Containers and Virtualization Technologies

Hideo Suzumiya (Gaoyang Wei)

College of Information Engineering (College of Artificial Intelligence)

March 2, 2023

Outline

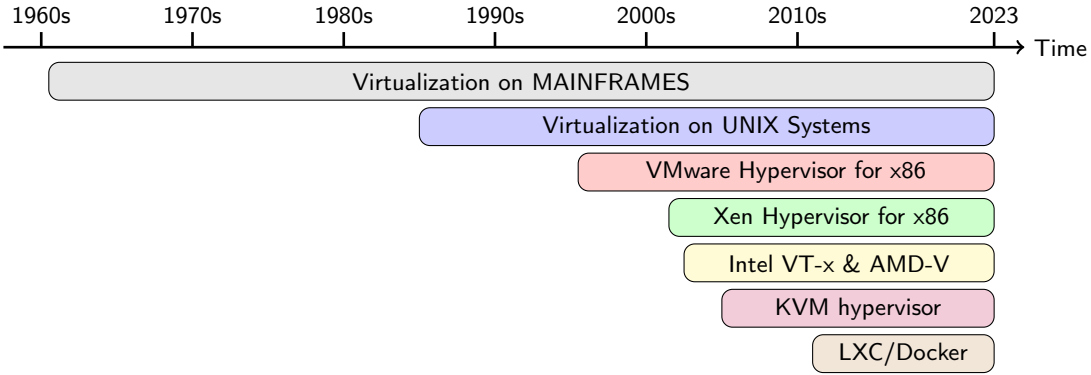
What is Virtualization ?

Virtualization is a technology that allows multiple virtual instances of hardware, operating systems, and software applications to run on a single physical server, creating a virtual environment that appears to be a separate and independent computer system.

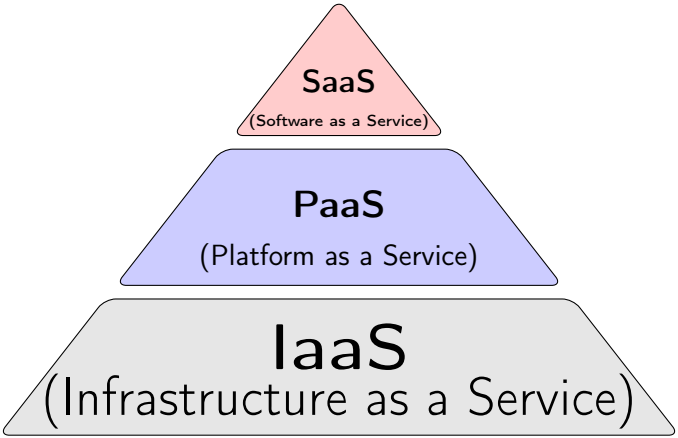
5 Levels of Virtualization Technologies

- Instruction Set Architecture Level *e.g.* QEMU
- Hardware Abstraction Level *e.g.* VMWare ESXi, Microsoft Hyper-V, KVM
- Operating System Level *e.g.* LXC, Docker
- Programming Language Level *e.g.* JVM, Microsoft dotNet
- Library Level *e.g.* Wine, WSL1

Evolution of Virtualization



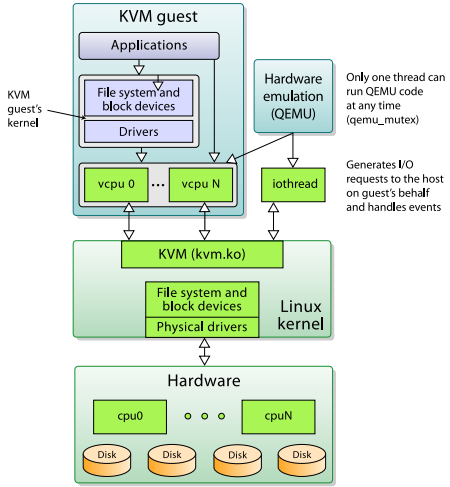
Cloud Computing Models

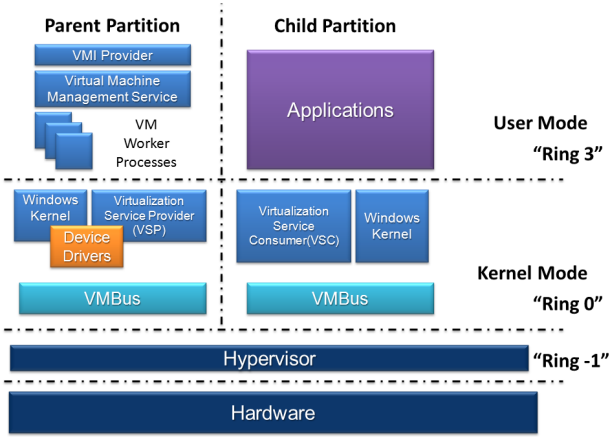


Virtualization

- Virtualization enables multiple virtual machines (VMs) to run on a single physical server, each with its own operating system and applications.
- Virtualization provides a way to consolidate servers, reduce hardware costs, and increase resource utilization.
- Popular virtualization platforms include VMware, Hyper-V, and Kernel-based Virtual Machine.

KVM & Hyper-V Architectures





Virtualization

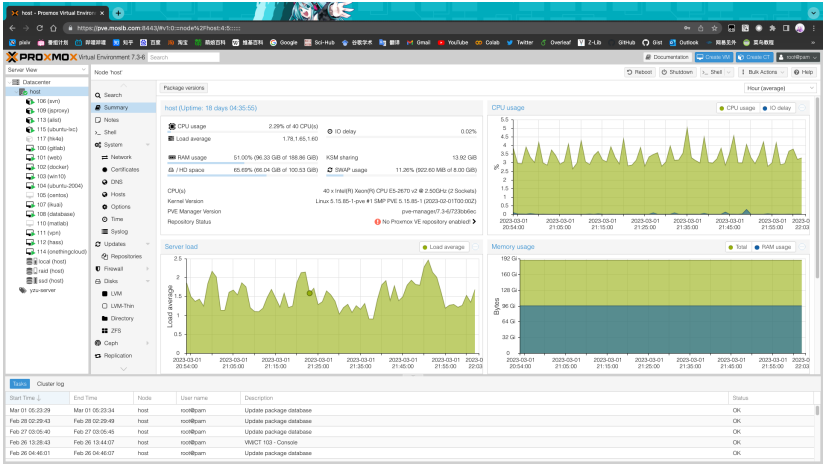


Figure: Proxmox VE 7.3 based on KVM and LXC

What is Container?

- Containers provide a way to package and deploy applications in a lightweight, portable, and reproducible way.
- Containers are Operating System Level VMs!
- Docker is a popular containerization platform that enables developers to build, ship, and run distributed applications in containers.

Philosophy of Docker

- Write once, run everywhere.
- One container runs one application.

Containers

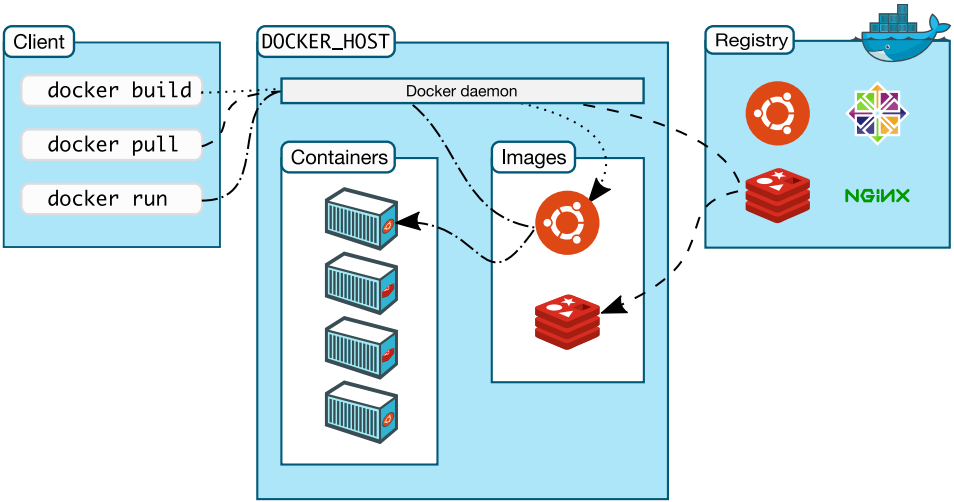
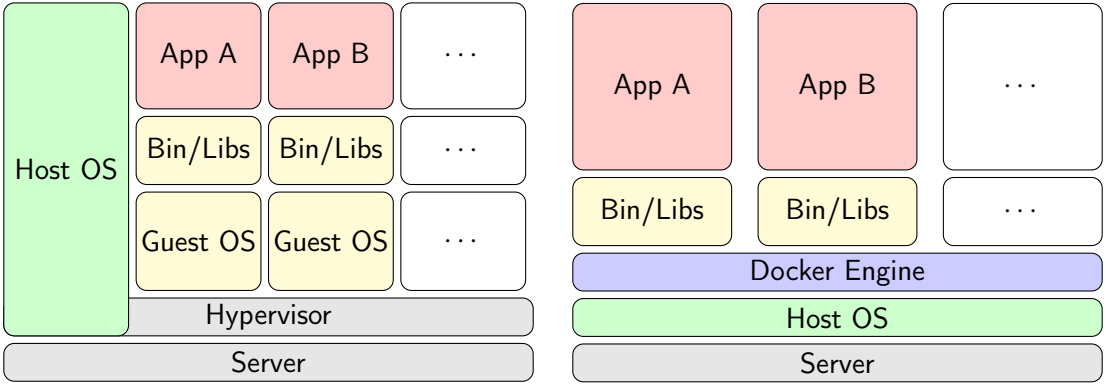


Figure: Docker Architecture

Containers V.S. VMs



Comparison

- Containers are faster, more lightweight, and more efficient than VMs.
- VMs provide greater isolation and security than containers.
- Containers are better suited for microservices-based architectures, while VMs are better suited for legacy applications and monolithic architectures.

Attribute	Container	VM
Start	Seconds	Minutes
Size	10s of Megabytes	10s of Gigabytes
Performance	Close to native	Weaker
Supported Quantity	1000s of containers	10s of Guests

Table: The comparison

Steps of installing Docker on linux

Running Docker on Windows & macOS is **NOT** recommended!

- Change source repo

```
export DOWNLOAD_URL="https://mirrors.bfsu.edu.cn/docker-ce"
```

- Use auto installation script

```
curl -fsSL https://get.docker.com/ | sh
```

- Or

```
wget -O- https://get.docker.com/ | sh
```

- Change DockerHub repo

```
touch /etc/docker/daemon.json
{
  "registry-mirrors": [
    "https://hub-mirror.c.163.com",
    "https://mirror.baidubce.com" ]
}
```

Install Docker

- Reload systemd services
  ```
  sudo systemctl daemon-reload
  sudo systemctl restart docker
  ```
- Execute `docker info` and if you see

  ```
  Registry Mirrors:
   https://hub-mirror.c.163.com/
  ```

  The configuration has been applied.

Container Management

Command	Description
`docker create` *image* [*command*]	Crete the container
`docker run` *image* [*command*]	= `create` + `start`
`docker start` *container* ···	Start the container
`docker stop` *container* ···	Graceful stop
`docker kill` *container* ···	Kill (SIGKILL) the container
`docker restart` *container* ···	= `stop` + `start`
`docker pause` *container* ···	Suspend the container
`docker unpause` *container* ···	Resume the container
`docker rm` [`-f`] *container* ···	Destroy the container

Container Inspection

Command	Description
`docker ps`	List running containers
`docker ps -a`	Lista all containers
`docker logs [ -f ]` *container* $\cdots$	Show the container output (stdout & stderr)
`docker top` *container* [*ps options*]	List the process running inside the container
`docker diff` *container*	Show the difference with images (modified files)
`docker inspect` *container* $\cdots$	Show low-level infos (in `json` format)

Container intertraction

Command	Description
`docker attach` *container*	attach to a running container (stdin/stdout/stderr)
`docker cp` *container:path hostpath*|-	Copy files from container
`docker cp` *hostpath*|- *container:path*	Copy files to container
`docker export` *container*	export the content of the container (Tar archive)
`docker exec` *container* $\cdots$	Run a command in an existing container
`docker wait` *container*	Wait until the container terminates and return the exit code
`docker commit` *container image*	Commit a new docker image (snapshot or container)

Image management

Command	Description
`docker images`	List all local images
`docker history` *image*	Show image history (list of ancestors)
`docker inspect` *image* ⋯	Show low-level infos (in `json` format)
`docker tag` *image tag*	Tag an image
`docker commit` *container image*	Create image from container
`docker import` *url*|- [*tag*]	Create image from tarball
`docker rmi` *image* ⋯	Delete images

Image transfer

Command	Description
`docker pull` *repo*[:*tag*] ⋯	Pull an image/repo from a registry
`docker push` *repo*[:*tag*] ⋯	Push an image/repo from a registry
`docker search` *text*	Search an image from the official registry
`docker login` ⋯	Login to a registry
`docker logout` ⋯	Logout from a registry

Table: Using the registry API

Image transfer

Command	Description
`docker save` *repo*[:*tag*] $\cdots$	Export an image/repo as a tarball
`docker load`	Load image from a tarball
`docker-ssh` $\cdots$	Proposed script to transfer images between 2 daemons over ssh

Table: Manual transfer

Builder

Command	Description
`FROM` *image*|*scratch*	Base image for the build
`MAINTAINER` *email*	Name of maintainer (metadata)
`COPY` *path dst*	Copy *path* from the context into the container at location *dst*
`ADD` *src dst*	Same as `COPY` but untar archives and accepts http urls
`RUN` *args* $\cdots$	Run an arbitrary command inside the container
`USER` *name*	Set the default username
`WORKDIR` *path*	Set the default working directory
`CMD` *args* $\cdots$	Set the default command
`ENV` *name value*	Set environment variables

Conclusion

- Containerization and virtualization are both important technologies for modern application development and deployment.
- Containers and VMs have different strengths and weaknesses, and choosing the right technology depends on the specific requirements of the application and infrastructure.

Conclusion

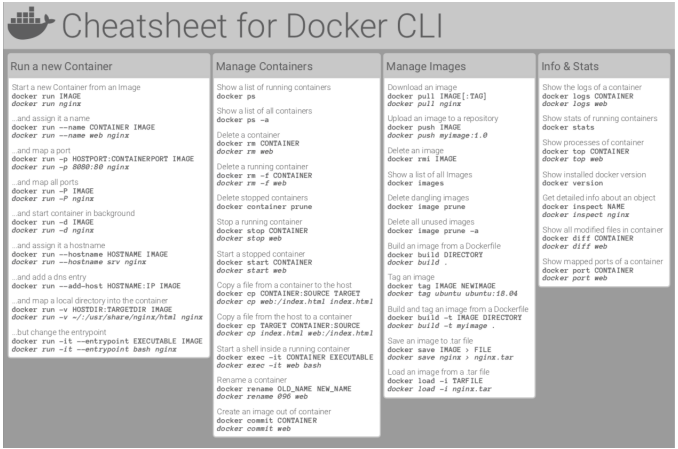


Figure: Docker Cheatsheet:
https://raw.githubusercontent.com/sangam14/dockercheatsheets/master/dockercheatsheet8.png

Thanks!