Faculty of Engineering and Applied Science

SOFE 4790U Distributed Systems

Lab #3

CRN: 43525

Date: 2022-10-09

Name	Student #
Owen Musselman	100657709

Objective:

- Learn how to create a service using NodeJS
- Learn how to create a Docker image
- Learn how to configure and use a circuit breaker using Nginx
- Get familiar with FaaS
- Learn how to configure and use OpenFaaS on GCP
- Get familiar with Decorator Pattern

Part 1:

The problem is that distributed applications need to be examined to see if they are working properly. This is known as health monitoring in this pattern. Through the use of health monitoring the checks are done with amalgamating two attributes. The first being the health checks that are done when requested by the health verification endpoint. The second being the analysis of what is returned by the check. Also assisting in the solution of this problem is there are tools and services that assist in auditing these web applications, and it is simple to make these tools/services endpoints that have the job to check the functionality of the system. Below is a list of checks that these services/tools perform:

- **Measuring response times:** The sum of the network latency, and the time that the service or application performs the request. If the sum is high, then there could be a potential problem on the network side (load balancing, traffic), or unoptimized usage of message transfer protocols.
- **Validating the response code:** Ensure that the correct response code is returning like 200 for OK.
- **Checking the content of the response:** Checking if the returned value of the response code 200 (OK) or something else. Also if the value is 20 it will still check other aspects of the page that will verify if the returned webpage is correct like a test phrase on the page.
- **Checking resources/services:** Checks the content delivery NW that is utilized to send content from global caches.
- **Checking expiration of SSL certs:** Checks if the SSL certificate is expired or not.
- **Measuring response time of DNS lookup:** Looks up the URL for the service/application being used to see if there is DNS failure, or check the response time.
- **Validating URL from the DNS lookup response:** To stop malicious redirections of requests if there is an attack aimed at the DNS server.

The Above checks also are requirements for the system to assure that it is working as intended.

Part 2:

```
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$ git clone https://github.com/GeorgeDaoud3/SOFE4790U-lab3.git
Cloning into 'SOFE4790U-lab3'...
remote: Enumerating objects: 27, done.
remote: Counting objects: 100% (27/27), done.
remote: Compressing objects: 100% (20/20), done.
remote: Total 27 (delta 4), reused 24 (delta 4), pack-reused 0
Receiving objects: 100% (27/27), 5.77 KiB | 1.92 MiB/s, done.
Resolving deltas: 100% (4/4), done.
```

Cloning the lab repository.

```
slikcaustic1@cloudshell:~/SOFE4790U-lab3/part2 (bubbly-granite-362015)$ cd DummyServiceContainer/
slikcaustic1@cloudshell:~/SOFE4790U-lab3/part2/DummyServiceContainer (bubbly-granite-362015)$
```

Changing to the DummyServiceContainer directory from the cloned repository.

```
slikcaustic1@cloudshell:~/SOFE4790U-lab3/part2/DummyServiceContainer (bubbly-granite-362015)$ docker build . -t us.gcr.io/bubbly-granite-362015/dummyservice
Sending build context to Docker daemon  6.656kB
Step 1/7 : FROM node:carbon
carbon: Pulling from library/node
146bd6a88618: Pull complete
9935d0c62ace: Pull complete
db0efb86e806: Pull complete
e705a4c4fd31: Pull complete
c877b722db6f: Pull complete
645c20ec8214: Pull complete
db8fbd9db2fe: Pull complete
1c151cd1b3ea: Pull complete
fbd993995f40: Pull complete
Digest: sha256:a681bf74805b80d03eb21a6c0ef168a976108a287a74167ab593fc953aac34df
Status: Downloaded newer image for node:carbon
 ---> 8eeadf3757f4
Step 2/7 : WORKDIR /usr/src/app
 ---> Running in 09f1ae185248
Removing intermediate container 09f1ae185248
 ---> 27784e84298b
Step 3/7 : COPY package*.json ./
 ---> 5a9a6e5a3cc5
Step 4/7 : RUN npm install
 ---> Running in 64231dbe6e5f
```

Creating docker image.

```
slikcaustic1@cloudshell:~/SOFE4790U-lab3/part2/DummyServiceContainer (bubbly-granite-362015)$ docker push us.gcr.io/bubbly-granite-362015/dummyservice
Using default tag: latest
The push refers to repository [us.gcr.io/bubbly-granite-362015/dummyservice]
c54aabad4fd7: Pushed
b589f5049c60: Pushed
20c342325374: Pushed
4fdcb28002fd: Pushed
423451ed44f2: Layer already exists
b2aaf85d6633: Layer already exists
88601a85ce11: Layer already exists
42f9c2f9c08e: Layer already exists
99e8bd3efaaf: Layer already exists
bee1e39d7c3a: Layer already exists
1f59a4b2e206: Layer already exists
0ca7f54856c0: Layer already exists
ebb9ae013834: Layer already exists
latest: digest: sha256:0391b559fe8363eb9f941f987b1c07641877e5d076fa667ff1956115db0376a0 size: 3048
```

Pushing docker image to docker hub.

```
20        image: us.gcr.io/bubbly-granite-362015/dummyservice
```

Setting the image in dummy-deployment.yaml

```
slikcaustic1@cloudshell:~/SOFE4790U-lab3/part2 (bubbly-granite-362015)$ kubectl create -f dummy-deployment.yaml
deployment.apps/dummy-deployment created
```

Creating the deployment that contains the image that was previously created.

```
slikcaustic1@cloudshell:~/SOFE4790U-lab3/part2 (bubbly-granite-362015)$ kubectl expose deployment dummy-deployment --port=80 --type=LoadBalancer --name dummy-deployment
service/dummy-deployment exposed
```

Exposing dummy-deployment using loadbalancer.

```
slikcaustic1@cloudshell:~/SOFE4790U-lab3/part2 (bubbly-granite-362015)$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
dummy-deployment-695cbd746d-zlkx2   1/1     Running   0          4m12s
mongodb-deployment-7945646c67-r94d6   1/1     Running   0          19d
mysql-deployment-5496fdc956-8g9rw     1/1     Running   0          19d
```

Showing dummy pods are created.

```
slikcaustic1@cloudshell:~/SOFE4790U-lab3/part2 (bubbly-granite-362015)$ kubectl get deployments
NAME               READY   UP-TO-DATE   AVAILABLE   AGE
dummy-deployment   1/1     1            1           4m19s
mongodb-deployment   1/1     1            1           34d
mysql-deployment   1/1     1            1           39d
```

Showing dummy deployments are created.

```
slikcaustic1@cloudshell:~/SOFE4790U-lab3/part2 (bubbly-granite-362015)$ kubectl get services
NAME               TYPE           CLUSTER-IP      EXTERNAL-IP     PORT(S)          AGE
dummy-deployment   LoadBalancer   10.108.5.125    35.234.244.62   80:32704/TCP     6m30s
kubernetes         ClusterIP      10.108.0.1      <none>          443/TCP          41d
mongodb-service    LoadBalancer   10.108.13.162   35.234.248.86   3306:30766/TCP   34d
mysql-service      LoadBalancer   10.108.8.33     34.152.15.114   3306:31301/TCP   39d
```

Showing dummy services were created.

```
20            image: us.gcr.io/bubbly-granite-362015/dummyservice
```

Editing the image to be used in backup-deployment.yaml

```
slikcaustic1@cloudshell:~/SOFE4790U-lab3/part2 (bubbly-granite-362015)$ kubectl create -f backup-deployment.yaml
deployment.apps/backup-deployment created
```

Create backup-deployment.

```
slikcaustic1@cloudshell:~/SOFE4790U-lab3/part2 (bubbly-granite-362015)$ kubectl expose deployment backup-deployment --port=80 --type=LoadBalancer --name backup-deployment
service/backup-deployment exposed
```

Expose the backup-deployment with loadbalancer.

```
slikcaustic1@cloudshell:~/SOFE4790U-lab3/part2 (bubbly-granite-362015)$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
backup-deployment-56ffcb8fd8-cz7vs    1/1     Running   0          4m55s
dummy-deployment-695cbd746d-zlkx2   1/1     Running   0          18m
mongodb-deployment-7945646c67-r94d6   1/1     Running   0          19d
mysql-deployment-5496fdc956-8g9rw     1/1     Running   0          19d
```

Showing the backup pods was created.

```
slikcaustic1@cloudshell:~/SOFE4790U-lab3/part2 (bubbly-granite-362015)$ kubectl get deployments
NAME               READY   UP-TO-DATE   AVAILABLE   AGE
backup-deployment    1/1     1            1           5m6s
dummy-deployment   1/1     1            1           18m
mongodb-deployment   1/1     1            1           34d
mysql-deployment   1/1     1            1           39d
```

Showing the backup deployment was created.

```
slikcaustic1@cloudshell:~/SOFE4790U-lab3/part2 (bubbly-granite-362015)$ kubectl get services
NAME               TYPE           CLUSTER-IP      EXTERNAL-IP     PORT(S)          AGE
backup-deployment    LoadBalancer   10.108.6.34     34.152.63.218   80:30637/TCP     4m16s
dummy-deployment   LoadBalancer   10.108.5.125    35.234.244.62   80:32704/TCP     15m
kubernetes         ClusterIP      10.108.0.1      <none>          443/TCP          41d
mongodb-service    LoadBalancer   10.108.13.162   35.234.248.86   3306:30766/TCP   34d
mysql-service      LoadBalancer   10.108.8.33     34.152.15.114   3306:31301/TCP   39d
```

Showing the backup service was created.

```
slikcaustic1@cloudshell:~/SOFE4790U-lab3/part2 (bubbly-granite-362015)$ kubectl create -f nginx-configmap.yaml
configmap/nginx-configuration created
```

Creating the configmap.

```
slikcaustic1@cloudshell:~/SOFE4790U-lab3/part2 (bubbly-granite-362015)$ kubectl create -f circuitbreaker.yaml
deployment.apps/circuitbreaker created
service/circuitbreaker created
```

Creating the circuit breaker deployment and service.

```
slikcaustic1@cloudshell:~/SOFE4790U-lab3/part2 (bubbly-granite-362015)$ kubectl get services
NAME                TYPE           CLUSTER-IP      EXTERNAL-IP     PORT(S)          AGE
backup-deployment   LoadBalancer   10.108.6.34     34.152.63.218   80:30637/TCP     8m29s
circuitbreaker      LoadBalancer   10.108.13.128   35.203.71.168   80:31993/TCP     103s
dummy-deployment    LoadBalancer   10.108.5.125    35.234.244.62   80:32704/TCP     19m
kubernetes          ClusterIP      10.108.0.1      <none>          443/TCP          41d
mongodb-service     LoadBalancer   10.108.13.162   35.234.248.86   3306:30766/TCP   34d
mysql-service       LoadBalancer   10.108.8.33     34.152.15.114   3306:31301/TCP   39d
```

Getting the external IP of the circuitbreaker (35.203.71.168)

```
slikcaustic1@cloudshell:~/SOFE4790U-lab3/part2 (bubbly-granite-362015)$ curl -v http://35.203.71.168
*   Trying 35.203.71.168:80...
* Connected to 35.203.71.168 (35.203.71.168) port 80 (#0)
> GET / HTTP/1.1
> Host: 35.203.71.168
> User-Agent: curl/7.74.0
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< Server: nginx/1.13.7
< Date: Fri, 21 Oct 2022 01:01:54 GMT
< Content-Type: text/html; charset=utf-8
< Content-Length: 29
< Connection: keep-alive
< X-Powered-By: Express
< ETag: W/"1d-Cvi4umLI3rerUQoAYYxO8GMsv2w"
<
* Connection #0 to host 35.203.71.168 left intact
SOMERESPONSE FROM 10.104.1.23slikcaustic1@cloudshell:~/SOFE4790U-lab3/part2 (bubbly-granite-362015)$
```

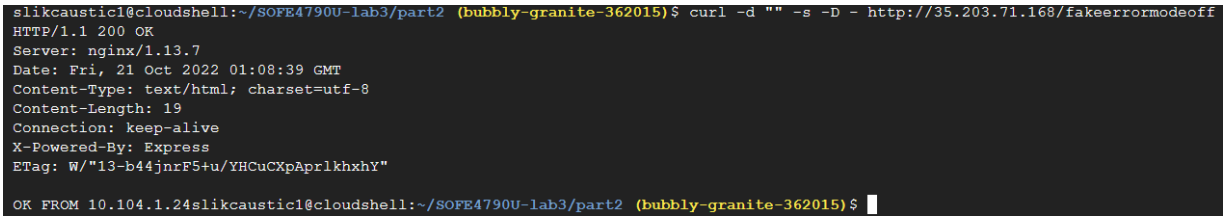
Testing the circuit breaker and looking at the ip of the local machine.

```
SOMERESPONSE FROM 10.104.1.23slikcaustic1@cloudshell:~/SOFE4790U-lab3/part2 (bubbly-granite-362015)$ curl -d "" -s -D - http://35.203.71.168/fakeerrormodeon
HTTP/1.1 200 OK
Server: nginx/1.13.7
Date: Fri, 21 Oct 2022 01:03:47 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 19
Connection: keep-alive
X-Powered-By: Express
ETag: W/"13-4VHxHCNFxsLVz0uMznou/JYPrAM"

OK FROM 10.104.1.23slikcaustic1@cloudshell:~/SOFE4790U-lab3/part2 (bubbly-granite-362015)$ curl -v http://35.203.71.168
```

Executing a mimicked error. Local ip still ends in 23.

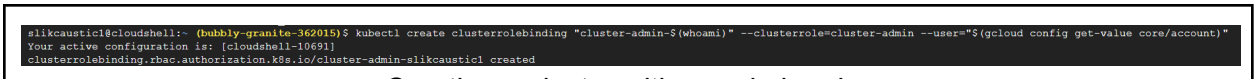
After executing the mimicked error we test the circuit breaker by using the first curl command and can see that the local ip is now ending in 24, when it was previously 24. This indicates the circuit breaker is functioning properly.



Disabling the error (resetting it). The ip of the local machine ends in 24.

Checking the local ip after resetting and the ip is still ending in 24, as that line was working it will continue to work.

Part 3:



Creating a cluster with an admin role.

```
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$ curl -SLsf https://dl.get-arkade.dev/ | sudo sh
arkade install openfaas --load-balancer
x86_64
Downloading package https://github.com/alexellis/arkade/releases/download/0.8.47/arkade as /tmp/arkade
Download complete.
```

Getting openfaas.

```
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$ curl -SLsf https://cli.openfaas.com | sudo sh
Finding latest version from GitHub
0.14.11
Downloading package https://github.com/openfaas/faas-cli/releases/download/0.14.11/faas-cli as /tmp/faas-cli
Download complete.

Running with sufficient permissions to attempt to move faas-cli to /usr/local/bin
New version of faas-cli installed to /usr/local/bin
Creating alias 'faas' for 'faas-cli'.
 ___                   _____           ____
/ _ \ _ __   ___ _ __ |  ___|_ _  __ _/ ___|
| | | | '_ \ / _ \ '_ \| |_ / _` |/ _` \___ \
| |_| | |_) |  __/ | | |  _| (_| | (_| |___) |
 \___/| .__/ \___|_| |_|_|  \__,_|\__,_|____/
      |_|

CLI:
 commit:  8820d8e4a15dab900d8a7e8fc271851ccb94012e
 version: 0.14.11
```

Getting openfaas client.

```
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$ kubectl -n openfaas get deployments -l "release=openfaas, app=openfaas"
NAME                READY   UP-TO-DATE   AVAILABLE   AGE
alertmanager        1/1     1            1           3m10s
basic-auth-plugin   1/1     1            1           3m10s
gateway             1/1     1            1           3m10s
nats                1/1     1            1           3m10s
prometheus          1/1     1            1           3m10s
queue-worker        1/1     1            1           3m10s
```

Verifying that openfaas is running.

```
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$ kubectl rollout status -n openfaas deploy/gateway
deployment "gateway" successfully rolled out
```

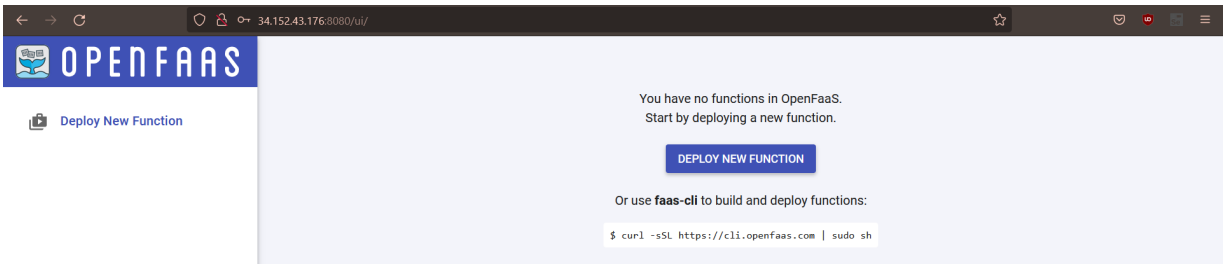
Checking that openfaas has been rolled out.

```
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$ kubectl get svc -o wide gateway-external -n openfaas
NAME               TYPE           CLUSTER-IP     EXTERNAL-IP     PORT(S)          AGE     SELECTOR
gateway-external   LoadBalancer   10.108.6.226   34.152.43.176   8080:32696/TCP   7m58s   app=gateway
```

Getting the openfaas external IP: 34.152.43.176

```
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$ export OPENFAAS_URL="34.152.43.176:8080"
PASSWORD=$(kubectl get secret -n openfaas basic-auth -o jsonpath="{.data.basic-auth-password}" | base64 --decode; echo)
echo $PASSWORD
echo -n $PASSWORD | faas-cli login --username admin --password-stdin
5SmUu8xj0NS6zLc0bO466E6P6
Calling the OpenFaaS server to validate the credentials...
WARNING! You are not using an encrypted connection to the gateway, consider using HTTPS.
credentials saved for admin http://34.152.43.176:8080
```

Getting the password and logging into the gateway. Password is: 5SmUu8xj0NS6zLc0bO466E6P6



Logging into openfaas through the browser using the 34.152.43.176:8080/ui/

```
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$ mkdir ~/OpenFaaS
cd ~/OpenFaaS
faas-cli template pull
faas-cli new --list
Fetch templates from repository: https://github.com/openfaas/templates.git at
2022/10/22 21:04:53 Attempting to expand templates from https://github.com/openfaas/templates.git
2022/10/22 21:04:53 Fetched 17 template(s) : [csharp dockerfile go java11 java11-vert-x node node12 node12-debian node14 node16 node17 php7 php8 python python3 python3-debian ruby] from https:
//github.com/openfaas/templates.git
Languages available as templates:
- csharp
- dockerfile
- go
- java11
- java11-vert-x
- node
- node12
- node12-debian
- node14
- node16
- node17
- php7
- php8
- python
- python3
- python3-debian
- ruby
```

Creating directory and pulling templates.

```
slikcaustic1@cloudshell:~/OpenFaaS (bubbly-granite-362015)$ faas-cli new --lang node12 --prefix us.gcr.io/bubbly-granite-362015 main
Folder: main created.
  ___                   _____           ____
 / _ \ _ __   ___ _ __ |  ___|_ _  __ _/ ___|
| | | | '_ \ / _ \ '_ \| |_ / _` |/ _` \___ \
| |_| | |_) |  __/ | | |  _| (_| | (_| |___) |
 \___/| .__/ \___|_| |_|_|  \__,_|\__,_|____/
      |_|


Function created in folder: main
Stack file written: main.yml

Notes:
You have created a new function which uses Node.js 12.

npm i --save can be used to add third-party packages like request or cheerio
npm documentation: https://docs.npmjs.com/

Unit tests are run at build time via "npm run", edit package.json to specify
how you want to execute them.
```

Creating an empty nodeJS function.

OpenFaaS > main > handler.js > ...

```
1  'use strict'
2
3  module.exports = async (event, context) => {
4      var parameters=JSON.stringify(event.body)
5    return context
6      .status(200)
7      .succeed(parameters)
8  }
```

Updating handler.js.

```
slikcaustic1@cloudshell:~/OpenFaaS (bubbly-granite-362015)$ cd ~/OpenFaaS
faas-cli build -f main.yml
[0] > Building main.
Image: us.gcr.io/bubbly-granite-362015/main:latest built.
[0] < Building main done in 21.54s.
[0] Worker done.

Total build time: 21.54s
slikcaustic1@cloudshell:~/OpenFaaS (bubbly-granite-362015)$
```

Building docker image

```
slikcaustic1@cloudshell:~/OpenFaaS (bubbly-granite-362015)$ docker push us.gcr.io/bubbly-granite-362015/main
Using default tag: latest
The push refers to repository [us.gcr.io/bubbly-granite-362015/main]
bebb255a5b67: Pushed
8e9fd02b7432: Pushed
49ed69b379c6: Pushed
ff159fd1cd47: Pushed
1b83c76eb93a: Pushed
236f14b62181: Pushed
2de17cee83f1: Pushed
5149a82bab67: Pushed
d86bda25f4c1: Pushed
2d1fd3cd4c03: Pushed
b2844da8506c: Pushed
7f30cde3f699: Layer already exists
fe810f5902cc: Layer already exists
dfd8c046c602: Layer already exists
4fc242d58285: Layer already exists
latest: digest: sha256:518b8d4906714eea97df8cbb72ded12eeff7beec1b262e46bc30c87b2b39b4a2 size: 3659
```

Pushing the created image to the registry.

```
slikcaustic1@cloudshell:~/OpenFaaS (bubbly-granite-362015)$ faas-cli deploy -f main.yml
Deploying: main.
WARNING! You are not using an encrypted connection to the gateway, consider using HTTPS.

Deployed. 202 Accepted.
URL: http://34.152.43.176:8080/function/main
```

Deploying the image to openfaas.

```
slikcaustic1@cloudshell:~/OpenFaaS (bubbly-granite-362015)$ curl http://34.152.43.176:8080/function/main -H 'Content-Type: application/json' -d '{ "Name": "Square", "Color": "Red", "Dimensions": 2 }'
{"Name":"Square","Color":"Red","Dimensions":2}slikcaustic1@cloudshell:~/OpenFaaS (bubbly-granite-362015)$
```

Sending a JSON object, and that same JSON object is echoed back.

```
slikcaustic1@cloudshell:~/OpenFaaS (bubbly-granite-362015)$ cd ~/OpenFaaS
faas-cli new --lang node12 --prefix us.gcr.io/bubbly-granite-362015 decorator
Folder: decorator created.
```

```
Function created in folder: decorator
Stack file written: decorator.yml

Notes:
You have created a new function which uses Node.js 12.

npm i --save can be used to add third-party packages like request or cheerio
npm documentation: https://docs.npmjs.com/

Unit tests are run at build time via "npm run", edit package.json to specify
how you want to execute them.
```

Creating a new empty nodeJS function.

```
const request = require('sync-request');

module.exports = async (event, context) => {
    var obj = event.body;
    if (obj['Name'] === undefined) {
        obj['Name'] = 'Nameless';
    }
    if (obj['Color'] === undefined) {
        obj['Color'] = 'Transparent';
    }
    var res = request('POST', 'http://34.152.43.176:8080/function/main', {

        body: JSON.stringify(obj)

      });

      console.log(res["body"].toString())
    return context.status(200).succeed(res["body"].toString('utf8', 1, res["body"].length-1).replace(/\\\"/g,'\"'));
```

Updating the handler.js file in the decorator directory.

```
{
    "name": "openfaas-function",
    "version": "1.0.0",
    "description": "OpenFaaS Function",
    "main": "handler.js",
    ▷ Debug
    "scripts": {
      "test": "echo \"Error: no test specified\" && exit 0"
    },
    "keywords": [],
    "author": "OpenFaaS Ltd",
    "license": "MIT",
    "dependencies": {
      "sync-request": "^6.1.0"
    }
  }
```

Adding sync requests to package.json.

```
slikcaustic1@cloudshell:~/OpenFaaS (bubbly-granite-362015)$ cd ~/OpenFaaS
faas-cli build -f decorator.yml
[0] > Building decorator.
```

```
Image: us.gcr.io/bubbly-granite-362015/decorator:latest built.
[0] < Building decorator done in 12.11s.
[0] Worker done.

Total build time: 12.11s
```

Building the decorator docker image.

```
slikcaustic1@cloudshell:~/OpenFaaS (bubbly-granite-362015)$ docker push us.gcr.io/bubbly-granite-362015/decorator
Using default tag: latest
The push refers to repository [us.gcr.io/bubbly-granite-362015/decorator]
6245ae223a34: Pushed
49daf6f97ccf: Pushed
5013d0f17a90: Pushed
3326f6520e9b: Pushed
1b83c76eb93a: Layer already exists
236f14b62181: Layer already exists
2de17cee83f1: Layer already exists
5149a82bab67: Layer already exists
d86bda25f4c1: Layer already exists
2d1fd3cd4c03: Layer already exists
b2844da8506c: Layer already exists
7f30cde3f699: Layer already exists
fe810f5902cc: Layer already exists
dfd8c046c602: Layer already exists
4fc242d58285: Layer already exists
latest: digest: sha256:a8d28b3c5138b9f697c1121413aaf8f5b38ecc4f6063ad1f0fd8f3e63fe934e5 size: 3663
```

Pushing the image to the registry.

```
slikcaustic1@cloudshell:~/OpenFaaS (bubbly-granite-362015)$ faas-cli deploy -f decorator.yml
Deploying: decorator.
WARNING! You are not using an encrypted connection to the gateway, consider using HTTPS.

Deployed. 202 Accepted.
URL: http://34.152.43.176:8080/function/decorator
```

Deploying the decorator to openfaas.

```
slikcaustic1@cloudshell:~/OpenFaaS (bubbly-granite-362015)$ curl http://34.152.43.176:8080/function/decorator -H 'Content-Type: application/json' -d '{ "Name": "Square", "Dimensions": 2 }'
{"Name":"Square","Dimensions":2,"Color":"Transparent"}slikcaustic1@cloudshell:~/OpenFaaS (bubbly-granite-362015)$
```

Now when a JSON object with the attributes of shape, and dimensions, the response will now contain a 3rd attribute of colour. The output reflects this and is functioning correctly.

Discussion:

The problem is that distributed applications need to be examined to see if they are working properly. This is known as health monitoring in this pattern. Through the use of health monitoring the checks are done with amalgamating two attributes. The first being the health checks that are done when requested by the health verification endpoint. The second being the analysis of what is returned by the check. Also assisting in the solution of this problem is there are tools and services that assist in auditing these web applications, and it is simple to make these tools/services endpoints that have the job to check the functionality of the system. Below is a list of checks that these services/tools perform:

- **Measuring response times:** The sum of the network latency, and the time that the service or application performs the request. If the sum is high, then there could be a potential problem on the network side (load balancing, traffic), or unoptimized usage of message transfer protocols.
- **Validating the response code:** Ensure that the correct response code is returning like 200 for OK.
- **Checking the content of the response:** Checking if the returned value of the response code 200 (OK) or something else. Also if the value is 20 it will still check other aspects of the page that will verify if the returned webpage is correct like a test phrase on the page.
- **Checking resources/services:** Checks the content delivery NW that is utilized to send content from global caches.
- **Checking expiration of SSL certs:** Checks if the SSL certificate is expired or not.
- **Measuring response time of DNS lookup:** Looks up the URL for the service/application being used to see if there is DNS failure, or check the response time.
- **Validating URL from the DNS lookup response:** To stop malicious redirections of requests if there is an attack aimed at the DNS server.

The Above checks also are requirements for the system to assure that it is working as intended.

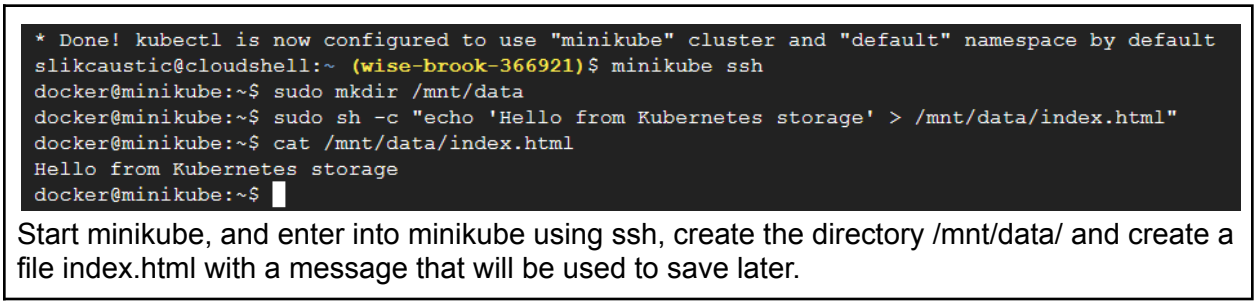
The problem is that distributed applications and services need to be analyzed to see if they are working properly. This is done through health monitoring. The tools/services assist in the auditing process of these applications, and it is simple to create these tools/services. But in order to create these tools there are some requirements that need to be followed. Like Measuring response times, validating the response codes, checking the content of the

response, checking the resources/services, checking the expiration of SSL certificates, measuring the response time of DNS lookup, and validating URL from the DNS lookup.

In part 2 the requirements are followed as well through the use of checking if there are errors in the system using endpoints created in the server.js. The errors in the system are detected using these endpoints and the requests are sent elsewhere, where they can be dealt with instead of bombarding the erroneous part indefinitely. Some of the requirements are used in part 3 where the handler for both the decorator and the main, are sending a successful response code when the operation is done.

Design:

Persistent volumes are an important feature as if persistent volumes are not implemented, any of the data that has been created, and or modified when the container is running will be lost. But with the use of K8s persistent volumes (PV), data to be stored after the container is no longer running. PVs allow storage to be mounted to a K8, which additionally allows the data to be shared among different nodes. PVs are running for as long as operations are conducted, which can outlive the container. PVs can be implemented through the use of PersistentVolume which uses storage classes, which are like a resource that is a part of the cluster. PersistentVolumeClaim is also used which will request storage by a user in need. The PersistentVolumeClaim uses PersistentVolume resources like RAM, and CPU attention; they can also be mounted with ReadWriteMany, ReadWriteOnce.



Start minikube, and enter into minikube using ssh, create the directory /mnt/data/ and create a file index.html with a message that will be used to save later.

pvvolume.yaml > ...

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pervol
  labels:
    type: local
spec:
  storageClassName: manual
  capacity:
    storage: 5Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/mnt/data"
```

Pv.yaml file. Kind sets it to a persistent volume. Setting the name to pv-volume. The storageClassName is set to manual which binds the claim requests to the now created PV. It also sets the size to 5GB. Additionally, the access mode is set to read read and write.

```
slikcaustic@cloudshell:~ (wise-brook-366921)$ kubectl apply -f pvvolume.yaml
persistentvolume/pervol created
slikcaustic@cloudshell:~ (wise-brook-366921)$ kubectl get pv
NAME     CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM                 STORAGECLASS   REASON   AGE
pervol   5Gi        RWO            Retain           Bound    default/pervolclaim   manual                  12m
```

Creating the persistent volume.

pvclaim.yaml > ...

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pervolclaim
spec:
  storageClassName: manual
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 3Gi
```

Set the kind to be PersistentVolumeClaim. This requests the volume of 3GB which will provide read and write for a single node.

```
slikcaustic@cloudshell:~ (wise-brook-366921)$ kubectl apply -f pvclaim.yaml
persistentvolumeclaim/pervolclaim created
```

Creating pvclaim.

```
slikcaustic@cloudshell:~ (wise-brook-366921)$ kubectl get pvc
NAME          STATUS   VOLUME   CAPACITY   ACCESS MODES   STORAGECLASS   AGE
pervolclaim   Bound    pervol   5Gi        RWO            manual         8m32s
```

Display the persistent volume claims.

pvpod.yaml > ...

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: pervolpod
spec:
  volumes:
    - name: pervolstorage
      persistentVolumeClaim:
        claimName: pervolclaim
  containers:
    - name: pervolcontainer
      image: nginx
      ports:
        - containerPort: 80
          name: "http-server"
      volumeMounts:
        - mountPath: "/usr/share/nginx/html"
          name: pervolstorage
```

Pvpod.yaml will create the pod that is going to be used by the claim pv that was previously created. In this case the claim is a volume.

```
slikcaustic@cloudshell:~ (wise-brook-366921)$ kubectl apply -f pvpod.yaml
pod/pervolpod created
```

Create the pvpod.

```
slikcaustic@cloudshell:~ (wise-brook-366921)$ kubectl exec -it pervolpod -- /bin/bash
root@pervolpod:/# curl http://localhost
Hello from Kubernetes storage
```

Open a shell that is running the pod. Check the output of the contents that are in the index.html file that was created using minikube. Since the message was the same as the message initially created earlier then the implementation of persistent volume is working correctly.

Conclusion:

After completing the lab's steps we now know and are comfortable with creating NodeJS services and Docker images. We are now familiar with Function as a Service (FaaS) and the configuration of nginx and circuit breakers. Through the completion of these steps we are now able to configure OpenFaaS on the Google Cloud Platform, and are now familiar with the Decorator Pattern that was implemented, and could now understand other implementations.