Docker:

- Docker is a computer program that performs operating-system- level virtualization also known as containerization.
- It was first released in 2013 and is developed by Docker, Inc.
- Docker is used to run software packages called "containers".
- In a typical example use case, one container runs a web server and web application, while a second container runs a database server that is used by the web application.
- Containers are isolated from each other and bundle their own tools, libraries and configuration files; they can communicate with each other through defined channels.
- All containers are run by a single operating system kernel and are thus more lightweight than virtual machines.
- Containers are created from "images" that specify their precise contents.
- Images are often created by combining and modifying standard images downloaded from repositories.

https://docs.docker.com

Hypervisor

- Hypervisor is a piece of software, firmware, or hardware that VMs run on top of (like our Ubuntu).
- The hypervisors themselves run on physical computers, referred to as the *“host machine”*.
- The host machine provides the VMs with resources, including RAM and CPU.
- These resources are divided between VMs and can be distributed as you see fit.
- So if one VM is running a more resource heavy application, you might allocate more resources to that one than the other VMs running on the same host machine.
- https://en.wikipedia.org/wiki/Hypervisor

Dockerfile

- A Dockerfile is where you write the instructions to build a Docker image.
- Once you’ve got your Dockerfile set up, you can use the **docker build** command to build an image from it.
- It supports a simple set of commands that you need to use in your Dockerfile.
- There are several commands supported like FROM, CMD, VOLUME, ENV and more
- A Dockerfile is a text file that Docker reads in from top to bottom.
- You can relate it to cooking. In cooking you have recipes. A recipe lets you know all of the steps you must take in order to produce whatever you’re trying to cook.

```
# Use an official Python runtime as a parent image
FROM python:2.7-slim

# Set the working directory to /app
WORKDIR /app

# Copy the current directory contents into the container at /app
COPY . /app
```

```
# Make port 80 available to the world outside this container
EXPOSE 80

# Define environment variable
ENV NAME World

# Run app.py when the container launches
CMD ["python", "app.py"]
```

Images

- Images are read-only templates that you build from a set of instructions written in your Dockerfile.
- Images define both what you want your packaged application and its dependencies to look like and what processes to run when it's launched.
- The Docker image is built using a Dockerfile.
- Each instruction in the Dockerfile adds a new "layer" to the image, with layers representing a portion of the images file system that either adds to or replaces the layer below it.
- Layers are key to Docker's lightweight yet powerful structure.

  https://docs.docker.com/engine/reference/commandline/images

Containers

- Unlike a VM which provides hardware virtualization, a container provides operating-system-level virtualization.
- For all intent and purposes, containers look like a VM.
- For example, they have private space for processing, can execute

commands as root, have a private network interface and IP address, allow custom routes and iptable rules, can mount file systems, etc.
- A Docker container, wraps an application's software into an invisible box with everything the application needs to run.
- That includes the operating system, application code, runtime, system tools, system libraries, etc.
- Docker containers are built off Docker images.
- Since images are read-only, Docker adds a read-write file system over the read-only file system of the image to create a container.
- https://www.docker.com/resources/what-container

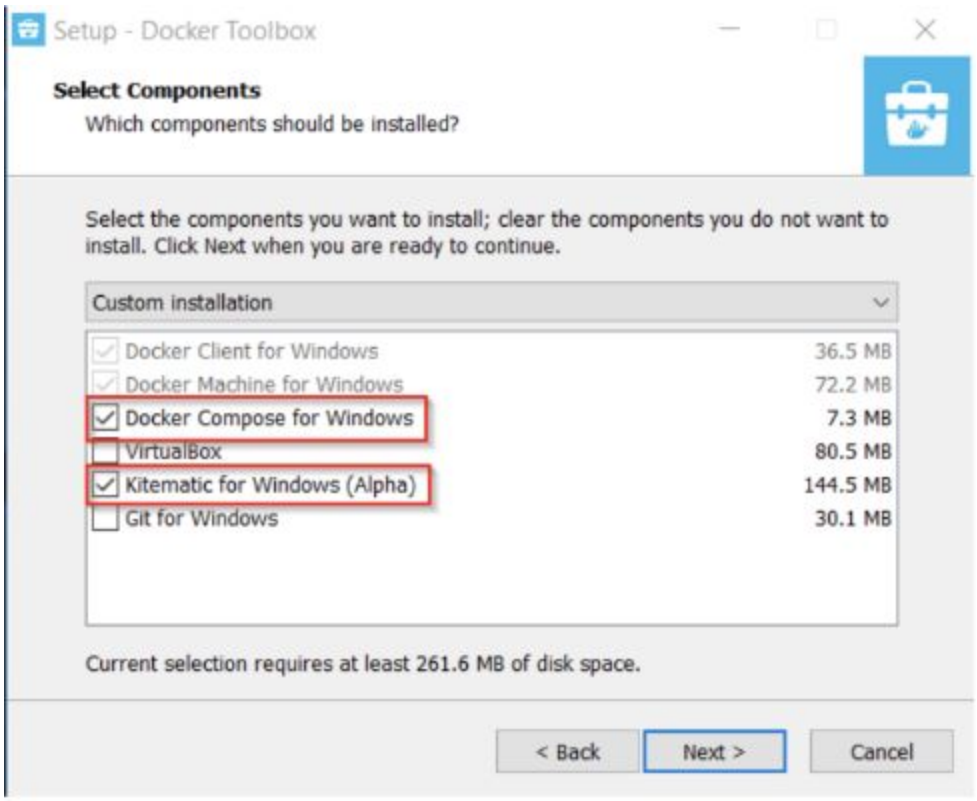
Setup

1. Download Docker toolbox from:

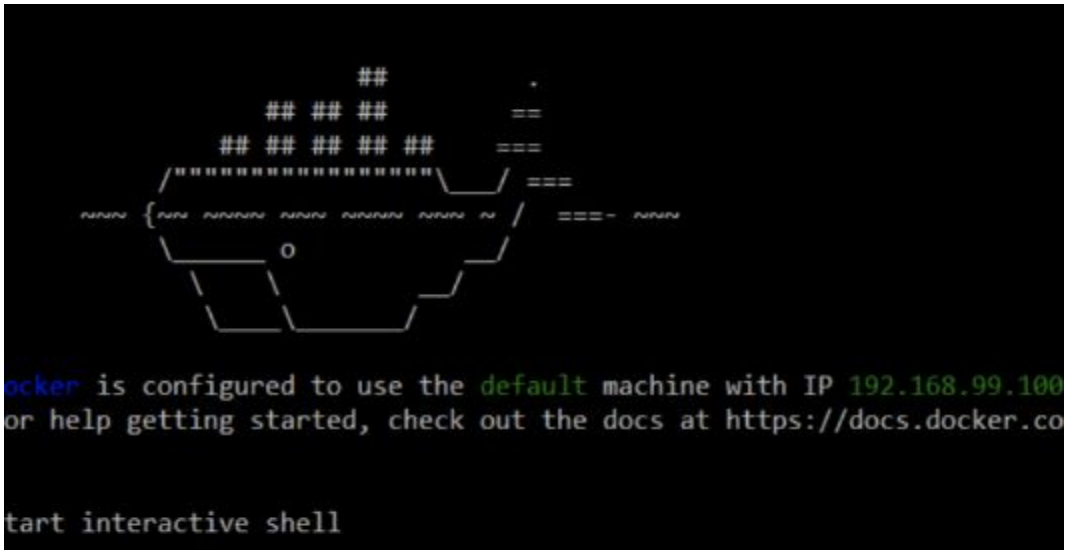
https://docs.docker.com/toolbox/overview

2. Follow the defaults and make sure Docker compose & Kitematic

are marked and Virtualbox and Git are not (we already have them installed installed):



3. Once installation is done open the Docker toolbox app: 4. Wait until you see the following:



Commands

• Docker comes with a set of commands, let's see a few of them:

- Getting current Docker version:

```
$ docker version
```

- Pull images from the **docker repository** (hub.docker.com)

```
$ docker pull <image_name>
```

- Docker Hub repositories let you share images with co-workers, customers, or the Docker community.
- Create a container from an image

```
$ docker run <image name>
```

- The run command can use a few different flags, for example:
- **--name =** give your container a name

```
$ docker run --name my_container
```

- **-p** = publish. usually this will look like that:

```
$ docker run -p 80:81 <name>
```

which will basically bind port **80** in the container to port **81** on the host system.

- **-d** = detached mode, so the container will run in background:

```
$ docker run -d test
```

- push an image to the docker hub repository

```
$ docker push <username/image name>
```

- Show a list of all the locally stored docker images

```
$ docker images
```

- Delete a stopped container

```
$ docker rm <container id>
```

- Delete an image from local storage

```
$ docker rmi <image id>
```

- Build an image from a specified docker file

```
$ docker build <path to docker file>
```

- Show a list of all running containers (which will print the containers id's):

```
$ docker ps
```

- Show a list of all containers (which will print the containers id's):

```
$ docker ps -a
```

- access a running container:

```
$ docker exec -it <container id> bash
```

- Stop a running container

```
$ docker stop <container id> o Stop container immediately
$ docker kill <image name>
```

https://docs.docker.com/engine/reference/commandline/docker/

Nginx

• Nginx (Engine X) is an open source reverse proxy server for HTTP, HTTPS, SMTP, POP3, and IMAP protocols, as well as a load balancer, HTTP cache, and a web server (origin server).

• According to Netcraft, nginx served or proxied 24.86% busiest sites in July 2018.

• Some of the success stories are: Dropbox, Netflix, Wordpress.com, FastMail.FM.

• We are going to use Nginx as a running Docker container:

- Pull Nginx image from hub: $ docker pull nginx
- Run Nginx image:

```
$ docker run --name docker-nginx -p 80:80 -d nginx
```

- Checking all running containers:

```
$ docker ps
```

- **Go to your host, open any browser and type your VM ip address : <port> $ ip a under enp0e8**
- Stopping Nginx container:

```
$ docker stop docker-nginx
```

- Removing Nginx container at the end:

```
$ docker rm docker-nginx
```

https://www.nginx.com/products/

Building a Dockerfile

• Create an image from Dockerfile using the following →

```
docker build -t my-image .
```

• Don’t miss the dot at the end!

• This will download and extract everything your image needs to be able to run.

• Finally, you will see your steps executing one after the other with the results

Once all steps are done, you can run your image using the run command followed by your image name:

```
docker run --name myfirstcontainer -d my-image
```

The result should be the current time.

https://docs.docker.com/engine/reference/builder/

```
$ docker build –t <image_name> .
```

VNC

• In computing, Virtual Network Computing (VNC) is a graphical desktop sharing system that uses the Remote Frame Buffer protocol (RFB) to remotely control another computer.

• It transmits the keyboard and mouse events from one computer to another, relaying the graphical screen updates back in the other direction, over a network.

• The closest comparison that can be used is TeamViewer.

• VNC is often used in Docker when running GUI apps (e.g. FIrefox).

• A “bigger” example is installing Ubuntu or even Mac OS X Docker image and being able to watch and take control of it via VNC.

• The Dockerfile has to specify this ability

https://qxf2.com/blog/view-docker-container-display-using-vnc-viewer/

Jenkins integration

• The main 2 usages of integrating Jenkins and Docker to get powerful results are:

• Using Docker containers as Jenkins slaves.

- We will need to download and run Jenkins image - **docker pull jenkins/jenkins**
- Run the image and use the container as a Jenkins slave.

• Using Jenkins to deploy software in containers and run it

- Have our Jenkins job create Dockerfile which will run on our containers.

https://hub.docker.com/r/jenkins/jenkins/

https://github.com/jenkinsci/docker/blob/master/README.md

Docker Networking

- With docker we can create a user-defined bridge network.
- It is a self-contained IP subnet and gateway.
- But unlike the default bridge, all containers within the user-defined bridge can communicate directly to one another without the need for port forwarding.
- Additionally, automatic discovery using DNS is fully supported on this network.
- If you want other devices in other networks to communicate with resources in your user-defined bridge network, you simply publish the TCP/UDP ports you need and Docker will expose the network addresses and ports to outside networks
- Create a network: `docker network create <network_name>`
- List all networks: `docker network ls`
- Remove a network: `docker network rm <network_name>`
- Specifying a network to a specific network:
- `docker run --name <container_name> --network <network_name> <image_name>`
- Create a network called wordpress
- Attach two containers to it:
    - MySQL container.
    - Wordpress container.

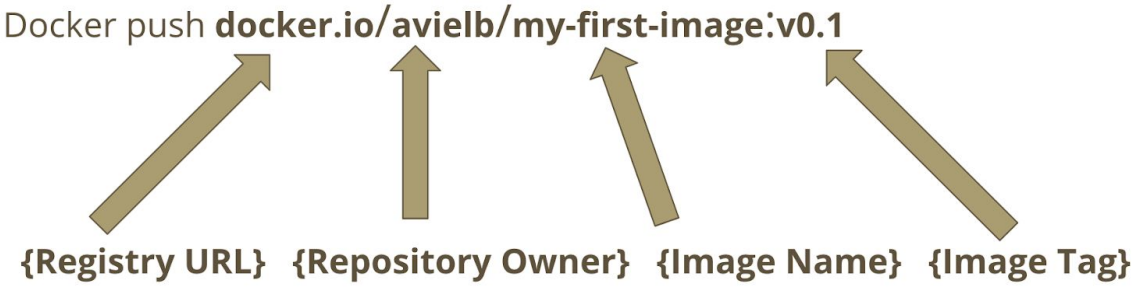
```
docker network create wordpress

docker container run --name mysql-container --rm --network
wordpress -e MYSQL_ROOT_PASSWORD=wordpress -d mysql:5.7

docker container run --name wordpress-container --rm
--network wordpress -e WORDPRESS_DB_HOST=mysql-container -e
WORDPRESS_DB_PASSWORD=wordpress -p 8090:80 -d wordpress
```

Docker registry and Dockerhub

- A registry is a storage and content delivery system, holding named Docker images, available in different tagged versions. Users interact with a registry by using docker push and pull commands.
- Docker repositories allow you share container images with your team, customers, or the Docker community at large.
- Docker images are pushed to Docker Hub through the docker push command.
- A single Docker Hub repository can hold many Docker images (stored as tags).



-

Docker Compose

- YAML is a human-readable data serialization language.
- It is commonly used for configuration files, but could be used in many applications where data is being stored (e.g. debugging output) or transmitted (e.g. document headers).
- YAML targets many of the same communications applications as XML but has a minimal syntax which intentionally breaks compatibility.
- It uses both Python-style indentation to indicate nesting, and a more compact format that uses [] for lists and {} for maps.
- Compose is a tool for defining and running multi-container Docker applications.
- With Compose, you use a Compose file to configure your application's services.
- Then, using a single command, you create and start all the services from your configuration.
- Compose is great for development, testing, and staging environments, as well as CI workflows.
- Using Compose is basically a three-step process.
- Define your app's environment with a Dockerfile so it can be reproduced anywhere.
- Define the services that make up your app in docker-compose.yml so they can be run together in an isolated environment.
- Lastly, run docker-compose up and Compose will start and run your entire app.
- Volumes key can be added to the compose file (yml) to mount the project directory (current directory) on the host to

/code inside the container, allowing us to modify the code on the fly, without having to rebuild the image.

-

```
version: '3.3'


services:
   db:
     image: mysql:5.7
     volumes:
       - db_data:/var/lib/mysql
     restart: always
     environment:
       MYSQL_ROOT_PASSWORD: somewordpress
       MYSQL_DATABASE: wordpress
       MYSQL_USER: wordpress
       MYSQL_PASSWORD: wordpress


   wordpress:
     depends_on:
       - db
```

```
    image: wordpress:latest
    ports:
      - "8000:80"
    restart: always
    environment:
      WORDPRESS_DB_HOST: db:3306
      WORDPRESS_DB_USER: wordpress
      WORDPRESS_DB_PASSWORD: wordpress
      WORDPRESS_DB_NAME: wordpress
volumes:
    db_data: {}
```