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

Due: Oct 23, 2022

Ridwan Hossain (100747897)

Discussion

Problem

A general practice in enterprise software is to ensure the availability of a system's functionalities by monitoring the responsiveness of the applications and services. This practice becomes troublesome when applied to larger-scale applications which are established in cloud environments as there is lesser jurisdiction over the system relative services which are locally based. This is because adopting a cloud infrastructure introduces some factors which may result in complications within the system. Those factors are the latency and bandwidth of the communication network, and the reliability and performance capabilities of the major hardware components supporting the system.

Solution

A viable solution for ensuring the availability of a cloud-based system's services is by deploying health monitoring. Similar to a heartbeat pattern, the health monitoring will periodically check the performance and availability of a system's various endpoints by making requests to them. These requests can be used to check if a service is available, the responsiveness and latency from the service, and so on. There exists frameworks for performing these health monitoring activities for cloud-based applications, where various configurations may be established in order to perform specific performance and availability tests against a specified set of endpoints and services.

Requirements for the Pattern

In order to sufficiently implement this health monitoring pattern into a system, some requirements must be considered. Those requirements are:

- Ensuring that the endpoints being monitored are secure from public/malicious access and perform authentication checks.
- Ensure responses for requests made by the health monitoring are validated.
- Ensure that an appropriate number of endpoints are exposed, generally one where major functionalities are accessible and one for inconsequential services.
- Ensure that endpoints for general access are distinguished from ones used for health monitoring, or implement a specific pathway for performing the monitoring.
- Ensure that the type and amount of data to be retrieved from the health monitoring is specified.
- Ensure that the health monitoring periodically performs checks on itself in order to confirm that the checking system itself is performing correctly.

Which of these requirements can be achieved by the procedures shown in parts 2 and 3?

The procedures in parts 2 and 3 help to achieve the requirements in performing authentication checks on user requests, validating requests made for health monitoring, and ensuring that specific endpoints that are used for health monitoring are exposed and differentiated from endpoints used for general functionality.

Design

Persistent volumes in Kubernetes is essentially storage that has a life cycle that is separated from that of any pods that may utilize the volume. The reason this is important is because if an application where pods require access to data that was stored previously outside of the pod's life cycle, it would not be possible without the use of persistent volumes. In order to show a sample use of persistent volumes in kubernetes, the following source was used: https://kubernetes.io/docs/tasks/configure-pod-container/configure-persistent-volume-storage/. We first start by saving an html file with some text within it. We use the pv-volume.yaml file in order to create a persistent volume. In this case, we created a file to emulate some network-attached storage. This persistent storage contains 10 gigabytes of space and uses the ReadWriteOnce access mode. After mounting the persistent volume, we need to create a persistent volume claim. This was done in the pv-claim.yaml file. This claim will request 3 gigabytes of storage using the ReadWriteOnce access mode. After this, we need to actually deploy a pod that will use the persistent volume claim in order to use some of that persistent volume storage. Following this, we open a shell environment into the container running within the pod in order to ensure that it is serving the html file we created initially from the persistent volume. We see from the curl command that yes, it is.

```
ridwanhossain989@cs-980211405983-default:~/Lab 3 Design$ minikube ssh
docker@minikube:~$ sudo mkdir /mnt/data
docker@minikube:~$ sudo sh -c "echo 'Hello from Kubernetes storage' > /mnt/data/index.html"
docker@minikube:~$ cat /mnt/data/index.html
Hello from Kubernetes storage
docker@minikube:~$ exit
logout
ridwanhossain989@cs-980211405983-default:~/Lab 3 Design$ kubectl apply -f pv-volume.yaml
persistentvolume/task-pv-volume created
ridwanhossain989@cs-980211405983-default:~/Lab 3 Design$ kubectl get pv task-pv-volume
NAME             CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM   STORAGECLASS   REASON   AGE
task-pv-volume   10Gi       RWO            Retain           Available           manual                  10s
ridwanhossain989@cs-980211405983-default:~/Lab 3 Design$ kubectl apply -f pv-claim.yaml
persistentvolumeclaim/task-pv-claim created
ridwanhossain989@cs-980211405983-default:~/Lab 3 Design$ kubectl get pv task-pv-volume
NAME             CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM                   STORAGECLASS   REASON   AGE
task-pv-volume   10Gi       RWO            Retain           Bound    default/task-pv-claim   manual                  39s
ridwanhossain989@cs-980211405983-default:~/Lab 3 Design$ kubectl get pvc task-pv-claim
NAME            STATUS   VOLUME           CAPACITY   ACCESS MODES   STORAGECLASS   AGE
task-pv-claim   Bound    task-pv-volume   10Gi       RWO            manual         26s
ridwanhossain989@cs-980211405983-default:~/Lab 3 Design$ kubectl apply -f pv-pod.yaml
\pod/task-pv-pod created
```

```
ridwanhossain989@cs-980211405983-default:~/Lab 3 Design$ kubectl get pod task-pv-pod
NAME          READY   STATUS    RESTARTS   AGE
task-pv-pod   1/1     Running   0          15s
ridwanhossain989@cs-980211405983-default:~/Lab 3 Design$ kubectl exec -it task-pv-pod -- /bin/bash
root@task-pv-pod:/# apt update
Get:1 http://deb.debian.org/debian bullseye InRelease [116 kB]
Get:2 http://deb.debian.org/debian-security bullseye-security InRelease [48.4 kB]
Get:3 http://deb.debian.org/debian bullseye-updates InRelease [44.1 kB]
Get:4 http://deb.debian.org/debian bullseye/main amd64 Packages [8184 kB]
Get:5 http://deb.debian.org/debian-security bullseye-security/main amd64 Packages [191 kB]
Get:6 http://deb.debian.org/debian bullseye-updates/main amd64 Packages [14.6 kB]
Fetched 8598 kB in 1s (6758 kB/s)
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
3 packages can be upgraded. Run 'apt list --upgradable' to see them.
root@task-pv-pod:/# apt install curl
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
curl is already the newest version (7.74.0-1.3+deb11u3).
0 upgraded, 0 newly installed, 0 to remove and 3 not upgraded.
root@task-pv-pod:/# curl http://localhost/
Hello from Kubernetes storage
root@task-pv-pod:/# exit
exit
```