docker_summary.md

DOCKER

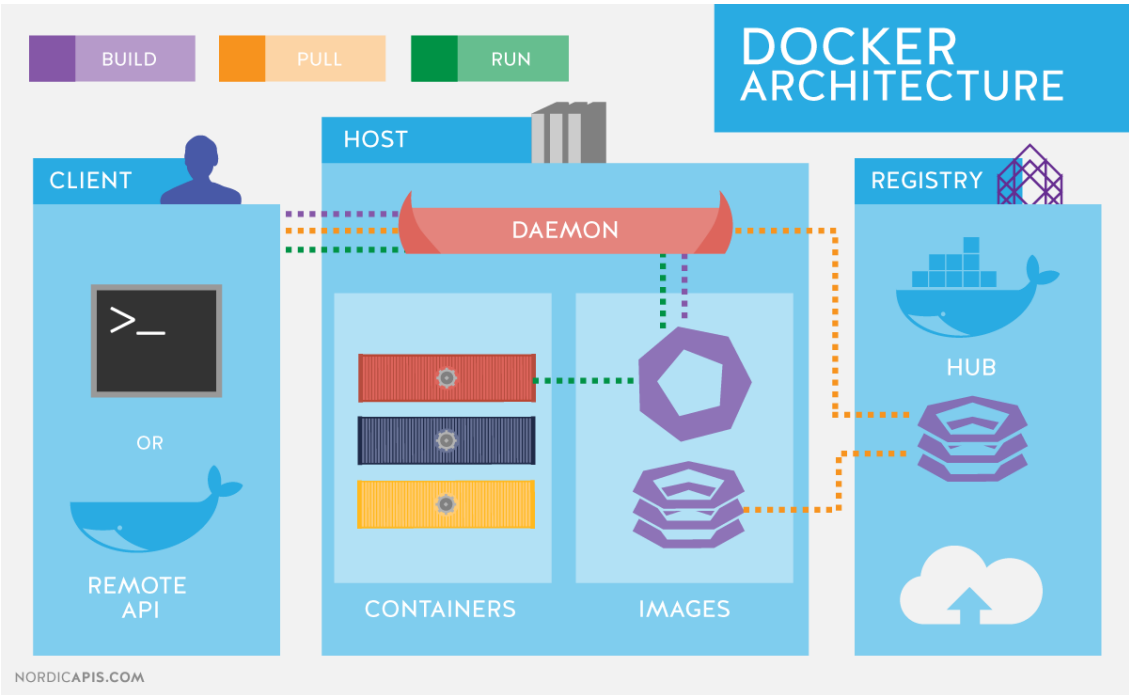
Introduction

1. **software stack**:
   - stack:
     a. **front-end components**
     b. **back-end workers**
     c. **database components**
     d. **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` :
     a. a **dockerfile** describes **steps to create a Docker image**
     b. **Docker images** are **templates** (*containing all the dependencies and requirements of the application*) used to create `docker containers`
     c. 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**:
     a. a **hypervisor** (*software*) is used to **create and run multiple virtual machines** (*i.e.* ***guest OSs***) on a **host OS**
     b. 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**
     c. **resource allocation** is **fixed** and does not change as per application needs
   - **containerization**:
     a. a **container engine** (*software*) is used to **create and run containers** (*i.e.* ***an application and all its dependencies***)
     b. the **containers** use the **host OS** -> they are run by **a single OS kernel**

c. **resource allocation** is **dynamic as per application needs**
d. a **docker container** could be **run within a virtual machine** (*i.e. on the virtual machine's OS*)
e. **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**:
     a. **no interference** with applications running on the **same OS**
     b. **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**:
     a. a `Docker daemon` **containing all the containers**
     b. can be on the **same machine** as that of the **client** or on a **different one**
     c. can **interact** with a **Docker Registry**
   - **interaction**: **client** -> **CLI commands** or **REST API requests** -> **server**
   - **actions**:
     a. **build a Docker image from a dockerfile**
     b. **run a docker container from a Docker image**
     c. **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

Basic commands

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**:
  i. **listing all commands**: `docker --help`
  ii. **help on a specific command**: `docker <command> --help`
- **version** of **Docker**:
  i. *short description*: `docker --version` or `docker -v`
  ii. *long description*: `docker version`
- **information on Docker**: `docker info`
- **stop the docker server**: `sudo service docker stop`

Docker registry commands

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

Docker image commands

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

Docker container commands

- **list container processes**:
  i. **running containers**: `docker ps`
  ii. **all containers**: `docker ps -a` or `docker ps --all`

- **start a container**: `docker container start <{containerID | containerNAME}>`
- **pause a container**:
  i. `docker container pause <{containerID | containerNAME}>`
  ii. **all actions on the stdin will be saved in the IO buffer**
- **unpause a container**:
  i. `docker container unpause <{containerID | containerNAME}>`
  ii. **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}>`

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`

Creating and building dockerfiles

1. **dockerfile**: *a text file with instructions to automatically build **docker images***
2. **basic instructions**:
   - `FROM <docker-image> | scratch` :
     a. `<docker-image>` : start from an **existing docker image** (*i.e. **base image***) to create a **custom docker image**
     b. `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**

DOCKER COMPOSE

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>`

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