Storage API Monitoring in Prometheus & Grafana with Dockerization and Kubernetes Cluster

Table of Contents

Step 1: Implementation

In the implementation part I have used Python's web application framework **flask** to develop storage-api. Here I am writing the steps that I have done to implement this.

- Modified the file `__init__.py` file and import the `prometheus_flask_exporter` to collect metrics HTTP request duration in seconds by path, method and status code for this storage api.
- Prepared a `Dockerfile` to dockerized the storage api project. In dockerfile I have provided instruction to install necessary tools/modules.

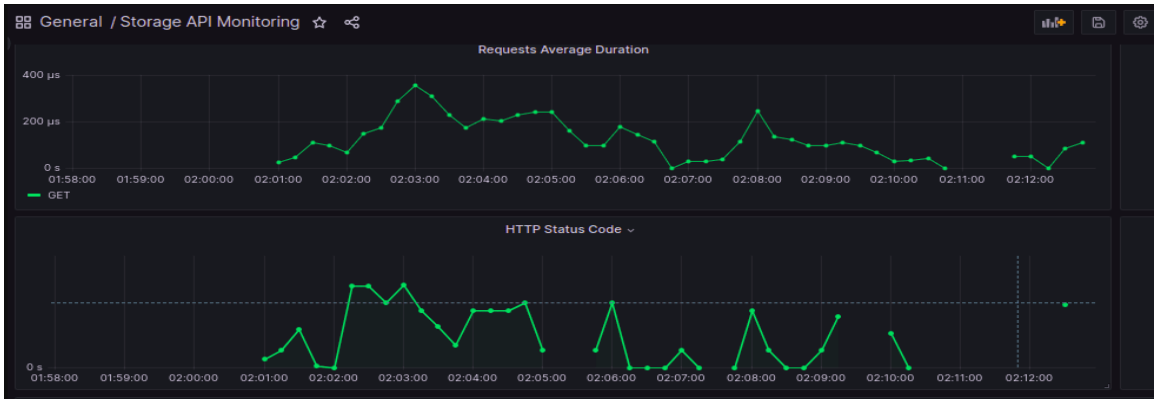
- Modified the `docker-compose.yml` file and added the necessary configuration for the storage-api to run the container. I have defined to run this storage-api container in 5000 port. The Prometheus Server expects it to run on `http://storage_api:5000`.

Step 2: Visualization

In the visualization part I have done this following steps.

- I have run `docker-compose up` command to run all of the container like storage-api, prometheus and grafana. By running the `docker ps` command we can see that all containers are running.
- The Prometheus is running on `http://localhost:9090` and Grafana is running on `http://localhost:3000`.
- I have prepared a grafana dashboard which will show two graphs for our Storage API
    - Average HTTP Request Duration
    - HTTP Status Codes

- I have attached the screenshot of the graph that I generated as described.



- Reason of getting HTTP 500 errors for some endpoints
    - In the `bucket.py` code file, The function checks if the ID exists in the "data" dictionary, which contains the bucket data. If the ID exists, the corresponding bucket is removed from the dictionary using the "pop()" method.
    - If the bucket is successfully deleted, the function returns an empty string with a status code of 500. This is incorrect, as a status code of 500

indicates an internal server error, but the bucket was successfully deleted, so a status code of 200 or 204 (No Content) would be more appropriate.
- Additionally, if the ID does not exist in the "data" dictionary, the function returns a JSON response with a status code of 400 (Bad Request). This is also incorrect, as a status code of 404 (Not Found) would be more appropriate, since the requested resource was not found.

Step 3: Deployment

In this step, I have prepared necessary deployment file for kubernetes cluster. Here I am writing the steps that I have done to deploy this in kubernetes cluster.

- For Storage API application, I have prepared a deployment kind file which name is `storage-api-deploy.yml` and a `NodePort` service file which name is `storage-api-svc.yml`.
- For Prometheus, I have prepared a deployment kind file which name is `prometheus-deploy.yml`, a `NodePort` service file which name is `prometheus-svc.yml` and a `ConfigMap` file for `prometheus.yml` which name is `prometheus-conf.yml`.
- For Grafana, I have prepared a deployment kind file which name is `grafana-deploy.yml` and a `NodePort` service file which name is `grafana-svc.yml`.
- All of the file I have attached in the main project files folder in zip format.

Step 4: Pros and Cons Using Docker & K8s in This Project

I have noticed some pros and cons during dockerizing this project and adding them kubernetes cluster. These are some pros and cons that I noticed.

Pros on Dockerization

- Docker container provides isolation between the application and the host system, which helps to improve security by preventing potential vulnerabilities in the application from affecting the underlying system.

- Docker containers provide a consistent environment for the application, ensuring that it runs the same way across different platforms and environments.
- Docker containers provide better control over the application environment, allowing you to specify the required dependencies and libraries needed to run the application.

Cons on Dockerization

- Docker containerization can add complexity to the deployment process, requiring additional steps and configuration.
- Docker containers require regular maintenance, such as updating the base image, to ensure that the application is secure.

Pros on Kubernetes Cluster

- Kubernetes makes it easy to scale the application horizontally by adding more containers to the cluster, which helps to improve security by distributing the workload and preventing single points of failure.
- Kubernetes provides fault-tolerance features such as self-healing and auto-restart, which helps to ensure that the application is always available.
- Kubernetes provides security policies that can be used to restrict access to the cluster and its resources, helping to improve the overall security of the application.
- Kubernetes provides an easy way to deploy and manage containers in a cluster, making it easy to update the application with the latest security patches and bug fixes.

Cons on Kubernetes Cluster

- Kubernetes can be complex to set up and maintain, requiring advanced knowledge of containerization and networking.
- Kubernetes can be resource-intensive, requiring additional resources to manage the cluster, which can increase costs.

The above descriptions are my observation during the time of doing this task. I have tried to fill up all requirements that asked on the assessment description.