NIST - PRACTICE 4

(Session 3)

DEPLOYMENT

During this final session, we will deploy services that are available on the Internet. To begin with, we can look at the wide range of services currently offered across numerous websites.

The largest repository of containers and services is Docker Hub:

https://hub.docker.com/

The logic behind this enormous diversity is clear. Every company and organization that develops system software aims to offer it in a simple, user-friendly way, minimizing the effort required for installation and use. By providing a Docker image, or even just a Dockerfile, they can distribute software with all its requirements already packaged and ready to run.

When we aim to deploy multi-component services, where different components depend on one another, the advantages become even greater, although the complexity also increases.

In this course, we do not cover the deployment of services on physically distributed machines. Kubernetes has been mentioned as one of the most established tools in this area, and indeed, most distributed application deployments are designed with Kubernetes in mind. However, this topic lies outside the scope of our course.

We will therefore focus on multi-component services that are practical to deploy on a single computer.

SESSION 3: PREFABRICATED SERVICES DEPLOYMENT

Let's practice deploying two services available on Docker Hub as examples:

1. Deploying WordPress.
2. Deploying Mongo Express.

The files required to deploy these services are included in the compressed file "**tsr_lab4-3_material.zip"**. When you extract this file, two folders will be created, one for WordPress and another for Mongo. The contents of these folders match the files described below and were obtained from the sources we will reference.

PART 1.- Deploying WordPress.

WordPress is a widely used platform for developing websites, especially popular among bloggers and small to medium-sized sites.

A typical WordPress deployment consists of an Apache server that hosts the WordPress templates and resources, along with a database. Therefore, this setup will include at least two containers: one running Apache and another running the database, with the web server depending on the database.

There are many pages, tutorials, and repositories that describe this common deployment. For our example, we will use one of the configurations referenced on the official Docker website:

https://docs.docker.com/reference/samples/wordpress/

Specifically, we chose this version:

https://github.com/docker/awesome-compose/tree/master/official-documentation-samples/wordpress/

1.1 The Docker compose file

From this page, we are specifically interested in the Docker Compose configuration, which includes everything needed to deploy WordPress on a machine running Docker:

docker-compose.yml

```
services:
  db:
    image: mariadb:10.6.4-focal
    # If you really want to use MySQL, uncomment the following line
    #image: mysql:8.0.27
    command: '--default-authentication-plugin=mysql_native_password'
    volumes:
      - db_data:/var/lib/mysql
    restart: always
    environment:
      - MYSQL_ROOT_PASSWORD=somewordpress
      - MYSQL_DATABASE=wordpress
      - MYSQL_USER=wordpress
      - MYSQL_PASSWORD=wordpress
    expose:
      - 3306
      - 33060
  wordpress:
    image: wordpress:latest
    volumes:
      - wp_data:/var/www/html
    ports:
      - 80:80
    restart: always
    environment:
      - WORDPRESS_DB_HOST=db
      - WORDPRESS_DB_USER=wordpress
      - WORDPRESS_DB_PASSWORD=wordpress
      - WORDPRESS_DB_NAME=wordpress
volumes:
  db_data:
  wp_data:
```

This file is included in the “wordpress” folder within the file you downloaded for this practice session.

By examining the docker compose file, we can see that all sections correspond to concepts we have already worked with in our CBW and CBWL deployments.

The "volumes" section deserves special attention. As we know, volumes allow us to link persistent storage on the host with storage inside the containers. Docker's volume system offers a variety of options and configurations. Among these possibilities, we are interested in two specific types that you should understand and review:

1. Volume as host directory: We used this option in the CBWL deployment. Be sure to review how it works.

2. Volumes with symbolic names: In this case, the volume is assigned a symbolic name, and Docker stores all the related data in its own directory, located at /var/lib/docker/volumes on the host. This is the option used in the WordPress deployment. Therefore, the two volumes mentioned, "db_data" and "wp_data", will be stored in this docker directory. Before deploying the service, examine the contents of the `/var/lib/docker/volumes/` folder. After the deployment, we will check the folder again to observe the changes. (Note: Access to this directory requires administrator permission. Use "sudo" before the "ls" command to view its contents.)

Questions:

1.- In the previous session, we used the environment variable $BROKER_HOST to inject the broker's IP address as a dependency. In this new deployment, identify an environment variable that serves a similar purpose and explain the relation.

2.- Before deploying, think about where our WordPress installation will be accessible (i.e., which URL we will use in our browser). Identify the keyword that indicates the port on which the service will be available.

3.- In this example deployment, the "latest" version of WordPress is used. Discuss whether using the "latest" tag is the most appropriate choice if we want to guarantee the reproducibility of the system.

1.2 Deployment of the service

Step 1.

Start deploying the service using “docker compose up”. You must run this command from the appropriate folder.

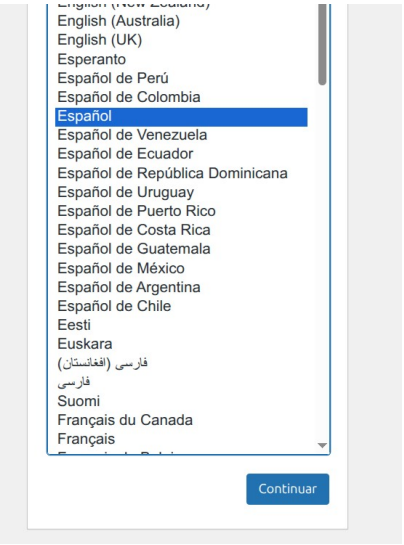
Wait until the deployment finishes. This process includes downloading the images and starting up two components; in practice, this is equivalent to downloading the images of two complete machines and booting them. Reflect on how quickly you can have everything up and running using this approach and compare it with the traditional alternative of installing all the required software on each machine and then starting it manually.

Step 2.

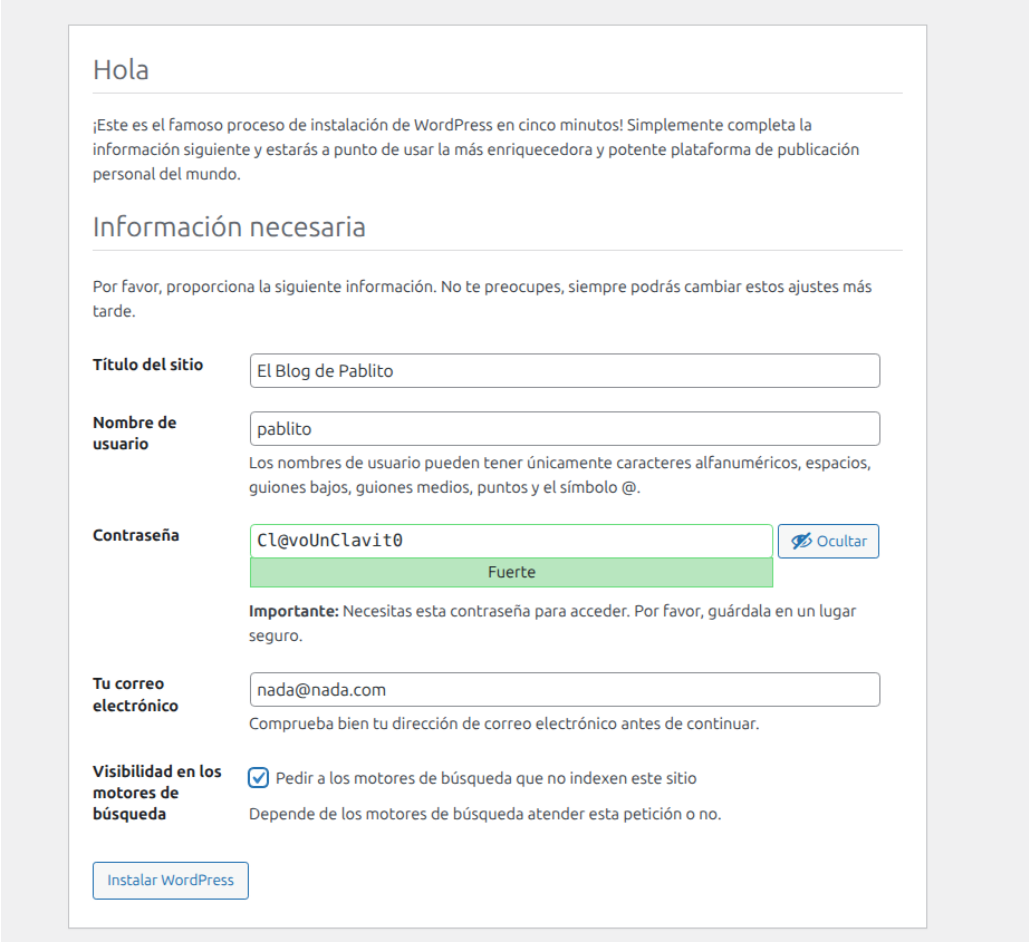
Once the startup is complete, verify that the deployment is working correctly. To do this, access your new WordPress installation, Directly from the host via the URL `http://localhost/`, or from another computer with access to the portal via `http://tsr-X`*`XX`*`.dsicv.upv.es/`.

The page you see should be the well-known WordPress 5-minute installation page. Choose the language, give the website a title, enter an email address (it can be fictitious), and set a password (the default password will work if you check the “remember password” option).

We start by specifying the language:



After selecting the language, proceed to fill in the basic information for the blog:



Then, after clicking “Install”, we can proceed to log in to our blog:

Step 3.

Once we have logged in, we can begin editing our new blog. Make a few changes, there's no need to spend much time on this. Modify elements such as the title to verify that the changes persist across consecutive launches of the deployment.

Switch between the URLs http://localhost and http://localhost/wp-admin to edit the site and see how it looks.

Once you have done this, stop the service using `docker-compose down`, and then restart it with `docker-compose up`.

Observe that the changes you made are still there. You can also check again the contents of the folder where Docker stores volumes with symbolic names: `/var/lib/docker/volumes/.` Remember that accessing this directory requires administrator permissions, so you will need to use "sudo".

Exercise:

Outline the steps necessary to manually deploy WordPress on one computer and MariaDB on another, so that WordPress acts as an external client of MariaDB. This deployment should also allow an external client to access the WordPress site.

Describe the required steps by writing them.

Optionally, you may verify that the deployment works. You can do this if you have access to two computers, for example, your own portal virtual machine and a classmate's virtual machine.

PART 2.- Deployment Mongo Express.

Mongo is a NoSQL database with a wide and active user community. Many aspects of this database are covered in Unit 6.

A MongoDB database can be deployed in multiple ways. Although deploying a full MongoDB cluster and performing fault-tolerance tests would be an interesting exercise, in this part we will take a more modest approach. We will deploy a single database instance together with a tool that allows us to verify the installation.

To verify that everything is working correctly, we will use mongo-express a web-based interface for accessing Mongo. This means that, once again, our deployment will consist of two components: the database itself and a web interface. Mongo-express is a web application built with NodeJS using the Express framework. Express is a NodeJS module designed to simplify the creation of web applications and services.

2.1. The Docker compose file

We will use the deployment proposed by Mongo on Docker Hub, which provides the official Mongo:

https://hub.docker.com/_/mongo

On this page, we can find the Mongo image along with several options for running and testing Mongo. Among these options, the section dedicated to Docker Compose is the one that interests us.

The Docker Compose configuration provided on this page is the following:

docker-compose.yml

```
services:
  mongo:
    image: mongo
    restart: always
    environment:
      MONGO_INITDB_ROOT_USERNAME: root
      MONGO_INITDB_ROOT_PASSWORD: example

  mongo-express:
    image: mongo-express
    restart: always
    ports:
      - 8081:8081
    environment:
      ME_CONFIG_MONGODB_URL: mongodb://root:example@mongo:27017/
      ME_CONFIG_BASICAUTH_ENABLED: true
      ME_CONFIG_BASICAUTH_USERNAME: mongoexpressuser
      ME_CONFIG_BASICAUTH_PASSWORD: mongoexpresspass
```

This same content is available in the “mongo” folder of the file you downloaded with the practice materials.

2.2. Deployment of the service

Step 1.

Start deploying the service using “docker compose up”. You must do this from the appropriate folder.

Wait for the deployment to finish.

Step 2.

Once the startup is complete, verify that the deployment is working correctly.

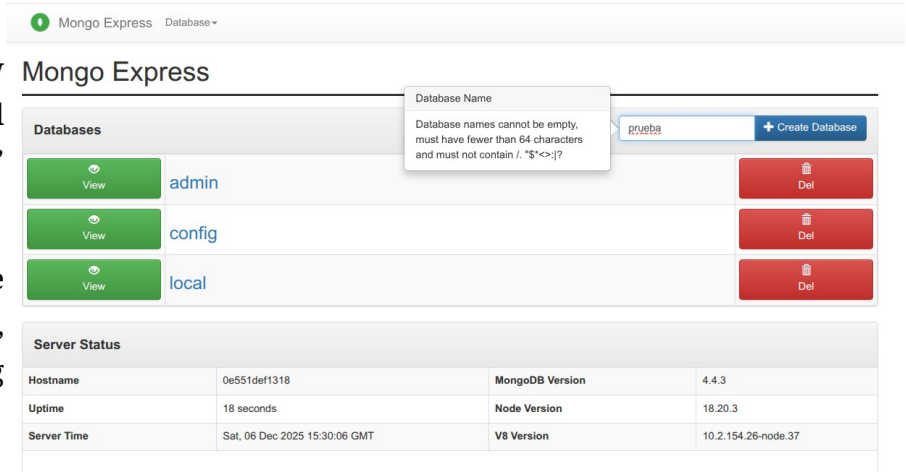
Check the Docker Compose file to see how to test the deployment. In that same file, you will also find the username and password required to log in to mongo-express.

Step 3.

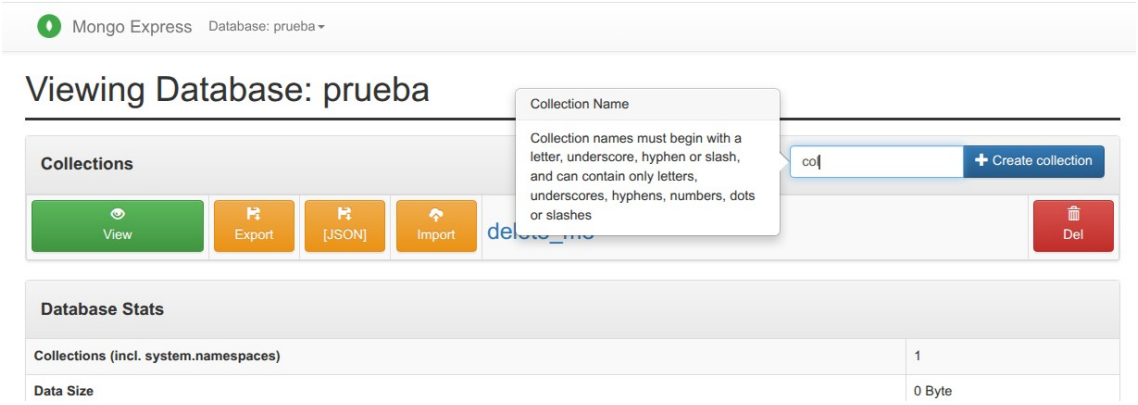
Create a test database, create a collection, and add a couple of ítems to that collection.

First, create a new database and name it “prueba” or “test”.

Once the database has been created, open it by clicking “View”.

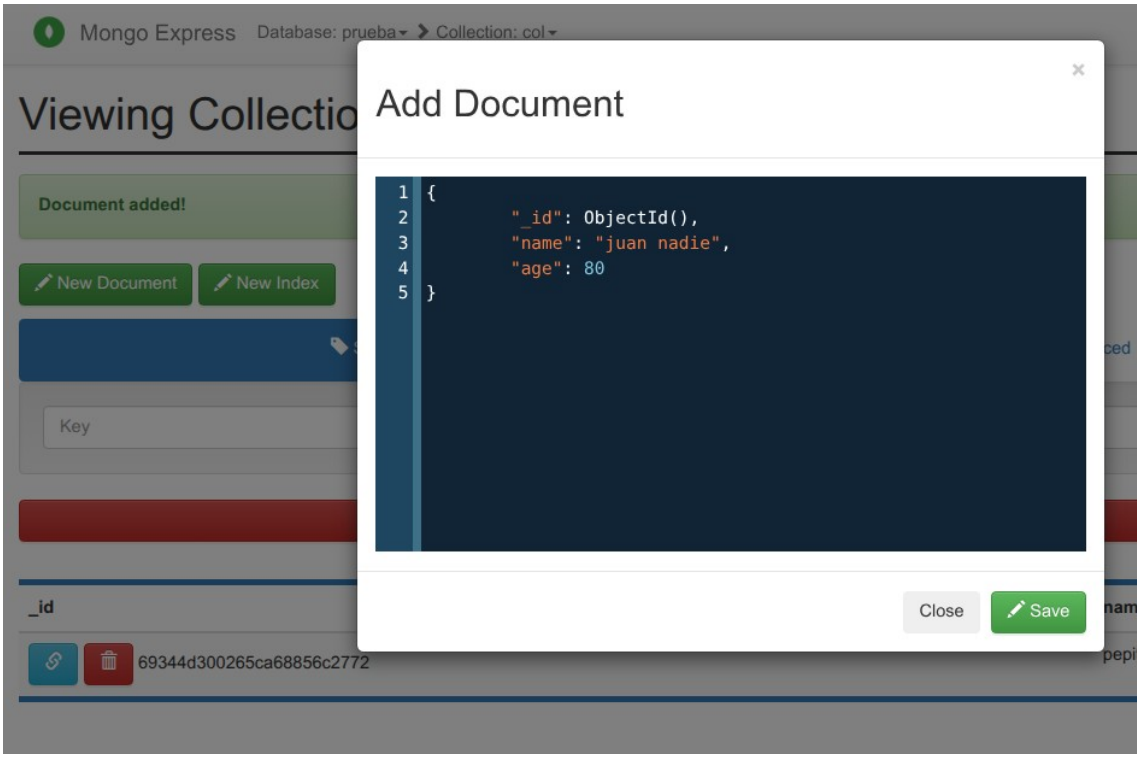


Next, create a collection, which in this example we will name “col”.



Open the collection by clicking “View,” and add a couple of objects by selecting “New Document”.

To add documents, you must create an object using proper JSON notation. Be careful with the syntax, your entry must be a valid JSON object, with attribute names in quotation marks and fields separated by commas. See the example:



With this, we will have confirmed that the combined deployment of mongo + mongo-express is functioning correctly.

Step 4. Stop the deployment

You can stop the deployment by pressing Ctrl-C.

From the same "mongo" folder where the docker-compose.yml file is located, you can remove the containers by running "`docker compose down`".

Exercise

If you redeploy the service, you will notice that the data you previously modified has been lost. This happens because the current deployment does not use volumes.

Since MongoDB stores its data inside the container in the "`/data/db` directory, you must create a volume so that any changes made to this directory inside the container are written to a directory on the host.

For example, create a directory called "`/tmp/mymongo`" with the following command:

```
sudo mkdir /tmp/mymongo
```

Then update the docker-compose.yml file to define a volume that maps "/data/db" inside the container to this new directory on the host.

After that, deploy the service again, repeat Step 3 to create a new database, and verify that the data now persists even after removing and redeploying the service.

Question

Explain the effect of the following commands or actions:

1. Pressing "Ctrl-C" in the terminal where "docker compose up" was launched.
2. `docker compose stop`
3. `docker compose down`
4. `docker compose down --volumes`