Universitat Oberta de Catalunya

Practice

NETWORK ADMINISTRATION AND OPERATING SYSTEMS

Computer Engineering Degree

March 29, 2025

Contents

1 Description and objectives 2

2 Statement 2

2.1 Practical Development . . . 2

3 Environment Preparation 3

4 Exercise 1: Installing the OS and Creating Users (1 point) 5

4.1 Installing the OS on the Virtual Machine . . . 5

4.1.1 Evidence . . . 5

4.2 Creating Users . . . 5

4.2.1 Evidence . . . 5

5 Exercise 2: Service Stack (3 points) 6

5.1 Server *Postgres* . . . 6

5.1.1 Evidence . . . 7

5.2 Web Server *Bun* . . . 8

5.2.1 Evidence . . . 9

5.3 Web Server *Bun* with *Postgres* . . . 9

5.3.1 Evidence . . . 9

6 Exercise 3: *reverse-proxy* (3 points) 11

6.1 Proxy por HTTP . . . 11

6.1.1 Evidence . . . 12

6.2 Proxy over HTTPS . . . 12

6.2.1 Evidence . . . 12

7 Exercise 4: Kubernetes (1 point) 14

7.1 Evidence . . . 15

8 Delivery (background and shape: 2 points) 16

9 Evaluation 17

1 Description and objectives

The practice is an individual activity of a mandatory nature, which will be developed during the course of the semester and which must be completed when the final delivery is made according to the subject's calendar.

The objective of the activity is to put into practice various theoretical knowledge acquired during the course by reading the different modules and encourage the investigation of new technical concepts that complement what has been learned. In this way, it is intended that the student acquire some of the skills associated with network administration:

- Use several services or programs that show different ways of exchanging information.
- Use the most common administrative and network analysis tools.
- Implement packet filtering mechanisms.
- Experience the basic concepts of communication, including architectures, protocols and services.

In addition to the acquisition of more transversal skills.

- Ability to analyze and search for information that complements the knowledge obtained to solve complex problems
- Improvement in the ability to write and present technical reports

2 Statement

A company's development department requests an infrastructure to host its web service. The main requirement is that it must be accessible via a browser or terminal commands from the outside and uses a database that cannot be accessed from the internet. As a security measure, the server must only expose ports 80 (HTTP) and 443 (HTTPS), the latter being the final browsing protocol.

2.1 Practical Development

To offer this type of service, there are several architectures: based on services hosted on a single machine, distributed, virtualized, and/or container-based. To facilitate development, the infrastructure to be implemented is provided. In this case, the choice was made to design the service based on virtualization techniques and containers, as shown in figure 1.

Basically, the following systems will be implemented:

1. Virtual machine hosting the services [1].

2. Implementation of the database server [1, 2].
3. Implementation of the web server [1, 3].
4. Implementation of the *reverse-proxy* server [1, 3].
5. Design and configuration of the firewall.[4, 5]

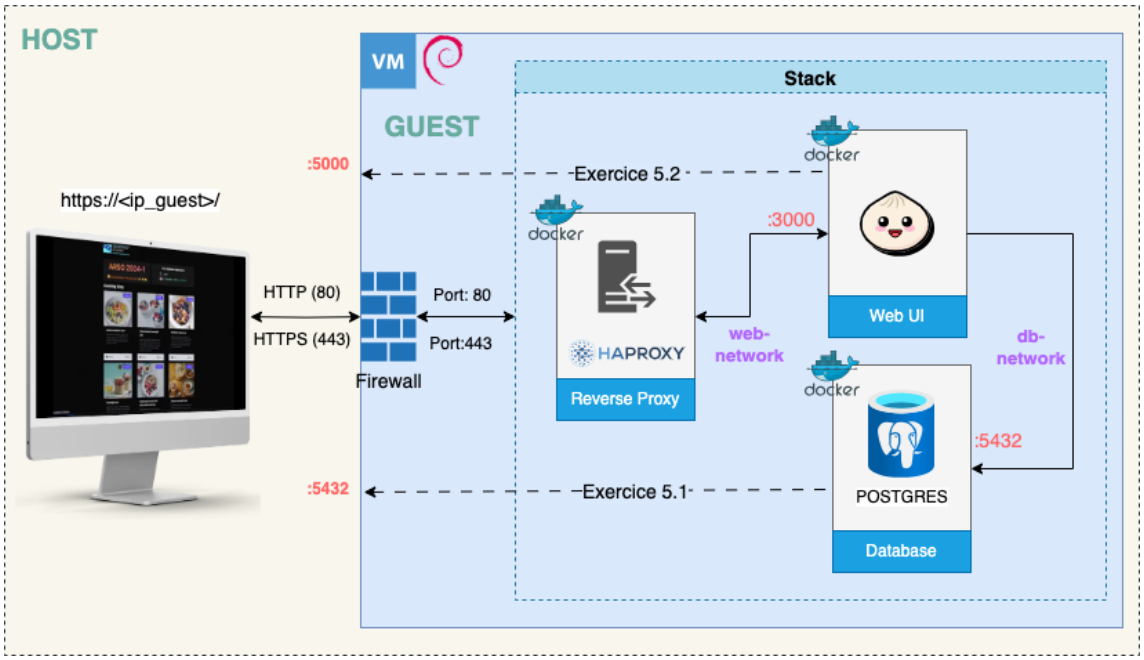

Figure 1: Diagram of the infrastructure to be implemented

To carry out this exercise, you will need to use a virtual machine to host the services, using the virtualization software you choose from the various available options (VirtualBox, VMware, Azure, etc.). Subsequently, the different systems mentioned above will be implemented in the form of exercises that will form part of the final infrastructure. The entire process will be documented in detail in a report, which will be the final document to be delivered (see 8 section for delivery method).

3 Environment Preparation

- Choose virtualization software and install it on your computer if necessary.
- Download the ISO image of the operating system to your computer. The operating system for this virtualized server will be:

  Debian 12.10 (https://cdimage.debian.org/debian-cd/current/amd64/iso-cd/debian-12.10.0-amd64-netinst.iso).

- Additionally, you will need the port scanning tool ***nmap*** (https://nmap.org) and the data transfer tool ***curl*** (https://curl.se) on both your host or external system and the virtual machine, if they are not included in the operating system itself.
- On the virtual machine, you must have the version control tool git (https://git-scm.com/) and the container tool ***Docker engine*** (https://docs.docker.com/engine/install/) with the plugin *docker-compose-plugin*

4 Exercise 1: Installing the OS and Creating Users (1 point)

4.1 Installing the OS on the Virtual Machine

Install the operating system ISO on the virtual machine, ensuring that the machine name is **ARSO20242** and that the **username** to be created is the same as your **campus username**. This user will be used to execute all commands. To avoid space issues, it is recommended that the disk where the installation will be performed has at least **15GB** of storage.

From now on, we will call your PC (Windows, macOS, etc.) *Host* and the virtualized system *Guest*.

4.1.1 Evidence

The following must be included in the report:

- Verification of the connection between the *Host* and the *Guest*

```
ping -c1 <ip_guest>
```

- Note both IP addresses in the header of the report pages: IP_HOST, IP_GUEST to clearly identify these references.

4.2 Creating Users

- Create ten users named: `userXX`, where XX is the user's consecutive number.
- Configure these users to have their passwords expire in one year.

4.2.1 Evidence

The following should be attached to the report:

- Show the instructions used to create the users. Also attach the command output:

```
sudo change -l userXX
```

  Replace XX with the user number that was created.

5 Exercise 2: Service Stack (3 points)

Configuring a Service Stack with *docker compose*

The idea of this section is to set up a series of services such as a database (*Postgres* [6]) and a web server (*Bun* [7]) so that they interact with each other. To make this possible, we need to use the *docker compose* tool, which allows us to define and share applications with various contents by defining them in a configuration file called `docker-compose.yml` [8]

- Access the virtualized *Guest* system
- Download the basic file structure that will serve as the basis for completing the exercises using the following instructions:

```
git clone https://github.com/jestebangr/prac20242-orig.git
```

5.1 Server *Postgres*

The Stack will be started by configuring the *Postgres* service.

The provided `docker-compose.yml` should look like the following:

```
services:
db:
image: postgres:17.4
container_name: dbhost
```

The official *Postgres* image from *dockerhub* will be used, labeled ***postgres:17.4***

Finish configuring the service, keeping in mind that:

- The ***Postgres*** port (**5432**) must be exposed for access from the *Guest* machine and for testing purposes.
- A **volume** called **dbdata** will need to be created so that the *Postgres* data is persistent and not deleted when we restart the container.
- When starting the service, make sure the following tasks are performed:
  - Enable the necessary variables to configure the server with a database called `uoc2024` and a user with your UOC alias.
  - Find an automatic way to load the `init.sql` file when starting the service. This file is located in the `dataset` directory and is responsible for loading data into the `uoc2024` database.

You could check official image documentation *Docker Postgres* [6] used to find how pass all information required to the container.

The fact of not putting sensitive information directly into the file will be valued `docker-compose.yml`. For this purpose, you can use environment variables [9].

Modifying files in the provided folder is not allowed `dataset`.

To make run the *Stack* you must be located on project folder *git* and execute command: (–build to rebuild the image, -d if you need to run on *background*) :

```
sudo docker compose up --build
```

If you want to stop the *Stack*, you can do so with the command:

```
sudo docker compose down
```

To see the output logs if you use the -d option, you can run the command:

```
sudo docker logs <container_name>
```

NOTE! When you create the container for the first time, the database will be created whether or not the data has been imported. Any data imports will be ignored on subsequent starts. To attempt to load the data again, you must delete the container and/or the created volume.

5.1.1 Evidence

- Contents of the `docker-compose.yml` file completed.

  ```
  cat docker-compose.yml
  ```

- Screenshot of the running containers:

  ```
  sudo docker ps
  ```

- Checking that port *Postgres* is listening and accessible:
  From the Guest machine:

  ```
  sudo netstat -a | grep 5432
  ```

  From the PC (either option, the second one for Windows Power Shell):

  ```
  nmap -p- --open -n <guest_ip>
  Test-NetConnection -Port <db_port> -Computer <guest_ip>
  ```

- Once the service has been successfully started, try and display access to the *Postgres* with any client. Use the user with the credentials configured in the container.

5.2 Web Server *Bun*

Once the *Postgres* is configured, create another service within the *Stack* that provides the web content you want to publish. In this case, it's a *Bun* server-based website, and therefore, the official *Bun* image from *Dockerhub*[7] tagged as ***oven/bun:latest*** will be used.

The `docker-compose.yml` working file should now look something like this:

```
services:
db:
image: postgres:17.4
container_name: dbhost
(...)
web:
build: bun-app
container_name: webhost
(...)
```

Complete the web service configuration using the `Dockerfile` file included in the `bun-app/` directory, keeping in mind that:

- The container must have the necessary files to start the web server with Python. These files can be found in the `app/` directory provided. Volumes are not allowed in this section.
- You should not modify any files in the `app/` folder.

Additionally, complete the web service configuration in the `docker-compose.yml` file, keeping in mind that:

- You must expose the web server's 3000 port to the external 5000 port to perform testing.

Once everything is configured correctly, you can start the *Stack* with the command:

```
sudo docker compose up --build
```

You can consult the official documentation for the *Docker* image *Bun* [7]

Also, use the reference to the `Dockerfile` [10]

5.2.1 Evidence

- Contents of the completed `docker-compose.yml` file.

```
cat docker-compose.yaml
```

- Contents of the completed `Dockerfile` file.

```
cat Dockerfile
```

- Screenshot of the running containers:

```
sudo docker ps
```

- Access the website (note the exposed port) from a *Guest* or *Host* browser. Attach a screenshot verifying the successful connection.

5.3 Web Server *Bun* with *Postgres*

Next, the ***db*** container will be linked to the ***web*** container so that the latter can obtain information from the database.

Aspects to keep in mind:

- The ***web*** container must be provided with the appropriate environment variables so that the server can use them to make the connection.
- You can use the *bun-app/app/src/app.ts* file to see which environment variables are required to connect to the database.
- Remember that no files in the *bun-app/app* directory can be modified.

5.3.1 Evidence

- Contents of the `docker-compose.yml` file completed.

```
cat docker-compose.yml
```

- Screenshot of the running containers:

```
sudo docker ps
```

- Checking the status of exposed ports: From the Guest machine:

```
sudo netstat -tulpn | grep LISTEN
```

From the PC (either option, the second one for Windows Power Shell):

```
nmap -p- --open -n <ip_guest>
Test-NetConnection -Port <db_port> -Computer <ip_guest>
```

- Test accessing the URL from a browser on the *Guest* or *Host*:

  http://<ip_guest>:5000/

  Attach a screenshot of what is displayed in the browser confirming the successful connection to the database.

6 Exercise 3: *reverse-proxy* (3 points)

In order to protect the identities of internal servers and as an additional defense against security attacks, it is recommended to install and configure a *reverse-proxy* server as shown in the diagram 1 (page 3) so that it displays the content of the web server from the exercise 5.3 (page 9). The following should be taken into account:

- The proxy server must be created using a Dockerized ***HAProxy*** server [11].
- The Docker container must be named: **proxyhost**
- It must accept HTTP and HTTPS requests through ports 80 and 443 respectively. The website must be accessible from the Host (PC) via the URL:

  https://<ip_guest>/

To make things easier, this will be done in two phases: first, using the HTTP protocol and then adding the secure HTTPS protocol.

6.1 Proxy por HTTP

Configure as needed the two files `Dockerfile` [11] and `haproxy.cfg` [12] located in the `reverse-proxy` directory.

Make the necessary changes to the `docker-compose.yml` to add the proxy service, keeping in mind that:

- The configuration file `haproxy.cfg` edited locally must be copied to the container at build time.
- For the *proxy* to be able to access the *web* container, they must be on the same Docker network [13], which will be named ***web-network***
- Make the necessary changes to additionally connect the *web* and *db* containers through another network called ***db-network***
- The correct hostnames and ports for connecting between nodes must be internal to the Docker network.

`docker-compose.yml` should have content like this:

```
services:
(...)
networks:
web-network:
name: web-network
```

6.1.1 Evidence

- Contents of the modified files from the terminal:

```
cat reverse-proxy/Dockerfile
cat reverse-proxy/haproxy.cfg
cat docker-compose.yml
```

- Checking that the *Guest* server has port 80 open and that they are accessible from *Host*.

  From the *Guest* machine:

```
sudo netstat -a | grep http
```

  From the PC (any of the options):

```
nmap -p- --open --min-rate=5000 -Pn -v -sS -n <ip_guest>
Test-NetConnection -Port 80 -Computer <ip_guest>
```

- Information about the containers in each network:

```
sudo docker network inspect -f '{{json .Containers}}'
 ↪  web-network | python3 -m json.tool
sudo docker network inspect -f '{{json .Containers}}'
 ↪  db-network | python3 -m json.tool
```

6.2 Proxy over HTTPS

Modify the *proxy* server configuration so that all HTTP requests are automatically forwarded to the secure HTTPS protocol.

Note: To use the HTTPS protocol, you must generate a self-signed certificate using the `openssl` command to obtain a .pem file for *HAProxy*. Include your UOC email address in the certificate's email address.

Modify the contents of the `haproxy.cfg` file so that it contains the 443 protocol configuration and the necessary redirection [14].

6.2.1 Evidence

- Contents from the terminal of the `Dockerfile` and `haproxy.cfg` files:

```
cat reverse-proxy/Dockerfile
cat reverse-proxy/haproxy.cfg
```

- Screenshot of the running containers:

```
sudo docker ps
```

- Checking that the *Guest* server has ports 80 and 443 open and that they are accessible from the *Host*.

  From the Guest machine:

```
sudo netstat -a | grep http
```

  From the PC (any of the options):

```
nmap -p- --open --min-rate=5000 -Pn -v -sS -n <ip_guest>
Test-NetConnection -Port <port> -Computer <ip_guest>
```

- Screenshot of the *Host* client browser with the web inspection console open, showing the 302 warning with the redirection or executing the command on `Host`:

```
curl -I http://<ip_guest>/
```

- Access the URL from a *Host* browser using http (port 80) and verify that it redirects to https (port 443): **https://<ip_guest>/** Attach a screenshot showing the URL and web content.
- Client browser screen *Host* with the **certificate details**

7 Exercise 4: Kubernetes (1 point)

The following section focuses on the use of containers in a highly orchestrated manner, known as Kubernetes or K8s. To this end, we will install a simplified but functional Kubernetes environment called minikube, along with a standalone example application.

- On the already installed virtual machine (minimum 2 CPUs and 4 GB of RAM), download the minikube environment and perform the installation. You can find more information at https://minikube.sigs.k8s.io/docs/start [15]
- Launch the minikube Kubernetes dashboard to view the deployed components via the web.

```
minikube dashboard
```

- We are now ready to deploy the cluster's first application. This is an example application that will help us analyze the syntax of K8s yaml files and learn how to use them. It is an example application of a 3-tier architecture, with a voting pod, databases for persistence, and another worker pod. The yaml code is in the K8S/example folder.

Figure 2: k8s example app

Now it's time to deploy the application. Using the provided files, deploy all the application components.

From the k8s/example directory

```
kubectl apply -f .
```

From here, we'll need to see the external IP address.

```
minikube service vote -url
minikube service result -url
```

7.1 Evidence

- List the nodes that make up the Kubernetes cluster, once installed. You can use the kubectl [16] tool.

```
kubectl get nodes
```

- Comment out the following sections of the *dashboard* that make up the Kubernetes cluster and its function with a maximum of 3 lines.
  - Cluster
  - Cluster role binding.
  - NameSpace
  - Network Policies.
  - Nodes.
  - Persistent Volumes.
  - Roles
  - Role Binding
  - Service Accounts.
- Identify the different application definition files in Kubernetes and comment in detail on the *pod* and *service* sections according to the 1 table.
- Take a screenshot of the voting website and the results website.

POD	Service
apiVersion: v1 kind: apiVersion: v1 kind: Pod metadata: name: result-app-pod labels: name: result-app-pod app: demo-voting-app spec: containers: - name: result-app image: codekloud/examplevotingapp_result:v1 ports:	# Results app service apiVersion: v1 kind: Service metadata: name: result-service labels: name: result-service app: demo-voting-app spec: type: NodePort ports: - port: 80 targetPort: 80 nodePort: 30005 selector: name: result-app-pod app: demo-voting-app

Table 1: POD and Service

8 Delivery (background and shape: 2 points)

The practice consists of the delivery of a report (a written document in PDF format), which contains the following:

- Cover and table of contents and figures
- One chapter for each exercise where each of the respective evidences is shown
  - Terminal screenshots must be clear and legible
  - Paragraphs of text that refer to commands and/or file content must be in a font that is proportional and different from the rest of the text.
- A final section with the conclusions drawn from the development of the practice.
- An annex with the content of the configuration files that have been created/modified.
- The practice must be delivered no later than the maximum delivery date defined in the subject calendar.

Note: To facilitate reading and correction, it is better to limit yourself to showing the evidence requested in each exercise and avoid installation screens and execution of intermediate commands typical of system configuration.

9 Evaluation

- The practice is a mandatory individual activity that constitutes 30% of the overall evaluation of the subject.
- Total or partial plagiarism detected in practice will result in a grade of 0.
- The mark of the practice will have to be at least 4 out of 10 to be able to pass the overall of the subject. A lower grade or not presenting the practice within the established deadlines will represent the failure of the overall subject.
- The content of the report that does not contain the necessary descriptions, explanations and reasoning will not be scored.
- The solution of the different sections is not unique, in the evaluation of the correctness of the given solution, aspects such as efficiency, elegance, and simplicity will be taken into account.

References

[1] Eduardo Marco Galindo and Javier Panadero Martínez. *Administració de servidors / Administración de servidores*. Ed. by UOC. PID_00275601. UOC, 2022.

[2] Manel Mendoza Flores, Miquel Colobran Huguet, and Javier Panadero Martínez. *Administració de les dades / Administración de los datos*. Ed. by UOC. PID_00275599. PID_00275599. UOC, 2022.

[3] Alberto José Mateos Bartolomé, Joan Ramon Esteban Grifoll, and Javier Panadero Martínez. *Administració dels serveis web /Administración de servicios web*. Ed. by UOC. PID_00275605. PID_00275605. UOC, 2022.

[4] Miguel Martín Mateo et al. *Administració de xarxa / Administración de redes*. PID_00275602. UOC, 2022.

[5] José Manuel Castillo Pedrosa and Javier Panadero Martínez. *Administració de la seguretat / Administración de la seguridad*. Ed. by UOC. PID_00275600. PID_00275600. UOC, 2022.

[6] Dockerhub. *Docker Potgres Documentation*. 2024. URL: https://hub.docker.com/_/postgres.

[7] Dockerhub. *Docker Bun Documentation*. 2025. URL: https://hub.docker.com/r/oven/bun.

[8] Docker Documentation. *Compose file version 3 reference*. Feb. 2022. URL: https://docs.docker.com/compose/compose-file/.

[9] Docker Documentation. *Environment variables in compose*. Feb. 2022. URL: https://docs.docker.com/compose/environment-variables/.

[10] Docker Documentation. *Dockerfile reference*. Feb. 2022. URL: https://docs.docker.com/engine/reference/builder/.

[11] DockerHub. *Docker HAProxy Documentation*. 2024. URL: https://hub.docker.com/_/haproxy/.

[12] HAProxy Site. *HAProxy Documentation - Configuration manual*. Feb. 2024. URL: https://docs.haproxy.org/2.9/configuration.html#2.7.

[13] Docker Network. *Networking in compose*. 2022. URL: https://docs.docker.com/compose/networking/.

[14] HAProxy Products Documentation. *HAProxy config tutorials*. Feb. 2024. URL: https://www.haproxy.com/documentation/haproxy-configuration-tutorials/.

[15] *Minikube start*. Feb. 2025. URL: https://minikube.sigs.k8s.io/docs/start.

[16] *Install and Set Up kubectl on Linux*. Feb. 2025. URL: https://kubernetes.io/docs/tasks/tools/install-kubectl-linux/.