Grégor JOUET
gregor.jouet@devinci.fr

1 Requirements

1. You must have a working Ubuntu (or other Linux based) system installed **PRIOR** to this class. Failure to do so will prevent you from doing this assignment.
2. It is recommended, although not required, to work on this assignment in advance and to use the actual assignment day to ask questions and engage in discussion on the topic.

Contents

2 Foreword

Containers have recently become the standard for shipping software in production. They are tiny, self-sufficient and mostly stateless environments containing the software to run as well as all the library dependencies necessary. The current standard container runtime is Docker [1] but other exist and rely on the same working principle: Instead of providing isolation through hardware emulation, containers use the kernel [4] to isolate the container environment from the rest of the system. They mainly rely on *cgroups* and *namespaces* [2]

3 Using Docker

Let's start by using Docker to master its usage before including it in a project.

3.1 Docker command line

1. After installing Docker using **THE DOCKER INSTALL SCRIPT [3]** and not the apt-get command. Check your installation with *docker ps*
2. Download and run the *ubuntu* container. Remove that container after use.
3. Launch a mysql container in the foreground and background.
4. With a mysql container in the background use the *docker exec* command to access it and look at the database. You can add a user for example.
5. Find the resources consumption of your containers in terms of CPU and memory.
6. Start a container and change a file in it. use the *docker export* and *import* commands to export its filesystem and import it again.
7. List all the images present on your machine.
8. Kill a container started in the background.

3.1.1 Building a container

1. In a new folder make a Dockerfile file and use it to create a container based on ubuntu with python3 installed.
2. Build that container and push it on the Dockerhub repository.
3. Change the default startup behavior of your container and have it execute a python script on startup (the script can be a hello world or anything else).
4. Push that container on the dockerhub repository and have a friend download and execute it on their machine.

3.2 Docker-compose

1. Write a simple docker-compose.yml file to launch a mysql server with a port redirection.
2. Write a docker-compose file to launch a node-export container with a grafana container. Visualize the graphs on the web interface.
3. Try out both the *image* and *build* directives in the docker-compose file.

4 Exercices

Let's use Docker in an example project to look at its integration pipeline

4.1 API in container

1. Use the rest API from the previous lesson and put in a docker image.
2. If necessary, change this API to connect to a mysql database.
3. Provide the connection information via environment variables to your API.
4. Use a docker-compose file to start your api with the appropriate variable values.

4.1.1 Volumes

1. Use a volume to mount a file in a container. Try to mount a directory as well.
2. Start a web server in docker that serves a folder in a volume mounted in it.
3. Use a volume to save logs of an application (like your API)

5 To Go further

This section is for more advanced applications and to dive deep into the understanding of Docker.

1. Start programs in different PID namespaces.
2. Change a process cgroups to put a limitation on the ram used.
3. Write a C program to start a program in an isolated namespace.
4. Use the chroot(2) call to start your program in a dedicated folder.

References

[1] *Docker Website*. URL: https://www.docker.com/.

[2] Quân Huỳnh. *Deep into Container - Linux Namespaces and Cgroups: What are containers made from?* 2022. URL: https://faun.pub/kubernetes-story-linux-namespaces-and-cgroups-what-are-containers-made-from-d544ac9bd622.

[3] *The Docker install script to use*. URL: https://github.com/docker/docker-install.

[4] *The Linux Kernel Archives*. URL: https://www.kernel.org/.