

FIREPASS

# FIREPASS TECHNICAL MANUAL

FTM

FIREPASS COMPANY  
26-9-2022

# INDEX

<b>1. INTRODUCTION .....</b>	<b>3</b>
<b>2. SERVICES .....</b>	<b>3</b>
2.1. DINAHOSTING .....	3
2.2. LOAD BALANCER .....	3
2.3. WEB SERVER .....	3
2.4. REST – API .....	3
2.5. DATA BASE .....	4
2.6. MONITORING .....	4
2.7. AUTOMATION SERVICE .....	4
2.8. FIREPASS SERVER .....	4
<b>3. SOFTWARE TOOLS .....</b>	<b>5</b>
3.1. REST-API .....	5
3.1.1. <i>Java-spring boot</i> .....	5
3.1.2. <i>Configuration</i> .....	5
3.1.3. <i>Development</i> .....	6
3.1.4. <i>Testing the solution</i> .....	9
3.2. MYSQL .....	11
3.2.1. <i>Installation and configuration</i> .....	11
3.2.2. <i>Fill the data base</i> .....	12
3.3. NGINX .....	13
3.3.1. <i>Installation and configuration</i> .....	13
3.4. ZABBIX .....	15
3.4.1. <i>Installation and configuration - server</i> .....	15
3.4.2. <i>Install and connect zabbix-agent</i> .....	15
3.5. GRAFANA .....	16
3.5.1. <i>Installation and configuration</i> .....	16
3.6. ANSIBLE .....	18
3.6.1. <i>Configuration and installation</i> .....	18
3.7. TERRAFORM .....	19
3.7.1. <i>Installation</i> .....	19
3.8. DOCKER-COMPOSE .....	20
3.8.1. <i>Installation and configuration</i> .....	20
3.8.2. <i>Dockerfile</i> .....	20
3.8.3. <i>Testing the solution</i> .....	21
<b>4. FLOWS .....</b>	<b>23</b>
4.1. CONTACT AND INSTALLATION FLOW .....	23
4.2. APPLICATION FLOW .....	24
4.3. MONITORING FLOW .....	25
4.4. BACKUP FILES FLOW .....	26
<b>5. SET UP FIREPASS SERVER .....</b>	<b>27</b>

# IMAGE INDEX

IMAGE 1 SPRING-BOOT POM.XML CONFIGURATION .....	5
IMAGE 2 DOCKER PLUGIN IN POM.XML .....	6
IMAGE 3 SPRINGBOOT SECURITY FILTER USERS BY API-KEY .....	6
IMAGE 4 LOGIN METHOD IN THE RESTAPI .....	7
IMAGE 5 API METHOD FOR GET USER DATA .....	7
IMAGE 6 DATABASE PROPERTIES FILE .....	8
IMAGE 7 CONNECTION TO A DATABASE USING PREPARED STATEMENT .....	8
IMAGE 8 POSTMAN APP .....	9
IMAGE 9 GOOD RESPONSE OF THE API .....	9
IMAGE 10 POSTMAN APIKEY ERROR 400 .....	10
IMAGE 11 MYSQL SERVICE RUNNING .....	11
IMAGE 12 FIREPASS DB USER ACTION RANGE .....	11
IMAGE 13 APP TO FILL THE DATABASE .....	12
IMAGE 14 FIREPASSDML.SQL FILE .....	12
IMAGE 15 NGINX PROXI MANAGER .....	13
IMAGE 16 NGINX DOCKER-COMPOSE FILE CONFIGURATION .....	13
IMAGE 17 NGINX PROXY HOST CONFIGURATION .....	14
IMAGE 18 SSL CERTIFICATES MANAGED BY NGINX .....	14
IMAGE 19 CREATE USER AND SET THE DATABASE .....	15
IMAGE 20 GENERATE PSK KEY .....	15
IMAGE 21 ADD A HOST TO THE SERVER .....	15
IMAGE 22 GRAFANA SERVICE RUNNING .....	16
IMAGE 23 ADD NEW DATABASE SOUCE ON GRAFANA .....	16
IMAGE 24 GRAFANA DASHBOARD .....	17
IMAGE 25 ADD ANSIBLE REPOSITORY .....	18
IMAGE 26 ANSIBLE CONFIGURATION FILE .....	18
IMAGE 27 INSTALL TERRAFORM .....	19
IMAGE 28 INSTALL DOCKER .....	20
IMAGE 29 EXECUTION OF A DOCKER-COMPOSE .....	20
IMAGE 30 DOCKERFILE EXAMPLE .....	20
IMAGE 31 IMAGES IN THE INSIDE THE SERVER .....	21
IMAGE 32 CONTAINERS RUNNING IN THE SERVER .....	21
IMAGE 33 LOGS IN THE CONTAINER .....	21
IMAGE 34 WEB PAGE RUNNING IN THE CONTAINER .....	22
IMAGE 35 CONTACT AND INSTALLATION FLOW .....	23
IMAGE 36 APPLICATION FLOW .....	24
IMAGE 37 MONITORING FLOW .....	25
IMAGE 38 BACKUP FILES FLOW .....	26
IMAGE 39 AMAZON AMI OPENVPN .....	27
IMAGE 40 TERMS & CONDITIONS OF OPENVPN .....	27
IMAGE 41 DEFAULT CONFIGURATION OPENVPN .....	28
IMAGE 42 SET STRONG PASSWORD TO OPENVPN .....	28

# 1. INTRODUCTION

This document contains all the information about the resources used by the firepass project. The document also explains all the work done in developing the firepass system with a detailed description of the services and techniques used for each element of the final product.

## 2. SERVICES

Microservices are both an architectural style and a way of programming software. With microservices, applications are broken down into their smallest, independent elements. Unlike the traditional, monolithic approach to applications, where everything is compiled into a single piece, microservices are independent elements that work together to perform the same tasks. Each of these elements or processes is a microservice.

### 2.1. DINAHOSTING

Dinahosting is one of the most popular eCommerce hosting and domain registration services in Spain. After 20 years dedicated to both tasks, we can say that they have earned a very good reputation, both among their e-commerce clients and those of any other type of web project (dinahosting, s.f.).

### 2.2. LOAD BALANCER

Load balancing is a concept used in computer systems administration that refers to the technique used to share the work to be done among several computers, processes, disks or other resources.

### 2.3. WEB SERVER

A web server or HTTP server is a computer programme that processes a server-side application, making bidirectional or unidirectional and synchronous or asynchronous connections to the client and generating or delivering a response in any language or client-side application.

### 2.4. REST – API

Representational State Transfer or REST is a style of software architecture for distributed hypermedia systems such as the World Wide Web.

## 2.5. DATA BASE

A database is responsible not only for storing data, but also for connecting them together in a logical unit. In general terms, a database is a collection of structured data belonging to the same context and, in terms of its function, it is used to manage large amounts of information electronically.<sup>1</sup> In this sense, a library can be considered a database consisting mostly of documents and texts printed on paper and indexed for consultation. Nowadays, due to the technological development of fields such as computing and electronics, most databases are in digital format, being an electronic component, therefore a wide range of solutions to the problem of data storage have been developed and are offered.

## 2.6. MONITORING

The monitoring of the computer system consists of the installation of a series of sensors in the different hardware and software elements so that, 24 hours a day and 7 days a week, these sensors register the situation of each of the aspects that we control.

## 2.7. AUTOMATION SERVICE

IT automation, also called infrastructure automation, is the use of software systems to create repeatable instructions and processes that replace or reduce human interaction with IT systems.

## 2.8. FIREPASS SERVER

The firepass server is an active service which will be deployed on the client's network. This service is in charge of making HTTPS requests to the firepass api to securely and effectively obtain the data. This data will be decrypted by the service and displayed to the user.

## 3. SOFTWARE TOOLS

Commonly used software tools serve the purpose of facilitating, optimising and improving the performance of our work. The solutions offered by these tools can be applied in different areas of a company and help in the development of the most complex to the simplest tasks.

### 3.1. REST-API

#### 3.1.1. JAVA-SPRING BOOT

Spring was born from the complexity that had to be done when creating java enterprise edition projects, the EJBs were very complex and the way in which the deployment of the project was done was very cumbersome. Spring Boot contains a lightweight infrastructure that eliminates most of the work of configuring Spring-based applications. The goal of Spring Boot is to provide a set of tools to quickly build Spring applications that are easy to configure (spring-boot, s.f.).

#### 3.1.2. CONFIGURATION

For the configuration of Spring-boot it is necessary to generate a java maven project and write the dependency "springframework". This dependency is in charge of generating a project which has all the spring boot facilities to be able to create code. Also, this framework has been



```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
5     http://maven.apache.org/xsd/maven-4.0.0.xsd">
6   <modelVersion>4.0.0</modelVersion>
7   <parent>
8     <groupId>org.springframework.boot</groupId>
9     <artifactId>spring-boot-starter-parent</artifactId>
10    <version>2.7.4</version>
11  </parent>
12  <groupId>com</groupId>
13  <artifactId>Rest-APIfirepass</artifactId>
14  <version>0.0.1-SNAPSHOT</version>
15  <name>Rest-APIfirepass</name>
16  <description>Firepass Rest API</description>
17  <properties>
18    <java.version>11</java.version>
19  </properties>
20  <dependencies>
21    <dependency>
22      <groupId>com.google.code.gson</groupId>
23      <artifactId>gson</artifactId>
24      <version>2.9.1</version>
25    </dependency>
26    <dependency>
27      <groupId>mysql</groupId>
28      <artifactId>mysql-connector-java</artifactId>
29      <version>8.0.30</version>
30    </dependency>
31    <dependency>
32      <groupId>org.springframework.boot</groupId>
33      <artifactId>spring-boot-starter-web</artifactId>
34    </dependency>
```

Image 1 Spring-boot pom.xml configuration

used to create a Rest api so the dependencies "gson", for the use of json files and "mysql-connector-java" for the secure connections against the database have been needed.

Another important aspect of the configuration is the creation of a docker image to be able to deploy this code quickly and effectively. For this, a docker plugin has been used which is in charge of generating a docker image.

```
<build>
  <plugins>
    <plugin>
      <groupId>com.google.cloud.tools</groupId>
      <artifactId>jib-maven-plugin</artifactId>
      <version>3.3.1</version>
      <configuration>
        <from>
          <image>openjdk:17</image>
        </from>
        <to>
          <image>firepass-apib:latest</image>
        </to>
        <container>
          <creationTime>USE_CURRENT_TIMESTAMP</creationTime>
          <ports>
            <port>8080</port>
          </ports>
        </container>
      </configuration>
      <executions>
        <execution>
          <phase>verify</phase>
          <goals>
            <goal>dockerBuild</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>
```

Image 2 Docker plugin in pom.xml

### 3.1.3. DEVELOPMENT

In terms of development, we should think that this api will be available to everyone, but that does not mean that anyone can use it. For this, a security method has been implemented

```
@Override
public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
    throws IOException, ServletException {

    HttpServletRequest req = (HttpServletRequest) request;
    HttpServletResponse res = (HttpServletResponse) response;

    if (checkApiKey(req.getHeader(name: "api_key"))) {
        chain.doFilter(request, response);
    }else{
        res.sendError(sc: 401, msg: "Unauthorized");
        chain.doFilter(request, res);
    }
}
```

Image 3 Springboot security filter users by api-key

by Api-key by which users are filtered and it is discovered if they have access to use the services of pa API or not.

Once the filter is created, we use the maps to know what the user wants to do. That is to say, there is a method for each different action that the client wants to perform. Likewise, as there is a different method for each call, there is also a different connection to the database created recursively with parallel calls.

```
@PostMapping(value = "/login")
public String getLoginPass(@RequestParam String username, @RequestParam String password) {

    System.out.println("username: " + username + " password: " + password);

    int user_id = dbController.checkLogin(username, password);

    if (user_id != -1) {
        List<Credential> credentials = dbController.loadCredential(user_id);
        String json = gson.toJson(credentials);
        return json;
    } else {
        throw new ResourceNotFoundException();
    }
}
```

**Image 4** Login method in the RestApi

The login is one of the most important examples as it is the one used by everyone to access their credentials in a secure way, but there are more methods which enable different actions such as loading user information, loading credentials and even creating new credentials.

```
@GetMapping(value = "/user/{user_id}")
public String getUserData(@PathVariable("user_id") int user_id) {

    user user = dbController.loadUser(user_id);
    if (user != null) {
        String userJson = gson.toJson(user);
        return userJson;
    } else {
        throw new ResourceNotFoundException();
    }
}
```

**Image 5** Api method for get user data



Finally, the connections to the database must be discussed. These connections are very delicate, so they must be made in a secure and private way. For this, there is a file which stores the credentials and the specific user of the database.

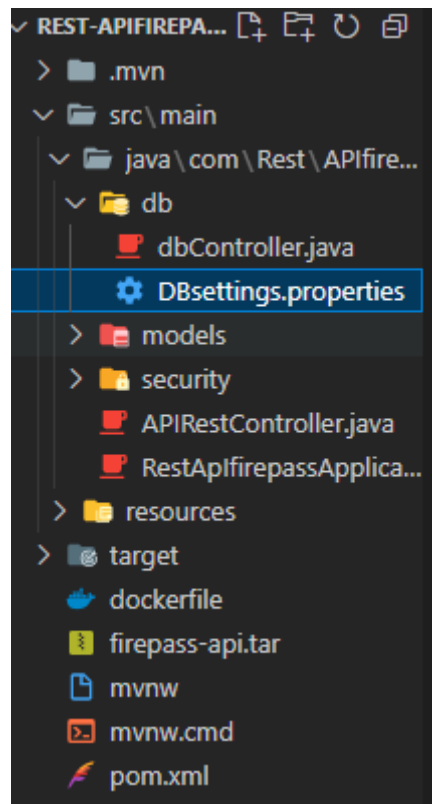


Image 6 Database properties file

In order to prevent MySQL injection attacks, a prepared statement system has been implemented to prepare the database request.

```
public static user loadUser(int user_id) {
    Connection connection = null;
    user user = null;
    try {
        connection = generateConnection();
        PreparedStatement preparedStatement = connection.prepareStatement(sql: "select * from users where user_id = ?");
        preparedStatement.setInt(parameterIndex: 1, user_id);
        ResultSet rs = preparedStatement.executeQuery();

        if (rs.next()) {
            user = new user(user_id: 0, rs.getString(columnLabel: "username"), rs.getString(columnLabel: "first_name"),
                rs.getString(columnLabel: "second_name"), rs.getString(columnLabel: "email"), rs.getString(columnLabel: "postal_code"),
                rs.getString(columnLabel: "country"), rs.getString(columnLabel: "birth_date"), rs.getString(columnLabel: "register_date"));
            user.setUser_id(rs.getInt(columnLabel: "user_id"));
        }

        connection.close();
    } catch (IOException | SQLException e) {
        e.printStackTrace();
    }

    return user;
}
```

Image 7 Connection to a database using prepared statement

### 3.1.4. TESTING THE SOLUTION

In order to make sure that the solution is well implemented and to prove that it works correctly, I have used the Postman application.

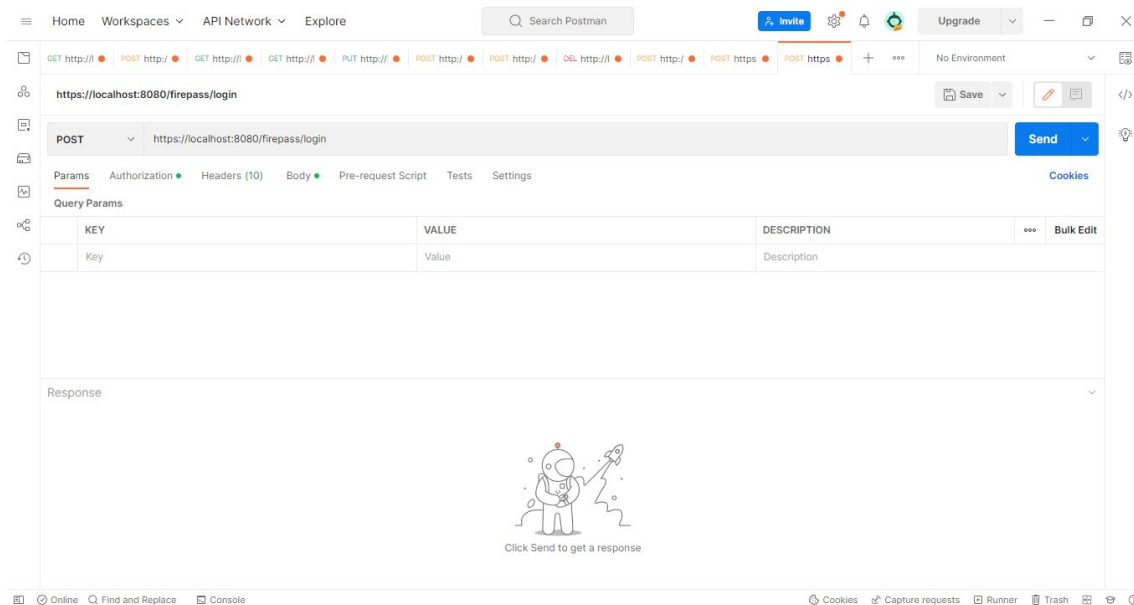


Image 8 Postman app

This application allows to make requests to the apis and therefore we add the method, the URL of our service in HTTPS and all the credentials.

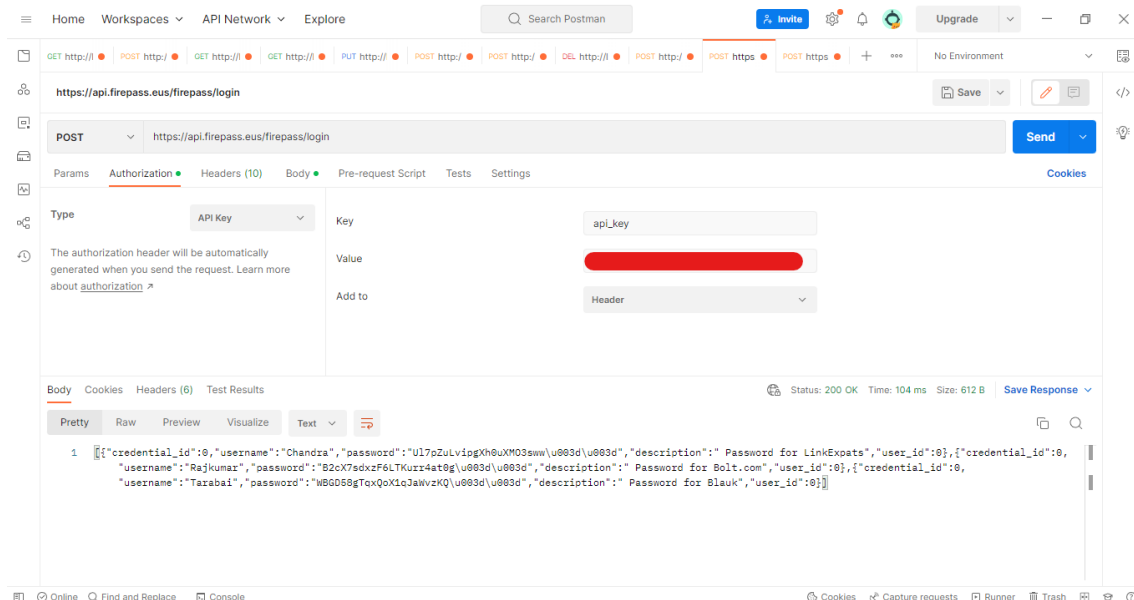
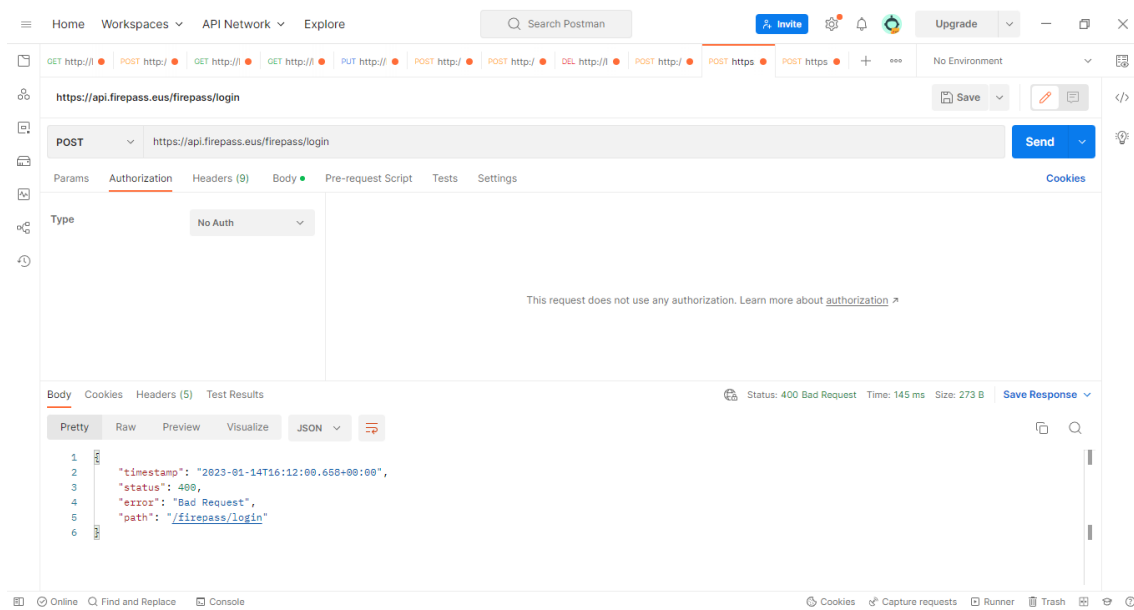


Image 9 Good response of the api

As we can see the api responds in a json format perfectly to the request but in the case of not adding the credential "apikey" it will return an error.



**Image 10** Postman apikey error 400

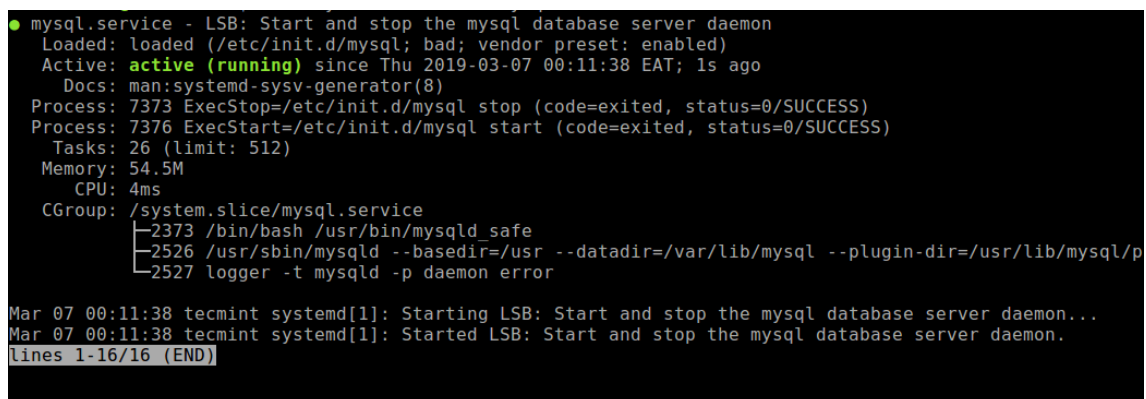
With this we prove that the api works correctly and is ready to be deployed in production.

## 3.2. MYSQL

MySQL is a relational database management system developed under a dual license: General Public License/Commercial License by Oracle Corporation and is considered to be the world's most popular open-source database,<sup>12</sup> and one of the most popular in general alongside Oracle and Microsoft SQL Server, all for web development environments (mysql, s.f.).

### 3.2.1. INSTALLATION AND CONFIGURATION

The installation of mysql was managed thanks to the blueocean website (digitalocean, s.f.). In this page it explains command by command what you must execute in order to have a database running on a server.

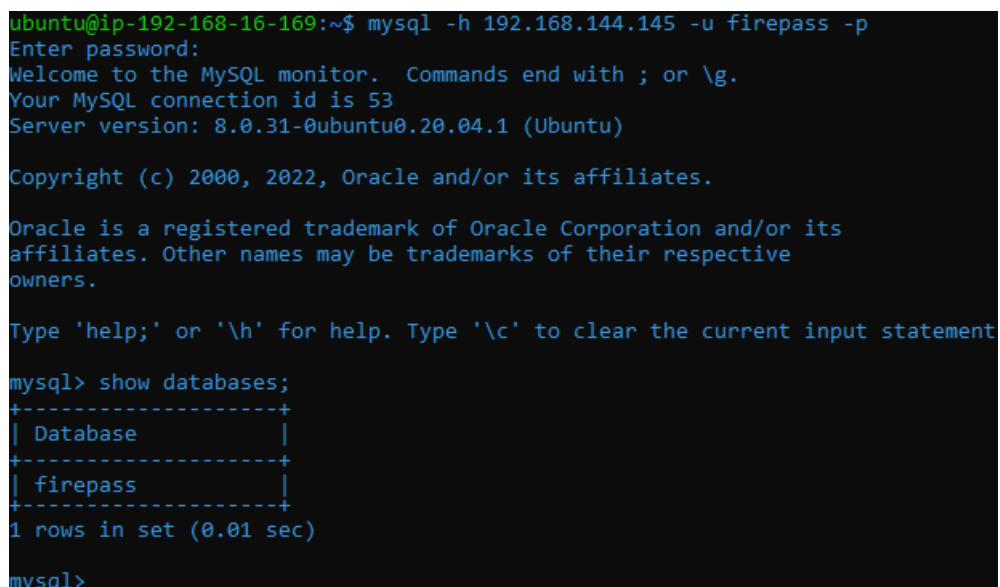


```
● mysql.service - LSB: Start and stop the mysql database server daemon
   Loaded: loaded (/etc/init.d/mysql; bad; vendor preset: enabled)
   Active: active (running) since Thu 2019-03-07 00:11:38 EAT; 1s ago
     Docs: man:systemd-sysv-generator(8)
  Process: 7373 ExecStop=/etc/init.d/mysql stop (code=exited, status=0/SUCCESS)
  Process: 7376 ExecStart=/etc/init.d/mysql start (code=exited, status=0/SUCCESS)
    Tasks: 26 (limit: 512)
   Memory: 54.5M
      CPU: 4ms
   CGroup: /system.slice/mysql.service
           └─2373 /bin/bash /usr/bin/mysqld_safe
             └─2526 /usr/sbin/mysqld --basedir=/usr --datadir=/var/lib/mysql --plugin-dir=/usr/lib/mysql/p
               └─2527 logger -t mysqld -p daemon error

Mar 07 00:11:38 tecmint systemd[1]: Starting LSB: Start and stop the mysql database server daemon...
Mar 07 00:11:38 tecmint systemd[1]: Started LSB: Start and stop the mysql database server daemon.
lines 1-16/16 (END)
```

Image 11 MySQL service running

Also, for security reasons it was decided to create a specific user who has access to the database. In other words, there is only one user who is only able to interact with the firepass database.



```
ubuntu@ip-192-168-16-169:~$ mysql -h 192.168.144.145 -u firepass -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 53
Server version: 8.0.31-0ubuntu0.20.04.1 (Ubuntu)

Copyright (c) 2000, 2022, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement

mysql> show databases;
+-----+
| Database |
+-----+
| firepass |
+-----+
1 rows in set (0.01 sec)

mysql>
```

Image 12 Firepass DB user action range

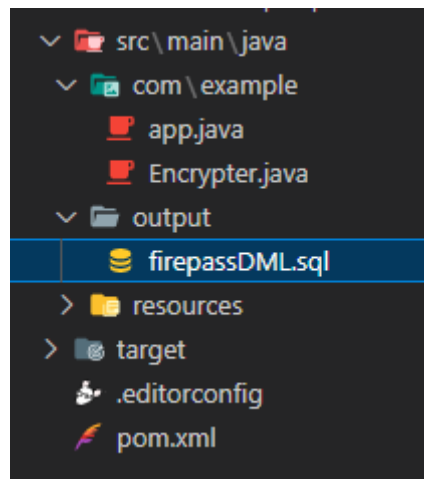
### 3.2.2. FILL THE DATA BASE

In order to do the tests and to be able to show much more attractive demos a java application was created which generates a random script with data. This application consists of a simple code which is fed from different files to generate the generation script.

```
public void run() throws IOException {  
    loadFiles();  
    generateUsers();  
    generateCredentials();  
    generateDMLScript();  
}  
  
private void loadFiles() throws FileNotFoundException { ...  
  
private void generateDMLScript() throws IOException { ...  
  
private List<String> loadFile(String path) throws FileNotFoundException { ...  
  
private void generateUsers() { ...  
  
private void generateCredentials() { ...  
  
private String passwordRandomer() { ...  
  
public int randomer(int max, int min) { ...
```

**Image 13** App to fill the database

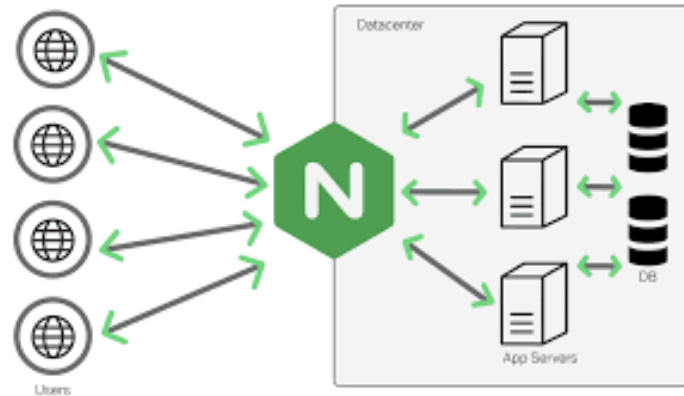
The result of this programme is a script with more than 1500 lines of data.



**Image 14** FirepassDML.sql file

### 3.3. NGINX

Nginx is a high-performance lightweight reverse proxy/web server and proxy for email protocols; it is free and open-source software, licensed under the simplified BSD License; there is also a commercial version distributed under the name Nginx Plus (nginx, s.f.).



**Image 15** Nginx proxy manager

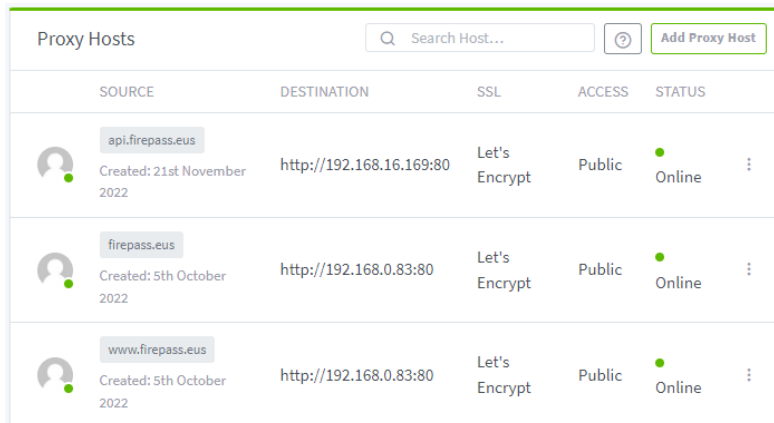
#### 3.3.1. INSTALLATION AND CONFIGURATION




The installation is directly dockerised to have an easier backup management thanks to the docker volume system. That said, it can be understood that in the case of Nginx the service is installed thanks to a docker-compose (docker, s.f.).

```
version: '3'
services:
  nginx:
    image: 'jc21/nginx-proxy-manager:latest'
    container_name: app
    restart: always
    ports:
      - '81:81'
      - '80:80'
      - '443:443'
    volumes:
      - './nginx_proxy_manager_data:/data
      - './nginx_proxy_manager_letsencrypt:/etc/letsencrypt
```

**Image 16** Nginx docker-compose file configuration

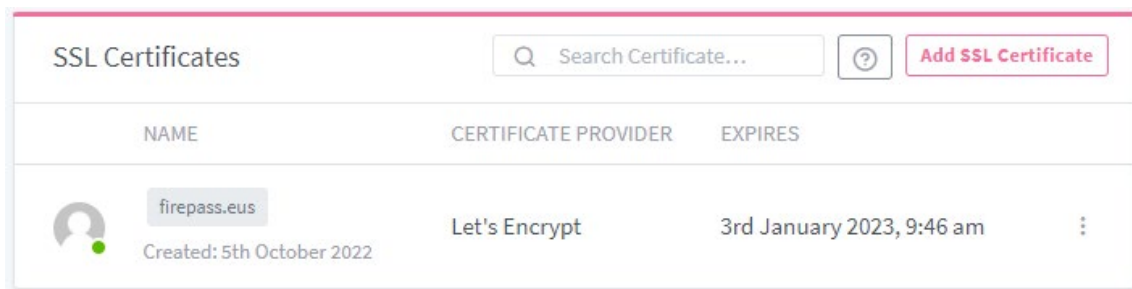
For the configuration of nginx we go to the configuration section of the UI and after changing the password and creating a different user to the administrator, we will create the first records of our servers.




SOURCE	DESTINATION	SSL	ACCESS	STATUS
 <b>api.firepass.eus</b> Created: 21st November 2022	http://192.168.16.169:80	Let's Encrypt	Public	<span>Online</span>
 <b>firepass.eus</b> Created: 5th October 2022	http://192.168.0.83:80	Let's Encrypt	Public	<span>Online</span>
 <b>www.firepass.eus</b> Created: 5th October 2022	http://192.168.0.83:80	Let's Encrypt	Public	<span>Online</span>

**Image 17** Nginx proxy host configuration

Likewise, nginx is in charge of managing the SSL certificates that we have thanks to Let's encrypt. In our case we have a wildcard type certificate that allows its use in all the subdomains that we have at our disposal as for example the api.



NAME	CERTIFICATE PROVIDER	EXPIRES
 <b>firepass.eus</b> Created: 5th October 2022	Let's Encrypt	3rd January 2023, 9:46 am

**Image 18** SSL certificates managed by Nginx

## 3.4. ZABBIX

Zabbix is a Network Monitoring System created by Alexei Vladishev. It is designed to monitor and record the status of various network services, Servers, and network hardware. It uses MySQL, PostgreSQL, SQLite, Oracle or IBM DB2 as database (zabbix, s.f.).

### 3.4.1. INSTALLATION AND CONFIGURATION - SERVER

For the installation of Zabbix it is necessary to install all the dependencies such as, zabbix-server-mysql, zabbix-frontend-php, zabbix-nginx-conf, zabbix-sql-scripts, zabbix-agent and mysql-server. Once all the dependencies are installed, the database is configured by creating a user able to interact with the Zabbix database.

```
mysql> create database zabbix character set utf8mb4 collate utf8mb4_bin;
mysql> create user zabbix@localhost identified by 'password';
mysql> grant all privileges on zabbix.* to zabbix@localhost;
```

**Image 19** Create user and set the database

When creating the database you configure the Zabbix server with the database password and in nginx you enable port 8080 and the public domain.

### 3.4.2. INSTALL AND CONNECT ZABBIX-AGENT

In order for the server to receive the data from the different servers that make up the product firpass, an agent has to be installed that collects the data and sends it in real time. To begin with, we install the agent and generate a psk key in hex 32 in a file.

```
heiko@DESKTOP-IDN3683:~$ sudo sh -c "openssl rand -hex 32 > zabbix_agentd.psk"
[sudo] password for heiko:
heiko@DESKTOP-IDN3683:~$ ls
zabbix_agentd.psk
heiko@DESKTOP-IDN3683:~$ cat zabbix_agentd.psk
4f6fde89a7844eefdeaa0e66cee14ccb54077d3f4042ad0bfd47d9627a38b1d8
```

**Image 20** Generate psk key

Once the key has been generated in the Zabbix agent configuration file, we have to specify the server ip and the TLS encryption enabled with the previously generated key. Finally, in the server we will add the new host inside the inventory and at the same time we will add the new generated key so that the communications between the client and the server are encrypted.

The screenshot shows the Zabbix web interface for adding a new host. The 'Encryption' tab is selected under the 'Host inventory' section. In the 'Connections to host' section, 'PSK' is chosen. The 'PSK identity' is set to 'PSK 001'. The 'Host name' is 'Second Ubuntu Server', and the 'Groups' are 'Linux servers'. The 'Agent interfaces' table shows the IP address '207.154.242.82' and the 'Connect to' dropdown set to 'IP'.

**Image 21** Add a host to the server



## 3.5. GRAFANA

Grafana is a free software based on Apache 2.0 licence, which allows the visualisation and formatting of metric data. It allows the creation of dashboards and graphs from multiple sources, including time series databases such as Graphite, InfluxDB and OpenTSDB (grafana, s.f.).

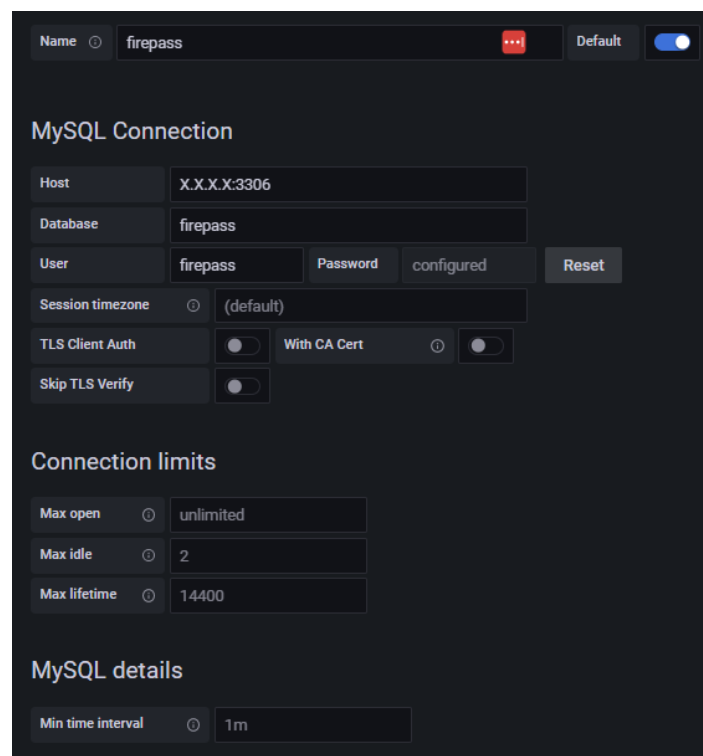
### 3.5.1. INSTALLATION AND CONFIGURATION

To install Grafana, simply download the monitoring software and run it.

```
• grafana-server.service - Grafana instance
  Loaded: loaded (/lib/systemd/system/grafana-server.service; disabled; vendor preset: enabled)
  Active: active (running) since Thu 2022-11-30 08:08:10 UTC; 4s ago
  Docs: http://docs.grafana.org
  Main PID: 15982 (grafana-server)
  Tasks: 7 (limit: 1137)
```

Image 22 Grafana service running

Once the service is activated, we will log in to the viewing page and update the administrator's password. We will also add a new MySQL resource to monitor the database.



The screenshot shows the Grafana web interface for configuring a new MySQL data source. At the top, the 'Name' field is set to 'firepass', and a 'Default' toggle is turned on. The 'MySQL Connection' section contains the following fields: 'Host' (X.X.X.X:3306), 'Database' (firepass), 'User' (firepass), 'Password' (configured), and a 'Reset' button. Below these are 'Session timezone' (default), 'TLS Client Auth' (disabled), 'With CA Cert' (disabled), and 'Skip TLS Verify' (disabled). The 'Connection limits' section has 'Max open' (unlimited), 'Max idle' (2), and 'Max lifetime' (14400). The 'MySQL details' section has 'Min time interval' (1m).

Image 23 Add new database source on grafana

Finally, we will create a dashboard that will allow us to see the non-sensitive data in the database and we will have updated information about the data stored in the database.

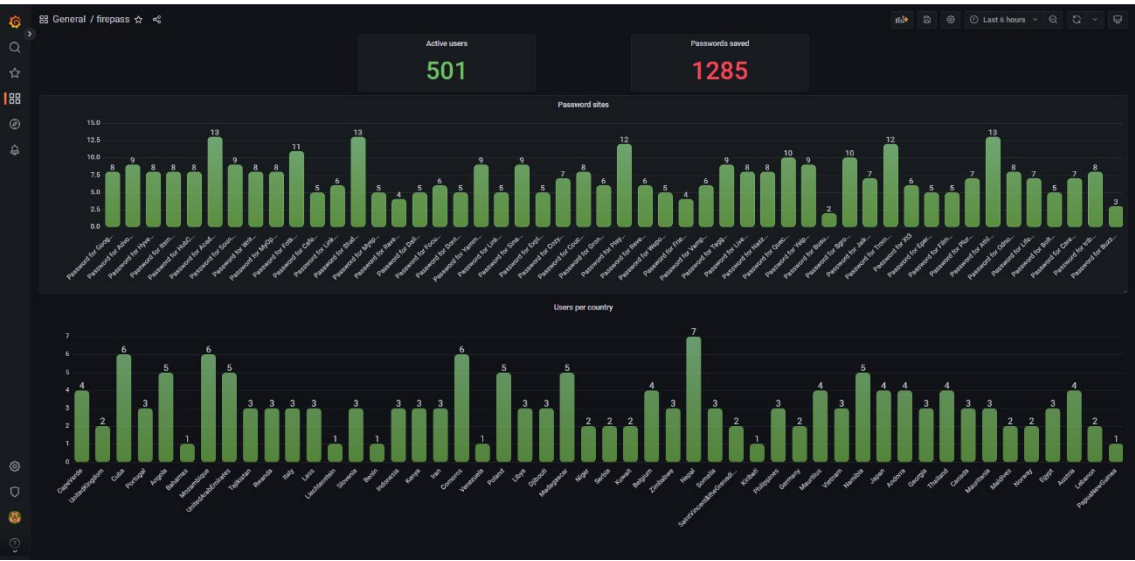


Image 24 grafana dashboard

## 3.6. ANSIBLE

Ansible is a free software platform for configuring and managing computers. It combines multi-node installation, ad hoc task execution and configuration management. Additionally, Ansible is categorised as an orchestration tool (ansible, s.f.).

### 3.6.1. CONFIGURATION AND INSTALLATION

For the installation of this service you have to be aware that the repositories are not included for the version of ubuntu you are using, so you have to add them manually and later update them to download and install the application.

```
[sudo] password for heiko:

Ansible is a radically simple IT automation platform that makes your applications and systems easier to deploy.
Avoid writing scripts or custom code to deploy and update your applications- automate in a language that approach
es plain English, using SSH, with no agents to install on remote systems.

http://ansible.com/

If you face any issues while installing Ansible PPA, file an issue here:
https://github.com/ansible-community/ppa/issues
More info: https://launchpad.net/~ansible/+archive/ubuntu/ansible
Press [ENTER] to continue or ctrl-c to cancel adding it
gpg: keybox '/tmp/tmp5h5s2hve/pubring.gpg' created
gpg: /tmp/tmp5h5s2hve/trustdb.gpg: trustdb created
gpg: key 93C4A3FD7BB9C367: public key "Launchpad PPA for Ansible, Inc." imported
gpg: Total number processed: 1
gpg:         imported: 1
gpg: no valid OpenPGP data found.
```

Image 25 Add ansible repository

Later we will go into the configuration file and add the servers and make sure they all have the correct version of python installed.

```
[servers_all]
Load_balancer ansible_host=x.x.x.x
web_page ansible_host=x.x.x.x
Rest_api ansible_host=x.x.x.x
Database ansible_host=x.x.x.x
Monitoring_serv ansible_host=x.x.x.x
Backup_Server ansible_host=x.x.x.x

[all:vars]
ansible_python_interpreter=/usr/bin/python3
```

Image 26 Ansible configuration file

## 3.7. TERRAFORM

terraform is infrastructure as code software developed by HashiCorp. It allows users to define and configure a data centre infrastructure in a high-level language, generating an execution plan to deploy the infrastructure on OpenStack, for example, or other service providers such as AWS, IBM Cloud (formerly Bluemix), Google Cloud Platform, Linode, Microsoft Azure, Oracle Cloud Infrastructure or VMware vSphere (terraform, s.f.).

### 3.7.1. INSTALLATION

The installation of terraform is very simple as you only need to add the terraform repository and install it.

```
root@ip-10-111-11-241:/home/ubuntu# sudo apt install terraform
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  terraform
0 upgraded, 1 newly installed, 0 to remove and 35 not upgraded.
Need to get 32.7 MB of archives.
After this operation, 79.3 MB of additional disk space will be used.
Get:1 https://apt.releases.hashicorp.com bionic/main amd64 terraform amd64 1.0.7
Fetched 32.7 MB in 1s (53.7 MB/s)
Selecting previously unselected package terraform.
(Reading database ... 92955 files and directories currently installed.)
Preparing to unpack .../terraform_1.0.7_amd64.deb ...
Unpacking terraform (1.0.7) ...
Setting up terraform (1.0.7) ...
```

**Image 27** Install terraform

## 3.8. DOCKER-COMPOSE

Docker is an open-source project that automates the deployment of applications inside software containers, providing an additional layer of abstraction and automation of application virtualisation across multiple operating systems (docker, s.f.).

### 3.8.1. INSTALLATION AND CONFIGURATION

To install docker-compose we first need to install docker. To do this, we will download the docker certificates, add the repository and install the docker service.

```
• docker.service - Docker Application Container Engine
  Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)
  Active: active (running) since Tue 2022-12-01 17:00:41 UTC; 17s ago
  TriggeredBy: • docker.socket
  Docs: https://docs.docker.com
  Main PID: 24321 (dockerd)
Creating network "firepassDN" with the default driver
Pulling web (nginx:alpine)...
alpine: Pulling from library/nginx
cbdbe7a5bc2a: Pull complete
10c113fb0c77: Pull complete
9ba64393807b: Pull complete
c829a9c40ab2: Pull complete
61d685417b2f: Pull complete
Digest: sha256:57254039c6313fe8c53f1acbf15657ec9616a813397b74b063e32443427c5502
Status: Downloaded newer image for nginx:alpine
Creating nginx_proxi_manager ... done
```

**Image 29** Execution of a docker-compose

Having docker installed we must download from github the latest version of docker-compose and give execution permissions to the bin file of docker-compose. This way we will be able to create a .yml file which has the configuration and execute it.

### 3.8.2. DOCKERFILE

Dockerfile is a text file containing the instructions needed to create a new container image.

```
FROM openjdk:17-jdk-alpine

RUN apk add --no-cache tzdata
ENV TZ='Europe/Amsterdam'
RUN ln -snf /usr/share/zoneinfo/$TZ /etc/localtime && echo $TZ > /etc/timezone

WORKDIR /

RUN mkdir app && chmod 777 app
COPY . /app

COPY target/firepassClient-1.0-SNAPSHOT-jar-with-dependencies.jar /app
WORKDIR /app

EXPOSE 8081

CMD ["java", "-jar", "firepassClient-1.0-SNAPSHOT-jar-with-dependencies.jar"]
```

**Image 30** Dockerfile example

In the above example we first start from the java image we need and set the time with a variable and running the command "RUN In -snf /usr/share/zoneinfo/\$TZ /etc/localtime && echo \$TZ > /etc/timezone". Later, we create the app folder, give it permissions, and copy the files. Finally, open port 8081 and execute the CMD command. This way we have an image that works perfectly in docker.

### 3.8.3. TESTING THE SOLUTION

In order to prove that the docker containers are working properly we will first see if the images are downloaded correctly to the server.

```
ubuntu@ip-192-168-0-83:~/docker-composefile$ sudo docker image ls
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
firepass      latest    ff97e0ce6ed5   3 months ago   345MB
```

Image 31 Images in the inside the server

As we can see we have the latest version of the firepass image available, so it is the correct image. Next, we will see if the container is running.

```
ubuntu@ip-192-168-0-83:~/docker-composefile$ sudo docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS
89cd8a432538   firepass:latest  "java -jar firepass-..." 3 months ago   Up 8 weeks    0.0.0.0:80->8080/tcp, :::80->8080/tcp
docker-composefile_web-app_1
```

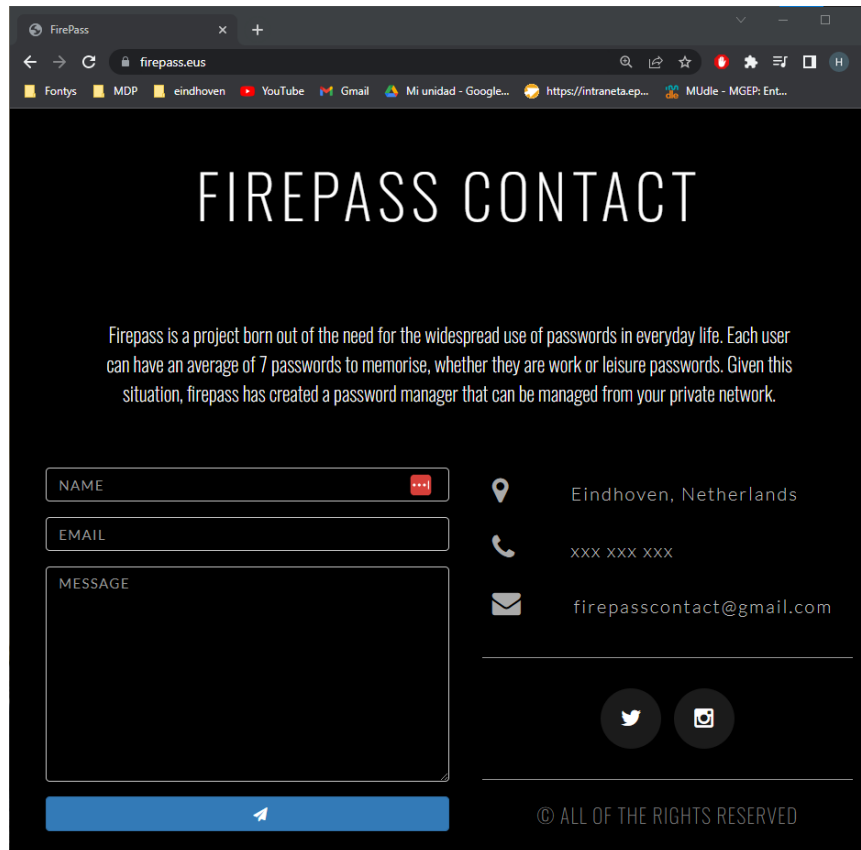
Image 32 Containers running in the server

As we can see it is running perfectly since it has been running for 8 weeks. Even so we will look at the logs of the container to know that there is no problem of execution in itself.

```
ubuntu@ip-192-168-0-83:~/docker-composefile$ sudo docker logs 89cd8a432538
Oct 14, 2022 6:44:52 PM org.apache.coyote.AbstractProtocol init
INFO: Initializing ProtocolHandler ["http-nio-8080"]
Oct 14, 2022 6:44:52 PM org.apache.catalina.core.StandardService startInternal
INFO: Starting service [Tomcat]
Oct 14, 2022 6:44:52 PM org.apache.catalina.core.StandardEngine startInternal
INFO: Starting Servlet engine: [Apache Tomcat/9.0.35]
Oct 14, 2022 6:44:52 PM org.apache.catalina.startup.ContextConfig getDefaultWebXmlFragment
INFO: No global web.xml found
Oct 14, 2022 6:44:53 PM org.apache.jasper.servlet.TldScanner scanJars
INFO: At least one JAR was scanned for TLDs yet contained no TLDs. Enable debug logging for this logger for a complete list of JARs that were scanned but no TLDs were found in them. Skipping unneeded JARs during scanning can improve startup time and JSP compilation time.
Oct 14, 2022 6:44:53 PM org.apache.coyote.AbstractProtocol start
INFO: Starting ProtocolHandler ["http-nio-8080"]
Oct 14, 2022 11:28:16 PM org.apache.coyote.http11.Http11Processor service
```

Image 33 Logs in the container

As we see there is no error within the application and finally as the example of this container is a web page we can check if it works by accessing the web.



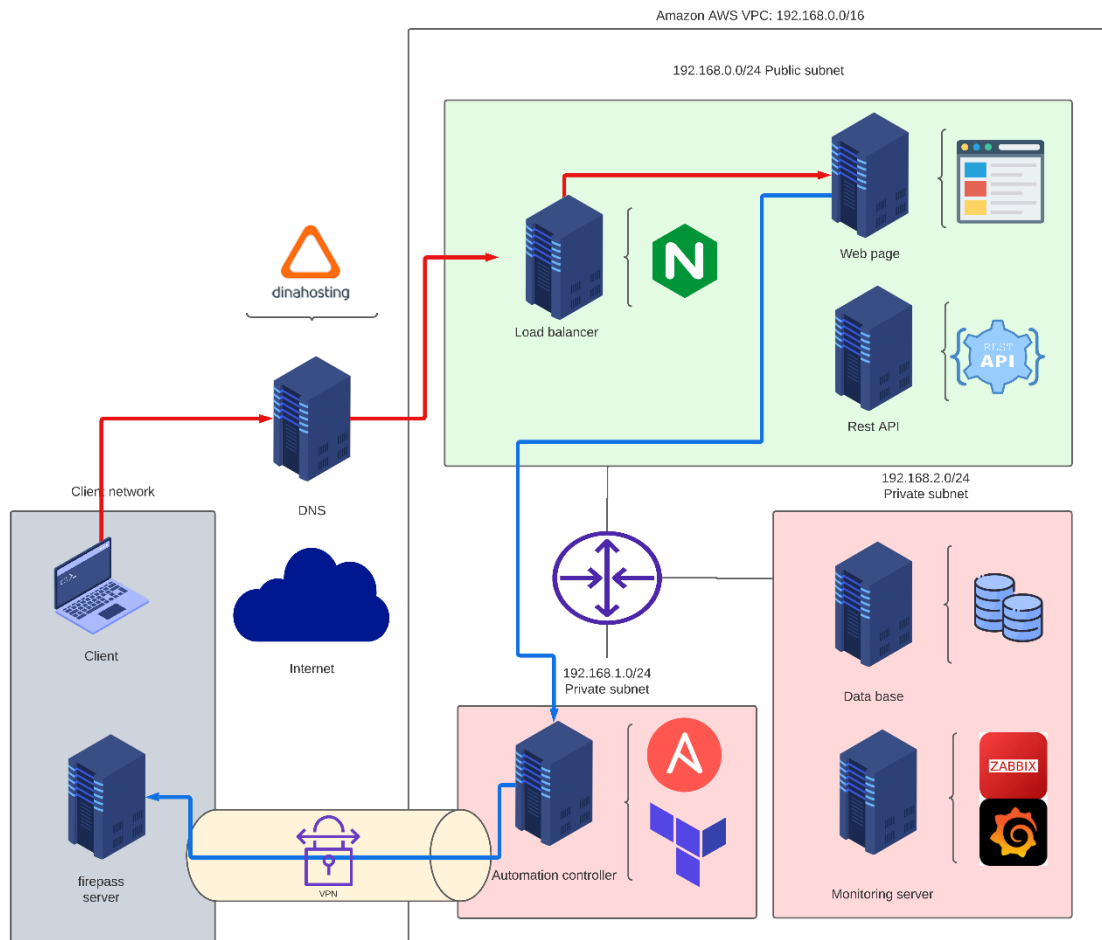
**Image 34** web page running in the container

As we can see we have perfect access to the web so docker, docker-compose and the container work perfectly.

## 4. FLOWS

In this project there are different data flows which will be explained through which sites they pass and which services are involved.

### 4.1. CONTACT AND INSTALLATION FLOW



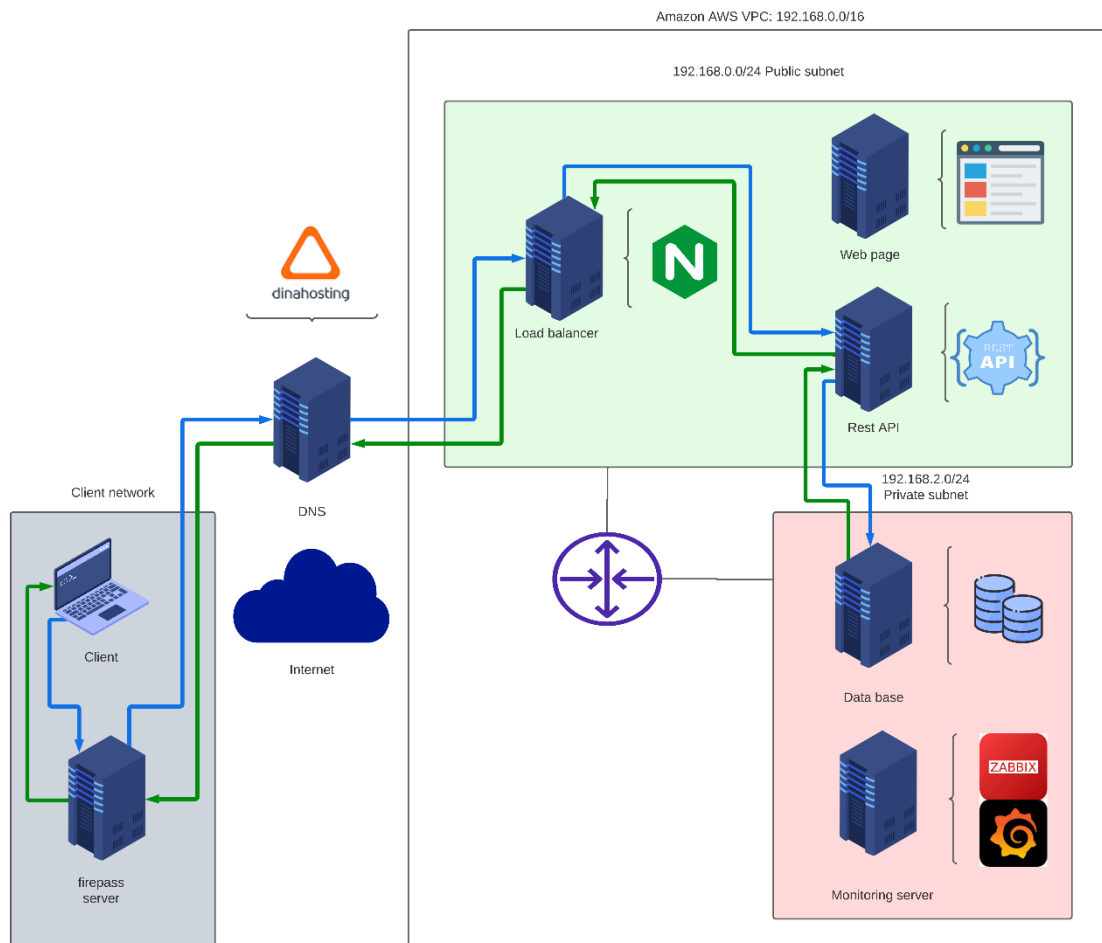
**Image 35** Contact and installation flow

As we can see in the image above, there are two flows, the red flow and the green flow. The red flow is the path that a client should take to contact us, starting from his computer he will go to the DNS servers that will redirect him to our balancer and finally he will reach the web page by sending an email alert.

On the other hand, we have the setup flow, which is activated when the user provides a VPN connection against the automation controller. This will execute the necessary commands to create the Firepass server.



## 4.2. APPLICATION FLOW

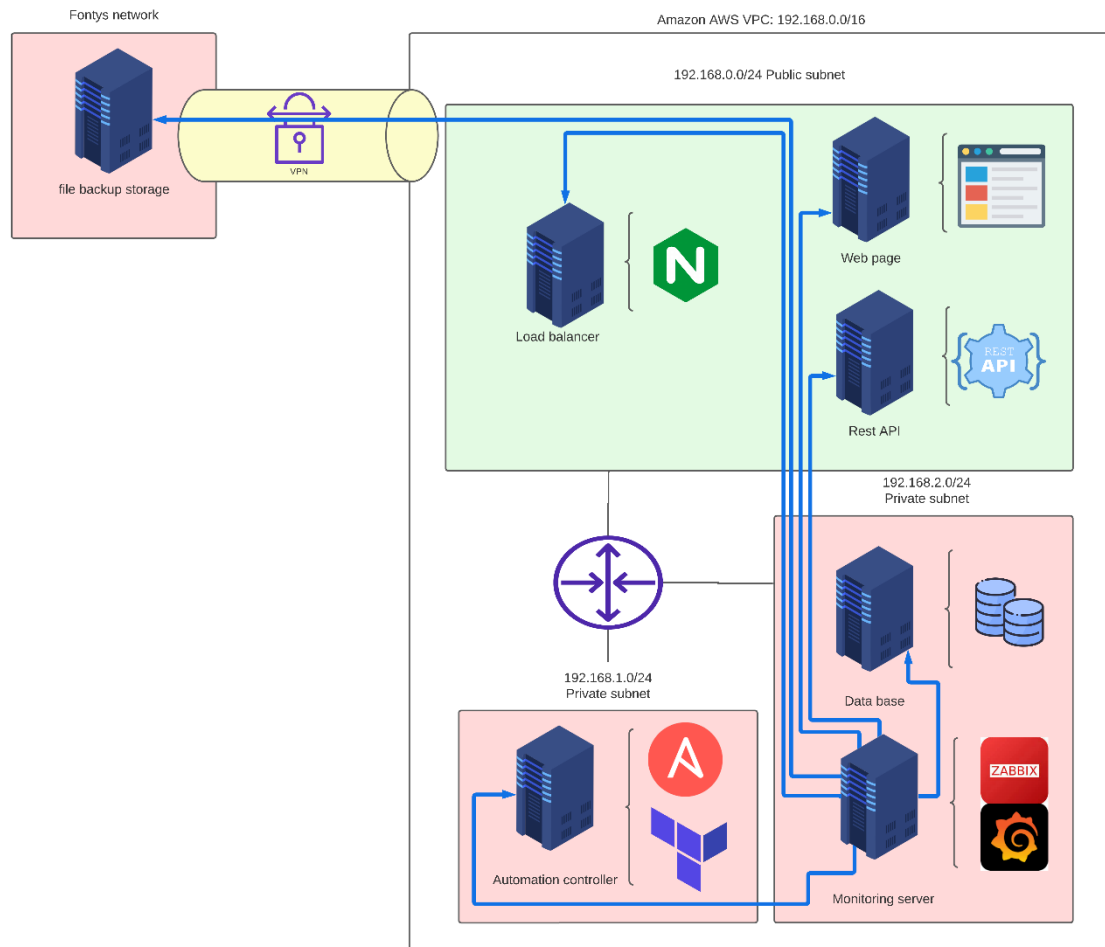


**Image 36** Application flow

The application flow is one of the most important flows of our application, because basically he is responsible for providing an active service to the customers who contract the product.

In this case, the flow has 2 parts. In the first one (blue route), the user asks for credentials and goes all the way through the DNS, load balancer and rest api until reaching the database with the encrypted data. This same path is repeated on the way back (green route) to deliver the data.

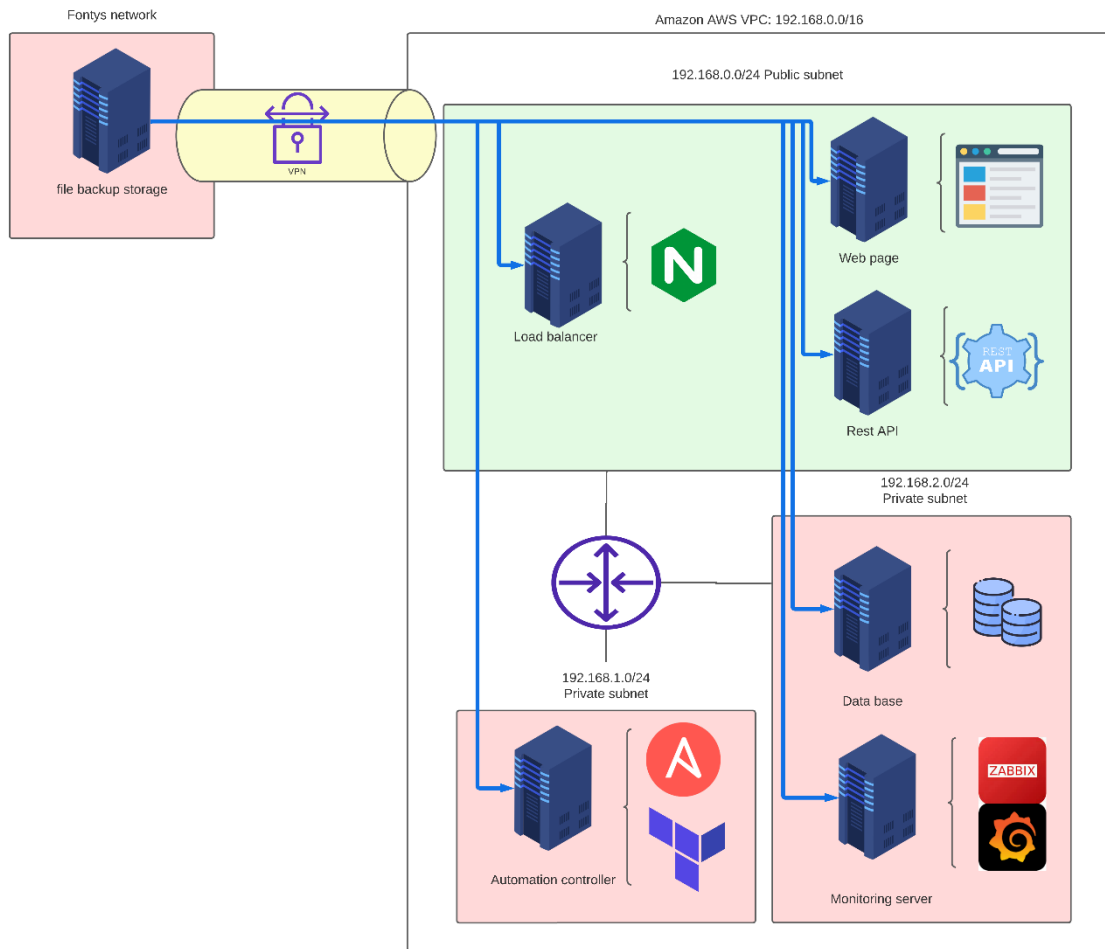
## 4.3. MONITORING FLOW



**Image 37** Monitoring flow

This flow is basically a centralised monitoring on the same server, so all paths are requests to the different servers in the infrastructure and then display the data on the web page of the Zabbix and Grafana server.

## 4.4. BACKUP FILES FLOW



**Image 38** Backup files flow

This flow is like the monitoring flow in the sense that all the flows leave from the same server and go to the rest of the servers, but with the difference that this time the backup server seeks to back up the different services and data in our network.

## 5. SET UP FIREPASS SERVER

For the installation you have to provide VPN credentials so that our server can automatically deploy our service. Once the server has been created, you will be able to remove the VPN with our network as it does not need a VPN. In order to create a vpn connexion first go to ec2 instances and create one

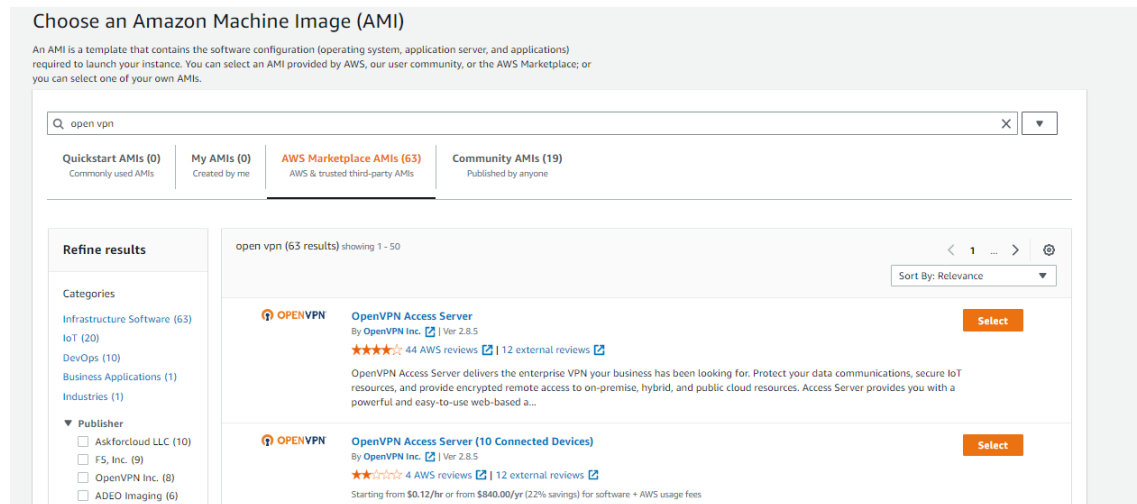


Image 39 Amazon AMI OpenVPN

select "continue" and finish configuring the parameters required by your network like giving access to ports 80 and 443 to http/s connections. Then connect to the newly created server and accept the OpenVPN terms and conditions.

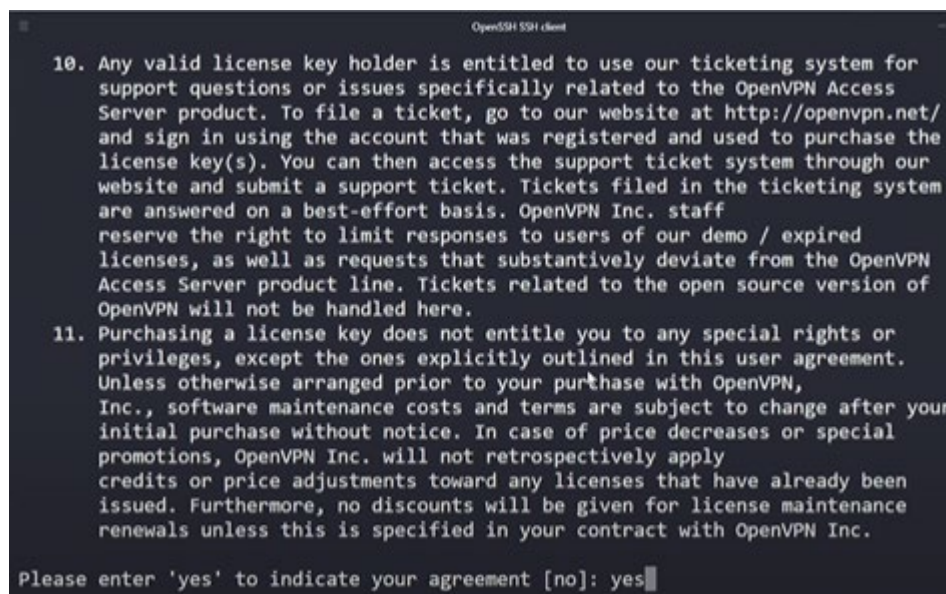


Image 40 Terms & Conditions of OpenVPN

Once you have accepted the terms and conditions, we will start the VPN with the default configuration by pressing enter all the time.

```
OpenVPN Access Server can be configured by accessing
its Admin Web UI using your Web browser.

Will this be the primary Access Server node?
(enter 'no' to configure as a backup or standby node)
> Press ENTER for default [yes]:

Please specify the network interface and IP address to be
used by the Admin Web UI:
(1) all interfaces: 0.0.0.0
(2) eth0: 172.31.88.21
Please enter the option number from the list above (1-2).
> Press Enter for default [1]:

Please specify the port number for the Admin Web UI.
> Press ENTER for default [943]:

Please specify the TCP port number for the OpenVPN Daemon
> Press ENTER for default [443]:

Should client traffic be routed by default through the VPN?
> Press ENTER for default [no]:
```

**Image 41** Default configuration OpenVPN

Finally, we will generate a reliable password and set it to openVPN with the command "sudo passwd openvpn".

```
openvpnas@ip-172-31-88-21:~$ sudo passwd openvpn
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
openvpnas@ip-172-31-88-21:~$
```

**Image 42** Set strong password to OpenVPN

Finally, the password I generated and the public ip of the server. This way our automation controller will be able to deploy the service in minutes.