School of Information Technology and Engineering

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

Done by: **Altair Kabdrakhmanov**
4th year student, ID: 21B030829,
Information Systems

Almaty, Fall 2024

Table of Contents

1. Executive Summary

The project aimed to develop a scalable web application a phonebook ensuring high performance and security. To achieve this, I utilized various technologies, including Google Cloud Platform (GCP) for hosting and serverless computing, Flask as the backend web framework, Docker for containerization, openAPI for cloud API, etc. The phonebook functionality allows users to add, update, retrieve, and delete contact information seamlessly. Through unit and integration testing, I ensured the application is robust and reliable by load testing.

2. Introduction

Google Cloud Platform (GCP) is a service that helps people build and run web applications in the cloud. It is important for creating scalable applications because it has features like auto-scaling, which means it can automatically adjust resources to handle more users or traffic. People choose GCP for its cost-effectiveness since you only pay for what you use, and it allows for faster development and deployment of applications. GCP also provides strong security to protect data. Also, one of the motivation it was Free Trail 300$ that helped me to accomplish this work.

3. Project Objectives

The main objective is to use serverless architecture. This approach allows us to build and run the application without managing servers, which makes it easier and more efficient. Another goal of the project is to create a responsive application that works well on different devices, including smartphones, tablets, and computers. This means that users will have a good experience, no matter how they access the application. Finally, I aim to ensure high availability. This means that the application should always be accessible, so users can rely on it whenever they need it. By achieving these objectives, I want to deliver a smooth and reliable service that meets the needs of all users.

4. Google Cloud Platform Overview

Google Cloud Platform (GCP) has a strong architecture that helps developers create and deploy applications easily. The core services of GCP include computing, storage, and databases. For

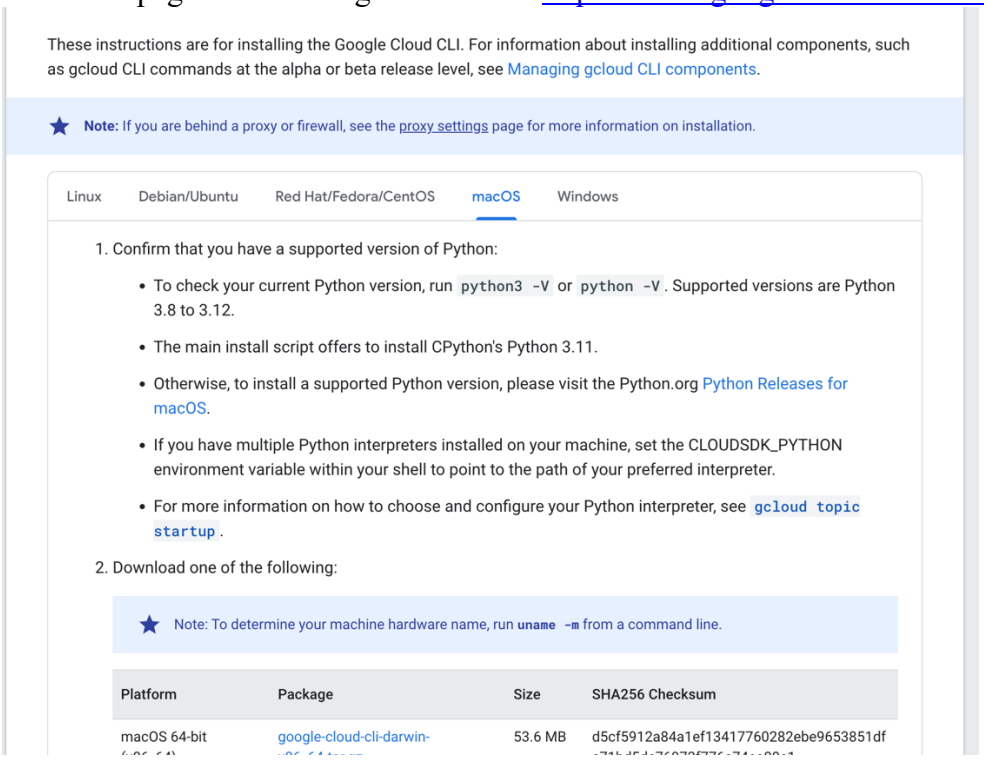
computing, GCP offers services like Google Compute Engine, which lets you run virtual machines, and Google App Engine, which allows you to build and host applications without managing the servers. For storage, you can use Google Cloud Storage to save files and Google Cloud SQL for managing databases.
One big benefit of using GCP is its scalability. This means that you can easily grow your application as more users start to use it. GCP also provides high security, ensuring that your data and applications are safe. Additionally, GCP offers tools for monitoring and managing your applications, making it easier to keep everything running smoothly

5. Google Cloud SDK and Cloud Shell

Setup

To start working on Google Cloud you need to download and install Google Cloud SDK. Do the followings to have Google Cloud SDK on your machine:

- Go to this page “Install the gcloud CLI” - https://cloud.google.com/sdk/docs/install

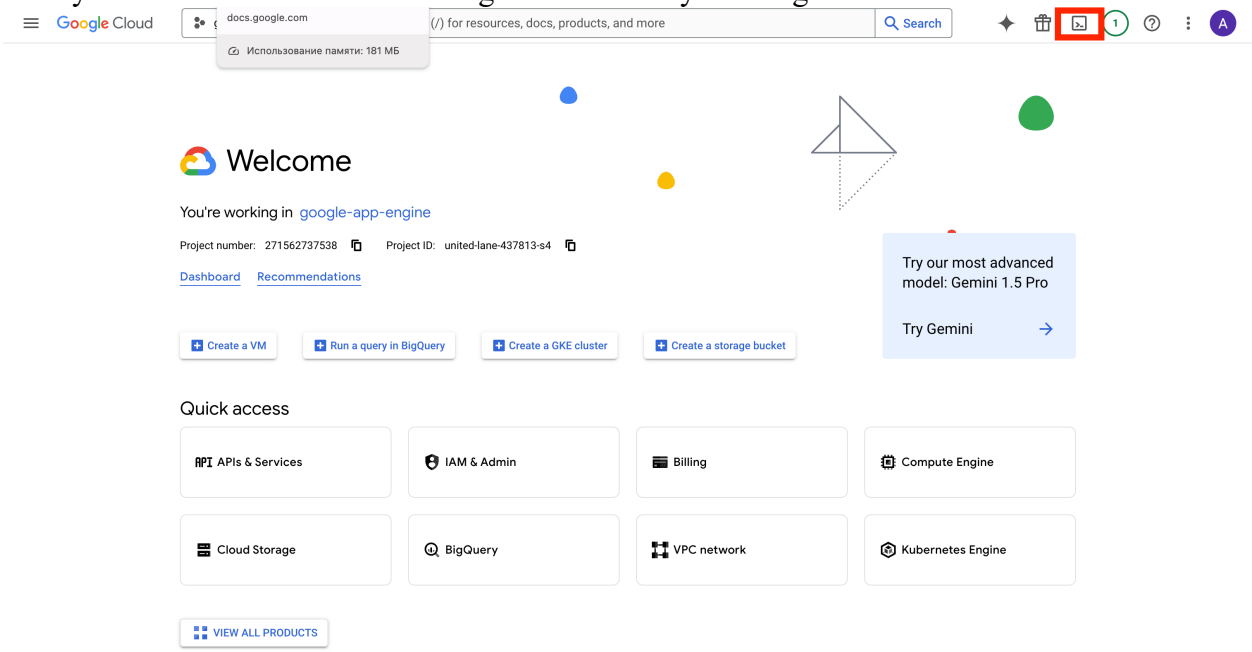


- Select your operating machine. For my case it is MacOS
- Check you current version of Python. It must match with requirements for Google Cloud SDK
- Download your package. For my case it is Apple Silicon
- Extract this archive file on Home directory.
- To add Google Cloud SDK to your PATH. You need to use this command in root of folder where you extracted the archive. For my case it is HOME directory. The command: ./google-cloud-sdk/install.sh

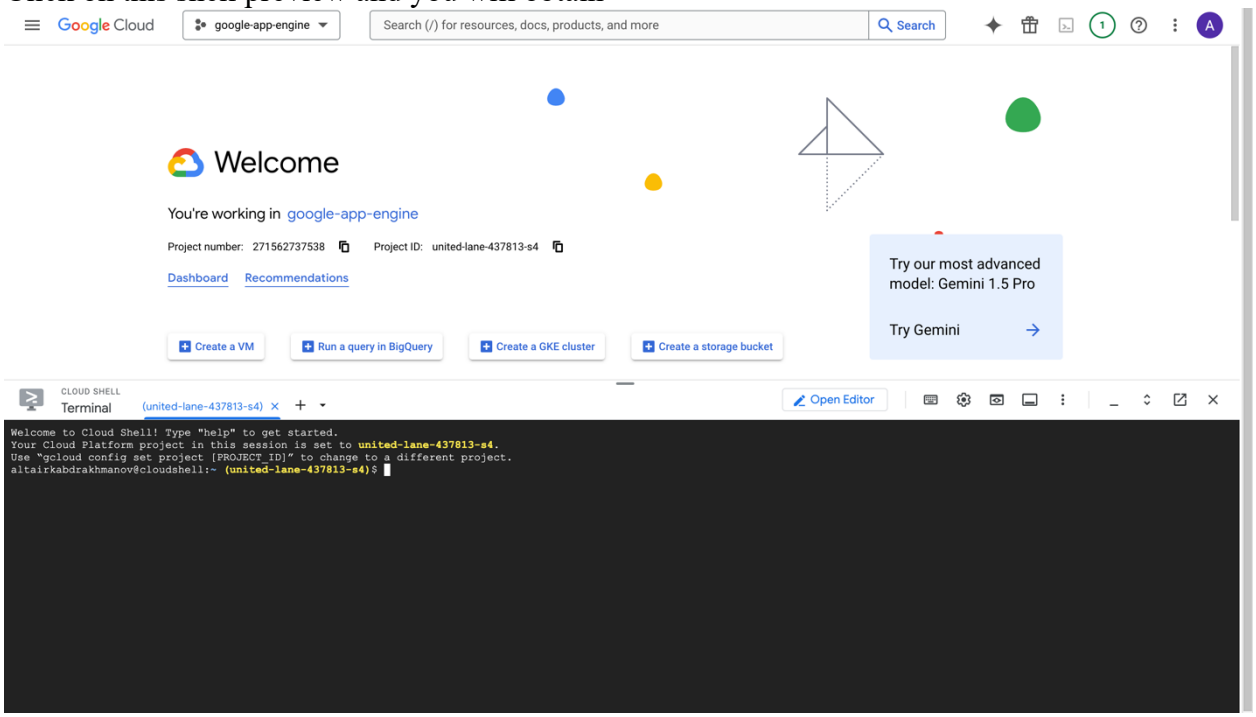
- Now you can run the gcloud auth login command and do authentication with your google cloud.

Cloud Shell Usage

Also you can utilize Cloud Shell Usage that is already in Google Cloud Shell.



Click on this shell preview and you will obtain



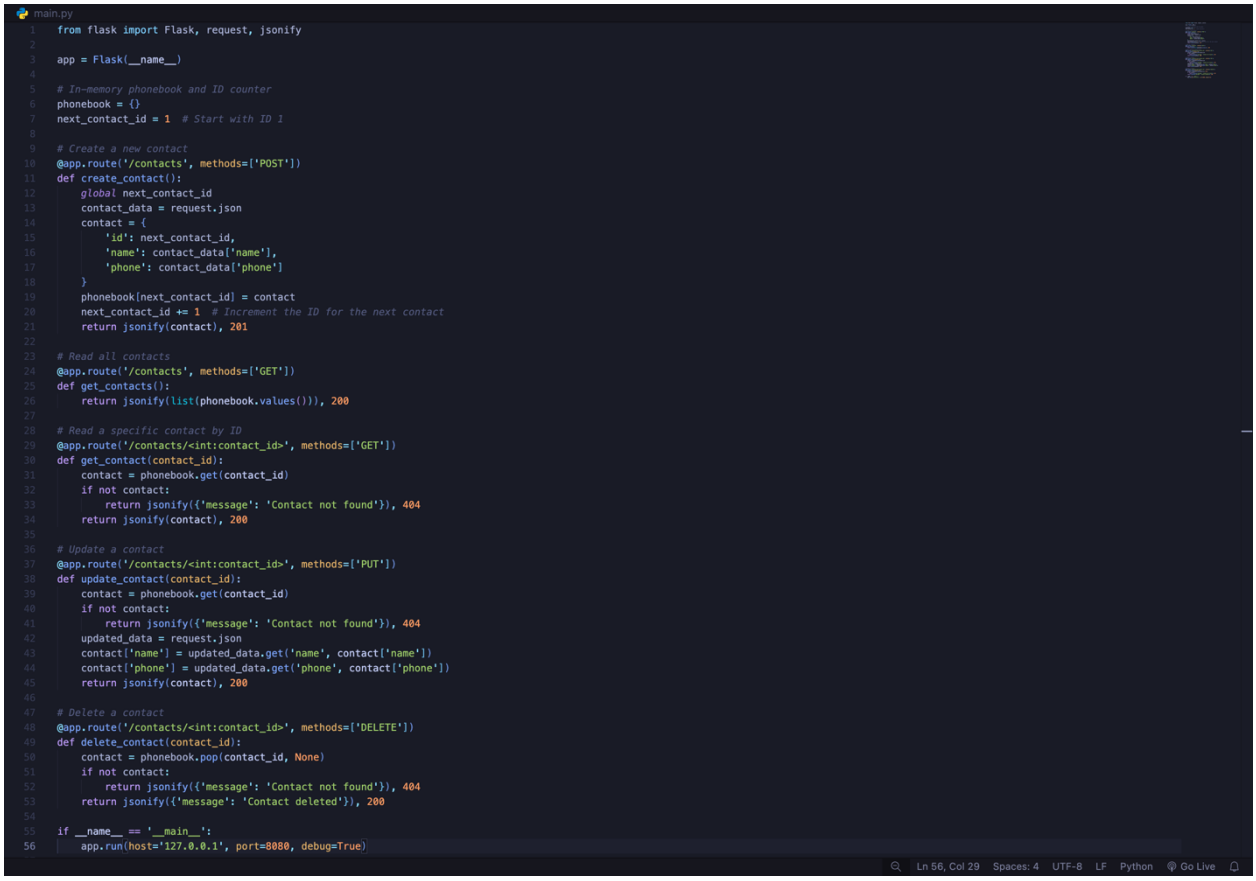
So it has the same functionality as Google Cloud SDK but it is within Google Cloud Console.

6. Google App Engine

Application Development

I created phonebook application with CRUD operations with using Flask python framework. Main endpoints: /contacts (get contacts, create contact), /contacts/{id} (update, create customers).

The code:



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

app = Flask(__name__)

# In-memory phonebook and ID counter
phonebook = {}
next_contact_id = 1  # Start with ID 1

# Create a new contact
@app.route('/contacts', methods=['POST'])
def create_contact():
    global next_contact_id
    contact_data = request.json
    contact = {
        'id': next_contact_id,
        'name': contact_data['name'],
        'phone': contact_data['phone']
    }
    phonebook[next_contact_id] = contact
    next_contact_id += 1  # Increment the ID for the next contact
    return jsonify(contact), 201

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

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

# Update a contact
@app.route('/contacts/<int:contact_id>', methods=['PUT'])
def update_contact(contact_id):
    contact = phonebook.get(contact_id)
    if not contact:
        return jsonify({'message': 'Contact not found'}), 404
    updated_data = request.json
    contact['name'] = updated_data.get('name', contact['name'])
    contact['phone'] = updated_data.get('phone', contact['phone'])
    return jsonify(contact), 200

# Delete a contact
@app.route('/contacts/<int:contact_id>', methods=['DELETE'])
def delete_contact(contact_id):
    contact = phonebook.pop(contact_id, None)
    if not contact:
        return jsonify({'message': 'Contact not found'}), 404
    return jsonify({'message': 'Contact deleted'}), 200

if __name__ == '__main__':
    app.run(host='127.0.0.1', port=8080, debug=True)
```

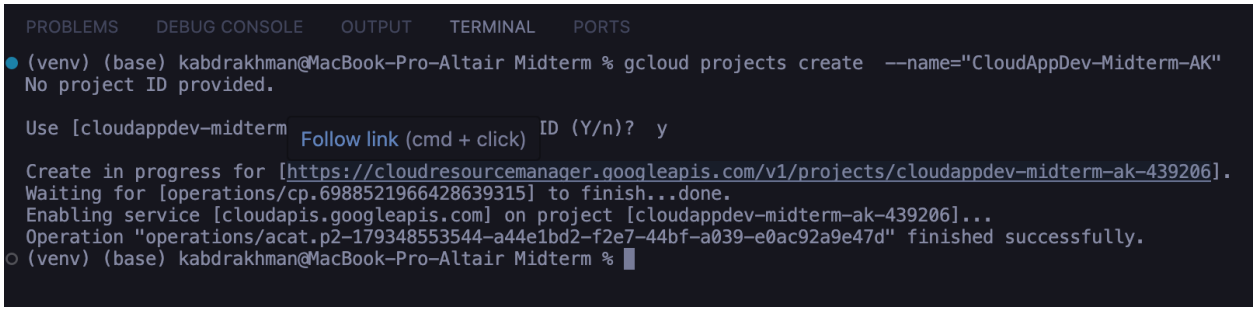
Deployment

To deploy a phonebook application to google cloud engine. We need to do the followings:

- Create project in Google Cloud
- Activate Billing
- Activate Google App Engine

Create Project

Used this command to create my project: gcloud projects create --name="CloudAppDev-Midterm-AK"



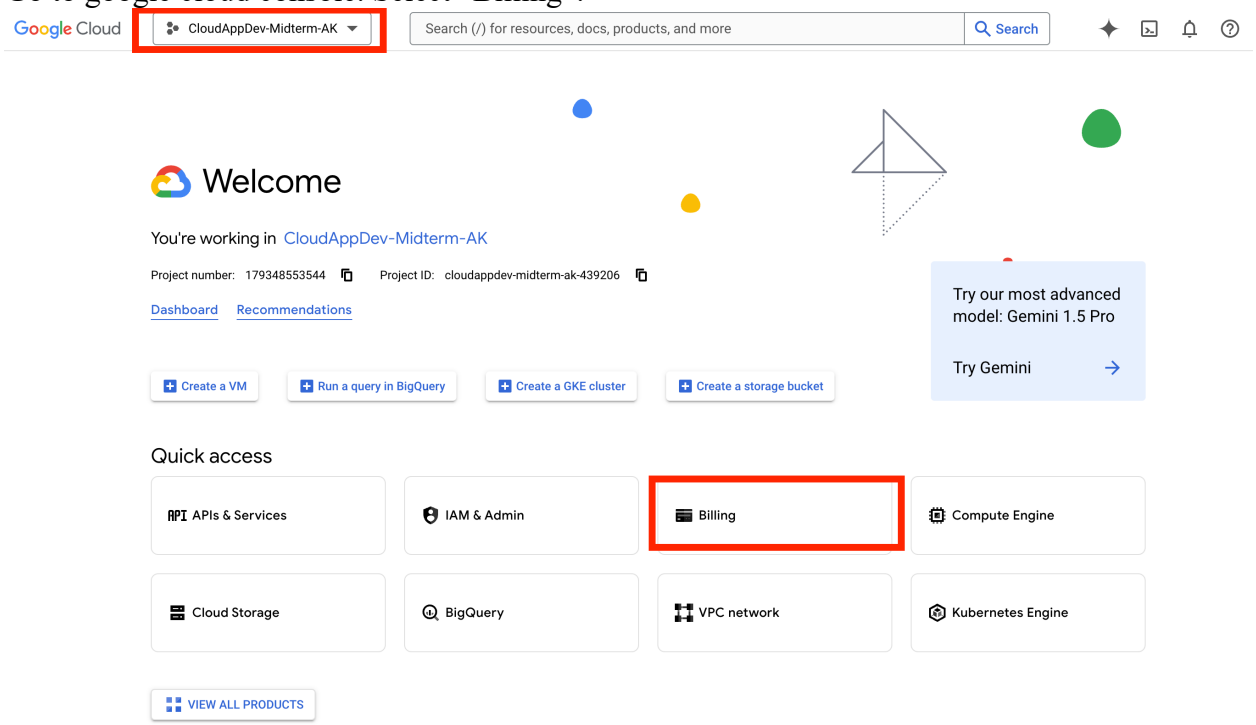
```
PROBLEMS  DEBUG CONSOLE  OUTPUT  TERMINAL  PORTS

(venv) (base) kabdrakhman@MacBook-Pro-Altair Midterm % gcloud projects create  --name="CloudAppDev-Midterm-AK"
No project ID provided.

Use [cloudappdev-midterm                Follow link (cmd + click)  ID (Y/n)?  y

Create in progress for [https://cloudresourcemanager.googleapis.com/v1/projects/cloudappdev-midterm-ak-439206].
Waiting for [operations/cp.6988521966428639315] to finish...done.
Enabling service [cloudapis.googleapis.com] on project [cloudappdev-midterm-ak-439206]...
Operation "operations/acat.p2-179348553544-a44e1bd2-f2e7-44bf-a039-e0ac92a9e47d" finished successfully.
(venv) (base) kabdrakhman@MacBook-Pro-Altair Midterm %
```

Set Project

Used this command to set as working project gcloud config set project cloudappdev-midterm-ak-439206

Activate Billing Account

Go to google cloud console. Select “Billing”.

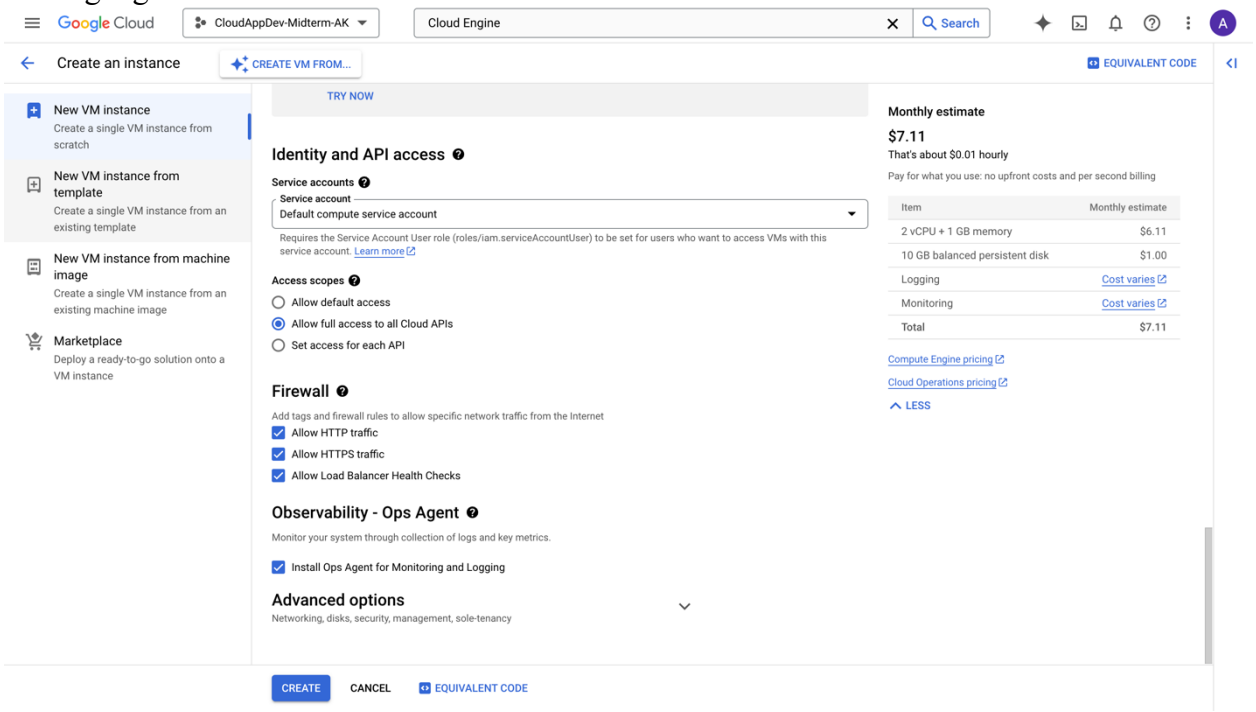


Press the “Link Billing Account” and choose you billing.
Now your project ready for starting the deployment on Google Cloud.

Deploy Phonebook application

Activate Cloud Engine: gcloud services enable compute.googleapis.com

Go to google cloud console and create VM Instance with e2-micro.



To deploy our application within Google Cloud Engine we need to create app.yaml file prior deploying. The app.yaml looks:

app.yaml

```yaml
runtime: python39
handlers:
  - url: /.*
    script: auto

```

Used command to create requirements.txt: pip freeze > requirements.txt

requirements.txt

```
blinker==1.8.2
click==8.1.7
Flask==3.0.3
itsdangerous==2.2.0
Jinja2==3.1.4
MarkupSafe==3.0.2
Werkzeug==3.0.4

```

Used command to deploy: gcloud app deploy

```
(base) kabdrakhman@MacBook-Pro-Altair Midterm % gcloud app deploy
Services to deploy:

descriptor:                  [/Users/kabdrakhman/Documents/KBTU/7th semester/Cloud-Application-Development/Midterm/app.yaml]
source:                      [/Users/kabdrakhman/Documents/KBTU/7th semester/Cloud-Application-Development/Midterm]
target project:              [cloudappdev-midterm-ak-439206]
target service:              [default]
target version:              [20241020t113342]
target url:                  [https://cloudappdev-midterm-ak-439206.df.r.appspot.com]
target service account:      [cloudappdev-midterm-ak-439206@appspot.gserviceaccount.com]


Do you want to continue (Y/n)?  y

Beginning deployment of service [default]...

╔═ Uploading 1 file to Google Cloud Storage                ═╗

File upload done.
Updating service [default]...done.
Setting traffic split for service [default]...done.
Deployed service [default] to [https://cloudappdev-midterm-ak-439206.df.r.appspot.com]

You can stream logs from the command line by running:
  $ gcloud app logs tail -s default

To view your application in the web browser run:
  $ gcloud app browse
(base) kabdrakhman@MacBook-Pro-Altair Midterm %
```

Ln 4, Col 1 Spa

Checking the result

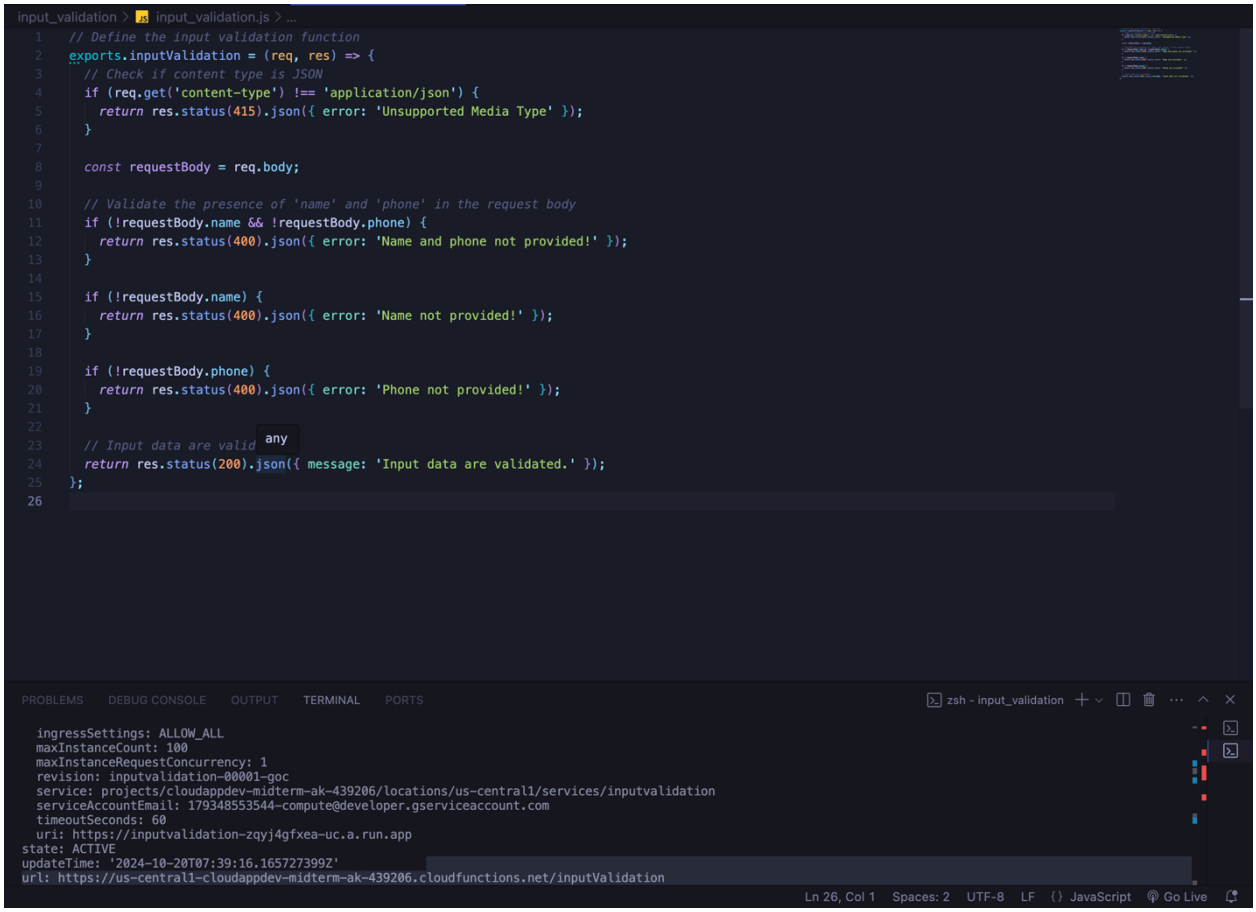
Go to page - https://cloudappdev-midterm-ak-439206.df.r.appspot.com/contacts

7. Building with Google Cloud Functions

Serverless Functions: Explain how Cloud Functions were created and integrated into the main application, including specific use cases.

1. Input Validation Cloud Function

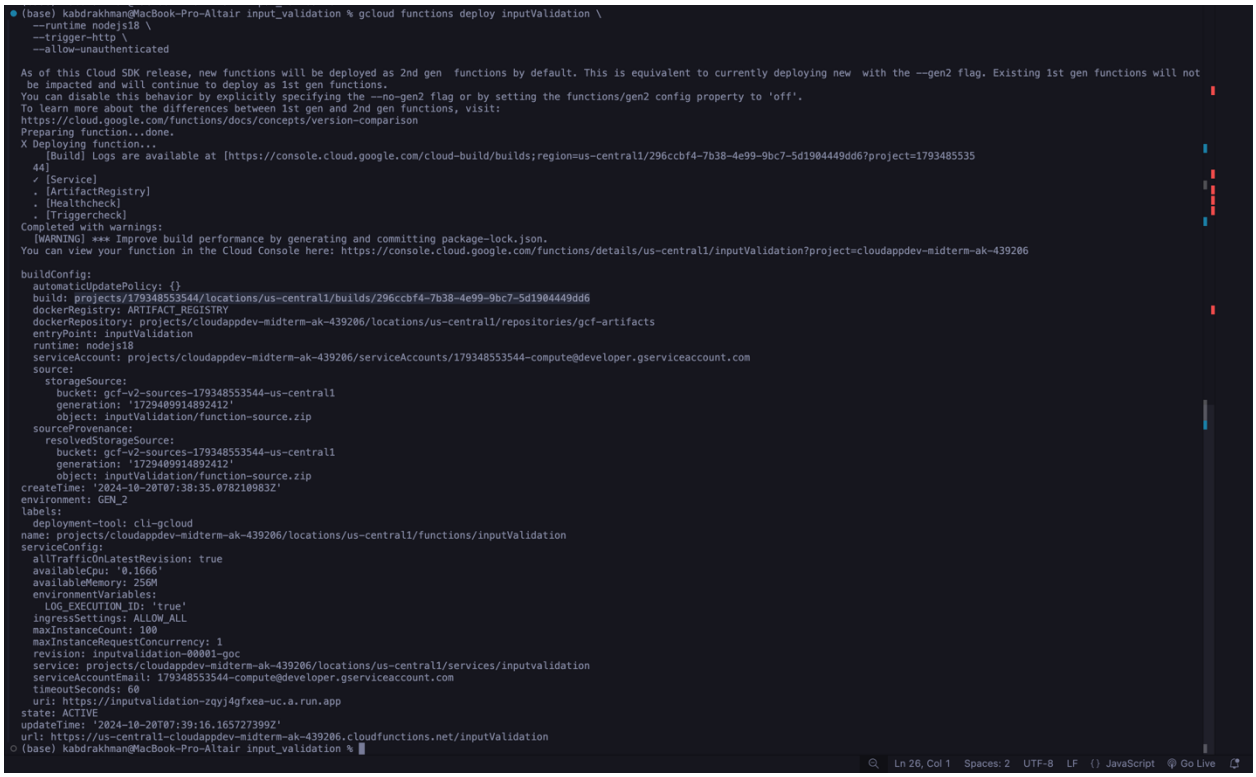
Code:



Command:

```
gcloud functions deploy inputValidation \
  --runtime nodejs18 \
  --trigger-http \
  --allow-unauthenticated
```

Result:

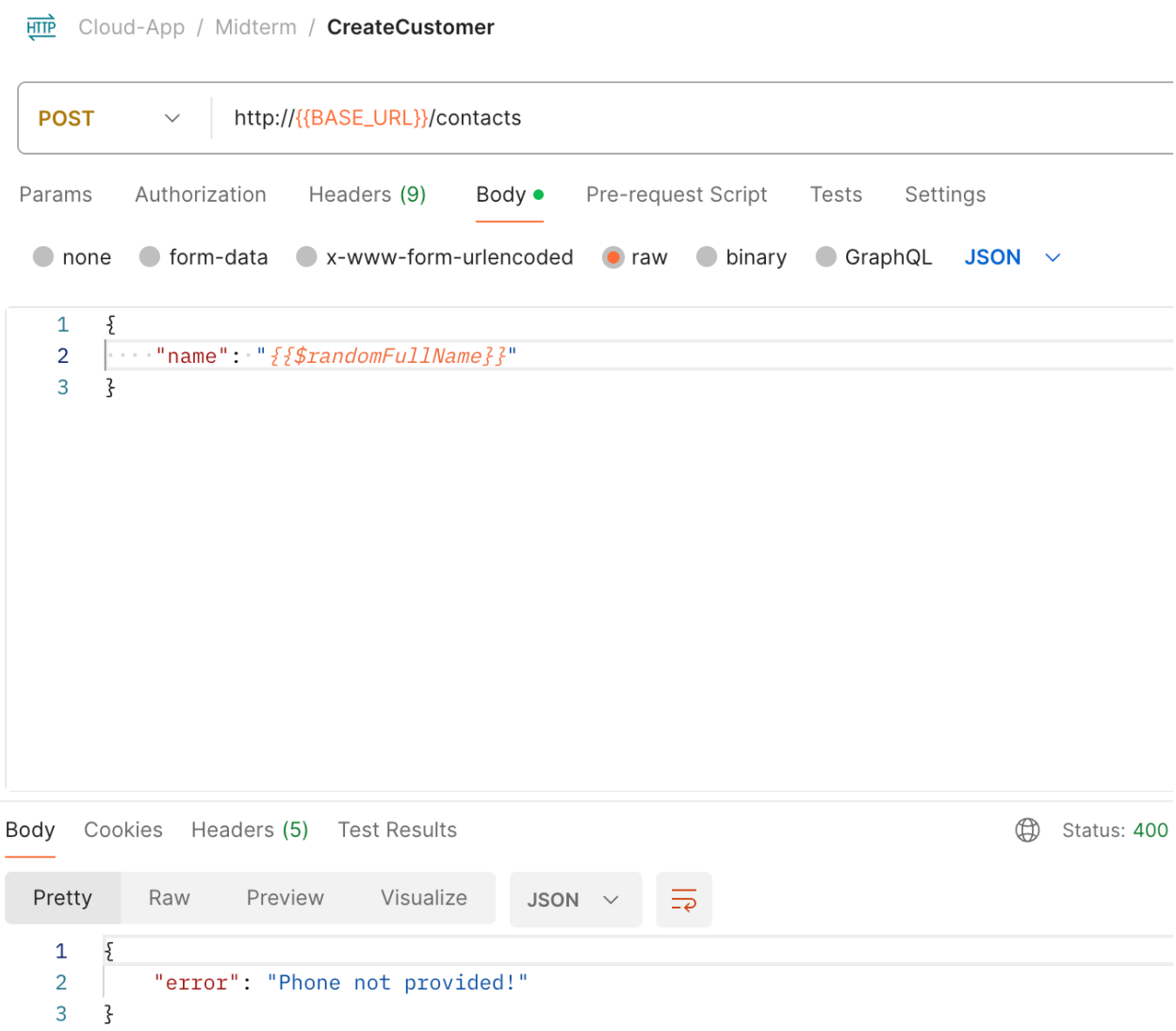


Integration with input_validation cloud function with phonebook application:

main.py

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

app = Flask(__name__)

# In-memory phonebook and ID counter
phonebook = {}
next_contact_id = 1  # Start with ID 1

# URL for the Cloud Fu                    /our actual Cloud Function URL)
CLOUD_FUNCTION_URL = 'https://inputvalidation-zqyj4gfxea-uc.a.run.app'

# Function to validate the request using the Cloud Function
def validate_request(contact_data):
    try:
        response = requests.post(
            CLOUD_FUNCTION_URL,
            json=contact_data,
            headers={'Content-Type': 'application/json'}
        )
        # Return the JSON response and status code
        return response.json(), response.status_code
    except requests.RequestException as e:
        return {'error': 'Failed to connect to validation service'}, 500

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

    # Validate the request data using the Cloud Function
    validation_result, status_code = validate_request(contact_data)
    if status_code != 200:
        # If validation fails, return the validation error response
        return jsonify(validation_result), status_code

    # Proceed to create the contact if validation succeeds
    contact = {
        'id': next_contact_id,
        'name': contact_data['name'],
        'phone': contact_data['phone']
    }
    phonebook[next_contact_id] = contact
    next_contact_id += 1  # Increment the ID for the next contact
    return jsonify(contact), 201
```

Result:



Send notification cloud function

Function log_event for adding logging when doing action CRUD

```python

# URL for the Cloud Function (replace with your actual Cloud Function URL)
CLOUD_INPUT_VALIDATION = 'https://inputvalidation-zqyj4gfxea-uc.a.run.app'
CLOUD_FUNCTION_LOGGING_URL = 'https://logevents-zqyj4gfxea-uc.a.run.app'

# Function to validate the request using the Cloud Function
def validate_request(contact_data):
    try:
        response = requests.post(
            CLOUD_INPUT_VALIDATION,
            json=contact_data,
            headers={'Content-Type': 'application/json'}
        )
        # Return the JSON response and status code
        return response.json(), response.status_code
    except requests.RequestException as e:
        return {'error': 'Failed to connect to validation service'}, 500

# Function to log events using the Cloud Function
def log_event(contact_data, method):
    try:
        response = requests.post(
            CLOUD_FUNCTION_LOGGING_URL,
            json={
                'name': contact_data.get('name'),
                'phone': contact_data.get('phone'),
                'method': method  # Include the HTTP method to indicate the ac
            },
            headers={'Content-Type': 'application/json'}
        )
        if response.status_code != 200:
            print(f"Failed to log {method} action.")
        else:
            print(f"{method} action logged successfully.")
    except requests.RequestException:
        print("Failed to connect to logging service.")

```

Create a new contact with logging

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

    # Validate the request data using the Cloud Function
    validation_result, status_code = validate_request(contact_data)
    if status_code != 200:
        # If validation fails, return the validation error response
        return jsonify(validation_result), status_code

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

    # Log the creation event
    log_event(contact_data, 'POST')

    return jsonify(contact), 201
```

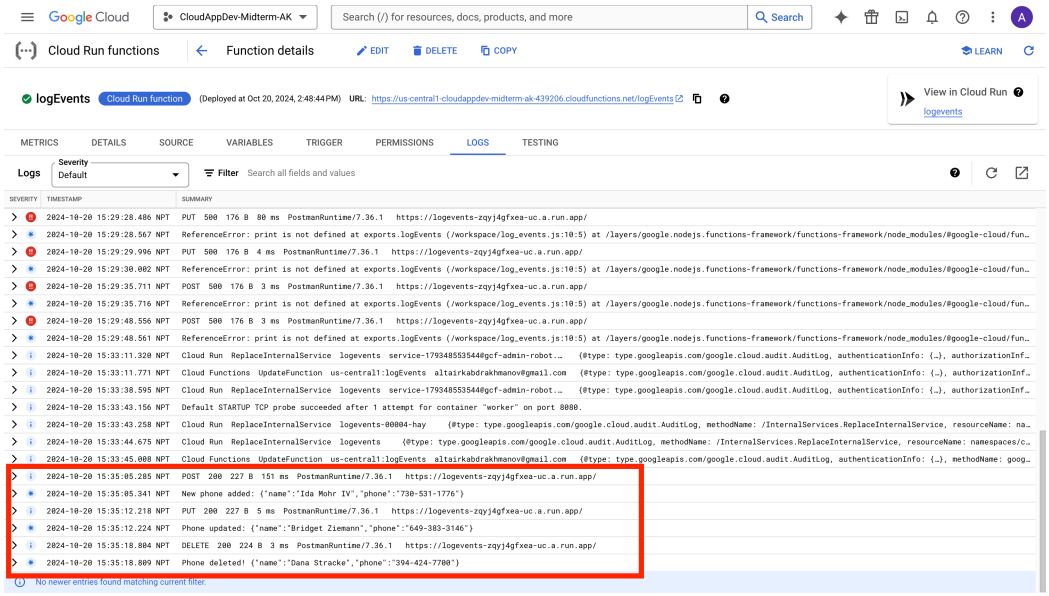
Update contact with logging

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

    # Log the creation event
    log_event(updated_data, 'PUT')

    return jsonify(contact), 200
```

Delete contact with logging

```python
# Delete task from To-Do List
@app.route('/delete/<string:todo_id>', methods=['DELETE'])
@require_api_key
def delete_todo(todo_id):
    global todos
    todo_to_delete = next((todo for todo in todos if todo['id'] == todo_id), None)

    if not todo_to_delete:
        return jsonify({"error": "Todo not found."}), 404
    else:
        requests.post(
            SEND_NOTIFICATION_URL,
            headers={'Content-Type': 'application/json'},
            json={
                "todo": todo_to_delete,
                "action": "deleted"
            }
        )

    print(f"Deleting todo with id: {todo_id}")
    todos = [todo for todo in todos if todo['id'] != todo_id]
    return jsonify({"message": "Todo deleted successfully"})
```

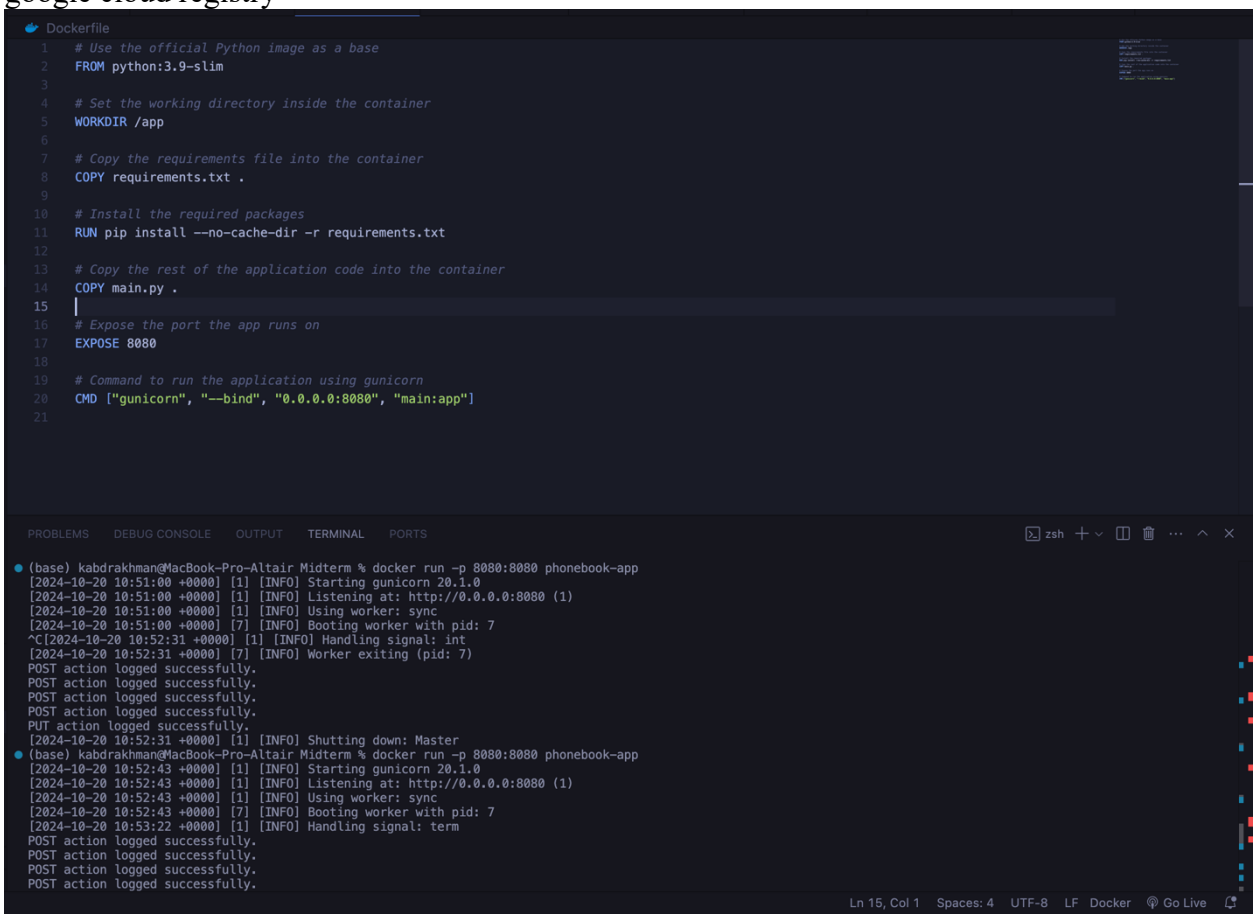
Result



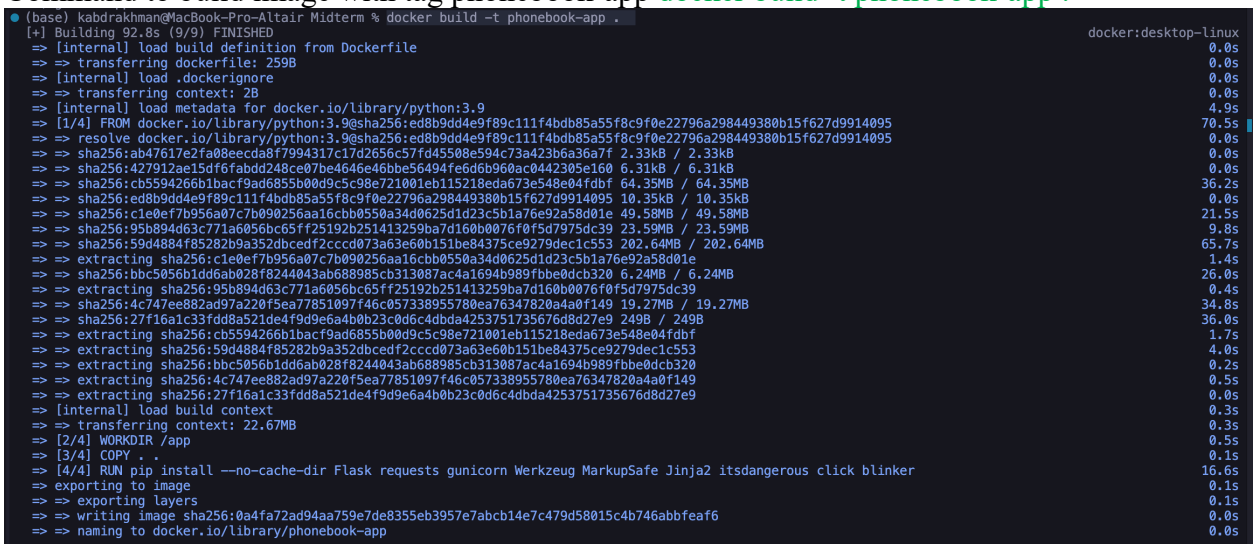
8. Containerizing Applications

Working with Dockerfile

With step that need to be taken is to create Dockerfile for building image and further push to google cloud registry



Command to build image with tag phonebook-app docker build -t phonebook-app .



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

Pushing image to GCR

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

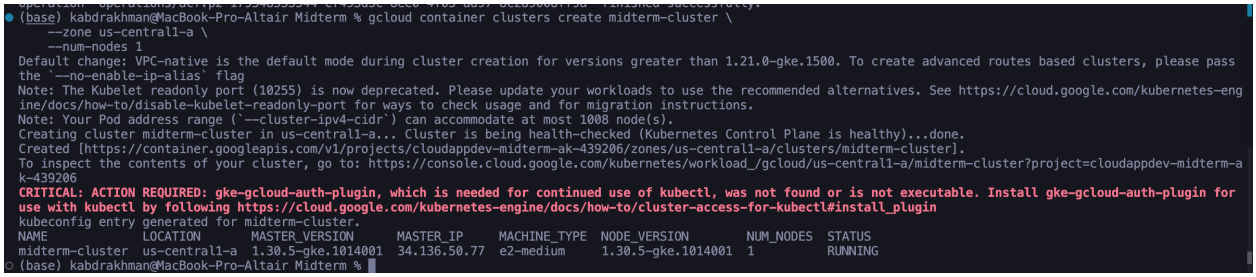
To push my docker images I used this command to tag phonebook with my google cloud registry (GCR):
docker tag phonebook-app gcr.io/cloudappdev-midterm-ak-439206/phonebook-app:latest

Only then I could push image to GCR
docker push gcr.io/cloudappdev-midterm-ak-439206/phonebook-app:latest

Working with K8S

To create kubernetes cluster used code:

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



To install kubectl
gcloud components install gke-gcloud-auth-plugin

Command to enable cluster in kubectl config file
gcloud container clusters get-credentials midterm-cluster --zone us-central1-a

Result

```
(base) kabdrakhman@MacBook-Pro-Altair Midterm % kubectl get namespaces
NAME                 STATUS   AGE
default              Active   5m38s
gke-managed-cim      Active   5m14s
gke-managed-system   Active   5m2s
gmp-public           Active   4m47s
gmp-system           Active   4m48s
kube-node-lease      Active   5m38s
kube-public          Active   5m38s
kube-system          Active   5m38s
(base) kabdrakhman@MacBook-Pro-Altair Midterm %
```

Create deployment and service yaml file with our application

Create separate namespace for our application
kubectl create namespace phonebook

9. Managing APIs with Google Cloud Endpoints

API Setup

To set up API using Cloud Endpoints we need to specify openAPI yaml file. Below you can find it.

openapi.yaml

```yaml
swagger: '2.0'
info:
  title: Phonebook API
  description: API for managing contacts in a phonebook.
  version: 1.0.0
host: "cloudappdev-midterm-ak-439206.appspot.com"
schemes:
  - https
  - http
paths:
  /contacts:
    get:
      summary: Get all contacts
      operationId: getContacts
      responses:
        '200':
          description: List of contacts
        '500':
          description: Internal server error
      security:
        - api_key: []

    post:
      summary: Create a new contact
      operationId: createContact
      parameters:
        - in: body
          name: contact
```

The host name must in this format: “project_id.appspot.com”, otherwise it doesn’t work. Below official documentation.

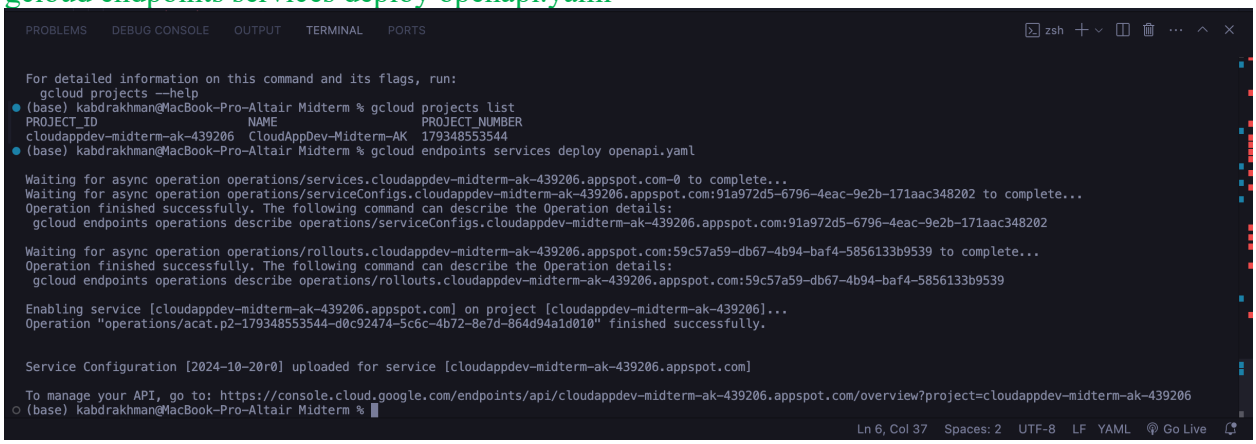
- If you are using the `cloud.goog` domain, confirm that the value for the `host` field is in the following format, and that the project ID is correct:

```
API_NAME.endpoints.YOUR_PROJECT_ID.cloud.goog
```

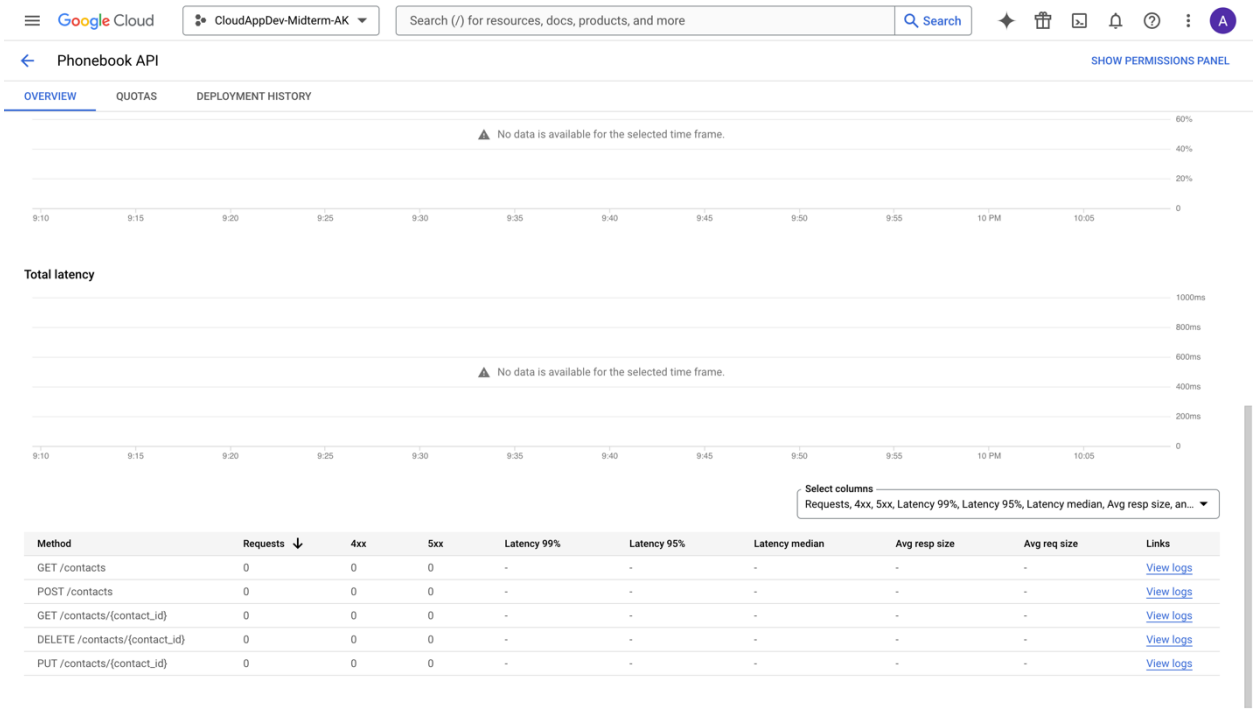
- If you are using the `appspot.com` domain (supported for App Engine only), confirm that the `host` field is in the following format, and that the project ID is correct:

```
YOUR_PROJECT_ID.appspot.com
```

Command to deploy endpoint:
gcloud endpoints services deploy openapi.yaml



As result we can monitor now Phonebook API by this link - https://console.cloud.google.com/endpoints/v2/api/cloudappdev-midterm-ak-439206.appspot.com/overview?project=cloudappdev-midterm-ak-439206.



Security and Monitoring: Describe how authentication and monitoring were implemented.

Used this command to create api key:
gcloud services api-keys create --display-name="secret-key"

```
(base) kabdrakhman@MacBook-Pro-Altair Midterm % gcloud services api-keys create --display-name="secret-key"
Operation [operations/akmf.p7-179348553544-8ed4fca4-801e-4158-b29b-ecb42c1d309e] complete. Result: {
    "@type":"type.googleapis.com/google.api.apikeys.v2.Key",
    "createTime":"2024-10-20T17:13:38.110786Z",
    "displayName":"secret-key",
    "etag":"W/\"MmFoNmvi9GwQx474zoAKXw==\"",
    "keyString":"AIzaSyAV8oQPuVDrC44gW9fOtSXZEYEW7JmVwLE",
    "name":"projects/179348553544/locations/global/keys/24424bfb-504e-446b-b92e-d058fc58dc68",
    "uid":"24424bfb-504e-446b-b92e-d058fc58dc68",
    "updateTime":"2024-10-20T17:13:38.131614Z"
}
(base) kabdrakhman@MacBook-Pro-Altair Midterm %
```

After obtaining the key we do the followings:

- Creating wrapper function

```python
# Security key
API_KEY = "AIzaSyAV8oQPuVDrC44gW9fOtSXZEYEW7JmVwLE"

# URL for the Cloud Function (replace with your actual Cloud Function URL)
CLOUD_INPUT_VALIDATION = 'https://inputvalidation-zqyj4gfxea-uc.a.run.app'
CLOUD_FUNCTION_LOGGING_URL = 'https://logevents-zqyj4gfxea-uc.a.run.app'

# Secret key check
def secret_key(func):
    def wrapper(*args, **kwargs):
        secret_key = request.headers.get('secret_key')
        if secret_key != API_KEY:
            return jsonify({"error": "Unauthorized: Invalid API Key"}), 401
        return func(*args, **kwargs)
    wrapper.__name__ = func.__name__
    return wrapper
```

- Adding this function to each method

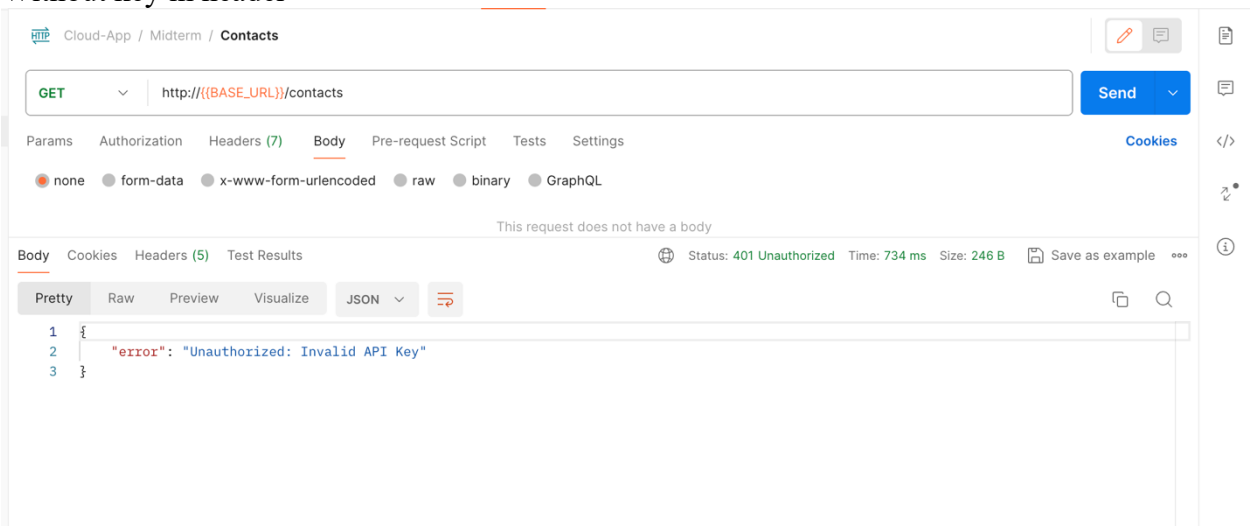
```python
# Read all contacts
@app.route('/contacts', methods=['GET'])
@require_api_key
def get_contacts():
    return jsonify(list(phonebook.values())), 200

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

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

    # Log the creation event
    log_event(updated_data, 'PUT')

    return jsonify(contact), 200

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

- Adding secret key as security in openapi.yaml file

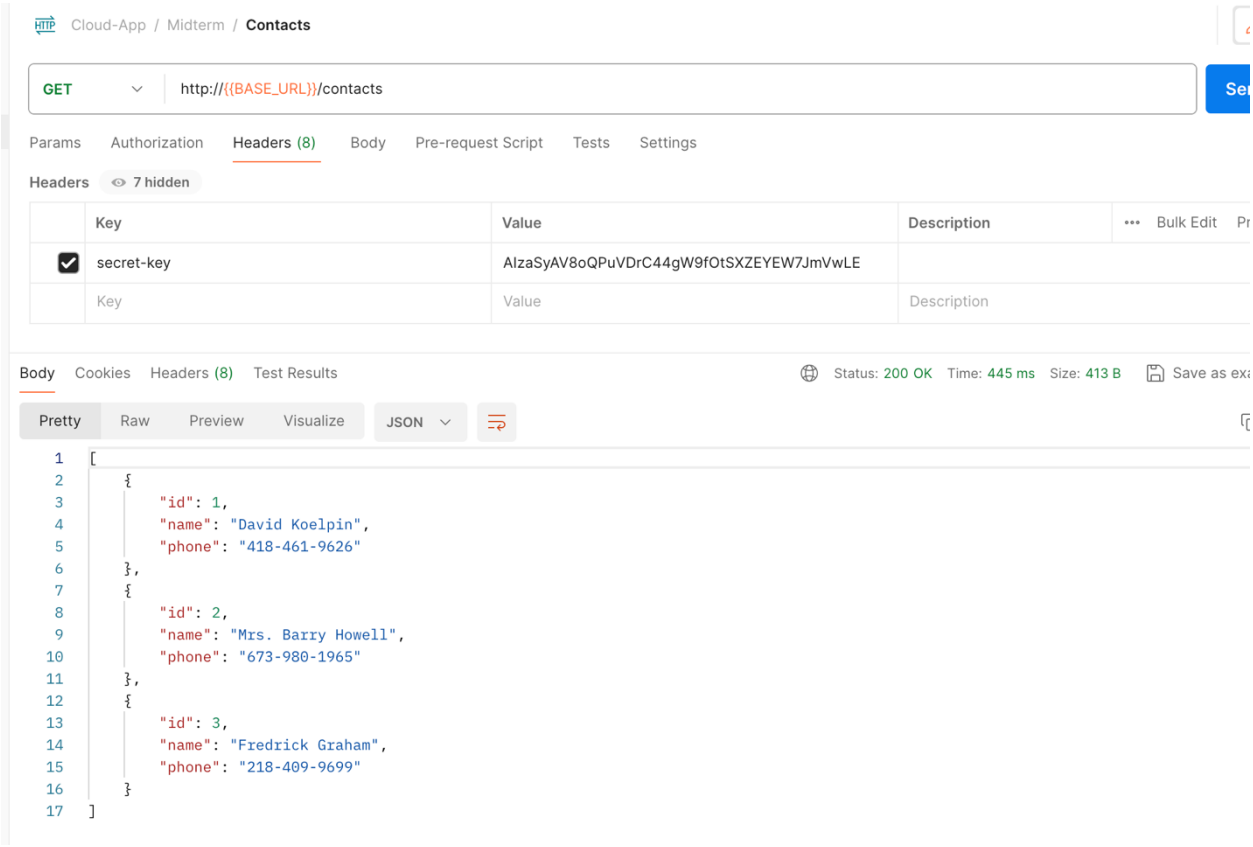
```yaml
    delete:
      summary: Delete a contact
      operationId: deleteContact
      parameters:
        - in: path
          name: contact_id
          type: integer
          required: true
      responses:
        '200':
          description: Contact deleted successfully
        '404':
          description: Contact not found
        '500':
          description: Internal server error
      security:
        - api_key: []

securityDefinitions:
  api_key:
    type: apiKey
    name: secret_key
    in: header
```

- Do gcloud app deploy and gcloud endpoints services deploy openapi.yaml

Result:
Without key in header



With key



10. Testing and Quality Assurance

Unit testing:

testing > unit-test.py

```python
import unittest
from flask import json
from main import app, phonebook  # Import the Flask app and the phonebook dictionary

class PhonebookAppUnitTests(unittest.TestCase):
    def setUp(self):
        self.app = app.test_client()
        self.app.testing = True
        # Create a contact to use in tests
        self.contact_data = {"name": "Test User", "phone": "1234567890"}
        response = self.app.post('/contacts', json=self.contact_data, headers={'secret_key': 'AIzaSyAV8oQPuVDrC44gW9fOtSXZEYEW7JmVwLE'})
        self.contact_id = response.get_json()["id"]

    def test_create_contact(self):
        contact_data = {"name": "New User", "phone": "0987654321"}
        response = self.app.post('/contacts', json=contact_data, headers={'secret_key': 'AIzaSyAV8oQPuVDrC44gW9fOtSXZEYEW7JmVwLE'})
        self.assertEqual(response.status_code, 201)
        self.assertIn("id", json.loads(response.data))

    def test_get_all_contacts(self):
        response = self.app.get('/contacts', headers={'secret_key': 'AIzaSyAV8oQPuVDrC44gW9fOtSXZEYEW7JmVwLE'})
        self.assertEqual(response.status_code, 200)
        self.assertGreater(len(json.loads(response.data)), 0)

    def test_get_contact(self):
        response = self.app.get(f'/contacts/{self.contact_id}', headers={'secret_key': 'AIzaSyAV8oQPuVDrC44gW9fOtSXZEYEW7JmVwLE'})
        self.assertEqual(response.status_code, 200)
```

Result:

```
(base) kabdrakhman@MacBook-Pro-Altair Midterm % python -m unittest testing.unit-test

POST action logged successfully.
POST action logged successfully.
.POST action logged successfully.
DELETE action logged successfully.
.POST action logged successfully.
.POST action logged successfully.
.POST action logged successfully.
Failed to log PUT action.
.
----------------------------------------------------------------------
Ran 5 tests in 7.977s

OK
(base) kabdrakhman@MacBook-Pro-Altair Midterm %
```

Integration testing:

testing > integration-test.py

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

# Define your secret key
SECRET_KEY = "AIzaSyAV8oQPuVDrC44gW9fOtSXZEYEW7JmVwLE"

class PhonebookAppIntegrationTests(unittest.TestCase):
    def setUp(self):
        self.app = app.test_client()
        self.app.testing = True

        # Create a contact with the secret_key header
        response = self.app.post('/contacts',
                                 json={"name": "Test Name", "phone": "1234567890"},
                                 headers={'secret_key': SECRET_KEY})

        print(response.data)  # Debug print

        # Check if the response is successful
        if response.status_code == 201:  # 201 for created
            self.contact_id = response.get_json()["id"]
        else:
            self.contact_id = None  # Handle the case when the contact isn't created

    @requests_mock.Mocker()
    def test_create_contact_integration(self, mock_requests):
        mock_requests.post('https://inputvalidation-zqyj4gfxea-uc.a.run.app', json={}, status_code=200)
        mock_requests.post('https://logevents-zqyj4gfxea-uc.a.run.app', json={}, status_code=200)

        response = self.app.post('/contacts',
                                 json={"name": "Integration Test Name", "phone": "0987654321"},
                                 headers={'secret_key': SECRET_KEY})
        self.assertEqual(response.status_code, 201)  # 201 for created
        self.assertIn("id", json.loads(response.data))

    @requests_mock.Mocker()
    def test_get_contact_integration(self, mock_requests):
        response = self.app.get(f'/contacts/{self.contact_id}', headers={'secret_key': SECRET_KEY})
        self.assertEqual(response.status_code, 200)
        self.assertIn("name", json.loads(response.data))

    @requests_mock.Mocker()
    def test_update_contact_integration(self, mock_requests):
        mock_requests.post('https://inputvalidation-zqyj4gfxea-uc.a.run.app', json={}, status_code=200)
```

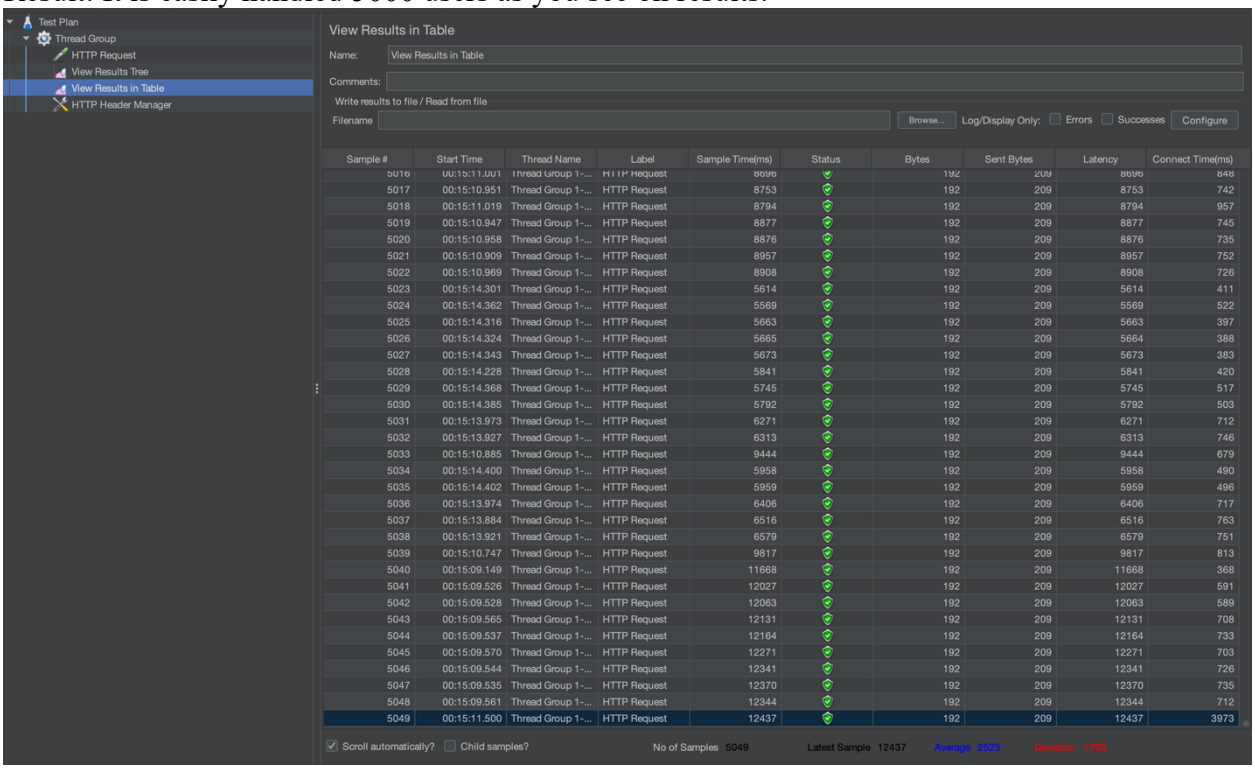
Ln 7, Col 54 Spaces:

Result:

```
(venv) (base) kabdrakhman@MacBook-Pro-Altair Midterm % python -m unittest testing.integration-test
POST action logged successfully.
b'{"id":1,"name":"Test Name","phone":"1234567890"}\n'
POST action logged successfully.
.POST action logged successfully.
b'{"id":3,"name":"Test Name","phone":"1234567890"}\n'
DELETE action logged successfully.
.POST action logged successfully.
b'{"id":4,"name":"Test Name","phone":"1234567890"}\n'
.POST action logged successfully.
b'{"id":5,"name":"Test Name","phone":"1234567890"}\n'
PUT action logged successfully.
.
----------------------------------------------------------------------
Ran 4 tests in 9.724s

OK
(venv) (base) kabdrakhman@MacBook-Pro-Altair Midterm %
```

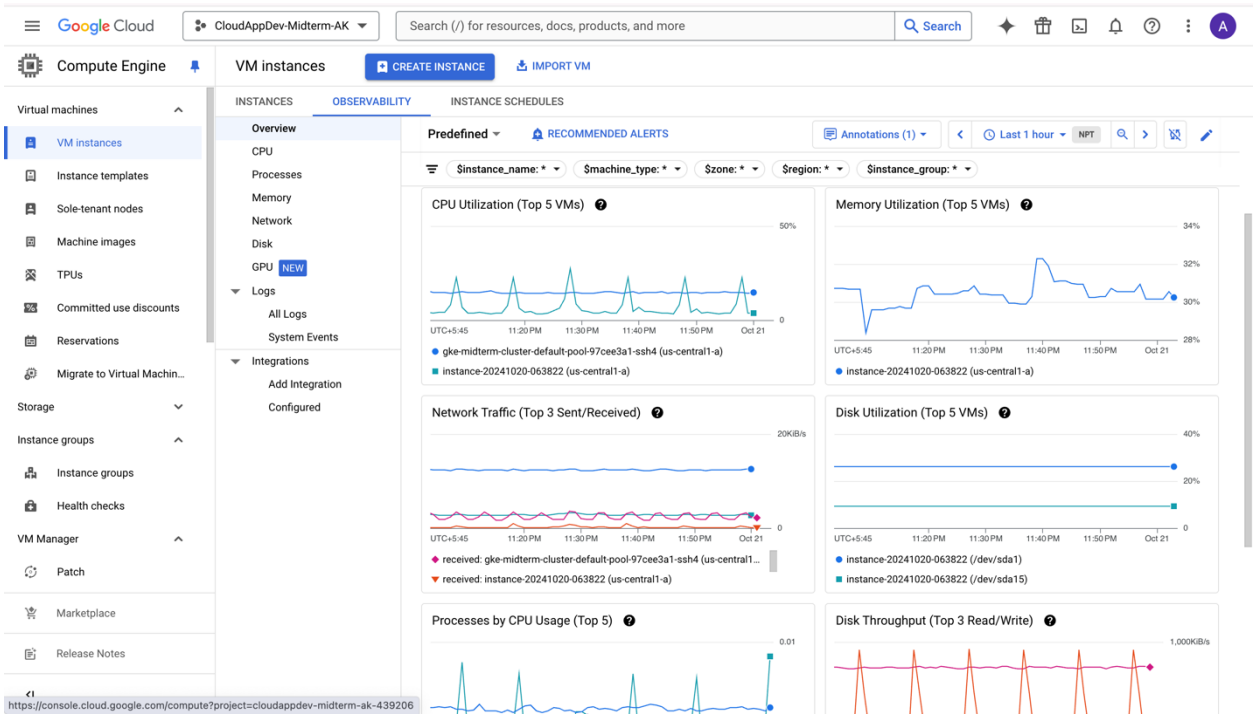
Outline the testing methodologies used, including unit tests, integration tests, and load testing results.

For load testing I used open source tool Jmeter with 5000 users request in 20 seconds.
Result: It is easily handled 5000 users as you see on results.

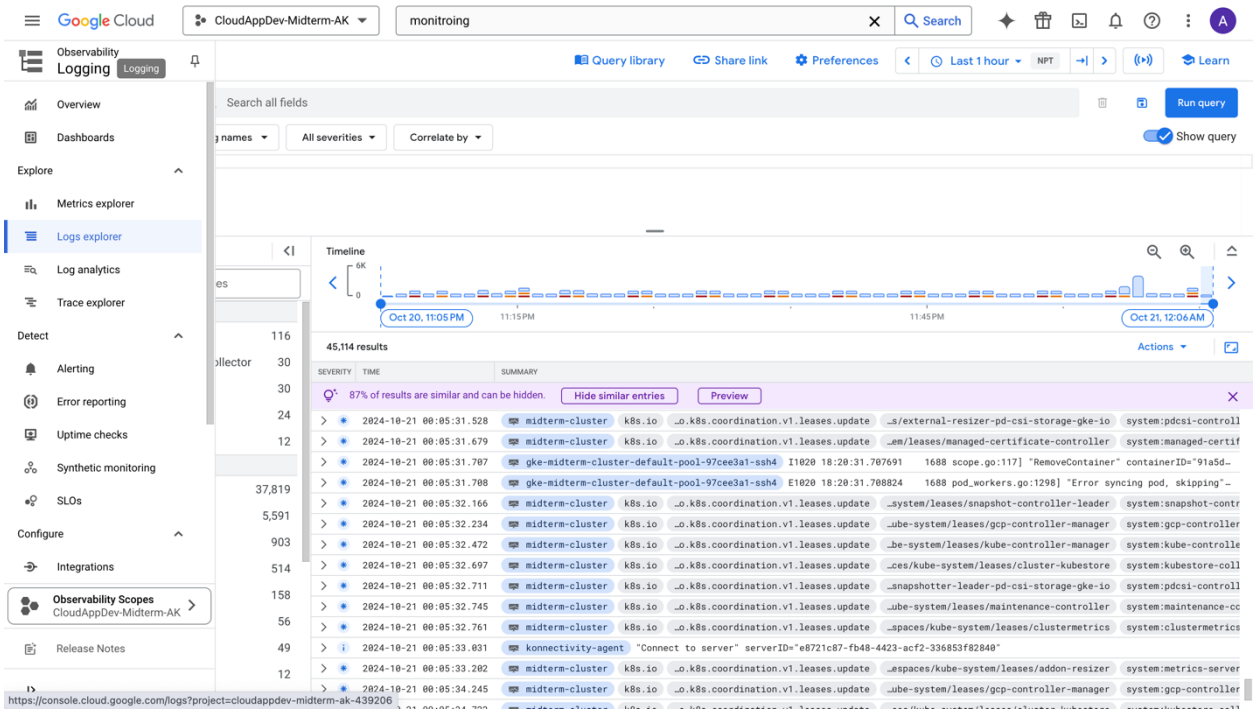


11. Monitoring and Maintenance

Google cloud already implemented monitoring features on their console.cloud.google.com website. As you can see here is example monitoring of compute engine



The most easy-to-use is observability and its logging where collected all logs from all cluster, vm intances, application and etc. Also it provides opportunity to create logging alerts.



12. Challenges and Solutions

During this midterm project I had several problems. Look at the most five problems:

- Forget to enable functions or apis/services
- Problem with deploying to Google Cloud App Engine. Solution: requirements.txt
- No access to Cloud Functions because I used another account accidently
- Working with GCR, didn't know about tagging as project
- With openapi to write host as project_id.appspot.com

13. Conclusion

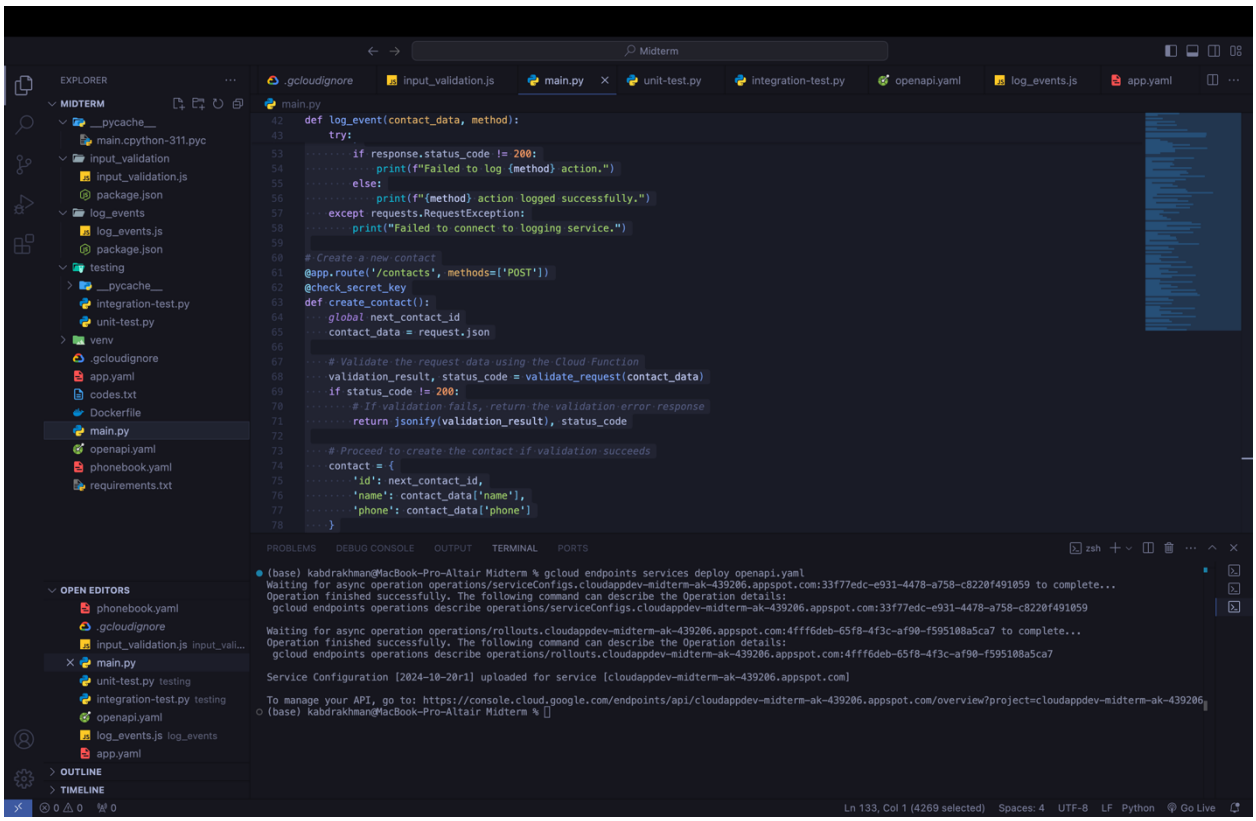
In this project, I developed a responsive web application using Google Cloud Platform (GCP), focusing on deploying a phonebook service. The architecture of the project includes core GCP services such as Cloud Run for serverless functions, Kubernetes for container orchestration, and Docker for application containerization. The application was built with a focus on security by implementing a header for API key validation. I utilized OpenAPI for defining the cloud API, which enhanced the documentation and usability of the service. During development, I faced challenges such as ensuring the application's scalability and high availability. To overcome these challenges, I conducted unit and integration testing, along with load testing to validate performance under various conditions. Overall, the project successfully demonstrated the effectiveness of the chosen technologies and met the objectives set for a robust cloud-based application.

14. References

1. Google Cloud SDK - https://cloud.google.com/sdk/docs/install
2. Google Compute Engine - https://cloud.google.com/products/compute?hl=ru
3. Cloud Run functions - https://cloud.google.com/functions?hl=ru
4. Docker Documentation - https://www.docker.com/
5. Google Kubernetes Engine (GKE) - https://cloud.google.com/kubernetes-engine/docs/concepts/kubernetes-engine-overview
6. Cloud Endpoints -https://cloud.google.com/endpoints/docs/openapi/get-started-app-engine
7. API keys - https://cloud.google.com/docs/authentication/api-keys
8. Jmeter - https://jmeter.apache.org/

15. Appendices

Project structure


Midterm
EXPLORER
MIDTERM
__pycache__
main.cpython-311.pyc
input_validation
input_validation.js
package.json
log_events
log_events.js
package.json
testing
__pycache__
integration-test.py
unit-test.py
venv
.gcloudignore
app.yaml
codes.txt
Dockerfile
main.py
openapi.yaml
phonebook.yaml
requirements.txt
OPEN EDITORS
phonebook.yaml
.gcloudignore
input_validation.js input_vali...
main.py
unit-test.py testing
integration-test.py testing
openapi.yaml
log_events.js log_events
app.yaml
OUTLINE
TIMELINE
.gcloudignore
input_validation.js
main.py
unit-test.py
integration-test.py
openapi.yaml
log_events.js
app.yaml
main.py
def log_event(contact_data, method):
try:
if response.status_code != 200:
print(f"Failed to log {method} action.")
else:
print(f"{method} action logged successfully.")
except requests.RequestException:
print("Failed to connect to logging service.")
# Create a new contact
@app.route('/contacts', methods=['POST'])
@check_secret_key
def create_contact():
global next_contact_id
contact_data = request.json
# Validate the request data using the Cloud Function
validation_result, status_code = validate_request(contact_data)
if status_code != 200:
# If validation fails, return the validation error response
return jsonify(validation_result), status_code
# Proceed to create the contact if validation succeeds
contact = {
'id': next_contact_id,
'name': contact_data['name'],
'phone': contact_data['phone']
}
PROBLEMS
DEBUG CONSOLE
OUTPUT
TERMINAL
PORTS
zsh
(base) kabdrakhman@MacBook-Pro-Altair Midterm % gcloud endpoints services deploy openapi.yaml
Waiting for async operation operations/serviceConfigs.cloudappdev-midterm-ak-439206.appspot.com:33f77edc-e931-4478-a758-c8220f491059 to complete...
Operation finished successfully. The following command can describe the Operation details:
gcloud endpoints operations describe operations/serviceConfigs.cloudappdev-midterm-ak-439206.appspot.com:33f77edc-e931-4478-a758-c8220f491059
Waiting for async operation operations/rollouts.cloudappdev-midterm-ak-439206.appspot.com:4fff6deb-65f8-4f3c-af90-f595108a5ca7 to complete...
Operation finished successfully. The following command can describe the Operation details:
gcloud endpoints operations describe operations/rollouts.cloudappdev-midterm-ak-439206.appspot.com:4fff6deb-65f8-4f3c-af90-f595108a5ca7
Service Configuration [2024-10-20r1] uploaded for service [cloudappdev-midterm-ak-439206.appspot.com]
To manage your API, go to: https://console.cloud.google.com/endpoints/api/cloudappdev-midterm-ak-439206.appspot.com/overview?project=cloudappdev-midterm-ak-439206
(base) kabdrakhman@MacBook-Pro-Altair Midterm %
Ln 133, Col 1 (4269 selected)
Spaces: 4
UTF-8
LF
Python
Go Live