gepardec

redhat

PREMIER

BUSINESS PARTNER

Docker 101

START >

Author: Ing. Thomas Herzog B.Sc / Version 1.0

Docker Basics

- What is Docker ?
- Why using Docker ?
- Containerization vs Virtualization
- Docker Architecture
  - Docker Engine
  - Docker Machine
  - Base Technologies
    (LXC, cgroups, namespaces, UnionFS)
- Docker Image/Image-Registry/Container
- Docker Volume/Network
- Setup Linux/Windows/MAC

https://docker.com

What is Docker ?

- Docker is a software based on Linux Containers *(LXC)*
- LXC is a kernel technology
- LXC uses Linux Kernel-Features
  - cgroups, namespaces
- Docker utilizes LXC
- Docker is application focused, LXC is machine focused
- Docker more convenient as plain LXC

Docker is the tool for LXC and not the container technology itself !

Why using Docker ?

- Application environments represented by Docker Images are
  - reproducible, // Run on any Docker Host
  - distributable, // Share Docker Image via registry
  - consistent, // Once defined, stays always the same
  - automatable, // Skripting, Kubernetes, Openshift
  - and reliable. // Docker Images are immutable
- Processes running in Docker Containers are
  - isolated, // Cannot interfere with each other
  - have their own libs // No collisions with binaries or binary versions
  - and resources limits. // Cannot consume more than assigned

Why using Docker ?

- Application binaries are not bound to Host OS. *(only Kernel)*
- Use any Linux-Distro you like
- Orchestrate lots of Docker Containers *(Swarm, Kubernetes, Openshift, ...)*
    - Multiple Instances *(Replicas)*
    - Release Workflows *(Blue-Green, Canary Releases)*
    - Management of configurations and secrets
    - Scale out to other Cloud providers
- Deploy on any Cloud *(Azure, Google, AWS, ...)*
    - All support Docker and Kubernetes

Why using Docker ?

- What happens if such a command is executed ?

    -
      ```
      while true; do mkdir badDir && cd badDir; done;
      ```
- Indefinitely creates a directory and steps into it
- The process occupies all available resources
- All other processes are going to starve
- Systems will become unusable

With Docker, the system wouldn’t be affected !

Why using Docker ?

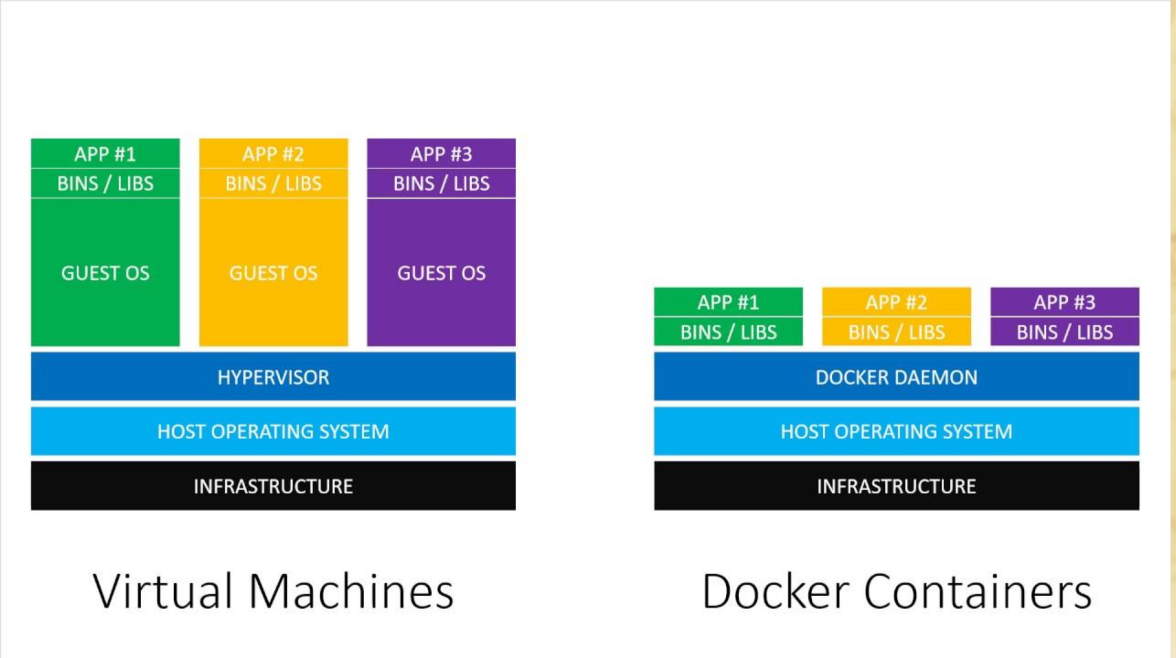
- What happens if such a command is executed ?
    - `rm -rf /`
- Recursively deletes the whole file-system
- All other processes are going to die
- Systems will become unusable

With Docker, only the Docker Container file-system would be affected !

Containerization vs. Virtualization

- Virtual Machines
  - Full blown OS for hosting application
  - Overhead of running OS
  - Kernel Emulation necessary
  - Application bound to host OS binaries
- Containers
  - Container provides necessary binaries
  - No OS overhead
  - Application uses host kernel
  - Application brings in own binaries

    (Compatible to host kernel)



https://www.youtube.com/watch?v=TvnZTi_gaNc

Containerization vs. Virtualization

- Linux wasn’t capable of providing isolation *(cgroups v1 2007)*
- Therefore, VMs were used to isolate applications
- VM has too much overhead
- VMs are too inflexible for hosting applications
- Nevertheless, the Cloud runs on VMs
    - But, one VM with Docker, not for each application.

Docker Architecture

- Docker Host *(Daemon)*

  Listens for API requests

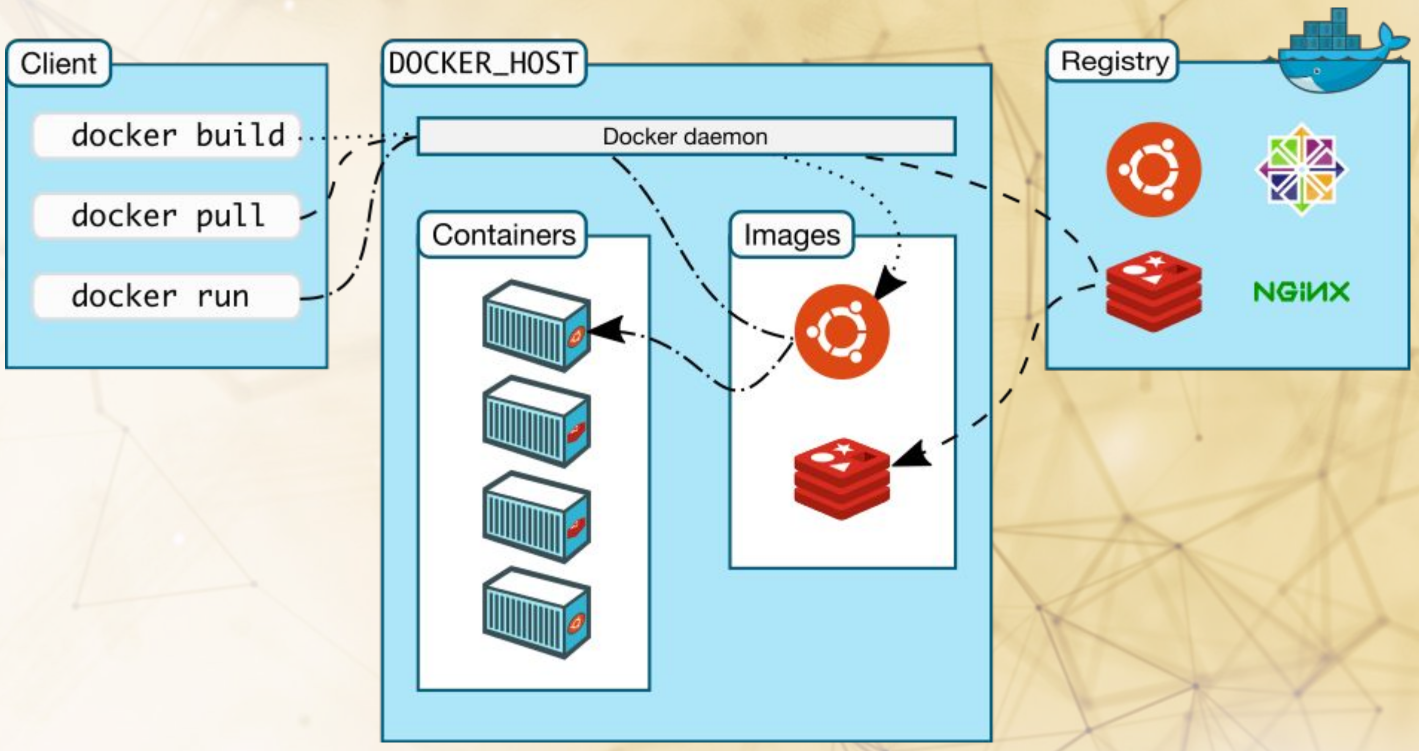
- Docker Client

  Executes commands via

  REST-API

- Docker Registry

  Stores Docker Images



https://docs.docker.com/engine/docker-overview

Docker Architecture - Docker Engine

- Docker Daemon

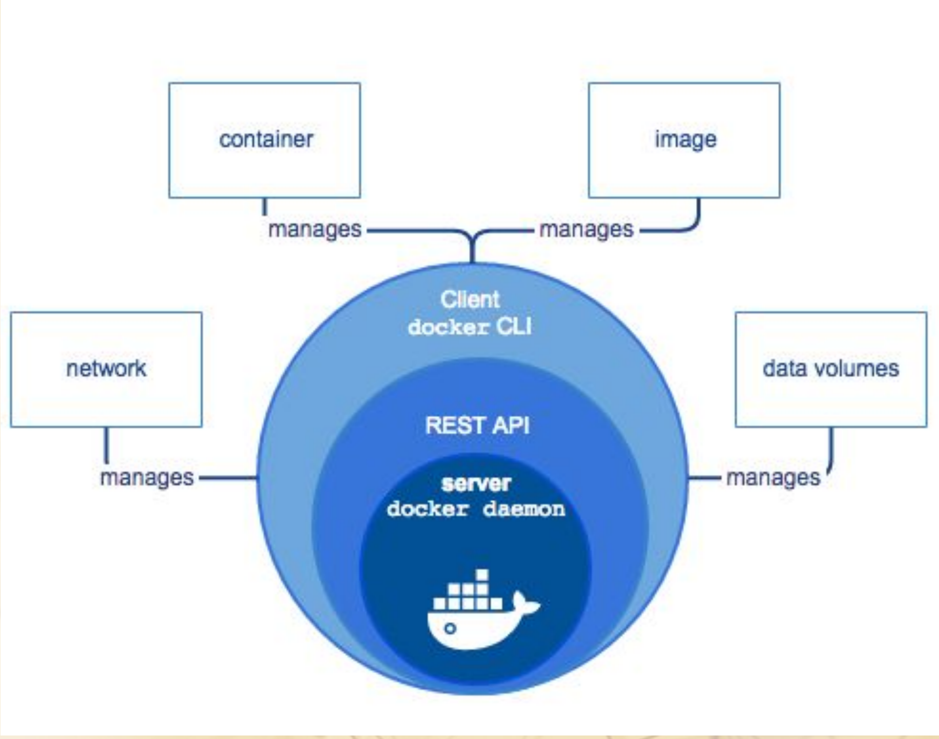
  Listens for API requests

- Docker REST-API

  Exposes the Docker Daemon

- Docker CLI

  Wrapper for Docker REST-API

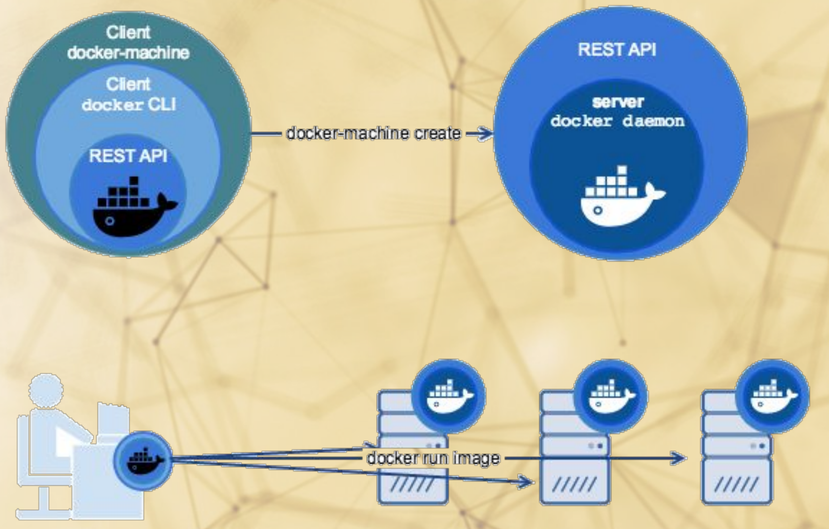


https://docs.docker.com/engine/docker-overview

Communication with the Docker Daemon is always performed via REST !!!

Docker Architecture - Docker Machine

- Tool for provisioning Docker Hosts
  - locally *(localhost, VM)*
  - remote *(Virtual Server in the cloud)*
- Create/Manage/Delete Docker Engines
- Execute commands on Docker Host
- Especially used in clustered environments
- Was used for Windows Docker-Toolbox



https://docs.docker.com/engine/docker-overview

Docker Architecture - **LXC**

- LXC is the containment feature of the Linux Kernel
- LXC is the actual container technology
- LXC uses cgroups and namespaces for
  - process isolation
  - and process resource management
- LXC is too low level and inconvenient for developers
- Plain LXC rarely used by developers, mostly via Docker
- Does not require any setup

Docker Architecture - **cgroups**

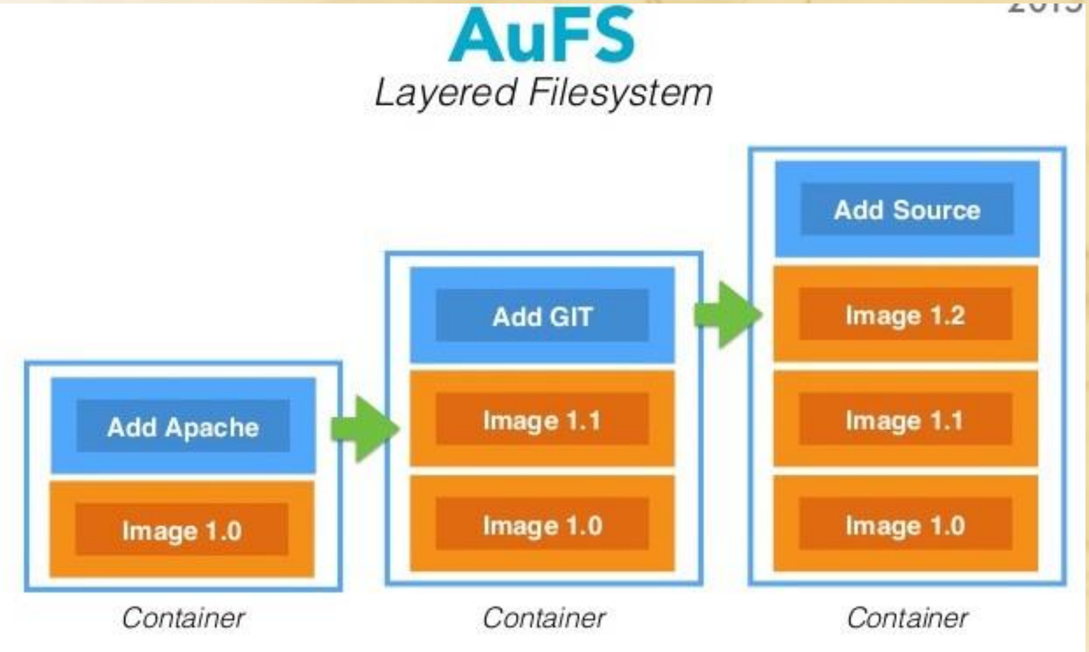
- cgroups (Control-Groups) is a Linux Kernel-Feature
- Started development in 2006
- Merged into Linux Kernel 2.6.24 which was released in 2008
- cgroups v2 (2016) replaced cgroups v1 (2007)
- Features:
  - Resource Limiting // Control how many resources a process can use
  - Prioritization // Prioritize the usage of process resources
  - Accounting // Measure of process resource consumption
  - Control // Allows to freeze and restart a group of processes

Docker Architecture - **namespaces**

- Namespaces is a Linux Kernel-Feature
- Isolates access to resources of process groups
- One process group cannot see resources of another process group
- Namespaces:
  - PID // Isolates the allocation of process identifiers (PIDs)
  - Network // Isolates physical or virtual network resources
  - IPC // Isolates the inter-process communication (IPC) between namespaces
  - User // Isolates the User-Ids between namespaces
  - ...

Docker Architecture - **UnionFS**

- File-system is layered
- Copy-on-Write for modified or new files
- Image fs-layers are ready-only
- Create container == Add fs-layer
- Delete container == Delete fs-layer
- Docker Storage-Driver ensures unified view to layered file-system



http://www.slideshare.net/FabioFerrari31/docker-containers-talk-linux-day-2015

Docker Image

- Defined by a Dockerfile
- Layer per Directive
  - `docker history <IMAGE>`
- Layer represented by hash
- Immutable Container-Template
- Flat hierarchy of images
- Images can be distributed
- Infrastructure as Code *(IaC)*

Added/Copied secrets stay in fs-layer !!

```
FROM library/java:8-alpine

MAINTAINER Thomas Herzog <thomas.herzog@gepardec.com>

ARG VERSION

LABEL name=”MyApp” \
          run='docker run --name <NAME> \
                 -p <HOST_PORT>:8080 \
                 -v <HOST_VOLUME>:/install/data \
                -d  \
               <IMAGE>'

WORKDIR /install

COPY ./app-${VERSION}.jar ./app.jar

PORT 8080
VOLUME /install/data

CMD [“java”, “-jar”, “app.jar”]
```

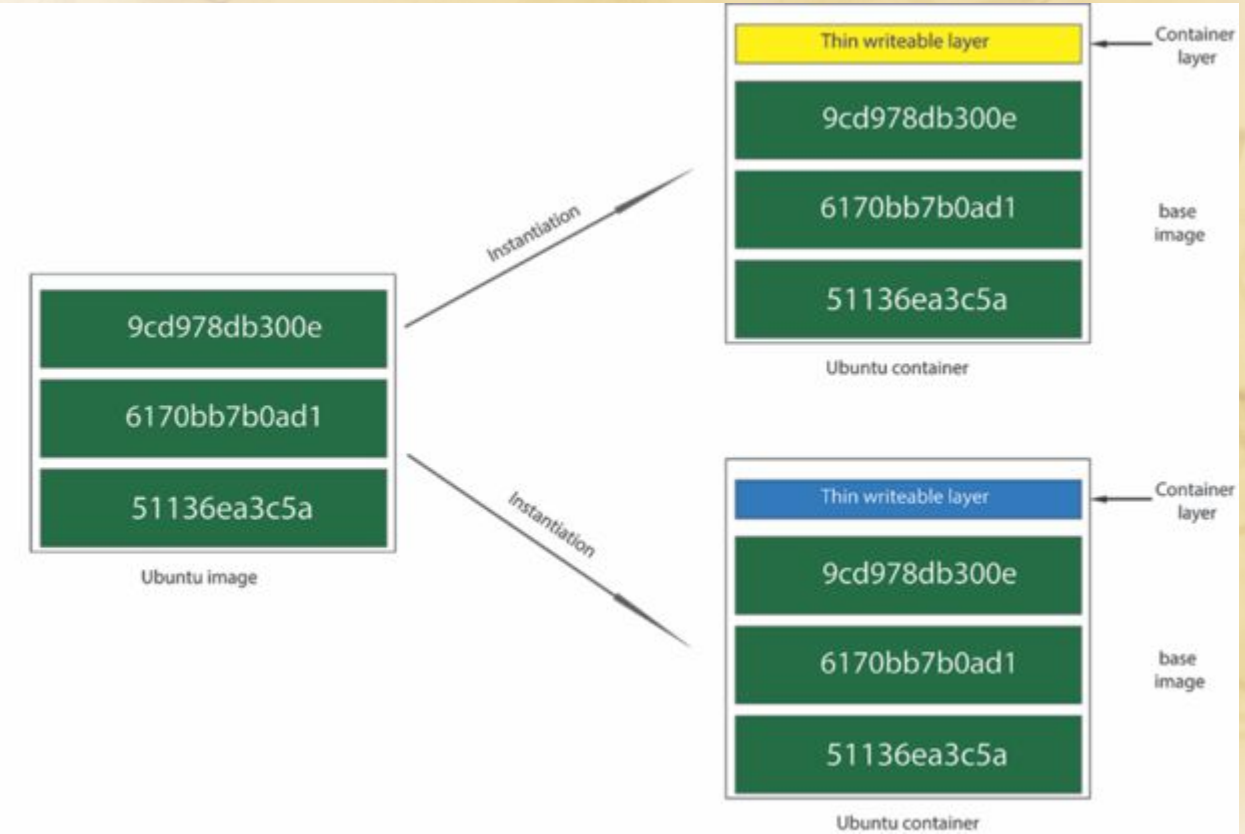
Docker Image-Registry

- Repository for Docker Images
  - Nexus 3, JFrog, ...
  - Docker Container *(https://hub.docker.com/_/registry/)*
- Images represented by tags *([ip:port/]namespace/image:version)*
  - `[localhost:5000/]library/java:8-alpine`
- Tags are not immutable, can be overwritten
- Image-Instance can be referenced via Image-Id
  - `[localhost:5000/]library/java@sha256288efb98b02870`

Added/Copied secrets still in fs-layer and are shared as well !!

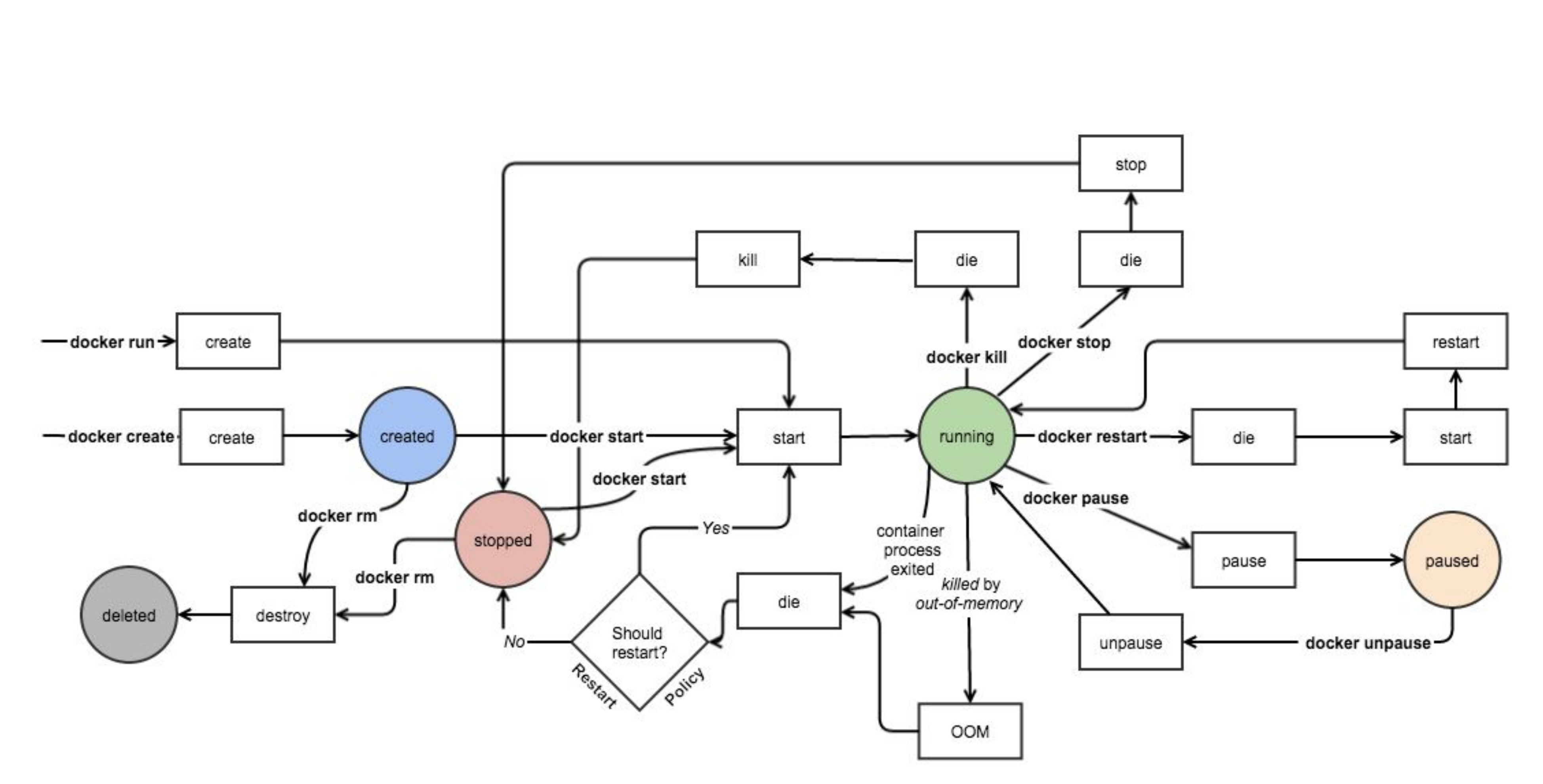
Docker Container

- Instance of Image *(Template)*
- Thin writable fs-layer appended
- Changes cause Copy-on-Write
- Two Containers have:
  - isolated processes
  - isolated namespaces
  - isolated file-systems
  - different volume mapping
  - different port mapping
  - different resource limits



https://www.infoworld.com/article/3077875/linux/containers-101-docker-fundamentals.html

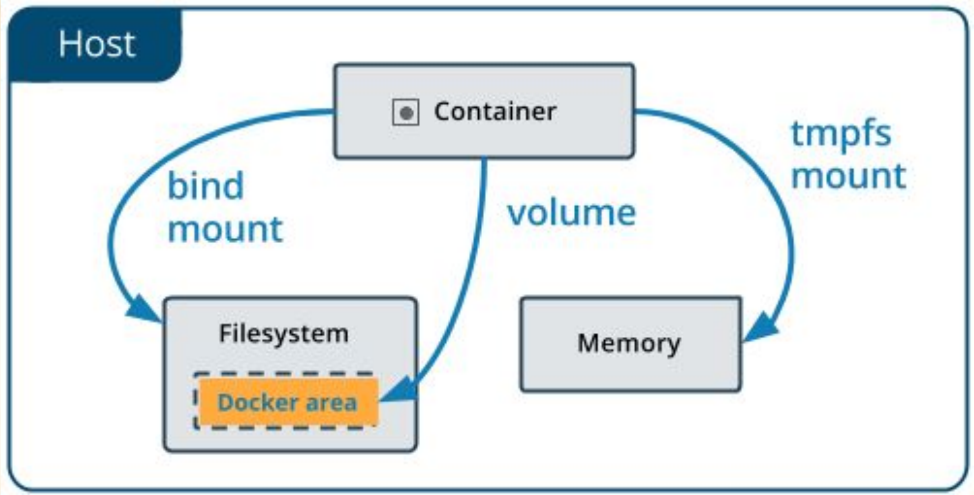
Docker Container Lifecycle



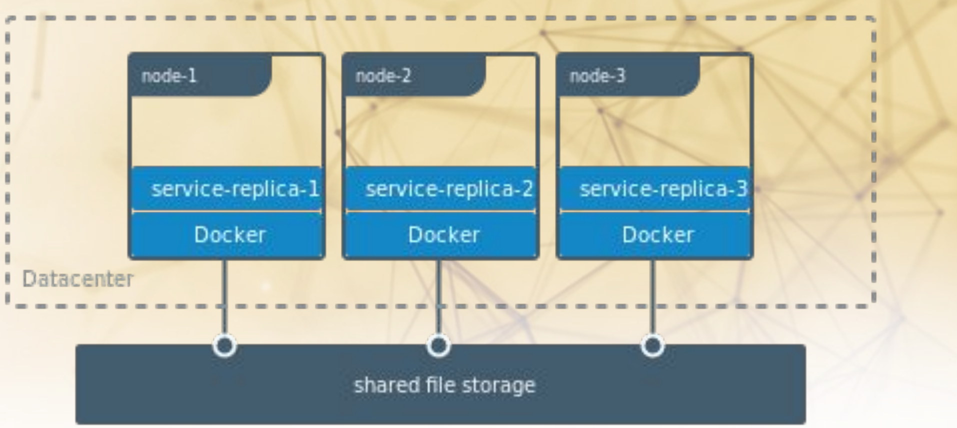
http://docker-saigon.github.io/post/Docker-Internals/

Docker Volumes

- Map container directory/file to host
- Keeps container data persistent
- Better than bind mount because:
  - Not depending on host directory structure
  - Docker CLI for volume management
  - Shareable between Linux/Windows/Containers
  - External storage support
  - Several drivers available *(NFS, AWS, Azure)*
  - Pre-population of data by a container
  - No UID/GID problems
- Never deleted automatically



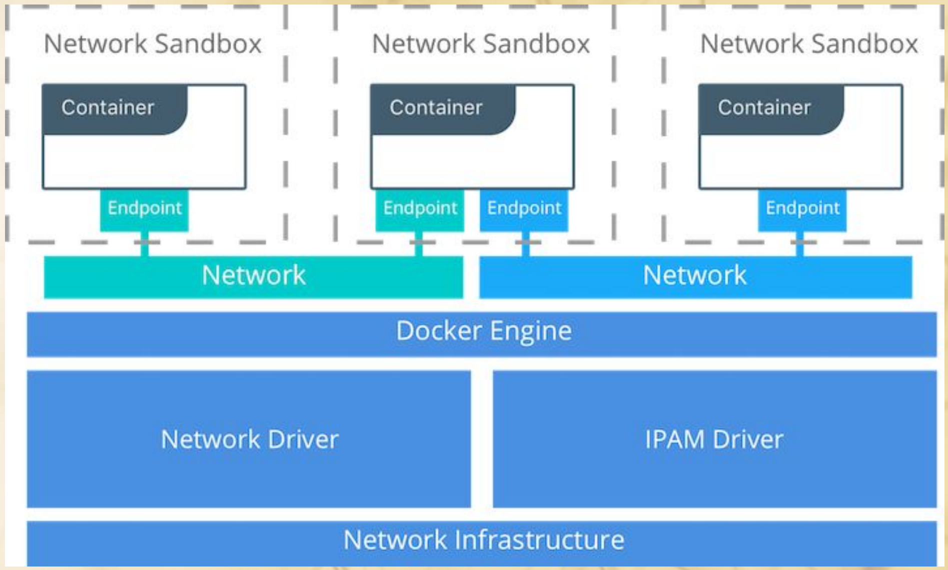
https://docs.docker.com/storage/volumes/#share-data-among-machines



https://docs.docker.com/storage/volumes/#share-data-among-machines

Docker Network

- Container Networking Model *(CNM, of Docker)*
  - **Sandbox**
    Stores all network config *(Linux namespace)*
  - **Endpoint**
    Connects Sandbox to Network
  - **Network**
    Collection of Endpoints, which can communicate
- Several drivers are available
  - **host / bridge / overlay / none**
  - **custom**
- Implemented with libnetwork



https://success.docker.com/article/networking

https://github.com/docker/libnetwork

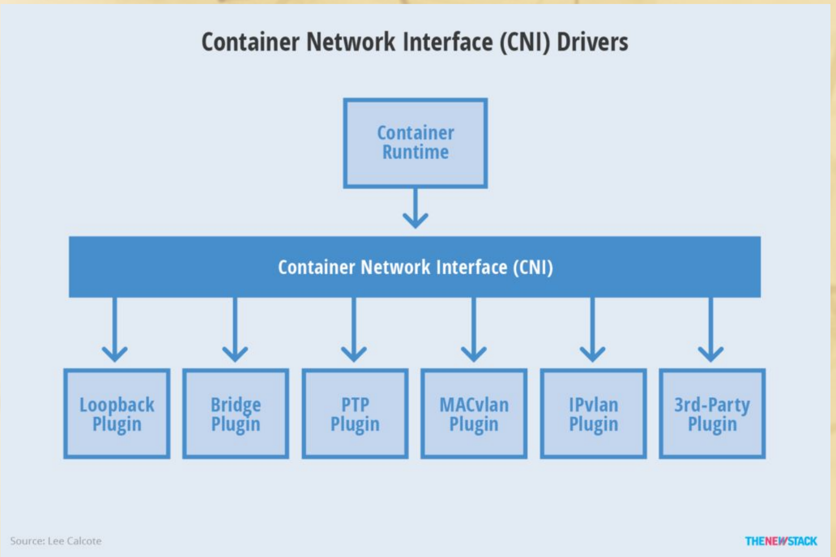
Docker Network

- Container Networking Interface *(CNI, of CNCF)*
- Specification for container networking
- Plugable networking interface
- Also provides libraries for writing plugins
- Third party drivers available

  (Amazon ECS, Weave, ...)
- Multiple container runtime support

  (Apache Mesos, Kubernetes, Openshift, ...)

https://github.com/containernetworking/cni



https://blog.gingergeek.com/2016/09/the-container-networking-landscape-cni-from-coreos-and-cnm-from-docker/

Setup for Linux/Windows/MAC

- Fo Linux go to:

  https://docs.docker.com/install/linux/docker-ce/centos/
  https://docs.docker.com/install/linux/docker-ce/fedora/
  https://docs.docker.com/install/linux/docker-ce/ubuntu/
  https://docs.docker.com/install/linux/docker-ce/debian/
  https://docs.docker.com/install/linux/docker-ce/binaries/ *(If your linux distro isn't listed here)*
- For Windows go to:

  https://docs.docker.com/docker-for-windows/install/
- For MAC go to:

  https://docs.docker.com/docker-for-mac/install/
- Docker Training sources available at:
  - https://github.com/Gepardec/docker-training