docker_summary

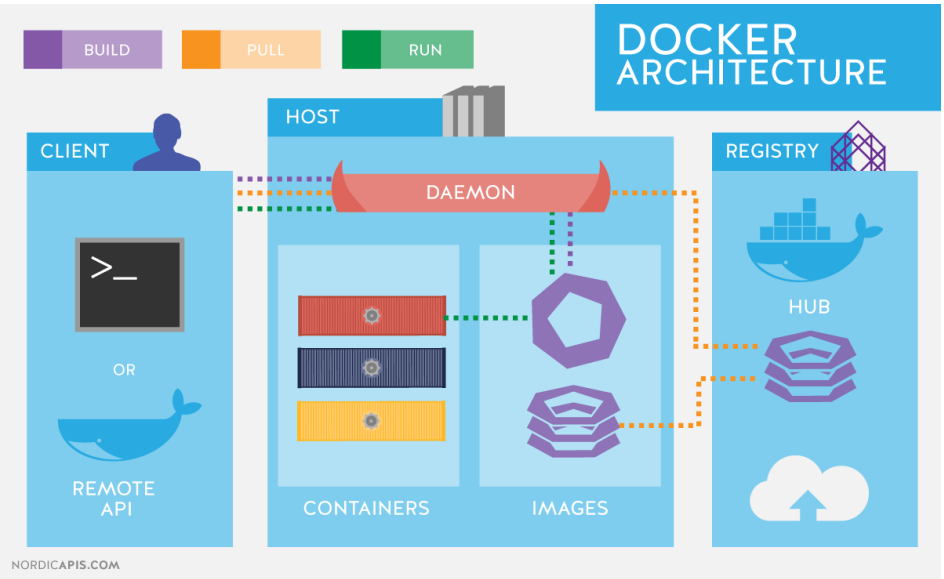
March 19, 2019

1 DOCKER

1.1 Introduction

1. **software stack**:
    - stack:
        1. **front-end components**
        2. **back-end workers**
        3. **database components**
        4. **environment and library dependencies**
    - these components can differ *greatly* on **different platforms**
2. **need**: make sure that a code running in a **development environment** works in a **testing environment as well** (*i.e. insuring portability of code during software deployment on every possible platform*)
3. **definition**:
    - `Docker` is a **software container platform**: *i.e.* a software that performs **operating-system level virtualization** called "**containerization**" at the **deployment stage of a software**
    - it allows a developer to **package up an application with all its dependencies** (*tools, libraries, configuration files, etc.*) in a **single package** (*bundle*) called a `container` and **abstracts** from him its **portability insuring details** (*i.e. shipping details*)
    - **containers** are **isolated** from each other and communicate via **well-defined channels**
4. **workflow**:
    - `dockerfile`: a developer will define the **application and its dependencies and requirements** in a file called a **dockerfile**
    - `docker images`:
        1. a **dockerfile** describes **steps to create a Docker image**
        2. **Docker images** are **templates** (*containing all the dependencies and requirements of the application*) used to create `docker containers`
        3. they can be stored in **online cloud repositories** called "`Docker registries`" (*e.g.* ***Dock Hub***) and can be **pulled** to create **containers in any environment** (*e.g. test environment, staging environment, etc.*)
    - `docker containers`: **docker containers** are the **runtime instances** of a **Docker image**

5. **containerization vs. virtualization**:
    - **virtualization**:
        1. a **hypervisor** (*software*) is used to **create and run multiple virtual machines** (*i.e. **guest OSs***) on a **host OS**
        2. the **virtual machines** have **their own OS and do not use the host OS** -> they are run by **multiple OS kernels** -> **overhead** on the **host platform**
        3. **resource allocation** is **fixed** and does not change as per application needs
    - **containerization**:
        1. a **container engine** (*software*) is used to **create and run containers** (*i.e. **an application and all its dependencies***)
        2. the **containers** use the **host OS** -> they are run by **a single OS kernel**
        3. **resource allocation** is **dynamic as per application needs**
        4. a **docker container** could be **run within a virtual machine** (*i.e. on the virtual machine's OS*)
        5. **less overhead**, more lightweight and **faster than virtual machines**
6. **advantages**:
    - **portability of code** on different platforms during **deployment**
    - **pushing/pulling docker images** from **docker registries** and using them in **different environments**
    - **built-in VCS** similar to `git` : **commit messages**
    - **container isolation**:
        1. **no interference** with applications running on the **same OS**
        2. **multiple containers** can be **executed simultaneously on the same OS**
    - **clean container purging**: **deleting a container deletes also all of its dependencies and requirements along**
7. **architecture**: a **client-server architecture**:
    - **client**: **CLI**
    - **server**:
        1. a `Docker daemon` **containing all the containers**
        2. can be on the **same machine** as that of the **client** or on a **different one**
        3. can **interact** with a **Docker Registry**
    - **interaction**: **client** -> **CLI commands** or **REST API requests** -> **server**
    - **actions**:
        1. **build a Docker image from a dockerfile**
        2. **run a docker container from a Docker image**
        3. **pull/push a Docker image from/to a Docker registry**
    - `Docker engine` = **client + server + their components**



Installation on Ubuntu 18.04 from the official Docker repository https://linuxconfig.org/how-to-install-docker-on-ubuntu-18-04-bionic-beaver

1.2 Basic commands

1.2.1 Starting, stopping and information about docker server commands

- **starting the docker server**: `sudo service docker start`
- **adding permissions to the current user to run docker**: `sudo usermod -a -G docker USER`(*then restarting*)
- **help** on **Docker commands**:
  1. **listing all commands**: `docker --help`
  2. **help on a specific command**: `docker <command> --help`
- **version** of **Docker**:
  1. *short description*: `docker --version` or `docker -v`
  2. *long description*: `docker version`
- **information on Docker**: `docker info`
- **stop the docker server**: `sudo service docker stop`

1.2.2 Docker registry commands

- **login to a docker registry**: `docker login [option...] [server]`:
  1. login with a **string username**: `docker login --username <username>`
  2. login with a **string password**: `docker login --password <password>`
  3. login with a **password written to the standard input**: `docker login --password-stdin`
  4. **default server value** = "https://hub.docker.com"

1.2.3 Docker image commands

- **list images**: `docker images [option...]`
  1. `-a`, `--all`: **show all images** (*default behavior*)
  2. `--digests`: **show digests**
  3. `-q`, `--quiet`: only show **numeric ID** of a **docker image**
  4. `-f, --filter <filter>`: **filter output based on provided conditions** (*e.g.* `dangling=true` *or* `dangling=false`)
  5. **dangling image**: **associated to a docker container**
- **remove an image**: `docker rmi [option...] <image>` :
  1. `-f`, `--force`: **remove an image forcibly**
- **create a docker container from an image**:
  1. `docker run <image>[:<image-tag>] [option...]`
     - `--name <container-name>`: assign "**container-name**" to the **created container**
     - `-it`: **create and start the container** (*run interactively*)
  2. if the **image** is **not present locally**, **docker** will try and **pull a docker image** called "**image**" from the **docker registry logged on to**
  3. *by default*: **image-tag** = "latest"
- **pull an image from a docker registry**: `docker pull <image>[:<image-tag>]`
- **inspect an image as a JSON file**: `docker inspect <image>[:<image-tag>]`
- **history of an image**: `docker history <image>[:<image-tag>]`

1.2.4 Docker container commands

- **list container processes**:
  1. **running containers**: `docker ps`
  2. **all containers**: `docker ps -a` or `docker ps --all`
- **start a container**: `docker container start <{containerID | containerNAME}>`
- **pause a container**:
  1. `docker container pause <{containerID | containerNAME}>`
  2. **all actions on the stdin will be saved in the IO buffer**
- **unpause a container**:
  1. `docker container unpause <{containerID | containerNAME}>`
  2. **all actions saved in the IO buffer will be flushed**
- **stop a container**: `docker container stop <{containerID | containerNAME}>`
- **inspect running processes on a container**: `docker top <{containerID | containerNAME}>`
- **inspect memory and IO stats of a container dynamically**: `docker stats <{containerID | containerNAME}>`
- **attach a running container process to the current process**: `docker attach <{containerID | containerNAME}>`
- **remove a container** (*that's not running*): `docker rm <{containerID | containerNAME}>`
- **kill a running container process**: `docker kill <{containerID | containerNAME}>`

1.2.5 Docker system commands

- **system memory stats on a running container**: `docker stats`
- **system disk stats on a running container**: `docker system df`
- **remove unused data**: `docker system prune`

1.3 Creating and building dockerfiles

1. **dockerfile**: *a text file with instructions to automatically build **docker images***
2. **basic instructions**:
   - `FROM <docker-image> | scratch`:
     1. `<docker-image>`: start from an **existing docker image** (*i.e.* ***base image***) to create a **custom docker image**
     2. `scratch`: an **empty docker image used for building docker images**
   - `MAINTAINER author <email>` (*optional*)
   - `RUN <command>` : **run the command during docker image creation**
   - `CMD ["command1"[, "command2"[, ...]]]` : **run the command(s) on the terminal during container creation from the docker image**
   - `# this is a comment`
3. **process**:
   - **create** a file named "**Dockerfile**"
   - **add instructions** to the **Dockerfile**
   - **build the dockerfile** to **create the docker image**: `docker build [-t imageName:imageTag] pathToDockerfile`
   - **run the image to create a container**

2 DOCKER COMPOSE

2.1 Introduction

1. **definition**:
   - a **tool** for **defining & running multi-container docker applications**
   - uses **YAML** (**Yet Another Markup Language**) **files** to **configure application services** (`docker-compose.yml`)
   - works in **all environments** (*production, staging, development, testing, etc.*)
   - can **start all services with a single command**: `docker compose up`
   - can **stop all services with a single command**: `docker compose down`
   - can **scale up selected services when required**
2. **installation**:
   - **method 1**:

```
#download the package and install it
curl -L https://github.com/docker/compose/releases/download/1.24.0-rc1/
docker-compose-`uname -s`-`uname -m` -o /usr/local/bin/docker-compose
#(always check for latest release before installing)
```

```
#give execution permissions on the installed file
sudo chmod +x /usr/local/bin/docker-compose
```

- **method 2** - *using pip*: `pip3 install -U docker-compose`

example of instructions:

```
services:

  web:
    image: <image-for-web-server>

  database:
    image: <image-for-db-server>

version: '3'
```

process:

1. **create** a file named "**docker-compose.yml**"
2. check the **validity of the file**: `docker-compose config`
3. **creating the docker containers and application** (*in detached mode*) and **starting all the services**: `docker-compose up -d`
4. **stopping the services and the application**: `docker-compose down`
5. **scaling services**: `docker-compose up -d --scale <service>=<nbContainers>`

3 ANNEXE - Keywords

software container platform
containerization
virtualization
containers
docker container
dockerfile
docker image
docker Hub/Registry
docker daemon
docker engine
portability
deployment
bundle
package
VCS
REST API
push/pull/commit
docker build
docker compose
YAML
scaling