Cloud Computing Systems

Diogo Rodrigues 56153, João Vargues 55185, and José Murta 55226

NOVA School of Science and Technology, Portugal

Abstract. The present document is the report for the Cloud Computing Systems course project two. This document has presented all the work developed in a clear, concise and complete way.

1 Assignment Solution

For the second project of the Cloud Computing Systems course, we were asked to use Docker and Kubernetes in the deployment of the project that we developed in the first phase.

Initially, we had to create a Docker image of the project using the compiled war file, which was retrieved from the compilation of our web app using maven, to gather all the necessary dependencies and libraries.

To test our web app image, we ran a container locally with the image previously created to understand if everything was running correctly. After it was verified, we started the implementation of our Kubernetes service.

1.1 Kubernetes deployment

To deploy our application using Kubernetes, we used a YAML file to define all the pods, services, and volumes needed to execute with the application's container. With this configuration, we ended up with two pods deployed, one defining the back end that encapsulates the Redis image and another defining the front that encapsulates the application's container and also mounts a volume in the path "/mnt/vol" that is claimed as a Persistent volume so the data lives even after a container restart or failure. For these pods, we launched two services one for the back pod accessible by port 6379, which is the port used to access the REDIS cache in the Java code, and another service for the front pod that acts as a load balancer and is accessible from an external user.

We also needed to do changes to our code so it could use the provided services. In order for the media resource to use the mounted persistent volume, so from the path defined in the deployment of the pod, we wrote

a file to upload an image using a FileOutputStream, read a file from the directory to download using FileInputStream, and to list all images we iterate the directory and printed all file names. To use the Redis image the code changes were slightly fewer because it was only needed to change the hostname to the name of the pod that contained it, the back one, and the port to the one defined in the deployment file and define the JedisPool with those characteristics.

Due to time constraints we were not able to finish the deployment of MongoDB image to Kubernetes but in the file project it can be found a YAML file for this purpose and a class with the Java code that creates a MongoDb client that creates, deletes and retrieves a user with some id from the collection users created.

1.2 Test System

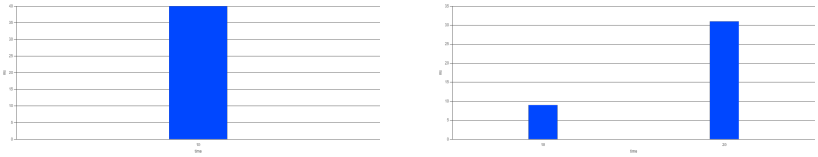
To be able to deploy our test system using Azure Container Instances to better test our application by deploying it in different locations, it was first necessary to build a Docker image composed of all the test files and the artillery tool, together with its corresponding dependencies. This image is based on the node.js image, as suggested in the labs, being specified in its Dockerfile that the test files are copied to the image, and for running these files, it was also necessary to execute 4 RUN commands, to install artillery, faker, node-fetch and the artillery plugin to produce metrics by an endpoint.

After building the described image we can then run containers of this image. We started by testing the system running containers locally to verify that the system runs adequately. Next, the image was pushed to the Docker Hub, and only after that, we were able to create an Azure Container Instance that runs a container of the desired image, using the *azure container create* command where we can specify on what resource group the instance will be deployed. This way we can have resource groups in different locations and deploy the test systems in those groups to have clients running in different data centers. It is important to mention that, since the containers that the image of our test system created has no long-running processes, the container would exit immediately after completing running. To solve this, it was necessary to include a start command on the *azure container create* command by specifying *–command-line "tail -f /dev/null"*.

2 Evaluation and results

In order to evaluate our implementation we deployed our test system in the West Europe location as well as in Canada Central region. This way we can provide clients running in different data centers. In this second phase of the project, we used the same test battery as in phase one, and still focused on the tests that create users, create auctions, and perform a mix of operations (retrieving the user's auction list, and placing bids, questions and replies in determined auctions), however this time we only tested our application with cache enabled.

The results obtained, display a clear improvement in response time for requests being made when the test system is running in west Europe region compared to when the test system is running in the central Canada location. In fact, the response time doubles, going from 10ms to 20ms, when running in the later location, as can be seen in Fig.1. However, the difference between the locations was not as significant as we predicted and as it was noticeable in the deployment of the web app from the first phase.



(a) West Europe (b) Central Canada

Fig. 1: Results of the response time of the Create users test

3 Annexes

3.1 Dockerfile for Web App image

```
FROM tomcat:10.0-jdk17-openjdk
WORKDIR /usr/local/tomcat
ADD scc2223-tp1-1.0.war webapps
EXPOSE 8080
```

3.2 Kubernetes Deployment file

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: a55185-55226-56153-scctp2-back
spec:
  replicas: 1
  selector:
    matchLabels:
      app: a55185-55226-56153-scctp2-back
  template:
    metadata:
      labels:
        app: a55185-55226-56153-scctp2-back
    spec:
      nodeSelector:
        "beta.kubernetes.io/os": linux
      containers:
      - name: a55185-55226-56153-scctp2-back
        image: mcr.microsoft.com/oss/bitnami/redis:6.0.8
        env:
        - name: ALLOW_EMPTY_PASSWORD
          value: "yes"
        resources:
          requests:
            cpu: 100m
            memory: 128Mi
          limits:
            cpu: 250m
            memory: 256Mi
```

```
        ports:
        - containerPort: 6379
          name: redis
---
apiVersion: v1
kind: Service
metadata:
  name: a55185-55226-56153-scctp2-back
spec:
  ports:
  - port: 6379
  selector:
    app: a55185-55226-56153-scctp2-back
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: a55185-55226-56153-scctp2-front
spec:
  replicas: 1
  selector:
    matchLabels:
      app: a55185-55226-56153-scctp2-front
  template:
    metadata:
      labels:
        app: a55185-55226-56153-scctp2-front
    spec:
      nodeSelector:
        "beta.kubernetes.io/os": linux
      containers:
      - name: a55185-55226-56153-scctp2-front
        image: jmurta15/scc2223-app5
        volumeMounts:
        - mountPath: "/mnt/vol"
          name: mediavolume
        resources:
          requests:
            cpu: 100m
            memory: 128Mi
```

```
          limits:
            cpu: 250m
            memory: 256Mi
        ports:
        - containerPort: 8080
        env:
        - name: REDIS
          value: "a55185-55226-56153-scctp2-back"
        - name: COSMOS_DB_NAME
          value: "scctp1db"
        - name: BLOB_STORE_CONNECTION
          value: "DefaultEndpointsProtocol=https;AccountName=scctp1storag
        - name: QUERY_KEY
          value: "8z8d9qAJ7ITtiu1b54FGm1ZbKPyfBrGaXPsQWpAypPAzSeCcUdl9"
        - name: COSMOS_CONNECTION_URL
          value: "https://scctp1cosmosdb.documents.azure.com:443/"
        - name: COSMOS_DB_KEY
          value: "aRQeHIFXAlwba2rPs34mPitOB98ALBOyPnlbAmZbPjDyT6d4KIRgKqJ
      volumes:
      - name: mediavolume
        persistentVolumeClaim:
          claimName: azure-managed-disk
---
apiVersion: v1
kind: Service
metadata:
  name: a55185-55226-56153-scctp2-front
spec:
  type: LoadBalancer
  ports:
  - port: 80
    targetPort: 8080
  selector:
    app: a55185-55226-56153-scctp2-front
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: azure-managed-disk
spec:
```

```
  accessModes:
    - ReadWriteOnce
  storageClassName: azurefile
  resources:
    requests:
      storage: 1Gi
```

3.3 Dockerfile for test system

```
FROM node
WORKDIR /usr/local/node
ADD testing tests
RUN npm install -g artillery
RUN npm install --save-dev faker@5.5.3
RUN npm install -g node-fetch -save
RUN npm install -g https://github.com/preguica
/artillery-plugin-metrics-by-endpoint.git
```