NOVA SCHOOL OF SCIENCE & TECHNOLOGY

Cloud Computing Systems

2024/2025

2nd Project Report - Azure Tukano

Authors:

70526, André Branco
70527, João Silveira

Lab class Nº P3

Assignment Analysis

Application deployment

The Tukano application was deployed in the Azure Kubernetes service using our own docker image. The application still maintains all of the functionalities implemented in the first project, but for the sake of simplicity, we now consider that the application is always running with Cache and that we only want to use PostgreSQL, as opposed to what was done in the first project where we could turn these on and off with environmental variables. The image for the main Tukano application was posted on Docker hub as “jpgsilveira/tukano”.

Tukano Deployment Yaml:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: tukano
spec:
  replicas: 1
  selector:
    matchLabels:
      app: tukano
  template:
    metadata:
      labels:
        app: tukano
    spec:
      containers:
      - name: tukano
        image: jpgsilveira/tukano
        ports:
        - containerPort: 8080
        env:
        - name: BLOB_HOST
          value: "http://172.205.24.157:8080/tukano-blobs-1/rest"
```

Tukano Service Yaml:

```yaml
apiVersion: v1
kind: Service
metadata:
  name: tukano
spec:
  type: LoadBalancer
  ports:
  - name: http
    port: 80
    targetPort: 8080
  selector:
    app: tukano
```

Deploy the Hibernate + SQL database

For the deployment of the SQL database, we utilized the base Hibernate logic given to us for the first project, adding the connection to the database pod in the hibernate.cfg.xml file.

```
<!-- JDBC Database connection settings -->
<property name="hibernate.connection.driver_class">org.postgresql.Driver</property>
<property name="hibernate.connection.url">jdbc:postgresql://database:5432/tukano</property>
<property name="hibernate.connection.username">tukano</property>
<property name="hibernate.connection.password">tukano</property>
```

The Docker image chosen for the database was “postgres”.

Database Deployment Yaml:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: database
spec:
  replicas: 1
  selector:
    matchLabels:
      app: database
  template:
    metadata:
      labels:
        app: database
    spec:
      containers:
      - name: database
        image: postgres
        ports:
        - containerPort: 5432
        env:
        - name: POSTGRES_DB
          value: "tukano"
        - name: POSTGRES_USER
          value: "tukano"
        - name: POSTGRES_PASSWORD
          value: "tukano"
```

Database Service Yaml:

```
apiVersion: v1
kind: Service
metadata:
  name: database
spec:
  type: ClusterIP
  ports:
  - name: db
    port: 5432
    targetPort: 5432
  selector:
    app: database
```

Blob Storage & Docker Images

In this project, we ended up separating the blob storage from the rest of the Tukano application due to the different containers and pods necessary for deployment, and therefore also Docker Images. We kept Tukano's logic as is, only removing the mentions of "Blobs", and moved those into another folder where the Blob logic would stand on its own. For simplicity, we still kept both codes in the same directory, under different folders, and have created separate Docker Images for each one for the separate deployment.

Blob Deployment Yaml:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: blob-storage
spec:
  replicas: 1
  selector:
    matchLabels:
      app: blob-storage
  template:
    metadata:
      labels:
        app: blob-storage
    spec:
      containers:
      - name: blob-storage
        image: jpgsilveira/tukano-blobs
        volumeMounts:
        - mountPath: "/tmp/blobs"
          name: blob-volume
      volumes:
      - name: blob-volume
        persistentVolumeClaim:
          claimName: blob-storage-pvc
```

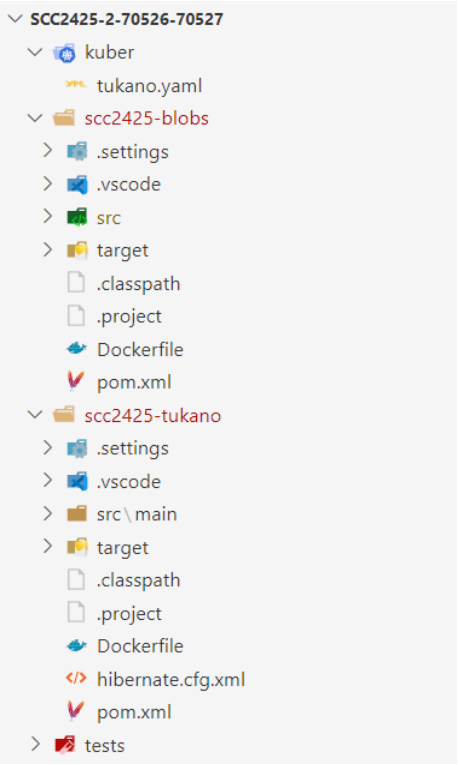
Blob Service Yaml:

```yaml
apiVersion: v1
kind: Service
metadata:
  name: blob-storage
spec:
  type: LoadBalancer
  ports:
  - name: http
    port: 8080
    targetPort: 8080
  selector:
    app: blob-storage
```

Blob Persistent Volume Yaml:

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: blob-storage-pvc
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

File Explorer View:



User Session Authentication

For this functionality, we took most of our implementations and understanding, from the Lab9 practical class. This validation happens with the intent of controlling the access Users have over some functions, mainly Blob upload and download. For these to be authorized, the User needs to execute a “getUser” function to have his cookie generated, and stored in the cache. From then on, said cookie must be sent as a header in the Blob upload or download request, for the User to be identified by the system. The Authentication logic exists both in the Tukano and Blob storage folders. To delete a blob the user logged in must be named “admin”.

Cache Service

For the cache, this one is also part of both Tukano and Blob storage, because, besides its original purpose as a cache for the database in Tukano, it is now also part of the User authentication, for storing and getting its cookie, from Tukano to the Blob storage. The cache is also deployed in a separate container and pod, using an already existing image of “redis” cache, which is a familiar cache structure from the first project.

Cache Deployment Yaml:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: cache
spec:
  replicas: 1
  selector:
    matchLabels:
      app: cache
  template:
    metadata:
      labels:
        app: cache
    spec:
      containers:
      - name: cache
        image: redis
        args:
        - redis-server
        - --requirepass
        - "key"
```

Cache Service Yaml:

```yaml
apiVersion: v1
kind: Service
metadata:
  name: cache
spec:
  type: ClusterIP
  ports:
  - name: cache
    port: 6379
    targetPort: 6379
  selector:
    app: cache
```

Kubernetes and YAML file

We started by gathering the examples given in Lab10 and practical classes after, where we knew our yaml file (or files) would need to have all four pods, their deployments, and backend service. The first iteration of this was pretty straightforward, but we started having issues when trying to access the Blob Storage pod. From what we could find, the Kubernetes, in the yaml file, can have access to the Internal IP from another container, but we couldn't find a way of accessing the External IP from the file, to have it as an environmental variable for Tukano to create the 'blobURL' of a short. After many attempts, we agreed our best option was to leave it "hard coded", having to use the terminal to check on the blob pod's service table, manually extract the external IP, and put it in the environmental variables in the Tukano Deployment yaml, and re-applying the yaml file.

Also with mentioning environmental variables, our previous solution from the first project had a separate '.env' file for handling those. But in this project, Kubernetes already has an integrated and easier solution for environmental variables, and we no longer use our file.

To make the deployment of the application easier we placed all yaml components into a single file called tukano.yaml, to which we added comments to denote what would be each different component, making reading it easier.

Other Alteration:

Parameter 'id' to Object classes

We maintained the alteration of the first project to add the "id" parameter to the User class in the database.

Therefore when executing our method 'createUser' it is necessary to also add a new 'id' parameter to the JSON body like so:

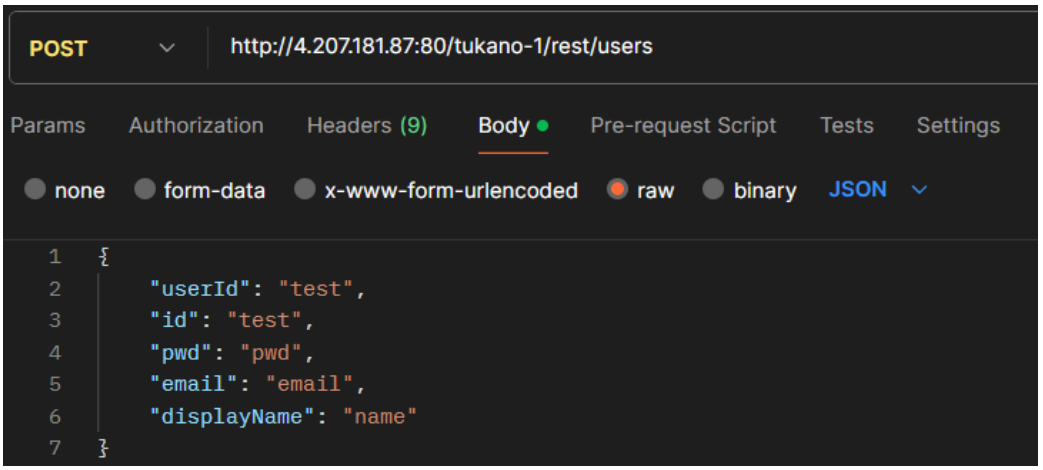


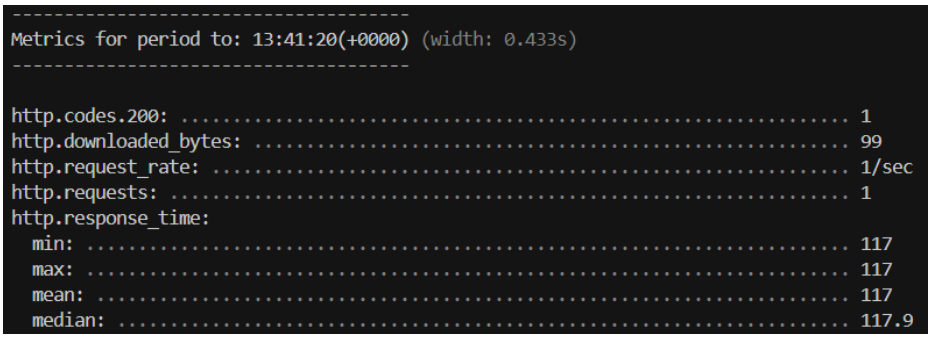
Image 1: Post User example

The parameters 'userId' and 'id' MUST be the same for the application to work correctly.

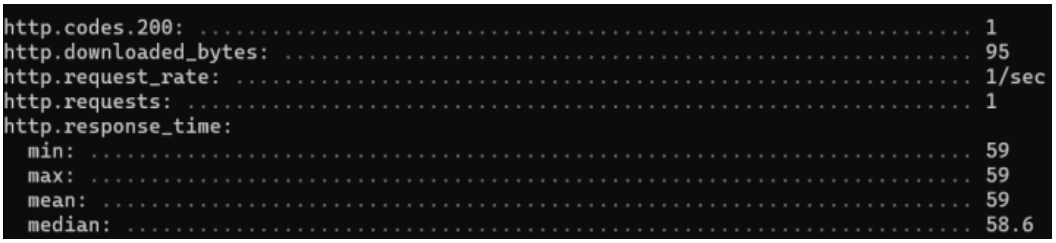
Test Comparison

create_getUser.yaml - Posting and ‘getting’ a User

- 1st Project :

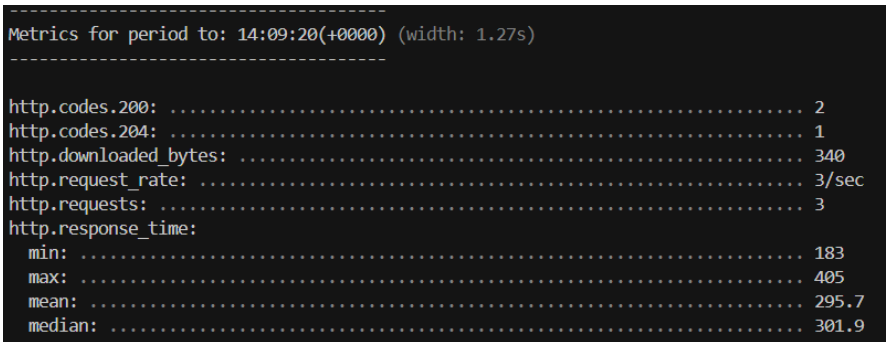


- 2nd Project :

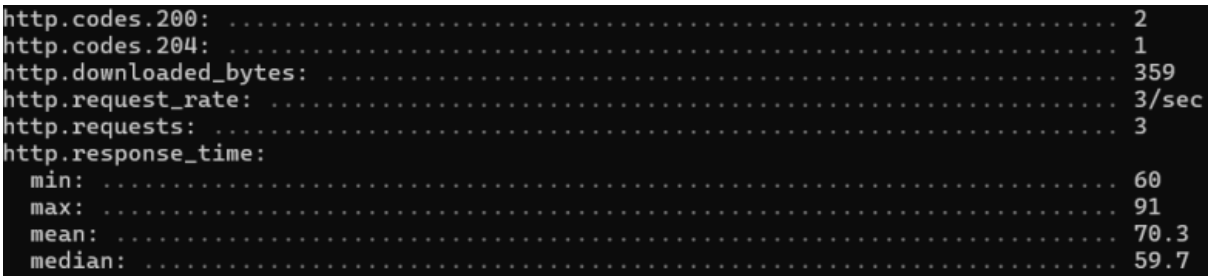


like_n_getLikes.yaml - createShort, liked by a user, and check the number of likes

- 1st Project :



- 2nd Project :



delete_User.yaml - Delete a User

- 1st Project :

```
-----------------------------------
Metrics for period to: 16:49:30(+0000) (width: 0.432s)
-----------------------------------

http.codes.200: ..................................................... 1
http.downloaded_bytes: .............................................. 99
http.request_rate: .................................................. 1/sec
http.requests: ...................................................... 1
http.response_time:
  min: .............................................................. 96
  max: .............................................................. 96
  mean: ............................................................. 96
  median: ........................................................... 96.6
```

- 2nd Project :

```
http.codes.200: ..................................................... 1
http.downloaded_bytes: .............................................. 115
http.request_rate: .................................................. 1/sec
http.requests: ...................................................... 1
http.response_time:
  min: .............................................................. 58
  max: .............................................................. 58
  mean: ............................................................. 58
  median: ........................................................... 58.6
```

Unauthorized_BlobUpload.yaml - Get

```
http.codes.200: ..................................................... 2
http.codes.401: ..................................................... 1
http.downloaded_bytes: .............................................. 428
http.request_rate: .................................................. 3/sec
http.requests: ...................................................... 3
http.response_time:
  min: .............................................................. 55
  max: .............................................................. 62
  mean: ............................................................. 59
  median: ........................................................... 59.7
```

For this “Unauthorized” test, it initially was meant to be using the cookies implemented, so it could post a short and upload a blob. But we had many issues with artillery not recognizing the cookie, even when we captured it, it couldn’t be sent in the upload request. It isn’t a code issue because we previously tested manually with Postman and so we ended up leaving the test to still show that a user ‘without’ a cookie, won’t be able to upload a blob.

Through all the test comparisons, the performance improvements in the 2nd Project compared to the 1st Project are due to the significant deployment differences. In the 1st Project, the application was hosted on Azure PaaS (Platform-as-a-Service), using services like Azure Blob Storage, CosmosDB, and Azure Cache for Redis. While these services are useful and scalable, they introduce network latency because each service operates independently. In contrast, the 2nd Project was deployed in a Kubernetes cluster, AKS (Azure Kubernetes Service), where the application server, database, and caching services are in containers and run in close proximity, significantly reducing communication latency between them.