Containers and Docker

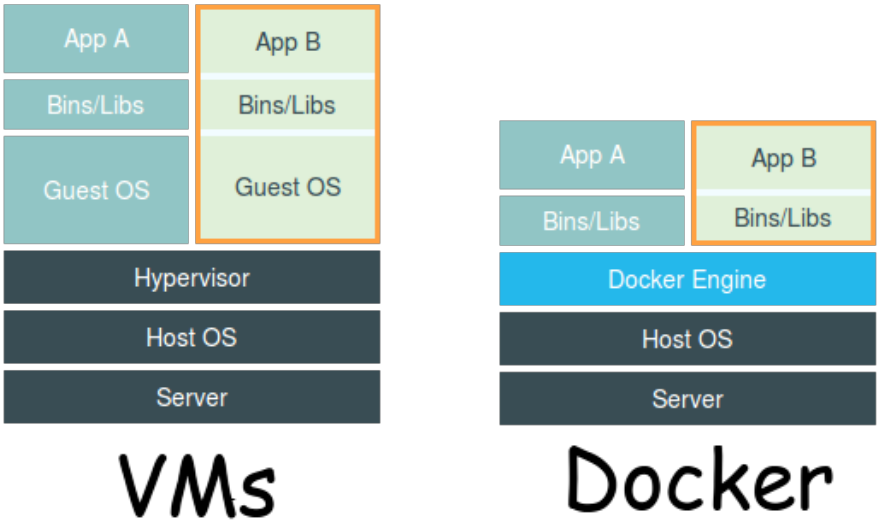
CS Tools, Tips and Tricks Workshop

Version 2019 Winter, McGill University

1. Overview of Virtualization

Using a virtualization method and building an image provides consistent environments. The virtualization method can be categorized based on how it mimics hardware to a guest operating system and emulates guest operating environment. Primarily, there are two main types of virtualization:

- **Emulation and paravirtualization** is based on some type of hypervisor which is responsible for translating guest OS kernel code to software instructions or directly executes it on the bare-metal hardware. A virtual machine provides a computing environment with dedicated resources, requests for CPU, memory, hard disk, network and other hardware resources that are managed by a virtualization layer. A virtual machine has its own OS and full software stack.
- **Container-based virtualization**, also known as operating system-level virtualization, enables multiple isolated executions within a single operating system kernel. It has the best possible performance and density, while featuring dynamic resource management. The isolated virtual execution environment provided by this type of virtualization is called a *container* and can be viewed as a well-defined group of processes.



2. Docker foundations

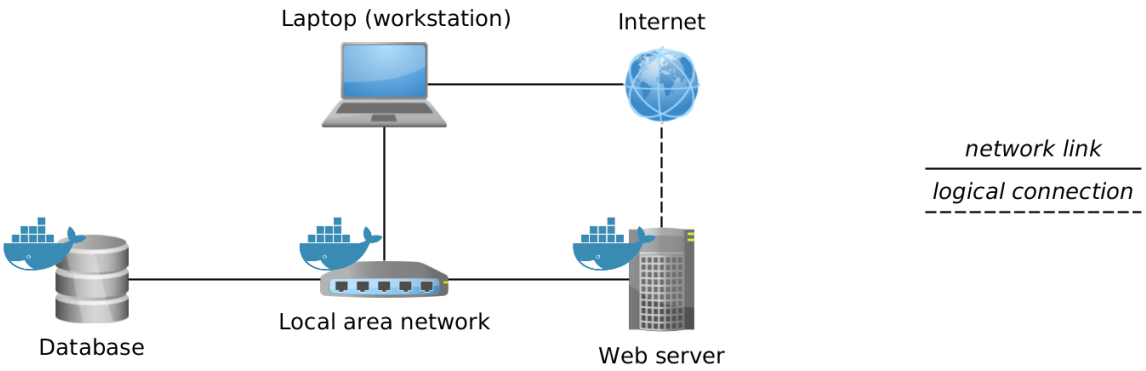
The core concepts of Docker are **images** and **containers**.

A **Docker image** is a read-only template with instructions for creating a Docker container. For example, an image might contain an Ubuntu operating system with an Apache web server and your web application installed. You can build or update images from scratch or download and use images created by others. An image may be based on, or may extend, one or more other images. A Docker image is described in text file called a Dockerfile, which has a simple, well-defined syntax.

A **Docker container** is a runnable instance of a Docker image. You can run, start, stop, move, or delete a container using Docker API or CLI commands. When you run a container, you can provide configuration metadata such as networking information or environment variables.

Exercise. We are going to create a simple virtual network on which a web server runs a *Java Spring*

application. The functionality provided by the web server via a RESTful API is to (i) add people and (ii) add events to the system, as well as to (iii) register selected people to selected events. This example application is adapted from the ECSE321 course tutorial. We are also exposing this functionality to the internet. Furthermore, we run a Postgres Database that is also connected to this same network. This database is used by the java application running on the web server to store the registration info. An architecture diagram of the system is shown below.



NOTE For any Docker command, you can use the `--help` command line switch to learn about that particular command and its possible parameters. For example, try `docker run --help`

2.1. Docker containers

2.1.1. Running a container

To create and run a container, one needs to specify an image on which the container is based on. Luckily, docker has the support for downloading images automatically from an online *repository* in which it identifies images by their names and versions.

Things to try:

- `docker run hello-world`
- `docker run --rm busybox ping www.google.com`
- `docker run -it busybox`

Example. Setting up the database can be done by

We can create a containerized version of PostgreSQL by issuing `docker run --name database-postgresql -e POSTGRES_PASSWORD=pass -e POSTGRES_USER=user -e POSTGRES_DB=eventregistrations -d postgres` command.

2.1.2. Container management

Docker offers commands (among many) to list, stop, start, and remove containers. Furthermore, the `docker inspect` command can tell several details about the configuration of the given container.

Things to try:

- `docker ps`
- `docker ps --all`
- `docker container prune`
- `docker inspect <CONTAINER_ID>`

2.2. Docker images

- **Dockerfile**: An image is defined in a `Dockerfile`. Every image starts from a base image, e.g. from `ubuntu`, a base Ubuntu image. The Docker image is built from the base image using a simple, descriptive set of steps we call instructions, which are stored in a `Dockerfile`. Main dockerfile **instructions**:
  - `FROM`
  - `RUN`
  - `ADD`
  - `VOLUME`
  - `CMD`
  - `WORKDIR`

[More details here](https://docs.docker.com/engine/reference/builder/); example for tomcat is here: https://blog.lukaspradel.com/dockerizing-a-tomcat-postgresql-java-web-application/

- **Command** `docker build .`: it builds an **image** from a **Dockerfile** and a **context**. The build's **context** is the set of files at a specified location PATH or URL (in this case the current directory, `.`). The PATH is a directory on your local filesystem. The URL is a Git repository location. Add the `-f` switch to specify the Dockerfile location, if it is not present in the root context.
- **Command** `docker run`: Run a command in a new container. Images come to life with the `docker run` command, which creates a container by adding a read-write layer on top of the image. This combination of read-only layers topped with a read-write layer is known as a [union file system](https://en.wikipedia.org/wiki/UnionFS). Changes exist only within an individual container instance. When a container is deleted, any changes are lost unless steps are taken to preserve them. Parameters to show: `-v`, `-i`, `-t`, `-a`
- **Command** `docker start/stop`: Start/stop a container.
- **Command** `docker exec`: Execute a command in a running container.
- Managing images and containers:

`docker ps --all`:

```
CONTAINER ID  IMAGE                 COMMAND                  STATUS
9bba8a2a3f81  makisyu/texlive-2016  "/bin/bash -c 'sleep…"  Exited (0) 4 days ago
cd005b9af0af  makisyu/texlive-2016  "/bin/bash -c 'sleep…"  Exited (0) 4 days ago
b92dd4d5886d  eclipse/che           "/scripts/entrypoint…"  Exited (2) 5 days ago
```

`docker images ls --all`:

```
REPOSITORY            IMAGE ID      CREATED       SIZE
eclipse/che           8956a46aa7e3  10 days ago   51.3MB
gradle                e7f185032db8  2 months ago  820MB
busybox               6ad733544a63  3 months ago  1.13MB
makisyu/texlive-2016  bb92f3e57f6b  9 months ago  5.42GB
```

#4. DockerHub + demo

Dockerhub and its usage. Related commands:

- **Command** `docker pull`
- **Command** `docker push`
- **Command** `docker save`
- **Command** `docker load`

#5 Exercises:

pull busybox ping a server with busybox (show ping, ping from docker) show interactive mode

```
docker attach
docker run -d
docker run -it

- if [[ -e docker/texbuilder.tar.gz  ]] ; then gzip -dc docker/texbuilder.tar.gz |
docker load ; else docker pull makisyu/texlive-2016 ;  docker save makisyu/texlive-
2016 | gzip > docker/texbuilder.tar.gz ; fi

docker run -d --name texbuilder -v `pwd`/:/mnt/ makisyu/texlive-2016 /bin/bash -c
"sleep 10"
docker ps -a
docker exec -ti texbuilder /bin/bash -c "cd /mnt && pdflatex /mnt/calendar_1.tex"
ls -la

docker run -it -v `pwd`/:/mnt/ openjdk
```

Miscellaneous

1. Running docker without sudo

Taking the steps from this answer on askubuntu

1. Add the docker group if it doesn't already exist: `sudo groupadd docker`
2. Add the connected user "$USER" to the docker group. Change the user name to match your preferred user if you do not want to use your current user: `sudo gpasswd -a $USER docker`

3. Either do a `newgrp docker` or log out/in to activate the changes to groups. You can use docker run hello-world to check if you can run docker without sudo.

2. Official Images

2.1. hello-world

2.2. busybox

docker run busybox ping www.google.com

3. MySQL

An official image for MySQL is available here. We can start using the database right away by following the steps included in section "How to use this image".

4. Web server: Tomcat

Docker inspect command