SOFE 4790U: Distributed Systems

Date: October 23, 2022

Lab 3: Deploying a Circuit Breaking ambassador and a Function-as-a-Service (FaaS)

Name: Preet Patel
Student Id: 100708239

Learning Objectives:

In this lab, we learned how to create a Docker image and a service using NodeJS. We learned about FaaS. We configured and used OpenFaaS on GCP. We also learned about the Decorator Pattern.

Links:

Github: https://github.com/preetpatel87/Distributed-Systems-Group-14

Demonstration Links:

- **Part 2 and Part 3:** https://drive.google.com/file/d/136xWSjJeDXUDK5xoQ797JiRziEWZ3YVh/view?usp=sharing
- **Persistent Volume Example:** https://drive.google.com/file/d/11Ala06-1joj1DsWk8C4qVbllCumD302R/view?usp=sharing

Discussion:

Summarize the problem, the solution, and the requirements for the pattern given in part 1. Which of these requirements can be achieved by the procedures shown in parts 2 and 3?

It is a good practice to monitor the health of web applications to verify if the applications are functioning correctly. However, it can be difficult to monitor cloud-hosted applications because the control of the host environment is in the hands of the cloud provider. On the cloud, services can fail entirely or partially due to factors such as network latency, performance and availability of the host machine, and the network bandwidth between the compute and storage systems. To ensure that the service is performing at the required availability level, we must verify the service health at regular intervals.

To solve this problem, we can implement a health monitoring application which performs necessary checks by sending requests to the application endpoints. A health monitor checks the response to the request to a health verification endpoint on the application and analyzes results by the tool or framework which is performing the health check. The health monitor checks the status of the application from the response code and response contents from the request, the latency by the monitoring tool or framework, database or cloud storage for availability and response time, and other resources or services which are part of the application.

Requirements:

- Validate response code and contents
- Measure response time - combination of the network latency and response time
- Check resources or services location outside the application
- Check for expired SSL certificates
- Measure response time of DNS lookup to check DNS latency and failures
- Checks should run for multiple different endpoints
- Validate URL returned from DNS lookup
- Checks should be short and thorough

In Part 2 of this lab, we deploy a circuit breaker pattern which can prevent, mitigate, and manage failures. It allows failing services to recover, re-route traffic to alternative services, and limit rate to the service. Similar to the health monitoring service, the circuit breaker monitors for any failure in the system and if there is a failure it can also help the system recover. It prevents the service from being overloaded. It activates when a failure amount rises above the set threshold. It will prevent any requests to the service for n amount of seconds and then half open the service (reduced rate) to see if the service has recovered. This is also the same in part 3 where we can test the endpoint health by creating sync-requests.

Design:

Kubernetes provides persistent volumes. Why can such a feature be important? How to implement it? Provide an example in which persistent volumes are needed. Configure a YAML file to implement the example. Run it and test the creation of persistent volume and its ability to provide the required functionality within the example.

- **Video Demonstration:** https://drive.google.com/file/d/11Ala06-1joj1DsWk8C4qVbllCumD302R/view?usp=sharing

Formatted: Indent: Left: 1.27 cm, No bullets or numbering

Persistent volumes in kubernetes can be used to preserve data even after a pod has been deleted or after the pod fails. In GKE, we can dynamically provision persistent volumes on demand using Kubernetes persistent volume claim manifests. Persistent volume stores data at the cluster level and can be shared by multiple pods. Persistent volumes can be used to maintain stateful applications. It helps prevent loss of data if a pod is accidentally restarted or terminated. It makes it easier to handle backups, logs storage, performance, and storage capacity allocations. To implement we need to create a YAML file which configures a Persistent Volume Claim, apply the file, and finally validate that it is available. We can add volume mounts and volumes to our deployments to save data even after deployment deletion.

In the example, I deployed a wordpress application and a mysql application which the wordpress will use to store data. I deployed both the applications with their own persistent volumes. In the example, we first create a post on the wordpress titled persistent with some content. After the post is created, we delete the pods for both the applications and restart them. We ensure that the persistent volumes and their claims were not deleted accidentally. On restart, we reopen the wordpress application where we can see that the post that we created named persistent still exists. If we launched these applications without a persistent volume, on restart of the application pod we would not see the posts that we had created in the past.