Report 2 - Reliable Deployment

Gestão de Infraestruturas de Computação

2023/2024

Filipe André Seabra Gonçalves, 98083
João Pedro Saraiva Borges, 98155
Vicente Samuel Gonçalves Costa, 98515

Universidade De Aveiro

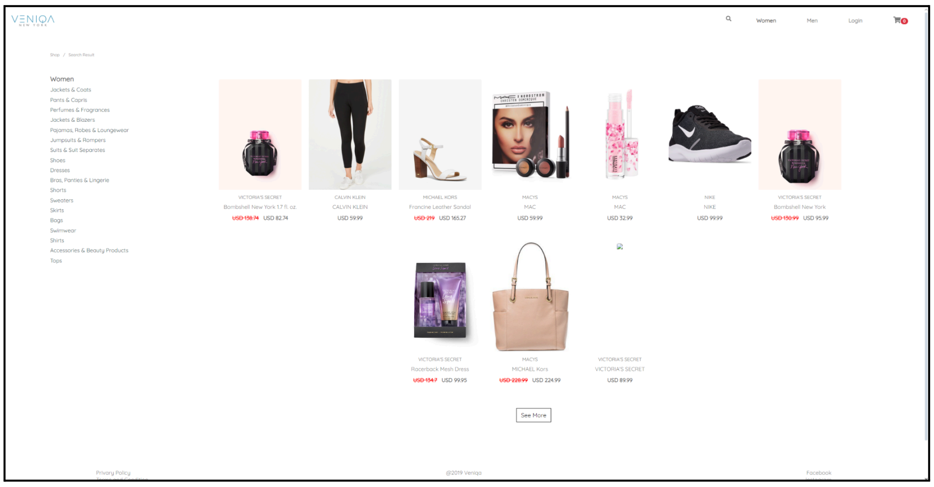
Index

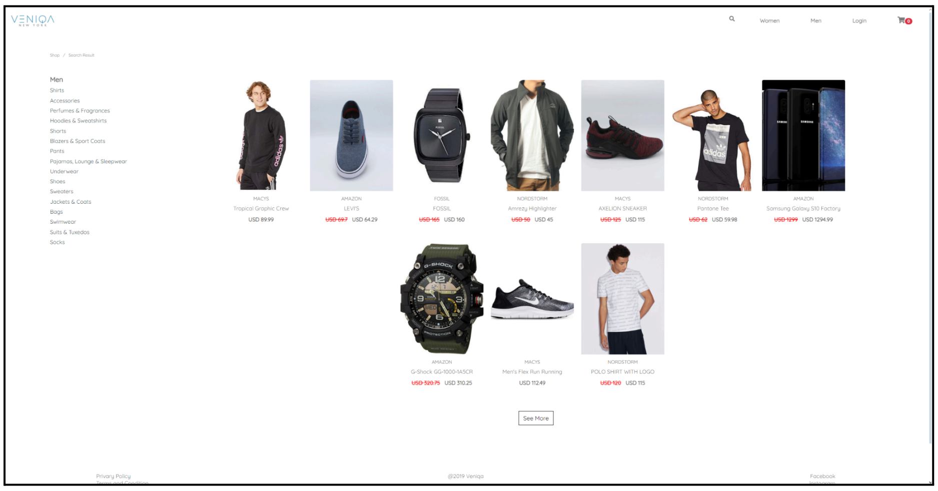
1. Contextualization

Our work, **A Senhora dos Anéis**, is a reliable e-commerce solution, based on an open source and self-hosted project called **Veniqa**, composed of a stack with frameworks such as **Redis**, **MongoDB**, **Vue.js**, **Node.js**, and many others, with scalable and flexible resources.

Our objective is to compose a cluster of deployed *kubernetes* services that can be managed, monitored and available almost all the year around, with low downtime, auto scalable and auto deployable.

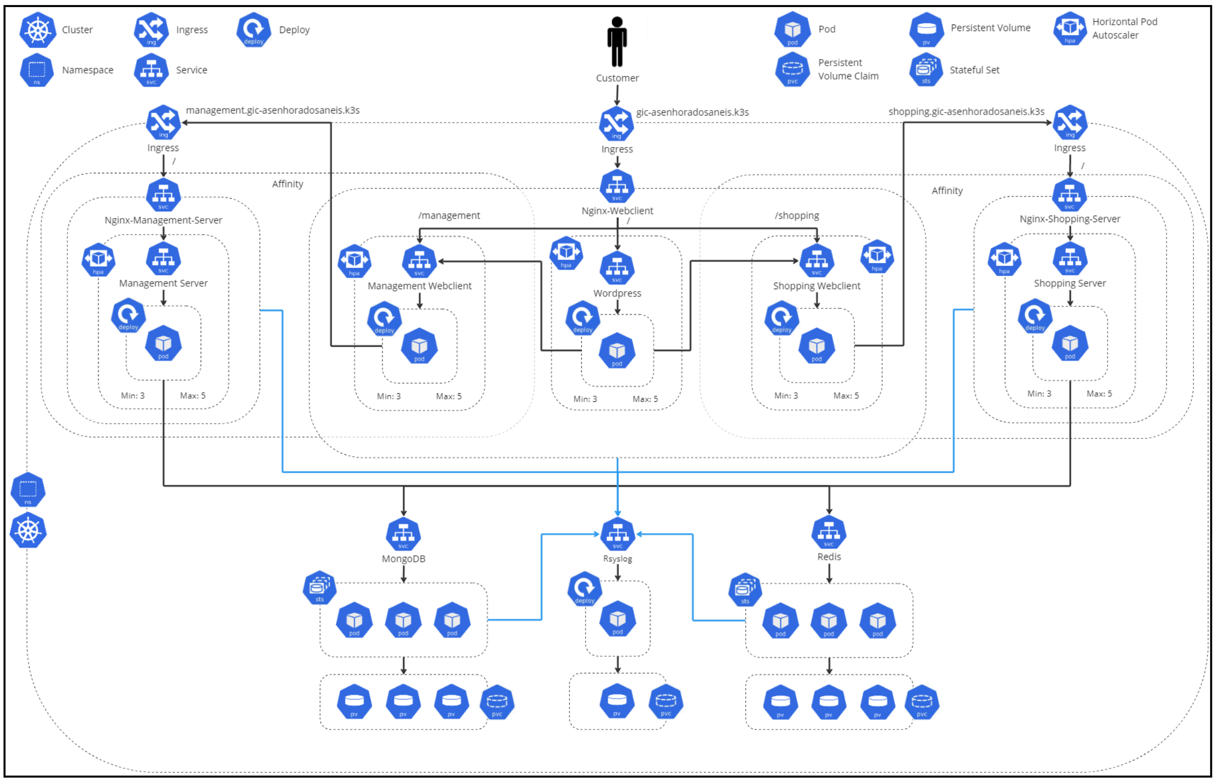
We will explain every strategy, service, deployment, volume claim and everything else in this report, and analyze the results.





2. Cluster Operation

Our *kubernetes* cluster currently takes this form:



Each service will be explained in more detail in "**3. Configurations and Strategies**", however the most important parts are the following:

- Inside our **cluster** we have our **namespace**, configured with the name **gic-asenhoradosaneis**, which englobes all the services, stateful sets and more of this deployment
- The customer will access our services by requesting to the **Ingress** controllers, which will later transfer them to the service they want to access
  - There are only five services which the customer can access via **ingress**, the management's server and webclient, shopping's server and webclient, and our wordpress website
  - Each **Ingress** only accesses a specific **Nginx** service that balances the load of the service it corresponds to
  - Each website can be accessed via a **Nginx** service, that redirects the requests from the **Ingress** controller with *http* hostname "**gic-asenhoradosaneis.k3s**", specific for all the websites
  - The management's server can be accessed via a **Nginx** service, that redirects the requests from the **Ingress** controller with *http* hostname "**management.gic-asenhoradosaneis.k3s**", specific for the management's server
  - The shopping's server can be accessed via a **Nginx** service, that redirects the requests from the **Ingress** controller with *http* hostname "**shopping.gic-asenhoradosaneis.k3s**", specific for the shopping's server

- Each server is also dependent on both databases, and every service on our logging service
- Each database and logging service are connected to at least one **persistent volume claim** which consequently contains a **persistent volume**

Replication is also a fundamental part of having our services continuously available. As such, we decided to have at least **three replicas** delivering the same service, so that even if one of the accessing points is not working, another one will be.

Ensuring an odd number of replicas minimizes the likelihood of two replicas being accessed simultaneously by the same customer. This setup ensures that there's always a *primary set* to manage request distribution in services with a *primary-replica* relationship. Additionally, the odd number of votes ensures that consensus is always reached.

2.1. Deployment Strategy

From the many strategies invented, we believe that applying **rolling deployment** would be the most optimized and safe option.

As the name suggests, this strategy slowly replaces the pods of the previous versions with the pods of the new version of the application, one at a time, in a rolling fashion, allowing the old pod versions to work while the new ones are down for updating. A rolling deployment typically waits for new pods to become ready via a readiness check before scaling down the old components.

Although errors can be propagated while implementing a new version, we can still correct them individually and not have downtime in the services, as one of the two gateways would be serving all the requests made by the user while the other would be fixed, which makes it easier to rollback the versions in the updated pods.

The **blue-green deployment** also seems fit for our case. This strategy involves running two versions of the same application at the same time, and whenever a new version passes all the necessary checks, and moving the traffic from the older version (the green version) to the newer version (the blue version). This makes it easier for the developers to deploy and test the services completely and give the safe and tested gateways to the users. Any rollbacks necessary would always be fast and reliable, without impacting the main production deployment.

However, this strategy will not be implemented as the amount of traffic this platform will have will not be big enough to compensate for the higher resources cost and human support.

2.2. Bottlenecks

There are many bottlenecks that we will eventually have to face in our project. The most troublesome is the scalability of the load balancer. In certain seasons the selling of the product might get more attention, either because of publicity done in broad social networks or because of promotions done in specific periods like black friday, summer sales, etc.. Our structure is not strongly made for huge amounts of visits everyday, as the lowest views expected are around 10 views to the site every day.

We are also currently working with applications made using **Node.js** when it was in their “early” versions, which makes deployment nowadays more complicated and difficult as some packages and functionalities may be deprecated. Our management and shopping servers demanded some refactoring as the version they should be running would be **Node.js 11**, but because of some packages we had to refactor them to **Node.js 8**. Each refactoring of the main open source code is a risk to any functionality that might be lost or corrupted.

3. Configurations and Strategies

The deployment configurations were mainly configured in the *deployment* folder, which can be accessed here.

Each service, or group of similar services, belongs to a singular deployment file, as it is easier to use the **rolling deployment** strategy when updating a service. It also makes it easier to organize and understand the deployment files and the deployment process.

We also developed a script for building all docker images, pushing them to the class registry and deploying them to our *kubernetes* namespace. It also comes equipped with an option to delete all the docker volumes, which can be useful when trying to have a fresh build - accessed by using the flag “-d 1”.

We will walk through all the services, deployments, stateful sets, persistent volumes and secrets, and explain the motives behind their creation and configuration.

3.1. Redis

Redis is used as a cache database for the management’s and shopping’s servers. As **Redis** is a stateful application, the service was also established with the **Stateful Set** property, which means that the instance requires stable, unique identifiers (maintaining the network identity as “hostname” in our configuration) and persistent storage, crucial for data durability and consistency.

It’s being deployed in a *primary-secondary* relationship, with a service for each side. The *primary* replica will be in charge of all the writes, while the *secondary* replicas will only be allowed to read - the replication of the data will be made by checking the logs made by the *primary* replicas.

In total there are always **three** replicas, one *primary* and two *secondary*, as it ensures an odd number of votes for a consensus. Each replica also mounts one volume, a **Redis** persistent data storage, which saves all the cached data, and one secret for the redis *primary-secondary* configuration. The relationship configuration is also being done by executing a bash file - *entrypoint.sh* - which makes sure that the replication mode is *master* for the *primary* replica, and *replica* for the *secondary* replicas, as well as identifying which is the *master* replica and waiting it to be available to finish the configuration.

If the **Redis** ends up failing or having an error, it has a restart policy to always restart when it happens. There is also an established templated container for easier replication, isolation and portability.

The persistent volume claims (**PVC**) should have an access mode of read/write once, which permits only one deployment at a time to access it and do the operations; it is also specified that the storage class is *longhorn*. This **PVCs** makes the data persistent even if the **Redis** service fails or is turned off.

To centralize all the logs, it was necessary to create a **Rsyslog** client in every **Redis** pod, so we can use the *RELP output module* to forward the logs from the pods to the centralized **Rsyslog** service.

3.2. MongoDB

MongoDB has the purpose of saving persistent data all over the system, like shopping items, users carts, login sessions, etc..

As **MongoDB** is a stateful application, the service was also established with the **Stateful Set** property, and, just as **Redis**, will have **three** replicas and a template container for isolation, consistency and repeatability in pod creation. The changes made to the template will reflect in all **MongoDB** pods because of the **Stateful Set**, ensuring uniformity and simplifying management.

It also implements a *master-slave* relationship where the *master* replica is the only replica allowed to write, and the other two replicas are considered *slaves* and only allowed to read the data based on the logs between the replicas. The *master* is elected with the help of a **Job**, *init-mongodb*, that waits for all the three replicas to be available and initiates a **MongoDB** replication vote.

There is also a **PVC** connected to each replica to make data persistent with an access mode of read/write once, in which the *secondary* pods can only read and the *primary* pod can also write.

Similarly to **Redis**, we also had to create a **Rsyslog** client in every **MongoDB** pod to forward the logs from the pods to the centralized **Rsyslog** service.

3.3. Traefik and Middleware

The **Traefik** is our **Ingress** controller, which redirects every request to a *reverse proxy* in a specific **Nginx** service. Based on the path of a request and their *hostname*, a different service will be redirected to the user.

We used different annotations, which provide additional configuration to the controller, which we list them below:

- Allowing both *http* and *https* request to the **Ingress**
- Always redirecting *http* traffic to *https*
- The redirection is always permanent
- Preserve the original hostname in the redirected request headers

We decided to have three **Ingress** controllers, each one redirects the traffic to the following three **Nginx** services:

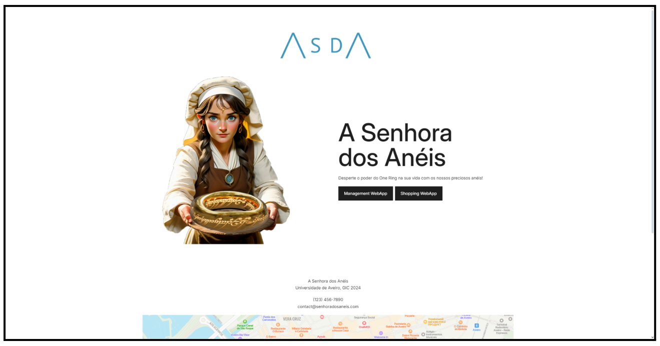
- Nginx-webclient
    - Wordpress: Path "/"
    - Shopping Webclient: Path "/shopping"
    - Management Webclient: Path "/management"
- Nginx-management-server
    - Management Server: Path "/"
- Nginx-shopping-server
    - Shopping Server: Path "/"

We decided to divide the servers into two different **Ingress** controllers, and consequently two **Nginx** services as there will be some problems in the servers *api* in each service.

Each **Nginx** service also contains a configuration for sending every service's access and error logs to our *syslog* service.

3.4. Wordpress

Our static website was created using **Wordpress**, which is a web content management system, where a user can create a new website, edit and style it and deploy it in a wordpress domain. This service was deployed by simply creating a **cherrypy** python application that returns the *html* file of the website.



A Senhora dos Anéis

Desperte o poder do One Ring na sua vida com os nossos preciosos anéis!

A Senhora dos Anéis
Universidade de Aveiro, GIC 2024

(123) 456-7890
contact@senhoradosaneis.com

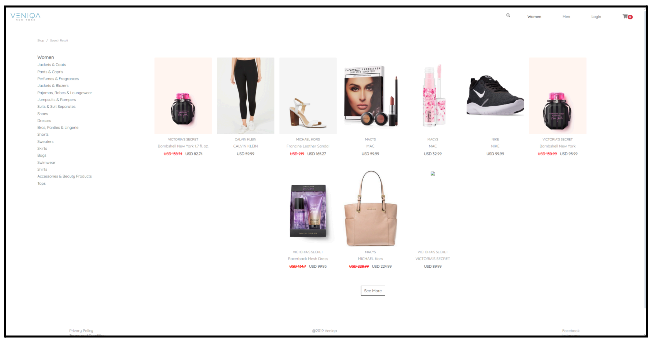
3.5. Shopping WebClient

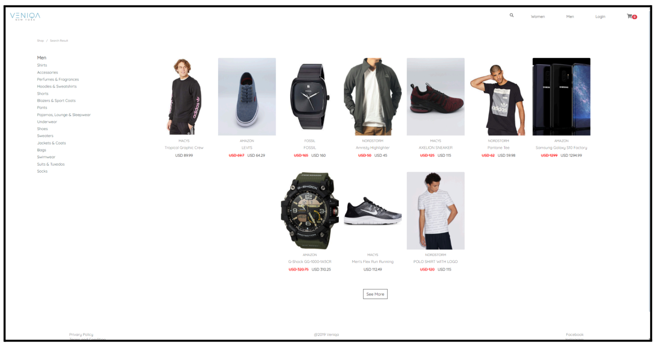
This will be the main interface for the user, it will contain the **Veniqa** store webpage as a template which was done with **Node.js** and **Vue.js**. The service's name is **shopping-webclient** and has an image pull policy that dictates that the latest version in the registry is always used above the others.

The service will listen on port 80, which is connected to the redirect from the **Ingress** controller, and will forward the traffic to port 8080, the internal port where the service is running inside the pod.

The Dockerfile uses a base image from Node.js version 8 and includes installation and setup for the HTTP server package. It defines a working directory, specifies the dependencies file, and sets up the volumes folder for data persistence. The Dockerfile also builds and runs the application, copies assets individually into the /var/static folder, and finally starts the HTTP server.

This service also has an affinity rule with the **shopping-server** service, ensuring that a **shopping-webclient** pod is located wherever a **shopping-server** pod is. This minimizes the request time between pods.





3.6. Management WebClient

This will be the main interface for the manager, which can check how the operations are going and change some elements on the shopping interface, it is also part of the **Veniqa** template and was done with **Node.js** and **Vue.js**. It is highly similar to the shopping webclient, the only things that differs from them are the identification, labeling this service with the name **management-webclient**.

This service also has an affinity rule with the **management-server** service.

3.7. Shopping Server

This will be the server in which the **shopping-webclient** will make their requests for security, data, etc.

This service is named **shopping-server**, it will have **three** replicas, and will listen to port 80, which is connected to the redirect from the **Ingress** controller, and will forward the traffic to port 3000, the internal port where the service is running inside the pod.

Within its configurations, this service will mount a secret to provide sensitive configuration data securely, containing information regarding database connection, database port and host, replication mode, service mode (deployment or local), etc., which can be accessed by the secret **shopping-server-secret**.

The dockerfile contains a base image derived from version 8 of **Node.js**, an installation of dependencies, containing libraries for handling *http* requests (*axios*), database connections (**mongoose** and **Redis**), session management (*express-session*), authentication (*bcrypt-nodejs* for securely hashing passwords before storing), email sending (*@sendgrid/mail*) and logging (*winston*).

This service also has an anti affinity to the **shopping-server** service, which means that two **shopping-server** pods will never stay together.

We solved some existent problems with this service:

- The server was in a loop trying to connect to our **Redis** service, which made every request to the *api* returning the status code 502 *Bad Gateway*
- The **CORS** politics were badly implemented for cloud deployment

3.8. Management Server

This will be the server in which the **management-webclient** will make their requests for security, data, etc.

Just as before with the webclients, the servers are also really similar, so it has fundamentally the same configuration, it only changes in questions of name, accessible ports, and secret name and values.

This service also has an anti affinity to the **management-server** service.

3.9. Rsyslog

Rsyslog makes logs out of every operation made inside of the *kubernetes* cluster.

Its service enables other components within the cluster to send logs to the syslog server. This service listens on port 514, for both **UDP** and **TCP**, and redirects traffic to port 514 of the **Rsyslog** pod.

We used this service to centralize all the logs provenient from our services and divided the logs into different files, all in the folder */var/log*:

- *gic-asenhoradosaneis-syslog.log*: logs from all the services
- *gic-asenhoradosaneis-mongodb.log*: logs from the **MongoDB** service
- *gic-asenhoradosaneis-redis.log*: logs from the **Redis** service
- *gic-asenhoradosaneis-webclients.log*: logs from the **Nginx-webclient** service, that logs every request made to the three websites
- *gic-asenhoradosaneis-shopping-server.log*: logs from the **Nginx-shopping-server** service, that logs every request made to the shopping server
- *gic-asenhoradosaneis-management-server.log*: logs from the **Nginx-management-server** service, that logs every request made to the management server

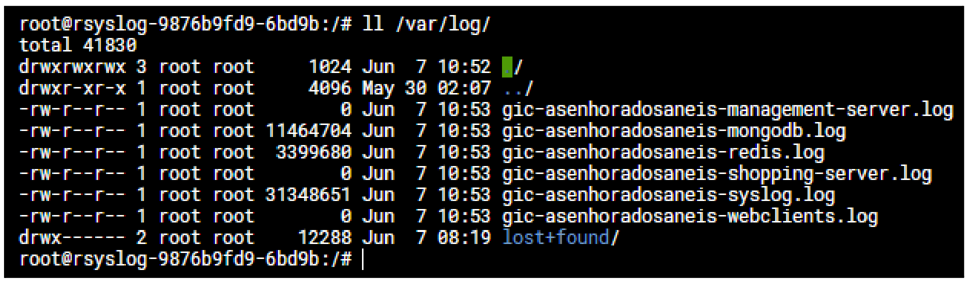
This logs are being redirected from the *syslog* server to different facilities:

- Local2 for the **Nginx-webclient** logs
- Local3 for the **Nginx-shopping-server** logs
- Local4 for the **Nginx-management-server** logs

The database logs are coming from the **Rsyslog** *RELP input module*, which are redirected to the correct file based on the hostname of the sender.

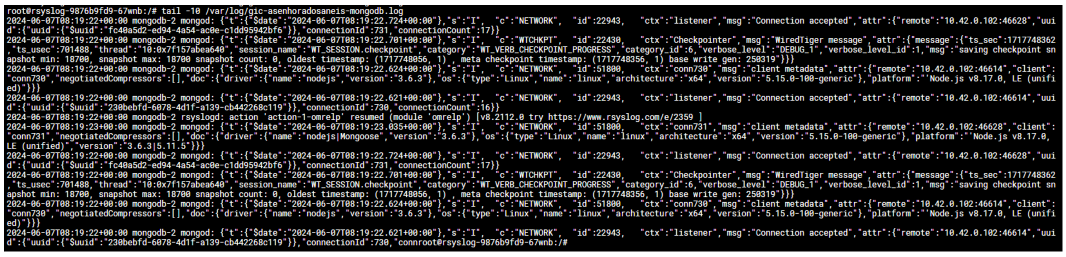
We also added a **Job,** named **rsyslog-ipchange**, which goes through every pod and updates the *IP* configuration for the *syslog* server. This is very important, as if there is any problem with the starting container of the **Rsyslog** deployment, or if there was a need to update the service, when it is initiated it will have a different *IP* address to the previously used in the other services. This **Job** prevents any possible problems.

This service also mounts a **PVC** so that all the data can be stored in a single place, and not have problems with volume size.



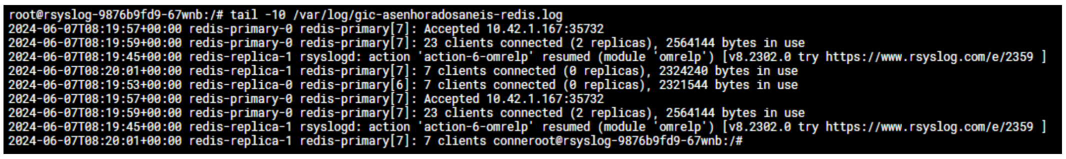
```
root@rsyslog-9876b9fd9-6bd9b:/# ll /var/log/
total 41830
drwxrwxrwx 3 root root     1024 Jun  7 10:52 ./
drwxr-xr-x 1 root root     4096 May 30 02:07 ../
-rw-r--r-- 1 root root        0 Jun  7 10:53 gic-asenhoradosaneis-management-server.log
-rw-r--r-- 1 root root 11464704 Jun  7 10:53 gic-asenhoradosaneis-mongodb.log
-rw-r--r-- 1 root root  3399680 Jun  7 10:53 gic-asenhoradosaneis-redis.log
-rw-r--r-- 1 root root        0 Jun  7 10:53 gic-asenhoradosaneis-shopping-server.log
-rw-r--r-- 1 root root 31348651 Jun  7 10:53 gic-asenhoradosaneis-syslog.log
-rw-r--r-- 1 root root        0 Jun  7 10:53 gic-asenhoradosaneis-webclients.log
drwx------ 2 root root    12288 Jun  7 08:19 lost+found/
root@rsyslog-9876b9fd9-6bd9b:/# |
```

MongoDB logs:



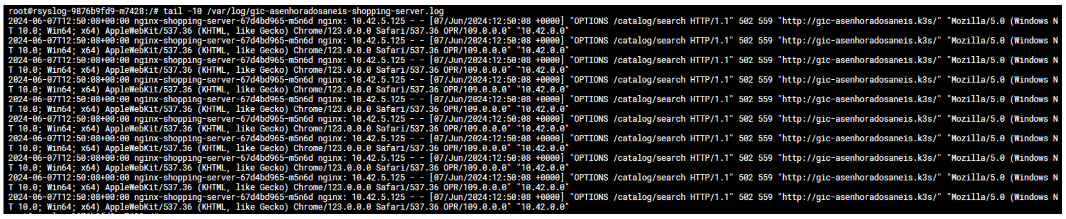
```
root@rsyslog-9876b9fd9-67wnb:/# tail -10 /var/log/gic-asenhoradosaneis-mongodb.log
2024-06-07T08:19:22+00:00 mongodb-2 mongod: {"t":{"$date":"2024-06-07T08:19:22.724+00:00"},"s":"I",  "c":"NETWORK",  "id":22943,   "ctx":"listener","msg":"Connection accepted","attr":{"remote":"10.42.0.102:46628","uui
d":{"uuid":{"$uuid":"fc40a5d2-ed94-4a54-ac0e-c1dd95942bf6"}},"connectionId":731,"connectionCount":17}}
2024-06-07T08:19:22+00:00 mongodb-2 mongod: {"t":{"$date":"2024-06-07T08:19:22.701+00:00"},"s":"I",  "c":"WTCHKPT",  "id":22430,   "ctx":"Checkpointer","msg":"WiredTiger message","attr":{"message":{"ts_sec":1717748362
,"ts_usec":701488,"thread":"10:0x7f157abea640","session_name":"WT_SESSION.checkpoint","category":"WT_VERB_CHECKPOINT_PROGRESS","category_id":6,"verbose_level":"DEBUG_1","verbose_level_id":1,"msg":"saving checkpoint sn
apshot min: 18700, snapshot max: 18700 snapshot count: 0, oldest timestamp: (1717748056, 1) , meta checkpoint timestamp: (1717748356, 1) base write gen: 250319"}}}
2024-06-07T08:19:22+00:00 mongodb-2 mongod: {"t":{"$date":"2024-06-07T08:19:22.624+00:00"},"s":"I",  "c":"NETWORK",  "id":51800,   "ctx":"conn730","msg":"client metadata","attr":{"remote":"10.42.0.102:46614","client":
"conn730","negotiatedCompressors":[],"doc":{"driver":{"name":"nodejs","version":"3.6.3"},"os":{"type":"Linux","name":"linux","architecture":"x64","version":"5.15.0-100-generic"},"platform":"'Node.js v8.17.0, LE (unifi
ed)"}}}
2024-06-07T08:19:22+00:00 mongodb-2 mongod: {"t":{"$date":"2024-06-07T08:19:22.621+00:00"},"s":"I",  "c":"NETWORK",  "id":22943,   "ctx":"listener","msg":"Connection accepted","attr":{"remote":"10.42.0.102:46614","uui
d":{"uuid":{"$uuid":"230bebfd-6078-4d1f-a139-cb442268c119"}},"connectionId":730,"connectionCount":16}}
2024-06-07T08:19:22+00:00 mongodb-2 rsyslogd: action 'action-1-omrelp' resumed (module 'omrelp') [v8.2112.0 try https://www.rsyslog.com/e/2359 ]
2024-06-07T08:19:23+00:00 mongodb-2 mongod: {"t":{"$date":"2024-06-07T08:19:23.035+00:00"},"s":"I",  "c":"NETWORK",  "id":51800,   "ctx":"conn731","msg":"client metadata","attr":{"remote":"10.42.0.102:46628","client":
"conn731","negotiatedCompressors":[],"doc":{"driver":{"name":"nodejs|Mongoose","version":"3.6.3"},"os":{"type":"Linux","name":"linux","architecture":"x64","version":"5.15.0-100-generic"},"platform":"'Node.js v8.17.0,
LE (unified)","version":"3.6.3|5.11.5"}}}
2024-06-07T08:19:22+00:00 mongodb-2 mongod: {"t":{"$date":"2024-06-07T08:19:22.724+00:00"},"s":"I",  "c":"NETWORK",  "id":22943,   "ctx":"listener","msg":"Connection accepted","attr":{"remote":"10.42.0.102:46628","uui
d":{"uuid":{"$uuid":"fc40a5d2-ed94-4a54-ac0e-c1dd95942bf6"}},"connectionId":731,"connectionCount":17}}
2024-06-07T08:19:22+00:00 mongodb-2 mongod: {"t":{"$date":"2024-06-07T08:19:22.701+00:00"},"s":"I",  "c":"WTCHKPT",  "id":22430,   "ctx":"Checkpointer","msg":"WiredTiger message","attr":{"message":{"ts_sec":1717748362
,"ts_usec":701488,"thread":"10:0x7f157abea640","session_name":"WT_SESSION.checkpoint","category":"WT_VERB_CHECKPOINT_PROGRESS","category_id":6,"verbose_level":"DEBUG_1","verbose_level_id":1,"msg":"saving checkpoint sn
apshot min: 18700, snapshot max: 18700 snapshot count: 0, oldest timestamp: (1717748056, 1) , meta checkpoint timestamp: (1717748356, 1) base write gen: 250319"}}}
2024-06-07T08:19:22+00:00 mongodb-2 mongod: {"t":{"$date":"2024-06-07T08:19:22.624+00:00"},"s":"I",  "c":"NETWORK",  "id":51800,   "ctx":"conn730","msg":"client metadata","attr":{"remote":"10.42.0.102:46614","client":
"conn730","negotiatedCompressors":[],"doc":{"driver":{"name":"nodejs","version":"3.6.3"},"os":{"type":"Linux","name":"linux","architecture":"x64","version":"5.15.0-100-generic"},"platform":"'Node.js v8.17.0, LE (unifi
ed)"}}}
2024-06-07T08:19:22+00:00 mongodb-2 mongod: {"t":{"$date":"2024-06-07T08:19:22.621+00:00"},"s":"I",  "c":"NETWORK",  "id":22943,   "ctx":"listener","msg":"Connection accepted","attr":{"remote":"10.42.0.102:46614","uui
d":{"uuid":{"$uuid":"230bebfd-6078-4d1f-a139-cb442268c119"}},"connectionId":730,"connroot@rsyslog-9876b9fd9-67wnb:/#
```

Redis logs:



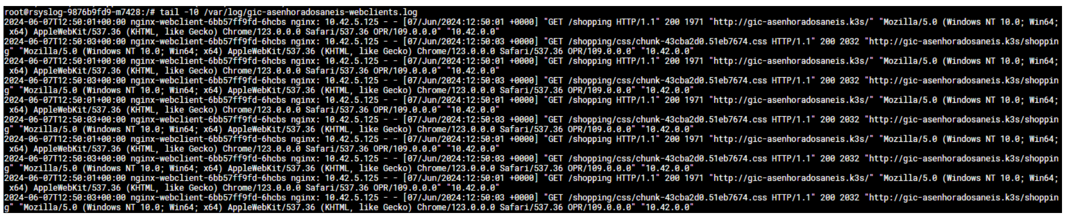
```
root@rsyslog-9876b9fd9-67wnb:/# tail -10 /var/log/gic-asenhoradosaneis-redis.log
2024-06-07T08:19:57+00:00 redis-primary-0 redis-primary[7]: Accepted 10.42.1.167:35732
2024-06-07T08:19:59+00:00 redis-primary-0 redis-primary[7]: 23 clients connected (2 replicas), 2564144 bytes in use
2024-06-07T08:19:45+00:00 redis-replica-1 rsyslogd: action 'action-6-omrelp' resumed (module 'omrelp') [v8.2302.0 try https://www.rsyslog.com/e/2359 ]
2024-06-07T08:20:01+00:00 redis-replica-1 redis-primary[7]: 7 clients connected (0 replicas), 2324240 bytes in use
2024-06-07T08:19:53+00:00 redis-replica-0 redis-primary[6]: 7 clients connected (0 replicas), 2321544 bytes in use
2024-06-07T08:19:57+00:00 redis-primary-0 redis-primary[7]: Accepted 10.42.1.167:35732
2024-06-07T08:19:59+00:00 redis-primary-0 redis-primary[7]: 23 clients connected (2 replicas), 2564144 bytes in use
2024-06-07T08:19:45+00:00 redis-replica-1 rsyslogd: action 'action-6-omrelp' resumed (module 'omrelp') [v8.2302.0 try https://www.rsyslog.com/e/2359 ]
2024-06-07T08:20:01+00:00 redis-replica-1 redis-primary[7]: 7 clients conneroot@rsyslog-9876b9fd9-67wnb:/#
```

Shopping-server logs:



```
root@rsyslog-9876b9fd9-m7428:/# tail -10 /var/log/gic-asenhoradosaneis-shopping-server.log
2024-06-07T12:50:08+00:00 nginx-shopping-server-67d4bd965-m5n6d nginx: 10.42.5.125 - - [07/Jun/2024:12:50:08 +0000] "OPTIONS /catalog/search HTTP/1.1" 502 559 "http://gic-asenhoradosaneis.k3s/" "Mozilla/5.0 (Windows N
T 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.0.0 Safari/537.36 OPR/109.0.0.0" "10.42.0.0"
2024-06-07T12:50:08+00:00 nginx-shopping-server-67d4bd965-m5n6d nginx: 10.42.5.125 - - [07/Jun/2024:12:50:08 +0000] "OPTIONS /catalog/search HTTP/1.1" 502 559 "http://gic-asenhoradosaneis.k3s/" "Mozilla/5.0 (Windows N
T 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.0.0 Safari/537.36 OPR/109.0.0.0" "10.42.0.0"
2024-06-07T12:50:08+00:00 nginx-shopping-server-67d4bd965-m5n6d nginx: 10.42.5.125 - - [07/Jun/2024:12:50:08 +0000] "OPTIONS /catalog/search HTTP/1.1" 502 559 "http://gic-asenhoradosaneis.k3s/" "Mozilla/5.0 (Windows N
T 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.0.0 Safari/537.36 OPR/109.0.0.0" "10.42.0.0"
2024-06-07T12:50:08+00:00 nginx-shopping-server-67d4bd965-m5n6d nginx: 10.42.5.125 - - [07/Jun/2024:12:50:08 +0000] "OPTIONS /catalog/search HTTP/1.1" 502 559 "http://gic-asenhoradosaneis.k3s/" "Mozilla/5.0 (Windows N
T 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.0.0 Safari/537.36 OPR/109.0.0.0" "10.42.0.0"
2024-06-07T12:50:08+00:00 nginx-shopping-server-67d4bd965-m5n6d nginx: 10.42.5.125 - - [07/Jun/2024:12:50:08 +0000] "OPTIONS /catalog/search HTTP/1.1" 502 559 "http://gic-asenhoradosaneis.k3s/" "Mozilla/5.0 (Windows N
T 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.0.0 Safari/537.36 OPR/109.0.0.0" "10.42.0.0"
2024-06-07T12:50:08+00:00 nginx-shopping-server-67d4bd965-m5n6d nginx: 10.42.5.125 - - [07/Jun/2024:12:50:08 +0000] "OPTIONS /catalog/search HTTP/1.1" 502 559 "http://gic-asenhoradosaneis.k3s/" "Mozilla/5.0 (Windows N
T 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.0.0 Safari/537.36 OPR/109.0.0.0" "10.42.0.0"
2024-06-07T12:50:08+00:00 nginx-shopping-server-67d4bd965-m5n6d nginx: 10.42.5.125 - - [07/Jun/2024:12:50:08 +0000] "OPTIONS /catalog/search HTTP/1.1" 502 559 "http://gic-asenhoradosaneis.k3s/" "Mozilla/5.0 (Windows N
T 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.0.0 Safari/537.36 OPR/109.0.0.0" "10.42.0.0"
2024-06-07T12:50:08+00:00 nginx-shopping-server-67d4bd965-m5n6d nginx: 10.42.5.125 - - [07/Jun/2024:12:50:08 +0000] "OPTIONS /catalog/search HTTP/1.1" 502 559 "http://gic-asenhoradosaneis.k3s/" "Mozilla/5.0 (Windows N
T 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.0.0 Safari/537.36 OPR/109.0.0.0" "10.42.0.0"
2024-06-07T12:50:08+00:00 nginx-shopping-server-67d4bd965-m5n6d nginx: 10.42.5.125 - - [07/Jun/2024:12:50:08 +0000] "OPTIONS /catalog/search HTTP/1.1" 502 559 "http://gic-asenhoradosaneis.k3s/" "Mozilla/5.0 (Windows N
T 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.0.0 Safari/537.36 OPR/109.0.0.0" "10.42.0.0"
2024-06-07T12:50:08+00:00 nginx-shopping-server-67d4bd965-m5n6d nginx: 10.42.5.125 - - [07/Jun/2024:12:50:08 +0000] "OPTIONS /catalog/search HTTP/1.1" 502 559 "http://gic-asenhoradosaneis.k3s/" "Mozilla/5.0 (Windows N
T 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.0.0 Safari/537.36 OPR/109.0.0.0" "10.42.0.0"
```

Webclients logs:



```
root@rsyslog-9876b9fd9-m7428:/# tail -10 /var/log/gic-asenhoradosaneis-webclients.log
2024-06-07T12:50:01+00:00 nginx-webclient-6bb57ff9fd-6hcbs nginx: 10.42.5.125 - - [07/Jun/2024:12:50:01 +0000] "GET /shopping HTTP/1.1" 200 1971 "http://gic-asenhoradosaneis.k3s/" "Mozilla/5.0 (Windows NT 10.0; Win64;
 x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.0.0 Safari/537.36 OPR/109.0.0.0" "10.42.0.0"
2024-06-07T12:50:03+00:00 nginx-webclient-6bb57ff9fd-6hcbs nginx: 10.42.5.125 - - [07/Jun/2024:12:50:03 +0000] "GET /shopping/css/chunk-43cba2d0.51eb7674.css HTTP/1.1" 200 2032 "http://gic-asenhoradosaneis.k3s/shoppin
g" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.0.0 Safari/537.36 OPR/109.0.0.0" "10.42.0.0"
2024-06-07T12:50:01+00:00 nginx-webclient-6bb57ff9fd-6hcbs nginx: 10.42.5.125 - - [07/Jun/2024:12:50:01 +0000] "GET /shopping HTTP/1.1" 200 1971 "http://gic-asenhoradosaneis.k3s/" "Mozilla/5.0 (Windows NT 10.0; Win64;
 x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.0.0 Safari/537.36 OPR/109.0.0.0" "10.42.0.0"
2024-06-07T12:50:03+00:00 nginx-webclient-6bb57ff9fd-6hcbs nginx: 10.42.5.125 - - [07/Jun/2024:12:50:03 +0000] "GET /shopping/css/chunk-43cba2d0.51eb7674.css HTTP/1.1" 200 2032 "http://gic-asenhoradosaneis.k3s/shoppin
g" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.0.0 Safari/537.36 OPR/109.0.0.0" "10.42.0.0"
2024-06-07T12:50:01+00:00 nginx-webclient-6bb57ff9fd-6hcbs nginx: 10.42.5.125 - - [07/Jun/2024:12:50:01 +0000] "GET /shopping HTTP/1.1" 200 1971 "http://gic-asenhoradosaneis.k3s/" "Mozilla/5.0 (Windows NT 10.0; Win64;
 x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.0.0 Safari/537.36 OPR/109.0.0.0" "10.42.0.0"
2024-06-07T12:50:03+00:00 nginx-webclient-6bb57ff9fd-6hcbs nginx: 10.42.5.125 - - [07/Jun/2024:12:50:03 +0000] "GET /shopping/css/chunk-43cba2d0.51eb7674.css HTTP/1.1" 200 2032 "http://gic-asenhoradosaneis.k3s/shoppin
g" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.0.0 Safari/537.36 OPR/109.0.0.0" "10.42.0.0"
2024-06-07T12:50:01+00:00 nginx-webclient-6bb57ff9fd-6hcbs nginx: 10.42.5.125 - - [07/Jun/2024:12:50:01 +0000] "GET /shopping HTTP/1.1" 200 1971 "http://gic-asenhoradosaneis.k3s/" "Mozilla/5.0 (Windows NT 10.0; Win64;
 x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.0.0 Safari/537.36 OPR/109.0.0.0" "10.42.0.0"
2024-06-07T12:50:03+00:00 nginx-webclient-6bb57ff9fd-6hcbs nginx: 10.42.5.125 - - [07/Jun/2024:12:50:03 +0000] "GET /shopping/css/chunk-43cba2d0.51eb7674.css HTTP/1.1" 200 2032 "http://gic-asenhoradosaneis.k3s/shoppin
g" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.0.0 Safari/537.36 OPR/109.0.0.0" "10.42.0.0"
2024-06-07T12:50:01+00:00 nginx-webclient-6bb57ff9fd-6hcbs nginx: 10.42.5.125 - - [07/Jun/2024:12:50:01 +0000] "GET /shopping HTTP/1.1" 200 1971 "http://gic-asenhoradosaneis.k3s/" "Mozilla/5.0 (Windows NT 10.0; Win64;
 x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.0.0 Safari/537.36 OPR/109.0.0.0" "10.42.0.0"
2024-06-07T12:50:03+00:00 nginx-webclient-6bb57ff9fd-6hcbs nginx: 10.42.5.125 - - [07/Jun/2024:12:50:03 +0000] "GET /shopping/css/chunk-43cba2d0.51eb7674.css HTTP/1.1" 200 2032 "http://gic-asenhoradosaneis.k3s/shoppin
g" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.0.0 Safari/537.36 OPR/109.0.0.0" "10.42.0.0"
```

3.10. Secrets

Inside the */deployment/secrets* folder, there is a script file created by us to establish all the secrets for the services. This script utilizes *kubectl* commands to create *kubernetes* secrets.

We define all the secrets for the **shopping-server** and **management-server**, including the database connections strings, environment variables and authentication credentials, and also define some secrets needed to run the **Redis** service.

3.11. Nginx

As said before, we use **Nginx** as a *load balancer* to be able to redirect the requests from the **Ingress** controller to the correct services. We created three **Nginx** services for each of the three **Ingress** controllers, each one redirecting to a specific service and with a different configuration file. The configuration file changes only in the *http server*, and consequently the paths and services it redirects to.

4. Evaluation of the Cluster Operations

In terms of applications functionality, most of the services inside of this cluster are correctly working with no problems whatsoever, be it in accessing web pages, or connecting to the databases.

However, we couldn't successfully deploy the management server due to an unexpected situation in the initiation of the server. The project installs all the necessary dependencies and correctly builds itself, however the server doesn't start. Consequently the pods in which the management server are located are constantly restarting as its "duty" was achieved. This prevents the management webclient from connecting to the server, returning a *502 Bad Gateway* error message.

Our cluster has a reasonable resource utilization, with average *cpu* cores usage around 0.02 in most of the services in rest, with some services consuming more resources than others, such as the management and shopping servers. In terms of memory, the webclients only occupy around 20 to 30 Mib, whereas the servers occupy almost 800 Mib each replica.

We also tried to scale the performance by changing the workload, which resulted in a more than average consumption, but still similar to the values given before.

Even if a service fails or is killed, the cluster will not have downtime and will keep on running, even if slower.

Overall, we believe that in theory, the implementation we made is optimal, reliable and secure so that it becomes difficult to be able to have downtime in our cluster. However, because we couldn't test all the functionalities, mostly the management server requests, we can't be sure that it will be as said.

Finally, to use our hostname in our services, we had to add the three hostnames from our **Ingress** controllers in the */etc/hosts* file in our system:

```
193.136.82.36      gic-asenhoradosaneis.k3s
193.136.82.36      management.gic-asenhoradosaneis.k3s
193.136.82.36      shopping.gic-asenhoradosaneis.k3s
```

5. Redundancy and Scalability

As explained, every service, either a deployment or a stateful set, is replicated across at least three replicas. However, replicability does not necessarily mean that the services also have redundancy and reliability. Another problem is that whenever a service is highly requested, having only three replicas might not be enough. In contrast, if a service is being requested a lot we should be able to have more replicas running, or if the service is not being requested so much, there might not be a need to have more than one replica.

The answer to our problems was to create **Horizontal Pod Scalers (HPA)**, which balances the number of replicas for one service based on the replicas metrics. In most of our cases, we simply created a new **HPA** to scale the service between 3 and 5 replicas. The metrics used were the *cpu* cores usage and memory occupied.

In each service we had to add the limits of resources, both minimal, to alert the cluster that a replica of this service needs at least that much resources to be created, and maximal, to alert the service that a new replica needs to be created. The **HPA**, will try to create a new replica when the *cpu* cores usage is above 90% of the maximal limit of resources.

The following are the limits of resources for the services:

Services	Minimal Limit	Maximum Limit
Management Server	CPU: 0.05 ; Memory: 800 Mi	CPU: 0.5 ; Memory: 1500 Mi
Shopping Server	CPU: 0.05 ; Memory: 800 Mi	CPU: 0.5 ; Memory: 1500 Mi
Management Webclient	CPU: 0.05 ; Memory: 200 Mi	CPU: 0.5 ; Memory: 800 Mi
Shopping Webclient	CPU: 0.05 ; Memory: 200 Mi	CPU: 0.5 ; Memory: 800 Mi
Wordpress	CPU: 0.05 ; Memory: 200 Mi	CPU: 0.4 ; Memory: 800 Mi

We derived these values after a simple stress test in all of the services, where we analyzed the *cpu* and memory usage and decided on the best values.

As our two database services, **MongoDB** and **Redis**, are Stateful Sets, the redundancy is automatically implemented, without the need to use any **HPA**.

6. Future Work

As for the next steps in our deployment, we should aim at implementing health checks and observability in our cluster.

Regarding the health checks, it's important to understand the health of our pods and services, as well as monitoring them to confirm if they are operating as intended, and if not, destroy the pod or service and replicate another to replace it. An example of possible health checks mechanisms to deploy in our cluster would be the **Rancher** health checks, which provide *TCP* and *http* health checks using its own microservices. We can also deploy a *kubernetes* health check service that can take into account the probe liveliness, readiness or startup.

For the observability, we could use some service like **Grafana** that creates a simple and interactive *UI* with the observable variables we configure. However, and even though observability is an important part of any *kubernetes* cluster, having a service like **Grafana** doesn't mean much if there are no values to be observed.

Finally, we would like to correct our management server's errors and deploy the whole of the **Veniqa** project in its entirety in our *kubernetes* cluster.

7. References

Top 5 Most-Used Deployment Strategies
Advanced Deployment Strategies - Deployments | Developer Guide | OpenShift Dedicated 3
Deployment Strategies - Deployments | Developer Guide | OpenShift Dedicated 3
https://ranchermanager.docs.rancher.com/v2.0-v2.4/how-to-guides/new-user-guides/migrate-from-v1.6-v2.x/monitor-apps
https://komodor.com/blog/kubernetes-health-checks-everything-you-need-to-know/