



Cloud application

Midterm Project: Deploying a Scalable Web Application on Google Cloud Platform

Prepared by:

Beldeubaeva Togzhan 21B030789

Almaty 2024

1. Executive Summary

The project goals were to develop and deploy a scalable to-do list web application, utilizing Google Cloud Platform (GCP) services.

The application was built using the Flask framework and deployed on google app engine. Key features included serverless functions using google cloud functions, containerization with docker for seamless deployment on google Kubernetes engine (GKE), and API management via google cloud endpoints.

The outcomes achieved, the successful development and deployment of a scalable to-do list application, capable of handling requests using google cloud services.

2. Introduction

Google cloud platform (GCP) is a cloud platform offering data solutions, scalable infrastructure, and tools for analysis and machine learning. GCP is important for deploying web applications, as it provides automatic scaling for handling workloads in the PaaS model. Choose the cloud infrastructure for my application because it offers high availability and modern data analysis capabilities, which allows to maintain stable and efficient operation of the application.

3. Project Objectives

1. Developing a responsive to-do list web application using Flask.
2. Deploying the application on google app engine to ensure automatic scaling and high availability based on traffic demand.
3. Implementing serverless architecture by using google cloud functions for processing user data.
4. Containerizing the application using Docker for seamless deployment on google Kubernetes engine (GKE).
5. Setting up and managing APIs with google cloud endpoints.
6. Conduct comprehensive testing.

4. Google Cloud Platform Overview

Several services that google cloud platform offers for my application have used:

1. Google cloud SDK and cloud shell – command-line tools and libraries for google cloud.
2. Google app engine – is a platform-based service (PaaS) that allows develop and deploy web application without having to manage the infrastructure.
3. Google cloud functions – offers a serverless service to create functions that automatically executed in response (HTTP requests or data changes).
4. Google cloud endpoints – deployment and development management for APIs on google cloud.
5. Google Kubernetes engine (GKE) – managed environment for running containerized apps.

5. Google Cloud SDK and Cloud Shell

First, the SDK was downloaded and installed on the local machine following the instructions for the Windows platform on the official website.

After installation, configuration was performed using the command `gcloud init`. This included authenticating the SDK with the google cloud account. Then, the default project was selected using the command `gcloud config set project [projectID]`. The configuration included setting the default region and zone for working with the application through commands `gcloud config set compute/region [regionName]` and `gcloud config set compute/zone[zoneName]`.

To check that the configuration is set up correctly:

```
C:\Users\HP\AppData\Local\Google\Cloud SDK>gcloud config list
[accessibility]
screen_reader = False
[compute]
region = asia-central1
zone = asia-central1-a
[core]
account = beldeubaevatogzhan17@gmail.com
disable_usage_reporting = True
project = cloudapp-project-123
```

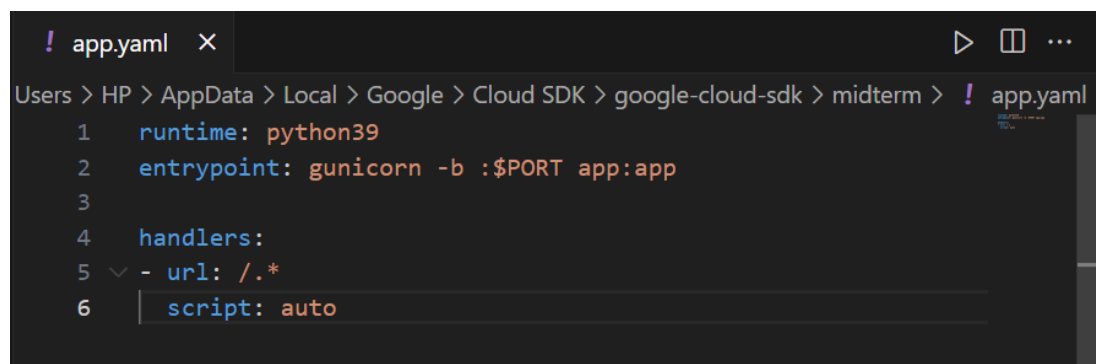
Figure1

Cloud shell was used to manage google cloud resources without the need to set up local tools. The deployment of the to-do list application was simplified by using google shell, where the application code was uploaded to google app engine and google Kubernetes engine (GKE).

6. Google App Engine

To-do list application is the application built using Flask, a lightweight web framework for Python. The user can add, delete and view their tasks.

To deploy an application on google app engine, it is necessary to create an app.yaml file. It is used to configure the application's settings in google app engine, specifying how the application will run and what resources it will use. App.yaml file:

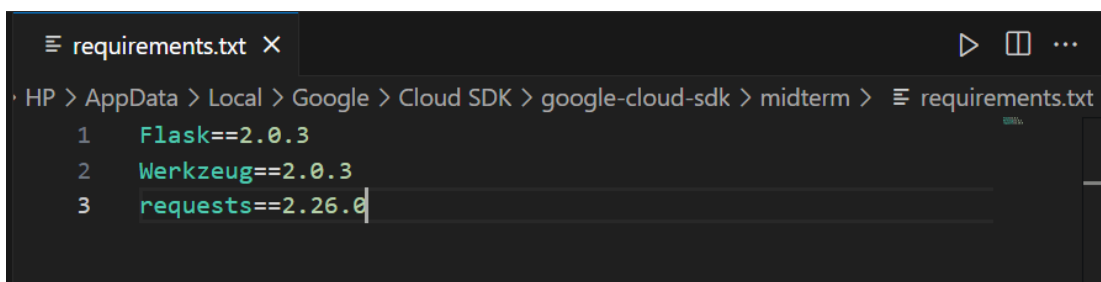


```
! app.yaml
1 runtime: python39
2 entrypoint: gunicorn -b :$PORT app:app
3
4 handlers:
5 - url: /.*
6   script: auto
```

Figure2

The file configures the to-do list application to use python3.9, run the gunicorn server, and handle requests through autoconfiguration.

The requirements.txt file is needed to specify the dependencies of python applications. It contains a list of libraries, and their versions required for the application to work correctly. Requirements.txt:



```
requirements.txt
1 Flask==2.0.3
2 Werkzeug==2.0.3
3 requests==2.26.0
```

Figure3

To deploy, used the command gcloud app deploy.

```
C:\Users\HP\AppData\Local\Google\Cloud SDK\google-cloud-sdk\midterm>gcloud app deploy
ERROR: (gcloud.app.deploy) Permissions error fetching application [apps/cloudapp-project-123]. Please make sure that you have permission to view applications on the project and that beldeubaevatogzhan17@gmail.com has the App Engine Deployer (roles/appengine.deployer) role.

C:\Users\HP\AppData\Local\Google\Cloud SDK\google-cloud-sdk\midterm>gcloud app create
ERROR: (gcloud.app.create) [beldeubaevatogzhan17@gmail.com] does not have permission to access apps instance [cloudapp-project-123] (or it may not exist): Read access to project 'cloudapp-project-123' was denied: please check billing account associated and retry. This command is authenticated as beldeubaevatogzhan17@gmail.com which is the active account specified by the [core/account] property
```

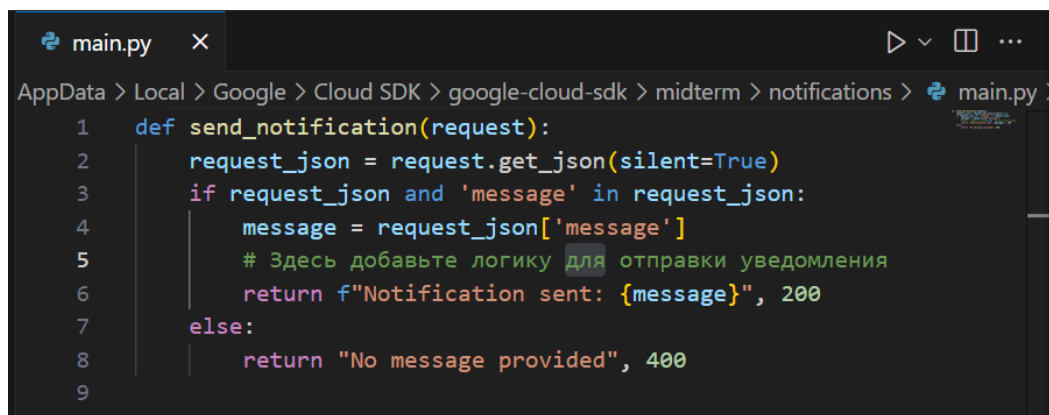
Figure4

There is no billing account, the system returns an error. If a billing account is available, the application will be deployed and use the command `gcloud app browse` to access it.

7. Building with Google Cloud Functions

Google cloud functions are serverless functions that allow to run code in response without managing servers. In to-do list application, cloud functions used to handle tasks: adding, updating, and deleting tasks, as well as sending notifications to users, improving scalability and simplifying the infrastructure.

Created a `main.py` file with the `send_notification` function. This function is a serverless function that handles HTTP requests and sends notifications. `Main.py`:



```
main.py
AppData > Local > Google > Cloud SDK > google-cloud-sdk > midterm > notifications > main.py
1 def send_notification(request):
2     request_json = request.get_json(silent=True)
3     if request_json and 'message' in request_json:
4         message = request_json['message']
5         # Здесь добавьте логику для отправки уведомления
6         return f"Notification sent: {message}", 200
7     else:
8         return "No message provided", 400
9
```

Figure5

To deploy, used the command `gcloud functions deploy send_notification --runtime python310 --trigger-http --allow-unauthenticated`.

```
C:\Users\HP\AppData\Local\Google\Cloud SDK\google-cloud-sdk\midterm>gcloud functions deploy send_notification --runtime python39 --trigger-http --allow-unauthenticated
ERROR: (gcloud.functions.deploy) ResponseError: status=[403], code=[0k], message=[Read access to project 'cloudapp-project-123' was denied: please check billing account associated and retry]
```

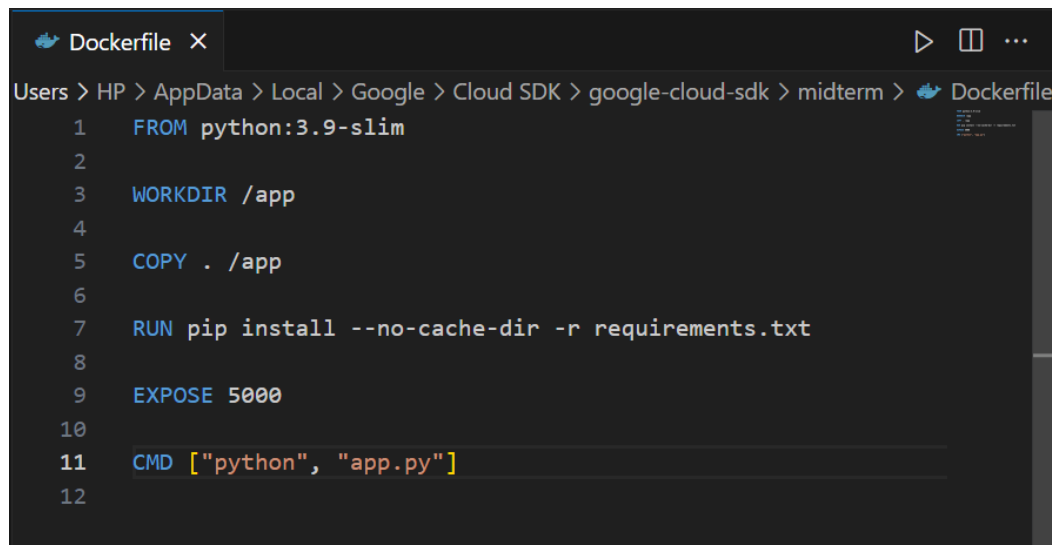
Figure6

There is no billing account, the system returns an error. If a billing account is available, the deployment will succeed, and the message indicating that the function is deployed and active (status: active), along with the url where the cloud function is accessible.

8. Containerizing Applications

For containerizing the application, used docker. It is needed to package the application and its dependencies into a single container, which can be run in any environment.

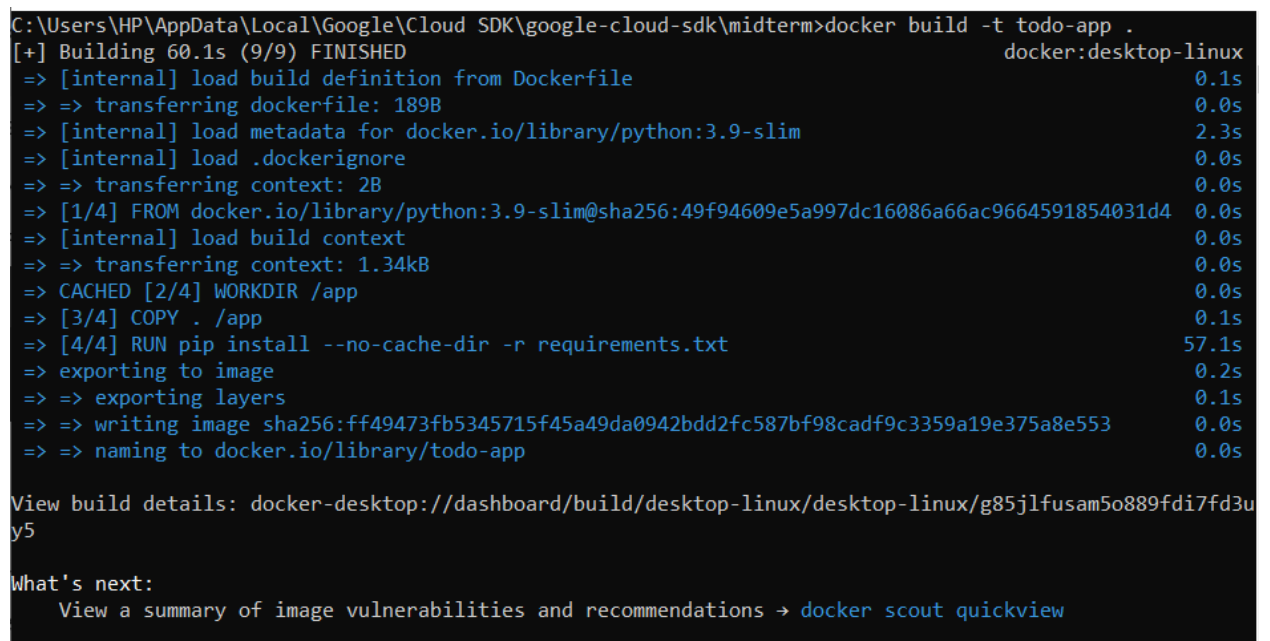
To containerize the application, created a dockerfile. It describes how to build the container, sets the working directory, copies and installs dependencies, copies all the application files into the container, and specifies the command to run the application. Dockerfile:



```
1 FROM python:3.9-slim
2
3 WORKDIR /app
4
5 COPY . /app
6
7 RUN pip install --no-cache-dir -r requirements.txt
8
9 EXPOSE 5000
10
11 CMD ["python", "app.py"]
12
```

Figure7

To build the docker image, used the command `docker build -t todo-app .`



```
C:\Users\HP\AppData\Local\Google\Cloud SDK\google-cloud-sdk\midterm>docker build -t todo-app .
[+] Building 60.1s (9/9) FINISHED
=> [internal] load build definition from Dockerfile                                docker:desktop-linux 0.1s
=> => transferring dockerfile: 189B                                              0.0s
=> [internal] load metadata for docker.io/library/python:3.9-slim                2.3s
=> [internal] load .dockerignore                                                  0.0s
=> => transferring context: 2B                                                    0.0s
=> [1/4] FROM docker.io/library/python:3.9-slim@sha256:49f94609e5a997dc16086a66ac9664591854031d4 0.0s
=> [internal] load build context                                                  0.0s
=> => transferring context: 1.34kB                                                0.0s
=> CACHED [2/4] WORKDIR /app                                                       0.0s
=> [3/4] COPY . /app                                                             0.1s
=> [4/4] RUN pip install --no-cache-dir -r requirements.txt                      57.1s
=> exporting to image                                                             0.2s
=> => exporting layers                                                            0.1s
=> => writing image sha256:ff49473fb5345715f45a49da0942bdd2fc587bf98cadf9c3359a19e375a8e553 0.0s
=> => naming to docker.io/library/todo-app                                       0.0s

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/g85jlfusam5o889fdi7fd3u
y5

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

Figure8

To check that everything is working correctly, used the command `docker run -p 8080:5000 todo-app`

```

C:\Users\HP\AppData\Local\Google\Cloud SDK\google-cloud-sdk\midterm>docker run -p 8080:5000 todo-app
* Serving Flask app 'app' (lazy loading)
* Environment: production
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
* Debug mode: on
* Running on all addresses.
WARNING: This is a development server. Do not use it in a production deployment.
* Running on http://172.17.0.2:5000/ (Press CTRL+C to quit)
* Restarting with stat
* Debugger is active!
* Debugger PIN: 776-745-375
172.17.0.1 - - [06/Oct/2024 08:57:44] "GET / HTTP/1.1" 200 -
172.17.0.1 - - [06/Oct/2024 08:57:44] "GET /static/todo.js HTTP/1.1" 200 -
172.17.0.1 - - [06/Oct/2024 08:57:44] "GET /static/todo.css HTTP/1.1" 200 -

```

Figure9

The container is created and can be deployed to google Kubernetes engine. To start, created a deployment.yaml file. It is used to describe how the to-do list application will be deployed in the Kubernetes cluster. Deployment.yaml:

```

! deployment.yaml X
HP > AppData > Local > Google > Cloud SDK > google-cloud-sdk > midterm > ! deployment.yaml
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: todo-app
5  spec:
6    replicas: 1
7    selector:
8      matchLabels:
9        app: todo-app
10   template:
11     metadata:
12       labels:
13         app: todo-app
14     spec:
15       containers:
16       - name: todo-app
17         image: gcr.io/cloudapp-project123/todo-app
18         ports:
19         - containerPort: 5000
20

```

Figure10

Next, created the cluster using the command `gcloud container clusters create todo-cluster --zone asia-central1-a`

```
C:\Users\HP\AppData\Local\Google\Cloud SDK\google-cloud-sdk\midterm>gcloud container clusters create tod
o-cluster --num-nodes=1
Note: The Kubelet readonly port (10255) is now deprecated. Please update your workloads to use the recom
mended 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).
ERROR: (gcloud.container.clusters.create) ResponseError: code=403, message=Permission denied on 'locatio
ns/asia-central1-a' (or it may not exist). This command is authenticated as beldeubaevatogzhan17@gmail.c
om which is the active account specified by the [core/account] property.
```

Figure11

There is no billing account, the system returns an error. If a billing account is available, the message should indicate that the cluster was successfully created, along with its information, such as the name, location, and version.

After this deploy the application and configured external access using the command `kubectl apply -f deployment.yaml`

```
C:\Users\HP\AppData\Local\Google\Cloud SDK\google-cloud-sdk\midterm>kubectl apply -f deployment.yaml
error: error validating "deployment.yaml": error validating data: failed to download openapi: Get "http:
//localhost:8080/openapi/v2?timeout=32s": dial tcp [::1]:8080: connectex: No connection could be made be
cause the target machine actively refused it.; if you choose to ignore these errors, turn validation off
with --validate=false
```

Figure12

If a billing account is available, the system would return the message deployment created

After check and see the external IP address that was created used the command `kubectl get services`.

```
C:\Users\HP\AppData\Local\Google\Cloud SDK\google-cloud-sdk\midterm>kubectl get services
E1006 15:54:51.127397 9568 memcache.go:265] couldn't get current server API group list: Get "http://l
ocalhost:8080/api?timeout=32s": dial tcp [::1]:8080: connectex: No connection could be made because the
target machine actively refused it.
```

Figure13

Output will display the external IP address, which will be accessible for reaching the to-do list application via the internet.

9. Managing APIs with Google Cloud Endpoints

To set up the API using google cloud endpoints, first created an `openeapi.yaml` file. It defines the api specification, describes which http methods are available for each route, specifies the urls for accessing api resources, and indicates the host address where the api will be accessible.

`Openapi.yaml`:

```
! openapi.yaml X
C: > Users > HP > AppData > Local > Google > Cloud SDK > google-cloud-sdk > midterm > ! openapi.yaml
1  swagger: '2.0'
2  info:
3    title: Todo API
4    description: API для управления задачами
5    version: 1.0.0
6  host: cloudapp-project-123.appspot.com
7  schemes:
8    - https
9  paths:
10   /tasks:
11     get:
12       summary: Получить список задач
13       operationId: listTasks
14       responses:
15         '200':
16           description: Успешный ответ
17         '404':
18           description: Задачи не найдены
19     post:
20       summary: Добавить новую задачу
21       operationId: addTask
22       parameters:
23         - in: body
24           name: task
25           required: true
26           schema:
27             type: object
28             properties:
29               title:
30                 type: string
31               description:
32                 type: string
33       responses:
34         '200':
35           description: Задача добавлена
36
```

Figure 14

To deploy the API in Google Cloud Endpoints, used the command:


```

`gcloud endpoints services deploy openapi.yaml`.
C:\Users\HP\AppData\Local\Google\Cloud SDK\google-cloud-sdk\midterm>gcloud endpoints services deploy openapi.yaml
Waiting for async operation operations/serviceConfigs.cloudapp-project-123.appspot.com:b2ec9839-d860-4830-a290-179aaf9089ab to complete...
Operation finished successfully. The following command can describe the Operation details:
  gcloud endpoints operations describe operations/serviceConfigs.cloudapp-project-123.appspot.com:b2ec9839-d860-4830-a290-179aaf9089ab

Waiting for async operation operations/rollouts.cloudapp-project-123.appspot.com:0570493e-7d24-454a-8ad3-ca2cfd93318a to complete...
Operation finished successfully. The following command can describe the Operation details:
  gcloud endpoints operations describe operations/rollouts.cloudapp-project-123.appspot.com:0570493e-7d24-454a-8ad3-ca2cfd93318a

Service Configuration [2024-10-06r0] uploaded for service [cloudapp-project-123.appspot.com]

To manage your API, go to: https://console.cloud.google.com/endpoints/api/cloudapp-project-123.appspot.com/overview?project=cloudapp-project-123

```

Figure15

The output indicates that the operation finished successfully. To verify in the google cloud console, navigate to the api&services section and check the list of active services.

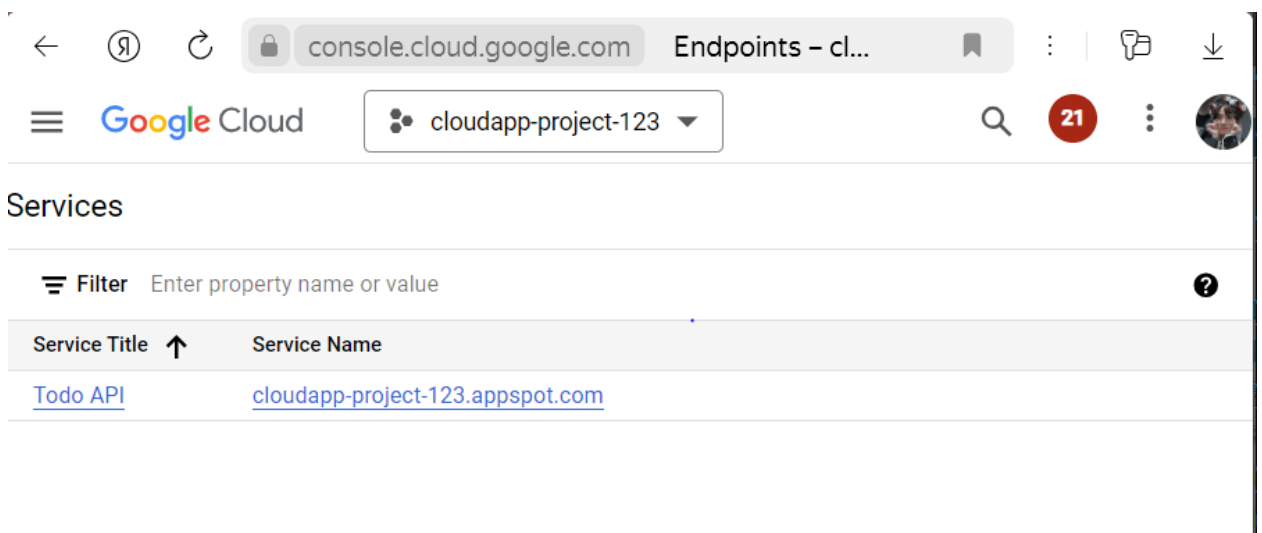


Figure16

Authentication and monitoring can be implemented using OAuth 2.0 and API keys. For this, created an OAuth 2.0 Client in API&Services, where specified the application type and the redirect URL.

OAuth 2.0 Client IDs

<input type="checkbox"/>	Name	Creation date ↓	Type	Client ID	Actions
<input type="checkbox"/>	Web client 2	Oct 6, 2024	Web application	828252158991-aq83...	

Figure17

Set up a service account to interact with the api on behalf of the project without the need for manual authentication.

Service Accounts [Manage service accounts](#)

<input type="checkbox"/>	Email	Name ↑	Actions
<input type="checkbox"/>	cloudapp-project-123@appspot.gserviceaccount.com	App Engine default service account	

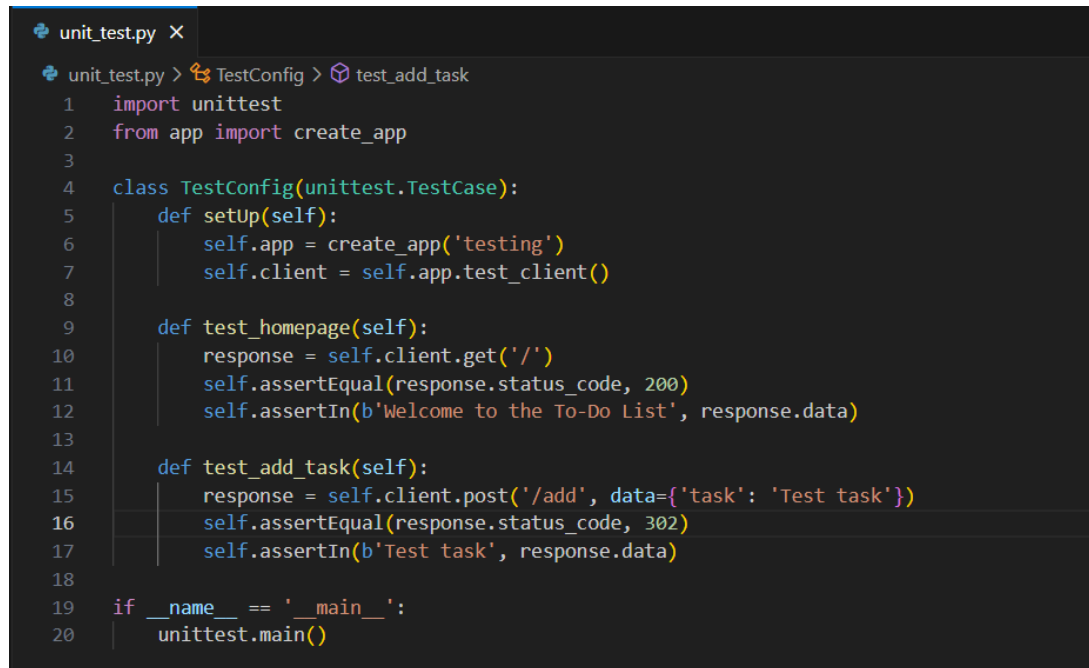
Figure18

10. Testing and Quality Assurance

To verify that the application functions correctly and meets functional and scalability requirements, used the following testing methodologies: unit testing, integration testing, and load testing results.

Unit test

Unit tests checked the components and functionality of the application. In this case, `test_homepage` verifies that a GET request is sent to the route `/` and expects a status code of 200. However, since the application could not be deployed on google app engine, the system cannot process requests correctly, resulting in errors during test execution.

A screenshot of a code editor window titled 'unit_test.py'. The editor shows a Python file with unit tests for a Flask application. The code includes imports for 'unittest' and 'create_app' from 'app'. A 'TestConfig' class inherits from 'unittest.TestCase'. It has a 'setUp' method that initializes 'self.app' and 'self.client'. There are three test methods: 'test_homepage' which performs a GET request to '/' and asserts a 200 status code and the presence of 'Welcome to the To-Do List'; 'test_add_task' which performs a POST request to '/add' with a task and asserts a 302 status code and the presence of 'Test task'; and a main block that runs the tests if the script is executed directly.

```
unit_test.py X
unit_test.py > TestConfig > test_add_task
1 import unittest
2 from app import create_app
3
4 class TestConfig(unittest.TestCase):
5     def setUp(self):
6         self.app = create_app('testing')
7         self.client = self.app.test_client()
8
9     def test_homepage(self):
10        response = self.client.get('/')
11        self.assertEqual(response.status_code, 200)
12        self.assertIn(b'Welcome to the To-Do List', response.data)
13
14    def test_add_task(self):
15        response = self.client.post('/add', data={'task': 'Test task'})
16        self.assertEqual(response.status_code, 302)
17        self.assertIn(b'Test task', response.data)
18
19 if __name__ == '__main__':
20     unittest.main()
```

Figure19

Integration test

This test checks the functionality of the API for adding tasks and processing requests, returning all tasks that have been created. However, since the application could not be linked with Google products, the system produces errors.

```
integration_test.py X
integration_test.py > test_api_get_tasks
1 import pytest
2 from app import create_app, db
3
4 @pytest.fixture
5 def client():
6     app = create_app('testing')
7     with app.test_client() as client:
8         with app.app_context():
9             db.create_all()
10        yield client
11        db.drop_all()
12
13 def test_api_endpoint(client):
14     response = client.post('/api/tasks', json={'task': 'Test task'})
15     assert response.status_code == 201
16     assert response.json['task'] == 'Test task'
17
18 def test_api_get_tasks(client):
19     client.post('/api/tasks', json={'task': 'Test task'})
20     response = client.get('/api/tasks')
21     assert response.status_code == 200
22     assert len(response.json) == 1
```

Figure20

Load testing results

This test uses the locust library to simulate user behavior. The userbehavior class defines actions where a user sends a GET request to the homepage and a POST request to add a task. Since the application could not be deployed on google app engine, the file cannot execute tests on the to-do list application.

```
load_testing.py X
load_testing.py > WebsiteUser
1 from locust import HttpUser, TaskSet, task
2
3 class UserBehavior(TaskSet):
4     @task(1)
5     def load_homepage(self):
6         self.client.get('/')
7
8     @task(2)
9     def add_task(self):
10        self.client.post('/add', json={'task': 'Load Test Task'})
11
12 class WebsiteUser(HttpUser):
13     tasks = [UserBehavior]
14     min_wait = 1000
15     max_wait = 3000
```

Figure21

11. Monitoring and Maintenance

To ensure the reliable operation of the to-do list application, it is essential to use Google Cloud Monitoring and Google Cloud Logging. These monitoring tools track performance metrics such as response times and the number of requests.

12. Challenges and Solutions

During the development and deployment of the to-do list application on google cloud platform, the issue was that google app engine and google cloud functions required an active billing account. This made it difficult not only to deploy the application but also to test its functionality.

To resolve the issue, connecting a temporary billing account was not helpful, as it also requested sensitive user information (such as card number, CVV, etc.). Therefore, the decision was made to explore additional resources online and find possible solutions for various deployment scenarios.

13. Conclusion

In conclusion, a scalable web application for a to-do list was successfully developed and deployed using Google Cloud Platform (GCP) services. The application was built on the Flask framework and deployed on Google App Engine. Additionally, serverless functions were integrated through Google Cloud Functions, and the application was containerized using Docker for deployment on Google Kubernetes Engine (GKE).

14. References

1. <https://cloud.google.com/why-google-cloud/?hl=ru>
2. <https://cloud.google.com/products?hl=ru>
3. <https://cloud.google.com/why-google-cloud/?hl=ru>
4. <https://cloud.google.com/products?hl=ru>
5. <https://cloud.google.com/sdk/docs/install>
6. <https://cloud.google.com/appengine/docs/legacy/standard/python/config/appref>
7. <https://cloud.google.com/functions/docs/writing/specifying-dependencies-python>
8. <https://cloud.google.com/functions/docs/console-quickstart>
9. <https://cloud.google.com/kubernetes-engine/docs/quickstarts/create-cluster>
10. <https://cloud.google.com/endpoints/docs/openapi>