binding my-project --member "serviceAccount:my-service-account@my-project.iam.gserviceaccount.com" --role "roles/storage.objectViewer"
gcloud projects add-iam-policy-binding my-project --member "serviceAccount:my-service-account@my-project.iam.gserviceaccount.com" --role "roles/cloudsql.client"
```

A.2. Enabling Encryption at Rest

```
# Enable Cloud KMS API
gcloud services enable cloudkms.googleapis.com

# Create a keyring and a key
gcloud kms keyrings create my-keyring --location global
gcloud kms keys create my-key --location global --keyring my-keyring --purpose encryption

# Encrypt a file using the key
gcloud kms encrypt --location global --keyring my-keyring --key my-key --plaintext-file my-data.txt --ciphertext-file my-data.txt.enc
```

A.3. Cloud Functions Example

```
import json

def hello_world(request):
    request_json = request.get_json()
    if request_json and 'message' in request_json:
        return json.dumps({'message': request_json['message']})
    else:
        return json.dumps({'message': 'Hello, World!'})
```

A.4. Auto-Scaling Configuration for GKE

```
# gke-autoscaling.yaml
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: my-app-autoscaler
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: my-app
  minReplicas: 1
  maxReplicas: 10
  targetCPUUtilizationPercentage: 80
```

B.1. Load Balancer Configuration

```
# Create a health check
gcloud compute health-checks create http my-health-check --port 80

# Create a backend service and attach the health check
gcloud compute backend-services create my-backend-service --protocol HTTP --health-checks my-health-check --global

# Add instance groups to the backend service
gcloud compute backend-services add-backend my-backend-service --instance-group my-instance-group --instance-group-zone us-central1-a --global

# Create a URL map to route incoming requests
gcloud compute url-maps create my-url-map --default-service my-backend-service

# Create an HTTP proxy to route requests to the URL map
gcloud compute target-http-proxies create my-http-proxy --url-map my-url-map

# Create a global forwarding rule to handle incoming requests
gcloud compute forwarding-rules create my-forwarding-rule --global --target-http-proxy my-http-proxy --ports 80
```

B.2. Security Scanning Integration in CI/CD Pipeline

```
# .gitlab-ci.yml example for integrating Snyk
stages:
  - test
snyk_test:
  stage: test
  script:
    - npm install -g snyk
    - snyk test
  only:
    - master
```