WEB AND CLOUD COMPUTING 2017

Project: Uber for Bikes

Group 18

Authors
Thom Carretero Seinhorst
Alexandru Tatarov
Anastasia Serebryannikova

November 3, 2017

1 Introduction

The application will represent a map on which the positions of bikes will be shown. These bikes are used by people for transportation. Apart from the map, there is a table with the information about the bikes (the address). When a bike is taken, it's corresponding point on the map disappears (because it is no longer available). When the trip is finished the bike appears back on the map.

2 Architecture

In Figure 1, a broad overview of the architecture of the application is given. When the docker swarm mode is used, this structure is duplicated at every node because we use the global mode setting.

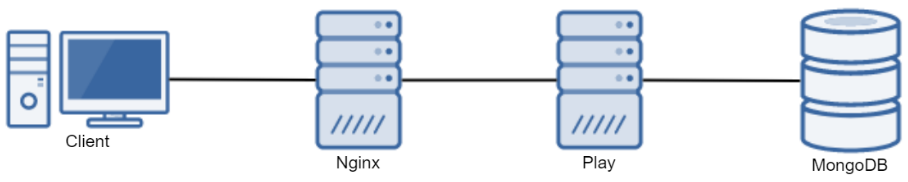


Figure 1: Overview of the architecture of Uber for Bikes

For our webserver we have written the main component in Scala. Scala is a functional programming language supporting high scalability. We used the Play framework which fits our purposes very well as it is designed for developing web applications. As well as this. it ensures the integration with the database and the other components.

For our project we used the MongoDB database with a Scala ReactiveMongo driver. ReactiveMongo provides fully non-blocking and asynchronous I/O operations. ReactiveMongo is designed to avoid any kind of blocking request. Every operation returns immediately, freeing the running thread and resuming execution when it is over.

In the frond-end, AngularJS and Bootstrap are used for providing a nice user interface. The UI serves a simple purpose: Renting and unrenting bikes. The main view of the UI is a Google Maps view. It is currently set to the location of Groningen, but could eventually be used to show the region where the user resides. In our project we use an external third party service. The service of Google is needed to convert a textural representation of a location into the corresponding geographical representation of the GPS system. This conversion is done via the Google Maps API.

2.1 Docker swarm

Docker swarm is a tool that enables high scalability and fault-tolerance. With Docker swarm, it is possible to run applications on a cluster of virtual or physical machines. If one of the nodes or one of the containers fails the other keep functioning and can handle the requests that are send to them. For every service, there are two modes available, global and replicated. If the replicated mode is used, the user can define the number of replicas of each container. If the global mode is used, every node in the cluster runs one instance of the service. In our application, we use global mode for all the services, which means that if we run our application on two (virtual) machines, each machine will have one instance of every service running on it.

3 Design decisions

3.1 Database

We use MongoDB for our database. We chose to use this database as it is easy to implement and easy to use. We have a MongoDB database with 1 collection, Bikes. The Bikes collection stores all bike locations and their availability. There should be a second database collection, Users, for storing the name, passwords and a list of rented bikes. Due to lack of time, we weren't able to implement this correctly.

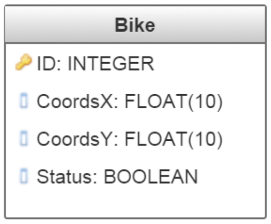


Figure 2: Overview of the database

In the docker swarm mode, the Mongo replica set is used to enable sharing of the data between the instances of MongoDB on different hosts. Replica sets provide redundancy and increase the availability of the data. All write operations are performed to the primary node and then replicated to the secondary nodes, as shown in Figure 4.

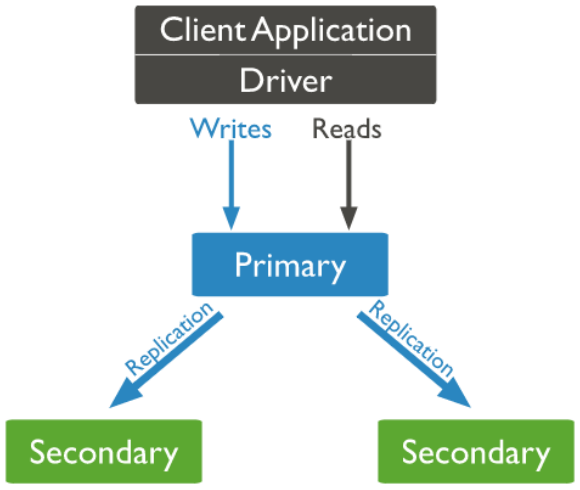
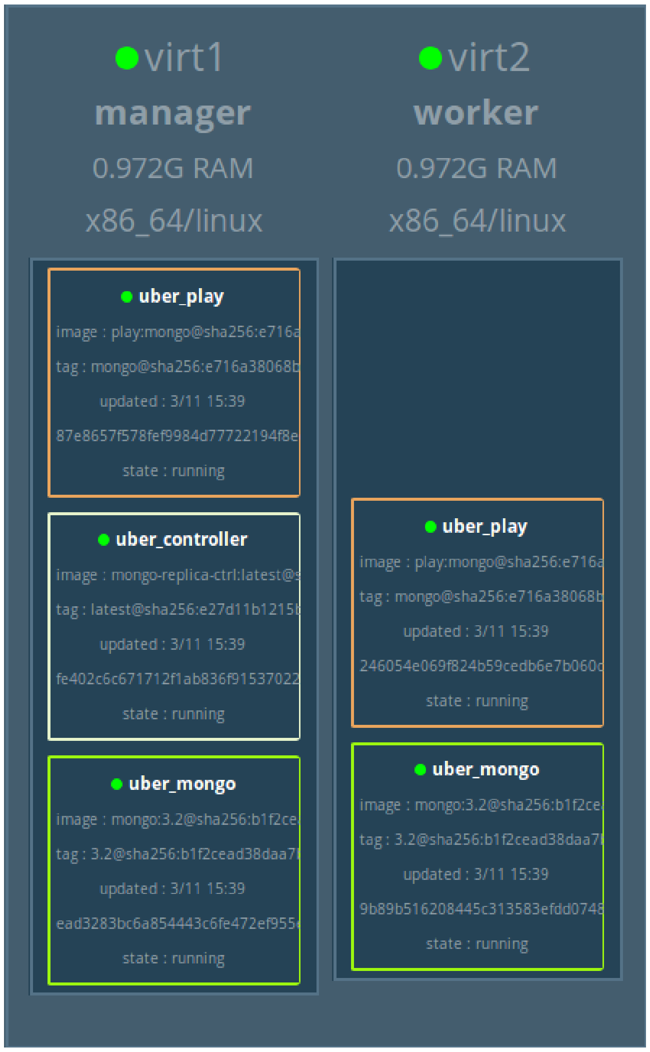


Figure 3: Replica sets in MongoDB

As opposed to the other services that are deployed in the docker swarm mode, MongoDB needs some additional configuration. Docker cannot take care of initializing the replica set and making sure that the data is correctly replicated among different nodes. This means that we have to manage the replica set by ourselves. If the replica set is configured manually there is a high risk that if one of the MongoDB instances goes down and appears again, its IP address will be different and the replica set will need to be reconfigured (new IP address of the node added and the old one removed).This may cause the read and write operations to fail, which is not what we want. So we have found a way to deal with this problem. We have a separate container (called controller) that maintains the replica set up and running and updates the configuration whenever necessary. This container does not need replication and is deployed only once on the manager node. So the backend, when deployed in the swarm mode, looks as it is represented in **??**.


virt1
manager
0.972G RAM
x86_64/linux

uber_play
image : play:mongo@sha256:e716a
tag : mongo@sha256:e716a38068b
updated : 3/11 15:39
87e8657f578fef9984d77722194f8e
state : running

uber_controller
image : mongo-replica-ctrl:latest@s
tag : latest@sha256:e27d11b1215b
updated : 3/11 15:39
fe402c6c671712f1ab836f91537022
state : running

uber_mongo
image : mongo:3.2@sha256:b1f2ce
tag : 3.2@sha256:b1f2cead38daa7
updated : 3/11 15:39
ead3283bc6a854443c6fe472ef955
state : running

virt2
worker
0.972G RAM
x86_64/linux

uber_play
image : play:mongo@sha256:e716a
tag : mongo@sha256:e716a38068b
updated : 3/11 15:39
246054e069f824b59cedb6e7b060c
state : running

uber_mongo
image : mongo:3.2@sha256:b1f2ce
tag : 3.2@sha256:b1f2cead38daa7
updated : 3/11 15:39
9b89b516208445c313583efdd0748
state : running

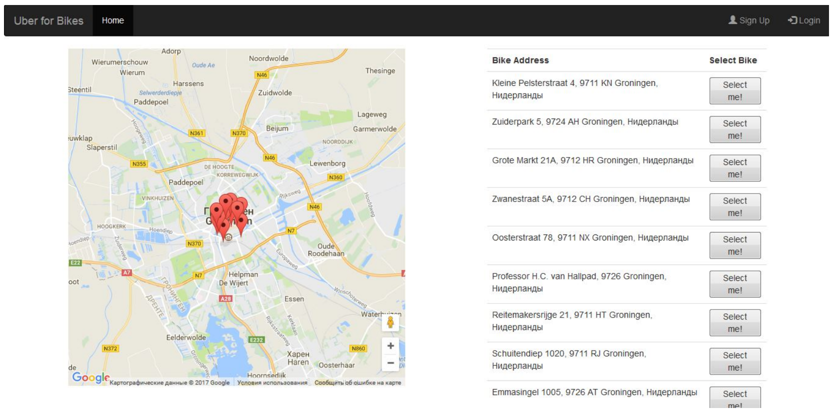
Figure 4: MongoDB replication in docker swarm mode

3.2 Webserver

We chose Play as the framework for back-end development of our application mostly because it supports asynchronous I/O. There were two main causes for choosing Scala: the first is the possibility to learn a new programming language and the second is professor's incentive to use Scala for back-end development. In the end we succeeded in building a working back-end. We benefited from online completed projects we found on the internet as it aided our understanding and development of the final product. During the development we had difficulties connecting our database with Play as well as using WebSockets for live data streaming.

3.3 Client

Angular is a framework for building Web single-page applications and Bootstrap is an easy to use front-end web framework for designing web applications. These are the main reasons why we chose to use these frameworks. Other advantage of using Angular is its highly readable and comprehensive code. It was an important factor because we wanted something that will take as little time to learn as possible. We created the project using Angular Cli and used components to create the navigation bar, map component, and two (login and registration) forms (unfortunately not seen in the final layout). Even if it is not linked to framework itself, we found useful the large community of developers using Angular as online answers helped us solve most encountered problems.



Bike Address	Select Bike
Kleine Pelsterstraat 4, 9711 KN Groningen, Нидерланды	Select me!
Zuiderpark 5, 9724 AH Groningen, Нидерланды	Select me!
Grote Markt 21A, 9712 HR Groningen, Нидерланды	Select me!
Zwanestraat 5A, 9712 CH Groningen, Нидерланды	Select me!
Oosterstraat 78, 9711 NX Groningen, Нидерланды	Select me!
Professor H.C. van Hallpad, 9726 Groningen, Нидерланды	Select me!
Reitemakersrijge 21, 9711 HT Groningen, Нидерланды	Select me!
Schuitendiep 1020, 9711 RJ Groningen, Нидерланды	Select me!
Emmasingel 1005, 9726 AT Groningen, Нидерланды	Select me!

Figure 5: Application layout

4 Discussion

In this chapter we review and reflect on the project.

In the beginning of the project we started with developing an UI. We used a nginx webserver, but we struggled to put it in a docker container and connect it to the back-end. It took us some time to figure it out how exactly to connect these parts. We managed to retrieve data from the database and to present it in the UI. We chose to use Angular as a framework for our UI.

It took us some time to set up the back-end with Play and Scala. Since the programming language Scala was new to all of us, we had to put some effort into it to understand how Scala works.

4.1 Angular

Angular proved to be the easiest component of the project. However not everything was straight-forward. One tricky part was using Ajax calls to retrieve information about available bikes and displaying it. We needed a little more time than expected to use the values from *.subscribe() in some other function.*

4.2 Scala

Scala was completely new for all of us. We were lucky enough to find a project that had some similar functionalities. Using that project as a guideline, we implemented a part of our own back-end functionality that communicates with front-end and the database. The biggest issues we encountered at using Scala was implementing WebSockets.

4.3 Docker

Explain difficulties of docker and/or swarm

4.4 Future work

One important part of the project that is missing is the login and register part for users of the application. Users should be able to create an account and to login. When a user is logged in, they can rent a bike and see their history of rented bikes. Due to the difficulties we encountered there was not enough time to implement this correctly.