NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

Cloud Computing Systems

2024/2025

Project Assignment #2

Authors:

55806, Tomás Santos
72287, Eduardo Silveiro

Professores:

Sergio Duarte
Kevin Gallagher

December 6, 2024

Introduction

The objective of this project was to adapt and deploy the Tukano application, initially developed during the first project assignment, using Docker and Kubernetes as Infrastructure as a Service (IaaS) tools provided by Azure. The central goal was to replace the previously used Platform as a Service (PaaS) components with Kubernetes-based alternatives while retaining the existing functionality.

The project aimed to explore the potential of containerization and orchestration technologies to create more flexible and scalable solutions. Additionally, it leverages existing Docker images available on Docker Hub, with the proposed infrastructure(PostegreSQL and Redis).

This report is the way to evaluate our solution in terms of performance, analyzing throughput and latency for deployments.

Code Choices

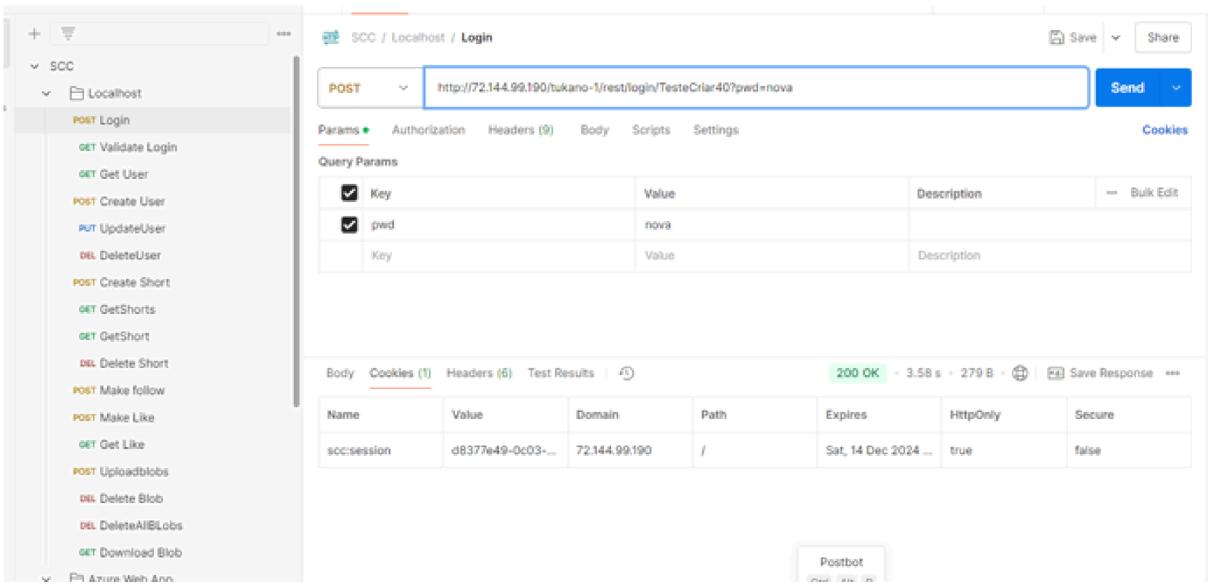
During the development, we adopted strategies to simplify code management and allow flexibility. When needed, we swapped between enum variables, avoiding to change the code directly in multiple places, which helped prevent hard-coded modifications.We also started with the base project instead of using the first project assignment.

For authentication we created a class that has an endpoint (/rest/login/userId?pwd=pwd) to handle the login and cookies logic, we also implemented the validation of users session in JavaBlobs.

The persistent volume storage, the hibernate.cfg.xml file was modified to update the database connection settings, adjust the File System Storage directory to /mnt/vol, and change the database schema generation mode to update during creation to keep the data safe after deleting a pod.

Authentication

We implemented the authentication system using cookies to manage user login sessions and the login functionality was designed to be available only when the cache is enabled.



SCC / Localhost / Login

POST http://72.144.99.190/tukano-1/rest/login/TesteCriar40?pwd=nova

Query Params

Key	Value	Description
✓ pwd	nova	

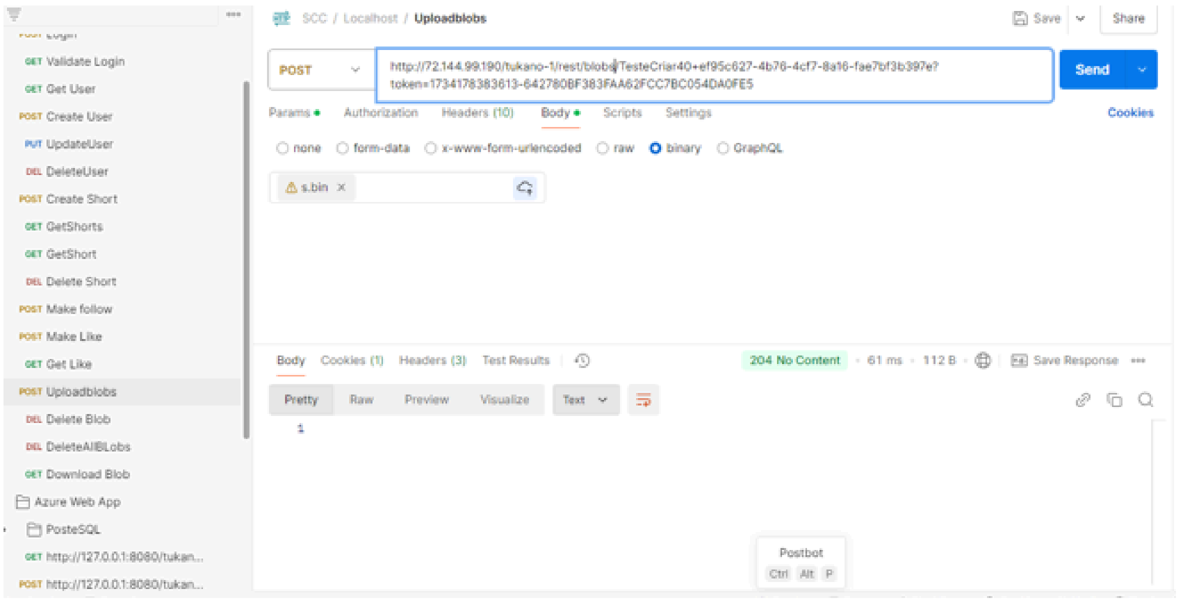
200 OK · 3.58 s · 279 B

Name	Value	Domain	Path	Expires	HttpOnly	Secure
scc:session	d8377e49-0c03-...	72.144.99.190	/	Sat, 14 Dec 2024 ...	true	false

Figure 1 - Endpoint Login Example

This cookie-based authentication mechanism was further integrated into the Blob Service for secure access.

In terms of permissions, regular users are allowed to upload and download blobs, but only administrators have the authority to delete blobs.



SCC / Localhost / Uploadblobs

POST http://72.144.99.190/tukano-1/rest/blobs/TesteCriar40+ef95c627-4b76-4cf7-8a16-fae7bf3b397e?token=1734178383613-642780BF383FAA62FCC7BC054DA0FE5

Body: ( ) none ( ) form-data ( ) x-www-form-urlencoded ( ) raw [✓ binary] ( ) GraphQL

s.bin

204 No Content · 61 ms · 112 B

```
1
```

Figure 2 - Endpoint UploadBlob Example

Results Analysis

In this section, we present a comparison of the performance results with cache active and without cache in both project to better understand what is the best option for the available resources.These comparisons provide insights into how each choice influences key metrics, such as latency and throughput, and help evaluate the effectiveness of caching in enhancing overall application performance.

It is important to denote that this data was generally the “average” of the best results when running the tests and all the endpoints on the second assignment are working correctly, unlike the first project.

TuKano Application Performance Comparison: Azure .vs Kubernetes Deployment

Comparison Summary

Metric	Azure (With Caching)	Azure (Without Caching)	Kubernetes (With Caching)	Kubernetes (Without Caching)
Request Rate	10 requests/sec	5 requests/sec	3 requests/sec	3 requests/sec
Downloaded Data	2116 bytes	3300 bytes	1438 bytes	1917 bytes
Mean Response Time	54.5 ms	55.2 ms	54.4 ms	54.4 ms
Session Length	1118 ms	1534.2 ms	129.7 ms	136.1 ms
HTTP 200 (Success)	39 responses	26 responses	17 responses	20 responses
HTTP 500 (Server Errors)	0 occurrences	3 occurrences	1 occurrence	2 occurrences
HTTP 404 (Not Found)	0 occurrences	0 occurrences	2 occurrences	8 occurrences
HTTP 403 (Forbidden)	0 occurrences	0 occurrences	5 occurrences	2 occurrences

Key Observations:

1. **Request Rate & Throughput:**

Azure with caching leads in terms of request handling, processing 10 requests/sec compared to Kubernetes' 3 requests/sec, indicating better scalability on Azure.

Kubernetes, though comparable in response time, processes fewer requests due to resource or configuration limitations.

2. **Data Volume:**
    - Azure with caching benefits from reduced data volume per request (2116 bytes), showing that caching effectively reduces backend load and data transfer.
    - Kubernetes with caching transfers less data (1438 bytes) but still less efficiently than Azure.
3. **Response Time:**
    - The response times are very similar across both platforms, with Azure slightly outperforming Kubernetes in some scenarios, though Kubernetes also shows efficient caching performance (54.4 ms).
4. **Error Rates:**
    - Azure performs better in terms of error handling, especially with fewer server errors (500) and no access issues (403).
5. **Session Duration:**
    - Azure shows a more significant reduction in session length with caching, indicating better task completion efficiency (1118 ms vs 136.1 ms in Kubernetes).

Summary of Performance Comparison:

- Azure provides higher throughput, better scalability, and more stable error management, making it more suitable for large-scale, high-traffic applications.
- Kubernetes handles the requests well but needs further resource optimizations to match Azure's performance, especially in handling a higher volume of requests and minimizing errors.

<u>Conclusion</u>

It is important to note that we opted not to use tools like Minikube during the development process, as we still had Azure credits available. However, due to the high cost of running Azure services, particularly for scaling and maintaining resources, we conducted more controlled and focused tests using artillery.

Additionally, we leveraged ChatGPT to organize our ideas, clarify doubts, and better understand errors encountered during development. This tool helps in streamlining our problem-solving process and enhancing the overall workflow of the project.

Azure typically outperforms Kubernetes in terms of scalability, ease of use, and overall performance, especially when leveraging managed services designed for high traffic and low latency.

Briefly, we consider that Kubernetes, while highly flexible and scalable, requires more careful setup and resource management. Its performance can be impacted by factors like resource allocation, pod configurations, and network performance. Without proper tuning, Kubernetes can underperform compared to managed cloud platforms like Azure.

However, Kubernetes can be optimized for better performance (e.g., using Kubernetes Autoscaling, optimizing resource requests/limits, enhancing pod scheduling, improving networking).