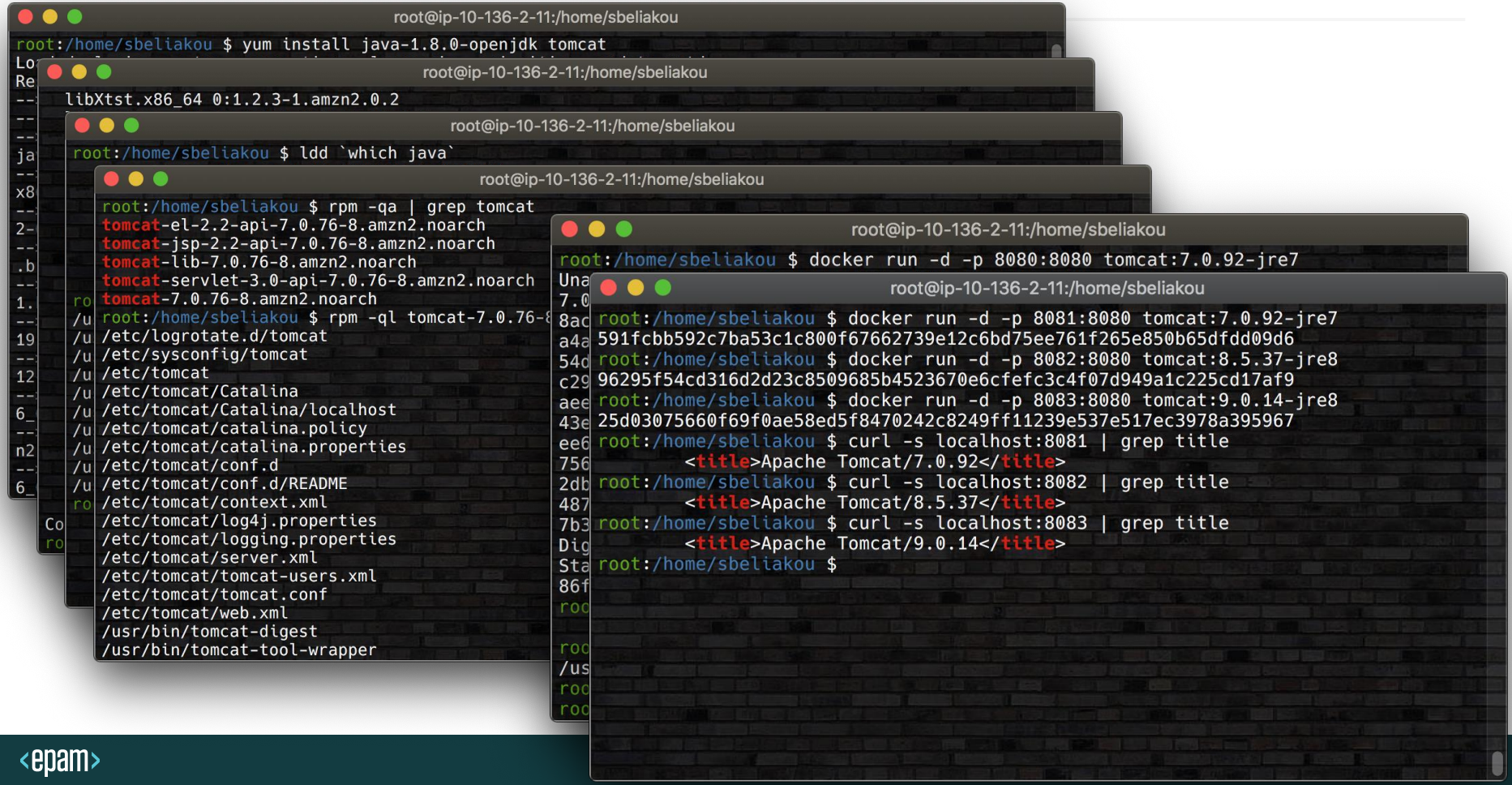
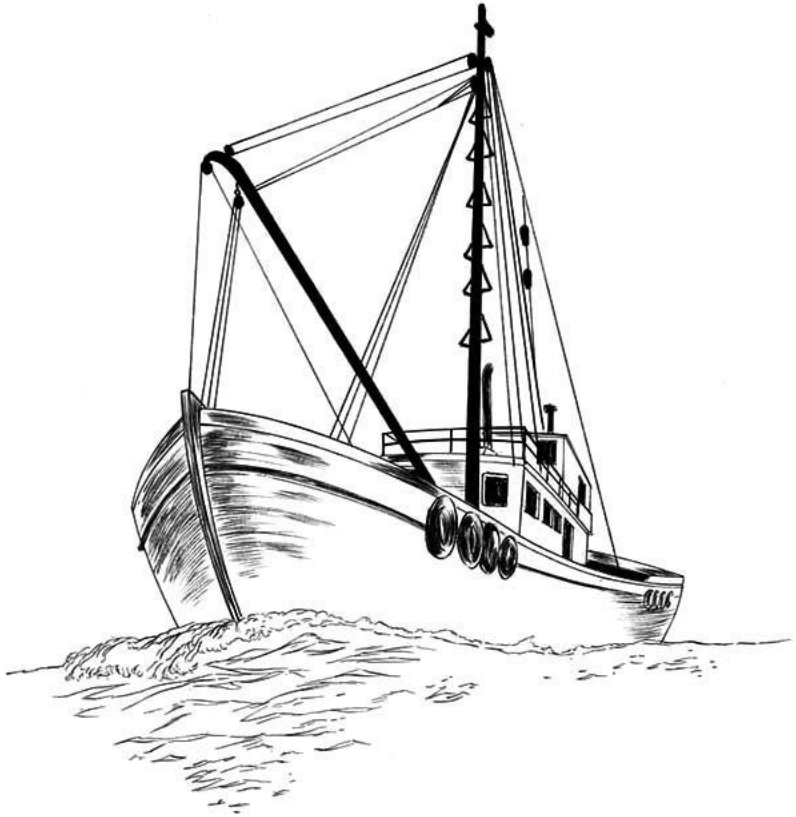

docker
for
DevOps
Engineers
2019
by
Siarhei Beliakou
part 1
LINUX

# Before We Begin. What's the problem? Why Docker?





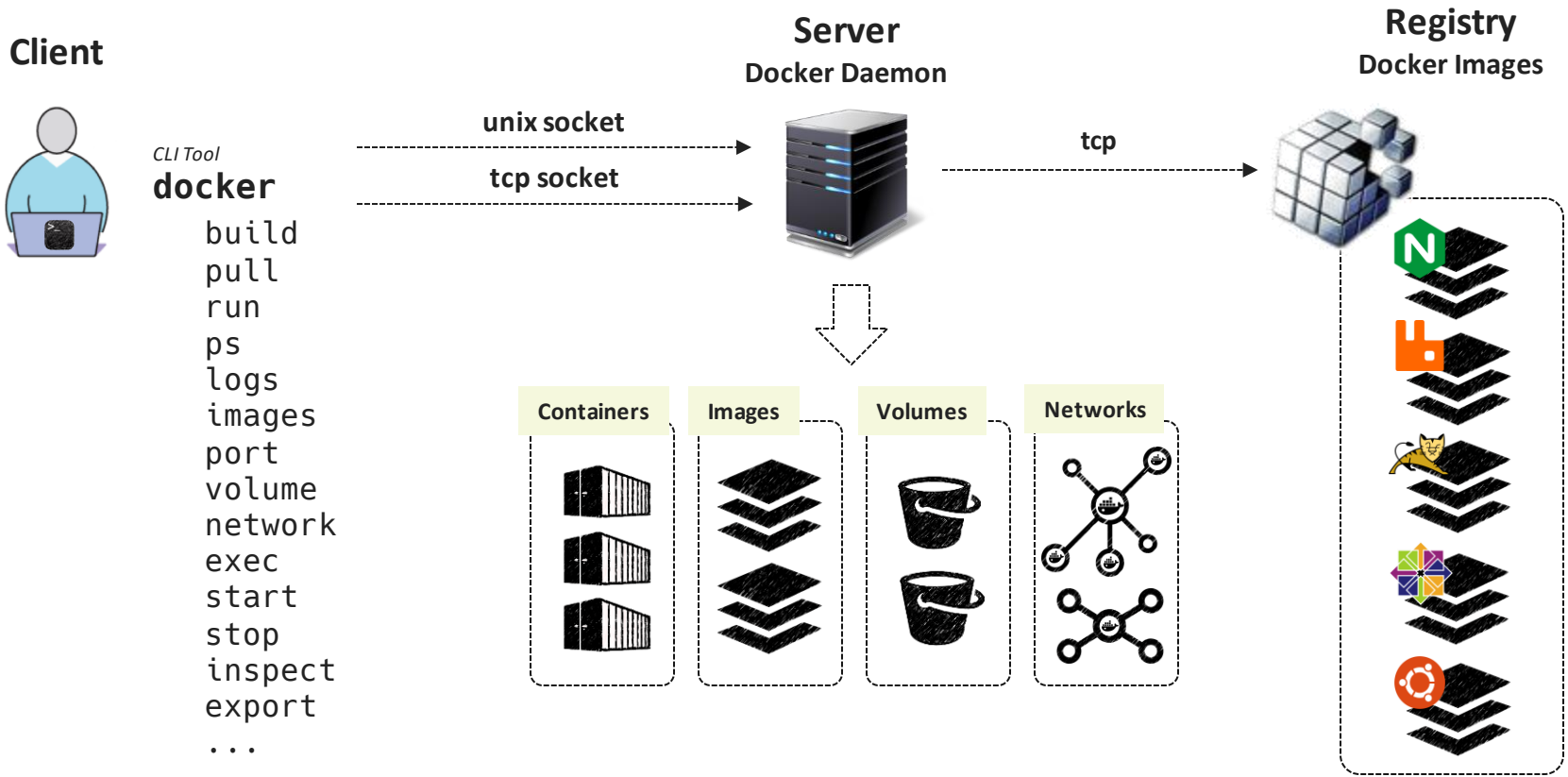
# AGENDA

- Docker Architecture Overview
- Docker Installation and Configuration
- Creating Docker Images
- Running Containers
- Mounting Data from the Host

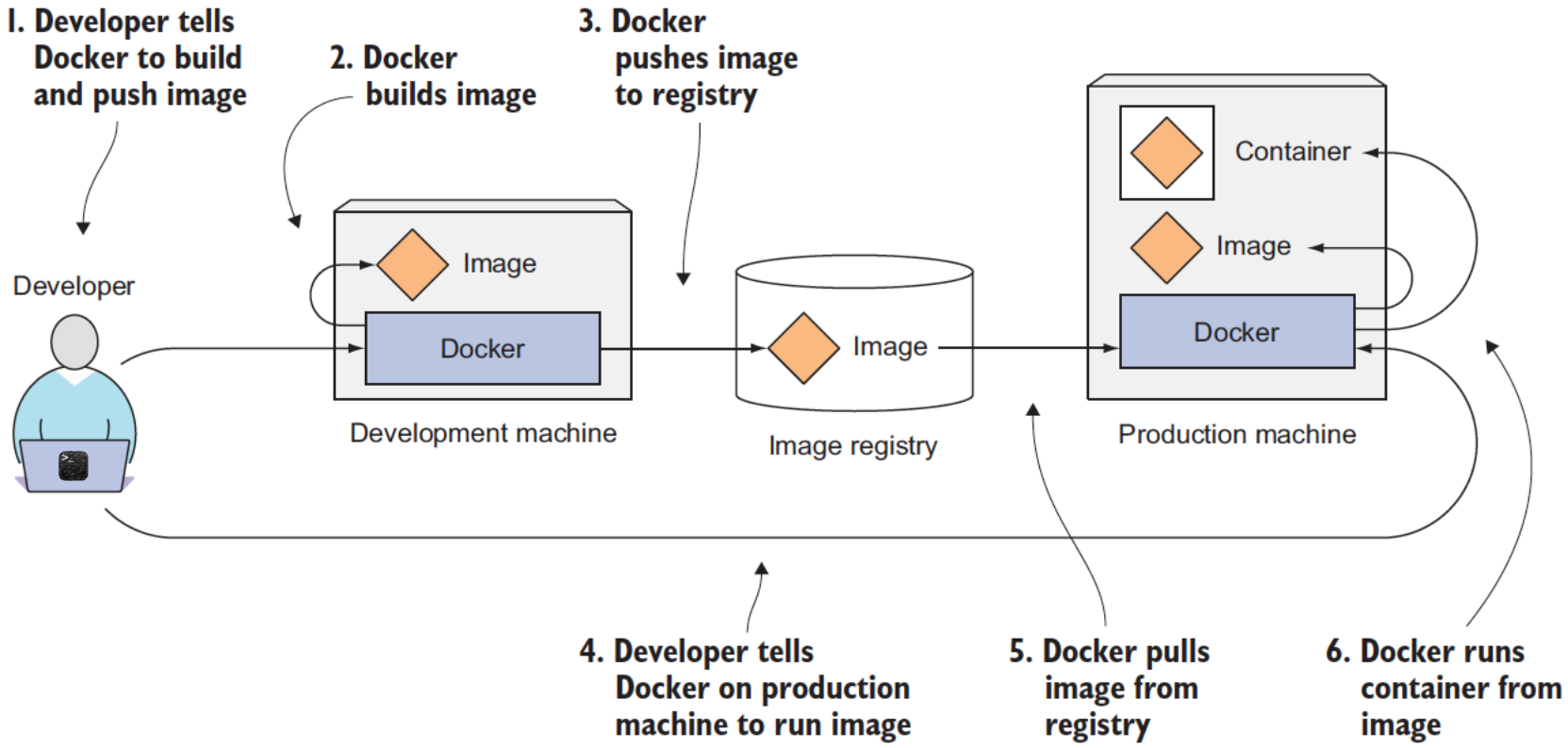
# ARCHITECTURE OVERVIEW

- Docker Architecture
- Docker Components
- Isolation Technologies
- What is a Container?
- What is an Image?

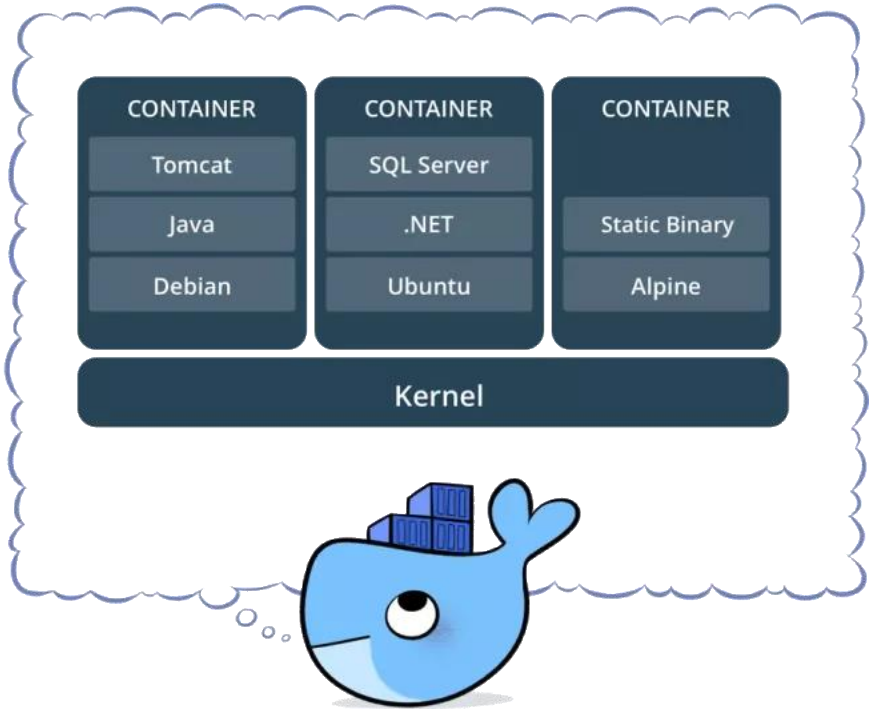
# Docker Architecture



# Docker Workflow



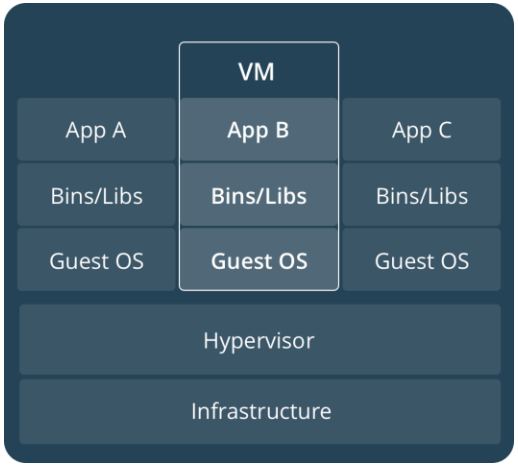
# What is a Container?




A **container** is something quite similar to a virtual machine, which can be used to **contain and execute** all the software required to run a particular program or set of programs.

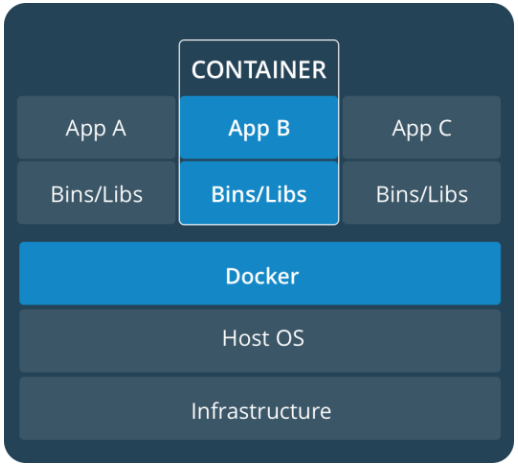
The container includes an operating system (typically some flavor of Linux) as base, plus any software installed on top of the OS that might be needed. This container can therefore be run as a self-contained virtual environment, which makes it a lot easier to reproduce the same analysis on any infrastructure that supports running the container, from your laptop to a cloud platform, without having to go through the pain of identifying and installing all the software dependencies involved. You can even have multiple containers running on the same machine, so you can easily switch between different environments if you need to run programs that have incompatible system requirements.

# Virtual Machines vs Containers




**VIRTUAL MACHINES**

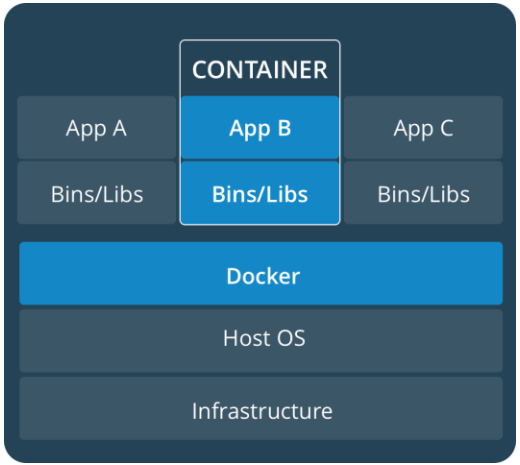
Virtual machines (VMs) are an abstraction (emulation) of physical hardware turning one server into many servers. The hypervisor allows multiple VMs to run on a single machine. Each VM includes a full copy of an operating system, one or more apps, necessary binaries and libraries - taking up tens of GBs. VMs can also be slow to boot.




**CONTAINERS**

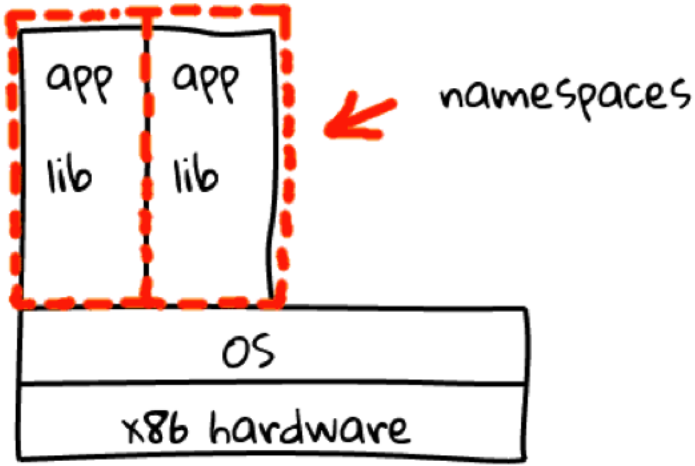
Containers are an abstraction at the app layer that packages code and dependencies together (worker process). Multiple containers can run on the same machine and share the OS kernel with other containers, each running as isolated processes in user space. Containers take up less space than VMs (container images are typically tens of MBs in size), and start almost instantly.

# Isolation Technologies: Functional Requirements




| Why | How |
|---|---|
| A process in the container shouldn't see other processes running in the system and should feel itself as the only process in the system | **Namespaces** |
| We should have facility to manage systems resources – (for example) to provide necessary amount of RAM, CPU shares and so on to the Container | **Control Groups** |
| A process in the container should see/use the only that part of the filesystems which is required by it | **Chroot** |
| A process in the container should have enough permissions to manage/use kernel | **Process Capabilities** |
| A process in the container should have access to ethernet device | **Virtual eth** |
| A few containers can expose the same ports, but is shouldn't cause any problem | **Port Binding** |
| A service should have facility to keep data, even it goes down accidently | **Volumes** |
| Service should be able to communicate with other services by IP/names | **Docker Network** |

# Isolation Technology



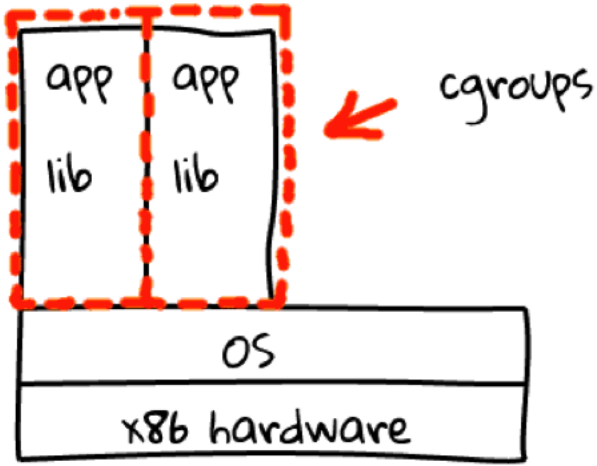

## Namespaces

When you run a container, Docker creates a set of *namespaces* for that container. This provides a layer of isolation: each aspect of a container runs in its own namespace and does not have access outside of it.

There are 6 Linux namespaces that Docker Engine uses:

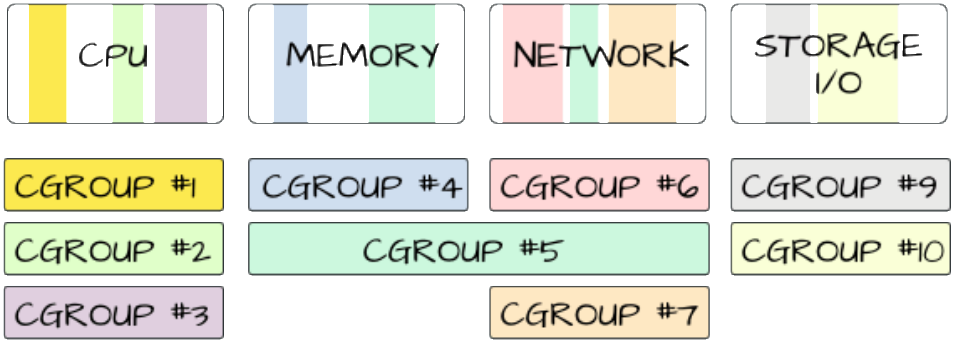
- ✓ The **mnt** namespace: Managing mount-points. *Linux 2.4.19*
- ✓ The **uts** namespace: Isolating kernel and version identifiers. (UTS: Unix Timesharing System). *Linux 2.6.19*
- ✓ The **ipc** namespace: Managing access to IPC resources (IPC: Inter Process Communication). *Linux 2.6.19*
- ✓ The **pid** namespace: Process isolation. *Linux 2.6.24*
- ✓ The **net** namespace: Managing network interfaces, *Linux 2.6.24*
- ✓ The **user** namespace. Linux 3.8

# Isolation Technology

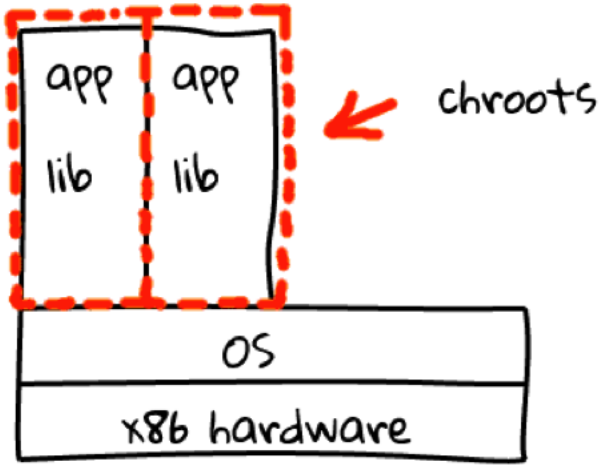



## Control Groups

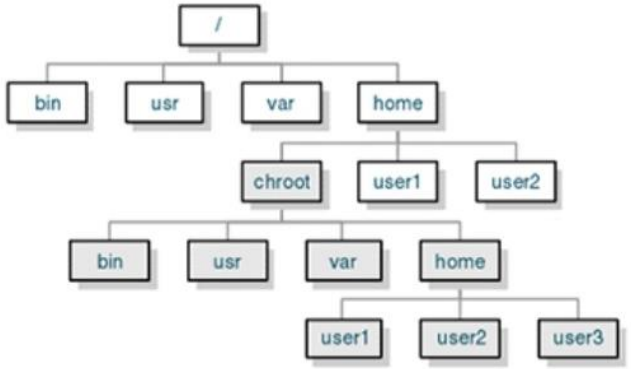
A key to running applications in isolation is to have them only use the resources you want. This ensures containers are good multi-tenant citizens on a host. Control groups allow Docker Engine to share available hardware resources to containers and, if required, set up limits and constraints. For example, limiting the memory available to a specific container.



# Isolation Technology



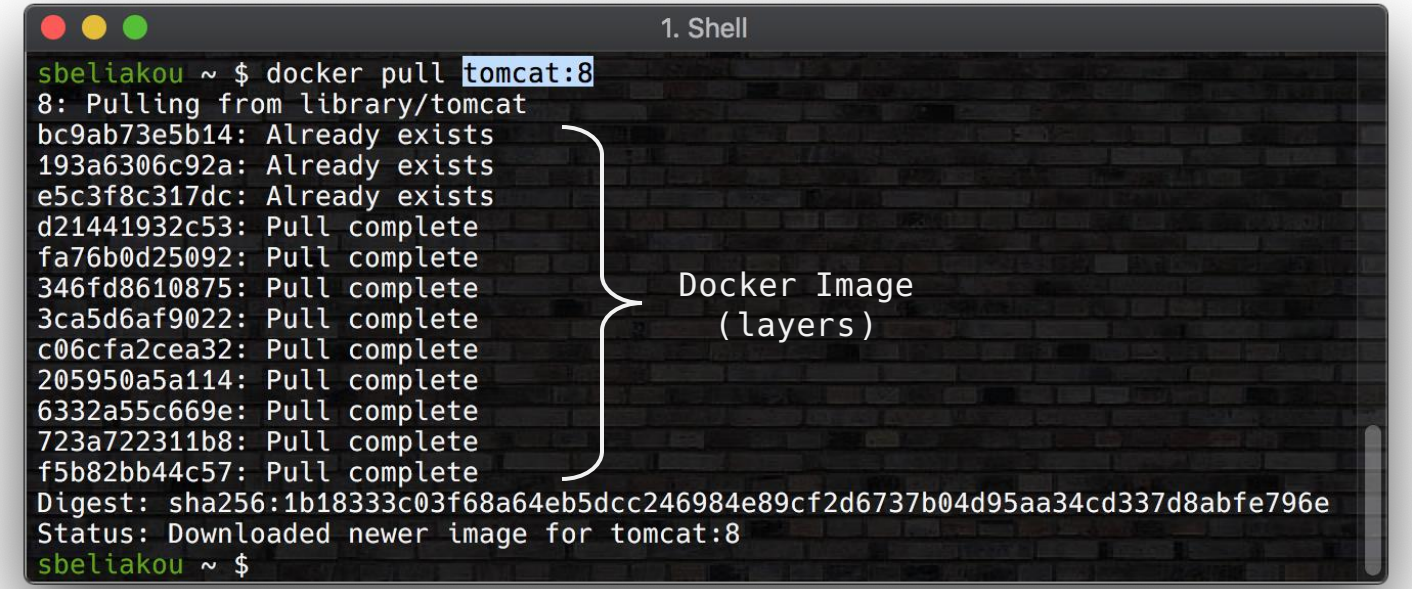

**chroot**
is simply isolation on the filesystem




**Union FS**
operate by creating layers, making them very lightweight and fast. Docker Engine uses union file systems to provide the building blocks for containers. Docker Engine can make use of several union file system variants including: AUFS, btrfs, vfs, and DeviceMapper

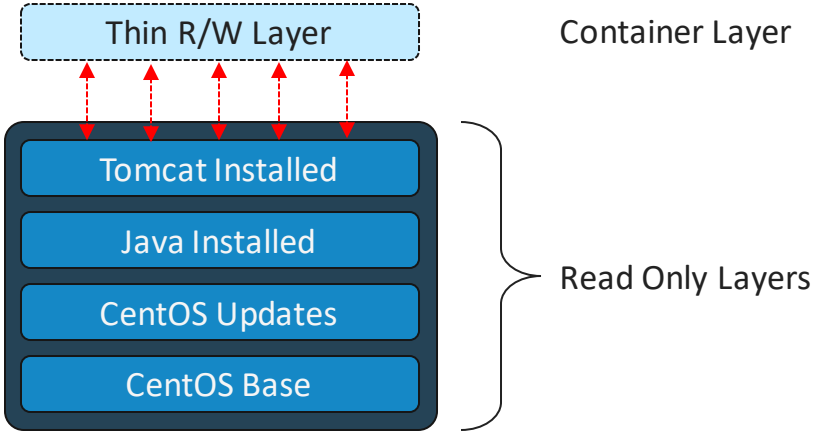
# Docker Image. What is this?

An image is an inert, immutable, file that's essentially a snapshot of a container. Images are created with the build command, and they'll produce a container when started with run. Images are stored in a Docker registry such as https://hub.docker.com. Because they can become quite large, images are designed to be composed of layers of other images, allowing a minimal amount of data to be sent when transferring images over the network



# Docker Storage (Graph) Drivers

**UnionFS** allows files and directories of separate file systems, known as branches, to be transparently overlaid, forming a single coherent file system. Contents of directories which have the same path within the merged branches will be seen together in a single merged directory, within the new, virtual filesystem.




### The copy-on-write (CoW) strategy

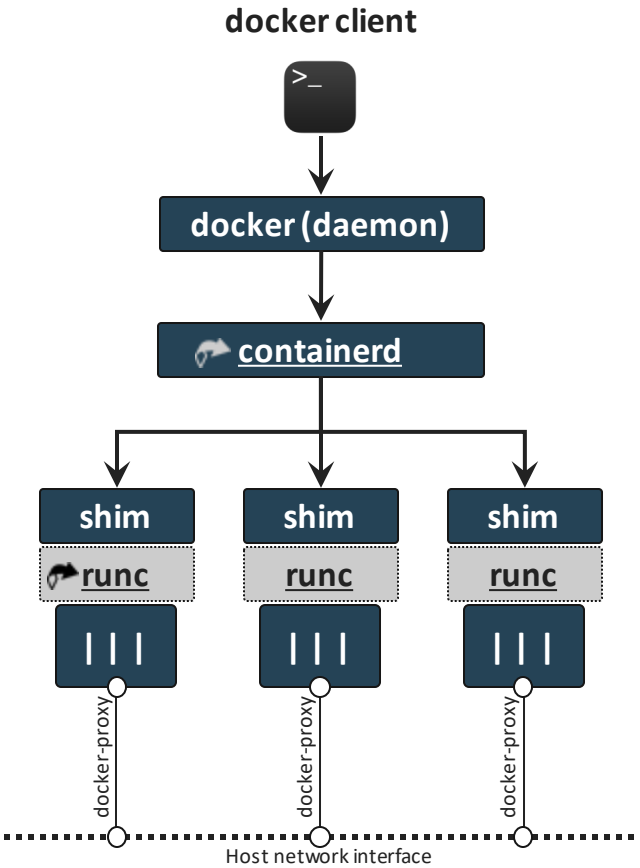
is a strategy of sharing and copying files for maximum efficiency. If a file or directory exists in a lower layer within the image, and another layer (including the writable layer) needs read access to it, it just uses the existing file. The first time another layer needs to modify the file (when building the image or running the container), the file is copied into that layer and modified. This minimizes I/O and the size of each of the subsequent layers.

# Docker Graphdrivers

Given the “image graph” of layer content represents the relationships between various layers, the driver to handle these layers is called a “graphdriver.”

- VFS - does not use a Union FS or CoW, is very valuable for simple validation and testing of other parts of the Docker engine
- AUFS - enables shared memory pages for different containers loading the same shared libraries from the same layer (because they are the same inode on disk). Currently it's available on Debian and Ubuntu and not considered as mainstream.
- Overlay2 - resolves the inode exhaustion problem as well as a few other bugs that were inherent to the old design of the original driver. It also enables shared memory between disparate containers using the same on-disk shared libraries.
- Device Mapper - It is quite unlike the union filesystems in that devicemapper works on block devices. There is no way to get default “out of the box” performance with devicemapper. Some of the features rely on specific versions of libdevmapper and it requires above-average skill to validate all these settings on a system.
- Btrfs - a disk formatted as a btrfs filesystem is required as the graphdriver root (by default, /var/lib/docker). It's not supported by RedHat.

# Docker Infrastructure




**dockerd:**

The Docker daemon itself. The highest level component in your list and also the only 'Docker' product listed. Provides all the nice UX features of Docker.

**docker-containerd:**

It's a daemon, listening on a Unix socket, exposes gRPC endpoints. Handles all the low-level container management tasks, storage, image distribution, network attachment, etc…

**docker-containerd-ctr:**

A lightweight CLI to directly communicate with containerd. Think of it as how 'docker' is to 'dockerd'.

**docker-containerd-shim:**

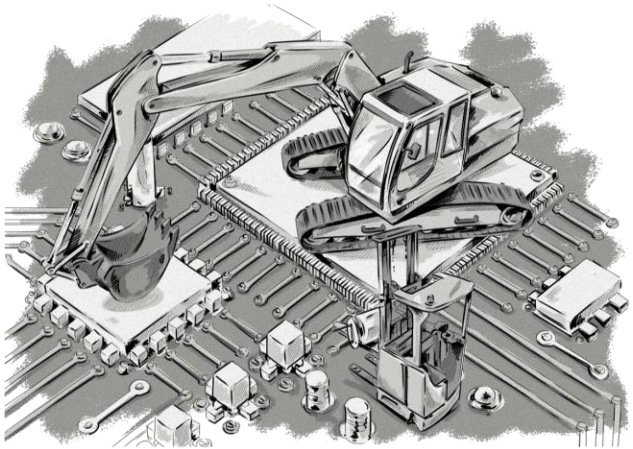
After runC actually runs the container, it exits (allowing us to not have any long-running processes responsible for our containers). The shim is the component which sits between containerd and runc to facilitate this.

**docker-runc:**

A lightweight binary for actually running containers. Deals with the low-level interfacing with Linux capabilities like cgroups, namespaces, etc…

**docker-proxy:**

A tool responsible for proxying container's ports to Host's interface



# Docker installation and Configuration

- Installing Docker on CentOS
- How Docker Works
- Configuring User Privileges to Talk to Dockerd
- Configuring Docker Daemon

# Learning Docker

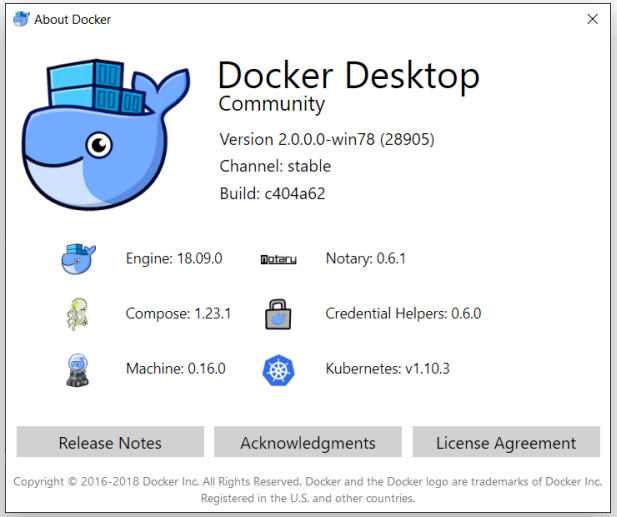
- Install Docker CE on CentOS
- Install Docker CE on Ubuntu

- Play with Docker Lab
- Play with Docker Classroom

- Docker Scenarios on Katacoda

# Docker on MacOS X and Windows


Install Docker for Mac

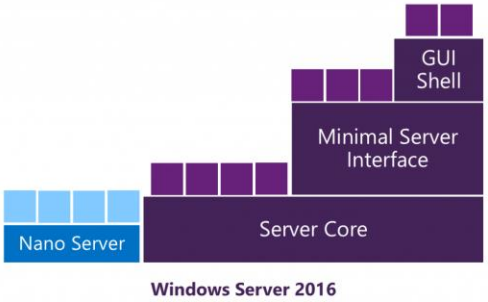



Install Docker for Windows

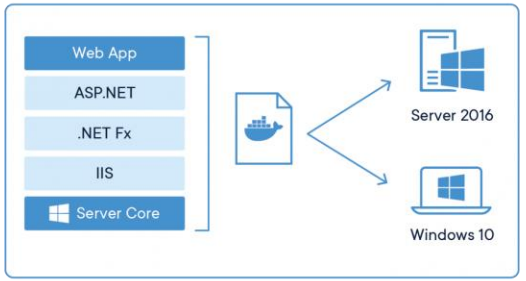
docs.microsoft.com/...

# Docker on Windows

**Microsoft includes two different types of containers:**




"**Hyper-V Container**",
based on the *Windows Nano Server* image




"**Windows Server Container**",
based on *Windows Server Core* image

**Check out more details here:**

- windows-containers
- quick-start-windows-10

**And here:**

- working-windows-containers-docker-basics
- working-windows-containers-docker-running
- working-windows-containers-docker-stride

# Local Engineering Environment

Get started with VAGRANT




Vagrantfile

```
1  Vagrant.configure("2")  do |config|
2    config.vm.box  = "sbeliakou/centos"
3    config.vm.network  :private_network,  ip: "192.168.56.15"
4
5    config.vm.provision  "shell", inline: <<-SHELL
6      yum install -y yum-utils jq net-tools
7      yum-config-manager  --add-repo \
8        https://download.docker.com/linux/centos/docker-ce.repo
9      yum-config-manager  --enable docker-ce-edge
10
11     yum install -y docker-ce
12     systemctl enable docker
13     systemctl start docker
14     usermod -aG docker vagrant
15   SHELL
16 end
17
```

Line 19, Column 1 Spaces: 2 Ruby

https://github.com/sbeliakou/docker-training

1. Shell

```
sbeliakou ~/project_folder $ vagrant up          # create a VM from Vagrantfile
sbeliakou ~/project_folder $ vagrant ssh         # ssh in to VM
sbeliakou ~/project_folder $ vagrant halt        # stop VM
sbeliakou ~/project_folder $ vagrant destroy     # destroy VM
```

# Installing Docker Service

The docker daemon binds to a Unix socket /var/run/docker.sock which is owned by root:docker.

```
[vagrant@localhost ~]$ docker info
Got permission denied while trying to connect to the Docker daemon socket at
unix:///var/run/docker.sock: Get http://%2Fvar%2Frun%2Fdocker.sock/v1.38/containers/json:
dial unix /var/run/docker.sock: connect: permission denied
```

Non-root user just needs to be added to the docker group.

```
[vagrant@localhost ~]$ sudo usermod -aG docker vagrant
# log out / log in

[vagrant@localhost ~]$ id
uid=1000(vagrant) gid=1000(vagrant) groups=1000(vagrant),993(docker)
```

Let's check docker info from vagrant user

```
[vagrant@localhost ~]$ docker info
Containers: 0
 Running: 0
 Paused: 0
 Stopped: 0
...
```

# How It Works

Either by using the docker binary or via the API, the Docker client tells the Docker daemon to run a container.

```
[vagrant@localhost ~]$ docker run -i -t ubuntu bash
```

The Docker Engine client is launched using the ***docker*** tool with the ***run*** option running a new container.

The bare minimum the *docker client* needs to tell the Docker daemon to run the container is:

- ✓ What Docker image to build/create the container from, for example, ubuntu
- ✓ The command you want to run inside the container when it is launched, for example,/bin/bash

So what happens under the hood when we run this command?
In order, Docker Engine does the following:

- ✓ Pulls the ***ubuntu*** image.
- ✓ Creates a new container.
- ✓ Allocates a filesystem and mounts a read-write *layer*.
- ✓ Allocates a network / bridge interface.
- ✓ Sets up an IP address.
- ✓ Executes a process that you specify.
- ✓ Captures and provides application output.

You now have a running container! Now you can manage your container, interact with your application and then, when finished, stop and remove your container.

# Try Something Simple

```
[vagrant@localhost vagrant]$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
9db2ca6ccae0: Pull complete
Digest: sha256:4b8ff392a12ed9ea17784bd3c9a8b1fa3299cac44aca35a85c90c5e3c7afacdc
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/engine/userguide/
```

# Docker Daemon, Hub/Registry Commands

**Display Docker Version and Info**

- `docker --version`
- `docker version`
- `docker version --format '{{.Server.Version}}'`
- `docker version --format '{{json .}}'`
- `docker info`
- `docker info --format '{{json .}}'`

**Working with Docker Hub/Registry:**

- `docker login` - to login to a registry.
- `docker logout` - to logout from a registry.
- `docker search` - searches registry for image.
- `docker pull` - pulls an image from registry to local machine.
- `docker push` - pushes an image to the registry from local machine.

# Inspecting Docker Configuration

```
[vagrant@localhost ~]$ docker info
Containers: 0
 Running: 0
 Paused: 0
 Stopped: 0
Images: 0
Server Version: 18.06.0-ce
Storage Driver: overlay2
 Backing Filesystem: xfs
Logging Driver: json-file
Runtimes: runc
Default Runtime: runc
Kernel Version: 3.10.0-862.9.1.el7.x86_64
Operating System: CentOS Linux 7 (Core)
Docker Root Dir: /var/lib/docker
Registry: https://index.docker.io/v1/
...
```

```
[vagrant@localhost]$ ps -eo %a | grep [d]ocker
/usr/bin/dockerd
docker-containerd --config /var/run/docker/containerd/containerd.toml
```

# Docker Root Dir

```
# tree -ACFL 1 /var/lib/docker
/var/lib/docker
├── builder/
├── buildkit/
├── containerd/
├── containers/
├── image/
├── network/
├── overlay2/
├── plugins/
├── runtimes/
├── swarm/
├── tmp/
├── trust/
└── volumes/
```

# Configuring Docker Daemon

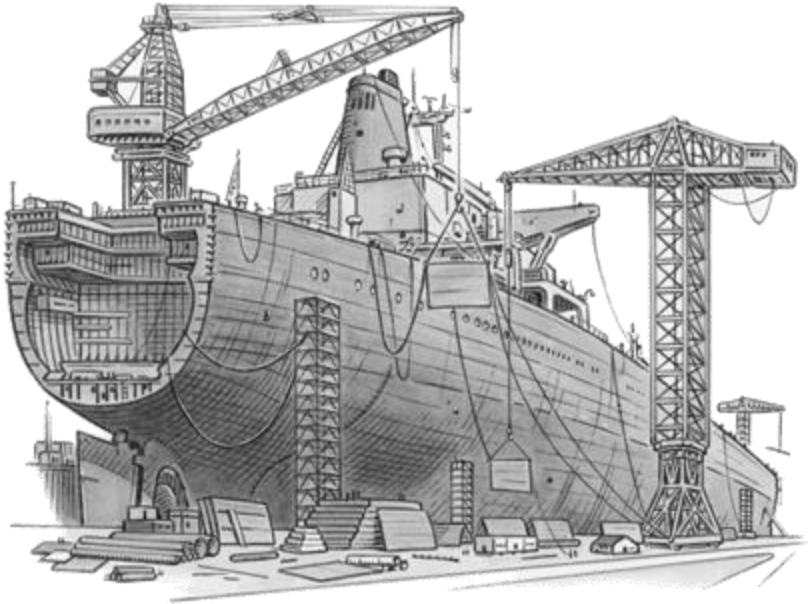
```
# docker info | egrep "(Cgroup|Storage) Driver"
Storage Driver: overlay2
Cgroup Driver: cgroupfs
```

/etc/docker/daemon.json

```
{
  "exec-opts": [
    "native.cgroupdriver=systemd"
  ],
  "storage-driver": "devicemapper"
}
```

```
# systemctl daemon-reload
# systemctl restart docker.service
# docker info | egrep "(Cgroup|Storage) Driver"
Storage Driver: devicemapper
Cgroup Driver: systemd
```


More Configuration Options



# Creating Docker Images

- Build own Image with Dockerfile
- Tagging Images
- Entrypoint and CMD
- Build Arguments
- Multistage Build
- ONEBUILD Instructions
- Health Checks

# Working with Images

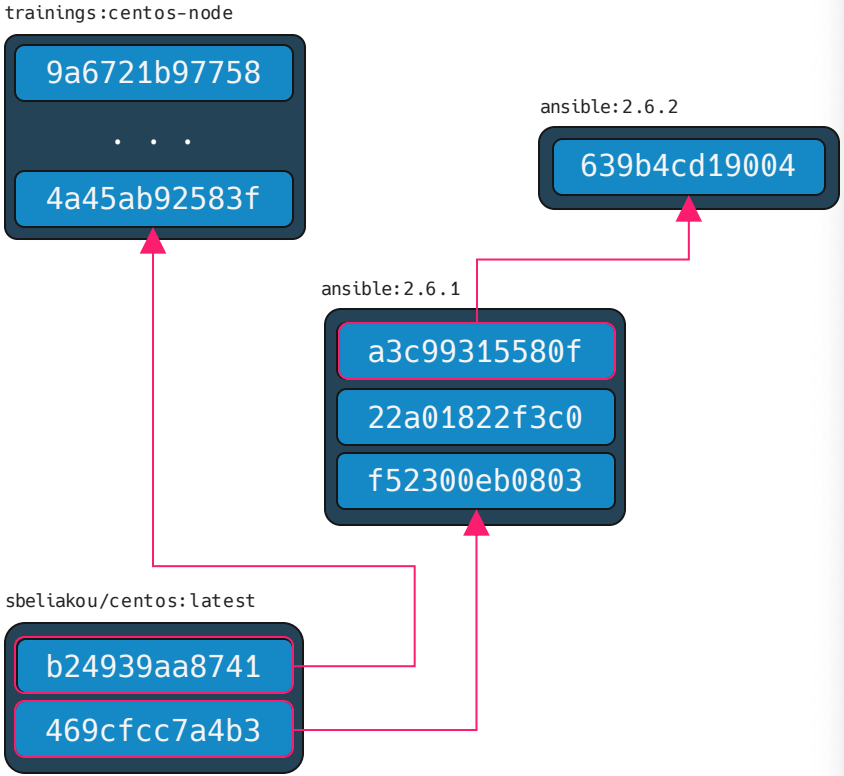
**Lifecycle:**

- `docker images` - shows all images.
- `docker import` - creates an image from a tarball.
- `docker build` - creates image from Dockerfile.
- `docker commit` - creates image from a container, pausing it temporarily if it is running.
- `docker rmi` - removes an image.
- `docker load` - loads an image from a tar archive as STDIN, including images and tags.
- `docker save` - saves an image to a tar archive stream to STDOUT with all parent layers, tags & versions.

**Info:**

- `docker history` - shows history of image.
- `docker tag` - tags an image to a name (local or registry).

# What is an Image?




```
@bb6140f107a7:~
sbeliakou ~ $ docker pull sbeliakou/centos:latest
latest: Pulling from sbeliakou/centos
469cfcc7a4b3: Pull complete
b24939aa8741: Pull complete
Digest: sha256:46bfd372002e25fb3505b0119853bdfbfee26395cdb97af45ead2aa83c685394
Status: Downloaded newer image for sbeliakou/centos:latest
sbeliakou ~ $ docker pull sbeliakou/ansible:2.6.1
2.6.1: Pulling from sbeliakou/ansible
469cfcc7a4b3: Already exists
f52300eb0803: Pull complete
22a01822f3c0: Pull complete
a3c99315580f: Pull complete
Digest: sha256:ce24829798b626c8f748dc225cbbaf805189cd3945f9a37f89a53f3da393baf2
Status: Downloaded newer image for sbeliakou/ansible:2.6.1
sbeliakou ~ $ docker pull sbeliakou/ansible:2.6.2
2.6.2: Pulling from sbeliakou/ansible
469cfcc7a4b3: Already exists
f52300eb0803: Already exists
22a01822f3c0: Already exists
a3c99315580f: Already exists
639b4cd19004: Pull complete
Digest: sha256:46904da2ce545c753f14b13e73d8d2097fe431c9ca013ac0e08e4c0e6fd8d247
Status: Downloaded newer image for sbeliakou/ansible:2.6.2
sbeliakou ~ $ docker pull sbeliakou/trainings:centos-node
centos-node: Pulling from sbeliakou/trainings
469cfcc7a4b3: Already exists
b24939aa8741: Already exists
4a45ab92583f: Pull complete
b607a6c71319: Pull complete
406aff0ed91f: Pull complete
7e5f845fcf24: Pull complete
40e8e164788e: Pull complete
e96b1ac125c0: Pull complete
b9debdb79786: Pull complete
20439a1df421: Pull complete
5565437f488a: Pull complete
1fb62f64a7ab: Pull complete
a194ef741153: Pull complete
7d5d09b24dcb: Pull complete
cd9bc492f001: Pull complete
cafa18c0e0f3: Pull complete
9a6721b97758: Pull complete
Digest: sha256:4c7c2b3ce1c22ddd99b377a1ac12413389fcf4b703ce9f24694ea3d5f5a4c205
Status: Downloaded newer image for sbeliakou/trainings:centos-node
sbeliakou ~ $
```

# Writing Own Dockerfile

http://flask.pocoo.org/
http://flask.pocoo.org/docs/1.0/quickstart/#quickstart

dockerfile_best-practices
sbeliakou/flask-hello

**hello.py**

```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello():
    return "Hello World!\n"
```

**Dockerfile**

```
FROM python:2.7

RUN pip install Flask
COPY . /

ENV FLASK_APP=hello.py
EXPOSE 5000
CMD flask run --host=0.0.0.0
```

```
$ docker build -t myflask .
    Successfully built 23bbddc9f137
    Successfully tagged myflask:latest
$ docker run -d -P myflask
    6644fd128502...
$ docker port 6644fd128502
    5000/tcp -> 0.0.0.0:32779
$ curl localhost:32779
```

Each instruction creates one layer:

- **FROM** creates a layer from the python:2.7 Docker image.
- **RUN** performs necessary commands
- **COPY** adds files from your Docker client's current directory
- **ENV** sets environment variable
- **EXPOSE** defines which ports will be exposed by container
- **CMD** specifies what command to run within the container

# Dockerfile Instructions

- `.dockerignore`
- `FROM` - Sets the Base Image for subsequent instructions.
- `RUN` - execute any commands in a new layer on top of the current image and commit the results.
- `CMD` - provide defaults for an executing container.
- `EXPOSE` - informs Docker that the container listens on the specified network ports at runtime. NOTE: does not actually make ports accessible.
- `ENV` - sets environment variable.
- `ADD` - copies new files, directories or remote file to container. Invalidates caches.
- `COPY` - copies new files or directories to container.
- `ENTRYPOINT` - configures a container that will run as an executable.
- `VOLUME` - creates a mount point for externally mounted volumes or other containers.
- `USER` - sets the user name for following RUN / CMD / ENTRYPOINT commands.
- `WORKDIR` - sets the working directory.
- `ARG` - defines a build-time variable.
- `ONBUILD` - adds a trigger instruction when the image is used as the base for another build.
- `STOPSIGNAL` - sets the system call signal that will be sent to the container to exit.
- `LABEL` - apply key/value metadata to your images, containers, or daemons.

# Useful Links and Examples

**Docker CLI Reference:** commandline/cli/
**Dockerfile Reference:** #/dockerfile-reference

**Many Samples**: docs.docker.com/samples/
**Examples**: #dockerfile-examples
**Jenkins**: Dockerfile-alpine
**Tomcat**: jre8-alpine/Dockerfile
**OpenJDK**: jdk/alpine/Dockerfile
**AmazonLinux**: 2018.03/Dockerfile

**Another example is here:** #define-a-container-with-dockerfile
**Best practices for writing Dockerfiles:** docs.docker.com/dockerfile_best-practices/
**Docker Lab Samples:** https://github.com/docker/labs

**Dockerfile Practice Questions:** docker-certification-practice-questions-dockerfile

**Many Docker based solutions:** veggiemonk/awesome-docker

# Dockerfile

Dockerfile

```
FROM centos
LABEL maintainer="Siarhei Beliakou"
RUN yum install -y httpd web-assets-httpd && \
    yum clean all
RUN echo "my httpd container" > /var/www/html/index.html
# ADD/COPY index.html /var/www/html/
EXPOSE 80
ENTRYPOINT ["httpd"]
CMD ["-DFOREGROUND"]
```

```
$ docker build [-t image_tag] [-f ./path/to/dockerfile] .
...
Status: Downloaded newer image for centos:latest
 ---> 49f7960eb7e4
...
Step 4/6 : RUN echo "my httpd container" > /var/www/html/index.html
 ---> Running in 37cc17740d0b
...
Successfully built 50a986f614d5
```

# Tagging Images

```
$ docker build -t myhttpd .
...
Successfully built 50a986f614d5
Successfully tagged myhttpd:latest
```

**$ docker tag << id | tag >> << new_tag >>**

```
$ docker tag 50a986f614d5 sbeliakou/myhttpd:1.0
```

```
$ docker tag myhttpd sbeliakou/myhttpd:1.0
```

```
$ docker tag myhttpd:latest sbeliakou/myhttpd:latest
```

```
$ docker push sbeliakou/myhttpd:latest
```

# ENTRYPOINT and CMD

```
...
ENTRYPOINT ["httpd"]
CMD ["-DFOREGROUND"]
```

- An **ENTRYPOINT** allows you to configure a container that will run as an executable.
- The main purpose of a **CMD** is to provide defaults for an executing container
- If you would like your container to run the same executable every time, then you should consider using **ENTRYPOINT** in combination with **CMD**
- **ENTRYPOINT/CMD** has 2 forms: exec and shell
  **exec form:** ["echo", "hello", "world"] ← preferred form
  **shell form:** echo hello world ← supports ENV Vars resolving

https://docs.docker.com/engine/reference/builder/#entrypoint
https://docs.docker.com/engine/reference/builder/#cmd
http://www.johnzaccone.io/entrypoint-vs-cmd-back-to-basics/

# ENTRYPOINT and CMD

```
FROM busybox
ENTRYPOINT ["ping"]
CMD ["-c1", "epam.com"]
```

```
$ docker build -t ping:1.0 .
```

```
$ docker run ping:1.0
```

will send 1 ICMP package to epam.com (default)

```
$ docker run ping:1.0 --help
```

will print help output

```
$ docker run ping:1.0 google.com
```

will ping "google.com" unless interrupted with ^C

```
$ docker run ping:1.0 -c1 google.com
```

will ping "google.com" for 1 time

# Dockerfile Example: Custom Packer Image

```
FROM centos:7
ARG PACKER_VERSION=1.2.3
RUN yum install -y \
      epel-release \
      yum-plugin-ovl \
      wget unzip \
      rsync \
      openssh openssh-clients && \
    yum install -y python-pip && \
    yum clean all

RUN wget -q https://releases.hashicorp.com/...${PACKER_VERSION}_linux_amd64.zip && \
    unzip -q packer_${PACKER_VERSION}_linux_amd64.zip -d /bin/ && \
    rm -f packer_${PACKER_VERSION}_linux_amd64.zip
RUN pip install -U ansible ansible-modules-hashivault

RUN useradd packer

USER packer
ENV USER packer

ENTRYPOINT ["/bin/packer"]
CMD ["--version"]
```

# A Few More Examples

```
FROM openjdk:8-jre

ADD customer-contact-service.jar /
EXPOSE 4040

ENTRYPOINT ["java", "-jar", "/customer-contact-service.jar"]
```

```
FROM openjdk:8

ENV PORT=8080
EXPOSE 8080
COPY wiremock /wiremock

ENTRYPOINT ["/wiremock/bin/startServer.sh"]
```

# Docker Base Images

- **scratch** – this is the ultimate base image and it has 0 files and 0 size.
- **busybox** – a minimal Unix weighing in at 2.5 MB and around 10000 files.
- **debian:jessie** – the latest Debian is 122 MB and around 18000 files.
- **alpine:latest** – Alpine Linux, only 8 MB in size and has access to a package repository

More details

```
FROM scratch

ADD   centos.tar.gz /

RUN   yum install -y epel-release && \
      yum update -y && \
      yum clean all

LABEL architecture="amd64" \
      OS="CentOS" \
      License=GPLv2 \
      maintainer="Siarhei Beliakou (sbeliakou@gmail.com)"

## Default command
CMD ["/bin/bash"]
```

# Building With Arguments

```
ARG BASE_IMAGE
FROM ${BASE_IMAGE}
...
```

```
# docker build --build-arg BASE_IMAGE=ubuntu:16.04 .
```

```
# docker build --build-arg BASE_IMAGE=ubuntu:18.04 .
```

```
ARG BUILD_NUMBER
ARG JOB_NAME
LABEL build_number="${BUILD_NUMBER}"
LABEL job_name="${JOB_NAME}"
```

```
# docker build \
  --build-arg BUILD_NUMBER=${BUILD_NUMBER} \
  --build-arg JOB_NAME=${JOB_NAME} \
  .
```

# Using ARG Directive

```
1   FROM openjdk:8-jdk
2
3   RUN apt-get update && apt-get install -y git curl && rm -rf /var/lib/apt/lists/*
4
5   ARG user=jenkins
6   ARG group=jenkins
7   ARG uid=1000
8   ARG gid=1000
9   ARG http_port=8080
10  ARG agent_port=50000
11  ARG JENKINS_HOME=/var/jenkins_home
    ...
19  RUN mkdir -p $JENKINS_HOME \
20    && chown ${uid}:${gid} $JENKINS_HOME \
21    && groupadd -g ${gid} ${group} \
22    && useradd -d "$JENKINS_HOME" -u ${uid} -g ${gid} -m -s /bin/bash ${user}
```

https://github.com/jenkinsci/docker/blob/master/Dockerfile

# Multistage Build

```
FROM maven:3.3-jdk-8 as builder
COPY . /build/
WORKDIR /build
RUN mvn clean install
```

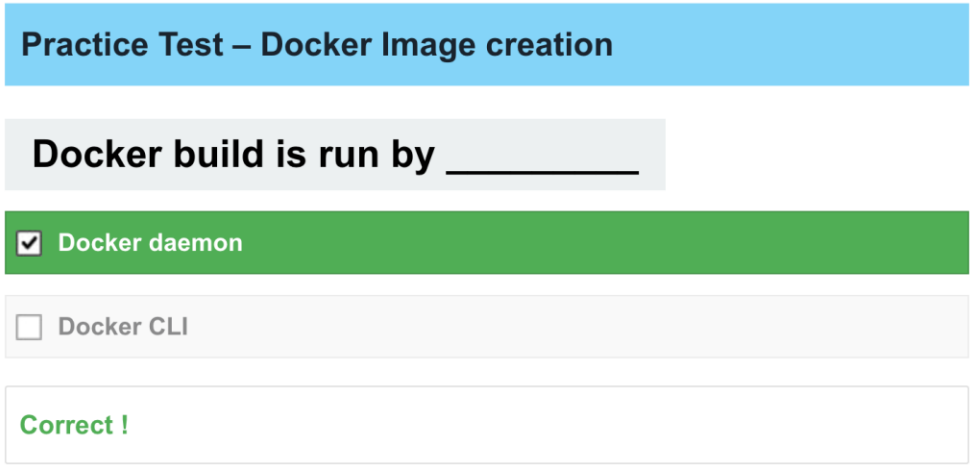
```
FROM openjdk:8-jre
COPY --from=builder /build/target/demoapp.jar /opt/
EXPOSE 8080
ENTRYPOINT ["java", "-jar", "/opt/demoapp.jar"]
```

```
# docker build -t myapp .
...
```

```
# docker build -t myapp --target builder .
...
```

sbeliakou/springboot_example

# Try Out Online Test




https://vitalflux.com/docker-certification-practice-questions-dockerfile/
https://djitz.com/certification/docker-certified-associate-test-review-questions-set-1-image-creation/

# Running Containers

- Running in detached mode
- Exposing Ports
- Managing Restart Policy
- Changing Workspace Directory
- Changing Runtime User
- Providing Environment Variables
- Providing Labels

# Working with Containers

**Lifecycle:**

- `docker create` - creates a container but does not start it.
- `docker rename` - allows the container to be renamed.
- `docker run` - creates and starts a container in one operation.
- `docker rm` - deletes a container.
- `docker update` - updates a container's resource limits.

**Starting and Stopping:**

- `docker start` - starts a container so it is running.
- `docker stop` - stops a running container.
- `docker restart` - stops and starts a container.
- `docker pause` - pauses a running container, "freezing" it in place.
- `docker unpause` - will unpause a running container.
- `docker wait` - blocks until running container stops.
- `docker kill` - sends a SIGKILL to a running container.
- `docker attach` - will connect to a running container.

# Working with Containers

**Info:**

- `docker ps` - shows running containers.
- `docker logs` - gets logs from container.
- `docker inspect` - looks at all the info on a container.
- `docker events` - gets events from container.
- `docker port` - shows public facing port of container.
- `docker top` - shows running processes in container.
- `docker stats` - shows containers' resource usage statistics.
- `docker diff` - shows changed files in the container's FS.

**Import / Export:**

- `docker cp` - copies files or folders between a container and the local filesystem.
- `docker export` - turns container filesystem into tarball archive stream to STDOUT.

**Executing Commands:**

- `docker exec` - to execute a command in container.

# docker run reference

```
# docker run options IMAGE Non_Default_CMD CMD_Args
```

```
options:
  -d              Run container in background and print container ID
  -P              Publish all exposed ports to random ports
  -p              Publish a container's port(s) to the host
  --restart       Restart policy to apply when a container exits (default "no")
  -i              Keep STDIN open even if not attached
  -t              Allocate a pseudo-TTY
  --rm            Automatically remove the container when it exits
  -v              Bind mount a volume
  -e              Set environment variables
  --label         Set meta data on a container
  --log-driver    Logging driver for the container

  -u              Username or UID (format: <name|uid>[:<group|gid>])
  -w              Working directory inside the container
  --entrypoint    Overwrite the default ENTRYPOINT of the image
```

https://docs.docker.com/engine/reference/run/

# Running the Container

```
# docker run 50a986f614d5 # myhttpd:1.0
^C
```

```
# docker run -d myhttpd:1.0
9f761335efe268e9a82c4828d8f4be67b5824eb3266e8ba311343a7da45c67ff
```

```
# docker ps
CONTAINER ID  IMAGE         COMMAND                  CREATED        STATUS         PORTS                  NAMES
9f761335efe2  50a986f614d5  "/bin/sh -c 'httpd -…"  5 seconds ago  Up 4 seconds   80/tcp                 trusting_kilby
```

```
# docker run -P -d myhttpd:1.0
74954ff14ec5e53ac9925bfd2873c654fe8978657764b4162ac494fc9afaab9f
```

```
# docker ps
CONTAINER ID  IMAGE         COMMAND                  CREATED        STATUS         PORTS                     NAMES
74954ff14ec5  myhttpd:1.0   "/bin/sh -c 'httpd -…"  9 seconds ago  Up 18 seconds  0.0.0.0:32768->80/tcp     fervent_noyce
9f761335efe2  myhttpd:1.0   "/bin/sh -c 'httpd -…"  3 minutes ago  Up 3 minutes   80/tcp                    trusting_kilby
```

```
# curl localhost:32768 # or curl <<VM_external_ip>>:32768
my httpd container
```

```
# docker run -d -p 8081:80 --name h8081  myhttpd:1.0
fca7f4525bc618e7c503b73bfa680c055300e8b5c767d48e33669831e0bc5bec
# docker run -d -p 127.0.0.1:8082:80 --name h8082 myhttpd:1.0
014e5efa5ca90d9b7e50eebef3c7f020f08f0b5238f98420681ee348a4097829
```

```
# docker ps --format "table {{.Names}}\t{{.Image}}\t{{.ID}}\t{{.Ports}}" -n2
NAMES               IMAGE               CONTAINER ID        PORTS
h8081               myhttpd:1.0         fca7f4525bc6        0.0.0.0:8081->80/tcp
h8082               myhttpd:1.0         014e5efa5ca9        127.0.0.1:8082->80/tcp
```

formatting

# Running the Container: Restart Policy

```
# docker run -d --restart=always --name sleeper centos sleep 5
6c3d24b3f89f13de92e710fcbb5b343b4cb81e7454ecc79445e94f0c5ba31a49
```

```
# docker ps --format "table {{.Names}}\t{{.Image}}\t{{.ID}}\t{{.RunningFor}}\t{{.Status}}" -n1
NAMES         IMAGE         CONTAINER ID        CREATED              STATUS
sleeper       centos        33675bff7f47        2 seconds ago        Up 1 second
```

```
# docker ps --format "table {{.Names}}\t{{.Image}}\t{{.ID}}\t{{.RunningFor}}\t{{.Status}}" -n1
NAMES         IMAGE         CONTAINER ID        CREATED              STATUS
sleeper       centos        33675bff7f47        11 seconds ago       Up 4 seconds
```

```
# docker ps --format "table {{.Names}}\t{{.Image}}\t{{.ID}}\t{{.RunningFor}}\t{{.Status}}" -n1
NAMES         IMAGE         CONTAINER ID        CREATED               STATUS
sleeper       centos        33675bff7f47        About a minute ago    Restarting (0) 6 seconds ago
```

```
# docker inspect -f "{{ .RestartCount }}" sleeper
19
```

| Policy | Result |
|---|---|
| **no** | Do not automatically restart the container when it exits. This is the default. |
| **on-failure[:max-retries]** | Restart only if the container exits with a non-zero exit status. Optionally, limit the number of restart retries the Docker daemon attempts. |
| **always** | Always restart the container regardless of the exit status. When you specify always, the Docker daemon will try to restart the container indefinitely. The container will also always start on daemon startup, regardless of the current state of the container. |
| **unless-stopped** | Always restart the container regardless of the exit status, including on daemon startup, except if the container was put into a stopped state before the Docker daemon was stopped. |

More details

# Running Containers in Interactive Mode

```
[root@localhost ~]# docker run centos cat /etc/redhat-release
CentOS Linux release 7.5.1804 (Core)
```

```
[root@localhost ~]# docker run ubuntu cat /etc/lsb-release
DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=18.04
DISTRIB_CODENAME=bionic
DISTRIB_DESCRIPTION="Ubuntu 18.04.1 LTS"
```

```
[root@localhost ~]# docker run -it centos bash
[root@dfc1b0d4f6a5 /]# cat /etc/redhat-release
CentOS Linux release 7.5.1804 (Core)
[root@dfc1b0d4f6a5 /]# yum install curl wget # and so on
...
[root@dfc1b0d4f6a5 /]# exit
exit
[root@localhost ~]#
```

```
[root@localhost ~]# docker ps --format "table {{.ID}}\t{{.RunningFor}}\t{{.Status}}" -n1 -a
CONTAINER ID        CREATED             STATUS
dfc1b0d4f6a5        7 minutes ago       Exited (1) 1 minutes ago
```

```
[root@localhost ~]# docker rm dfc1b0d4f6a5
dfc1b0d4f6a5
```

```
[root@localhost ~]# docker run --rm centos cat /etc/redhat-release
CentOS Linux release 7.5.1804 (Core)
```

# Executing Commands Inside Running Container

```
# docker run -d centos sleep infinity
9626a94669c935ea140bcb9ea83339bd325b28195fc088a690b620eb12902b33
```

```
# docker ps -l
CONTAINER ID   IMAGE   COMMAND           CREATED          STATUS         PORTS  NAMES
9626a94669c9   centos  "sleep infinity"  11 seconds ago   Up 10 seconds         priceless_einstein
```

```
# docker exec -it 9626a94669c9 bash
[root@9626a94669c9 /]# ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 13:43 ?        00:00:00 sleep infinity
root        34     0  2 13:48 pts/0    00:00:00 bash
root        47    34  0 13:48 pts/0    00:00:00 ps -ef
[root@9626a94669c9 /]# exit
exit
#
```

# Stopping/Deleting Containers

```
# docker ps --format "table {{.Image}}\t{{.Names}}\t{{.ID}}\t{{.RunningFor}}\t{{.Status}}"
IMAGE          NAMES    CONTAINER ID        CREATED             STATUS
myhttpd:1.0    h8081    fca7f4525bc6        About an hour ago   Up About an hour
myhttpd:1.0    h8082    014e5efa5ca9        About an hour ago   Up About an hour
```

```
# docker stop h8082
014e5efa5ca9
```

```
# docker rm 014e5efa5ca9
014e5efa5ca9
```

```
# docker rm $(docker stop 014e5efa5ca9)
014e5efa5ca9
```

```
# docker rm $(docker stop $(docker ps -a -q))
014e5efa5ca9
fca7f4525bc6
```

☆

```
# docker container prune
WARNING! This will remove all stopped containers.
Are you sure you want to continue? [y/N]
```

☆

```
# docker image prune
WARNING! This will remove all dangling images.
Are you sure you want to continue? [y/N]
you want to continue? [y/N]
```

# Changing Container's Build Defaults

```
...
RUN useradd jenkins -u 1000
USER jenkins
```

### 1. Default User

```
# docker run jenkins id
uid=1000(jenkins) gid=1000(jenkins) groups=1000(jenkins)
```

```
# docker run --user 0 jenkins id
uid=0(root) gid=0(root) groups=0(root)
```

--user == -u

```
# docker run --user 1000:0 jenkins id
uid=1000(jenkins) gid=0(root) groups=0(root)
```

```
# docker run --group-add 123 jenkins id
uid=1000(jenkins) gid=1000(jenkins) groups=1000(jenkins),123
```

```
...
WORKDIR /
```

### 2. Default Workdir

```
# docker run jenkins pwd
/
```

```
# docker run --workdir /var/jenkins_home jenkins pwd
/var/jenkins_home
```

--workdir == -w

```
# docker run -w $(pwd) -v $(pwd):$(pwd) maven clean package
```

# Changing Container's Build Defaults

### 3. Default Entrypoint

```
...
ENTRYPOINT ["httpd"]
...
```

```
# docker run -it myhttpd:1.0 bash
# echo $?
1
```

```
# docker run -it --entrypoint=/bin/bash myhttpd:1.0
[root@e4e7976cf838 /]# ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  1 19:27 pts/0    00:00:00 /bin/bash
root        14     1  0 19:27 pts/0    00:00:00 ps -ef
[root@e4e7976cf838 /]# exit
#
```

### 4. Environment Variables

```
# docker run -it -e MYVAR="My Variable" centos env | grep MYVAR
MYVAR=My Variable
```

```
# docker run -it --env-file <(env| grep ARM | cut -f1 -d=) centos env | grep ARM
ARM_SUBSCRIPTION_ID=64a3f30f-xxx-xxxx-xxxx-yyyyfe63fe9a
ARM_TENANT_ID=bd5c6713-xxx-yyyy-xxxx-78f2d078e543
ARM_CLIENT_SECRET=xxxxxxxxxxxxxxx
ARM_CLIENT_ID=808f38ed-xxxx-xxxx-yyyy-ebbce91bcfee
```

# Running Containers with Labels

```
# docker run -d --label app=web1 nginx
```

```
# docker run -d --label app=web2 nginx
```

```
# docker run -d --label app=web3 nginx
```

```
# docker ps --filter 'label=app=web1' --format "table {{.ID}}\t{{.Status}}"
CONTAINER ID        STATUS
978dc141e8e0        Up 2 minutes
```

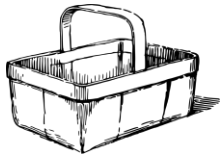
```
# docker stop $(docker ps --filter 'label=app=web1' -q)
978dc141e8e0
```

```
# docker ps --filter 'label=app=web1' --format "table {{.ID}}\t{{.Status}}"
CONTAINER ID        STATUS
<< no containers, all stopped by command above >>
```

# Mounting Data from the Host

- Mounting Folders
- Mounting Files
- R/O Mounts
- Mounting TMPFS

# Use Cases



1. Running Containers with Local Configs / Data
2. Running Containers for Processing Data (Tools)
3. Saving Changed Data (Stateful Application)
4. Sharing Data between Containers
5. Mounting Application Logs to the Host ***

# Mounting Data from Host

```
docker run ... -v ${PATH_HOST}:${PATH_CONTAINER} <<image_name>>
docker run ... -v ${PATH_HOST}:${PATH_CONTAINER}:ro <<image_name>>
```

## Examples:

Run httpd service with custom "index.html"

```
# ls -l ./
total 40
-rw-r--r--   1 sbeliakou  wheel  119 July 25 22:05 index.html


# docker run -d -P -v $(pwd):/var/www/html httpd
# docker run -d -P -v $(pwd):/var/www/html:ro httpd

# docker run -d -p 127.0.0.1:8080:80 -v $(pwd):/var/www/html:ro httpd
```

Run nginx service with custom config

```
# docker run -d -v $(pwd)/nginx.conf:/etc/nginx/nginx.conf nginx
```

# Use Cases

```
# docker run --rm -v $(pwd):$(pwd) -w $(pwd) maven:3.3-jdk-8 clean package
```

```
# docker run -d -v /var/log/httpd:/var/log/httpd httpd
```

```
# docker run -d -v /var/log/tomcat:/usr/local/tomcat/logs tomcat
```

```
# docker run -d -v /data:/etc/mongo mongo
```

https://hub.docker.com/_/jenkins/

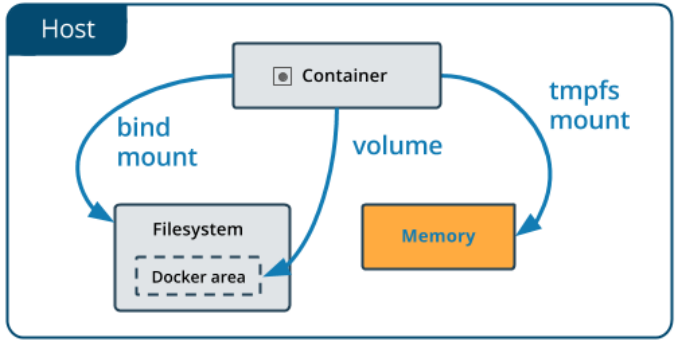
```
# docker run -p 8080:8080 -p 50000:50000 -v /your/dir:/var/jenkins_home jenkins
```

# Mounting tmpfs

A *tmpfs* mount is temporary, and only persisted in the host memory. When the container stops, the tmpfs mount is removed, and files written there won't be persisted.

Limitations of *tmpfs* mounts:

- You can't share tmpfs mounts between containers.
- This functionality is only available if you're running Docker on Linux.




```
# docker run -d \
  --mount type=tmpfs,destination=/app \
  nginx:latest
```

```
# docker run -d \
  --tmpfs /app \
  nginx:latest
```

https://docs.docker.com/storage/tmpfs/#use-a-tmpfs-mount-in-a-container

# to be continued

Thank you for your attention!

*Siarhei Beliakou,*

*2019*