Cloud Computing Systems

Diogo Rodrigues 56153, João Vargues 55185, and José Murta 55226

NOVA School of Science and Technology, Portugal

Abstract. The present document is the report for the Cloud Computing Systems course project one. This document has presented all the work developed in a clear, concise and complete way.

1 Introduction

Cloud computing is the delivery of computing services on the Internet in order to create faster innovation services. Cloud computing is a big shift from the usual way businesses think about IT resources. It brings many benefits for both big and small businesses, such as reliability, security, speed and many others. Starting to be used at a global scale, knowing how to work with a cloud computing service is a powerful tool. In this first project we created an application on Azure, the Windows Cloud Computing Service[1]. The application asked in this project to be built is the implementation of the backend for an auction system like EBay. The system we implemented can create and update users, users can create auctions and bid on open auctions and questions can also be placed about a product of an auction.

In this report it will be presented the design of our system, the implementation of the application in Azure and also discuss the evaluation results we gathered using artillery.

2 Design

During the implementation of our application, we tried always to be efficient both in the time that some actions take performing, but also in the resources consumed in Azure.

First, it was necessary to create an initial Azure Application choosing a name and creating a resource group where it will be grouped together with the resources described next. This application was deployed in the west Europe region since that is the closest region to where the clients will be accessing (assuming that the main clients are us, testing our back end service), and using the free pricing tier, F1. This tier was later changed

for the Basic pricing tier, B1 to provide a consistent and predictable quality of service when testing our developed system. In the configuration of our runtime deployment, we followed the suggested given in the labs using Linux as our Operating System, Java 17 as our java version and TOMCAT 10.0 for our server.

Then, we created a Storage account where we store all the images of the users and auctions in a Blob Storage. In order to do this we created a java class MediaResource with the methods to upload, download and get the list of images in the container. This class with the described methods is accessible through the endpoint */media*.

To save all the information about the users, auctions, bids and questions, we created a container for each of these objects in our Azure Cosmos Database. When users, auctions, bids and questions are created, they are added to our Redis Cache. When information about one of the objects is needed in a method, we check if it is saved in the cache first, and if it is not, then we do the get operation in the cosmos database and the respective container. It is relevant to mention that we added a flag to disable the cache when requested to. This is important to better test and evaluate our system, as it will be described in section 5.

Detailing the information and briefly describing the variables of each object stored for each of the four containers:

1. **Users:**
   - String *id*: the unique identifier of the user (can be seen as the primary key of the object)
   - String *name*: the name of the user
   - String *pwd*: hash of the password of the user
   - String *photoId*: unique identifier of the user's photo
   - Set<String> *auctionsIds*: set of identifiers of the auctions that the user created
   - Set<String> *bidsIds*: set of the identifiers of the bids that the user made
2. **Auctions:**
   - String *id*: the unique identifier of the auction (can be seen as the primary key of the object)
   - String *title*: the title of the auction
   - String *description*: the description of the auction
   - String *photoId*: unique identifier of the auction's photo
   - String *user*: the unique identifier of the user that created the auction

- String *endTime*: the date and time of the auction's closure (this information is saved as String for a simpler implementation, but when needed for computing purposes, it is parsed to a Date Object)
- String *lastBid*: the unique identifier of the most recent bid placed on the auction
- String *winnerBid*: the unique identifier of the winner bid of the auction (only computed when the auction closes)
- String *status*: the status of the auction (the auction can be open, and in this time, bids on the auction can be made, or closed, where no more bids can be made)
- Set<String> *listBids* : set containing the id of the bids placed on the auction
- Map<String, Set<String>> *listQuestions*: map that contains the identifier of the question as key, and a set of the replies of each question as the value)

3. **Bids:**
   - String *id*: the unique identifier of the bid (can be seen as the primary key of the object)
   - String *auctionId*: the unique identifier of the auction where this bid is place
   - String *user*: the unique identifier of the user that is placing the bid
   - String *value*: the monetary value of the bid (converted to float when operations need to be done over this variable)
4. **Questions** (it is important to mention that, in our implementation, this object is used for both the questions and replies of the questions as described in the Implementation section of this report):
   - String *id*: the unique identifier of the question/reply (can be seen as the primary key of the object)
   - String *auctionId*: the unique identifier of the auction where this question/reply is placed
   - String *userId*: the unique identifier of the user that is placing the question/reply
   - String *text*: the text of the question/reply being made

To save the adequate data on the respective containers, it was necessary to create a java class to contain the desired data, together with a DAO class for each of the containers. The second class is composed of a Data Access Objects to be able to store the objects on the database and the first one is used to transfer the information to the java client.

We also created an Azure Function App with two functions, one to flush the cache each hour and other to close auctions when the limit date and hour defined in its creation is reached. The function app creates and uses its own storage.

3 Implementation

During the implementation process, some details that are in the final project had to be carefully discussed and tested, and we will now present the ones that we find more relevant to mention.

The first detail that is worth mentioning and explaining is the user authentication. Most of the methods in this application need to have a prior user authentication since it does not make sense, for example, for a user to update himself or to create an auction if he is not validated throw a login before performing this action. For this matter, we created a method called auth, that consume a JSON that includes a user id and its password. If the password given in the json is the same of the one saved in the cosmos database for that user, a cookie is created and saved in our system's cache, and it will be used to validate every user that tries to perform a creation of auctions, bids, questions, replies and update of user and auctions.

We also find relevant to explain some of the information we save of each user in the users container. We have all the general information that a user needs to have like name and password, but we decided also to have two sets of strings, one to save the identifiers of the users' auctions, and another to save his bids. We find this approach easier and efficient in order to be able to perform some actions like listing all the bids performed by a given user and the auctions that a user has created.

Regarding the auctions, we added a lastBid variable, that whenever a new valid bid is made on that auction, it saves the bid id. This is helpful for when a bid is closed, the WinnerBid that is set to null until this point can be instantly updated to the value of the lastBid. Also on the Auction we have a map that stores the Questions and the Replies. We only save its ids, and this procedure shows to be very helpful when we need to list all the questions from that auction, not needing because of this to go to the cosmos database and search for every question made on that auction.

Lastly focusing on the questions and replies, it is important to mention that we decided to create only one data object named Question, but it can either be a question or a reply, since there is no particular differences

between questions and replies in order to create a new data class just for reply.

Finally, we implemented the Azure Cognitive Search with two methods with a specific endpoint for each. One receives a String to execute a search query on the auctions and returns a set with the auction ids that are have some relation with the search query in their description text . The other receives an auction id on the path url and searches for auctions that are related to the one given, returning also a set with auction ids that verify such similarity on the description.

4 Evaluation

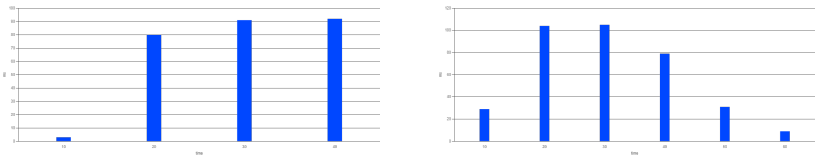
In order to evaluate and test our implementation we used both postman and artillery. Postman, which is an API platform that is used for testing, presenting a GUI to test HTTP requests[2], was only used for simple and one-time tests however, artillery, which is an open-source performance testing software, that contains a set of tools to test web applications, returning fast and reliable responses[3], was used to exhaustively test our system and all the details of it.

As suggested, we tested our implementation using different deployment settings. For that, we tested and evaluated the results of the application deployed in the West Europe region, with both cache enabled and disabled, as well as deployed in the Central Canada region with cache. Our test battery is composed of 6 different tests:

1. Create Users: this test consists in creating 20 users by uploading its image and storing the user's information in the database as well as in a text file (also in cache if enabled)
2. Update users: this test consists of updating the data of a single user that was previously created and saving the new information to the database and the text file (also in cache if enabled)
3. Delete users: this test consists of deleting the data of a single user previously created and updating the database and text file (also in cache if enabled)
4. Create auction: this test consists on creating 10 auctions by first selecting a user to create the auction, then upload the auction's photo, and lastly creating the auction with the desired data and store it in the database (also in cache if enabled)
5. Create auctions: this test is an extension of the previous one where bids, questions and replies are also made and stored in the respective containers of the database (also in cache if enabled)

6. Workload 1: this test consists of retrieving the list of auctions that a user has created as well as the questions, replies and bids of this auction and creating more replies to certain questions

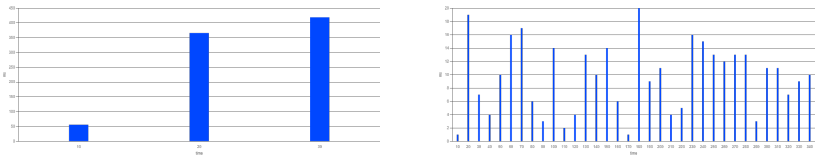
To evaluate the performance of our implementation we focused on the results of the first, fifth and sixth tests. Comparing firstly the results obtained when the application was deployed in West Europe Region, with cache enabled and with cache disabled, it is noticeable that the response time of each HTTP increases when the cache is not enabled, in some cases almost doubling. In the worst case, the maximum response time rose about 8 times when testing with the cache disabled. As it is presented in Fig.1 it is possible to check that the response time of the requests of the Create auctions test grew from 40ms to 60ms.



(a) Cache (b) No cache

Fig. 1: Results of the response time of the Create auctions test (appication deployed in West Europe Region)

Now contrasting the results obtained with the application running with cache, deployed in West Europe and Central Canada, the difference in response times is even larger, with responses times increasing by 10 times as it is presented in Fig.2, which represents the results obtained when performing the Workload1 test. These results would be even higher if the application was deployed using no cache.



(a) West Europe (b) Central Canada

Fig. 2: Results of the response time of the Workload1 test (with cache)

5 Conclusion

Cloud Computation has, each day, a bigger importance on developing systems because the cost is smaller, speed is given on demand as the resources need scalability, bigger reliability and security [1]. With this project we had the opportunity to work with the Azure, one of the biggest cloud computing providers in the world, developing a back-end system for auctions control and maintaining it with resources like Azure Cosmos DB, Redis Cache, Azure Blob Storage,Functions and Cognitive Search.

These tools are very important while developing a cloud system, Blob storage helps to store binary files, Azure Cosmos DB allows to have a consistent structure with containers to organize our data, Redis Cache allows server side caching decreasing the number of requests done to the database that are way more expensive and can took more time. Azure Functions are a serverless system which needs much less written code and resources only used when needed saving cost and infrastructure and Azure Cognitive Search offers a search experience through texts in the database (in our case, it can be used in other tools). Our application was also launched in other continent with just a few configurations changed, showing how easy it is to test new needs to the app owners. All of this, if not developed in a cloud computing system, would have a much higher degree of difficulty and it would take more time and a higher investment (if done in a enterprise environment), revealing the use of the Cloud development.

References

1. What is cloud computing? https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-cloud-computing/#benefits Last accessed 18 November 2022.
2. What is Postman? https://www.postman.com/product/what-is-postman/ Last accessed 20 November 2022.
3. Artillery https://www.artillery.io/ Last accessed 20 November 2022.