Containers

As described in lecture 4 containers are a staple of packaging applications in a self-contained fashion. Linux containers contain everything a Linux application needs to run on a Linux kernel in an isolated fashion. This exercise will run you through the basics of packaging an application in a container and running it on a Linux server.

This guide is an abridged version of “Getting started with Docker for the Inquisitive Mind&rdqou by Janos Pasztor, one of the lecturers on this course. We highly recommend reading the original version for more detail.

Installing Docker

One of the most popular small-scale container runtimes, and indeed the first to popularize build recipes is Docker. Docker can be installed on Linux as well as Windows and MacOS, but the latter two have several issues a user needs to work around so we recommend using an Ubuntu Linux 20.04 virtual machine for the purposes of this exercise.

To install Docker you can either follow the installation guide or use the convenience script to do so:

```
curl -fsSL https://get.docker.com -o get-docker.sh
sudo sh get-docker.sh
```

Please note that this script is not recommended for production use, but will be accepted in a solution to the project work.

Creating a basic Dockerfile

In order to package an application you can create a `Dockerfile` containing the instructions to install the dependencies and package an application. This file consists of the following format:

```
COMMAND PARAMETERS
```

The first command is the `FROM` command. This command specifies which base image to use, for example:

```
FROM ubuntu:20.04
```

As mentioned before, containers are not virtual machines. This command only pulls in the userland piece of an Ubuntu system. It can be used to install libraries and packages required for an application to run.

As this file has now been created we can build our container image as follows:

```
docker build -t myimagename .
```

> **Warning**
>
> The dot at the end is important! It tells Docker to look for the Dockerfile in the current directory.

Now that the image is built it can be launched:

```
docker run -ti myimagename
```

> **Tip**
>
> The `-ti` parameter is only required for interactive sessions. For non-interactive sessions it is not required.

Inside the container you can do operations just as you normally would. For example, you could run `apt update`, then `apt install mc`, and finally `mc` to launch the midnight commander. Once you are finished with the interactive session you can use the `exit` command or `Ctrl D` to exit.

However, as discussed in lecture 4 the main benefit containers bring to the table is reusability. If you treat a container like you would treat a virtual machine, updating, installing things manually, you lose the main benefit of using containers.

Let's extend our `Dockerfile` to install the `nginx` webserver:

```
FROM ubuntu:20.04

ENV DEBIAN_FRONTEND=noninteractive

RUN apt update
RUN apt install nginx
```

There are two new commands here: the `ENV` and the `RUN` command. The `ENV` command instructs Docker to set an environment variable. The `DEBIAN_FRONTEND=noninteractive` environment variable instructs the `apt` and `dpkg` packaging utilities in Ubuntu to run in non-interactive mode and not ask questions. Instead, these commands will use default values. The `RUN` command simply runs the specified command.

If you run `docker build .` command now the build will fail: