클라우드 기술개요

<가상화 및 컨테이너를 중심으로>

과정 학습 목표

- 본 과정에서는 학습을 위해 필요한 이론적 내용을 제공하고, 필요한 경우 이해를 돕기 위해 동작사례를 시연합니다. 학습 목표는 다음과 같습니다.
  - 다음 개념들에 대하여 설명할 수 있습니다.
    - Linux OS, Virtual Machine, and Container
  - Docker, Docker Inc., Moby 등에 대하여 구분하여 설명할 수 있습니다.
  - 다음 Docker 요소들에 대하여 설명할 수 있습니다.
    - Docker Images
    - Docker Containers
    - Docker Registries

사전 학습 사항 (권장)

- Linux system management skills
- Linux network administration skills
- Shell scripting (bash) skills

- 지적 호기심
- IT Trends에 대한 관심
- 참여 (participation)

Agenda

- Module 0 – Course Overview
- Module 1 – Basic Concepts: Linux Architecture, Virtualization, and Containers
  - Linux OS Architecture
  - Concepts of Virtualization
  - Server Virtualization and Virtual Machines
  - Containers
  - Docker
  - Concepts of UnionMount Filesystem
  - History of:
    - Virtual Machines, Cloud, Docker, Kubernetes, Microservices

Agenda

- Module 2 – Docker
  - What is Docker?
    - Docker Inc, Docker software, and Docker Open Source Project
    - Tale of Docker, Rocket, and OCI
  - Using Docker
    - Docker CLI
    - Creating Images
    - Working with Images
    - Sharing Images using Docker Registry

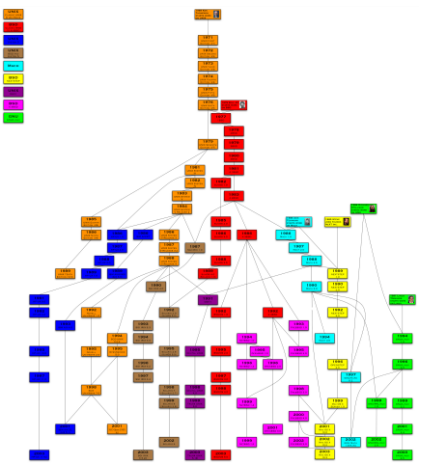
Linux OS Architecture

OS의 역사에 대하여

- The First Generation (1940 to early 1950s)
  - ENIAC + NO OS
- The Second Generation (1955 - 1965)
  - IBM Mainframe + GMOS
- The Third Generation (1965 - 1980)
  - DEC PDP-1 + MULTICS
- The Fourth Generation (1980 - Present Day)
  - PCs + DOS/Windows/MacOS

https://www.javatpoint.com/history-of-operating-system

(Optional) UNIX와 Linux의 역사에 대하여

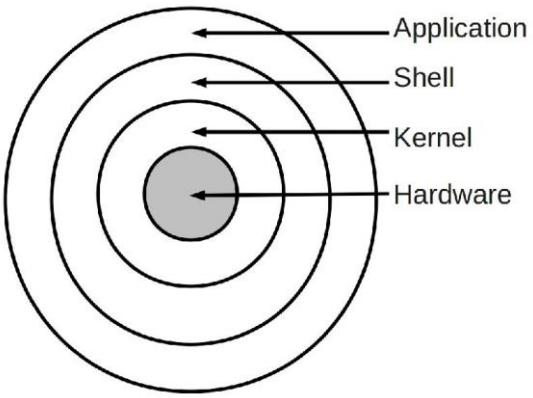


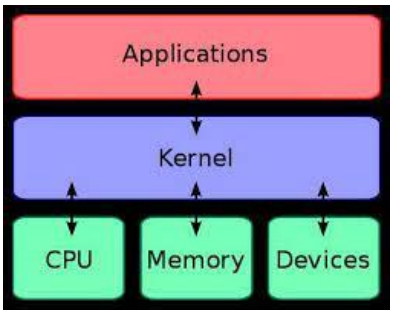
http://www.netneurotic.de/mac/unix/images/UNIX.png

(Optional) Linux와 GNU의 관계에 대하여

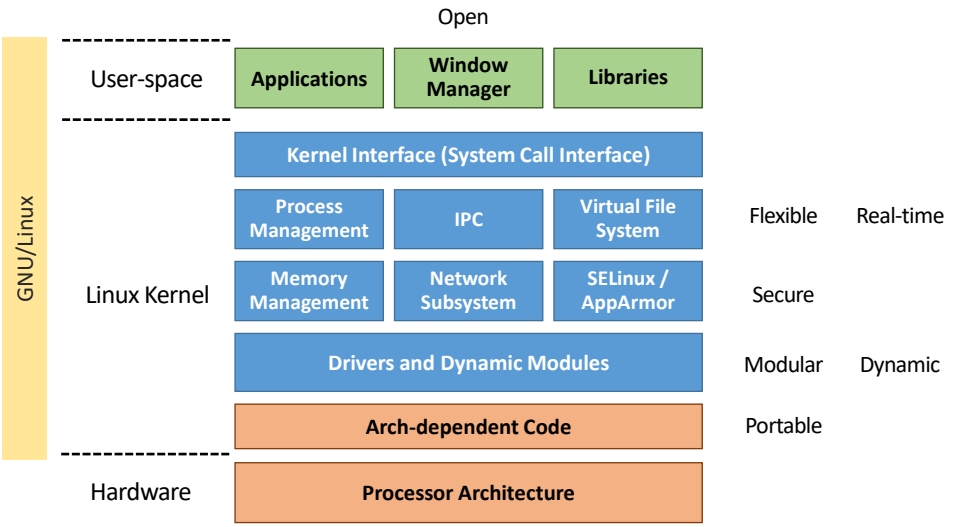
- GNU (Operation System project)
  - Richard Stallman에 의해 시작됨
  - USENET newsgroup에 1983년 9월 27일 공표됨
  - 1985년 Free Software Foundation (FSF) 설립
  - GCC, glibc, GDB, coreutils, binutils, bash 등의 소프트웨어가 제작됨
  - GNU는 여러 종류의 kernel과 조합이 가능함
    - Linux (GNU/Linux) (1991)
    - FreeBSD (1993)
    - Linux-Libre (2012)
    - GNU Hurd (GNU/Hurd) (2015)
  - 1989년, GNU General Public License (GNU GPL)을 발표함
- Linux
  - Linus Torvarlds에 의해 시작됨
  - USENET newsgroup에 1991년 9월 7일 공표됨
  - Linux kernel은 GNU GPLv2 라이선스 하에 배포됨

OS 구조에 대한 두 가지 관점





Linux on the Surface



Virtualization Concepts

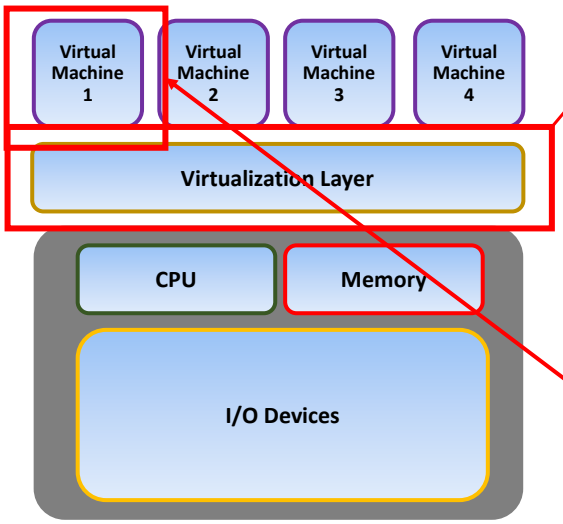
Virtualization

- Virtualization (가상화) 란?
  - 컴퓨터 하드웨어, 스토리지 장치, 네트워크 등의 ( )을
  - 가상 버전, 즉 ( )으로 만드는 것
- 가상화의 종류 (대상)
  - 서버가상화
  - 네트워크 가상화
  - 스토리지 가상화
  - 데이터센터 가상화
- 가상화를 하는 목적은?
  - 왜 가상화를 하는가? / 가상화의 Pros and Cons?

Server Virtualization (서버 가상화)

- 가상 서버
- 여러 가지 형태로 구현됨
  - 전체 하드웨어 형태
  - 혹은 특정 요소의 논리적 추상화 형태
  - 혹은 운영체제 실행을 위한 기능만을 구현한 형태 등
- 사용자에게 물리적인 특징을 감추고 추상화된 컴퓨팅 플랫폼을 표현함
- 여러 가지 방법으로 구현 가능함:
  - Hardware-based Partition model
  - Software-based VM model

VM-based virtualization



- **Virtualization Layer**는 다음 두 가지로 구성됨:
  - Host 운영체제
  - 가상화 소프트웨어
- **Virtualization Layer**는 Hypervisor라고 부르기도 하며, Type I과 Type II로 구분하기도 함.
- 각각의 **Virtual Machine**에는 다음 구성 요소들이 포함됨:
  - 가상화된 하드웨어 (pseudo F/W)
    - CPU, Memory, Storage, Network resources
  - Guest 운영제체

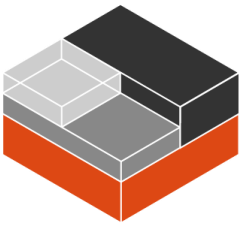
Virtual Machine 의 유형

- Full virtualization (전가상화)
  - 시스템의 모든 것이 가상화됨
    - CPU, storage, networking, etc.
  - VMWare, VirtualBox
- Para-virtualization ( )
  - VM guest를 수정함 (virtual HW and/or OS)
  - VM의 I/O 성능이 개선됨
  - Xen, VMWare with VMWare tools, VirtualBox with Vbox Guest Addition
- OS level virtualization (OS 수준 가상화, 컨테이너)
  - HW나 OS는 그대로이며, 애플리케이션이 OS를 바라보는 환경만 격리됨
  - Solaris ZONE, **Linux Container (LXC)**
  - OS(system) Containers vs. App Containers

Container Concepts

Linux Container (LXC) and Docker

What is LXC (Linux Container)?



- An **operating-system-level virtualization** method for:
  - running **multiple isolated Linux systems (containers)**
  - on a control host using a **single Linux kernel**
- The Linux kernel provides:
  - the **cgroups** functionality
    - allows limitation and prioritization of resources (CPU, memory, block I/O, network, etc.)
    - without the need for starting any virtual machines
  - **namespace** isolation functionality
    - allows complete isolation of an applications' view of the operating environment
    - including **process trees**, **networking**, **user IDs** and **mounted file systems**

Namespaces

- Linux Kernel Feature
  - 커널 자원을 파티션하여
  - **프로세스 셋마다 서로 다른 자원 셋을 보게 함**
- 자원들의 유형
  - Process IDs, hostnames, user IDs, file names, and some names associated with network access, and interprocess communication.
- 컨테이너 **모양**의 기초 형태

6 Kinds of Namespace (as of kernel 3.8, 2013)

- PID namespace (pid)
  - PIDs의 할당, 프로세스 리스트 및 상세 정보가 격리됨
  - 새로운 namespace는 그 형제로부터 격리되지만, “부모” namespace에서는 여전히 모든 자녀 namespace들을 볼 수 있음. (PID 숫자는 서로 다르게 보이지만)
- Network namespace (net)
  - 네트워크 인터페이스 컨트롤러 (physical or virtual), iptables 방화벽 규칙, 라우팅 테이블 등이 격리됨
  - 네트워크 namespace들은 "veth" 가상 Ethernet 장치를 통해 서로 연결될 수 있음

6 Kinds of Namespace (as of kernel 3.8, 2013)

- "UTS" namespace은 호스트네임을 변경할 수 있도록 함.
- Mount namespace (mnt)를 통하여
  - 별도의 파일시스템 레이아웃을 만들거나
  - 특정 마운트 지점을 read-only로 만들 수 있음.
- IPC namespace (ipc)
  - Namespace 간에 System V IPC 자원 (shmem, semaphore, msq) 을 격리시킴
- User namespace (user)
  - Namespace 간에 user ID를 격리시킴

Namespaces added since kernel 3.8, 2013

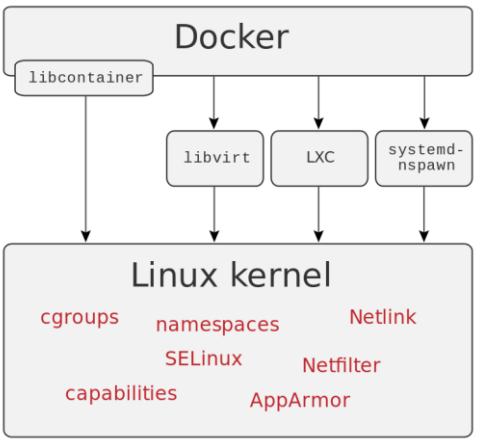
- Control Group (cgroup) – Linux 4.6, 2016
- Time – Linux 5.6, 2020
- Syslog – Proposed as of 2021

CGroups

- “Control Group”의 약자
- Linux Kernel Feature
- Resource Manager
  - CPU, Memory, disk I/O, network, etc.
  - Collection of processes
  - Limits, accounts for, and isolates the resource usage
- 컨테이너 **관리**의 기초 형태
- Author
  - V1 - Engineers at Google (Paul Menage and Rohit Seth) / Linux 2.6.24
  - V2 – Tejun Heo / Linux 4.5 (2016/03/14)

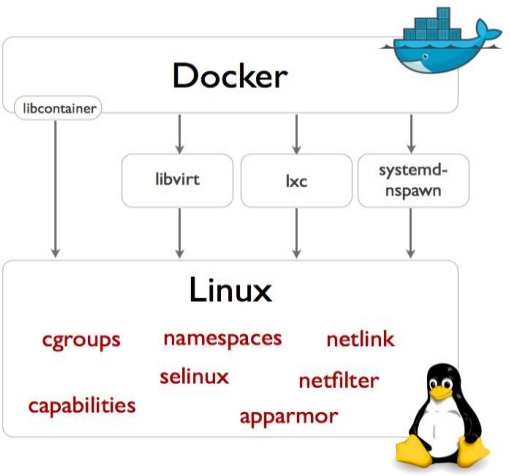
How Docker handled Namespace and CGroup

- Docker의 초기버전 (~v1.10)는 다음과 같은 **Linux 커널 기능**을 사용하기 위해 "libraries"를 사용함
  - Linux namespaces, cgroups, netfilter, iptables, capabilities, etc.
- 이러한 "libraries"에는 다음과 같은 것들이 사용됨
  - `LXC, libvirt, system-nspawn, and` **`libcontainer`** (Docker v0.9, 2014)

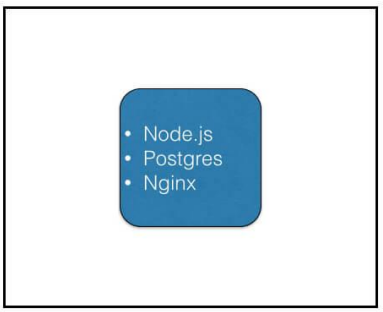


LXC and Docker (early versions < v1.11)

- 초기 버전 Docker는 LXC를 container 실행 드라이버로 사용함
- LXC 는 Docker v0.9에서 optional이 되고, libcontainer로 대체됨
- LXC 지원은 Docker v1.10에서 중단됨

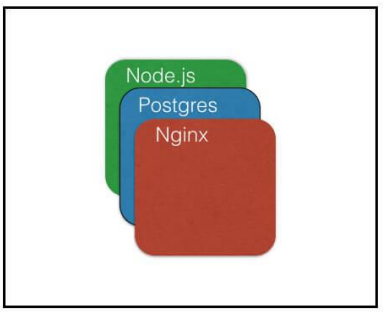


OS(system) Containers vs. App Containers



OS containers

- Meant to used as an OS - run multiple services
- No layered filesystems by default
- Built on cgroups, namespaces, native process resource isolation
- Examples - LXC, OpenVZ, Linux VServer, BSD Jails, Solaris Zones



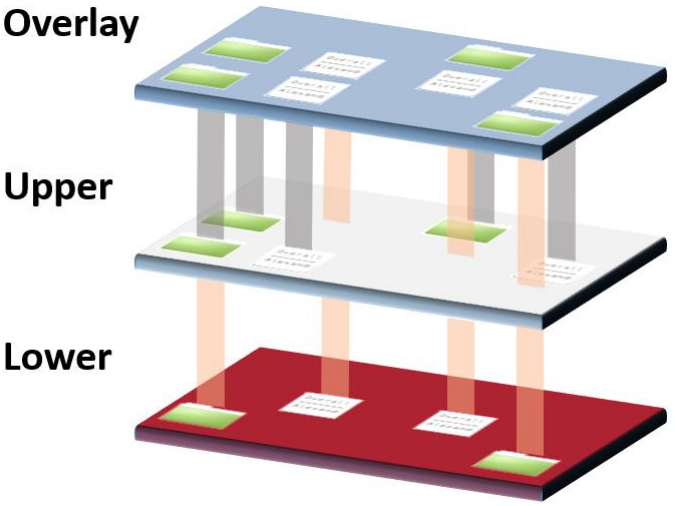
App containers

- Meant to run for a single service
- Layered filesystems
- Built on top of OS container technologies
- Examples - Docker, Rocket

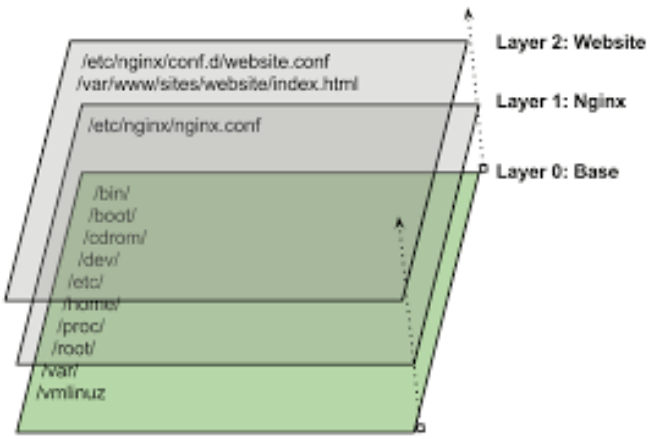
Linux Filesystem Concepts for Docker

Union Filesystem

Union FS - Concepts



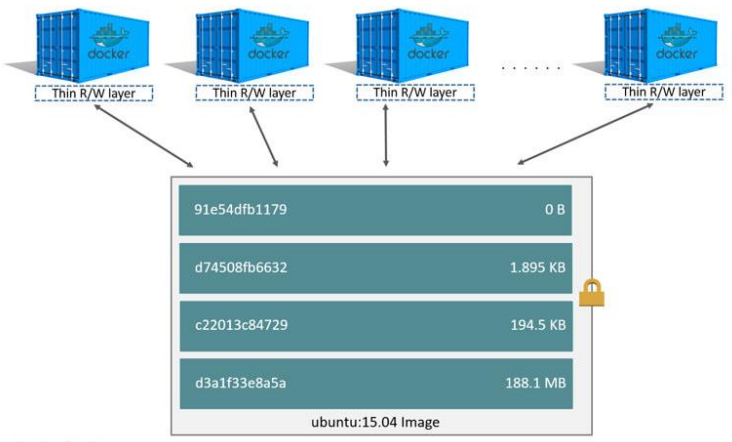
Union FS - Concepts



UnionFS, Linux and Docker

- UnionFS는 Linux, FreeBSD, NetBSD에서 구현된 파일시스템
- Linux에는 unionfs와 유사한 implementatio이 몇 가지 있음
  - UnionFS v1.x
  - UnionFS v2.x
  - AUFS
  - OverlayFS (merged to Mainline Linux kernel 3.8 on 26 Oct 2014)
- 현재 Docker에서 권장하는 Storage Driver는 Overlay2 FS
  - https://docs.docker.com/storage/storagedriver/select-storage-driver/

Docker Images / Containers / Storage Drivers



Docker Concepts

What is Docker?

- By **Docker**, we may be referring to:
  - Docker Inc., 회사이름
  - Docker 컨테이너 런타임 및 오케스트레이션 기술
  - Docker 오픈소스 프로젝트 이름 (현재는 *Moby*)

Docker Inc.

- History
  - 2010: Solomon Hykes에 의해 **dotCloud, Inc.**라는 이름으로 설립
  - 9/13/2013: **dotCloud** 와 **Red Hat** 이 전략적 제휴를 발표함
    - RedHat의 PaaS 솔루션인 OpenShift에 Docker를 통합시키기 위해
  - 10/29/2013: 회사 이름을 **Docker, Inc.**로 개명함
  - 8/4/2014: **dotCloud** 기술 및 브랜드를 **cloudControl**에 매각함
- Docker, Inc. 는 *Moby*오픈 소스 프로젝트를 관리함
- Docker, Inc. 는 Docker software를 공급함
  (Personal/Pro/Team/Business as of 2023)
- Docker, Inc. 는 매년 *DockerCon*이라 부르는 컨퍼런스를 개최함

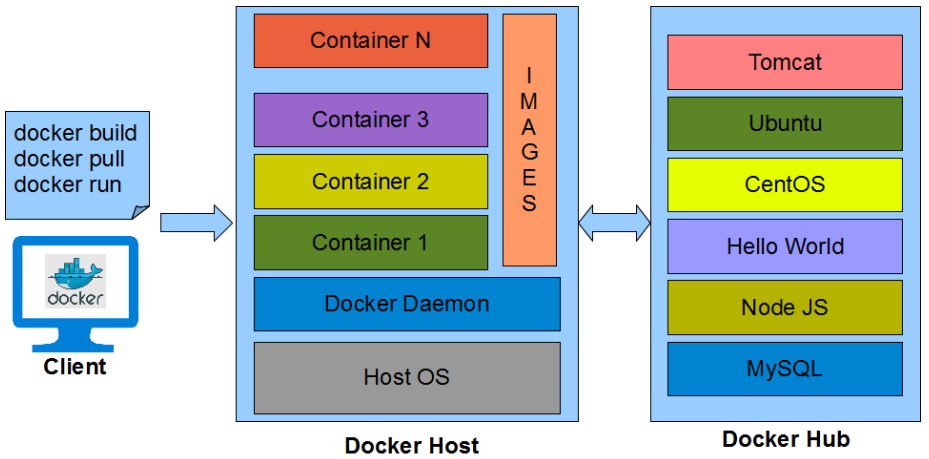
Docker (software)

- *"**Docker** is a computer program that performs operating-system-level virtualization. It was first released in 2013 and is developed by Docker, Inc."* – Wikipedia
- Linux 및 Windows에서 실행됨
- **컨테이너**를 관리하고 오케스트레이트함
- *Moby* 오픈 소스 프로젝트의 일부로서 공개적으로 개발됨
- 이 시간에 다루는 주제임

Moby Project

- https://mobyproject.org/
- 원래 이름은 *Docker 프로젝트*.
  - https://github.com/docker/docker
- 공식적으로 Moby 프로젝트로 개명됨
  - at DockerCon 2017 in Austin, TX
  - https://github.com/moby/moby
- Docker의 Upstream 코드
- 대부분의 코드는 Golang으로 쓰여짐

Docker의 동작 원리

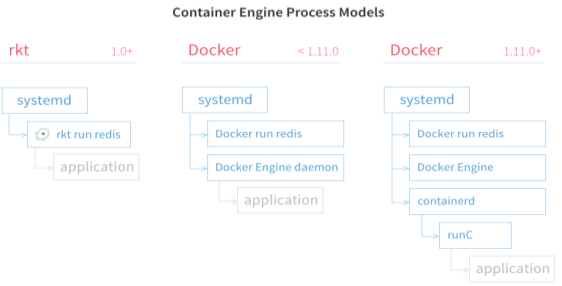


Container OS

- Container OS
  - 컨테이너 실행을 목적으로 최적화된 경량 OS
- https://blog.codeship.com/container-os-comparison/
  - Container Linux (Formerly CoreOS, https://www.coreos.com)
    - Red Hat 개발/ Google 투자
    - Gentoo Linux, Chrome OS, Chromium OS와 SDK 공유
  - RancherOS
  - Snappy Ubuntu Core OS
  - RedHat Project Atomic
    - CentOS, Fedora, RHEL의 upstream RPM을 사용하여 구성됨
  - Mesosphere DCOS
  - VMware Photon

CoreOS, appc, rkt, and OCI

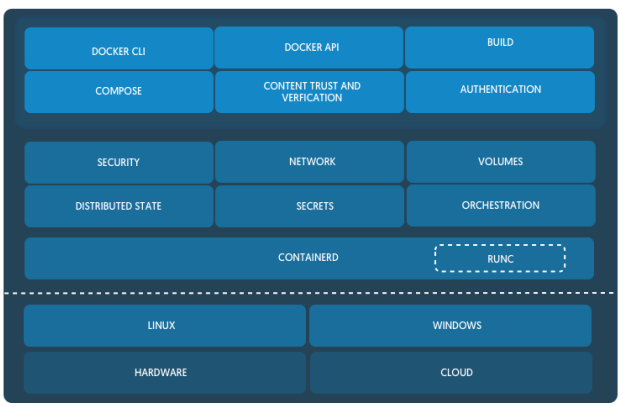
- CoreOS에서 Application Container 를 위한 새로운 표준을 정의함 **(appc)**
  - https://github.com/appc/spec
- **appc** 스펙은 자체의 **image spec** 과 **container runtime spec**을 포함함
- CoreOS는 appc에 기반하여 **rkt** 를 구현함



OPEN CONTAINER INITIATIVE

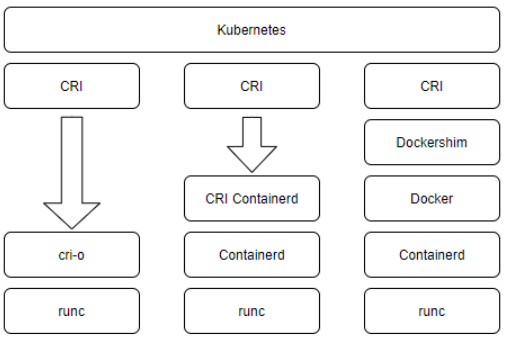
- 향후 OCI 가 설립되어 다음 표준이 정의됨:
  - **image-spec** and **runtime-spec**

Docker and containerd/runc (>= v1.11+)



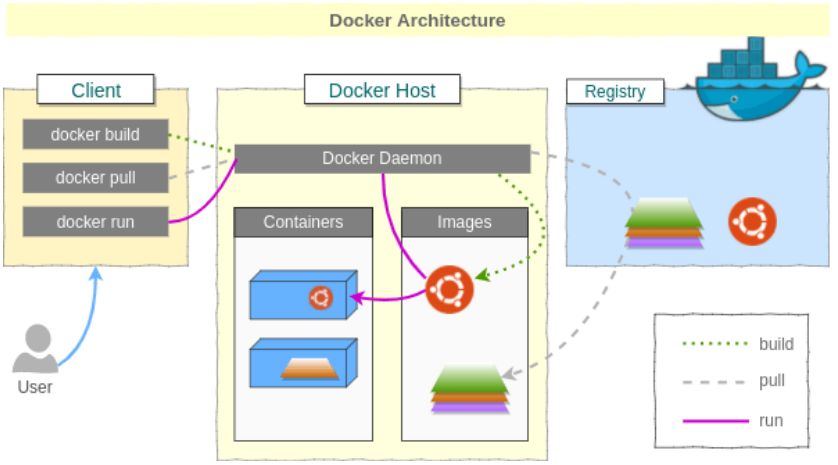
containerd

Kubernetes and Docker/Containerd/CRI-O



Docker Architecture

Docker Architecture

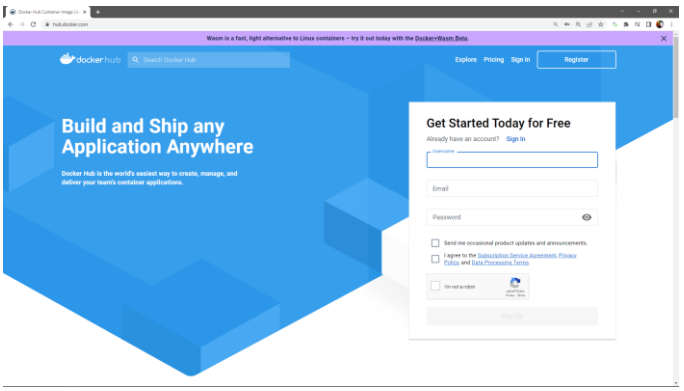


Docker Components

- Docker Software
  - Docker daemon, called `dockerd`
  - Docker client, called `docker`
- Docker objects
  - Docker containers (running instance of images)
  - Docker images
  - Swarm, multi-node Docker daemon coordination service
- Registries
  - A Docker registry 는 Docker images의 저장소
  - Docker clients 는 registries에 접속하여 사용할 이미지를 다운로드("pull") 하거나 제작한 이미지를 업로드 ("push")함.
  - Registries는 퍼블릭 혹은 프라이빗일 수 있음
  - Docker가 이미지를 찾기 위해 사용하는 기본 레지스트리는 Docker Hub (docker.io)

Docker HUB – https://hub.docker.com

- 기본 public repository
- 이미지 이름으로 검색가능
- 여러가지 필터 적용 가능
  - Official Images
  - OS
  - Architecture



Docker Theory of Operation

Using Docker

Docker CLI – working with containers

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

Docker base command

- Docker CLI has 2 different types of syntax
  - `$ docker <subcommand> <arg>`
  - `$ docker <mgmt command> <subcommand> <arg>`
  - <mgmt command> includes:
    - builder/checkpoint/config/container/image/network/node/plugin/secret/service/stack/swarm/system/trust/volume

Rules regarding Image Naming (a.k.a. Tagging)

- 이미지 이름은 항상 다음 형태로 기술됨

[registry / location /] image(repo name) [: tag]

- registry/location 정보가 생략될 경우 기본값은
  - **docker.io** (docker hub)
- tag 정보가 생략될 경우 기본값은
  - **latest** (repository 상의 다른 이미지의 버전 정보 및 작성시점 등과 전혀 상관 없음)

(Optional) Basic **docker** subcommands

- Some basic `docker` subcommands are:

```
$ docker search
$ docker pull
$ docker images
$ docker run
$ docker ps
$ docker start
$ docker restart
$ docker attach
$ docker exec
$ docker stop
$ docker rm
$ docker rmi
```

(Optional) Search and Pulling Images

- `$ docker search <keyword>`
- `$ docker image search <keyword>`

- `$ docker pull <image>`
- `$ docker image pull <image>`

(Optional) List Images that are in local repo

- List Images that are in local repo
  - `$ docker images`
  - `$ docker image ls`
- Examine image meta data
  - `$ docker image inspect`

(Optional) Run Images with default command

- When default command does not need TTY input
  - `$ docker run <image>`
  - `$ docker container run <image>`
- If image is not found in local repo, docker run command pulls image first.
- When default command requires TTY input (e.g. shells)
  - `$ docker run –it <image>`
  - `$ docker container run –it <image>`

(Optional) Listing Containers (running, exited)

- Listing Running containers
  - `$ docker ps`
- Listing Exited containers
  - `$ docker ps -a`
- Listing container ids only
  - `$ docker ps –q`
  - `$ docker ps –aq`

(Optional) When starting/restarting default command (pid 1) from exited container

- Start default command with pid 1 from **dead** container

```
$ docker ps -a
$ docker start <container id or name>
$ docker container start <container id or name>
```

- Restart default command with pid 1 from **alive** container

```
$ docker ps
$ docker restart <container id or name>
$ docker container restart <container id or name>
```

(Optional) Attaching/Detaching client's TTY to/from container

- Detach from attached TTY
  - `(inside container) # ^P^Q`
- Run a container with TTY detached:
  - `$ docker run –d <image>`
  - `$ docker container run –d <image>`
- Attach client's TTY to running container:
  - `$ docker attach <container>`
  - `$ docker container attach <container>`

(Optional) Run additional command on running container

- Run additional command (e.g. ps) on running container environment
  - `$ docker exec <container> <cmd> [ARG…]`
  - `$ docker container exec <container> <cmd> [ARG…]`
- When running command that needs TTY attachment (e.g. bash)
  - `$ docker exec –it <container> <cmd> [ARG…]`
  - `$ docker container exec –it <container> <cmd> [ARG…]`

(Optional) Stopping running container

- Stop running container
  - `$ docker stop <container>`
  - `$ docker container stop <container>`
  - `$ docker ps`
  - `$ docker ps -a`

(Optional) Deleting exited(stopped) containers

- Docker keeps R/W layers that every container created. This may consume disk space of Docker host. (/var/lib/docker)
- To delete R/W layers of stopped(exited) containers,
  - `$ docker ps –a`
  - `$ docker rm <container>`
- To delete all R/W layers of stopped(exited) containers,
  - `$ docker rm $(docker ps –aq)`
  - `$ docker container prune`

(Optional) Deleting Images that are on local repo

- Delete image from local repo
  - `$ docker rmi <image>`
  - `$ docker image rm <image>`
- Delete ALL image from local repo
  - `$ docker image prune`
- Note: There are other prune subcommands as well such as:
  - `$ docker volume prune`
  - `$ docker network prune`
  - `$ docker system prune`

(Optional) Working with Images

- https://docs.docker.com/engine/reference/commandline/docker/
- Working with images
  - To display history of actions taken to an image: `$ docker history`
  - To copy files from container to local file system: `$ docker cp`
  - To build new image file based on current container: `$ docker commit`
  - To display delta from original image: `$ docker diff`
  - To display detail information: `$ docker inspect`

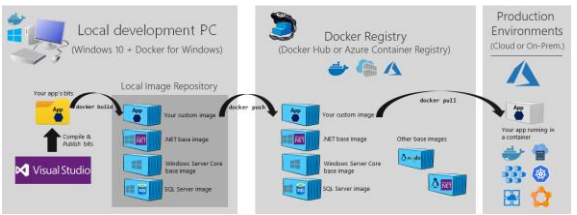
Uploading image

- You can upload images to:
  - The Docker Hub
  - Other Public registries:
    - Google: https://cloud.google.com/container-registry
    - Amazon: https://aws.amazon.com/ecr/
  - Private Docker Registry

Sharing Images on Docker Hub

- Docker Hub에 이미지를 공유하기 위해서는:
  1. Docker Hub에 repository를 생성
  2. Repository property 속성 (public / private) 설정
  3. 생성한 이미지에 tag한 후 Docker Hub에 push (from Dev Docker Engine)
  4. Docker Hub으로부터 Pull 하여 이미지 실행 (from Ops Docker Engine)

Build(Commit) and Push



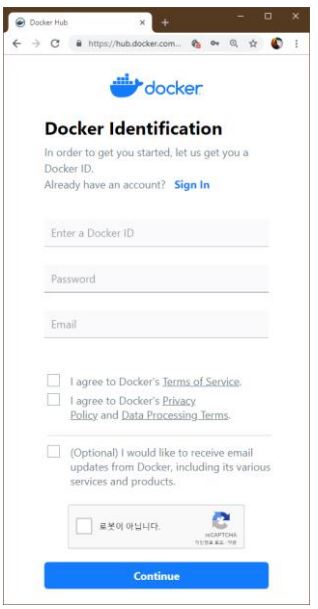
Pull and Run

https://github.com/dotnet-architecture/eShopModernizing/wiki/03.-Publishing-your-Windows-Container-images-into-a-Docker-Registry

(Optional) Creating Repository on Docker Hub

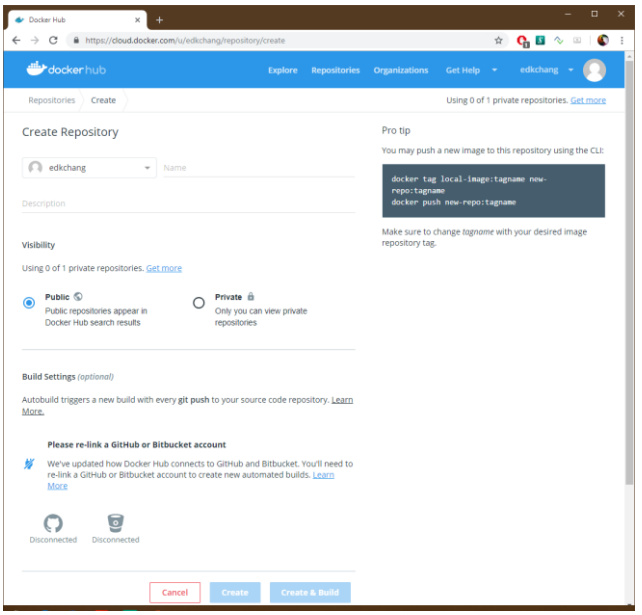
1.Sign up for an account
- https://hub.docker.com/
- Docker ID
- Password
- Email

2.Verify email and add a repository
- Open your email inbox.
- Open the email and click the Confirm Your Email button.
- The browser opens the **Create Repository** page



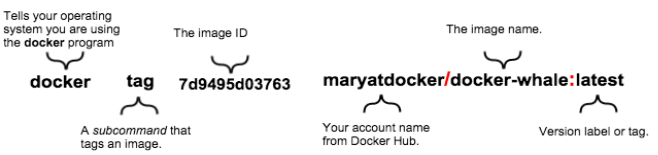
(Optional) Setting Repository Property

3. Provide a Repository Name and Short Description.
4. Make sure Visibility is set to Public.
5. Press Create when you are done.



(Optional) Tag and Push the image

1. Find image id by typing:
   ```
   $ docker images
   ```
2. Fill REPOSITORY and TAG field by:
   ```
   $ docker tag <image>
   ```



3. Verify with:
   ```
   $ docker images
   ```

(Optional) Tag and Push the image (cont'd)

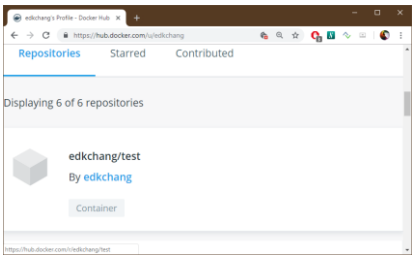
4. Log in to Docker Hub by login command:

```
$ docker login \
    --username=<Your DockerID> \
    --email=<Your email>
```

5. Push your image to new repository:

```
$ docker push \
   <Your DockerID>/<Image>
```

6. Return to your profile on Docker Hub to see your new image:



(Optional) Pull your new image

- In order to pull new image from registry, you have to delete old image from local image store.
- List the images you currently have on your local machine by:

```
$ docker images
```

- Remove images by:

```
$ docker rmi –f <image_name or image_id>
```

- Pull and load a new image by:

```
$ docker run yourusername/imagename
```

- Verify it is running:

```
$ docker ps
```

Thank You!