Cloud Computing Systems

2024/2025

Tukano Project

Project Assignment #1

Authors:

70070, Pedro Peralta
71794, Tsimafei Nestsiarau

Professor:
Sérgio Duarte

November 10, 2024

Introduction

This project gave us the chance to dive into the world of cloud computing and explore how its tools can transform a web application into something faster, more scalable, and more reliable. Our task was to bring an existing app, called Tukano, to Microsoft Azure's cloud platform, adapting it to leverage Azure's powerful resources while following today's best practices in cloud engineering.

Tukano is a social network inspired by video-sharing platforms like TikTok and YouTube Shorts, where users upload and share short videos. The app's social side comes from users following one another, which builds up a personalized video feed for each person. Moving Tukano to Azure involved updating key services to manage users, videos, and media files in the cloud, using Azure tools like Blob Storage and CosmosDB.

In this report, we'll look at how Azure's resources helped us improve Tukano's performance and user experience. We'll dig into how cloud services can help an app scale to handle high demand and what benefits this brings to users and developers alike.

Unfortunately, we haven't managed to run the artillery tests provided by the professor Gallagher since we couldn't deploy the app using Azure due to some unexpected error ("The subscription **** could not be found."), Tomcat was not accessible for some reason as well. We haven't implemented our own tests before that issue. In the comparison section, we had to analyze the performance and make a comparison between different solutions assuming the most probable outcomes. We can upload the real results later if you approve it (the results shouldn't diverge with our analysis).

Structure and Azure Resources

In the Tukano project, we work with three main resources: **BlobsResource**, **ShortsResource**, and **UsersResource**. These resources handle key functions, and they needed some updates to run smoothly in the Azure cloud environment.

Starting with the user resource class, it includes methods for all the CRUD actions related to users. The first step here was to set up the database to use Azure CosmosDB. We managed to implement two alternative storages: CosmosDB for NoSQL and PostgreSQL, however, we were unable to resolve issues connected with PostgreSQL version. We created a new helper class called **CosmosDBLayer** in the *utils* folder and updated the main **DB** class to use this new **CosmosDBLayer** for NoSQL database interactions. A **Hibernate** class was used to operate with PostgreSQL database, we were unable to establish the correct connection as it was expected, but the could observe a database response on Azure Portal monitoring.

Once we got Users working with the first version of CosmosDB, it was easy to make similar updates for Shorts since both resources use similar database functions.

For the Blob resource, which stores our media files (like videos and images), we needed a way to store this data in the cloud so it could be displayed in user feeds. We set up Azure Blob Storage to handle this, creating a new implementation to replace the original system that was using local file storage.

We set up Azure Cache for Redis in order to store recently accessed users and shorts. The implementation of cache level is written in **RedisCache** stored in the *utils* folder. When user/short is created/uploaded or retrieved, it is temporarily stored in the cache layer by (key, value) pair. The usage of Azure Cache should offload the main database, improving performance of a Tukano app. Unfortunately, we haven't documented a performance comparison of system with cache and without due to some unexpected problems with deployment using both Azure and Tomcat.

Tukano Baseline vs Tukano in Azure

Comparing the two versions of the **Tukano** application: the **Baseline** version (original) and the **Azure** version (migrated to the cloud). Let's analyze and compare because each version has its own advantages and disadvantages.

Baseline Version:

The Baseline version of Tukano is the original application that runs in a local environment or on a centralized server. This version is simpler and more straightforward, but has some limitations when it comes to scalability and performance on a large scale.

Advantages:

Simple Configuration	The Baseline version has a more straightforward architecture, with data and services located on a single server, which makes initial configuration and development easier
Initial Cost Low	As it runs in a centralized environment, the initial cost can be lower, especially if there is no need to scale to many users or regions

Disadvantages:

Limited scalability	The Baseline version is not designed to support large amounts of traffic. As users increase, performance decreases rapidly, as all data and functions depend on a single server
High Maintenance	All operations, including backups and security updates, need to be done manually, which is time-consuming
Regional Availability	If the application receives global access, the centralized version can present latency for users in different regions, impacting the user experience

Azure Version with NoSQL data model:

The Tukano Azure version is the application migrated to the Azure platform, using cloud services such as CosmosDB for data storage and Blob Storage for media files. This version is designed to be more scalable, faster and more flexible

Advantages:

Dynamic Scalability	Using Azure, Tukano can automatically grow to support more users without compromising performance. Services such as CosmosDB and Redis Cache help manage large amounts of data efficiently
High Performance	Latency is lower and response time is faster, especially in a distributed environment. Using Azure Cache for Redis helps frequent content load faster, which improves the user experience
Global Availability	Azure's geographic replication allows users from various regions to access the application with lower latency, improving global performance
Less Maintenance Required	The Azure platform automatically handles updates, backups and security, reducing maintenance effort

Disadvantages:

Operating Cost	The use of cloud resources can increase operating costs as usage grows. Resources such as CosmosDB and Blob Storage have ongoing costs that must be monitored
Configuration Complexity	Configuring the application in the Azure environment requires greater technical knowledge and initial configuration time, especially when managing services in different regions and optimizing for performance

In general, for an application like Tukano, the Azure version offers significant advantages, especially if the goal is to have a global user base and the need to scale quickly. The ability to scale automatically and global availability improve the user experience and make Tukano more prepared to grow.

For an application with a limited number of users or a local focus, the Baseline version may be more practical and cost-effective in the short term. Therefore, when we are aiming for continuous growth for our application and are based on a larger number of users, it is recommended to migrate the application to Azure, due to the benefits in terms of performance, scalability and simplified maintenance.

CosmosDB for NoSQL vs PostgreSQL

Overall Performance comparison:
PostgreSQL offers higher efficiency when the structural relations between various entities are required. It offers better consistency when writing and retrieving data which is essential for some operations like followers or likes counters, updating follower's feed when a new short is uploaded. Although, with greater consistency comes lower performance which might be prior in some cases. Usage of a relational database model such as PostgreSQL might be excessive for the Tukano model (at least at the current stage) since relations between Users and Shorts entities are not so complex. NoSQL data model provides better performance because of lower level of consistency compared to the relational model. In our opinion NoSQL is more preferable as it has better overall read and write performance.

Overall Latency Comparison:
PostgreSQL has higher latency compared to NoSQL due to the relational model approach. Additionally, NoSQL model will have a lower latency when scaling due to its optimization for horizontal scaling. Theoretically, PostgreSQL solution may achieve a similar latency but it requires more complex configuration when scaled.

NoSQL with vs without cache:
NoSQL model with cache attenuates latency when frequently used data is requested, specifically, users and shorts in Tukano app. Usage of cache offloads the primary database leading to higher availability and lower latency for users due to the possibility to scale-up an application layer. On the other hand, unwise management of cache may lead to data irrelevancy, for instance, when a user is trying to access a deleted short that is still in cache memory. Data irrelevance issues can be easily resolved with an appropriate expiration time (we added such a timer but didn't test it). Absence of cache layer impacts negatively on latency/performance at peak load. ComsosDB for NoSQL can still be used without cache because of great optimization, however, adding a cache layer is recommended.

PostgreSQL with vs without cache:
Adding a cache layer for PostgreSQL data model significantly increases read performance. Cache for Redis achieves better performance/latency by holding "hot-accessed" data in-memory rather than standard disk-based queries. As with NoSQL, data irrelevancy issues should be taken into account. By eliminating the cache layer, performance and latency are significantly worse, this difference is more noticeable compared to NoSQL without cache.

Summary of performance evaluation:
In general, Azure's API for NoSQL shows a better performance and offers an opportunity for great scalability, both of these factors are crucial for an application like Tukano. On the other hand, the API for PostgreSQL offers a great level of consistency by default having a worse performance both with and without cache. Our final decision is that

usage of CosmosDB for NoSQL would be preferable as it has better overall performance and scalability. CosmosDB for NoSQL with Cache for Redis is the best solution mentioned in this report .