docker

Agenda

- Containers VS Virtual Machines
- Container technology
- Docker architecture
- Docker Images
- Docker Containers
- Install Docker
- Images Commands
- Containers Commands
- Conclusion

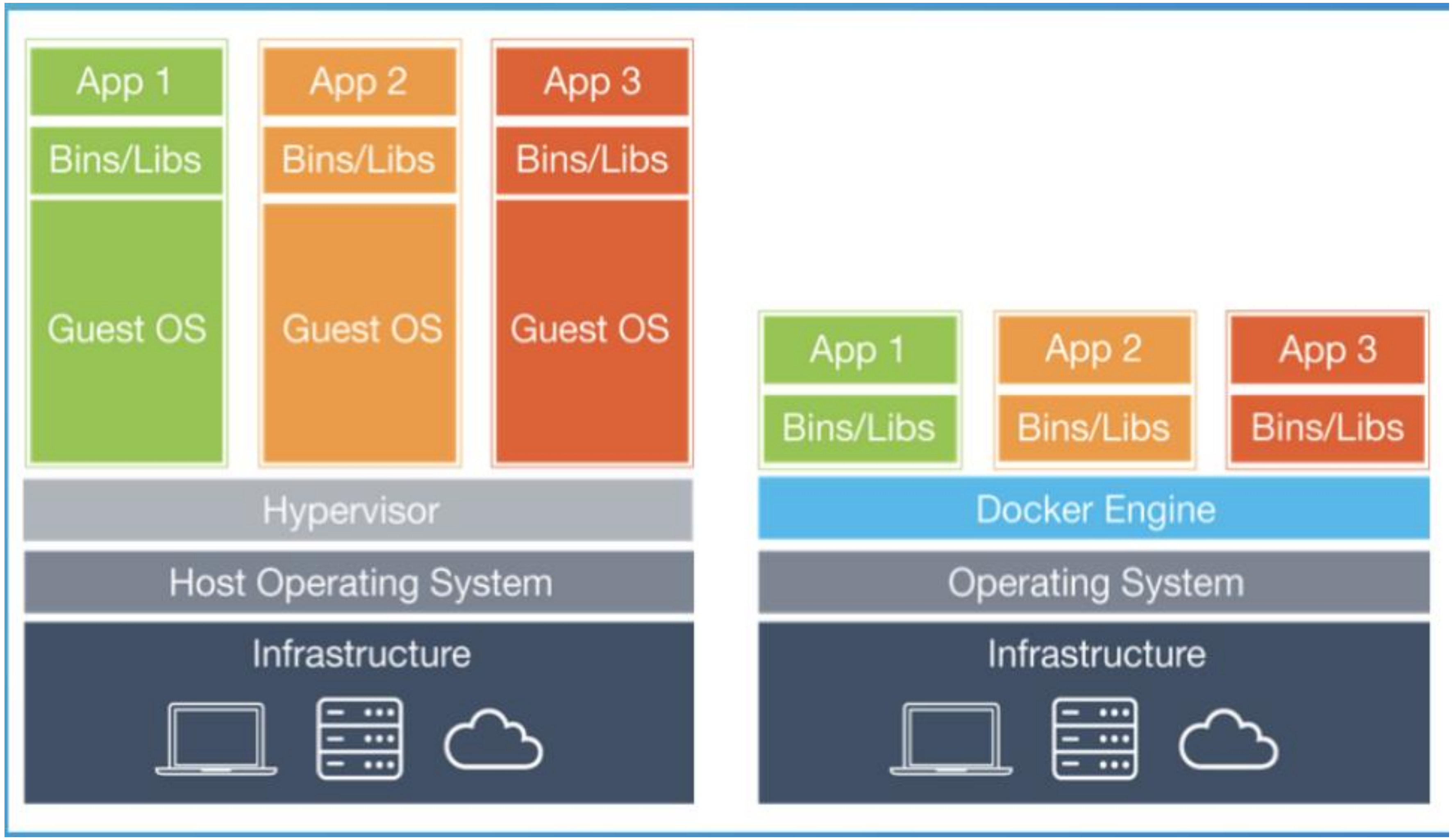
Agenda

- Docker file basics & Sample
- Building Docker file, tagging and creating image
- Logging to your Docker repo from CLI
- Pushing/Pulling image to docker repo

Day 2

- Docker Volume
- Docker Networking
- Docker Compose
    - Docker Compose Basics
    - Docker Compose Commands

Virtual Machines VS Containers



Virtual Machines VS Containers

VMs	Containers
Heavyweight	Lightweight
Limited performance	Native performance
Each VM runs in its own OS (More secure and isolated)	All containers share the host OS
Slower Startup	Faster Startup
More memory is required	Fewer memory is required

Container technology

Namespaces

A namespace isolates specific system resources usually visible to all processes. Inside a namespace, only processes that are members of that namespace can see those resources. Namespaces can include resources like network interfaces, the process ID list, mount points, IPC resources, and the system's host name information.

Control groups (cgroups)

Control groups partition sets of processes and their children into groups to manage and limit the resources they consume. Control groups place restrictions on the amount of system resources processes might use. Those restrictions keep one process from using too many resources on the host.

Container History cont'd

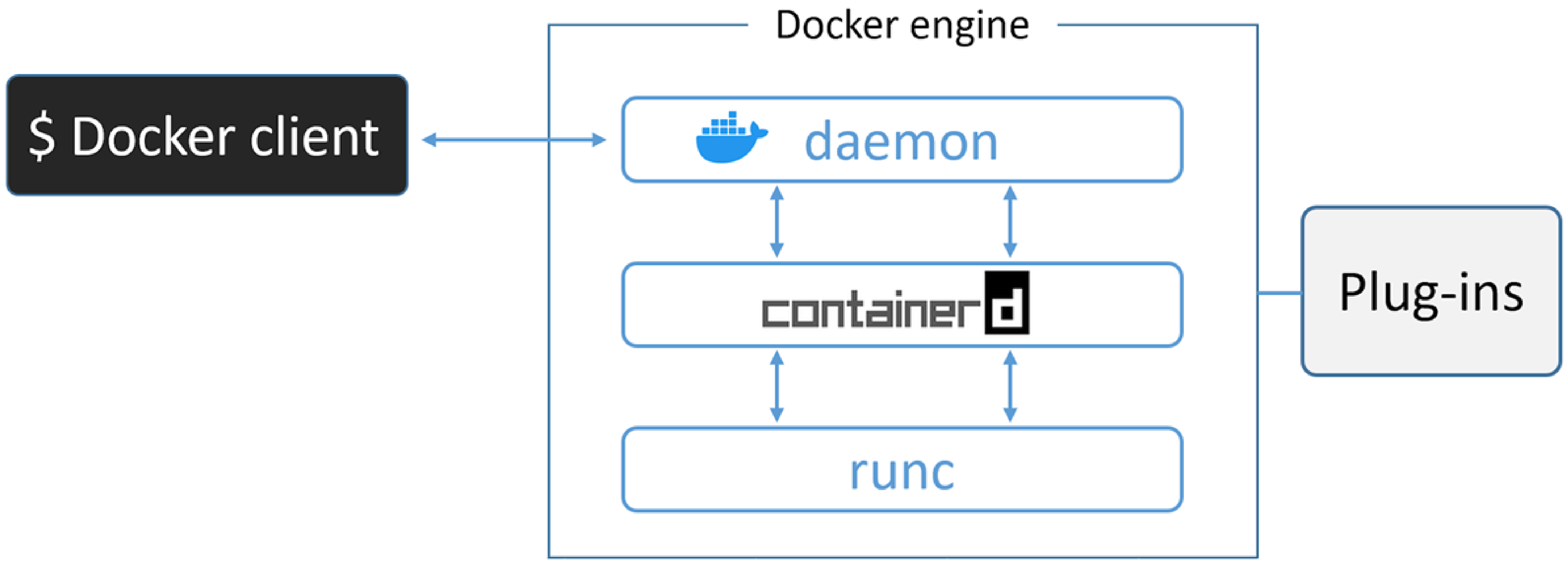
Seccomp

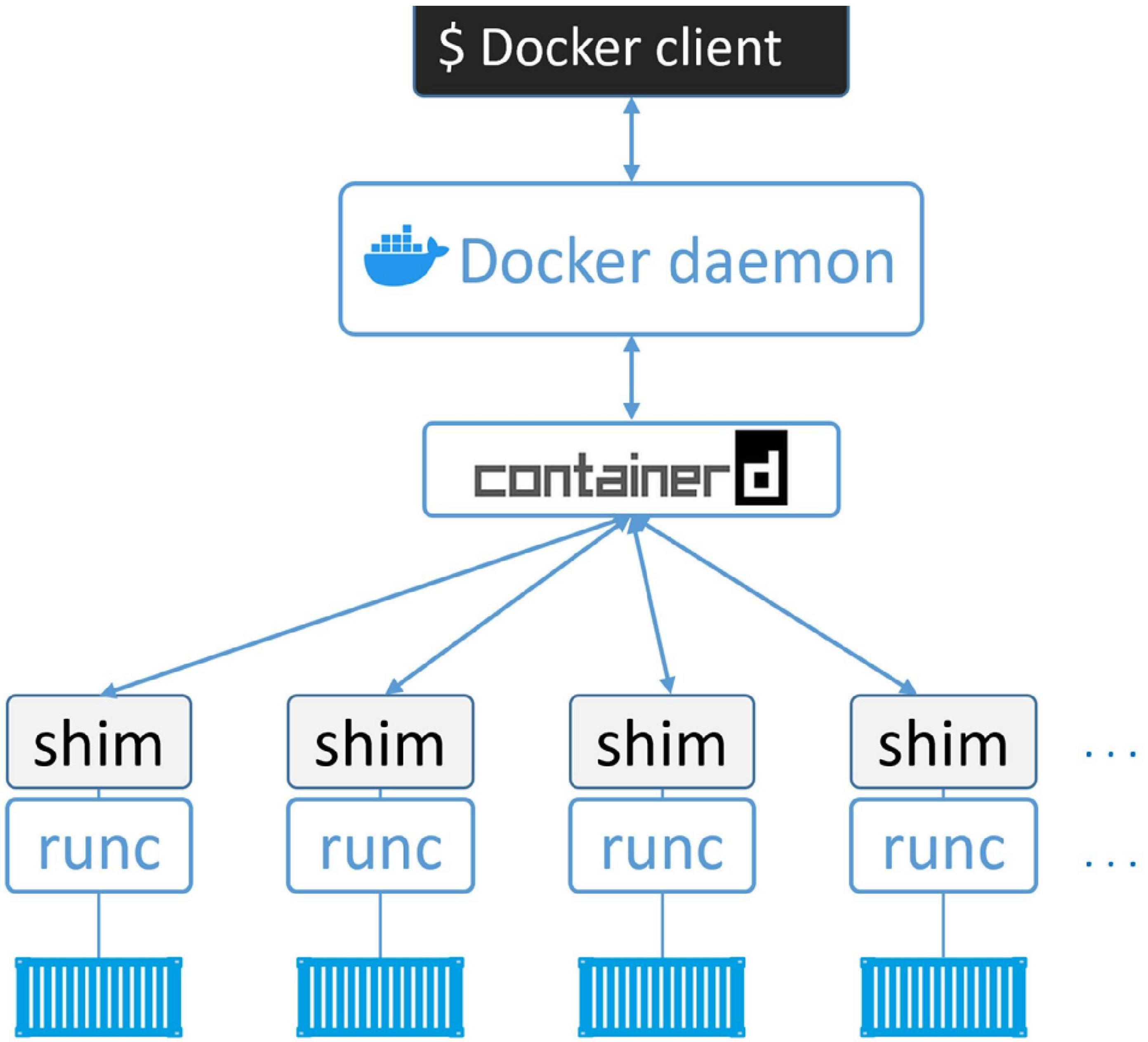
Developed in 2005 and introduced to containers circa 2014, Seccomp limits how processes could use system calls. Seccomp defines a security profile for processes that lists the system calls, parameters and file descriptors they are allowed to use.

SELinux

Security-Enhanced Linux (SELinux) is a mandatory access control system for processes. Linux kernel uses SELinux to protect processes from each other and to protect the host system from its running processes. Processes run as a confined SELinux type that has limited access to host system resources

Docker architecture





Docker commands (CLI)

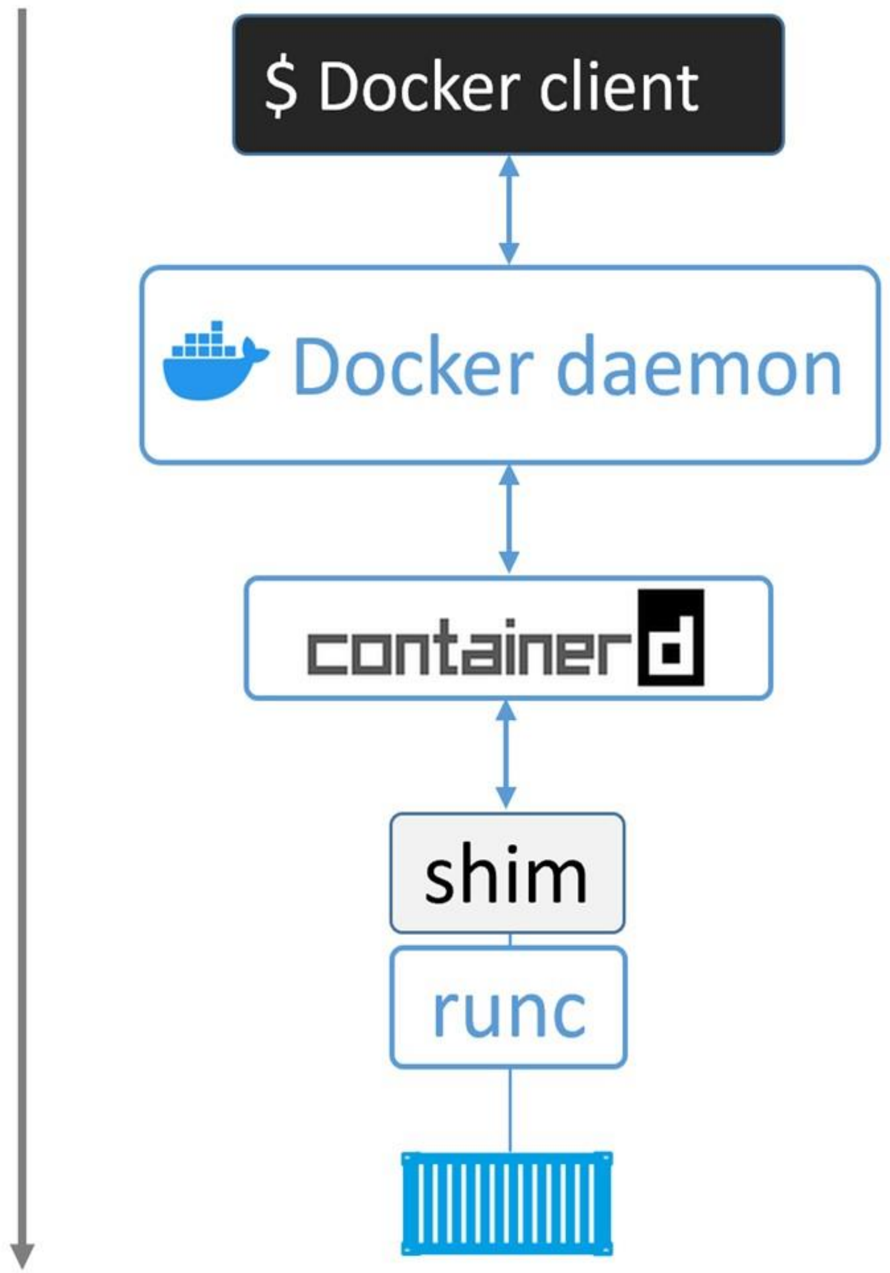
API and other features

Container supervisor
`start|stop|pause..`

Enables daemonless containers

Container runtime
(interface to kernel primitives)

Running containers



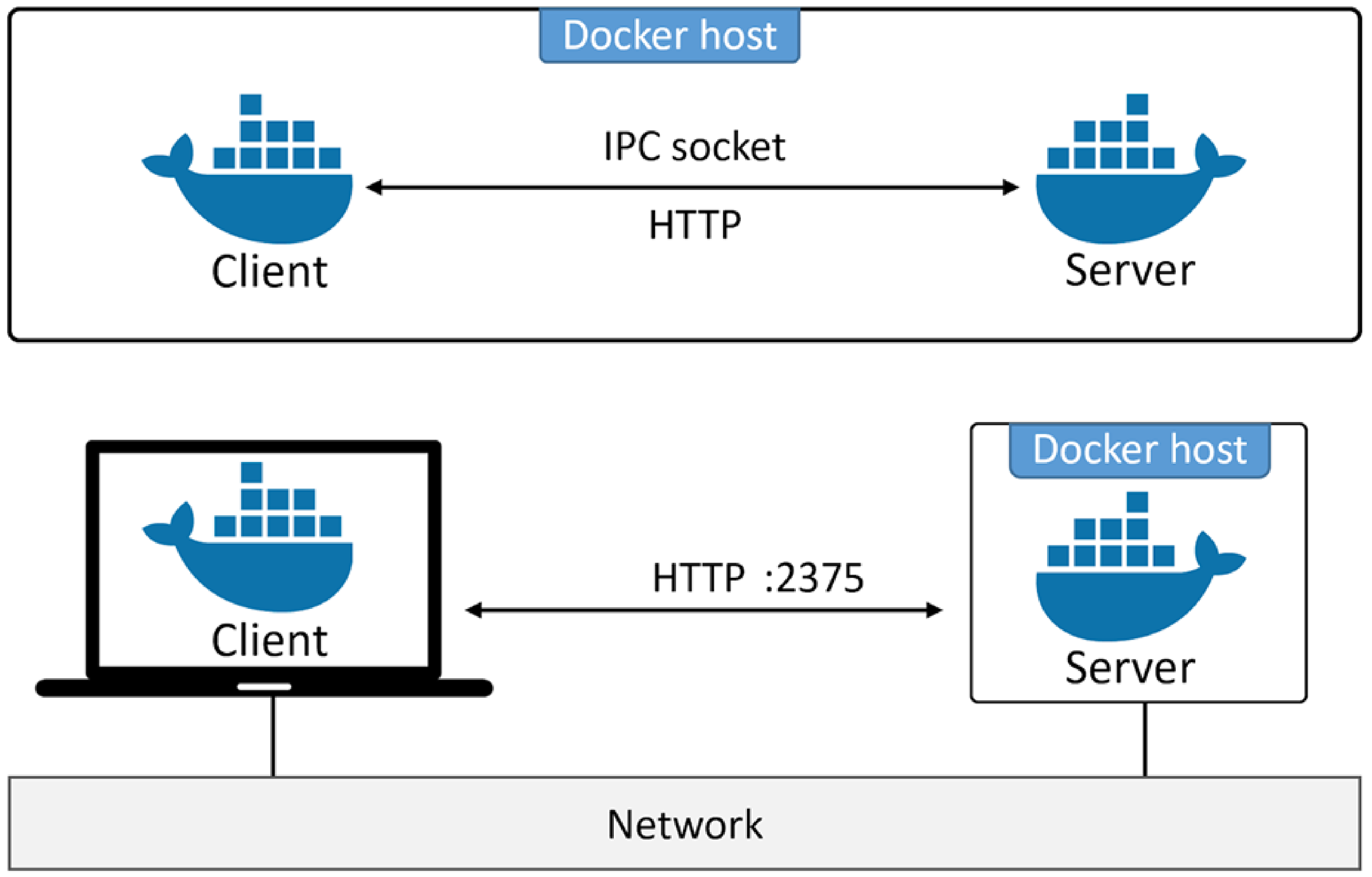
Issue `docker container run` command to **Docker API** exposed by Docker daemon

Receive instruction at API endpoint.
Instruct **containerd** (via gRPC API) to start new container based on OCI bundle and ID provided

Receive instruction to create containers
Instruct **runc** to create container.

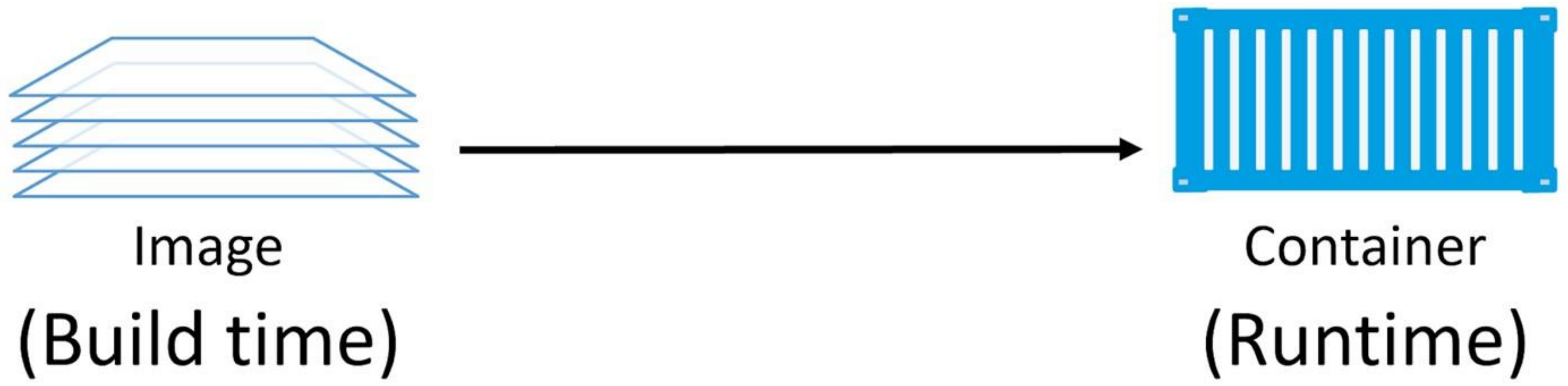
Build and start container
runc exit after container start
shim become container's parent process

Securing client and daemon communication

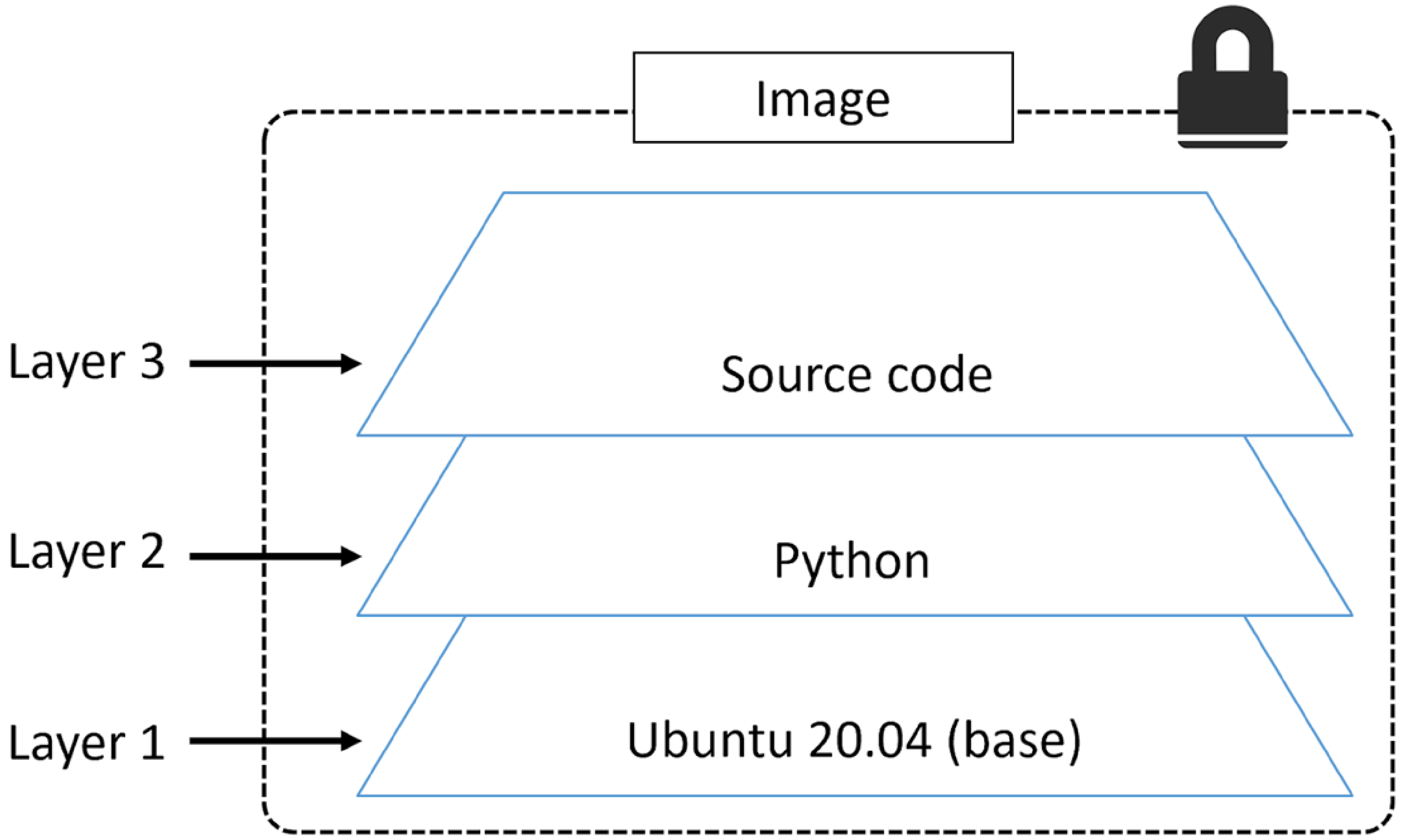


Docker Images

Read-only template that contains a set of instructions for creating a container that can run on the Docker platform.



Images and layers



Images and layers

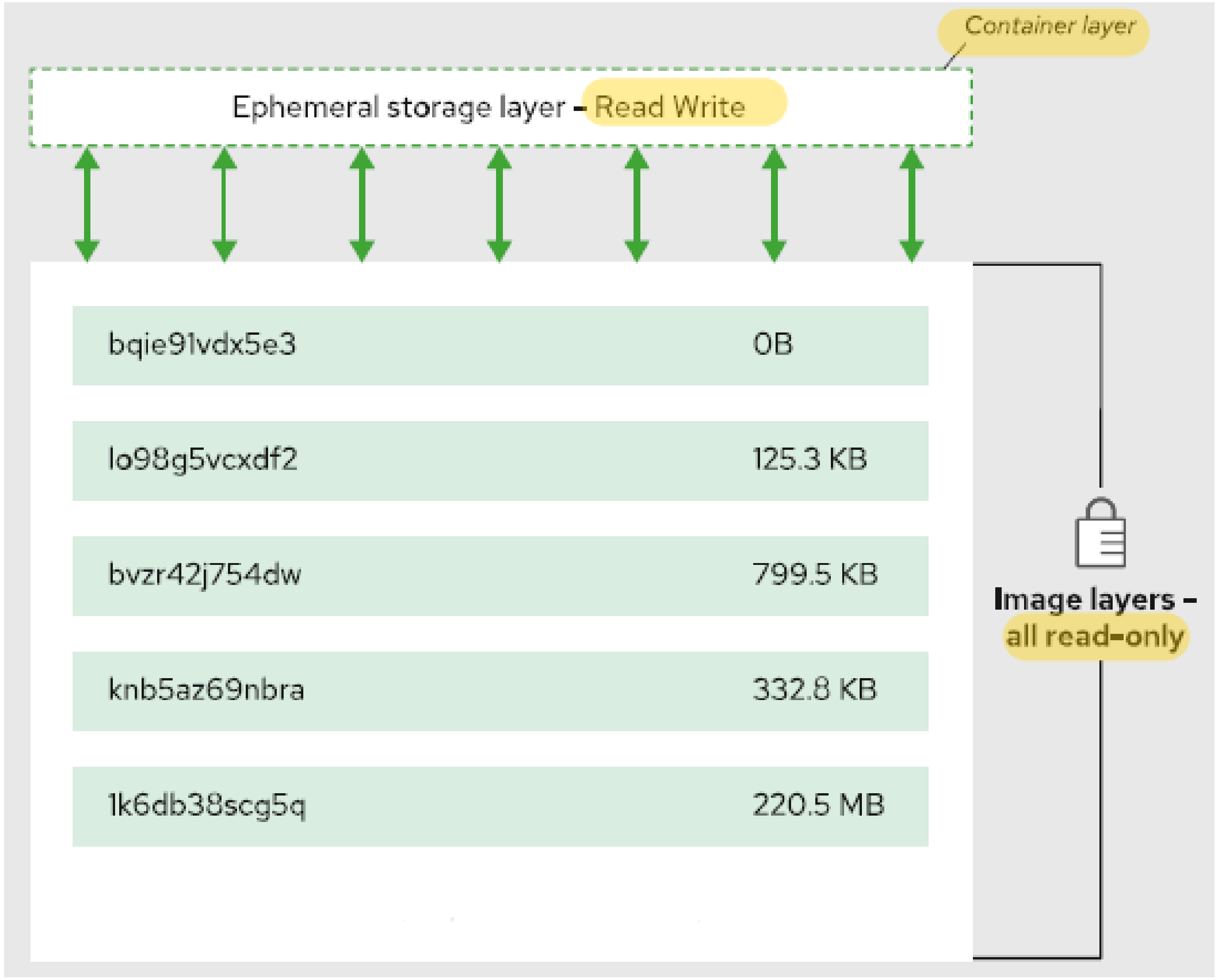
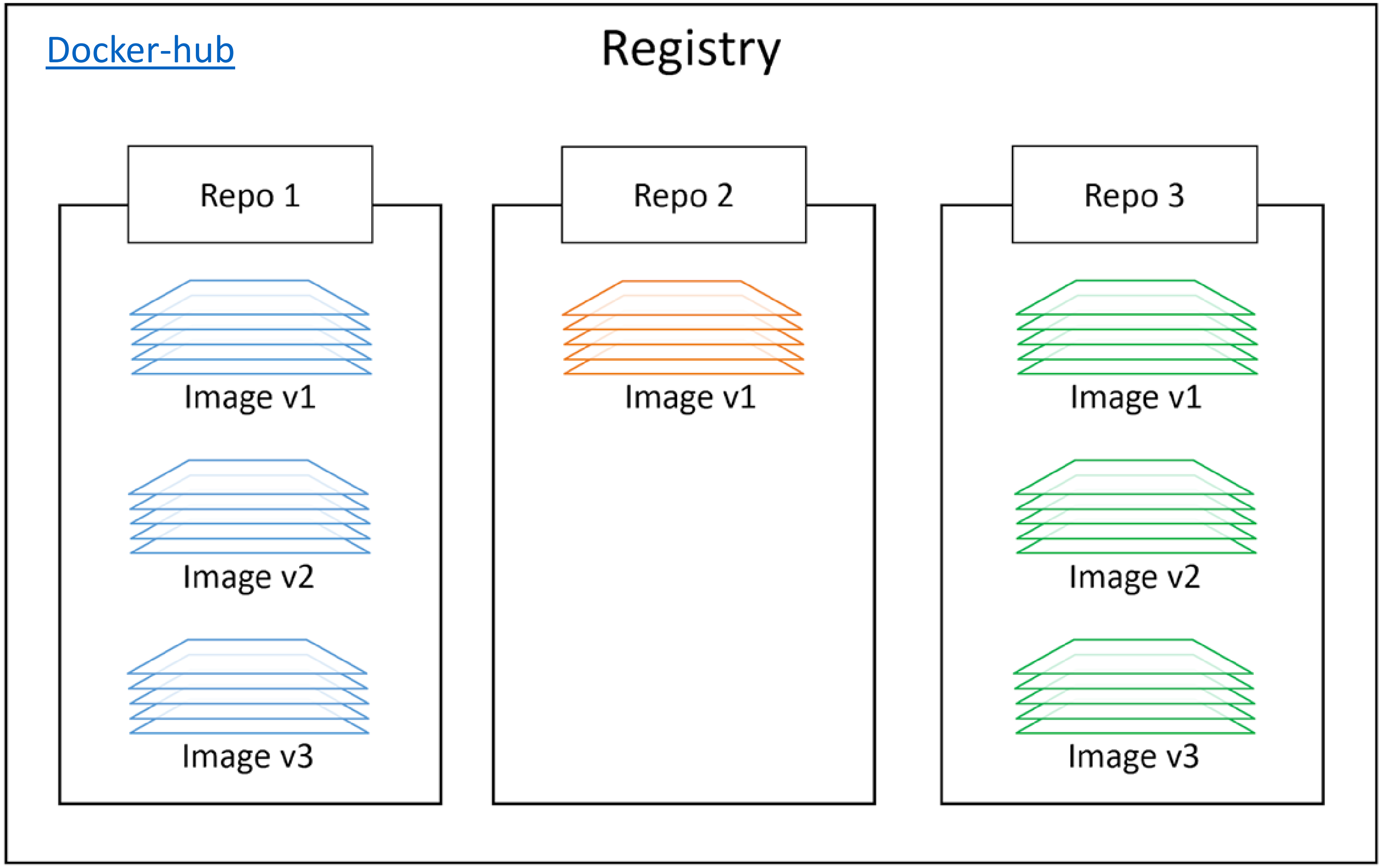
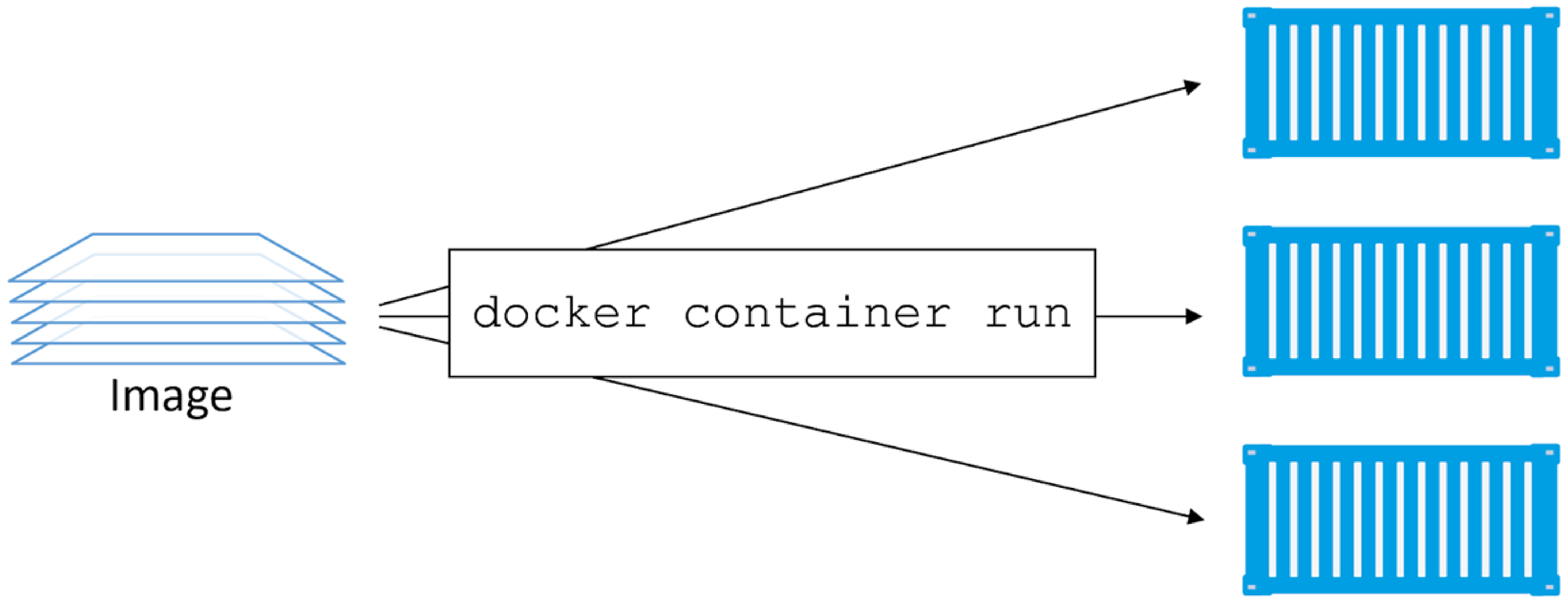


Image Registries



Docker Containers



Self-healing containers with restart policies

Always

- container will always restart if the main process is killed from inside the container but won't restart if you manually stopped it. Will restart if the Docker daemon restarts

unless-stopped

- container will always restart if the main process is killed from inside the container but won't restart if you manually stopped it. However will NOT restart if the Docker daemon restarts.

on-failure

- container will always restart if the main process exits with non-zero code (i.e. with error) but won't restart if you manually stopped it. However will NOT restart if the Docker daemon restarts

Install Docker

[How To Install Docker]

[Docker-hub]

Images Commands

docker images shows all images.

docker rmi removes an image.

docker search image search

docker history shows history of image.

docker tag tags an image to a name (local or registry).

Images Commands cont’d

docker commit creates image from a container, pausing it temporarily if it is running.

Ex: docker commit [OPTIONS] CONTAINER_ID [REPOSITORY[:TAG]]

docker save saves an image to a tar archive stream to STDOUT with all parent layers, tags & versions

Ex : sudo docker save busybox-1 > /home/save.tar

docker load loads an image from a tar archive as STDIN, including images and tags (as of 0.7).

N.B: All the above commands will require the IMAGE_ID

Containers Commands

docker start/stop/restart starts/stops/restarts a container.

docker run creates and starts a container in one operation.

Ex: docker run -p $HOSTPORT:$CONTAINERPORT --name CONTAINER some image

Ex: docker run –it IMAGE bash   (it will open the container)

docker ps shows running containers.

docker ps -a shows running and stopped containers.

docker logs gets logs from container.

Containers Commands cont’d

docker inspect looks at all the info on a container (including IP address).

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.

Containers Commands cont'd

docker export – Exports a container's filesystem as a tar archive

Ex: docker export <CONTAINER ID> > /home/export.tar

docker import creates an image from a tarball.

docker cp copies local file to container and vice versa.

Ex: docker cp [OPTIONS] CONTAINER_ID:SRC_PATH DEST_PATH

Ex: docker cp [OPTIONS] SRC_PATH CONTAINER_ID:DEST_PATH

docker exec to execute a command inside a container.

Ex: docker exec –it CONTAINER_ID bash

N.B: All the above commands will require the CONTAINER_ID

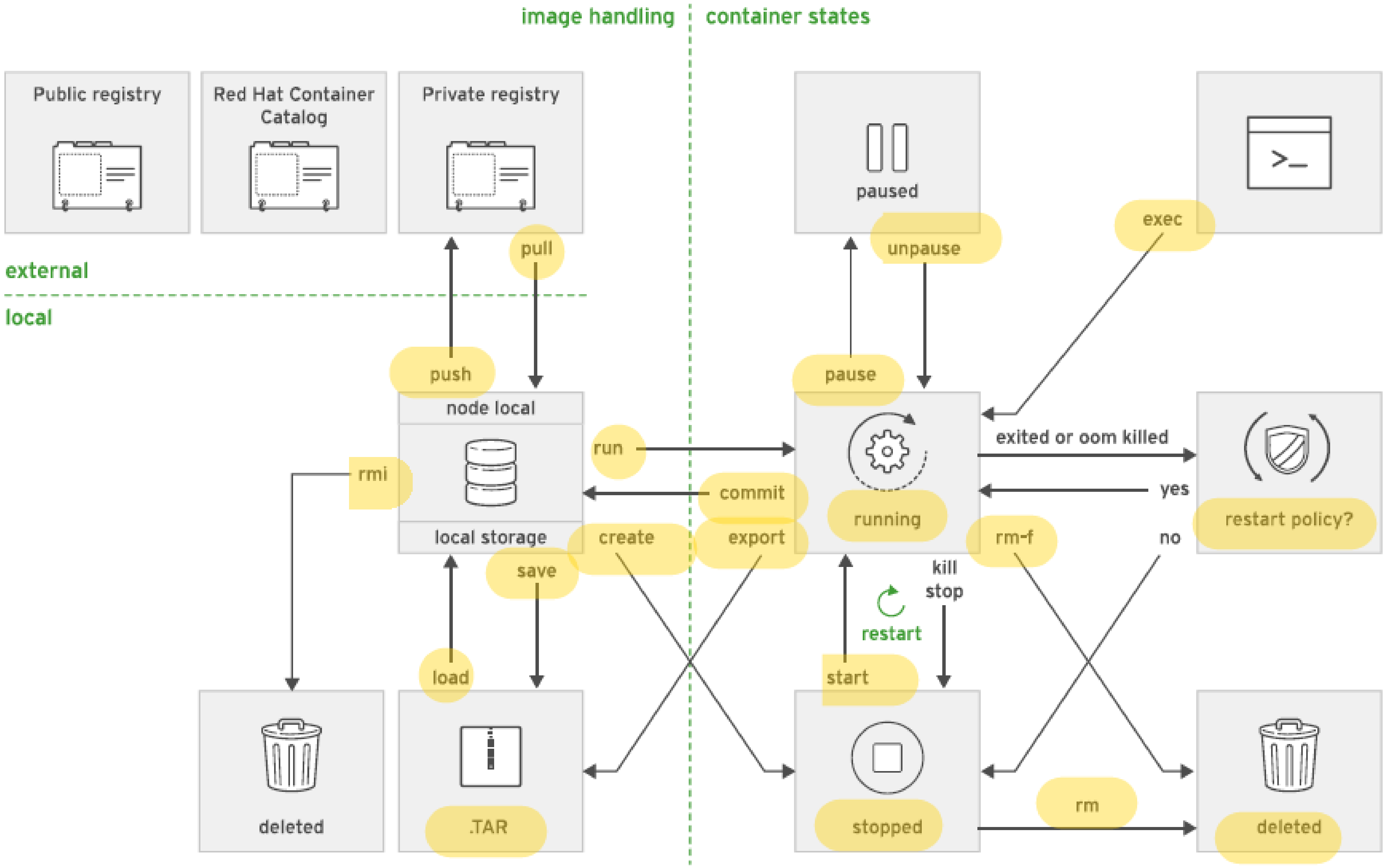


image handling
container states
Public registry
Red Hat Container Catalog
Private registry
paused
external
local
pull
push
unpause
exec
pause
node local
local storage
run
commit
create
export
save
load
rmi
running
exited or oom killed
yes
no
restart policy?
rm-f
kill
stop
restart
start
deleted
.TAR
stopped
rm
deleted

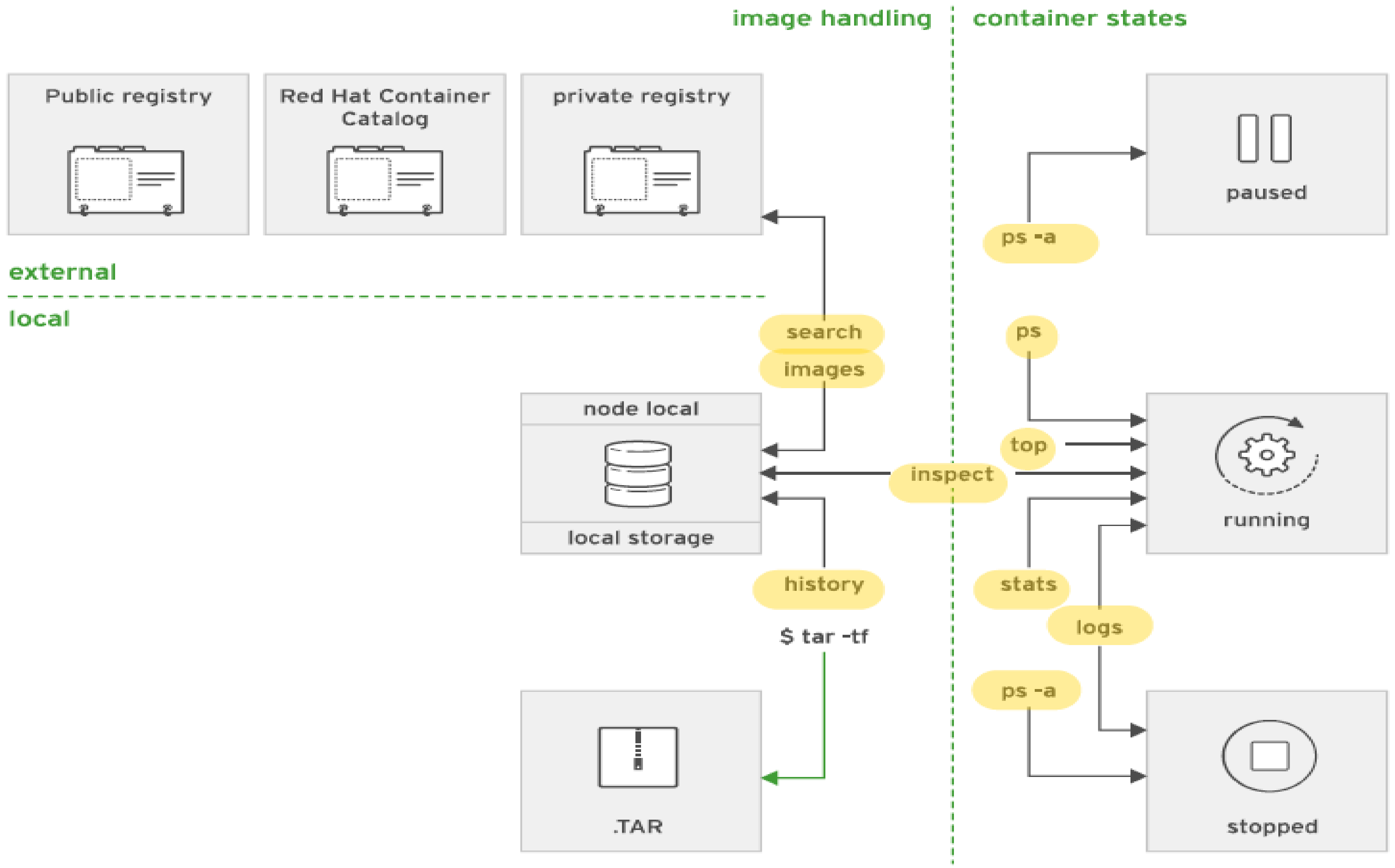


image handling
container states
Public registry
Red Hat Container Catalog
private registry
external
local
search
images
node local
local storage
history
$ tar -tf
.TAR
inspect
ps -a
paused
ps
top
running
stats
logs
ps -a
stopped

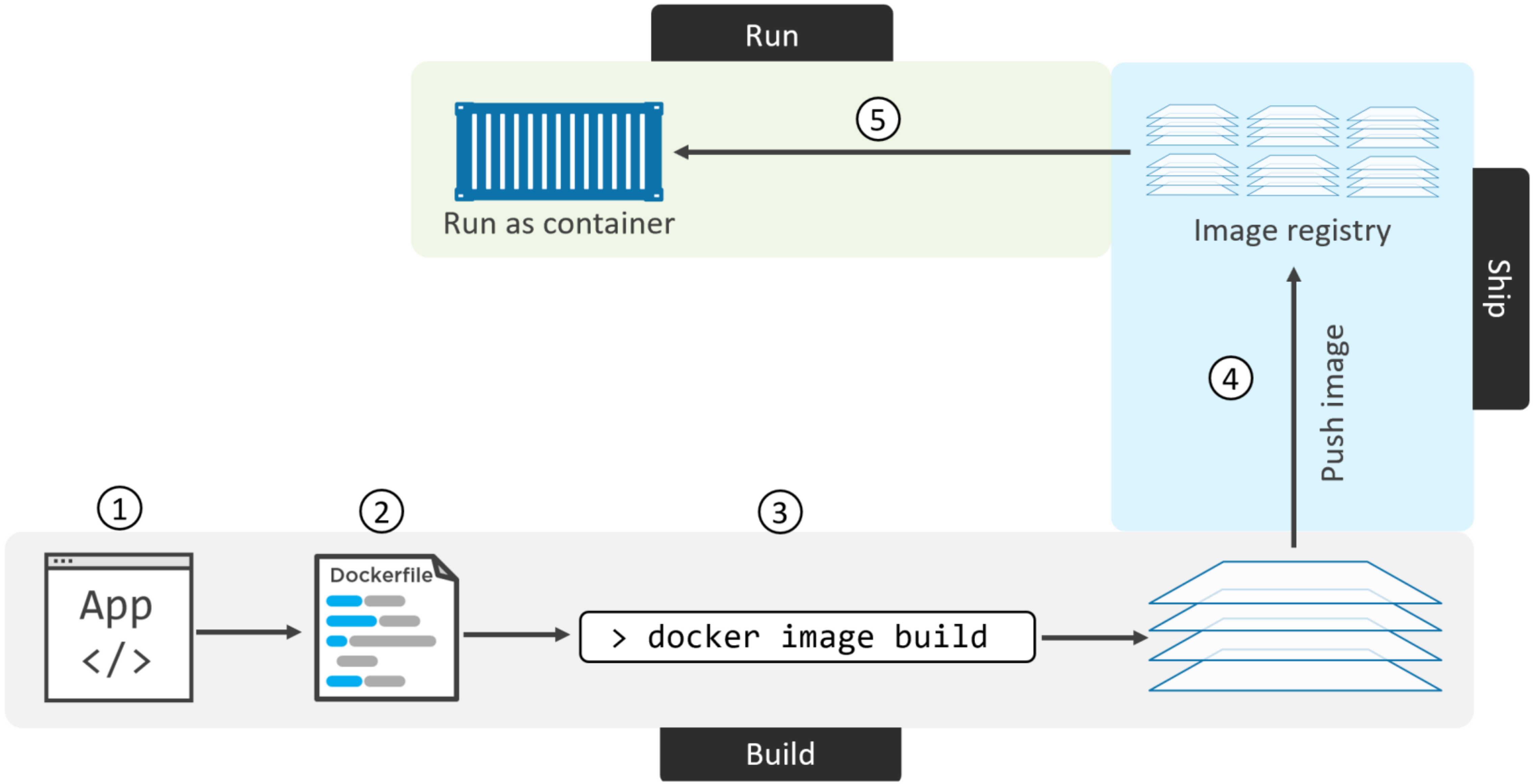
Docker file

Dockerfile is a script that contains instructions for building a customized docker image.

Each instruction in a Dockerfile creates a new layer in the image, and the final image is composed of all the layers stacked on top of each other

It includes instructions for installing dependencies, copying files, setting environment variables, and configuring the container

Docker file


Run
⑤
Run as container
Image registry
Ship
④
Push image
①
②
③
App
</>
Dockerfile
> docker image build
Build

Dockerfile basics

FROM Sets the Base Image for subsequent instructions, you should write this command at least in Dockerfile

RUN execute any commands in a new layer on top of the current image and commit the results. (Used mainly for installing packages)

CMD provide defaults for executing command inside a container.

EXPOSE informs Docker that the container listens on the specified network ports at runtime. NOTE: does not actually make ports accessible.

Dockerfile basics cont’d

ENV sets environment variable.

COPY copies new files or directories to container Note that this only copies as root, so you have to chown manually regardless of your USER / WORKDIR setting. See https://github.com/moby/moby/issues/30110

ADD lets you do that too, but it also supports 2 other sources. First, you can use a URL instead of a local file / directory. Secondly, you can extract a tar file from the source directly into the destination.

Dockerfile basics cont'd

ENTRYPOINT configures a container that will run as an executable = CMD

VOLUME creates a mount point for externally mounted volumes or other containers.

USER sets the user name for following RUN / CMD / ENTRYPOINT commands, By default it run commands as a root

WORKDIR sets the working directory, By default it run commands in home directory

ARG defines a build-time variable.

Dockerfile Sample

```
FROM  node:14-alpine3.16

WORKDIR  /app

COPY   .  .

RUN   npm install

CMD  [ "npm", "start" ]
```

Building Docker file, tagging and creating image

Traditionally, the Dockerfile is called ”Dockerfile”

build Dockerfile > docker build .(current dictory)

build Dockerfile with image tag add “-t” > docker build –t “tag” .(current dictory)

If the Dockerfile is not named as “Dockerfile”

build Dockerfile > docker build -f /path/to/a/Dockerfile

build Dockerfile with image tag add “-t” > docker build –t “tag” -f /path/to/a/Dockerfile

Logging to your Docker repo from CLI

docker login “URL”
By Default, docker login > redirects login to docker hub
Enter your username and password

Pushing/Pulling image to docker repo

```
docker tag local-image:tagname remote-repo:tagname
docker push remote-repo:tagname
docker pull new-repo:tagname
```

```
docker tag     image-id  hossamesaaa/firstproject:hossam
docker push  hossamesaaa/firstproject:hossam
docker pull   hossamesaaa/firstproject:hossam
```