NOVA SCHOOL OF SCIENCE & TECHNOLOGY

SCC Report - Azure IaaS

Sistemas de Computação Cloud

2024/2025

TuKano

Authors:

69369, Aurélio Miranda
70068, Rafael Pires

Lab class Nº P1

Professor:
Sérgio Duarte

Introduction **3**
Architecture Design **3**
Implementation Details **4**
Testing Setup 5
New vs. Old Performance 5
Evaluation and Discussion **10**
Summary of Results 10
Challenges & Improvements 10
Conclusion **11**

Introduction

In this assignment we continued the development of the TuKano system, this time deploying the various TuKano services using Kubernetes in the Azure IaaS or locally using Minikube.

We were able to successfully deploy the application both in Azure and Minikube. Eventually, we opted to develop and test the deployment in Minikube since a local deployment is faster. The only technical issue we ran into utilizing Minikube was that our performance was very slow, but that was quickly solved by editing the yaml files and increasing available resources.

Architecture Design

TuKano's architecture: Original TuKano architecture, now incorporating authentication for Blob management.

Azure IaaS Integration: Aims to deploy the TuKano system using kubernetes.

- **Tukano Container**: Application logic.
- **Database Container**: Data storage for user and shorts metadata.
- **Redis Container**: Cache storage system for login, user and shorts metadata.
- **Persistent Volume**: Persistent storage for shorts data.

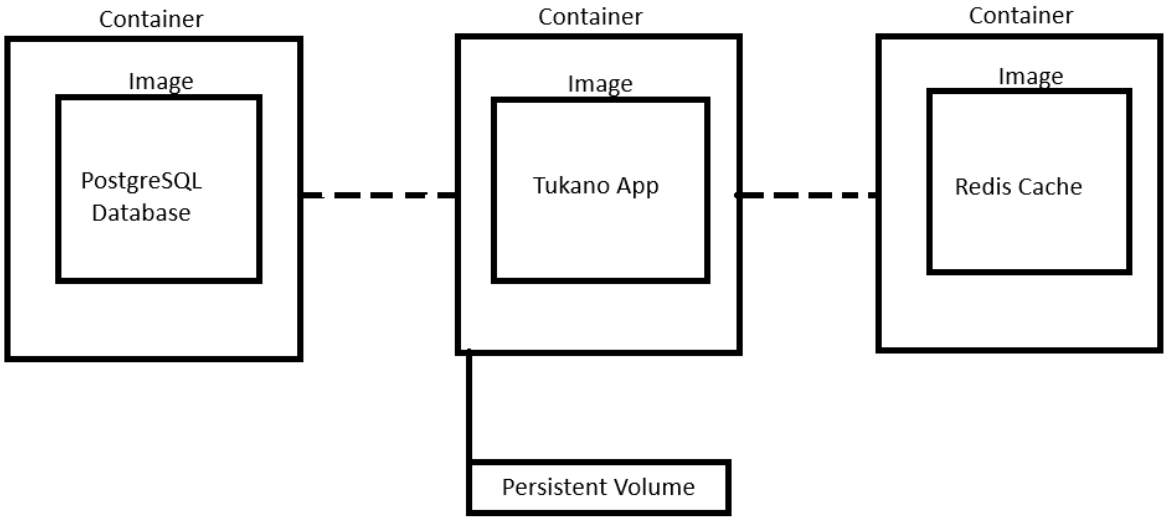

Figure 1 - Project Architecture

Implementation Details

- **Authentication**: In order to make the system more secure, authentication was implemented, requiring users to log in to be able to manage Blobs (adding, downloading or deleting a blob). Only administrators can manage blobs.
- **Kubernetes**: In total, we utilize 3 kubernetes pods to run our whole project. Having a dedicated container for the web app, the database and the cache. The web app container has the image of our Tukano project, while the database and cache utilize an already built postgreSQL image and Redis Cache image, respectively.
- **Redis Cache:** Caching implementation follows the same principles as the previous project deliverable, with the added functionality of caching for authentication functions.

Performance Analysis

Testing Setup

The testing setup consisted of a couple artillery tests using both scripts and CSV files in order to bulk insert either users or shorts.

New vs. Old Performance

We performed standard inserts and gets to assess speed and reliability when comparing to the previous solution, and, we found out that this solution was quicker and more stable when it comes to

```
All VUs finished. Total time: 13 seconds

--------------------------------
Summary report @ 17:13:52(+0000)
--------------------------------

http.codes.200: ..................................................................... 43
http.codes.409: ..................................................................... 7
http.downloaded_bytes: .............................................................. 5001
http.request_rate: .................................................................. 7/sec
http.requests: ...................................................................... 50
http.response_time:
  min: .............................................................................. 7
  max: .............................................................................. 642
  mean: ............................................................................. 90.4
  median: ........................................................................... 19.1
  p95: .............................................................................. 459.5
  p99: .............................................................................. 608
http.response_time.2xx:
  min: .............................................................................. 7
  max: .............................................................................. 642
  mean: ............................................................................. 102.3
  median: ........................................................................... 22.9
  p95: .............................................................................. 459.5
  p99: .............................................................................. 608
http.response_time.4xx:
  min: .............................................................................. 8
  max: .............................................................................. 29
  mean: ............................................................................. 17.3
  median: ........................................................................... 10.9
  p95: .............................................................................. 27.9
  p99: .............................................................................. 27.9
http.responses: ..................................................................... 50
```

Figure 1 - Bulk user insert (PostgreSQL in Kubernetes)

From figure 1 we can see that around 80% of users are added successfully, which is an improvement from the previous 70% we had with Postgre, additionally, the time to add improved by 8 seconds when compared to the implementation done on the first phase.

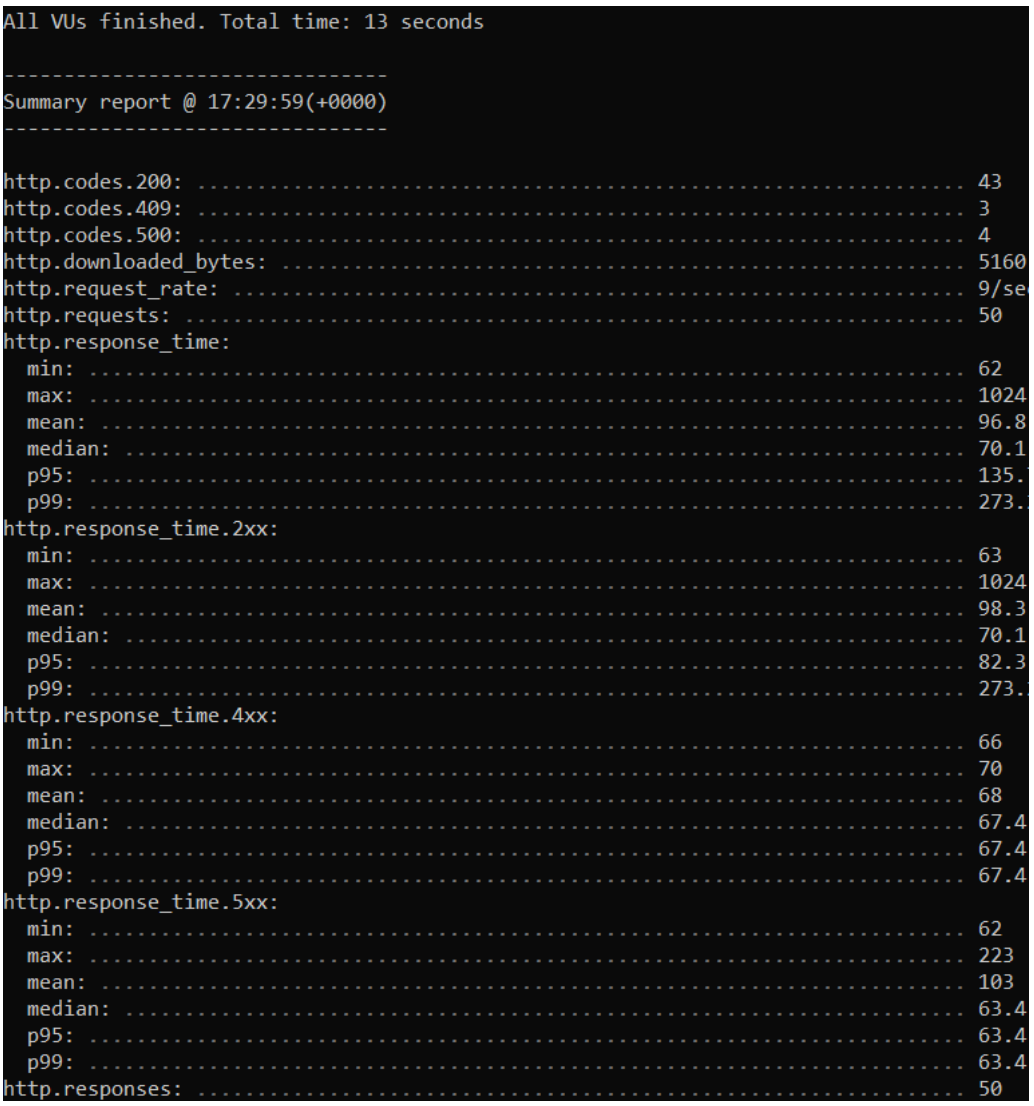

Figure 2 - Bulk user insert for NoSQL (previous implementation)

The new implementation in Kubernetes can certainly be compared to the NoSQL implementation, present in figure 2, done in the first phase, as both have similar success rates and insertion times, even though the new Kubernetes implementation has a slightly lower throughput (7 requests per second instead of 9).

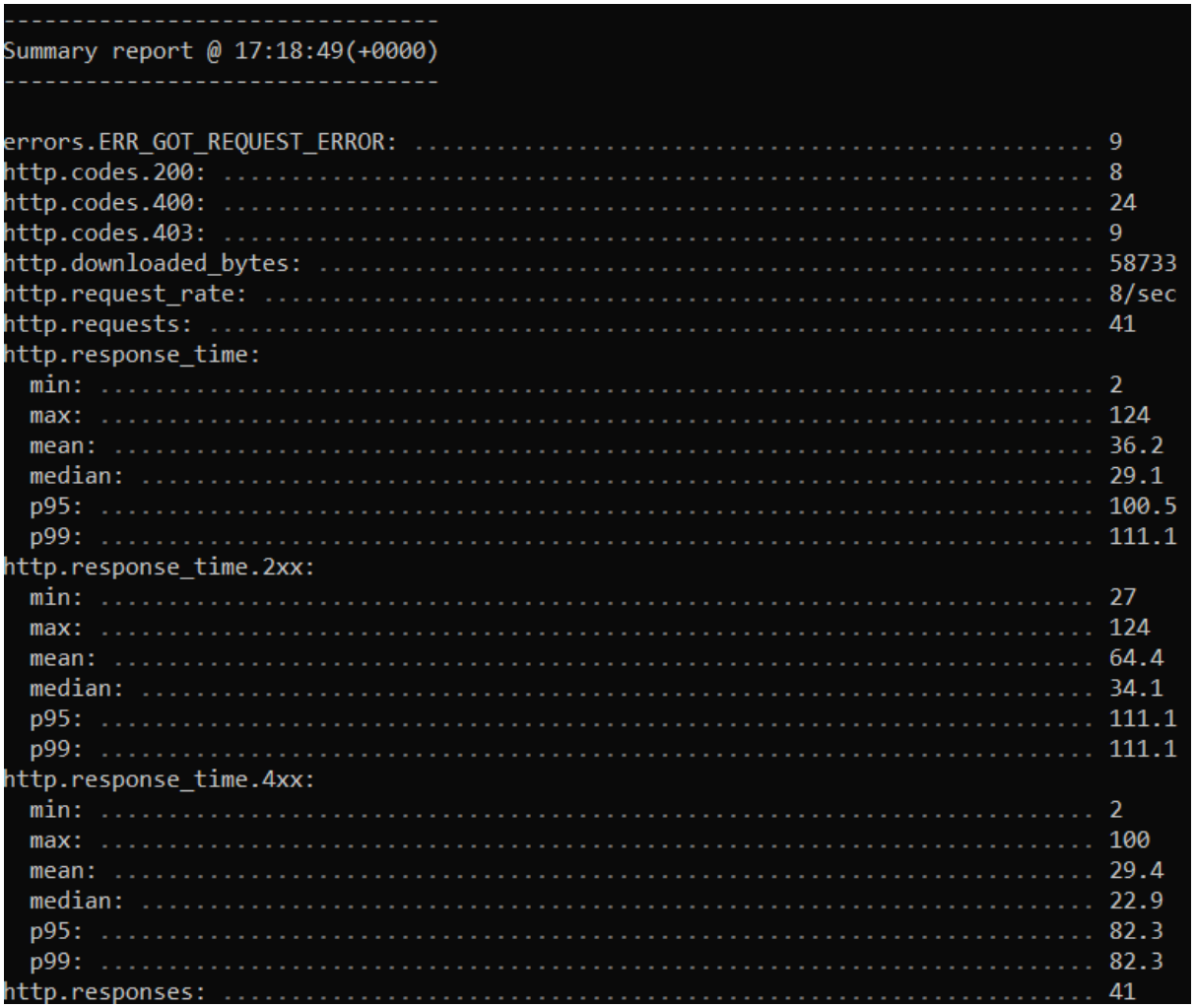

Figure 3 - Bulk short inserts for PostgreSQL

We attempted to test bulk insertion of shorts, however, the results were inconclusive due to some unexpected challenges during testing in Minikube. Specifically, we observed that data appeared to be highly volatile, with certain users/shorts disappearing and requiring recreation. This behavior could be attributed to the testing environment or configuration issues. Despite these challenges, the observed request rate remains consistent with that of user requests, indicating that the system can handle multiple requests per second.

Figure 4 - Testing user functionalities for PostgreSQL in Kubernetes

Observing figure 4 we can take away that the CRUD operations for Users are functional and relatively quick (even though we are doing a single test).

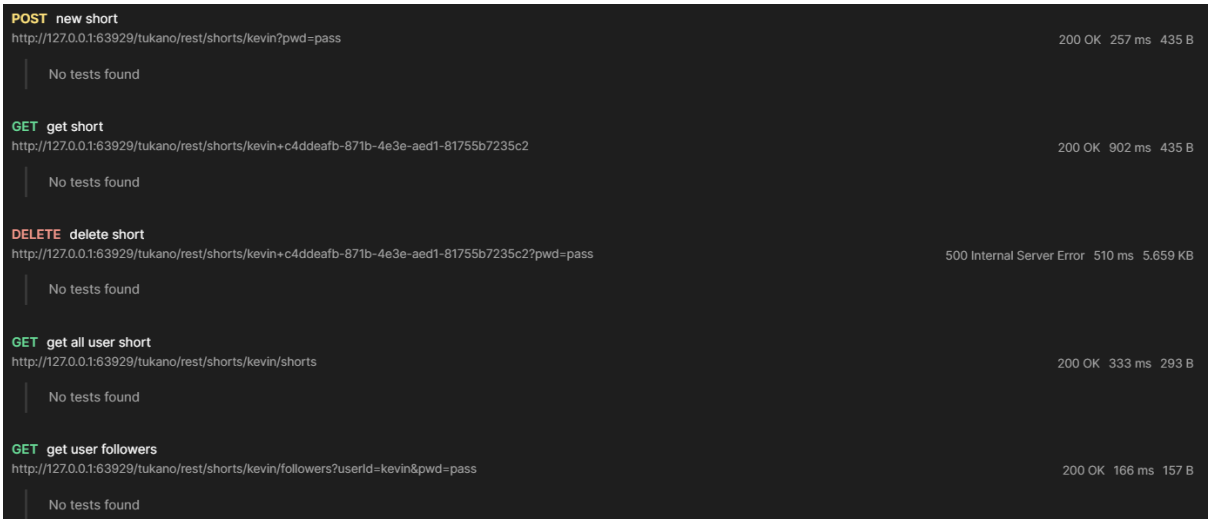

Figure 5 - Testing short functionalities for PostgreSQL in Kubernetes

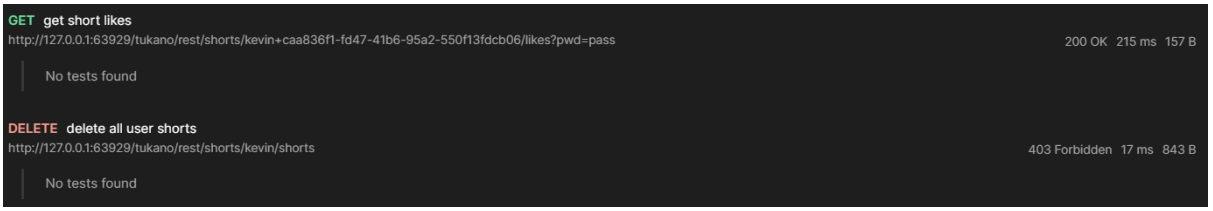

Figure 6 - Testing user functionalities for PostgreSQL in Kubernetes

The key takeaway from figures 5 and 6 is the same as figure 4, operations to manage shorts are functional and most work as intended. Still, we did not perform many artillery tests for these functionalities as we did not understand well how to associate smoothly and dynamically short operations.

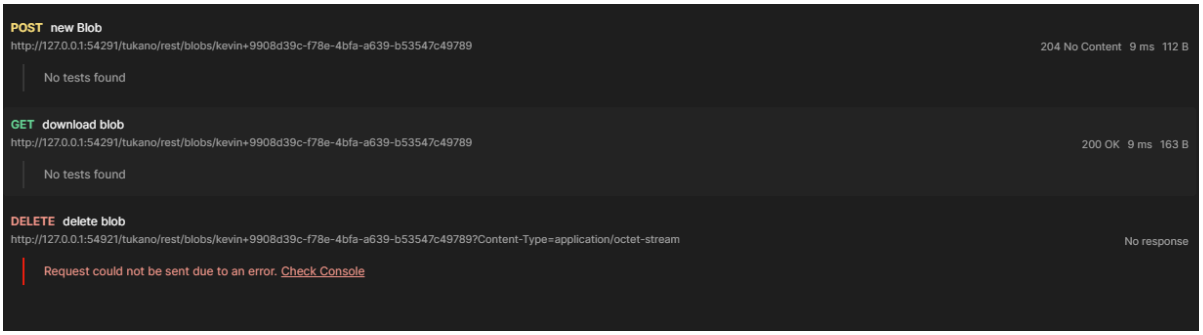

Figure 7 - Testing blob functionalities for PostgreSQL in Kubernetes

Blob storage operations were even harder to develop and test, however, even though its a limited implementation, it requires a fully authenticated admin user. That being said, the delete operation works intermittently.

Evaluation and Discussion

Summary of Results

Conversion to Kubernets includes:

- Pod + Service for the TuKano application (w/ authentication);
- Pod + Service for the Database;
- Persistent Storage for Blobs.

From the performance analysis we were able to conclude that NoSQL + Redis Cache is the best approach given the implementation we performed as it delivers a high availability which is a must for any social media application.

Challenges & Improvements

During this part of the assignment we had, like in the first part, a lot of difficulties regarding the TuKano project itself. The implementation of the authentication was very straightforward and without issues, however, the deployment on Azure using Kubernetes proved to be challenging as the endpoints were not being mapped as we expected. Eventually, we were able to resolve the issue and move forward with the project, but it cost us a lot of time which resulted in a late delivery.

Another issue that we ran into was that the table “User” utilizes a reserved name for the postgreSQL image we were using. Because of this, the User table on our database was not being created, which we verified by accessing the database container via the command terminal. We were able to solve this issue by explicitly naming our table and columns in the User class via the jakarta.persistence package.

Conclusion

We were able to utilize Kubernetes and Azure to deploy a functional backend for the TuKano application, with a proper architecture and authentication. The usage of Kubernetes allowed us to separate the application into different services which provides numerous benefits (improved scalability, easier fault isolation, simplified deployment, better resource management, etc).

As the development went on, we observed a spike in costs that was directly tied to the usage of Azure containers which was expected but also another little challenge to handle.

In the end, we couldn’t fully convert all functionalities of the TuKano application to work with Azure and Kubernetes but we feel like we learned a lot about container orchestration, service management, and kubernetes architecture, which will definitely help us in the long run.