Cloud Computing Systems – 2021/22

Project 1

1. Introduction

The goal of this project is to understand how services available in cloud computing platforms can be used for creating applications that are scalable, fast, and highly available.

The project will consist in the design and implementation of the backend for a system like discord / slack and companion testing scripts.

The system will manage a set of public and private **channels**, where each channel consists of a partially ordered set of **messages**. While any user can access a public channel, a private channel is only accessible by the members of the channel.

For implementing its features, the system must maintain, at least, the following information:

- **Users**: information about users, including the nickname, name, password, photo, list of subscribed channels;
- **Media**: manages images and videos used in the system;
- **Channels**: information about channels, including name, public / private status, owner, list of members (if private).
- **Messages**: channels' message. Each message must include the sender, the channel it belongs to, the text, (optionally) an associated image/video and (optionally) the message it replies to.

The system must support, at least, the following basic operations using standard REST rules for defining endpoints:

- **Users** (/rest/user): create user, delete user, update user. After a user is deleted, messages from the user appear as been performed by a default "Deleted User" user.
- **Media** (/rest/media): upload media, download media.
- **Channels** (/rest/channel): create channel, delete channel, update channel information.
- **Messages** (/rest/messages): create message, delete message.

Operations that create an entity with an associated media object (e.g., message) can be implemented either by having clients executing two consecutive calls, the first for uploading the image and the second for the corresponding operation; or by executing a single operation that includes the media and the remaining data.

1.1 Environment

The project will be implemented in Azure, using the resources available to students under the Azure student program. Check the documentation in lab 1 for more information on how to claim the credits for you Azure account.

NOTE: Azure services have associated costs. Be careful with the configuration of your services and the cost when you leave services/resources running, specially, those that are more expensive, such as Cosmos DB and Redis.

2. Base version

The base version consists in the basic version of the backend service, that includes the base REST endpoints associated with Users, Media, Channels, Messages.

You must decide how the service information is stored in the underlying storage services / database services, creating the necessary objects/documents.

Besides the base endpoints associated with the previous resources, the service should provide the following endpoint:

- List of channels of a user;
- List of trending channels.

Additionally, optional features presented in section 4.4 will require additional endpoints.

2.1 Services to use

The system should use the following Azure services:

- App service: for creating the REST servers (see lab 1 for more info).
- Blob storage: for storing BLOB (see lab 2 for more info).
- Cosmos DB database: for storing structure data (see lab 3 for more info).
  NOTE: Cosmos DB provides several data models and interface. Students are allowed to use any that they want.

3. Testing

You should develop the necessary scripts for testing your backend service, using artillery. During the labs, a base version will be provided – you should extend it to test all operations of your system.

4. Advanced features

NOTE: make sure that it is easy to turn off the advanced features, so that you can execute tests that compare both – suggestion: have flags that control access to the additional code needed for the advanced feature.

4.1 Application-level caching [Mandatory]

Caching is fundamental for improving the performance of a system by avoiding slow database operations and by avoiding repeating the same computations multiple times.

Improve the base version of your backend by using application-level caching whenever it makes sense. To this end, use the Azure Cache for Redis service.

4.2 Azure Functions [Mandatory]

Use Azure functions to implement some functionality in your application.

Suggestion: Use Azure Functions to perform some periodic computation (e.g., garbage-collection).

Note: Using Azure Functions to solve the next feature does not count.

4.3 Extend your system to run in a geo-replicated setting [Optional – 1 points]

Extend your system for having an efficient geo-replicated deployment in, at least, two data centers.

Note: Cosmos DB allows to specify the data centers in which data should be replicated. The same does not occur for Blob Storage (this was discussed in lab 6). A way to address this issue is to replicate the data in the blob store in multiple data centers using Azure functions.

4.4 Support for advanced search or computations [Optional – up to 2 points]

Extend your application to include some computation-based functionality (only one is needed).

Suggestion: extends your application to support search on the contents of the messages. To this end, integrate your system with Azure Cognitive Search.

Suggestion: Compute the trending topics, based on recently added messages. To this end, use Apache Spark.

Suggestion: For each channel, compute a list of other suggested channels based on the users that post in both channels or the contents of the messages of each channel. To this end, use Apache Spark.

5. REPORT AND EVALUATION

The project report should present the work developed in a clear, concise and complete way (suggested maximum dimension: 6 pages in a 11pt font, A4 with decent margins). The report should not include code, but can include pseudo-code to help presenting some design features. The following structure is suggested:

1. Introduction : briefly explain the context and goal of the system being developed;
2. Design : introduce the design of the system, including the architecture with its main components and how these components interact for implementing the different features of the system;
3. Implementation : briefly introduce any implementation detail that you think it is worth being highlighted;
4. Evaluation: evaluation results (and discussion) using artillery.
5. Conclusions

The report can be written in Portuguese or English.

5.1 Minimum evaluation [Mandatory]

As a minimum, your report should present an evaluation of your system comparing the following settings, using artillery:

a) Application deployed in region West Europe, with caching;
b) Application deployed in region West Europe, without caching;
c) Application deployed in some other region outside of Europe, with or without caching.

5.2 Evaluation in geo-replicated setting [Optional – 1 point]

Evaluate your work by deploying your system in multiple data centers and having clients calling the different replicas.

Note about evaluation: when evaluating your system, use pricing tiers that provide predictable quality of service (e.g., avoid using F1 for app services).

6. GRADING

The mandatory part of the project (including code, evaluation and report) will be graded in up to 16 points (14 points for the code + 2 points for the report). Maximum grading from optional part as discussed in this document.

7. IMPORTANT DATES

5/November – delivery of checkpoint – at this point, you are expected to have implemented the basic functionality + application-level caching;
28/November – final delivery of the project.