Technische Universität München

Cloud Computing Course

WS - 2017

Exercise - 2

“Docker Installation & Configuration”

Authors:
Prof. M. Gerndt,
Prof. S. Benedict,
Anshul Jindal

Table of Contents

I. Introduction

1.1 Purpose of this document

This document provides guidance and material to assist the relevant person, in understanding the Docker configuration and deployment for an application on the cloud. This includes following:

- Installation of Docker
- Creation of an account on docker hub
- Adding your application to a container and creating image
- Push image to docker hub
- Pull, run and deploy the application

1.2 Prerequisites

Following requirements are mandatory:

- Account on LRZ [1]

Following requirements are recommended (not mandatory):

- Basic knowledge of Linux console commands.

1.3 Background

Docker is an open-source project that automates the deployment of applications inside software containers [2]. Docker provides an additional layer of abstraction and automation of operating-system-level virtualization on Windows and Linux. Docker uses the resource isolation features of the Linux kernel such as cgroups and kernel namespaces, and a union-capable file system such as OverlayFS and others to allow independent "containers" to run within a single Linux instance, avoiding the overhead of starting and maintaining virtual machines.

Docker implements a high-level API to provide lightweight containers that run processes in isolation. Building on top of facilities provided by the Linux kernel (primarily cgroups and namespaces), a Docker container, unlike a virtual machine, does not require or include a separate operating system. Instead, it relies on the kernel's functionality and uses resource isolation (CPU, memory, block I/O, network, etc.) and separate namespaces to isolate the application's view of the operating system. Because Docker containers are so lightweight, a single server or virtual machine can run several containers simultaneously. A 2016 analysis found that a typical Docker use case involves running five containers per host, but that many organizations run 10 or more.

By using containers, resources can be isolated, services restricted, and processes provisioned to have an almost completely private view of the operating system with their own process ID space, file system structure, and network interfaces. Multiple containers share the same kernel, but each container can be constrained to only use a defined amount of resources such as CPU, memory, and I/O.

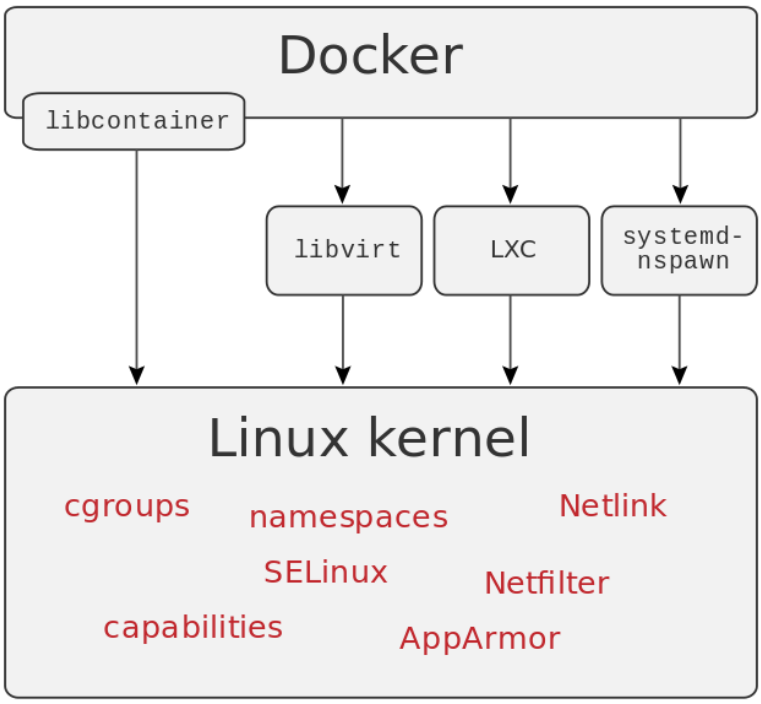

Figure 1: Docker Architecture [2]

Using Docker to create and manage containers may simplify the creation of highly distributed systems by allowing multiple applications, worker tasks and other processes to run autonomously on a single physical machine or across multiple virtual machines. This allows the deployment of nodes to be performed as the resources become available or when more nodes are needed, allowing a platform as a service (PaaS)-style of deployment and scaling for systems like Apache Cassandra, MongoDB etc. Docker also simplifies the creation and operation of task or workload queues and other distributed systems.

1.3.1 Fundamental Docker Concepts

Now that we've got the big picture in place, let's go through the fundamental parts of Docker piece by piece:

1.3.1.1 Docker Engine[1]

Docker engine is the layer on which Docker runs. It's a lightweight runtime and tooling that manages containers, images, builds, and more. It runs natively on Linux systems and is made up of:

[1] `docker.io` used to be the domain for the Docker Project (now it redirects to `docker.com`), and since there's already a docker package, Ubuntu package maintainers chose `docker.io` as a package name. Since Docker maintains its own Ubuntu package, they chose to name it as `docker-engine` to avoid naming confusion

1. A Docker Daemon that runs in the host computer.
2. A Docker Client that then communicates with the Docker Daemon to execute commands.
3. A REST API for interacting with the Docker Daemon remotely.

1.3.1.2 Docker Daemon

The Docker daemon is what executes commands sent to the Docker Client—like building, running, and distributing your containers. The Docker Daemon runs on the host machine, but as a user, you never communicate directly with the Daemon. The Docker Client can run on the host machine as well, but it's not required to. It can run on a different machine and communicate with the Docker Daemon that's running on the host machine.

1.3.1.3 Docker Client

The Docker Client is what you, as the end-user of Docker, communicate with. Think of it as the UI for Docker.

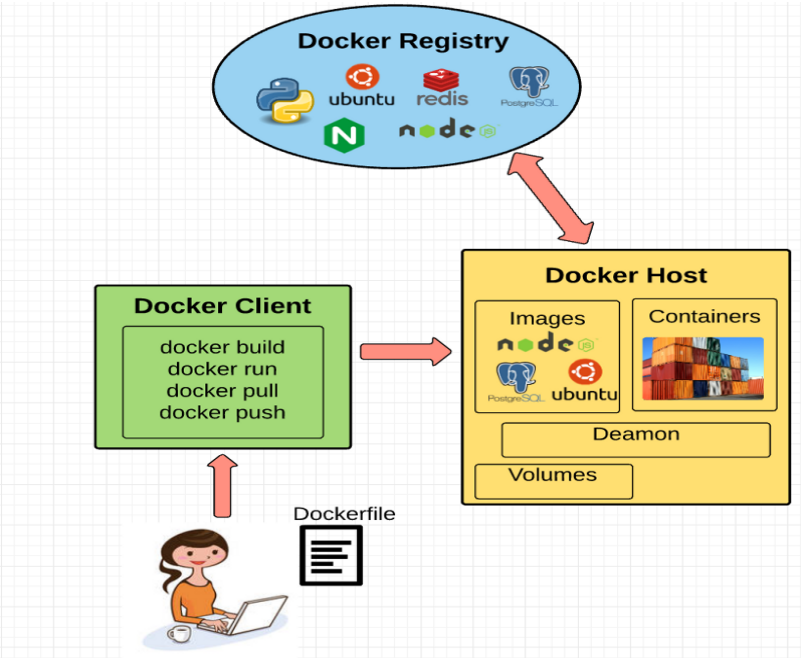

Figure 2: Docker Sub-Parts Interaction [3]

1.3.1.4 Dockerfile

A Dockerfile is where you write the instructions to build a Docker image. These instructions can be:

- **RUN apt-get y install some-package**: to install a software package
- **EXPOSE 8000:** to expose a port
- **ENV ANT_HOME /usr/local/apache-ant** to pass an environment variable

and so forth. Once you've got your Dockerfile set up, you can use the **docker build** command to build an image from it.

1.3.2 Docker Hub

Docker Hub is a cloud-based registry service which allows you to link to code repositories, build your images and test them, stores manually pushed images, and links to Docker Cloud so you can deploy images to your hosts. It provides a centralized resource for container image discovery, distribution, and change management, user and team collaboration, and workflow automation throughout the development pipeline [4]

II. Exercise Steps

2.1 Installation of Docker [5]

2.1.1 Linux – Ubuntu 16.04

1. Before you install Docker for the first time on a new host machine, you need to set up the Docker repository. Afterward, you can install and update Docker from the repository.
    - Install packages to allow `apt` to use a repository over HTTPS:

```
sudo apt-get install \
  apt-transport-https \
  ca-certificates \
  curl \
  software-properties-common
```

    - Add Docker's official GPG (GNU Privacy Guard) key[2]:

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

      Verify that the key fingerprint is **9DC8 5822 9FC7 DD38 854A E2D8 8D81 803C 0EBF CD88**

```
sudo apt-key fingerprint 0EBFCD88
```

```
pub   4096R/0EBFCD88 2017-02-22
      Key fingerprint = 9DC8 5822 9FC7 DD38 854A  E2D8 8D81 803C 0EBF CD88
uid                  Docker Release (CE deb) <docker@docker.com>
sub   4096R/F273FCD8 2017-02-22
```

    - Use the following command to set up the stable repository.

```
sudo add-apt-repository \
  "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
  $(lsb_release -cs) \
  stable"
```

      Note:
      To add the edge repository[3], add edge after stable on the last line of the command.
        - Stable builds are released once per quarter.
        - Edge builds are released once per month
2. Now the repository is setup, we can install the Docker. First update the `apt` package index.

[2] The key is used to verify the installation package. Packages which have been authenticated using these keys will be considered trusted.
[3] More recent versions are provided here.

```
$ sudo apt-get update
```

3. Install the latest version[4] of Docker by using this command.

```
$ sudo apt-get install docker-ce
```

4. After the installation is complete you can verify the installation by running this command.

```
$ sudo docker run hello-world
```

This command downloads a test image and runs it in a container. When the container runs, it prints an informational message and exits.

```
root@vm-141-40-254-118:~# sudo docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://cloud.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/engine/userguide/
```

Figure 3: Docker: Hello World

Now the Docker is installed. You can try additional commands from here Docker Commands.

2.2 Information about Images and Containers

Docker Engine provides the core Docker technology that enables images and containers. An *image* is a filesystem and parameters to use at runtime. It doesn't have state and never changes. A *container* is a running instance of an image. As the last step in your installation, you ran the docker run hello-world command. The command you ran had three parts:

[4] Docker community edition

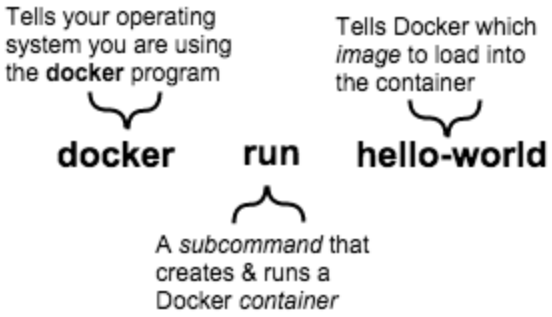

Figure 4: Docker Run Command

When you ran the command, Docker Engine:

- checked to see if you had the hello-world software image
- downloaded the image from the Docker Hub (more about the hub later)
- loaded the image into the container and **"ran"** it

Depending on how it was built, an image might run a simple, single command and then exit. This is what hello-world did. A Docker image, though, is capable of much more. An image can start software as complex as a database, wait for you (or someone else) to add data, store the data for later use, and then wait for the next person.

Here the "hello-world" software image is built by Docker and shared with anyone. Docker Engine lets people (or companies) create and share software through Docker images. Using Docker Engine, you don't have to worry about whether your computer can run the software in a Docker image — a Docker container can always run it.

2.3 Account Creation on Docker Hub

I. In your web browser, go to the Docker Hub signup page.

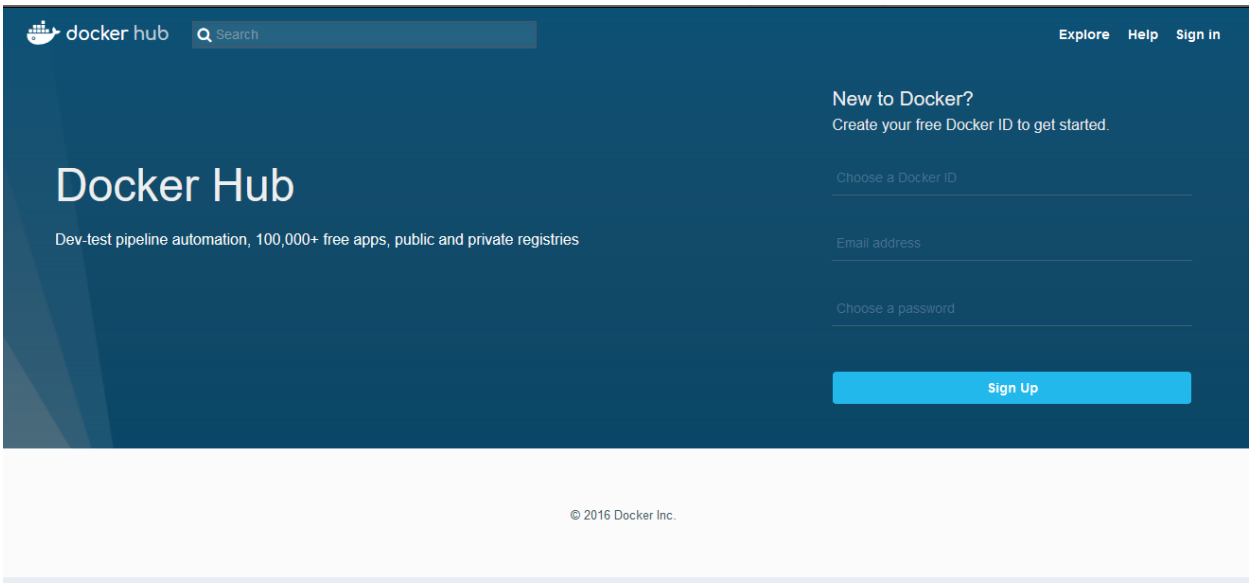

Figure 5: Docker Hub

II. Create a new account on it, by entering your dockerID, Email and Password.
III. After successful creation of account, you will get to your profile page where you can create and see all your repositories.
IV. You can create a repository by clicking on Create Repository, where you must provide the repository name.

2.4 Building an own Docker Image

Above you have seen when we ran the docker run hello-world command a prebuilt image was used. Here we will learn to build our own image of the client application.

I. Go into the root directory of your project, where you have kept all the files of the client application.
II. Create a new file with the name "**Dockerfile**[5]**"**, Docker engine will use this file to build the image.
III. Open the file and start writing the following commands in the file:
    a. FROM node:boron
    The FROM keyword tells Docker which image your image is based on. Here the node:boron is the latest Linux image with npm and node installed into it.
    b. RUN mkdir -p /usr/src/"name_of_your_application"
    WORKDIR /usr/src/"name_of_your_application"
    Here we are creating a directory which will contain our application inside the container and set the directory as the working directory.
    c. COPY package.json /usr/src/"name_of_your_application"
    RUN npm install

[5] Docker can build images automatically by reading the instructions from a Dockerfile. A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image

The next task is to install all the dependencies of our application into the container. As you already know all dependencies are inside the package.json file. So first copy that file into your application folder and then install them by running "npm install"

d. COPY . /usr/src/"name_of_your_application"
   This command will bundle up your source files into your application directory.
e. EXPOSE "port_number"
   This command exposes the port number of your application for outside world to access it.
f. CMD ["node", "clientServer.js"]
   This tells the image the final command to run after its environment is set up

Save your file, your docker file is ready.

IV. The next task is to add the .dockerignore file. This file tells the docker engine which files/directories to ignore while building the image. Create a new file with the name ".dockerignore" inside the project directory. Open it up and add the following in different lines
   a. node_modules
      To ignore all the npm_modules installed in your project directory.
   b. npm-debug.log
      To ignore all the debug logs.

V. Now we are all set to build our docker image. Inside the project directory run the following command

```
$ docker build -t "image_name" .
```

In the end, you will see it to be successfully built.

VI. Now to check your local images you can run the command

```
$ docker images
```

You will see your image in the list.

VII. Now your image is built, it's time to run it. For running the image type the following command

```
$ docker run -p ExternalPORT-NUMBER:InternalDockerPORT-NUMBER "image _name"
```

VIII. After this command your application will start in the docker container. You can view all the running containers by typing the command

```
$ docker ps
```

2.5 Tag, Push and Pull your Image

As you already have built the image and created the docker hub account, so now it's the time to tag your image and push it to your docker hub account. Tags are used to identify different versions.

2.5.1 Tag and Push

I. Open the terminal and run the $ docker images command.
II. Find the image ID for the "image_name" image, in the third column.
III. Tag the "image_name" image using the docker tag command and **the image ID**. The command you have to type looks like this:

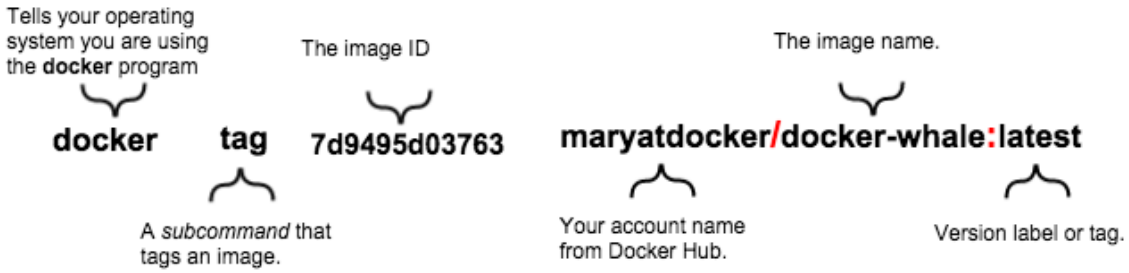

Figure 6: Docker tag [6]

Make sure to use your own Docker Hub account name and the name of your image.

IV. Run docker images again to verify that the "image_name" image has been tagged. The same image ID now exists in two different repositories.

V. Before you can push the image to Docker Hub, you need to log in, using the docker login command. The command doesn't take any parameters, but prompts you for the username and password, as below:

```
$ docker login

Username: *****
Password: *****
Login Succeeded
```

VI. Push your tagged image to Docker Hub, using the docker push command.

```
$ docker push "your_account_id"/"your_application_image_name"
```

VII. Go back to the Docker Hub website to see the newly-pushed image in your account.

2.5.2 Pull the Image

You have your application image pushed into docker hub, now you can access it from any Docker host using docker pull. First, though, you need to remove the local copy. Otherwise, docker pull will not have any work to do, because it will see that you already have the latest version of the image locally. Open the terminal and run the following commands:

I. Use docker images to list the images you have locally.

II. Use the docker rmi command to remove the images. You can refer to an image by its ID or its name. Since they share an ID, if you wanted to keep one of them, you'd need to refer to the other one by name. For this example, use the ID to remove both.

```
$ docker rmi -f "your_application_image_id"
```

III. When you use docker run it automatically downloads (pulls) images that don't yet exist locally, creates a container, and starts it. So after removing the image run the following command

$ docker run yourusername/"your_application_image_name"

Since the image is no longer available on your local system, Docker pull and run it.

2.5.3 Application Deployment

Now you have learned how to run a node.js application in the docker container. So, if you want to deploy this application on a VM, instead of running on the VM directly you should run it inside the docker container. This will deploy your application on the VM inside the docker container and make the management and deployment very easy.

2.6 Tasks to be completed

As part of the exercise2 complete the following tasks:

1. Add a service in your application with the name
   a. **/exercise2 :** Send a message "group 'GroupNumber' application deployed using docker".
2. Create the docker file of your application and then build the image.
3. Create your docker hub account and add your application image to it by following the procedure stated above. Name of your application image should be as: cloudcomputinggroup'GroupNumber'
4. Start a VM and pull this application image on it and run it inside the docker container (you should pull and run the image otherwise exercise would not be able to pass).
5. Enable docker remote API:
   a. Edit the file /lib/systemd/system/docker.service
   b. Modify the line that starts with ExecStart to look like this
      ExecStart=/usr/bin/docker daemon -H fd:// -H tcp://0.0.0.0:4243
      Where the addition is "-H tcp://0.0.0.0:4243"
   c. Save the modified file
   d. Run systemctl daemon-reload
   e. Run sudo service docker restart
   f. Test that the Docker API is indeed accessible:
      curl http://localhost:4243/version
   g. Enable port **4243** on your VM so that docker API can be accessed from outside network using your VM IP.

* Replace GroupNumber with your group number in above tasks.

III. Result Delivery

You already know how this works ☺. To submit your application results you need to follow this:

1. Open the cloud Class server url
2. Login with your username and password.
3. After logging in, you will find the button for exercise2
4. Click on it and a form will come up where you must provide your VM ip on which your application is running and the **dockerhub** image path name.

   Example:

   10.0.23.1

   dockerHubuserId/myImage
5. Then click submit.

Important points to Note:

1. Make sure your VM and your application is running after following all the steps mentioned in this manual.
2. We will grade you based upon the number of exercises completed by you.
3. You will get to see, what your application has submitted to the server.
4. You can submit as many times until the deadline of exercise.
5. Multiple submission will overwrite the previous results.

2 References

[1] "LRZ," [Online]. Available: https://www.lrz.de/english/.

[2] "DockerWiki," Docker, [Online]. Available: https://en.wikipedia.org/wiki/Docker_(software).

[3] "DockerParts," [Online]. Available: (https://cdn-images-1.medium.com/max/800/1*K7p9dzD9zHuKEMgAcbSLPQ.png).

[4] "DockerHub," [Online]. Available: https://hub.docker.com/.

[5] "Docker Installation," Docker, [Online]. Available: https://docs.docker.com/engine/installation/.

[6] "Docker tag," Docker, [Online]. Available: https://docs.docker.com/edge/engine/reference/commandline/tag/.