

**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**

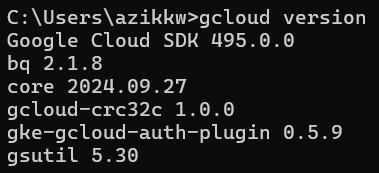
**1) Setup**:

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



1. 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"

1. 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:**

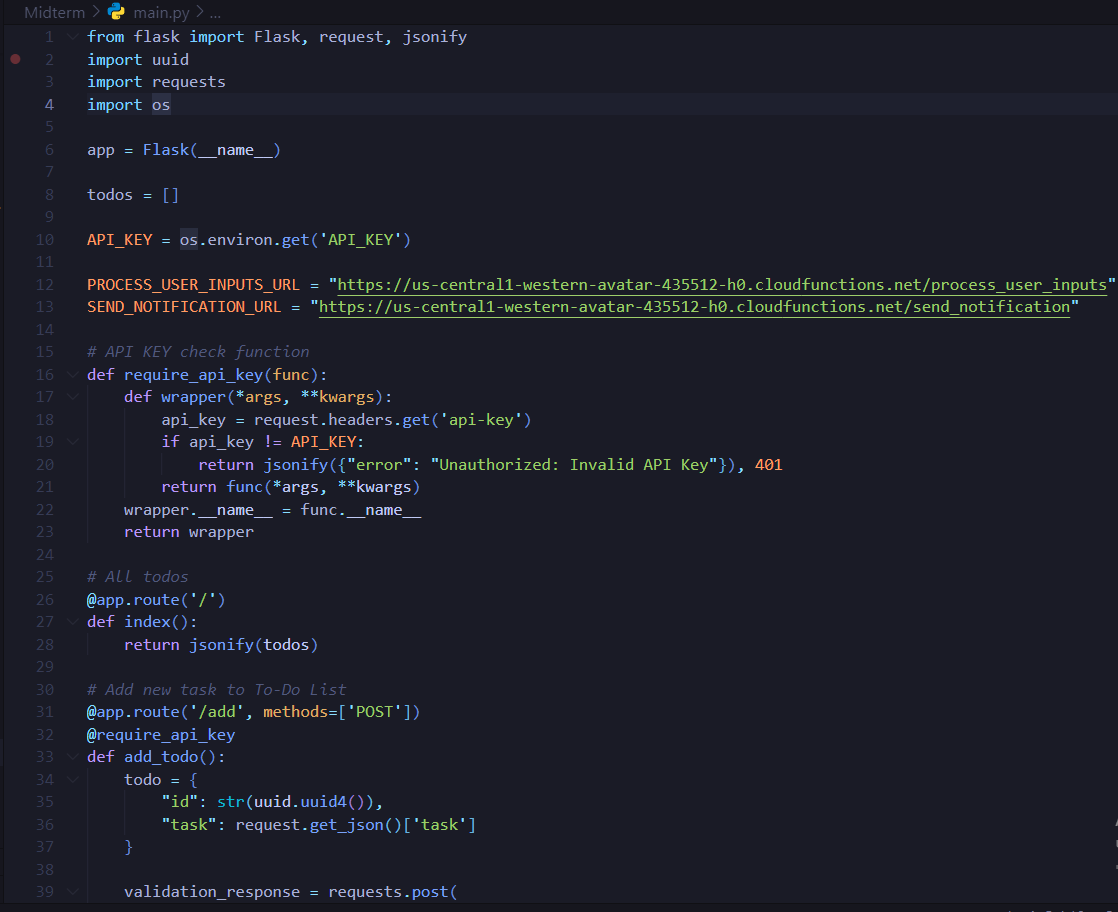
I created a **To-Do List** web application using the **Flask** framework. This web application is hosted on Google Cloud Platform (GCP). The application utilizes various GCP services, including:

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:

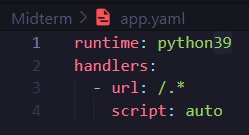
* Creating a task
* Completing the task
* View all tasks

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

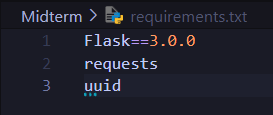
1. Firstly, we need to create our web-application using python **Flask** framework. For example: 

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

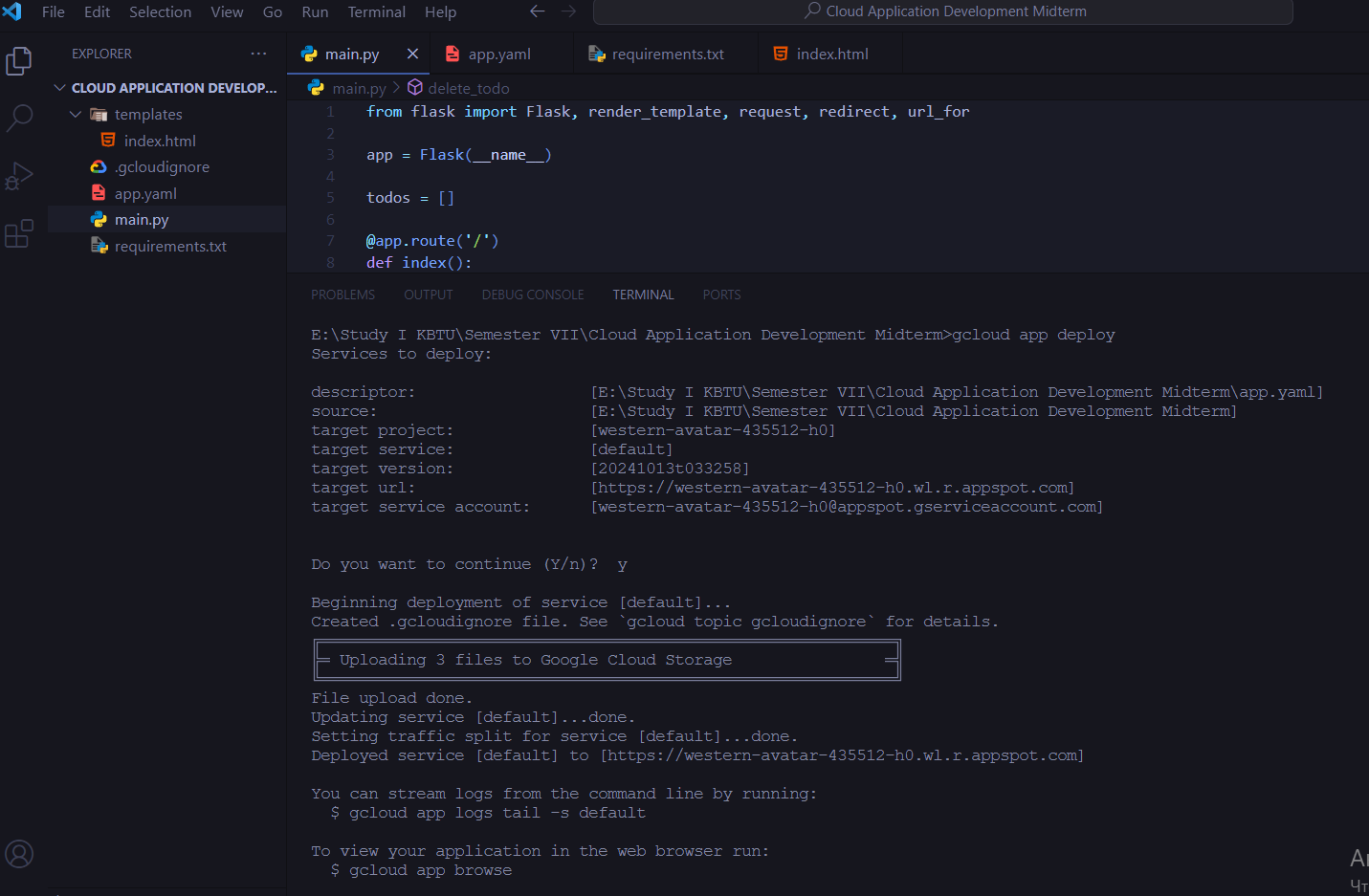
1. At the second step, we should to add **app.yaml** configuration file with necessary instructions for our app deployment. For example:



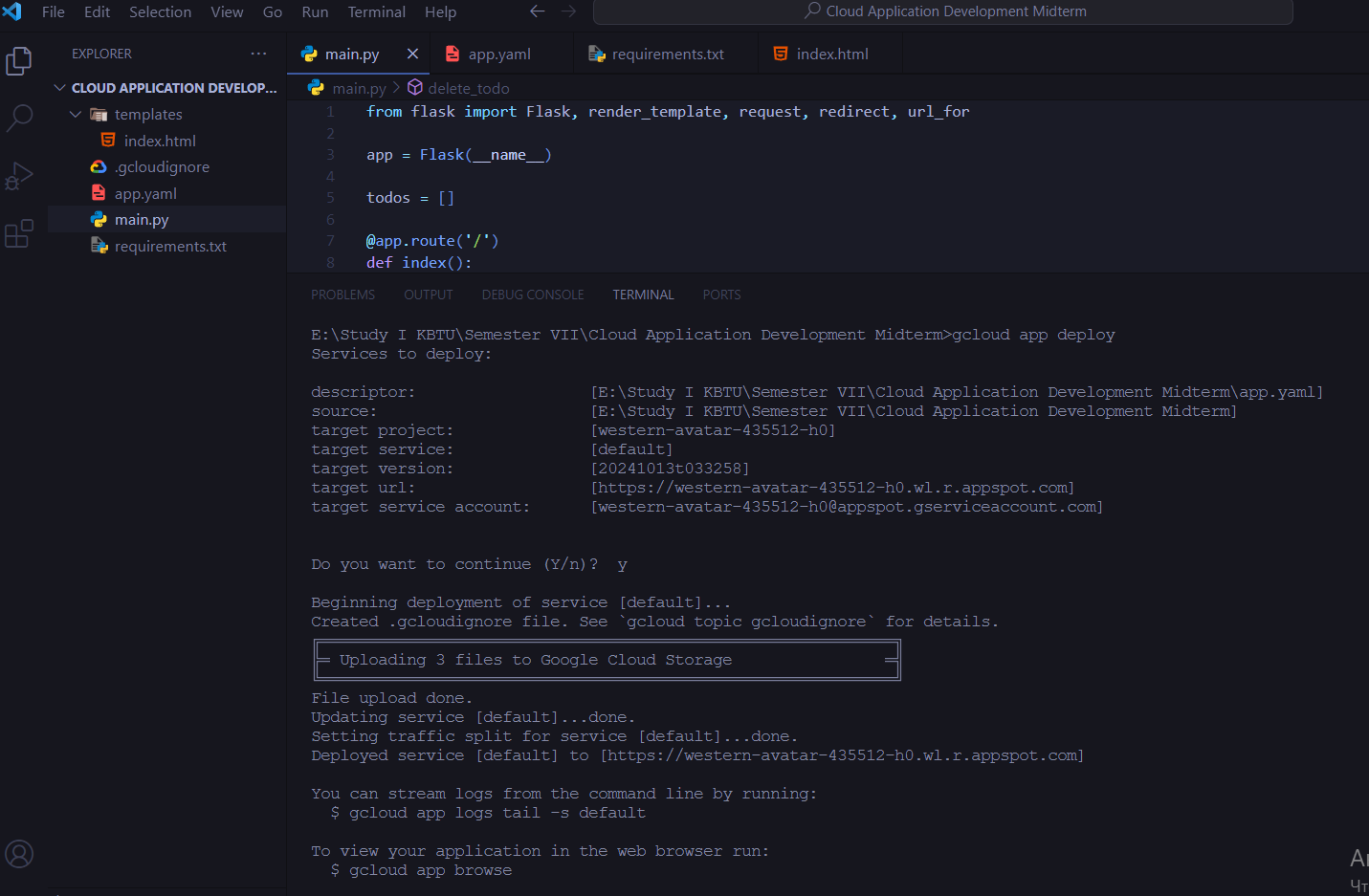
Also, we need to add **requirements.txt** with the necessary python libraries and dependencies for deployment. For example:

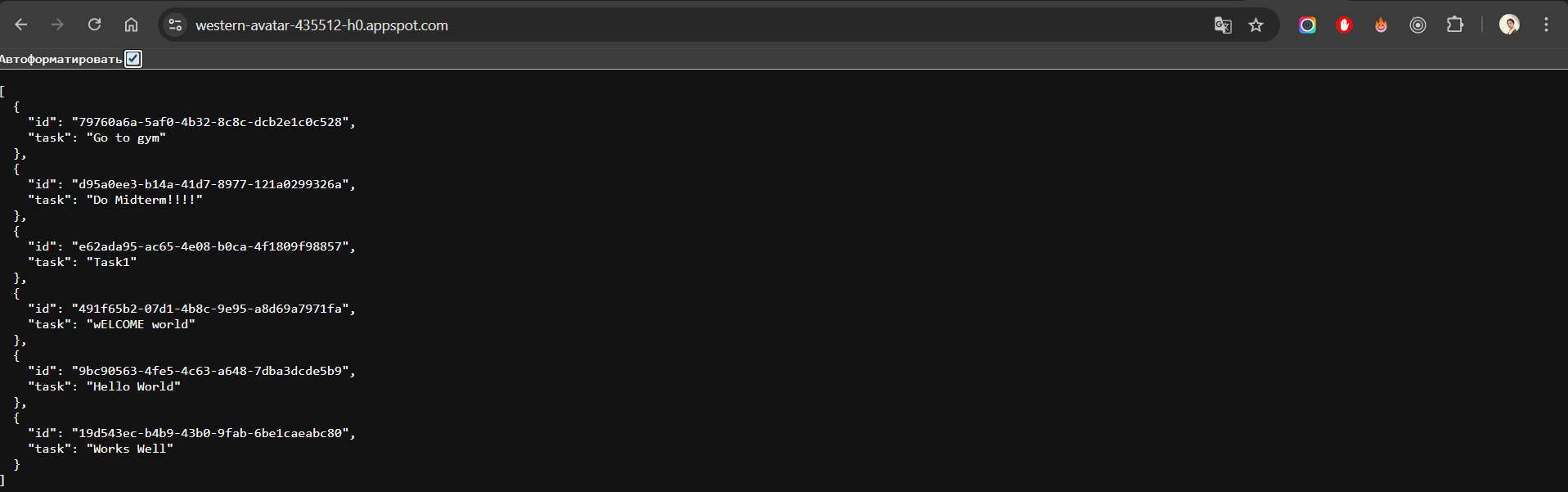
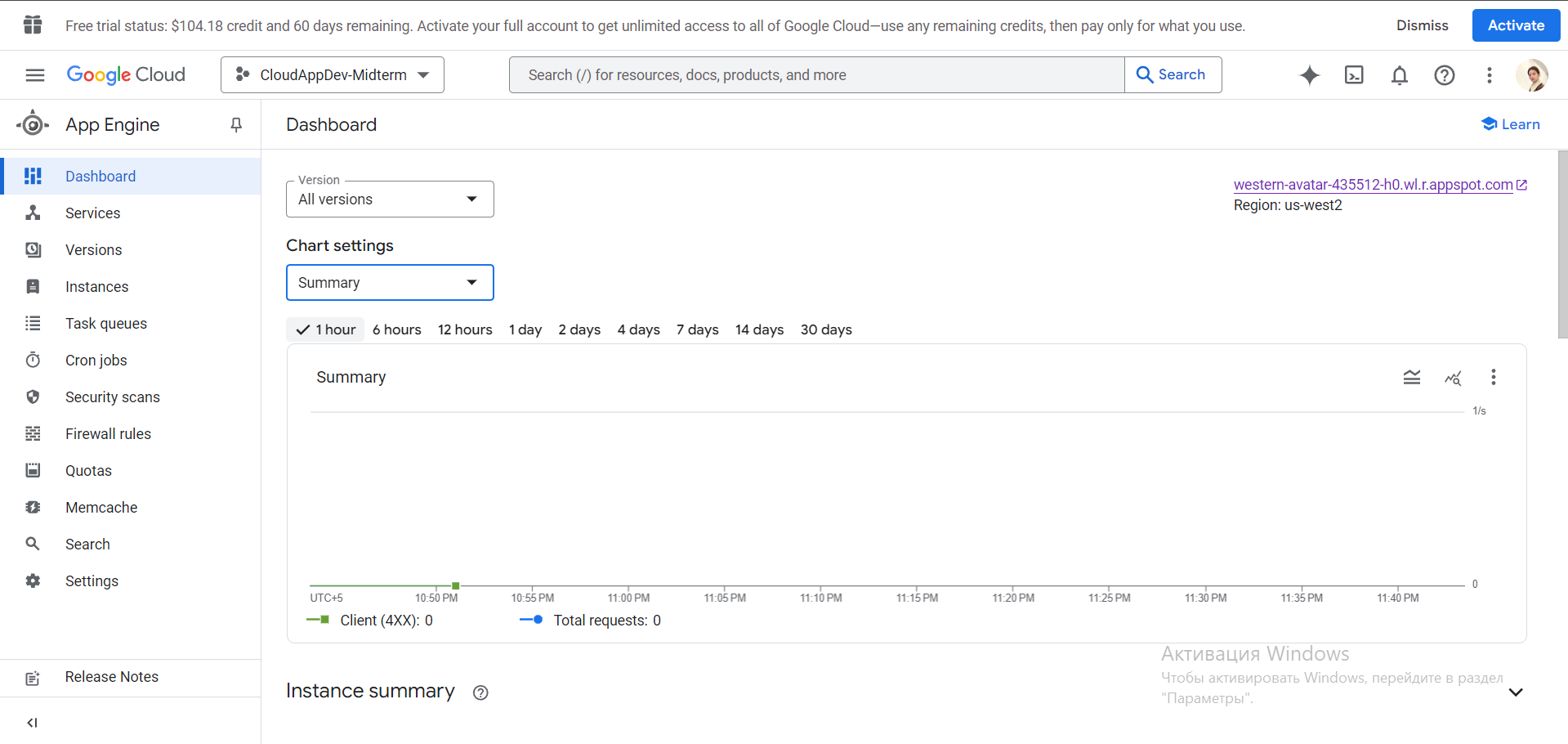


As a result, you will have following project structure:

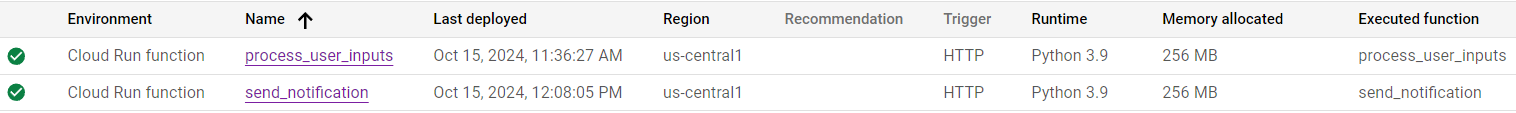


1. And finally, you need to deploy the application. Use gcloud app deploy command to deploy it to Google App Engine:



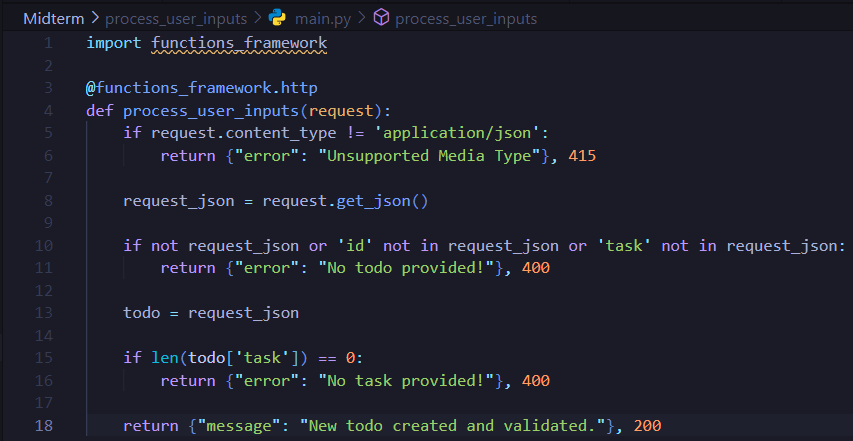


**7. Building with Google Cloud Functions**

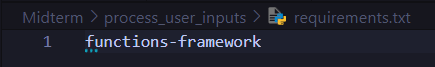
In general, I created two cloud functions for processing user inputs and sending notifications. Let's take a step-by-step look at how I created and deployed them: 

**1) “process\_user\_inputs”** **cloud function**:

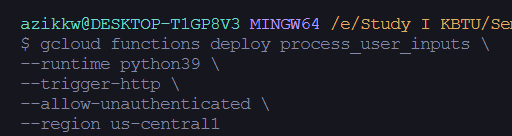
1. Firstly, create function called processing\_user\_inputs. I also used python:



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

1. Add requirements.txt file with dependencies: 
2. 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)

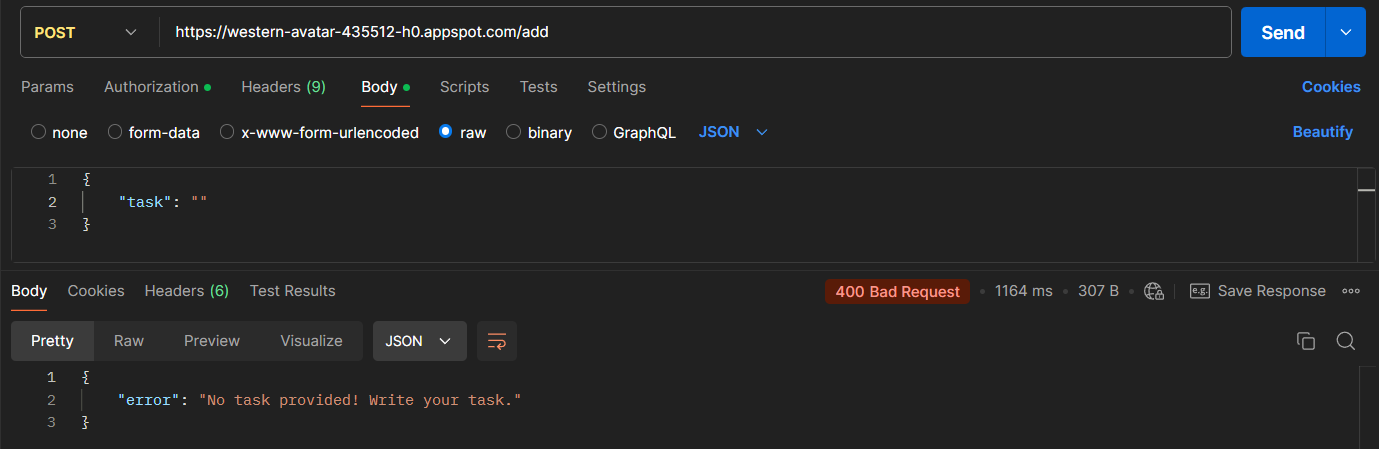


1. Integrate cloud function to the main application: 

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.

1. Example of **process\_user\_inputs**:

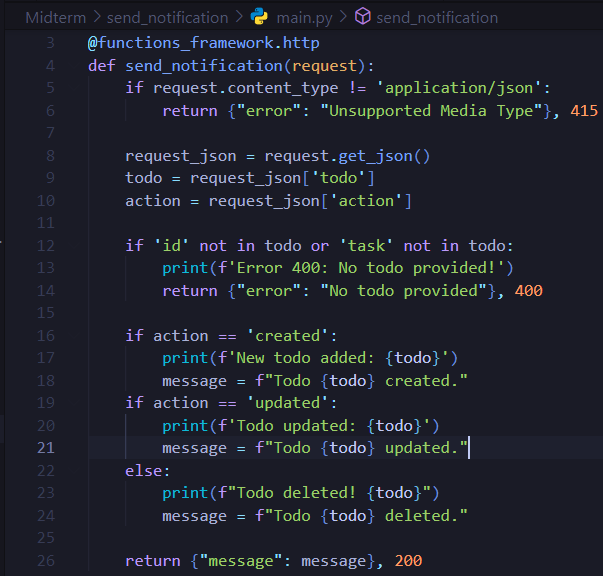
I am trying to add new to-do with empty task:



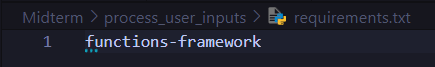
And after validation receive following response from function: ‘No task provided! Write your task’. So, it means that my cloud function work well and process user inputs.

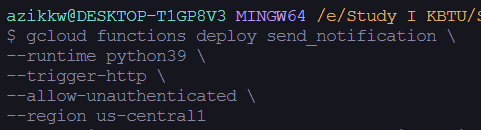
**2. “send\_notification” cloud function:**

1. Also create function but call it send\_notifcation. I also used python:

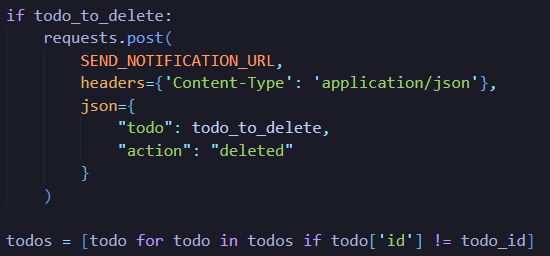
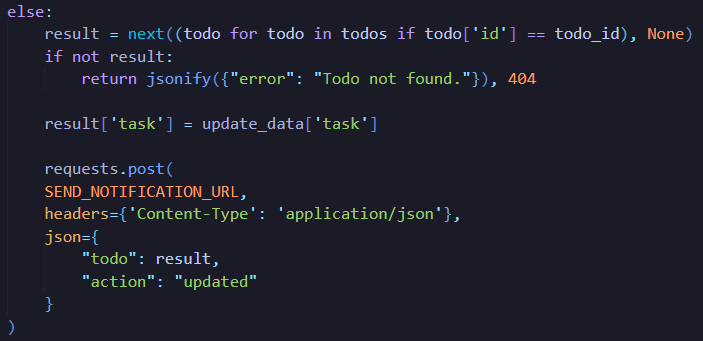


This function sends notification (in logs) that to-do created or deleted according to action.

1. Add requirements.txt file with dependencies: 
2. Deploy cloud function using with configuration like in first function but change name to send\_notification:



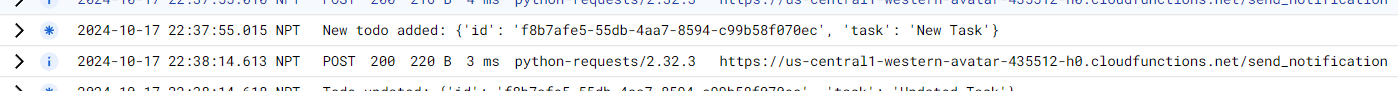
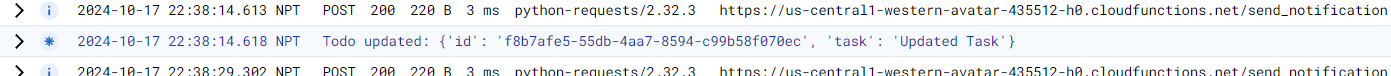
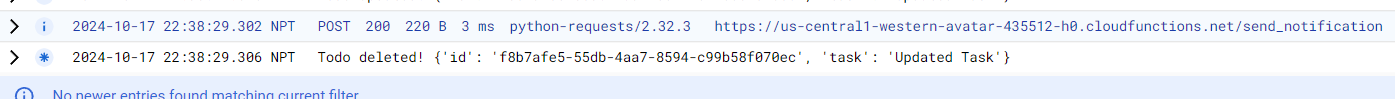
1. Integrate cloud function to the main application:

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.

1. 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.

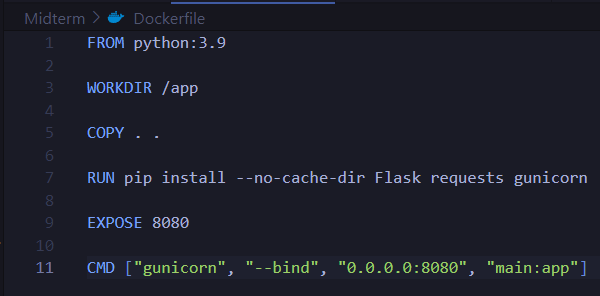
Containerization process:

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.

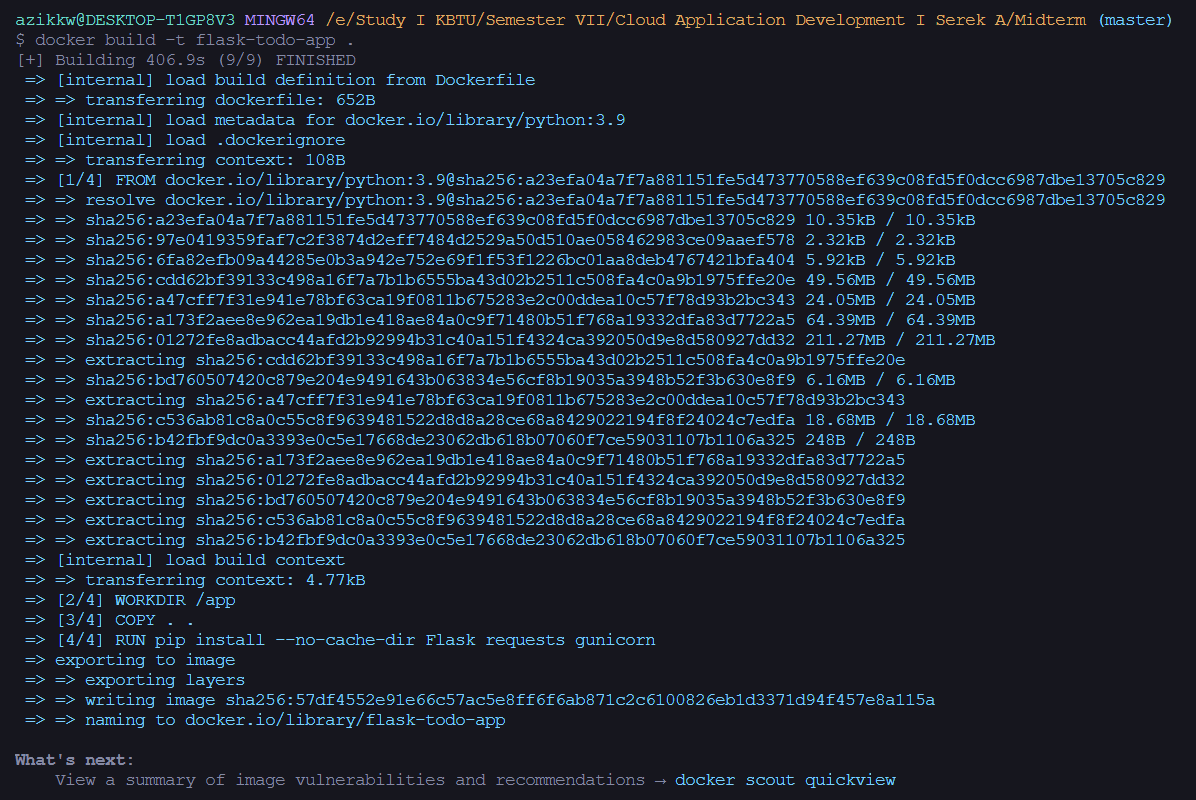
**2) GKE Deployment:**

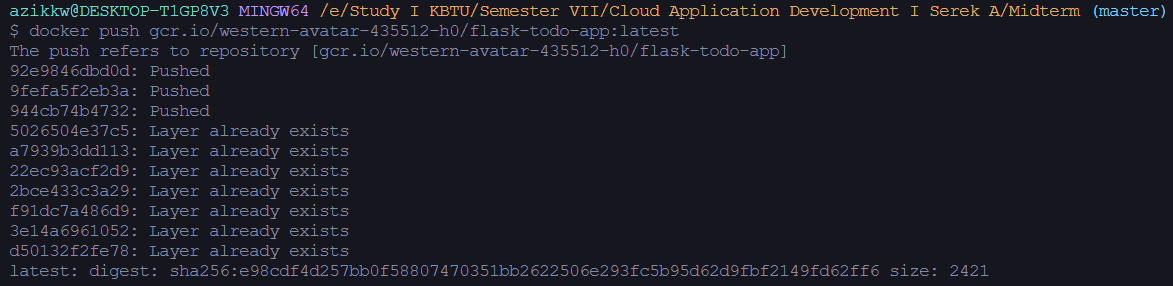
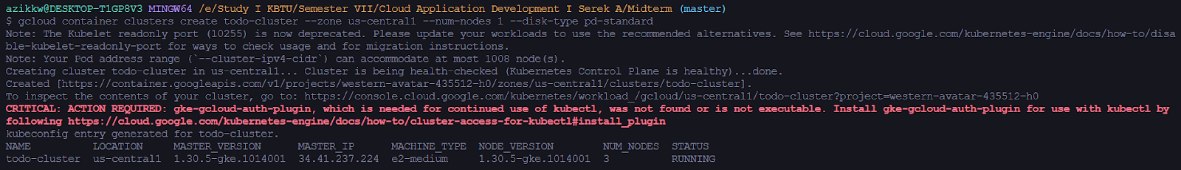
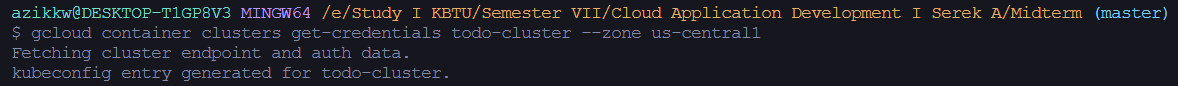
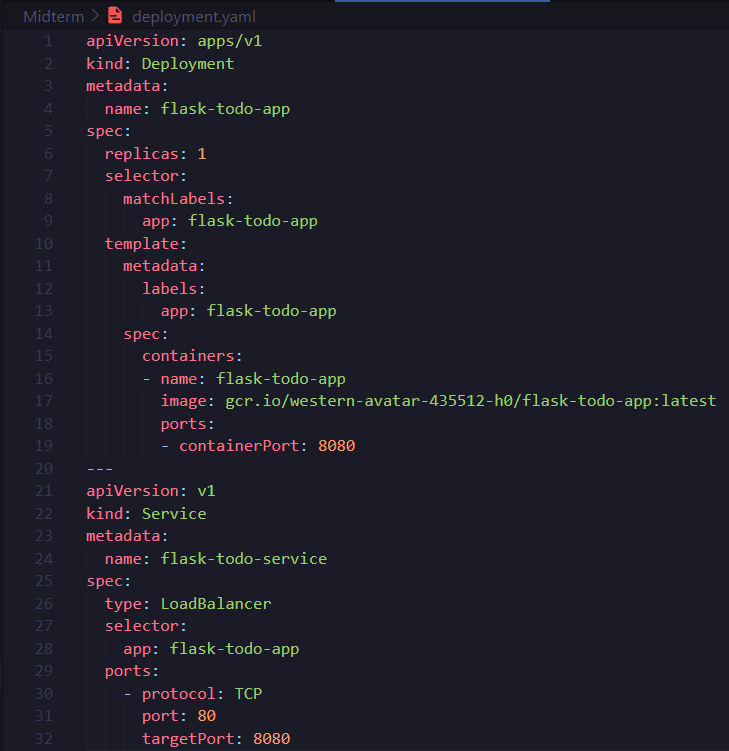
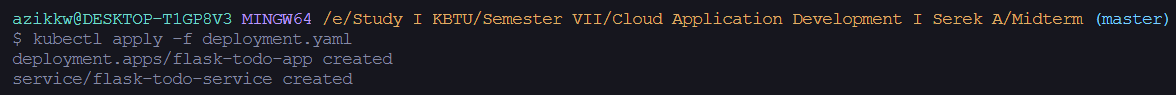
Steps taken to deploy the containerized application on GKE.

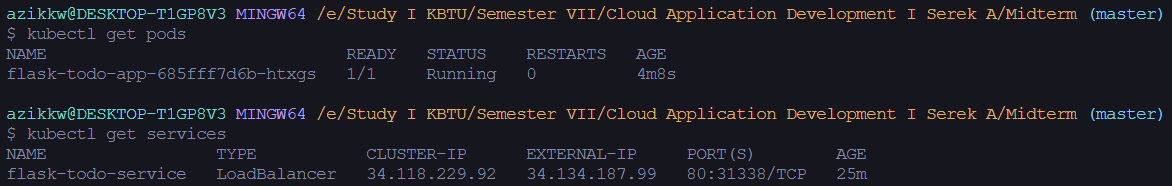
1. Create a Dockerfile with instrucations for creating Docker Image:



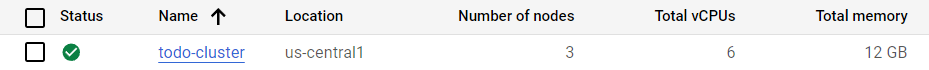
1. Create a Image using docker build:

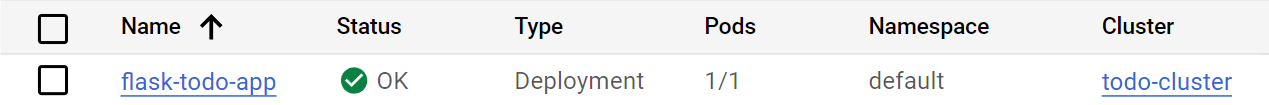


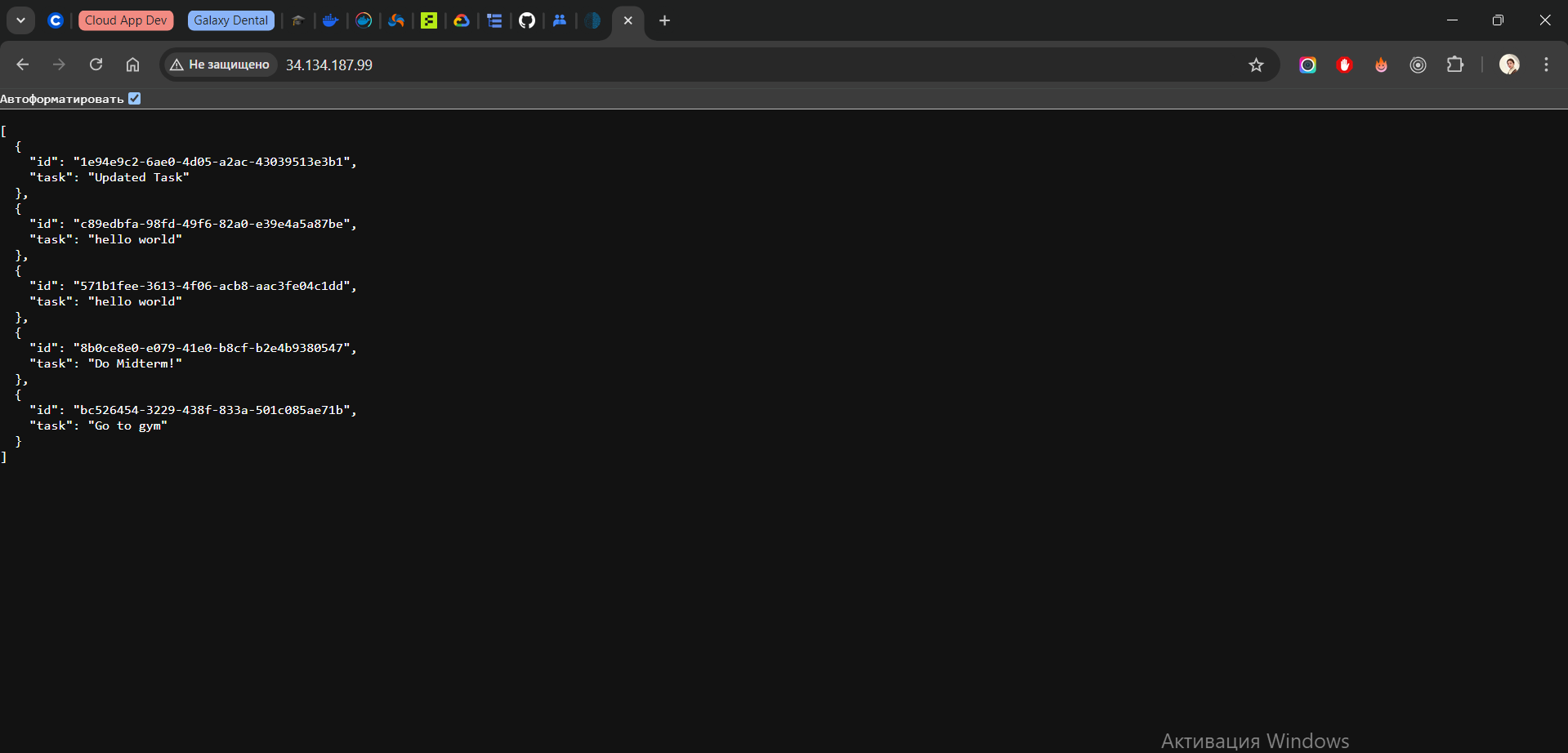
1. Tag Docker Image to send it to Google Container Registry (GCR): 
2. Push Image to GCR: 
3. Next step is to create kluster in Google Kluster Engine (GKE): 
4. Connecting to todo-cluster cluster: 
5. Next, create deployment.yaml with deployment instructions for Kubernetes: 
6. Deploying the application using deployment.yaml: 
7. And finally, check deployed app:



As a result we will have:

1. Cluster named todo-cluster: 
2. And containerized app deployed to the GKE on the <http://34.134.187.99/> IP address:

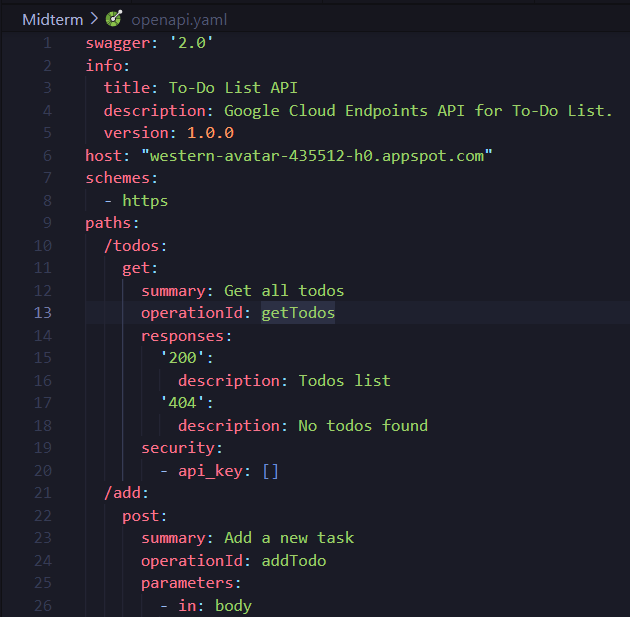




**9. Managing APIs with Google Cloud Endpoints**

**1) Api Setup:**

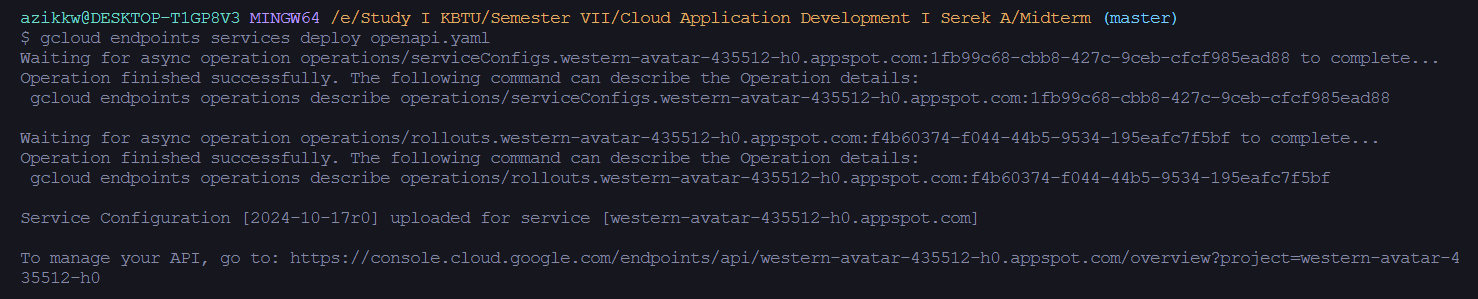
1. The first step is create openapi.yaml configuration file in the project directory:



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.

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

gcloud endpoints services deploy openapi.yaml

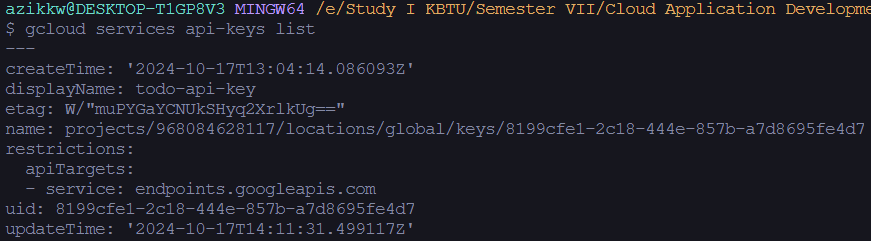


1. After API deployed, you need to deploy your project to App Engine using command:

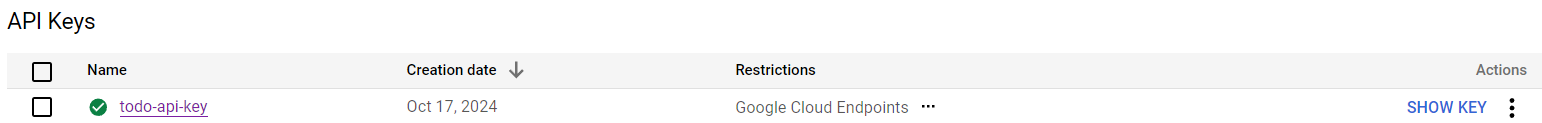
gcloud app deploy

**2) Security and Monitoring:**

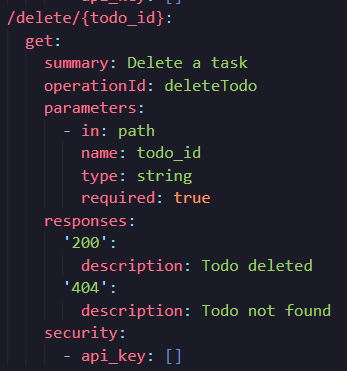
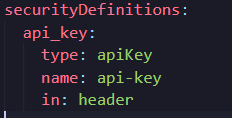
To implement **authentication** to the project we need to do following:

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: 

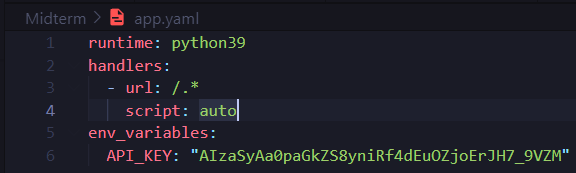
Also you can check it on the Google Cloud Console in “API & Services”.



1. Use **SHOW KEY** button to see API Key.
2. 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.
3. Also is important to update openapi.yaml. Add securityDefinitions and configure security **to each path (Endpoint)**. For example:

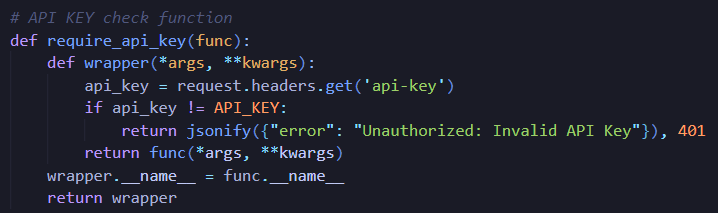
 

1. Once you do this, add your API Key to app.yaml file like that:

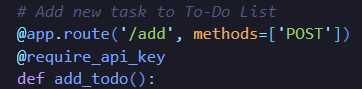


1. The next step is to add api key check in your application:

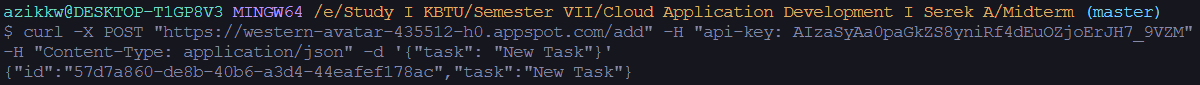




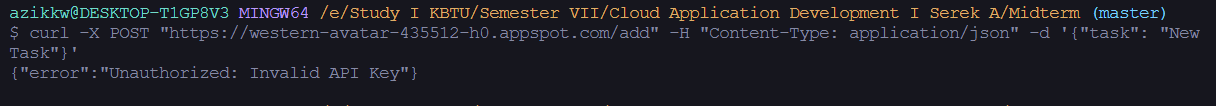
We created require\_api\_key function that checks api key from headers and returns response according to the api key existance.

1. And add this to all your functions (like add\_todo, delete\_todo, etc…): 
2. 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
3. Finally lets test it:

* With API Key in header:

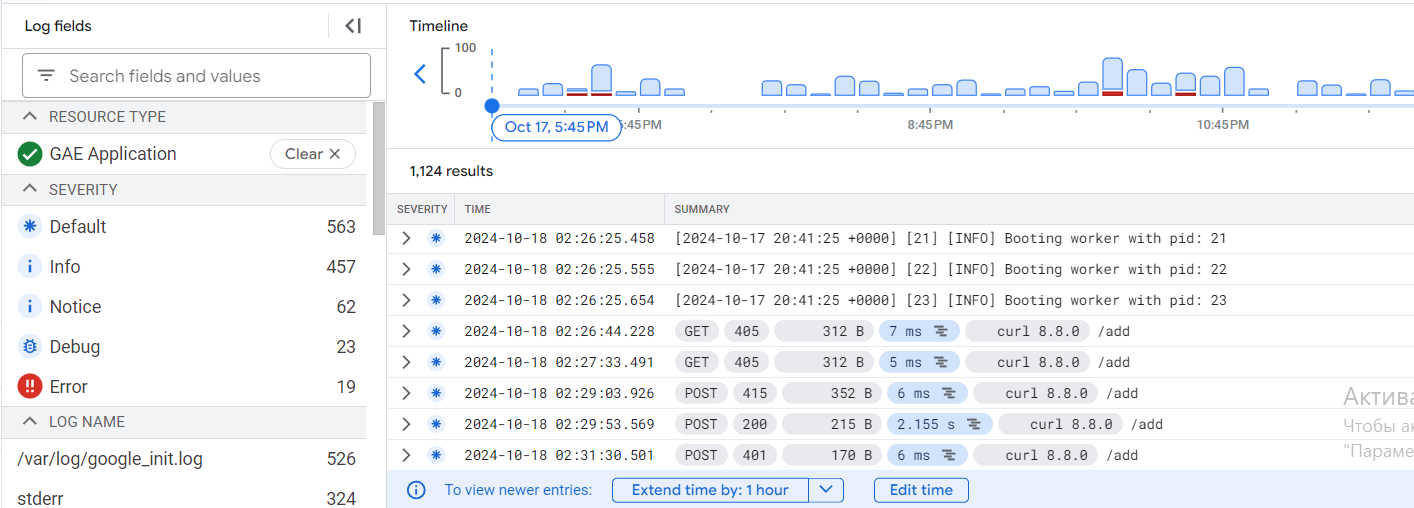


* Without API Key:



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

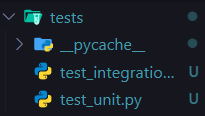
**Monitoring** is already implemented in Google Cloud Platform. Just open your project in the Logs Explorer (Its monitoring tool):



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

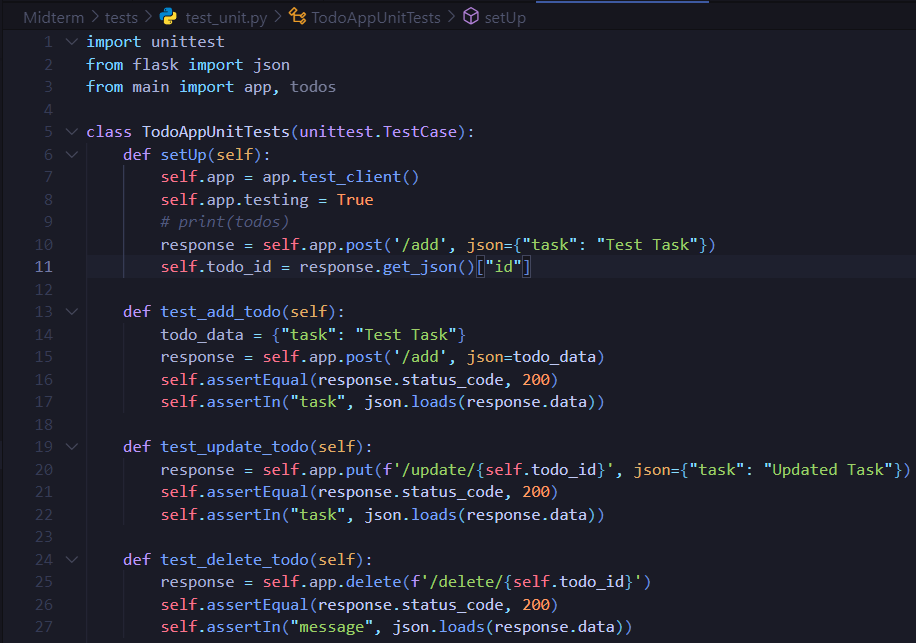
**10. Testing and Quality Assurance**

I created unit, integration and load test.

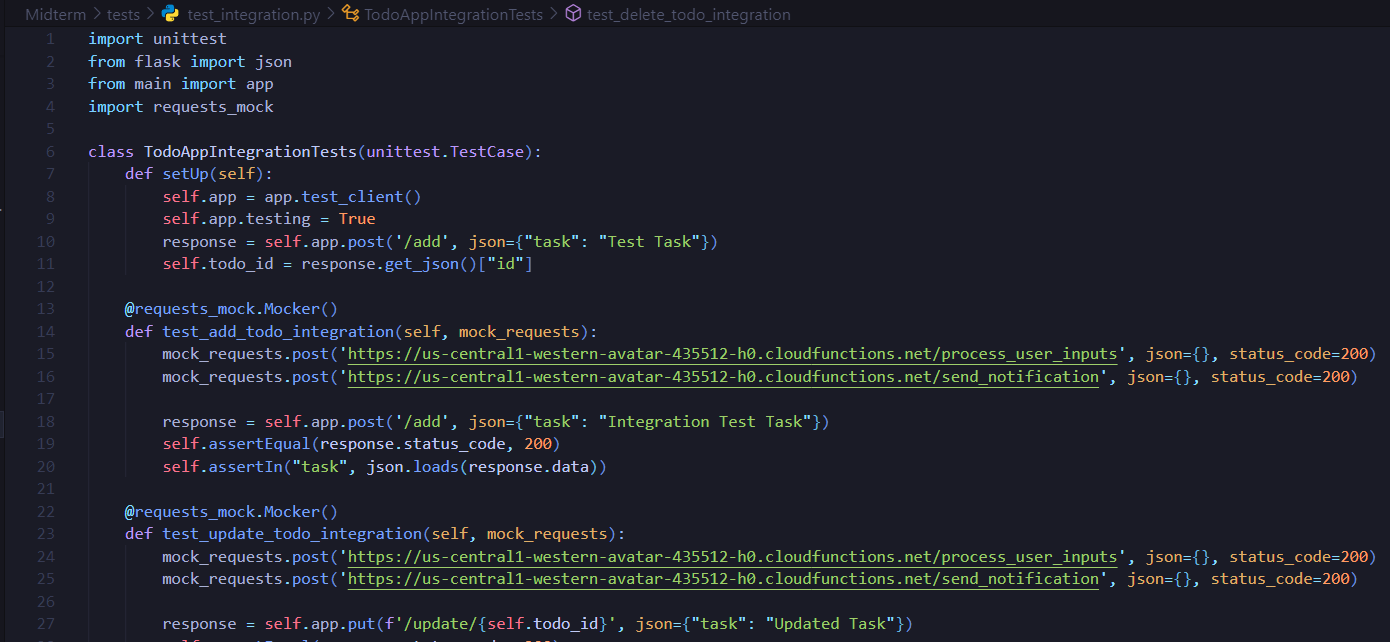
 

Under you will see the results of my unit and integration tests:

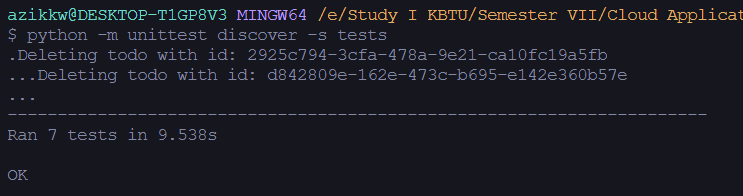
1. Unit testing is a testing of individual pieces of code, like a functions. (Full code in the zip):



1. 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:

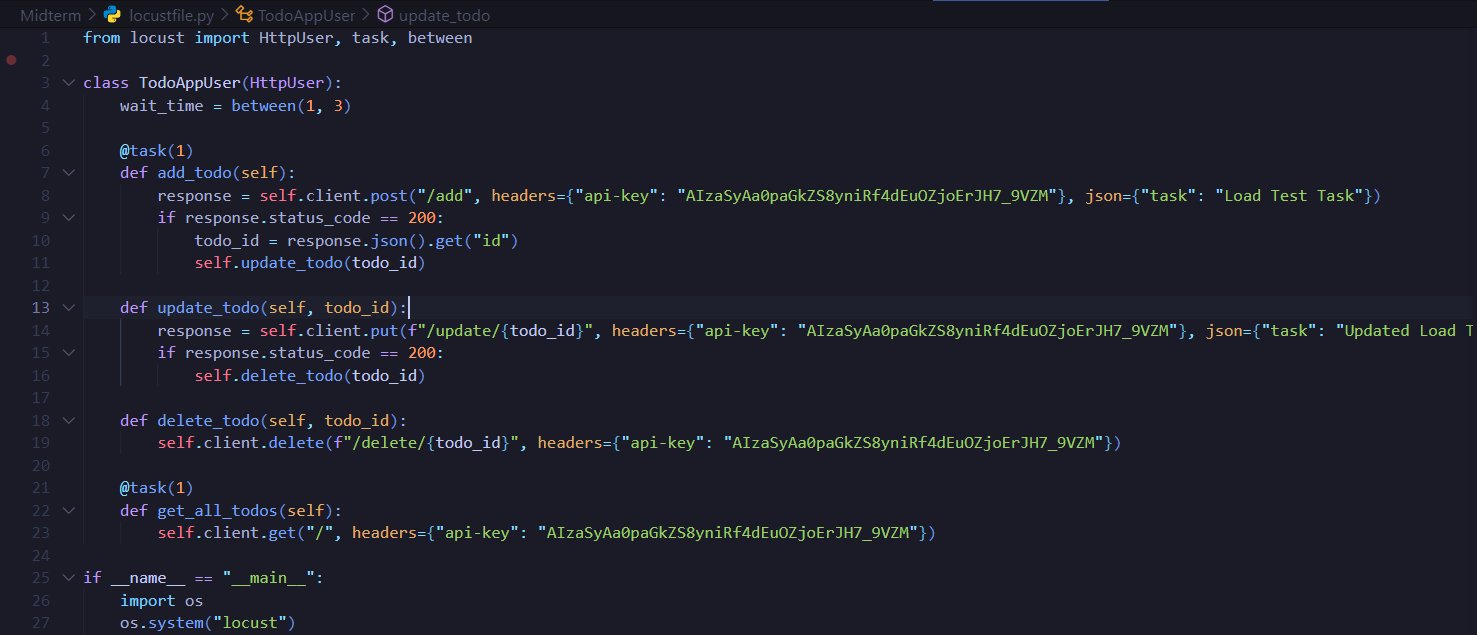


1. Results:

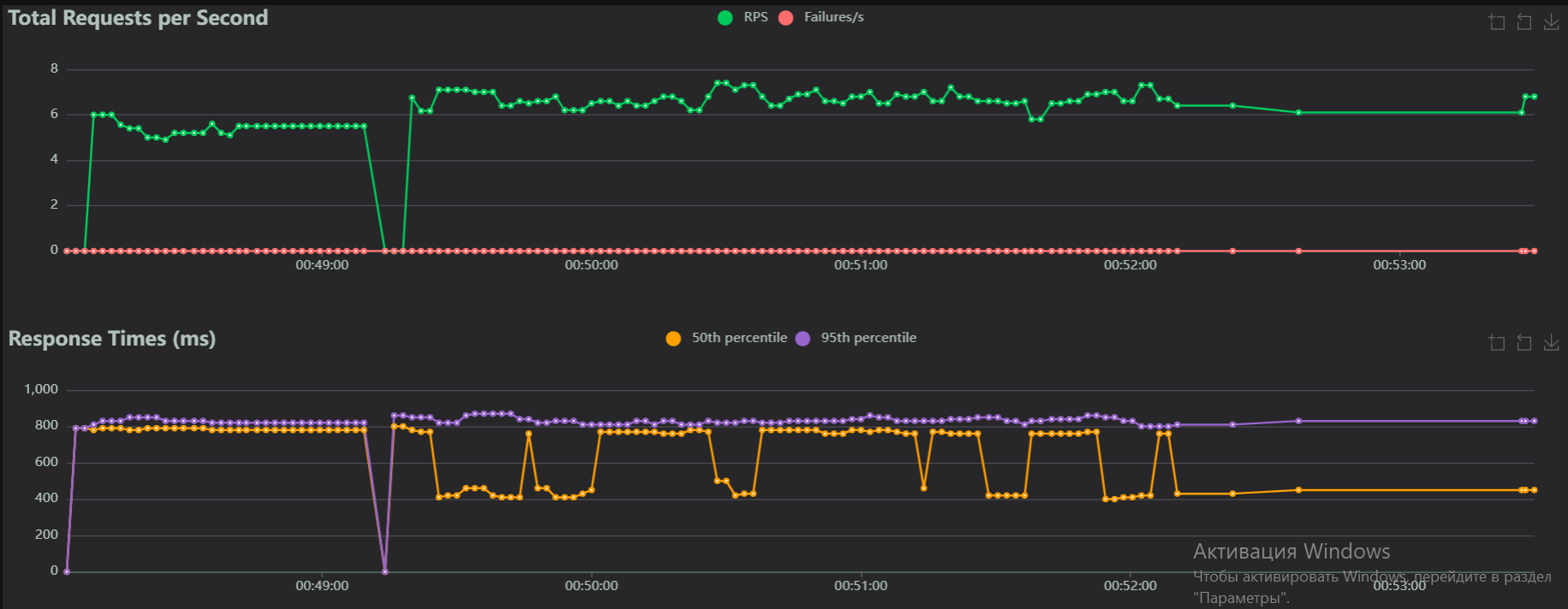
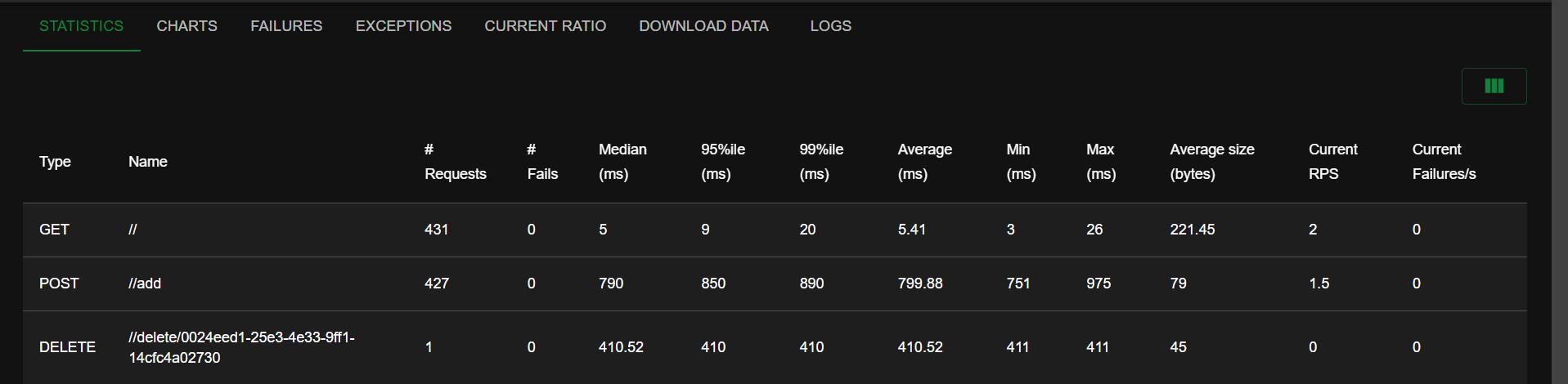


Also load testing was performed. Load testing is testing that is necessary to verify the functionality of the application. Using it, you can find out how many requests the system will be able to process in an n-th amount of time.

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



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

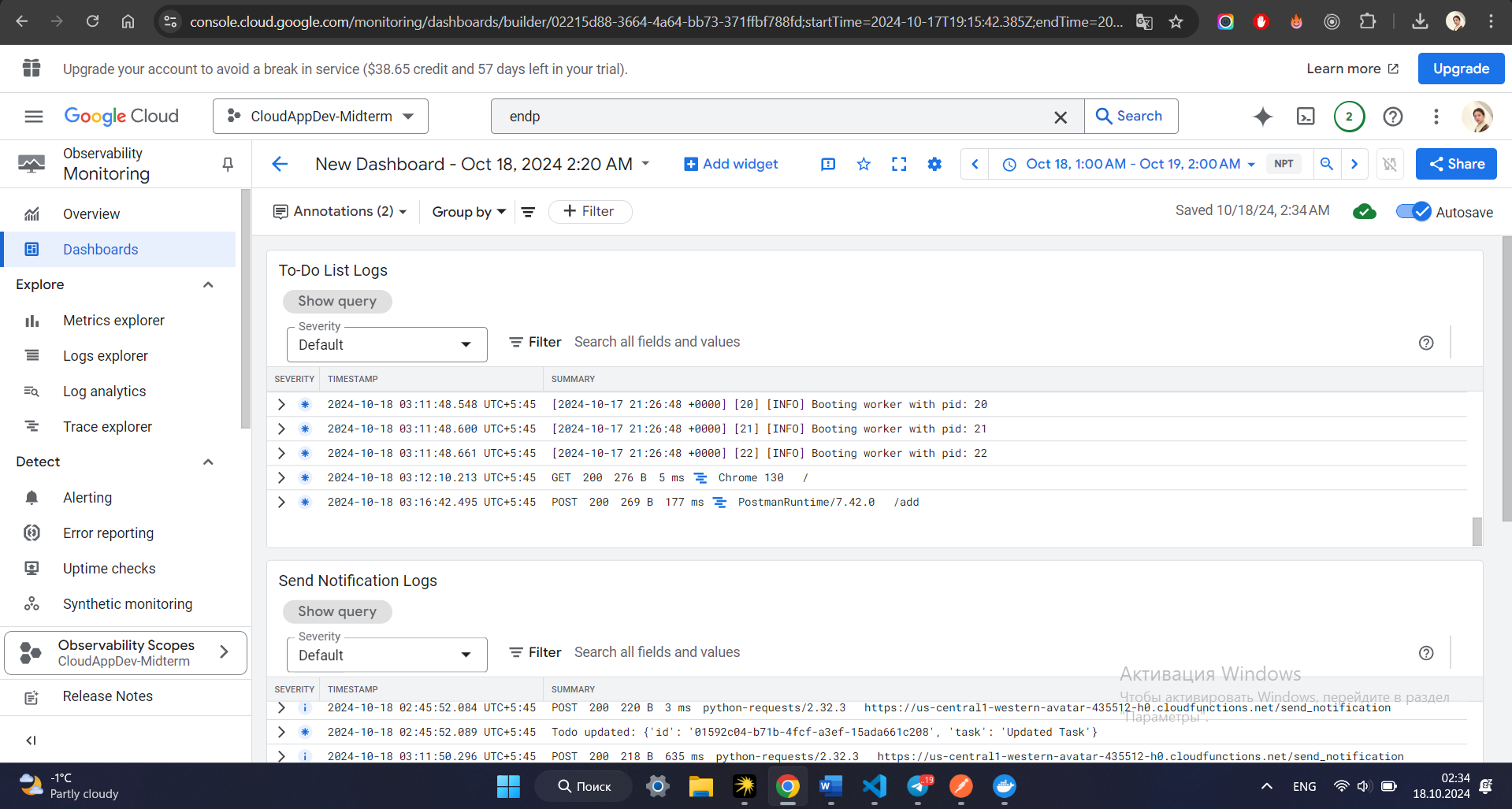


**11. Monitoring and Maintenance**

**1) Google Cloud’s monitoring tools:**

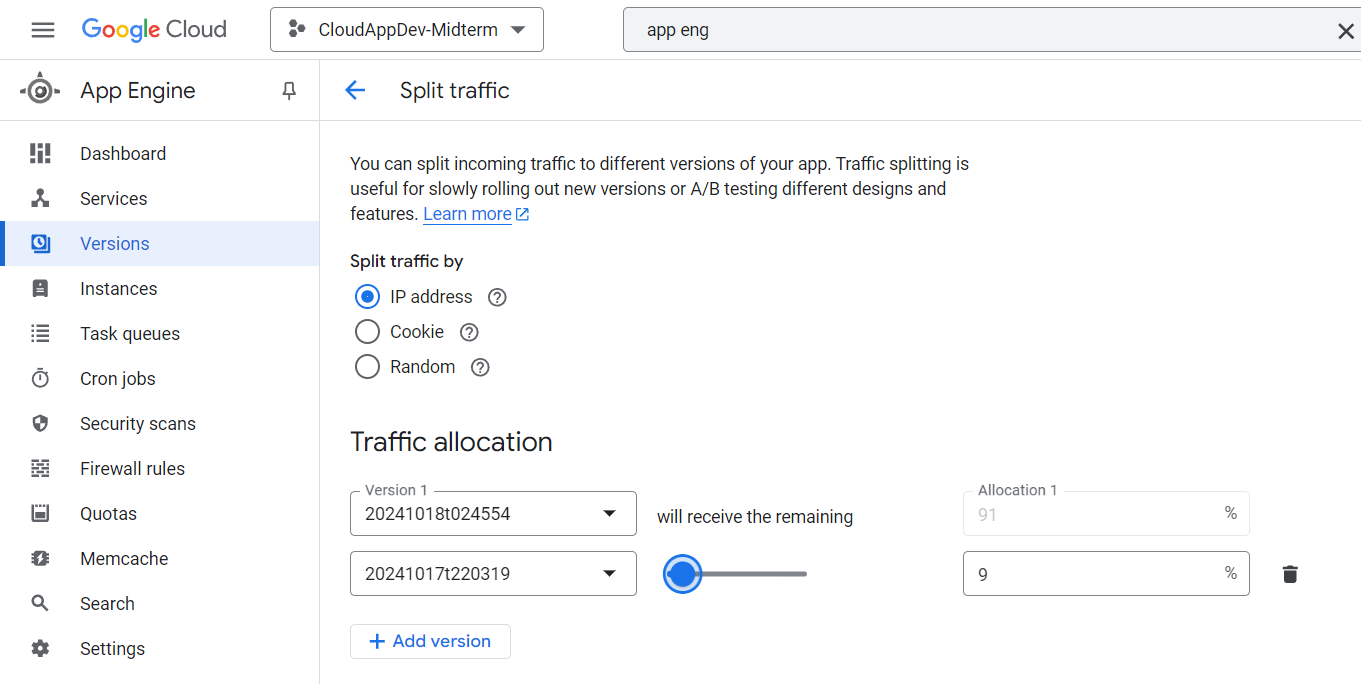
I used Google Cloud Monitoring to create a dashboard with some widgets that show logs of my project and logs of the my cloud functions. By doing this, I have significantly reduced the amount of time to view my app's statistics. Now, by combining all the query logs I need, logs about some errors, warnings,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

Screenshot of created Monitoring Dashboard:

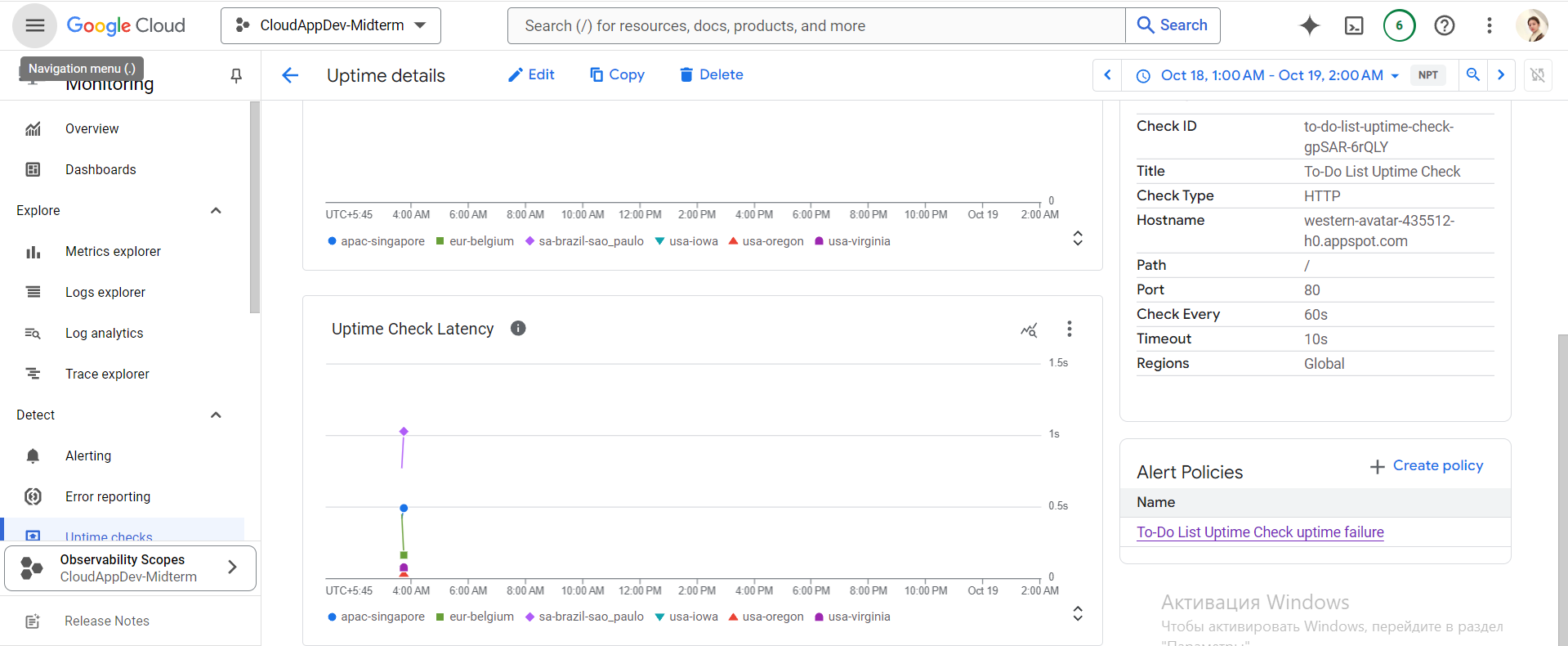


**2) Maintenance Practices:**

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:



1. 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](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](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.