# This document is written to support students, taking Technology 2 at KEA Computer Science.

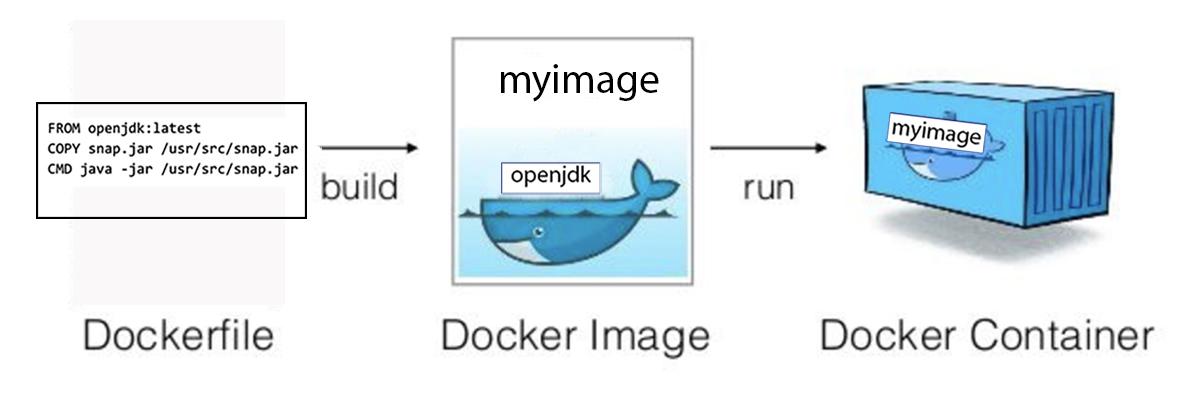
# Main sections of Docker.

# Install Docker: see [step 3](https://docs.google.com/document/d/1Zx0L6uYW04l6GdZ9jWUUnjfdOvEgSBM3yQyDMYM7uSs/edit#heading=h.ywiieq1rb6l6) of Jon's Dev-ops tutorial

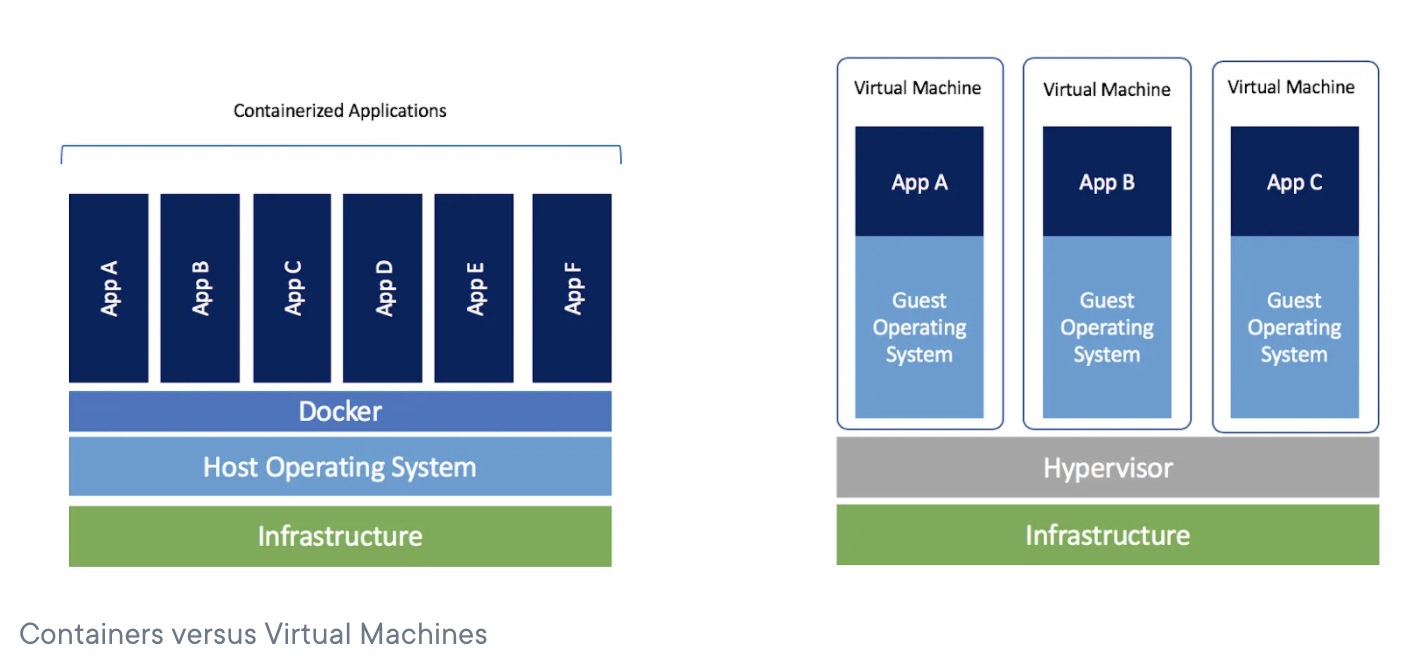
# Dockerfile.

* 1. A Dockerfile is needed in order to create a Docker image.
  2. The file must have the name Dockerfile (with no file extension)
  3. A Dockerfile to run a Java Spring Project can look like this:  
     FROM openjdk:latest  
     COPY snap.jar /usr/src/snap.jar  
     CMD java -jar /usr/src/snap.jar  
       
     Explanation of the 3 lines:
     1. Line 1: Downloads the Docker image "openjdk" from dockerhub.com, because Java is needed to run the .jar file.
     2. Line 2: Copies snap.jar into the image, to the folder /usr/src. From Maven in IntelliJ this file was created, and then transferred to EC2 using an FTP client, like Filezilla..
     3. Line 3: Tells Docker to execute the commmand "java -jar /usr/src/snap.jar" which will start up the Spring project.
  4. A Dockerfile can be created using the terminal text editor nano. To create a new file, type:  
     $ sudo nano Dockerfile
     1. When you are done typing, press CTRL + x to exit
     2. press Y to accept to save the file
     3. press ENTER to accept the filename
     4. "sudo" allows you to run programs with the security privileges as 'superuser'.

# Docker Image

* 1. A Docker image is the foundation of a Docker container. Similar to how an .iso file is needed to install an operating system, like Windows or Ubuntu. In the example below I have made a Dockerfile which will be used to create myimage. In this case, myimage is based on the *openjdk* image, which is available from [hub.docker.com/\_/openjdk](https://hub.docker.com/_/openjdk). It is required to run a Java webapplication from a .jar file.  
       
       
     source: <https://medium.com/platformer-blog/practical-guide-on-writing-a-dockerfile-for-your-application-89376f88b3b5>
  2. The Dockerfile has information about, what software should be in the running container. In our example above 2.c), we decided to add the Java JDK and a .jar file with a Java Spring project inside.
  3. After you have created the Dockerfile, it is possible to create a new image:  
     $ sudo docker image build . --tag myimage  
     Explanation of this command, word by word:
     1. sudo: runs Docker with 'superuser' privileges
     2. docker: runs a Docker command
     3. 'image build' : creates a Docker image, based on the information in the Dockerfile
     4. . (period) : tells Docker, that the Dockerfile is located in the current directory
     5. '--tag myimage' : gives this image the repository-name 'myimage' and the default tag=latest.   
        It is also possible to create another version of the image:  
        '--tag myimage:version2'  
        In this way, it is possible to create multiple versions of an image
  4. You can see all images with:  
     $ sudo docker images
  5. You can remove an image with this: (but you have to stop any container first, which is based on this image)  
     $ sudo docker rmi myimage
  6. Instead of creating your own Docker image, you can start with an existing image. Use this command:  
     $ sudo docker pull myimage  
     to pull any of the 5.000.000+ images for download at [DockerHub](https://hub.docker.com/search?q=&type=image). For example: mysql, mongo, mariadb, ubuntu, node, python, openjdk etc.
  7. All Docker image commands are listed [HERE](https://docs.docker.com/engine/reference/commandline/image/)

# Docker Container

1. A Docker container is like a mini "virtual machine" (see drawing below). It uses fewer resources than a virtual machine. That is why Docker is being used a lot for deploying application and services.  
     
   source: <https://www.docker.com/blog/containers-replacing-virtual-machines/>
2. A Docker container can be created when there is an image ready. (see 3.)
3. To start a Docker container the first time, use this syntax:  
   $ sudo docker run -d -p 80:8080 --name mycontainer myimage  
   Explanation of this command, word by word:
   1. sudo: runs Docker with 'superuser' privileges
   2. docker: runs a Docker command
   3. -d : run container in "detached" mode, which means, that one can interact with the terminal, while the container is running.
   4. '-p 80:8080' : publish port 80 to "the world" while mapping to port 8080 inside the container. Why 8080? Because Java Spring projects will use port 8080 as default. This can be changed in the application.properties file, in IntelliJ.
   5. '--name mycontainer' : gives this container an easy to remember name. The name can be used later, for stopping, starting and removing.
   6. myimage: is the name of the image, from which the container should be built. You can also specify a tag (highlighted) if you want a particular version:  
      $ sudo docker run -d -p 80:8080 --name mycontainer myimage:**version2**
4. A Docker container can either be running or stopped. When running, the container does whatever is written in the Dockerfile. In our example, we ran a Java Spring project on port 8080. When stopped, the container preserves it's state and internal file system. It can be started later, and will continue.   
   Here are the necessary Docker commands regarding containers: (# means comment)
   1. $ sudo docker run --name mycontainer myimage # will create AND start a container, named mycontainer
   2. $ sudo docker stop mycontainer # will stop a running container. The container will keep its state. (files inside the container are preserved)
   3. $ sudo docker start mycontainer # will start a stopped container. The container will restart in the same state as before.
   4. $ sudo docker rm mycontainer # will remove a stopped container
   5. $ sudo docker rm -v mycontainer # will remove a stopped container AND delete the associated volume

1. MySQL on Docker. When running a MySQL server on docker, there is a special option needed, called --env (short for environment):  
   $ sudo docker run -d -p 3306:3306 --env="MYSQL\_ROOT\_PASSWORD=mypassword" --name=mysqlcontainer mysql  
   This --env option sets the password for the root user, inside the MySQL server. This password will be required to connect the MySQL server from, say your Java Spring project.
2. To see the output from a container:  
   $ sudo docker logs mycontainer  
   Here you can see, what happened when you for example started a Java Spring container or a MySQL container
3. All Docker container commands are listed [HERE](https://docs.docker.com/engine/reference/commandline/container/)

------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**Useful Docker and terminal commands:**

1. See amount of free diskspace:  
   $ df
2. To clear space from the harddrive, related to Docker: (**NOTE: This will remove ALL containers and images**)  
   $ docker system prune --all --force --volumes
3. To connect to a remote MySQL server, using macOS terminal (requires that mysql is installed locally)
   * 1. $ cd /usr/local/mysql/bin
     2. $ sudo ./mysql -u'root' -p'mypassword' -h'your.ip.address.here'

# Resources:

Short Cheat-sheet with many useful Docker commands:

<https://keraton.gitbooks.io/docker-cheat-sheet/content/dockercommand_essentials_md.html>

Docker state diagram (Docker container's lifecycle)

<https://medium.com/@nagarwal/lifecycle-of-docker-container-d2da9f85959>

*By:*

*Jon Eikholm, KEA*

[*jone@kea.dk*](mailto:jone@kea.dk)

*February 2021*