PROCESSES TO CREATE A DATABASE MANAGER IN CONTAINER MODE

Content

1. Objective

Make a document where the process of creating a database manager as a container is explained.

2. Scope

Document written to understand the application of database managers as a container, understanding its preconfiguration, understanding and execution in school classrooms, workshops and workplaces.

3. Process

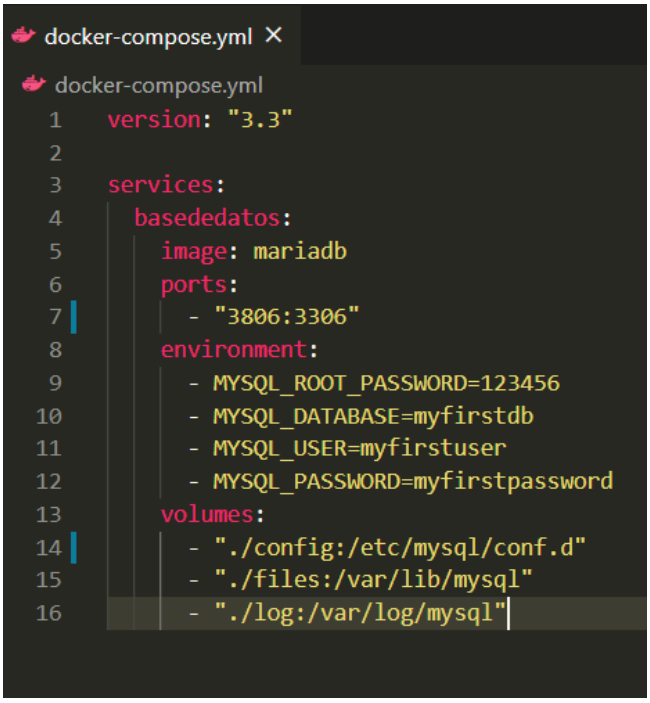
The docker-compose.yml file

A docker-compose.yml is a configuration file for docker-compose.

It allows to deploy, combine and configure multiple docker-containers at the same time. Docker's "rule of thumb" is to outsource each process to its own dockable container.

For example, a simple docker-compose database manager. you need to set a docker-container with mariadb.

The advantage of docker-compose is the easy configuration. you don't have to write a bunch of parameters to bash. you can predefine it in the docker-compose.yml.



Explanation of the lines in the docker-compose.yml file

Line 1. Docker version on which you are working.

Line 3. Service area or container area.

Line 4. Name of the container or service, here the name may vary, depending on the regulations or service that we are going to lift.

Line 5. The “image” is a file, made up of several layers, that is used to execute code in a Docker container.

Line 6. Zone to configure ports

Line 7. Declaration of the ports, in this example “3806” would be the public port and “3306” would be the private one.

Line 8. Area to declare environment variables, the variables to be declared and explained in the next lines are not always the same, it depends on the service or container is that we will declare them, in this case, as a database manager, we will use 4 main ones for this example.

Line 9. Declaration of the "MYSQL_ROOT_PASSWORD" which in simple words is our ROOT password of our manager.

Line 10. Declaration of the "MYSQL_DATABASE" that refers to the name of our database created initially, this can be optional.

Line 11. Declaration of the "MYSQL_USER" that refers to our user in order to access our database manager.

Line 12. Declaration of the "MYSQL_PASSWORD" that refers to our NO ROOT password to access our database manager.

Line 13. Area to declare our volumes, volumes are the preferred mechanism for preserving the data generated and used by Docker containers. While link mounts depend on the directory structure of the host machine, Docker fully manages volumes.

Line 14. Declaration of where the data to be managed by Docker will be stored "./config "and where to go to look for docker which will contain the previously mentioned folder"/etc/mysql/conf.d ".

Line 15. Declaration of where the data to be managed by Docker will be stored "./files "and where to go to find docker what will contain the previously mentioned folder"/var/lib/mysql ".

Line 16. Declaration of where the data to be managed by Docker will be stored "./log "and where to go to find docker what will contain the previously mentioned folder"/var/log/mysql ".

Manager configuration process

1. Find the image to use, in this case it would be mariadb, in this case we will need the my.cnf file to achieve a public connection to external devices to the computer running the manager.

2. Access inside the container to be able to extract the 50-server.cnf.

3. Modify in this same file the value of bin-adress from 127.0.0.1 to 0.0.0.0 in order to allow the use of external connections to the container.

4. Restart the container.

Process for configuring audit logs

In the same my.cnf file, comment out the lines in the "Loggin and replication" section, which would be: "general_log_file", "log_error", "slow_query_log_file", "log_bin", "expire_log_file".

4. Differences

First of all, we must bear in mind that, in the case of containers, the fact that they do not need a complete operating system, but rather reuse the underlying one, greatly reduces the load that the physical machine must bear, the storage space used and the time required to launch the applications. An operating system can occupy from just under 1GB for some Linux distributions with the bare minimum, up to more than 10GB in the case of a full Windows system. In addition, these operating systems require a minimum of reserved RAM to function, which can range from 1 to several GB, depending on our needs. Therefore, containers are much lighter than virtual machines.

When we define a virtual machine we must indicate in advance how many physical resources we must dedicate to it. For example, we can say that our VM will need 2 vCores (virtual processors), 4GB of RAM and a disk space of 100 GB. In the case of processors, it is possible to

share them between several virtual machines (but it is not advisable to pass or they will be fatal in performance), and the disk space can be made to only occupy what is really being used, so that it grows in depending on the needs and not always occupy as much as we had reserved. But in the case of memory and other elements (access to external drives or USB devices) the reservation is total. Therefore, although our application does not actually use the 4GB of RAM reserved, it does not matter: they cannot be used by other virtual machines or by anyone else. In the case of containers, this is not the case. In fact, we do not indicate what resources we are going to need, but rather it is Docker Engine, depending on the needs of each moment, in charge of assigning what is necessary for the containers to function properly.

This makes Docker runtimes much lighter, hardware utilization much better, as well as allowing many more containers to be erected than VMs on the same physical machine. While a VM can take a minute or more to start up and have our application available, a Docker container gets up and responds in a few seconds (or less, depending on the image). The occupied disk space is much lower with Docker as we do not need to install the full operating system.

On the other hand, Docker does not allow the use of containers / applications other than for the same operating system on a "host" operating system. That is, we cannot run a container with a Linux application on Windows or vice versa. Which can be an impediment on some occasions.

5. Conclusions

Docker is an excellent alternative to execute low, medium and high level projects in the industry. In addition, to be able to make advanced deployments of containerized applications, you have to go beyond Docker and use technologies such as Kubernetes, which allow us to orchestrate and control deployments with many moving parts. These technologies can be complex to learn and master, but once mastered you will be unstoppable.