KAZAKH-BRITISH TECHNICAL UNIVERSITY

Midterm Project Report

Deploying a Scalable Web Application on Google Cloud Platform

Done by:

Amen Azat, 21B030774

Almaty, 2024

Table of Contents:

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

---

1. Executive Summary

The goals of my project were to create a responsive and scalable web application. I fulfilled these goals by creating a to-do list application hosted on the Google Cloud Platform (GCP).

In my project, I used the Flask framework to develop a web application and various Google Cloud Platform services for deployment, logging, containerization, API management and other purposes. These services include Google App Engine, Google Cloud Functions, Google Cloud Endpoints and Google Kubernetes Engine.

As a result, I managed to create my **To-Do List** project deployed on the Google Cloud Platform. Using the above technologies, the web application became responsive, highly available and scalable.

2. Introduction

Google Cloud Platform is one of the best cloud platforms on a par with platforms such as Amazon Web Services (AWS), Microsoft Azure. The main advantage of such platforms is scalability, flexibility and fault tolerance. Also, an important advantage of such cloud platforms is a wide range of ready-made services and tools, which significantly saves development time and resources.

The Google Cloud Platform offers a user-friendly console with a user interface and analytics tools. GCP includes technologies such as the App Engine, Kubernetes and a wide range of different services, technologies and APIs.

The motivation for choosing GCP was that it provides a free period of 90 days with 300 dollars. Also, as I wrote above, their wide range of services and technologies.

4. Project Objectives

My project objectives include:

1. Develop To-Do List web-application using Flask framework.
2. Deploy my application to the Google App Engine.

3. Create 2 Google Cloud Functions for processing user inputs and sending notifications, after that implement them to the main application.
4. Containerize app using Docker and deploy it to Google Kubernetes Engine.
5. Set up API with authentication and monitoring for the application using Google Cloud Endpoints.
6. Write unit, integration and load test to evaluate application functionality and scalability.
7. Use GCP monitoring tools to track app performance and establish maintenance practices to ensure uptime and reliability.

4. Google Cloud Platform Overview

The Google Cloud Platform architecture includes a large number of servers around the world. GCP has a huge number of services and technologies for different types of tasks. GCP includes services like:

- Compute Engine for creating virtual machines.
- App Engine for deploying web applications.
- Google Kubernetes Engine for containerizing applications.
- Cloud Function for creating serverless functions.
- Cloud Endpoints for API management.
- Various cloud storages such as Cloud SQL, Cloud Storage, Firestore for data storage.
- And many other useful services and tools.

The advantage of GCP is that it can be used to create scalable, flexible, highly available and fault-tolerant applications.

5. Google Cloud SDK and Cloud Shell

1) Setup:

To work with the Google Cloud SDK, you must download the Google Cloud CLI:

1. Visit Google Cloud SDK installation page using this url: https://cloud.google.com/sdk/docs/install
2. Press the following button on the website to download it:

   1. Download the Google Cloud CLI installer.

3. After downloading open installer and follow the instructions install the CLI to your operating system.
4. Once installation is complete, go to the terminal and run gcloud init to initialize the SDK and authenticate with your Google account.
5. And finally, to verify the installation run gcloud version and gcloud info.

```
C:\Users\azikkw>gcloud version
Google Cloud SDK 495.0.0
bq 2.1.8
core 2024.09.27
gcloud-crc32c 1.0.0
gke-gcloud-auth-plugin 0.5.9
gsutil 5.30
```

6. The next step is to create a project for our future application. Use following command to create project:

gcloud projects create PROJECT_ID --name="your project name"

7. And use following command to set this project as your default project:

   gcloud config set project [PROJECT_ID]

2) Cloud Shell Usage: While doing Midterm Project, I did not use the Google Cloud Shell, because I used the Google Cloud CLI and executed all the necessary operations and commands in my terminal. Then you can see how I did it, because I described all the steps in detail.

6. Google App Engine

1) Application Development:

I created a **To-Do List** web application using the **Flask** framework. This web application is hosted on Google Cloud Platform (GCP). The application utilizes various GCP services, including:

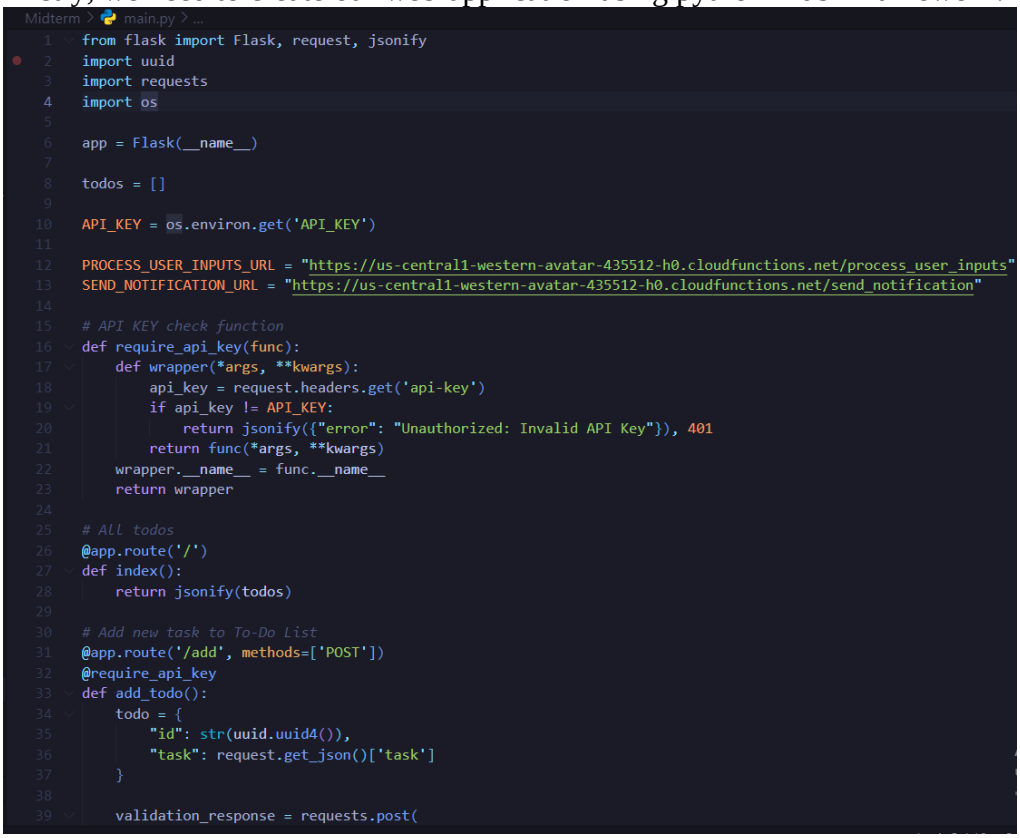
1. Google App Engine - for app deployment.
2. Google Cloud Functions - serverless functions for processing user inputs and sending notifications.
3. Google Cloud Endpoints - …
4. Google Kubernetes Engine (GKE) - using containerization for deployment.

My **To-Do List** has simple functionality:

- Creating a task
- Completing the task
- View all tasks

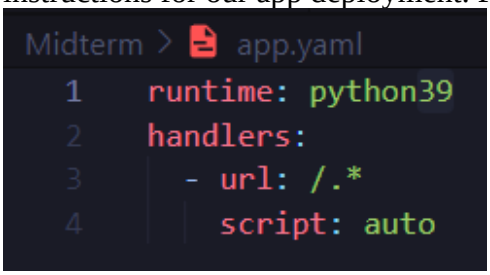
2) Deployment: Step-by-step guide on deploying the application to App Engine.

1. Firstly, we need to create our web-application using python **Flask** framework. For example:

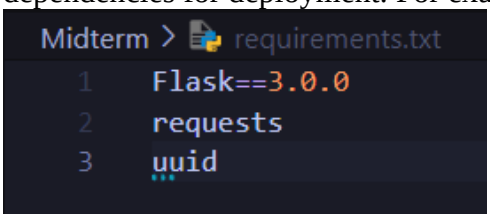


I created **main.py** file that contains app logic and **index.html** with UI.

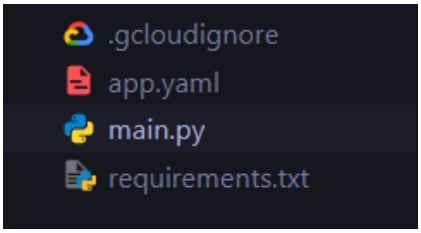
2. At the second step, we should to add **app.yaml** configuration file with necessary instructions for our app deployment. For example:



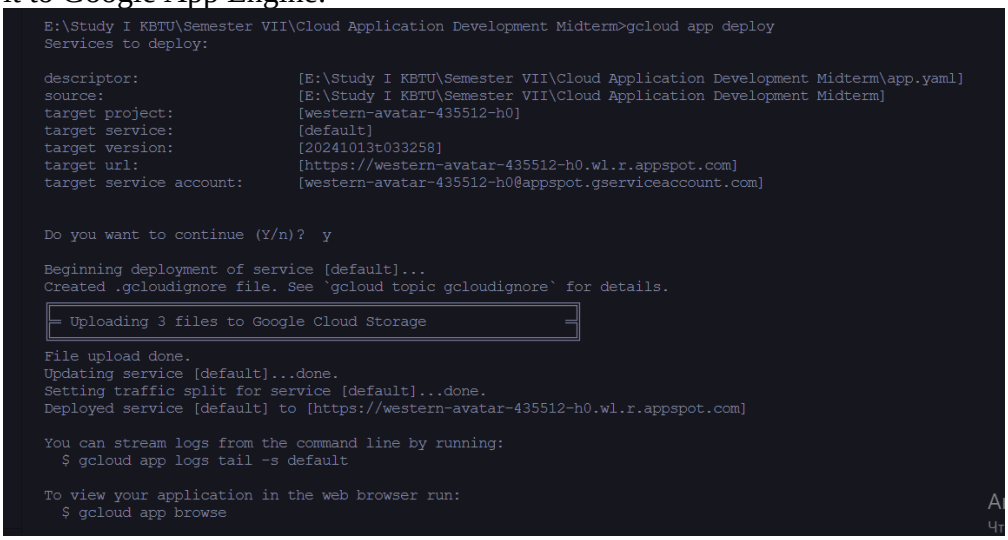
Also, we need to add **requirements.txt** with the necessary python libraries and dependencies for deployment. For example:

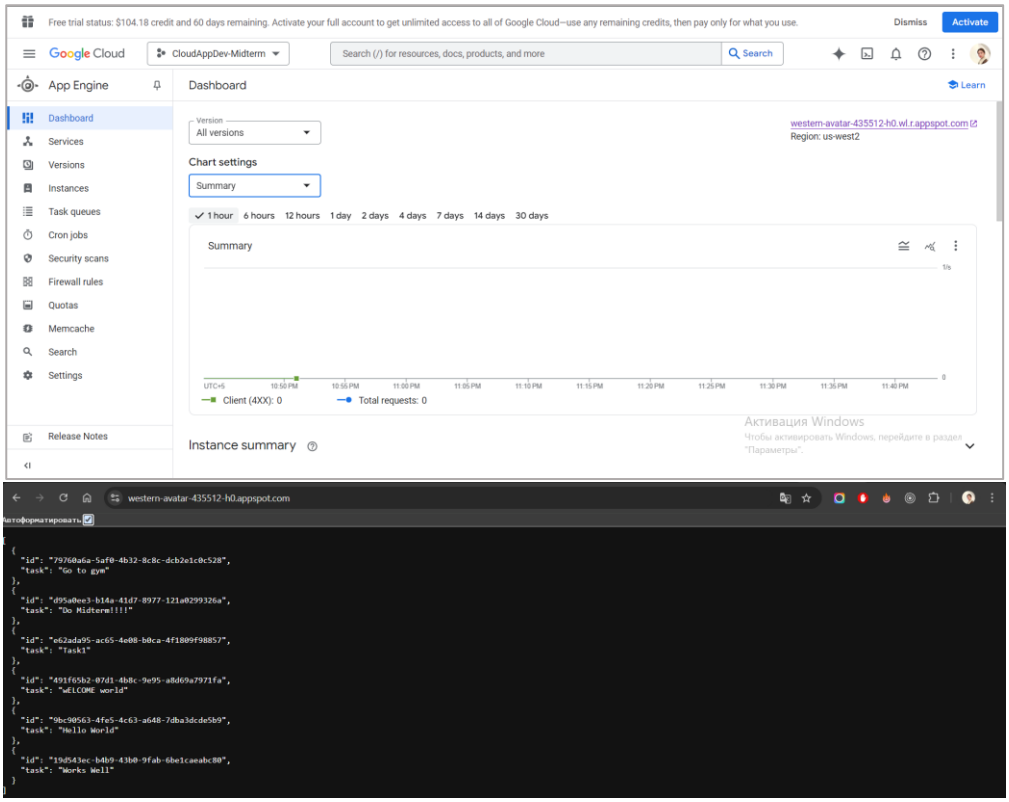


As a result, you will have following project structure:



3. And finally, you need to deploy the application. Use gcloud app deploy command to deploy it to Google App Engine:





7. Building with Google Cloud Functions

In general, I created two cloud functions for processing user inputs and sending notifications. Let's take a step-by-step look at how I created and deployed them:

	Environment	Name	Last deployed	Region	Recommendation	Trigger	Runtime	Memory allocated	Executed function
✓	Cloud Run function	process_user_inputs	Oct 15, 2024, 11:36:27 AM	us-central1		HTTP	Python 3.9	256 MB	process_user_inputs
✓	Cloud Run function	send_notification	Oct 15, 2024, 12:08:05 PM	us-central1		HTTP	Python 3.9	256 MB	send_notification

1) “process_user_inputs” cloud function:

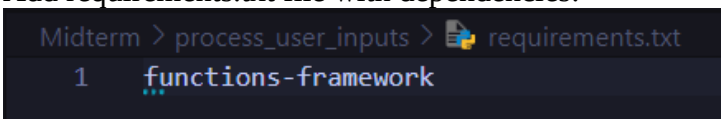
1. Firstly, create function called processing_user_inputs. I also used python:

Midterm > process_user_inputs > main.py > process_user_inputs

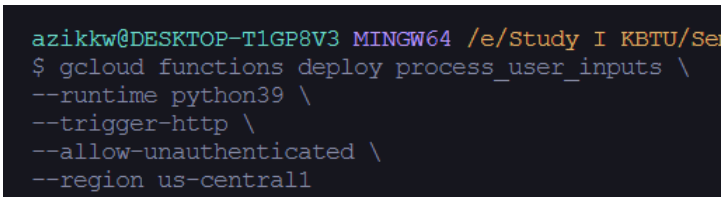
```python
import functions_framework

@functions_framework.http
def process_user_inputs(request):
    if request.content_type != 'application/json':
        return {"error": "Unsupported Media Type"}, 415

    request_json = request.get_json()

    if not request_json or 'id' not in request_json or 'task' not in request_json:
        return {"error": "No todo provided!"}, 400

    todo = request_json

    if len(todo['task']) == 0:
        return {"error": "No task provided!"}, 400

    return {"message": "New todo created and validated."}, 200
```

This function receive to-do, validate it and send response back with some answers according to the to-do condition.

2. Add requirements.txt file with dependencies:



3. Deploy cloud function using the following configuration:
   - **Name:** process_user_inputs
   - **Trigger:** HTTP
   - **Runtime:** python39 (or another supported runtime)
   - Allow unauthenticated sources, requests, etc.
   - **Region:** us-central1 (or another region)

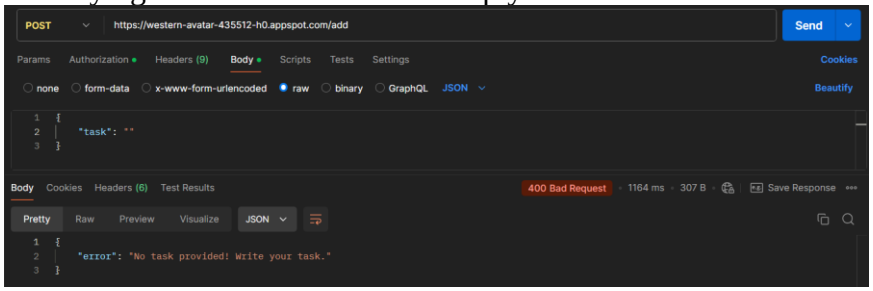


4. Integrate cloud function to the main application:

```
# Add new task to To-Do List
@app.route('/add', methods=['POST'])
@require_api_key
def add_todo():
    todo = {
        "id": str(uuid.uuid4()),
        "task": request.get_json()['task']
    }

    validation_response = requests.post(
        PROCESS_USER_INPUTS_URL,
        headers={'Content-Type': 'application/json'},
        json=todo
    )

    if validation_response.json().get('error') == 'No todo provided!':
        error_message = {f"{validation_response.status_code}": f"{validation_response.text}"}
        return jsonify(error_message), 400
    elif validation_response.json().get('error') == 'No task provided!':
        error_message = 'No task provided! Write your task.'
        return jsonify({"error": error_message}), 400
    else:
        todos.append(todo)

        requests.post(
            SEND_NOTIFICATION_URL,
            headers={'Content-Type': 'application/json'},
            json={
                "todo": todo,
                "action": "created"
            }
        )

    return jsonify(todo)
```

So, it sends POST request (validation_response) to process_user_inputs cloud function. Cloud function returns response. According to response, it shows error if status_code is not 200, else if to-do is empty show that task not provided, if everything is well it adds to-do.

5. Example of **process_user_inputs**:
I am trying to add new to-do with empty task:



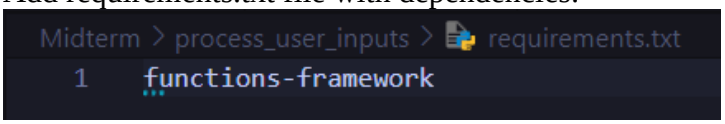
And after validation receive following response from function: ‘No task provided! Write your task’. So, it means that my cloud function work well and process user inputs.

2. “send_notification” cloud function:

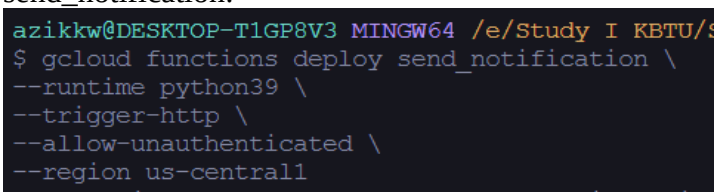
1. Also create function but call it send_notifcation. I also used python:

This function sends notification (in logs) that to-do created or deleted according to action.

2. Add requirements.txt file with dependencies:



3. Deploy cloud function using with configuration like in first function but change name to send_notification:



4. Integrate cloud function to the main application:

So, on first image it sends POST request with **action: created** to cloud function and it will show message in logs that to-do created. On second **action: deleted**, so it will show message in logs that to-do deleted.

5. Example of **send_notification**:
I added new to-do with task “New Task”, after updated it to “Updated Task” and finally I deleted it:

8. Containerizing Applications

1) Docker Overview:

First of all, what is a Docker? Docker is a tool that allows developers to containerize their applications and run them on various systems.

The containerization process using Docker includes the following: create an image and assemble the container from the image.

Docker Image is a template containing code, dependencies, libraries, and all other instructions necessary to create a container.
Container is an assembled image that runs in an isolated environment that does not affect the main operating system.

Containerization process:

1. Create a Dockerfile (A file with instructions for creating a Docker Image).
2. Using the docker build command, create an image.
3. Launch the container using the docker run command.

2) GKE Deployment:

Steps taken to deploy the containerized application on GKE.

1. Create a Dockerfile with instrucations for creating Docker Image:

```
Midterm > Dockerfile
1  FROM python:3.9
2
3  WORKDIR /app
4
5  COPY . .
6
7  RUN pip install --no-cache-dir Flask requests gunicorn
8
9  EXPOSE 8080
10
11 CMD ["gunicorn", "--bind", "0.0.0.0:8080", "main:app"]
```

2. Create a Image using docker build:

```
azikkw@DESKTOP-T1GP8V3 MINGW64 /e/Study I KBTU/Semester VII/Cloud Application Development I Serek A/Midterm (master)
$ docker build -t flask-todo-app .
[+] Building 406.9s (9/9) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 652B
 => [internal] load metadata for docker.io/library/python:3.9
 => [internal] load .dockerignore
 => => transferring context: 108B
 => [1/4] FROM docker.io/library/python:3.9@sha256:a23efa04a7f7a881151fe5d47377058 8ef639c08fd5f0dcc6987dbe13705c829
 => => resolve docker.io/library/python:3.9@sha256:a23efa04a7f7a881151fe5d47377058 8ef639c08fd5f0dcc6987dbe13705c829
 => => sha256:a23efa04a7f7a881151fe5d473770588ef639c08fd5f0dcc6987dbe13705c829 10.35kB / 10.35kB
 => => sha256:97e0419359faf7c2f3874d2eff7484d2529a50d510ae058462983ce09aaef578 2.32kB / 2.32kB
 => => sha256:6fa82efb09a44285e0b3a942e752e69f1f53f1226bc01aa8deb4767421bfa404 5.92kB / 5.92kB
 => => sha256:cdd62bf39133c498a16f7a7b1b6555ba43d02b2511c508fa4c0a9b1975ffe20e 49.56MB / 49.56MB
 => => sha256:a47cff7f31e941e78bf63ca19f0811b675283e2c00ddea10c57f78d93b2bc343 24.05MB / 24.05MB
 => => sha256:a173f2aee8e962ea19db1e418ae84a0c9f71480b51f768a19332dfa83d7722a5 64.39MB / 64.39MB
 => => sha256:01272fe8adbacc44afd2b92994b31c40a151f4324ca392050d9e8d580927dd32 211.27MB / 211.27MB
 => => extracting sha256:cdd62bf39133c498a16f7a7b1b6555ba43d02b2511c508fa4c0a9b1975ffe20e
 => => sha256:bd760507420c879e204e9491643b063834e56cf8b19035a3948b52f3b630e8f9 6.16MB / 6.16MB
 => => extracting sha256:a47cff7f31e941e78bf63ca19f0811b675283e2c00ddea10c57f78d93b2bc343
 => => sha256:c536ab81c8a0c55c8f9639481522d8d8a28ce68a8429022194f8f24024c7edfa 18.68MB / 18.68MB
 => => sha256:b42fbf9dc0a3393e0c5e17668de23062db618b07060f7ce59031107b1106a325 248B / 248B
 => => extracting sha256:a173f2aee8e962ea19db1e418ae84a0c9f71480b51f768a19332dfa83d7722a5
 => => extracting sha256:01272fe8adbacc44afd2b92994b31c40a151f4324ca392050d9e8d580927dd32
 => => extracting sha256:bd760507420c879e204e9491643b063834e56cf8b19035a3948b52f3b630e8f9
 => => extracting sha256:c536ab81c8a0c55c8f9639481522d8d8a28ce68a8429022194f8f24024c7edfa
 => => extracting sha256:b42fbf9dc0a3393e0c5e17668de23062db618b07060f7ce59031107b1106a325
 => [internal] load build context
 => => transferring context: 4.77kB
 => [2/4] WORKDIR /app
 => [3/4] COPY . .
 => [4/4] RUN pip install --no-cache-dir Flask requests gunicorn
 => exporting to image
 => => exporting layers
 => => writing image sha256:57df4552e91e66c57ac5e8ff6f6ab871c2c6100826eb1d3371d94f457e8a115a
 => => naming to docker.io/library/flask-todo-app

What's next:
    View a summary of image vulnerabilities and recommendations → docker scout quickview
```

3. Tag Docker Image to send it to Google Container Registry (GCR):

```
azikkw@DESKTOP-T1GP8V3 MINGW64 /e/Study I KBTU/Semester VII/Cloud Application Development I Serek A/Midterm (master)
$ docker tag flask-todo-app gcr.io/western-avatar-435512-h0/flask-todo-app:latest
```

4. Push Image to GCR:

```
azikkw@DESKTOP-T1GP8V3 MINGW64 /e/Study I KBTU/Semester VII/Cloud Application Development I Serek A/Midterm (master)
$ docker push gcr.io/western-avatar-435512-h0/flask-todo-app:latest
The push refers to repository [gcr.io/western-avatar-435512-h0/flask-todo-app]
92e9846dbd0d: Pushed
9fefa5f2eb3a: Pushed
944cb74b4732: Pushed
5026504e37c5: Layer already exists
a7939b3dd113: Layer already exists
22ec93acf2d9: Layer already exists
2bce433c3a29: Layer already exists
f91dc7a486d9: Layer already exists
3e14a6961052: Layer already exists
d50132f2fe78: Layer already exists
latest: digest: sha256:e98cdf4d257bb0f58807470351bb2622506e293fc5b95d62d9fbf2149fd62ff6 size: 2421
```

5. Next step is to create kluster in Google Kluster Engine (GKE):

```
azikkw@DESKTOP-T1GP8V3 MINGW64 /e/Study I KBTU/Semester VII/Cloud Application Development I Serek A/Midterm (master)
$ gcloud container clusters create todo-cluster --zone us-central1 --num-nodes 1 --disk-type pd-standard
Note: The Kubelet readonly port (10255) is now deprecated. Please update your workloads to use the recommended alternatives. See https://cloud.google.com/kubernetes-engine/docs/how-to/disable-kubelet-readonly-port 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... Cluster is being health-checked (Kubernetes Control Plane is healthy)...done.
Created [https://container.googleapis.com/v1/projects/western-avatar-435512-h0/zones/us-central1/clusters/todo-cluster].
To inspect the contents of your cluster, go to: https://console.cloud.google.com/kubernetes/workload_/gcloud/us-central1/todo-cluster?project=western-avatar-435512-h0
CRITICAL: ACTION REQUIRED: gke-gcloud-auth-plugin, which is needed for continued use of kubectl, was not found or is not executable. Install gke-gcloud-auth-plugin for use with kubectl by following https://cloud.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  1.30.5-gke.1014001  34.41.237.224  e2-medium     1.30.5-gke.1014001  3          RUNNING
```

6. Connecting to todo-cluster cluster:

```
azikkw@DESKTOP-T1GP8V3 MINGW64 /e/Study I KBTU/Semester VII/Cloud Application Development I Serek A/Midterm (master)
$ gcloud container clusters get-credentials todo-cluster --zone us-central1
Fetching cluster endpoint and auth data.
kubeconfig entry generated for todo-cluster.
```

7. Next, create deployment.yaml with deployment instructions for Kubernetes:

Midterm > deployment.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: flask-todo-app
spec:
  replicas: 1
  selector:
    matchLabels:
      app: flask-todo-app
  template:
    metadata:
      labels:
        app: flask-todo-app
    spec:
      containers:
      - name: flask-todo-app
        image: gcr.io/western-avatar-435512-h0/flask-todo-app:latest
        ports:
        - containerPort: 8080
---
apiVersion: v1
kind: Service
metadata:
  name: flask-todo-service
spec:
  type: LoadBalancer
  selector:
    app: flask-todo-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
```

8. Deploying the application using deployment.yaml:

```
azikkw@DESKTOP-T1GP8V3 MINGW64 /e/Study I KBTU/Semester VII/Cloud Application Development I Serek A/Midterm (master)
$ kubectl apply -f deployment.yaml
deployment.apps/flask-todo-app created
service/flask-todo-service created
```

9. And finally, check deployed app:

```
azikkw@DESKTOP-T1GP8V3 MINGW64 /e/Study I KBTU/Semester VII/Cloud Application Development I Serek A/Midterm (master)
$ kubectl get pods
NAME                              READY   STATUS    RESTARTS   AGE
flask-todo-app-685fff7d6b-htxgs   1/1     Running   0          4m8s

azikkw@DESKTOP-T1GP8V3 MINGW64 /e/Study I KBTU/Semester VII/Cloud Application Development I Serek A/Midterm (master)
$ kubectl get services
NAME                TYPE           CLUSTER-IP      EXTERNAL-IP     PORT(S)        AGE
flask-todo-service  LoadBalancer   34.118.229.92   34.134.187.99   80:31338/TCP   25m
```

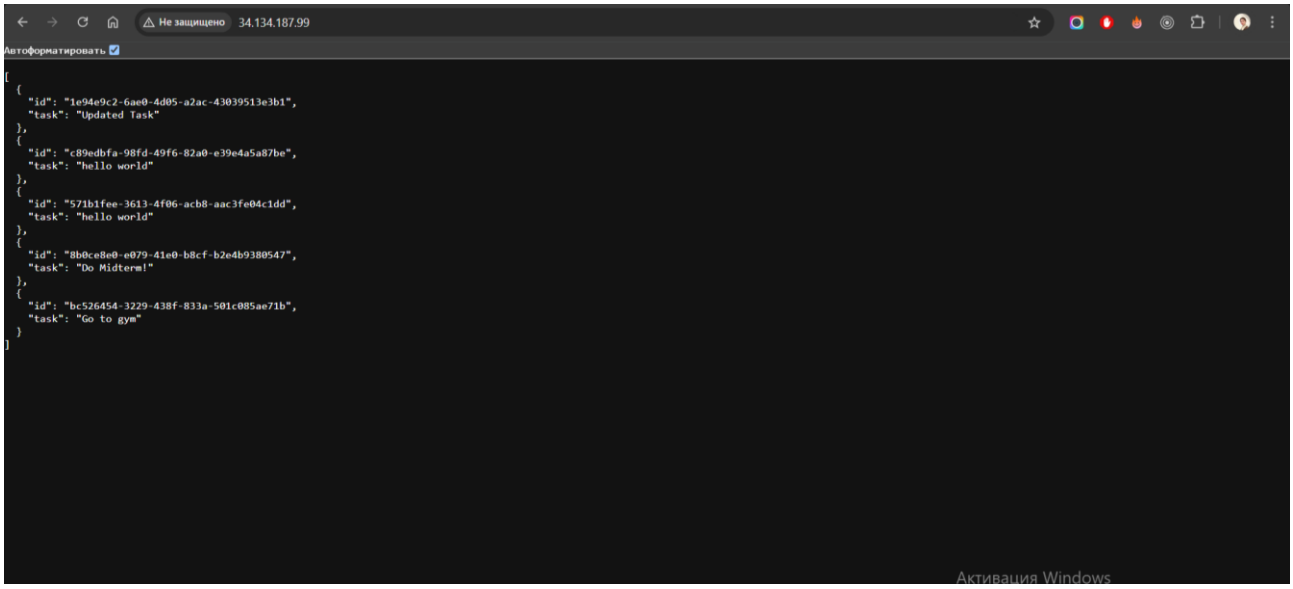
As a result we will have:

1. Cluster named todo-cluster:

Status	Name ↑	Location	Number of nodes	Total vCPUs	Total memory
✓	todo-cluster	us-central1	3	6	12 GB

2. And containerized app deployed to the GKE on the http://34.134.187.99/ IP address:

Name ↑	Status	Type	Pods	Namespace	Cluster
flask-todo-app	OK	Deployment	1/1	default	todo-cluster



9. Managing APIs with Google Cloud Endpoints

1) Api Setup:

1. The first step is create openapi.yaml configuration file in the project directory:

```
Midterm > openapi.yaml
swagger: '2.0'
info:
  title: To-Do List API
  description: Google Cloud Endpoints API for To-Do List.
  version: 1.0.0
host: "western-avatar-435512-h0.appspot.com"
schemes:
  - https
paths:
  /todos:
    get:
      summary: Get all todos
      operationId: getTodos
      responses:
        '200':
          description: Todos list
        '404':
          description: No todos found
      security:
        - api_key: []
  /add:
    post:
      summary: Add a new task
      operationId: addTodo
      parameters:
        - in: body
```

This image does not contain all openapi.yaml, but I will just explain. As you can see on image, in this file we need to write version of swagger, info about your API, host (choose your project), schemes with https for safety, after that you write your pathes (Endpoints) and describe them including method type, parameters, responses and security.

2. Once openapi.yaml is complete, deploy your API to Google Cloud Endpoints using following command:

   gcloud endpoints services deploy openapi.yaml

```
azikkw@DESKTOP-T1GP8V3 MINGW64 /e/Study I KBTU/Semester VII/Cloud Application Development I Serek A/Midterm (master)
$ gcloud endpoints services deploy openapi.yaml
Waiting for async operation operations/serviceConfigs.western-avatar-435512-h0.appspot.com:1fb99c68-cbb8-427c-9ceb-cfcf985ead88 to complete...
Operation finished successfully. The following command can describe the Operation details:
 gcloud endpoints operations describe operations/serviceConfigs.western-avatar-435512-h0.appspot.com:1fb99c68-cbb8-427c-9ceb-cfcf985ead88

Waiting for async operation operations/rollouts.western-avatar-435512-h0.appspot.com:f4b60374-f044-44b5-9534-195eafc7f5bf to complete...
Operation finished successfully. The following command can describe the Operation details:
 gcloud endpoints operations describe operations/rollouts.western-avatar-435512-h0.appspot.com:f4b60374-f044-44b5-9534-195eafc7f5bf

Service Configuration [2024-10-17r0] uploaded for service [western-avatar-435512-h0.appspot.com]

To manage your API, go to: https://console.cloud.google.com/endpoints/api/western-avatar-435512-h0.appspot.com/overview?project=western-avatar-435512-h0
```

3. After API deployed, you need to deploy your project to App Engine using command:

   gcloud app deploy

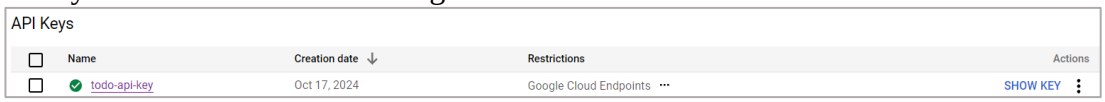
2) Security and Monitoring:

To implement **authentication** to the project we need to do following:

1. Create API Key using gcloud services api-keys create --display-name="your api name" command.
2. Use gcloud services api-keys list to see api key is created:

```
azikkw@DESKTOP-T1GP8V3 MINGW64 /e/Study I KBTU/Semester VII/Cloud Application Developme
$ gcloud services api-keys list
---
createTime: '2024-10-17T13:04:14.086093Z'
displayName: todo-api-key
etag: W/"muPYGaYCNUkSHyq2XrlkUg=="
name: projects/968084628117/locations/global/keys/8199cfe1-2c18-444e-857b-a7d8695fe4d7
restrictions:
  apiTargets:
  - service: endpoints.googleapis.com
uid: 8199cfe1-2c18-444e-857b-a7d8695fe4d7
updateTime: '2024-10-17T14:11:31.499117Z'
```

   Also you can check it on the Google Cloud Console in “API & Services”.



3. Use **SHOW KEY** button to see API Key.
4. The next step is to restrict the key to use only with certain APIs, and we need Google Cloud Endpoints. To do that use gcloud api-keys update api_key_uid --api targets=service=endpoints.googleapis.com command.

5. Also is important to update openapi.yaml. Add securityDefinitions and configure security **to each path (Endpoint)**. For example:

```yaml
/delete/{todo_id}:
  get:
    summary: Delete a task
    operationId: deleteTodo
    parameters:
      - in: path
        name: todo_id
        type: string
        required: true
    responses:
      '200':
        description: Todo deleted
      '404':
        description: Todo not found
    security:
      - api_key: []
```

```yaml
securityDefinitions:
  api_key:
    type: apiKey
    name: api-key
    in: header
```

6. Once you do this, add your API Key to app.yaml file like that:

```yaml
Midterm > app.yaml
1  runtime: python39
2  handlers:
3    - url: /.*
4      script: auto
5  env_variables:
6    API_KEY: "AIzaSyAa0paGkZS8yniRf4dEuOZjoErJH7_9VZM"
```

7. The next step is to add api key check in your application:

```python
10  API_KEY = os.environ.get('API_KEY')
```

```python
# API KEY check function
def require_api_key(func):
    def wrapper(*args, **kwargs):
        api_key = request.headers.get('api-key')
        if api_key != API_KEY:
            return jsonify({"error": "Unauthorized: Invalid API Key"}), 401
        return func(*args, **kwargs)
    wrapper.__name__ = func.__name__
    return wrapper
```

We created require_api_key function that checks api key from headers and returns response according to the api key existance.

8. And add this to all your functions (like add_todo, delete_todo, etc…):

```python
# Add new task to To-Do List
@app.route('/add', methods=['POST'])
@require_api_key
def add_todo():
```

9. When you complete all this step deploy changes in API using gcloud endpoints services deploy openapi.yaml and deploy your project updates to the App Engine using gcloud app deploy
10. Finally lets test it:
    - With API Key in header:

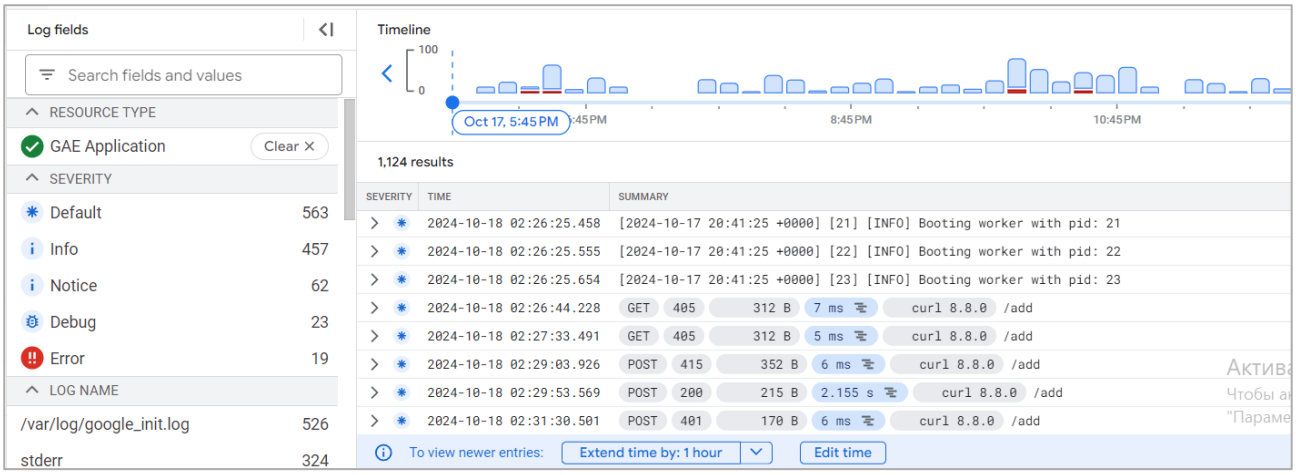
```
azikkw@DESKTOP-T1GP8V3 MINGW64 /e/Study I KBTU/Semester VII/Cloud Application Development I Serek A/Midterm (master)
$ curl -X POST "https://western-avatar-435512-h0.appspot.com/add" -H "api-key: AIzaSyAa0paGkZS8yniRf4dEuOZjoErJH7_9VZM"
-H "Content-Type: application/json" -d '{"task": "New Task"}'
{"id":"57d7a860-de8b-40b6-a3d4-44eafef178ac","task":"New Task"}
```

    - Without API Key:

```
azikkw@DESKTOP-T1GP8V3 MINGW64 /e/Study I KBTU/Semester VII/Cloud Application Development I Serek A/Midterm (master)
$ curl -X POST "https://western-avatar-435512-h0.appspot.com/add" -H "Content-Type: application/json" -d '{"task": "New
Task"}'
{"error":"Unauthorized: Invalid API Key"}
```

Thus, as a result, the application has become more secure and safer, because the implemented authentication works very well and blocks unauthorized access.

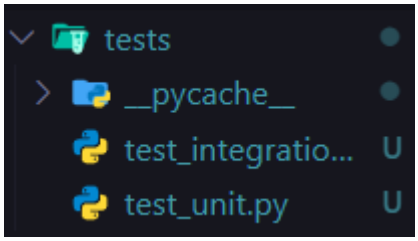
Monitoring is already implemented in Google Cloud Platform. Just open your project in the Logs Explorer (Its monitoring tool):

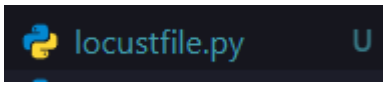


As you can see here, its my POST methods when I trying add new todo in the application.

10. Testing and Quality Assurance

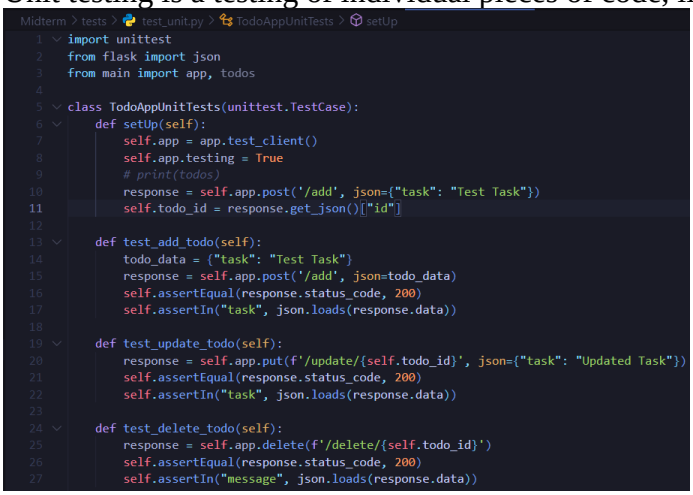
I created unit, integration and load test.





Under you will see the results of my unit and integration tests:

1. Unit testing is a testing of individual pieces of code, like a functions. (Full code in the zip):



```python
import unittest
from flask import json
from main import app, todos

class TodoAppUnitTests(unittest.TestCase):
    def setUp(self):
        self.app = app.test_client()
        self.app.testing = True
        # print(todos)
        response = self.app.post('/add', json={"task": "Test Task"})
        self.todo_id = response.get_json()["id"]

    def test_add_todo(self):
        todo_data = {"task": "Test Task"}
        response = self.app.post('/add', json=todo_data)
        self.assertEqual(response.status_code, 200)
        self.assertIn("task", json.loads(response.data))

    def test_update_todo(self):
        response = self.app.put(f'/update/{self.todo_id}', json={"task": "Updated Task"})
        self.assertEqual(response.status_code, 200)
        self.assertIn("task", json.loads(response.data))

    def test_delete_todo(self):
        response = self.app.delete(f'/delete/{self.todo_id}')
        self.assertEqual(response.status_code, 200)
        self.assertIn("message", json.loads(response.data))
```

2. Integration testing is a testing, where individual pieces of code are combined and tested in a group. It performs after unit testing. (Full code in the zip). Also used request_mock for mock results from url:

```python
import unittest
from flask import json
from main import app
import requests_mock

class TodoAppIntegrationTests(unittest.TestCase):
    def setUp(self):
        self.app = app.test_client()
        self.app.testing = True
        response = self.app.post('/add', json={"task": "Test Task"})
        self.todo_id = response.get_json()["id"]

    @requests_mock.Mocker()
    def test_add_todo_integration(self, mock_requests):
        mock_requests.post('https://us-central1-western-avatar-435512-h0.cloudfunctions.net/process_user_inputs', json={}, status_code=200)
        mock_requests.post('https://us-central1-western-avatar-435512-h0.cloudfunctions.net/send_notification', json={}, status_code=200)

        response = self.app.post('/add', json={"task": "Integration Test Task"})
        self.assertEqual(response.status_code, 200)
        self.assertIn("task", json.loads(response.data))

    @requests_mock.Mocker()
    def test_update_todo_integration(self, mock_requests):
        mock_requests.post('https://us-central1-western-avatar-435512-h0.cloudfunctions.net/process_user_inputs', json={}, status_code=200)
        mock_requests.post('https://us-central1-western-avatar-435512-h0.cloudfunctions.net/send_notification', json={}, status_code=200)

        response = self.app.put(f'/update/{self.todo_id}', json={"task": "Updated Task"})
```

3. Results:

```
azikkw@DESKTOP-T1GP8V3 MINGW64 /e/Study I KBTU/Semester VII/Cloud Applicat
$ python -m unittest discover -s tests
.Deleting todo with id: 2925c794-3cfa-478a-9e21-ca10fc19a5fb
...Deleting todo with id: d842809e-162e-473c-b695-e142e360b57e
...
----------------------------------------------------------------------
Ran 7 tests in 9.538s

OK
```

Also load testing was performed. Load testing is testing that is necessary to verify the functionality of the application. Using it, you can find out how many requests the system will be able to process in an n-th amount of time.

For load testing I used library called **locust** with UI and statistical tools. Load testing code (Full):

```python
from locust import HttpUser, task, between

class TodoAppUser(HttpUser):
    wait_time = between(1, 3)

    @task(1)
    def add_todo(self):
        response = self.client.post("/add", headers={"api-key": "AIzaSyAa0paGkZS8yniRf4dEuOZjoErJH7_9VZM"}, json={"task": "Load Test Task"})
        if response.status_code == 200:
            todo_id = response.json().get("id")
            self.update_todo(todo_id)

    def update_todo(self, todo_id):
        response = self.client.put(f"/update/{todo_id}", headers={"api-key": "AIzaSyAa0paGkZS8yniRf4dEuOZjoErJH7_9VZM"}, json={"task": "Updated Load T
        if response.status_code == 200:
            self.delete_todo(todo_id)

    def delete_todo(self, todo_id):
        self.client.delete(f"/delete/{todo_id}", headers={"api-key": "AIzaSyAa0paGkZS8yniRf4dEuOZjoErJH7_9VZM"})

    @task(1)
    def get_all_todos(self):
        self.client.get("/", headers={"api-key": "AIzaSyAa0paGkZS8yniRf4dEuOZjoErJH7_9VZM"})

if __name__ == "__main__":
    import os
    os.system("locust")
```

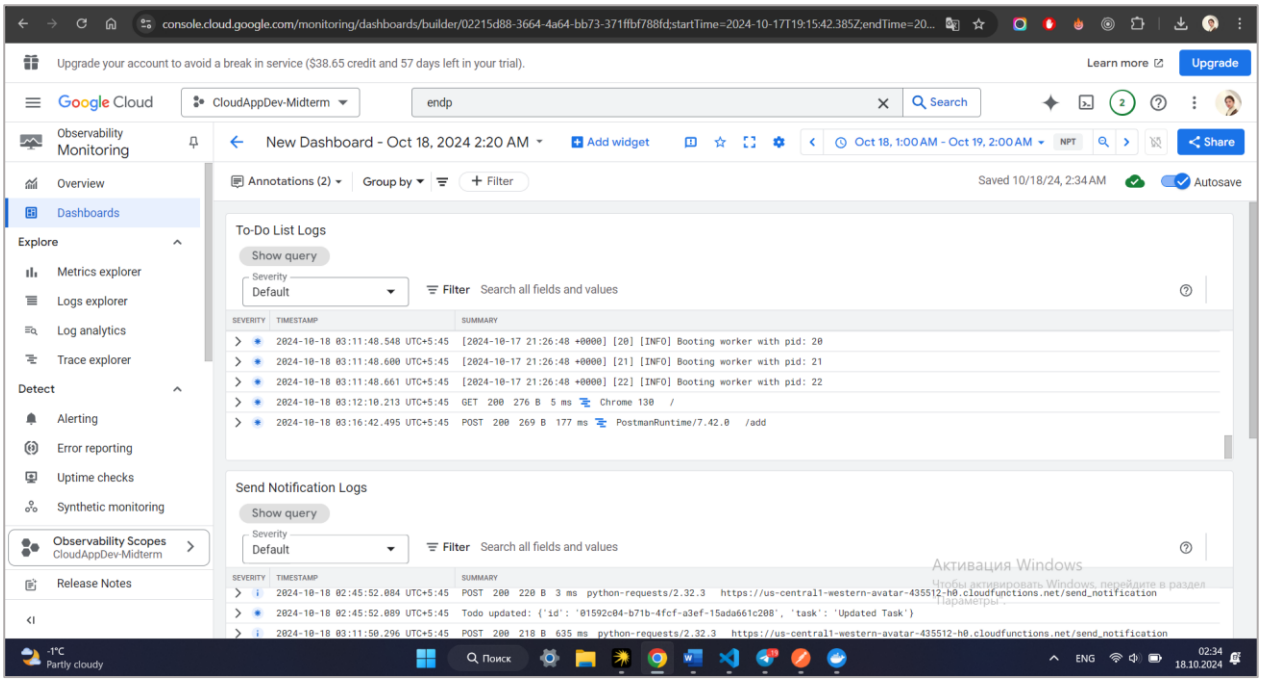
Full results of load testing you can see in the **load_testing_result.html** file in my project zip. Also providing some statistics:

11. Monitoring and Maintenance

1) Google Cloud's monitoring tools:

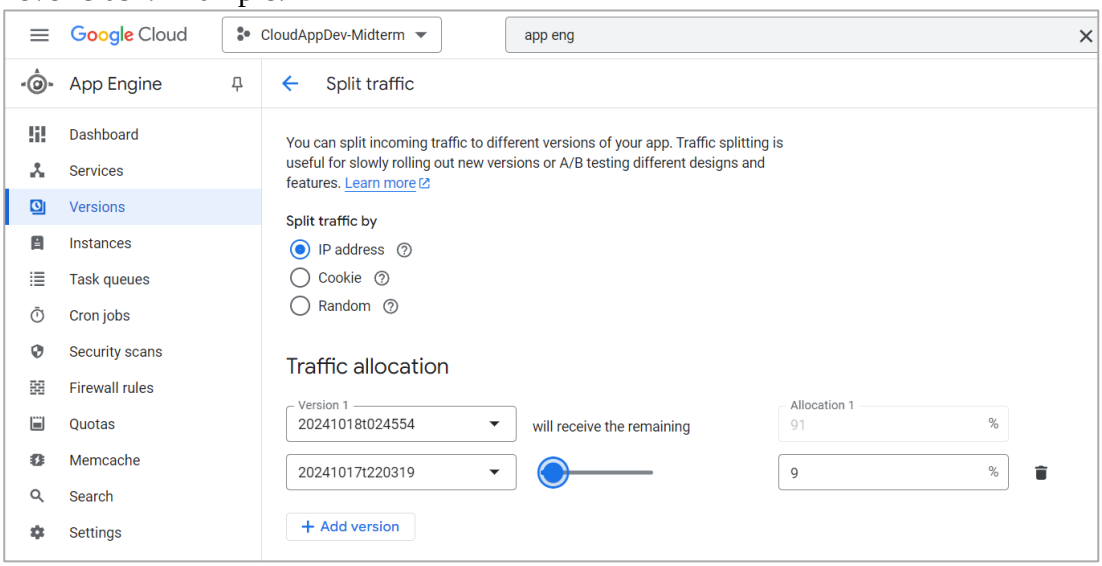
I used Google Cloud Monitoring to create a dashboard with some widgets that show logs of my project and logs of the my cloud functions. By doing this, I have significantly reduced the amount of time to view my app's statistics. Now, by combining all the query logs I need, logs about some errors, warnings, I can log into my created dashboard for monitoring and quickly track errors.

Screenshot of created Monitoring Dashboard:

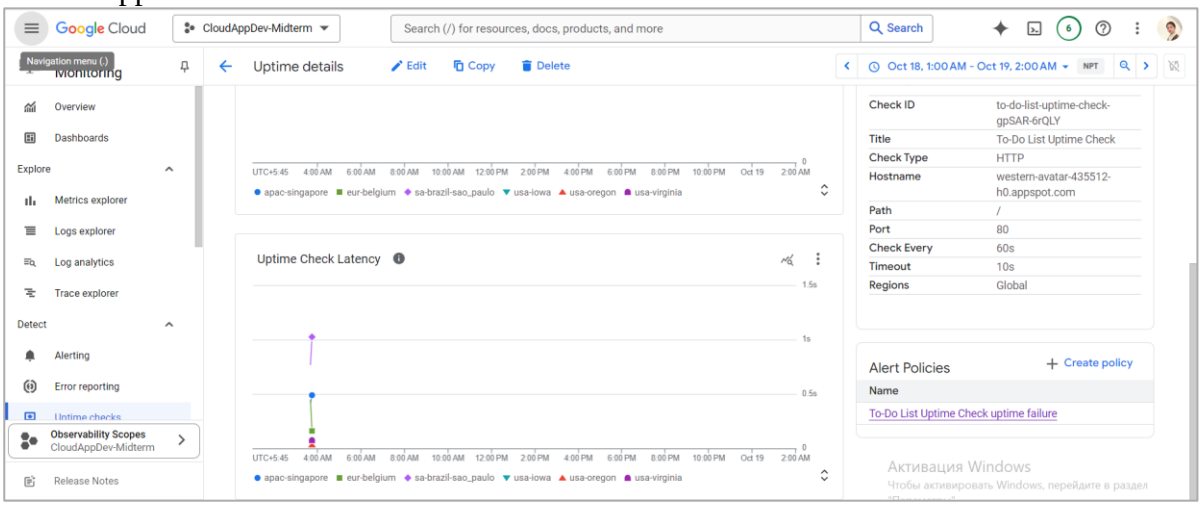


2) Maintenance Practices:

1. The first is **App Engine migrate and split trafic.** Using this tool we can manage our application versions. And you can, for example, deploy new version of app without stopping old version. Also you can distribute traffic between versions of your project like 80% new version, 20% old version. This allows us to create a fault-tolerant system that will almost never crash. Example:



2. Google **Cloud Monitoring Uptime Checks** is a tool that checks the availability of your application from different regions every time interval. It can be configured to check various endpoints, to check for errors, and more. Thus, by regularly checking the performance of your application, you will be able to understand whether it is working correctly and how fast it responds to sent requests. In general, this tool is very useful for building uptime and reliable application.



12. Challenges and Solutions

The problems I have encountered:

1. The first problem is probably Cloud Functions, I suffered with them because from the beginning it was unclear how I could integrate them into my code. But after sitting for a while and thinking about several ways, I created two functions: one for validating client data and the other for sending notifications to logs about adding, deleting and updating todos.

2. The next problem was that when creating the API and implementing authentication, I did not know how to do it, and when searching for solutions in the documentation and the Internet, I was confused and could not do everything right, but in the end, by trial and error, I found right commands and created the API key.
3. The next problem was already the implementation of authentication in my application. She didn't work for me. Then, by adding the require_api_key function and changing openapi.yaml, I managed to add authentication.
4. Another problem was that after the implementation of authentication in GKE, the old version of my application remained. And I had to update my image and re-deploy it in GKE.
5. Well, I also had some minor problems with load testing, when creating tests there, I did not specify their order correctly and it turned out that get todos, add todos were executed without errors, and update, delete todo were with errors from time to time. Then I solved this problem by changing the order to the correct one.

13. Conclusion

Reflect on the achievements of the project, the effectiveness of the technologies used, and suggestions for future improvements. In conclusion, first of all, I would like to say that it was a very interesting experience. This is the first time I've put together a full-fledged project on a cloud platform that included so many technologies.

As a result, I have a To-Do List web application with functionality for viewing, adding, updating and deleting tasks. I have integrated cloud functions for validating client data and sending notifications to logs. I also containerized my application and deployed it in GKE. In addition, I created an API in Cloud Endpoints and added authentication to my application. I conducted tests and made sure that my application is working properly and its ability to withstand loads. In the end, I added tools for monitoring logs and statistics, and also used tools to ensure uptime and reliablity.

In the future, I would probably improve and increase the functionality of the web application itself. I also added more tools for better monitoring. I would have secured the application even more by adding oauth authorization. I would improve the number of cloud functions and add many other things.

14. References

[1] Install the gcloud CLI

https://cloud.google.com/sdk/docs/install

[2] Python 3 Runtime Environment. App Engine

https://cloud.google.com/appengine/docs/standard/python3/runtime

[3] Create a Cloud Run function by using the Google Cloud CLI

https://cloud.google.com/functions/docs/create-deploy-gcloud#functions-clone-sample-repository-python

[4] Docker Documentation

https://www.docker.com/

[5] Google Kubernetes Engine (GKE) Documentation

https://cloud.google.com/kubernetes-engine/docs/concepts/kubernetes-engine-overview

[6] Getting started with Cloud Endpoints for the App Engine flexible environment with ESP

https://cloud.google.com/endpoints/docs/openapi/get-started-app-engine?_gl=1*1xr3iqw*_up*MQ..&gclid=CjwKCAjw68K4BhAuEiwAylp3kiy63Cz_zsM4Em3dBWihFNK0a2or2xcU1JBWtDSnXlg5FrvpDf2ENhoCUCwQAvD_BwE&gclsrc=aw.ds

[7] Manage API keys

https://cloud.google.com/docs/authentication/api-keys?_gl=1*nbbwyw*_ga*MTA3MTc4MjY5MS4xNzIxMDY4MzAy*_ga_WH2QY8WWF5*MTcyOTE3MzUzMi40MS4xLjE3MjkxNzQwMTEuNi4wLjA.#securing

[8] unittes – Unit testing framework

https://docs.python.org/3/library/unittest.html

15. Appendices

I have described my every step in great detail for each of the sections, so there is no need for this.