

Cloud Computing Systems Second Work Report

Diogo Almeida⁵⁸³⁶⁹, Diogo Fona⁵⁷⁹⁴⁰, and Bruno Cabrita⁵⁷⁸³³

NOVA School of Science and Technology - MSc in Computer Engineering
`{brm.cabrita,daro.almeida,d.fona}@campus.fct.unl.pt`

1 Assignment Solution

In this work, we implement the application presented in the first work and deploy it in Azure in a Kubernetes cluster, instead of using the built-in Azure components. To achieve this we build containers, using Docker, with the application's constituents and deploy them in Kubernetes Pods.

1.1 Implementation

As in the previous work, for the application to support the storage, querying, and processing of information, we use cloud software and hardware components that we deploy with Kubernetes: (settings described in deployment files in appendix)

Application: The application's backend that was implemented in the first work is built in a container and deployed. Since we use different components the communication API calls with them are altered. The application is accessed externally its a LoadBalancer type service.

Database: The database we use is MongoDB instead of Azure CosmosDB. Particularly in this work for simplicity its information is not stored persistently.

Media Storage: For the storage of media content in the application we use a Persistent Volume Claim instead of Azure Blob Storage. It stores the contents persistently in a file system volume.

Cache: The caching system we use is Redis like in the first work.

Serverless Functions: Regarding serverless functions that were previously implemented with Azure Functions, functions that were timer based (TimerTrigger) are implemented with CronJobs, and the rest in separate deployments.

Message Queues: The message queue system we use is RabbitMQ instead of Azure Service Bus.

1.2 Pods

Each of the Kubernetes cluster’s Pods are single-container and contain the application’s services: backend, mongodb, rabbitmq, redis, and one for each function.

2 Evaluation

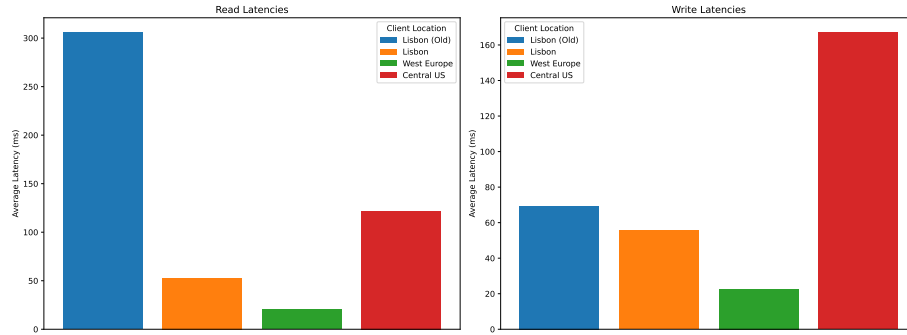


Fig. 1. Average latency in milliseconds of read and write operations from clients in different regions with the application in West Europe. The blue bar concerns the previous implementation that used Azure services and the remaining concerns the new one that uses the Kubernetes cluster.

To perform the evaluation of our implementation we deploy a Docker container with workload tests executed with Artillery, using Azure Container Instances, in different regions and also run it locally with Docker.

The results shown in Fig. 1 demonstrate that using either the Azure services or Kubernetes with alternate services has no significant difference in impact.

In the graph of writes it’s observed that using Kubernetes perceives slightly lower latency but that is probably due to the different mechanism’s performance or variance.

In the graph of reads we can see that the previous implementation has a much higher perceived latency. This is because we repaired an implementation issue that caused some listing operations to make too much requests which we mentioned in the first work.

A Dockerfiles

```

1 FROM docker.io/tomcat:10-jdk17-corretto
2 WORKDIR /usr/local/tomcat/webapps
3 RUN curl https://github.com/open-telemetry/opentelemetry-java-
    instrumentation/releases/download/v1.20.2/opentelemetry-javaagent.jar
    -L --output opentelemetry-javaagent.jar
4 ENV JAVA_OPTS="-ea -javaagent:opentelemetry-javaagent.jar"
5 COPY modules/backend-app/target/scc-backend-app-1.0-SNAPSHOT.war ROOT.war
6 EXPOSE 8080

```

Listing 1.1. backend.dockerfile

```

1 FROM docker.io/amazoncorretto:19
2 WORKDIR /usr/local/app
3 ENV JAVA_OPTS="-ea"
4 COPY modules/worker-assembly/target/scc-worker-assembly-1.0-SNAPSHOT-jar-
    with-dependencies.jar app.jar
5 ENTRYPOINT ["java", "-cp", "app.jar"]

```

Listing 1.2. worker.dockerfile

```

1 FROM docker.io/artilleryio/artillery
2 WORKDIR /usr/local/app
3 RUN npm install @faker-js/faker --save-dev
4 ADD tester/testing/artillery/*.yaml .
5 ADD tester/testing/artillery/*.js .
6 COPY tester/artillery-entrypoint.sh .
7 ENTRYPOINT ["/bin/sh", "./artillery-entrypoint.sh"]

```

Listing 1.3. tester.dockerfile

B Kubernetes deployment files

backend.yaml

```

1 apiVersion: v1
2 kind: ConfigMap
3 metadata:
4   name: scc-backend-cmap
5 data:
6   KUBE_CACHING_ENABLED: "true"
7   KUBE_MEDIA_DATA_DIRECTORY: /var/data
8   KUBE_MONGO_CONNECTION_URI: mongodb://scc-mongodb:27017
9   KUBE_MONGO_DATABASE_NAME: scc-backend
10  KUBE_MONGO_AUCTION_COLLECTION: auctions
11  KUBE_MONGO_BID_COLLECTION: bids
12  KUBE_MONGO_QUESTION_COLLECTION: questions

```

```

13 KUBE_MONGO_USER_COLLECTION: users
14 KUBE_REDIS_URL: scc-redis
15 KUBE_REDIS_PORT: "6379"
16 KUBE_RABBITMQ_HOST: scc-rabbitmq
17 KUBE_OTLP_ENABLED: "true"
18 KUBE_RABBITMQ_PORT: "5672"
19 ---
20 apiVersion: v1
21 kind: PersistentVolumeClaim
22 metadata:
23   name: scc-backend-pvc
24 spec:
25   resources:
26     requests:
27       storage: 2Gi
28     storageClassName: azurefile
29     volumeMode: Filesystem
30     accessModes:
31       - ReadWriteOnce
32
33 ---
34 apiVersion: apps/v1
35 kind: Deployment
36 metadata:
37   name: scc-backend
38 spec:
39   selector:
40     matchLabels:
41       app: scc-backend
42   template:
43     metadata:
44       labels:
45         app: scc-backend
46     spec:
47       containers:
48         - name: scc-backend
49           image: git.d464.sh/diogo464/scc-backend
50           resources:
51             limits:
52               memory: "1Gi"
53               cpu: "500m"
54           ports:
55             - containerPort: 8080
56           volumeMounts:
57             - mountPath: /var/data

```

```
58         name: scc-backend-pvc
59     envFrom:
60     - configMapRef:
61         name: scc-backend-cmap
62     volumes:
63     - name: scc-backend-pvc
64       persistentVolumeClaim:
65         claimName: scc-backend-pvc
66 ---
67 apiVersion: v1
68 kind: Service
69 metadata:
70     name: scc-backend
71 spec:
72     type: LoadBalancer
73     selector:
74         app: scc-backend
75     ports:
76     - port: 80
77       targetPort: 8080
78 ---
79 apiVersion: apps/v1
80 kind: Deployment
81 metadata:
82     name: scc-worker-auction-close
83 spec:
84     selector:
85         matchLabels:
86             app: scc-worker-auction-close
87     template:
88         metadata:
89             labels:
90                 app: scc-worker-auction-close
91     spec:
92         containers:
93         - name: scc-worker-auction-close
94           image: git.d464.sh/diogo464/scc-worker
95           args: [scc.worker.AuctionClose]
96           resources:
97             limits:
98                 memory: "128Mi"
99                 cpu: "80m"
100         envFrom:
101         - configMapRef:
102             name: scc-backend-cmap
```

```

103 ---
104 apiVersion: apps/v1
105 kind: Deployment
106 metadata:
107   name: scc-worker-auction-popularity-consumer
108 spec:
109   selector:
110     matchLabels:
111       app: scc-worker-auction-popularity-consumer
112   template:
113     metadata:
114       labels:
115         app: scc-worker-auction-popularity-consumer
116     spec:
117       containers:
118         - name: scc-worker-auction-popularity-consumer
119           image: git.d464.sh/diogo464/scc-worker
120           args: [scc.worker.AuctionPopularityConsumer]
121           resources:
122             limits:
123               memory: "128Mi"
124               cpu: "50m"
125           envFrom:
126             - configMapRef:
127               name: scc-backend-cmap
128 ---
129 apiVersion: batch/v1
130 kind: CronJob
131 metadata:
132   name: scc-worker-auction-popularity-updater
133 spec:
134   schedule: "*/1 * * * *"
135   jobTemplate:
136     spec:
137       template:
138         spec:
139           containers:
140             - name: scc-worker-auction-popularity-updater
141               image: git.d464.sh/diogo464/scc-worker
142               args: [scc.worker.AuctionPopularityUpdater]
143               resources:
144                 limits:
145                   memory: "128Mi"
146                   cpu: "80m"
147               envFrom:

```

```
148         - configMapRef:
149           name: scc-backend-cmap
150         restartPolicy: OnFailure
```

mongo.yaml

```
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: scc-mongodb
5  spec:
6    selector:
7      matchLabels:
8        app: scc-mongodb
9    strategy:
10     type: Recreate
11    template:
12     metadata:
13       labels:
14         app: scc-mongodb
15     spec:
16       containers:
17         - name: scc-mongodb
18           image: docker.io/mongo:5.0.14
19           resources:
20             limits:
21               memory: "256Mi"
22               cpu: "500m"
23           ports:
24             - containerPort: 27017
25     ---
26     apiVersion: v1
27     kind: Service
28     metadata:
29       name: scc-mongodb
30     spec:
31       selector:
32         app: scc-mongodb
33       ports:
34         - port: 27017
35
```

rabbitmq.yaml

```

1  apiVersion: v1
2  kind: ConfigMap
3  metadata:
4    name: scc-rabbitmq-cmap
5  data:
6    config: ""
7    plugins: |
8      [rabbitmq_management,rabbitmq_management_agent].
9  ---
10 apiVersion: v1
11 kind: ConfigMap
12 metadata:
13   name: scc-rabbitmq-cmap-env
14 data:
15   RABBITMQ_ENABLED_PLUGINS_FILE:
16     ↪ /usr/local/configs/enabled_plugins
17   #RABBITMQ_CONFIG_FILE: /usr/local/configs/rabbitmq
18 ---
19 apiVersion: apps/v1
20 kind: Deployment
21 metadata:
22   name: scc-rabbitmq
23 spec:
24   selector:
25     matchLabels:
26       app: scc-rabbitmq
27   template:
28     metadata:
29       labels:
30         app: scc-rabbitmq
31     spec:
32       containers:
33       - name: scc-rabbitmq
34         image: rabbitmq
35         resources:
36           limits:
37             memory: "196Mi"
38             cpu: "500m"
39         ports:
40         - containerPort: 5672
41         - containerPort: 15672
42       volumeMounts:
43       - mountPath: /usr/local/configs
44         name: scc-rabbitmq-config-vol

```



```
44         envFrom:
45         - configMapRef:
46             name: scc-rabbitmq-cmap-env
47     volumes:
48     - name: scc-rabbitmq-config-vol
49       configMap:
50         name: scc-rabbitmq-cmap
51       items:
52       - key: config
53         path: rabbitmq.config
54       - key: plugins
55         path: enabled_plugins
56 ---
57 apiVersion: v1
58 kind: Service
59 metadata:
60   name: scc-rabbitmq
61 spec:
62   selector:
63     app: scc-rabbitmq
64   ports:
65   - port: 5672
66
```

redis.yaml

```
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: scc-redis
5  spec:
6    selector:
7      matchLabels:
8        app: scc-redis
9    template:
10     metadata:
11       labels:
12         app: scc-redis
13     spec:
14       containers:
15       - name: scc-redis
16         image: docker.io/redis:7.0.5
17         resources:
18           limits:
19             memory: "256Mi"
```

```
20         cpu: "500m"
21     ports:
22     - containerPort: 6379
23 ---
24 apiVersion: v1
25 kind: Service
26 metadata:
27     name: scc-redis
28 spec:
29     selector:
30         app: scc-redis
31     ports:
32     - port: 6379
```