

RocketChat & Monitoring deployed in Cloud

A Scalable Microservices-based Web Application in a Public Cloud

Management and Administration of IT Infrastructures and
Services

Team nr.: 06

98678: Bruno Freitas

98844: André Barrocas

102108: Ricardo Ribeiro

Information Systems and Computer Engineering
IST-ALAMEDA

2021/2022

Contents

List of Figures	iii
1 Executive Summary	2
2 System Architecture	4
2.1 Analysis	5
2.1.1 Built With	7
2.1.2 Requirements	7
2.1.3 Constraints	8
2.1.4 Project Structure	9
3 Deployment	12
3.1 Notes	14
4 Versioning	16
5 Work Evolution	17
Bibliography	19

List of Figures

2.1	Overview System Architecture	4
2.2	System Architecture in Kubernetes	5
3.1	All resources from application namespace	13
3.2	All resources from istio-system namespace	14
3.3	Portforwarding to the RocketChat Pod and Curl command to get the RocketChat page directly from the Pod is successful	15
5.1	Docker Compose Version Project Structure	18

Acronyms

GCP Google Cloud Platform

AWS Amazon web Services

mongoDB Mongo database

VM Virtual Machine

K8s Kubernetes

DNS Domain Name System

Chapter 1

Executive Summary

The goal of this work is to show an example on how to deploy and provision tiered (front-end, backend) Microservices-based containerized Web Application on a Public Cloud provider, such as Google Cloud Platform ([GCP](#)) or Amazon web Services ([AWS](#)), using automation tools such as Terraform, Ansible, Pulumi, as well as implementing instrumentation on the applications, services and infrastructure components of the solution, to allow monitoring and logging features by using tools such as Prometheus, Fluentd, etc.

This work studies the RocketChat which is an open-source platform for team collaboration, allowing live chat, direct message and group messages, upload the images and, other interesting features. [1] It is written in full-stack JavaScript and uses the Mongo database ([mongoDB](#)). Mongo is a NoSQL database, using JSON-like documents as schema. [2] To complete a monitoring system it is also implemented. To make the monitoring, the Grafana and Prometheus were used. Grafana is a open source analytics and interactive visualization web application that provides charts, graphs, and alerts for the web. [3] The purpose of Grafana dashboards allows the users to better understand the metrics. Prometheus is an open-source event monitoring and alerting tool that records real-time metrics, with flexible queries and real-time alerting. [4]

This report starts by talking about the System Architecture [2](#), and approaching its requirements and constraints. The next chapter [3](#) shows how to deploy the project and some images of the project running. In the Versioning Chapter [4](#) how the system versions were managed will be explained and for the last part, which is the Evolution Chapter [5](#), it approaches the extra steps the group has taken in order to achieve the

best solution possible in the time given (for example, it is approached the previous version of the project, the docker-compose version).

There is also a Video related to this project, stating how to deploy and giving an overview oh the work done. This video can be found in the following link: <https://www.youtube.com/watch?v=e5jRSE9jek8>

Chapter 2

System Architecture

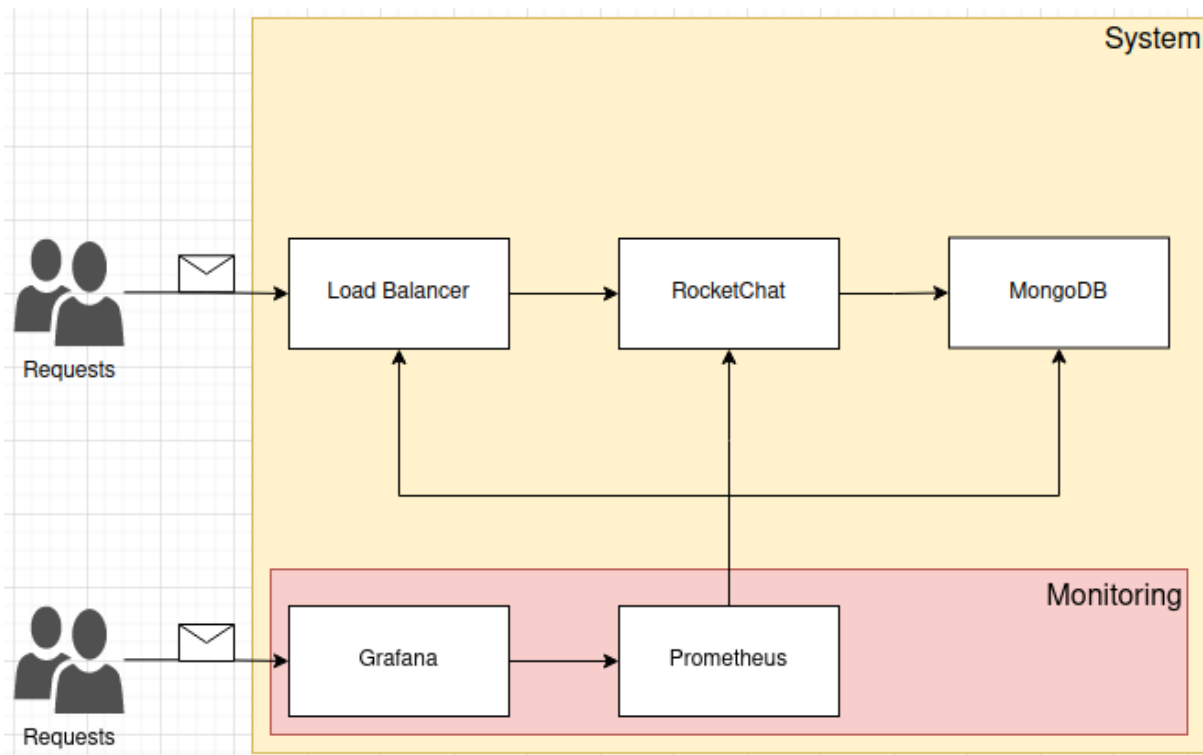


Figure 2.1: Overview System Architecture

By the Figure 2.1, it is possible to see in a generic way how the system is composed. The system is used to satisfy two use cases, the monitoring use case and the "normal" web service use case. The first one is related to how a user can watch all the metrics. The second one is related to how a user can use the web service application.

- **Monitoring use case:** The user will access the Grafana dashboard to see the metric collected. Grafana will interact with Prometheus who is responsible to collect the metrics related with all the components of the system (in the next

section 2.1 it will be explained that the system will be deployed as a cluster and the metrics collected will be from the cluster itself).

- **Web Service Application use case:** The user will access the frontend web page of RocketChat to use the application. The application's frontend will talk with [mongoDB](#) to handle the database operations. The load balancer says that the requests from users will be distributed to the RocketChat Servers (more than one to support more users with better performance)

2.1 Analysis

The system will be built on top of Kubernetes ([K8s](#)) in order to satisfy the system requirements.

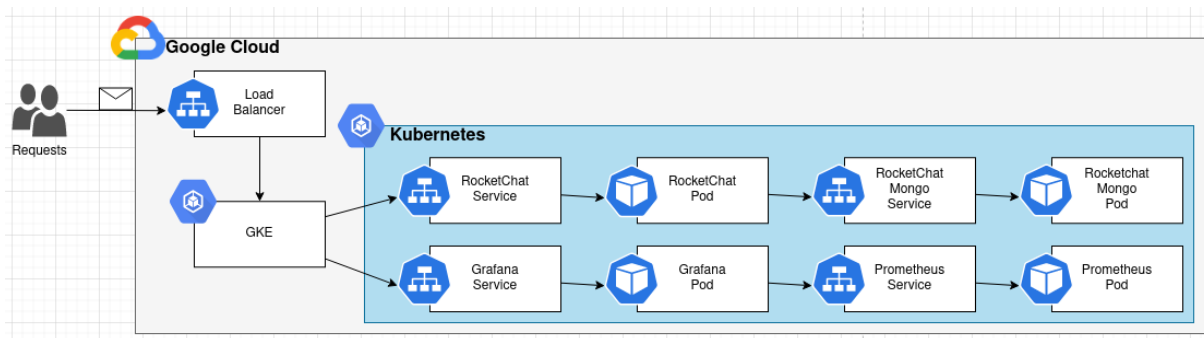


Figure 2.2: System Architecture in Kubernetes

As it can be seen in the Figure 2.2, the [K8s](#) environment is composed with several objects:

- **RocketChat Pods** to handle the requests from the client (RocketChat servers). The Pod contains a RocketChat container inside of it, and the port 3000 is exposed to be possible to contact the RocketChat app;
- **RocketChat Service** to load-balance the client requests for the RocketChat Pods. The Service type is LoadBalancer, provisioning a load balancer for the service. It is exposing port 3000 to outside and forwards traffic to port 3000 to RocketChat Pods;
- **RocketChat Mongo Pods** to handle the requests from RocketChat servers and persist the data into the database, persisted in a volume; The Pod contains the

RocketChat-Mongo container responsible for what has been written behind, and a second container called "rocketchat-mongo-sidecar" which is the sidecar container that will automatically configure the new Mongo nodes to join the replica set;

- **RocketChat Mongo Service** to forward the traffic to Mongo Pods. The Service is a headless service so, there is no load balancing or proxying done. It defines the selector so, endpoint records will be created in the API, and the Domain Name System ([DNS](#)) returns A records (IP addresses) that point directly to the Pods backing the Service;
- **Grafana Pods** to handle the requests for monitoring purposes. The Pod contains a Grafana container inside of it, exposing the port 3000 to be possible to contact the Grafana app;
- **Grafana Service** to load-balance the client requests for Grafana Pods. The Service type is LoadBalancer, provisioning a load balancer for the service. It is exposing port 3000 to outside and forwards traffic to port 3000 to Grafana Pods;
- **Prometheus Pods** that collect metrics about the cluster and its resources. The Pod contains a Prometheus container inside of it, exposing the port 9090 to be possible to contact the Prometheus app;
- **Prometheus Service** to load-balance the client requests for the Prometheus Pods. The Service type is LoadBalancer, provisioning a load balancer for the service. It is exposing port 9090 to outside and forwards traffic to port 9090 to Prometheus Pods.

There is two important factor to notice here. The first one is that there is two [K8s](#) namespaces which organizes the cluster into virtual sub-clusters leading to better management by different teams in a real world environment. With this in mind, an **application** namespace was created, containing the RocketChat Server and Mongo Pods and their Services. The other name space is called **istio-system** and contains all resources related to monitoring, the Grafana and Prometheus Pods, their Services and other important resources. The second important factor is the fact that all these Pods can be scaled up or scaled down depending on the need.

2.1.1 Built With

- **Vagrant** (<https://www.vagrantup.com/>) - Vagrant provides the same, easy workflow regardless of your role as a developer, operator, or designer.
- **Terraform** (<https://www.terraform.io/>) - Terraform is an open-source infrastructure as code software tool that provides a consistent CLI workflow to manage hundreds of cloud services.
- **Docker** (<https://docker.com/>) - Docker is the #1 most wanted and 2 most loved developer tool, and helps millions of developers build, share and run any app, anywhere-on-prem or in the cloud.
- **Google Cloud Platform** (<https://cloud.google.com>) - Reliable and high performance cloud platform from Google.
- **Kubernetes** (<https://kubernetes.io>) - Kubernetes is an open-source container-orchestration system for automating computer application deployment, scaling, and management.

The following technologies were also used in the docker-compose version of the project (previous version - more information in Evolution Chapter 5):

- **Ansible** (<https://www.ansible.com/>) - Ansible is an open source community project sponsored by Red Hat, it's the simplest way to automate IT.
- **Docker Compose** (<https://docs.docker.com/compose/>) - Compose is a tool for defining and running multi-container Docker applications. With Compose, you use a YAML file to configure your application's services.

2.1.2 Requirements

The Requirements to Run the Application are the following software:

R01 Vagrant (latest);

R02 Virtualbox (latest).

To confirm that the software is installed, type in a terminal the following commands:

```
$ vagrant --version
$ vboxmanage --version
```

Vagrant is necessary to create the **management machine**. This is done with a Vagrantfile that uses a Virtualbox provider (this Vagrant file is not prepared to run on systems with architecture ARM arch64, namely Apple Silicon M1 computers - to run this on a machine with these characteristics, another provider such as Docker must be used).

It is also a requirement to have a **GCP account**. With this it is possible to create a project, enable the API (see the note below to further information) and then it is needed to download the credentials.json to the terraform folder (to authenticate the project).

NOTE: It is needed to ENABLE APIs AND SERVICES for the Project, by choosing in the Google Cloud Console API services and next selecting the Dashboard, where it is possible to see a button on the top menu for enabling those services. Then enable **Kubernetes Engine API**.

The Requirements for the System are:

R01 Easy and Automatic Deployment;

R02 Scalability;

R03 Portability;

R04 Flexibility.

For the **R01** requirement, the system is deployed using Terraform. It is only needed to run the commands described in the Deployment Chapter 3 and everything will be deployed. For the **R02**, **R03** and **R04** requirements, the system is deployed on top of Kubernetes which although it is more complex, it allows for autoscaling, it is flexible and portable, meaning it works in any type of underlying infrastructure and it can be used on many different infrastructure and environment configurations.

2.1.3 Constraints

Here are the Constraints for the System:

C01 The configuration of MONGO_URL and MONGO_OPLOG_URL for the RocketChat-Server Pods need to be written manually.

Scaling up the [mongoDB](#) it is automatic and the new Pods will automatically join the replica-set of [mongoDB](#). This happens because of the sidecar container used, explained in the Analysis Section [2.1](#). The **C01** constraints exists because the RocketChat application needs two environment variables called MONGO_URL and MONGO_OPLOG_URL [\[5\]](#). These variables need to declare how to access the [mongoDB](#) nodes of the replica-set. For example, let's say that a new fifth Pod of RocketChat Mongo was added to the replica-set. Now, it is needed to add to MONGO_URL and MONGO_OPLOG_URL how to get to this new Pod, by adding the DNS name of this new Pod, for this example would be something like "rocketchat-mongo-5.rocketchat-mongo.application" (<pod-name>.<service-name>.<namespace>).

All this research is supported by the Kubernetes documentation on how to Run MongoDB on Kubernetes with StatefulSets document [\[6\]](#) that says the following: "Include the two new nodes in your connection string URI and you are good to go. Too easy!"

2.1.4 Project Structure

```
| - Vagrantfile
| - bootstrap-mgmt.sh
| - docs
| - docker-compose-version
\ - tools
    \ - terraform
        | - agisit-2021-rocketchat-06.json
        \ - cluster
            | - gcp-gke-main.tf
            | - gcp-gke-provider.tf
            | - terraform.tfvars
            | - gcp_gke
                | \ - gcp-gke-cluster.tf
            \ - gcp_k8s
```

```
| - monitoring
|   | - grafana.yaml
|   \ - prometheus.yaml
| - k8s-istio.tf
| - k8s-monitoring.tf
| - k8s-namespaces.tf
| - k8s-pods.tf
| - k8s-provider.tf
| - k8s-services.tf
\ - k8s-variables.tf
```

- At the root there is the 'Vagrantfile' needed to create the management VM ('mgmt'). For this, the 'bootstrap-mgmt.sh' script will be executed and, the goal of this script to install the necessary software, or in other words, prepare the management machine. The management machine will be used to do all the operations of the project;
- The 'docs' folder contains the reports from all checkpoints and images;
- the 'docker-compose-version' contains the previous checkpoints' implementation (more information in the Evolution Chapter [5](#));
- The 'tools' folder contains all the project's infrastructure/services files that will be deployed with the help of our 'mgmt' VM. Inside of this folder there is:
 - The 'terraform' folder to provision the infrastructure. Here we have the following folders and files:

The json file with the credentials to be used in the deployment of the cluster. (you should change this file to the credentials of your project!);

The folder 'cluster' that contains everything used to deploy the entire cluster infrastructure:

 - * 'gcp-gke-main.tf', defines the modules used for the deployment and provisioning;
 - * 'gcp-gke-provider.tf', defines the provider to be used;
 - * 'terraform.tfvars', defines some variables used with the Terraform files;

- * 'gcp_gke', folder for the cluster definition file;
 - 'gcp-gke-cluster.tf', defines the cluster as well as outputs certain values.
- * 'gcp_k8s', folder containing the Terraform files defining the pods, services, namespaces and other resources needed.
 - the 'monitoring' folder with :
 - 'grafana.yaml', entire configuration file for the Grafana system.
 - 'prometheus.yaml', entire configuration file for the Prometheus system.
 - 'k8s-istio.tf', ISTIO Service Mesh deployment via helm charts.
 - 'k8s-monitoring.tf', deploys Grafana and Prometheus referring to the yaml files in folder monitoring
 - 'k8s-namespaces.tf', defines namespaces used.
 - 'k8s-pods.tf', defines the Pods to be deployed.
 - 'k8s-provider.tf', defines the providers that Terraform needs and certain configurations for each of them.
 - 'k8s-services.tf', defines the Pods' Services.
 - 'k8s-variables.tf', variables to be used that are obtained after the provisioning of the cluster.

Chapter 3

Deployment

This project was design to have a easy deployment. So, the first thing to do is to go to the project directory and run the following commands to put up and connect to the 'mgmt' - Management Virtual Machine ([VM](#)) (bastion):

```
$ vagrant up
$ vagrant ssh mgmt
```

After this, inside the 'mgmt', run the following command to authenticate in the [GCP](#) (it will give a link, open it on a browser, login in with an IST account, copy the response code and past the verification code in the command line):

```
vagrant@mgmt:~$ gcloud auth login
```

Then, inside the 'mgmt', let's create the infrastructure by running the commands (first will initialize Terraform, in order to satisfy some plugin requirements; then it will create a plan and create the infrastructure by running apply):

```
vagrant@mgmt:~/tools/terraform/cluster$ terraform init
vagrant@mgmt:~/tools/terraform/cluster$ terraform plan
vagrant@mgmt:~/tools/terraform/cluster$ terraform apply
```

After this, all the resources will be created. It is possible to see all the resources created running the command *kubectl get all -all-namespaces* that will get all information about all namespaces available. This command also gives some [K8s](#) resources like *kube-dns* which is not useful for our test case. So let's simple get all resources

focus on the project namespaces. First, let's get all resources related to the application namespace running the command `kubectl get all -n application` producing the output in the Figure 3.1.

```
vagrant@mgmt:~/tools/terraform/cluster$ kubectl get all -n application
```

NAME	READY	STATUS	RESTARTS	AGE
pod/rocketchat-mongo-0	3/3	Running	0	38m
pod/rocketchat-mongo-1	3/3	Running	0	37m
pod/rocketchat-server-76c48f65d7-ng75l	2/2	Running	0	37m

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
service/rocketchat-mongo	ClusterIP	None	<none>	27017/TCP	39m
service/rocketchat-server	LoadBalancer	10.99.254.3	34.89.32.114	3000:30351/TCP	39m

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
deployment.apps/rocketchat-server	1/1	1	1	37m

NAME	DESIRED	CURRENT	READY	AGE
replicaset.apps/rocketchat-server-76c48f65d7	1	1	1	37m

NAME	READY	AGE
statefulset.apps/rocketchat-mongo	2/2	38m

Figure 3.1: All resources from application namespace

In a briefly way, it is possible to see that what was deployed was (more information in Analysis Section 2.1):

- RocketChat Pods to handle the requests from the client (RocketChat servers).
- RocketChat Service to load-balance the client requests for the RocketChat Pods.
- MongoDB Pods to handle the requests from RocketChat servers (are the pods called RocketChat-Mongo).
- MongoDB Service to be forward the traffic to MongoDB Pods.

Second, let's get all resources related to the application namespace running the command `kubectl get all -n istio-system` producing the output in the Figure 3.2.

In a briefly way, it is possible to see that what was deployed was (more information in Analysis Section 2.1):

- Grafana Pods to handle the requests for monitoring purposes.
- Grafana Service to load-balance the client requests for Grafana Pods.
- Prometheus Pods that collect metrics about the cluster and its resources.
- Prometheus Service to load-balance the client requests for the Prometheus Pods.


```

vagrant@mgmt:~/tools/terraform/cluster$ kubectl get all -n istio-system
NAME                                READY    STATUS    RESTARTS   AGE
pod/grafana-79bd5c4498-4sz4t        1/1      Running   0           39m
pod/istiod-687f965684-thz6c         1/1      Running   0           39m
pod/prometheus-9f4947649-fb2nj       2/2      Running   0           39m

NAME                                TYPE                CLUSTER-IP      EXTERNAL-IP      PORT(S)                                     AGE
service/grafana                     LoadBalancer        10.99.243.54     35.242.176.199   3000:30992/TCP                             39m
service/istiod                      ClusterIP             10.99.243.100    <none>           15010/TCP,15012/TCP,443/TCP,15014/TCP       39m
service/prometheus                  LoadBalancer        10.99.253.10     34.142.120.210   9090:31560/TCP                             39m

NAME                                READY    UP-TO-DATE    AVAILABLE    AGE
deployment.apps/grafana              1/1      1              1            39m
deployment.apps/istiod               1/1      1              1            39m
deployment.apps/prometheus           1/1      1              1            39m

NAME                                DESIRED    CURRENT    READY    AGE
replicaset.apps/grafana-79bd5c4498  1          1          1        39m
replicaset.apps/istiod-687f965684    1          1          1        39m
replicaset.apps/prometheus-9f4947649 1          1          1        39m

NAME                                REFERENCE            TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
horizontalpodautoscaler.autoscaling/istiod  Deployment/istiod    1%/80%    1          5          1          39m

```

Figure 3.2: All resources from istio-system namespace

After the deployment, if it is needed to use the tool *kubectl* for any purpose, first it is needed to run the following command, changing the parameters with anchors(<parameter>) with the respective project data:

```

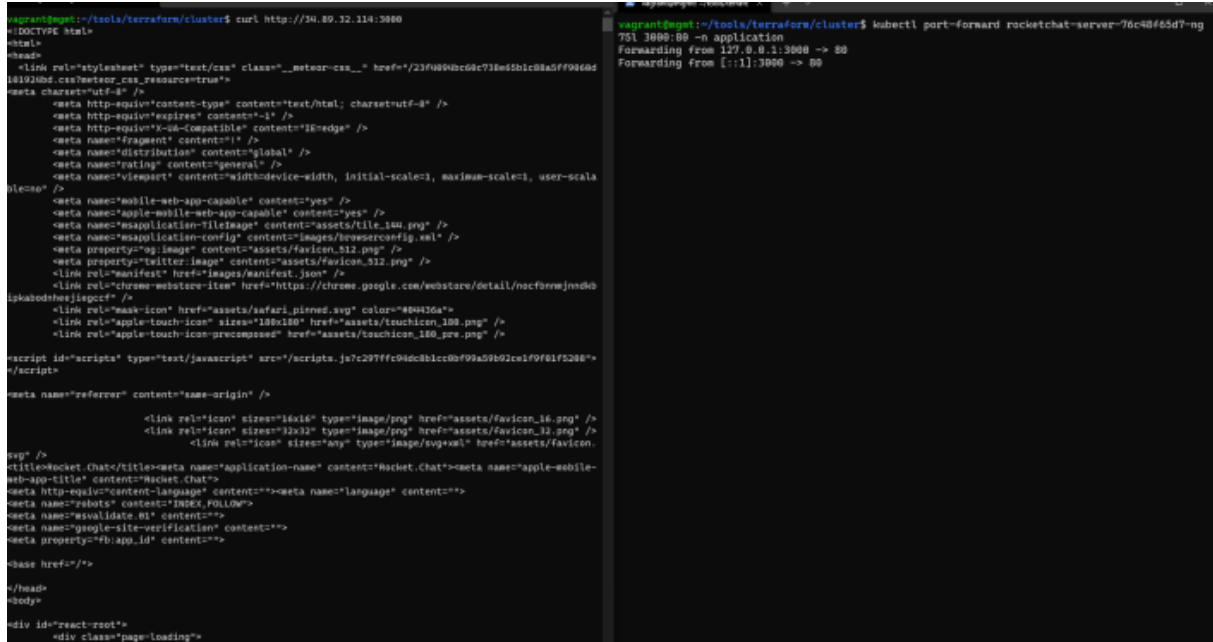
vagrant@mgmt:~/tools/terraform/cluster$ gcloud container clusters
get-credentials <project_name> --zone <project_zone>

```

3.1 Notes

- If it is needed more worker nodes, it is possible to do it by changing the *workers_count* variable in the file *terraform.tfvars*.
- If it is needed more replicas of the RocketChat-server or RocketChat-mongo, it is possible to do it by changing the *replicas* variable in the file *k8s-pods.tf*. For grafana and prometheus the same variable should be change in their respective .yml file inside the monitoring folder. If the cluster is already running it can also be added replicas using the kubernetes command line tool.
- If it is needed to change the region, it is possible to do it by changing the *region* variable in the file *terraform.tfvars*.
- Currently there is a bug by the application's development team that prevents the rocket-chat's setup-wizard from working as intended but that is outside the scope of our work. The problem is in the developer team of RocketChat, because there

is full connectivity between the respective resources of the cluster. As we can in 3.3 see rocketchat-server service responds perfectly well to a client request. If that didn't happen, an 50X error would be given.



```
vagrant@puppet:~/tools/terraform/cluster$ curl http://34.89.32.114:3000
<!DOCTYPE html>
<html>
<head>
<link rel="stylesheet" type="text/css" class="__meteor-css__" href="/23f4894dc6dc73bd5b1c88a5f986d4
16192b8d.css?meteor_css_resource=true"
<meta charset="utf-8" />
<meta http-equiv="content-type" content="text/html; charset=utf-8" />
<meta http-equiv="expires" content="1" />
<meta http-equiv="X-UA-Compatible" content="IE=edge" />
<meta name="fragment" content="!" />
<meta name="distribution" content="global" />
<meta name="rating" content="general" />
<meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1, user-scala
ble=no" />
<meta name="mobile-web-app-capable" content="yes" />
<meta name="apple-mobile-web-app-capable" content="yes" />
<meta name="msapplication-tileimage" content="assets/tile_144.png" />
<meta name="msapplication-config" content="images/browserconfig.xml" />
<meta property="og:image" content="assets/favicon_512.png" />
<meta property="twitter:image" content="assets/favicon_512.png" />
<link rel="manifest" href="images/manifest.json" />
<link rel="chrome-webstore-item" href="https://chrome.google.com/webstore/detail/nccfmrwjndkb
ipkabodshesjiegccf" />
<link rel="mask-icon" href="assets/safari.pinned.svg" color="#00433a">
<link rel="apple-touch-icon" sizes="180x180" href="assets/touchicon_180.png" />
<link rel="apple-touch-icon-precomposed" href="assets/touchicon_180_pre.png" />
<script id="scripts" type="text/javascript" src="/scripts.js?c297ffc94dc8b1ccbf99a59b02c1f9f61f5208">
</script>
<meta name="referrer" content="same-origin" />
<link rel="icon" sizes="16x16" type="image/png" href="assets/favicon_16.png" />
<link rel="icon" sizes="32x32" type="image/png" href="assets/favicon_32.png" />
<link rel="icon" sizes="any" type="image/svg+xml" href="assets/favicon.
svg" />
<title>Rocket Chat</title><meta name="application-name" content="Rocket.Chat"><meta name="apple-mobile-
web-app-title" content="Rocket Chat">
<meta http-equiv="content-language" content=""><meta name="language" content="">
<meta name="robots" content="INDEX,FOLLOW">
<meta name="canonical" content="">
<meta name="google-site-verification" content="">
<meta property="fb:app_id" content="">
<base href="/>
</head>
<body>
<div id="react-root">
<div class="page-loading">
```

```
vagrant@puppet:~/tools/terraform/cluster$ kubectl port-forward rocketchat-server-76c48f65d7-ng
761 3000:80 --n application
Forwarding from 127.0.0.1:3000 -> 80
Forwarding from [::]:3000 -> 80
```

Figure 3.3: Portforwarding to the RocketChat Pod and Curl command to get the RocketChat page directly from the Pod is successful

Chapter 4

Versioning

For this project, the RNL Git (<https://git.rnl.tecnico.ulisboa.pt/AGISIT-21-22/team-06>) was used for versioning.

Chapter 5

Work Evolution

With this project was possible to show an example on how to deploy and provision tiered Microservices-based containerized Web Application on a Public Cloud using automation tools as well as implementing instrumentation on the applications, services and infrastructure components of the solution, to allow monitoring and logging features.

Initially, the system was deployed using the following tools: Terraform to create the infrastructure, Ansible to configure the infrastructure created and then Docker Compose to deploy the services needed. This version is called Docker Compose Version.

The goal for this project was to try several tools and learn as much as possible and that is the reason why the Docker Compose was chosen over Kubernetes as first hand. With the Docker Compose, some limitations were found. The first limitation is that it was needed more files and commands to run the application because the method used is more manual. With Docker Compose it is possible to configure and start multiple Docker containers on the same host so, it was needed multiple commands Docker Compose to deploy the services needed on each host (There was a host for RocketChat, another for Mongo, one more for Grafana, and another one for Prometheus - as can be seen in the Figure [5.1](#) which shows the project structure of the Docker Compose Version). The second limitation found resides in the fact that the scaling up/down mechanism of Docker Compose is also "kinda" manual because it is needed to run a command each time some scaling is needed (for example, running the docker-compose command with the flag `-scale serviceX=5` which would increase the serviceX number of containers to 5).

After analysing the limitations, the group chose to change the orchestration tool for

```

|- Vagrantfile
|- bootstrap-mgmt.sh
|- docs
\-- tools
    |-- ansible
    |   |-- ansible.cfg
    |   |-- ansible-gcp-servers-setup-all.yml
    |   |-- gcphosts
    |   \-- templates
    |       \-- haproxy.cfg.j2
    |-- scripts
    |   |-- bootstrap.sh
    |   |-- db_restore.sh
    |   |-- start_services.sh
    |   \-- stop_services.sh
    |-- terraform
    |   |-- terraform-gcp-networks.tf
    |   |-- terraform-gcp-outputs.tf
    |   |-- terraform-gcp-servers.tf
    |   |-- terraform-gcp-variables.tf
    |   |-- terraform-provider.tf
    |   \-- agisit-2021-rocketchat-06.json
    \-- docker
        |-- grafana
        |   \-- docker-compose.yml
        |-- prometheus
        |   |-- docker-compose.yml
        |   \-- config
        |       |-- alert.yml
        |       \-- prometheus.yml
        \-- rocket-chat
            \-- docker-compose.yml

```

Figure 5.1: Docker Compose Version Project Structure

[K8s](#) for a better reliable and automated solution. Although more complex, [K8s](#) gives more flexibility and the possibility of making an Autoscalable System (scaling mechanism can be automatic compared to the scaling mechanism of Docker Compose). Also, in the final solution the deployment process is now faster and more autonomous.

Bibliography

- [1] RocketChat. <https://rocket.chat/>
- [2] MongoDB. <https://en.wikipedia.org/wiki/MongoDB>
- [3] Grafana. <https://en.wikipedia.org/wiki/Grafana>
- [4] Prometheus. [https://en.wikipedia.org/wiki/Prometheus_\(software\)](https://en.wikipedia.org/wiki/Prometheus_(software))
- [5] RocketChat. High Availability. Create rocket.chat docker container. <https://docs.rocket.chat/quick-start/installing-and-updating/docker-containers/high-availability-install#create-rocket.chat-docker-container>
- [6] Running MongoDB on Kubernetes with StatefulSets. (2017, Jan 30). Document Website: <https://kubernetes.io/blog/2017/01/running-mongodb-on-kubernetes-with-statefulsets/>