DevOpsPro

Docker Security Workshop

Vilnius

14th March 2018

Agenda

- Lab setup
- Back to Basics
- Images
- Swarm
- Networking
- Advanced
- Opensource Tools
- Secure Supply Chain

Rachid Zarouali

C.I.O Synolia

Docker Community Leader

Docker Certified Associate

Twitter / Slack : @xinity

rzarouali@gmail.com

Goals of this workshop

Docker Security Features :

- Swarm Security
- Secrets Management
- Security Scanning
- Image Signing
- Docker Content Trust
- Secure Networking

Linux Security Features :

- Kernel Capabilities
- Seccomp
- AppArmor

Lab Setup

Chat and stuff :)

Chat room : https://gitter.im/DevopsProSecurity/Lobby#

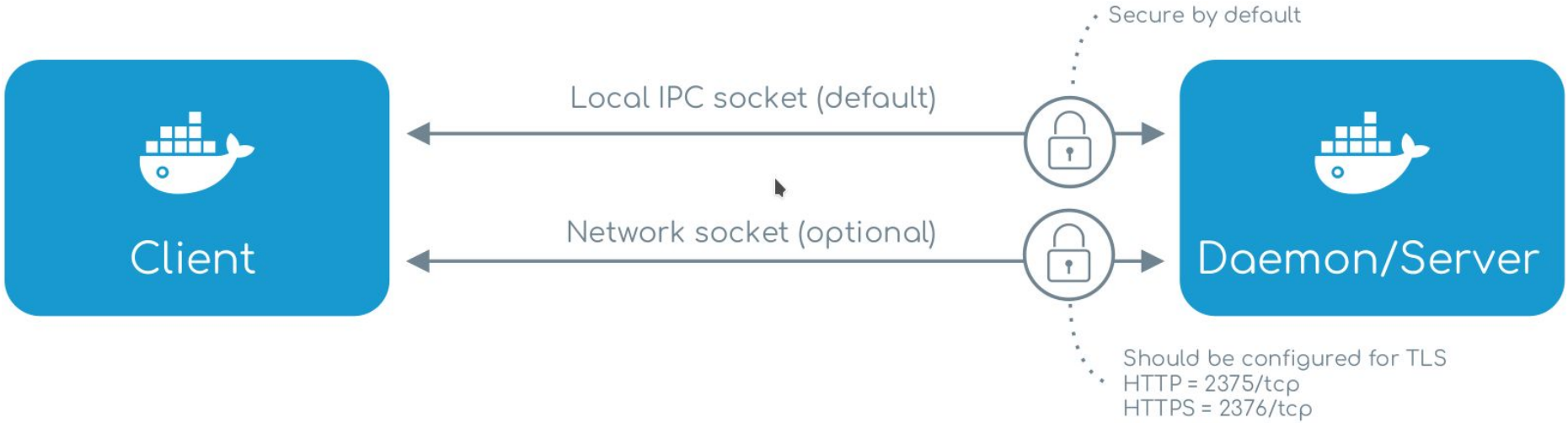
PWD instances: https://goto.docker.com/DevopsPro_LP.html

Github labs: https://github.com/xinity/security-workshop

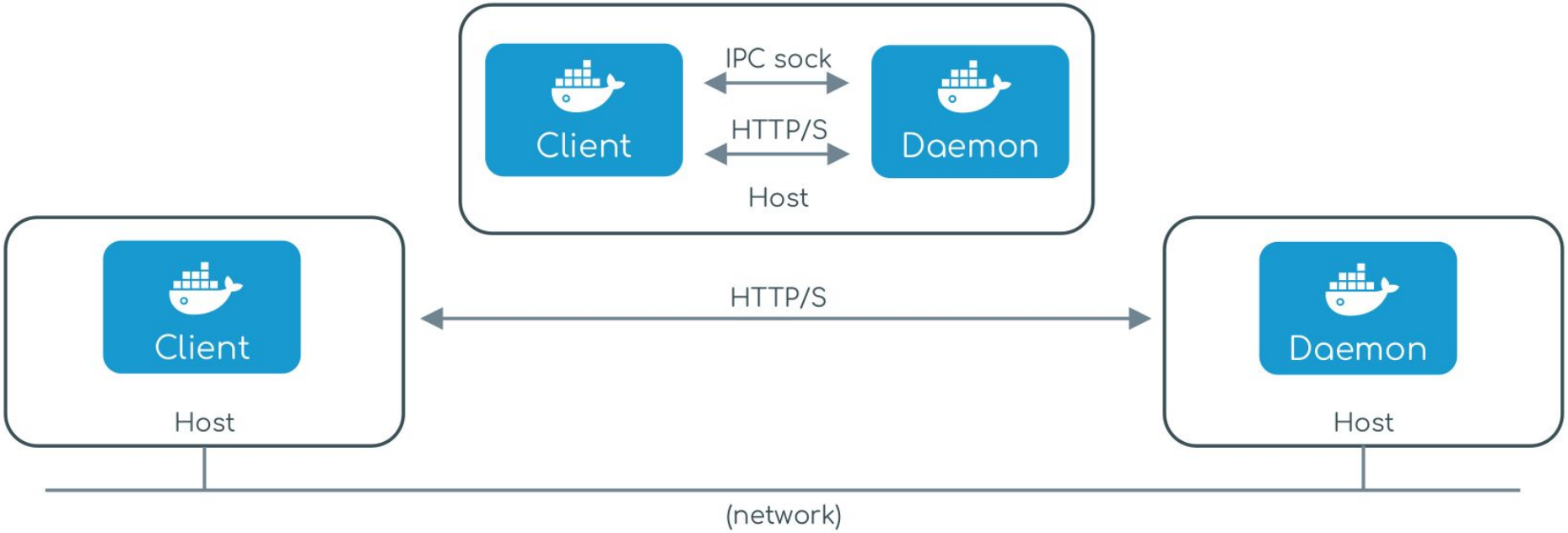
Back to basics

Docker Client-Server Architecture

Docker Client-Server Architecture



Docker Client-Server Architecture



Docker Client-Server Architecture

Docker Client-Server Architecture

Communication with Registry **must** be secured using http/s

Use PKI infrastructure for Self-Signed certificates

MTLS **highly** recommended

Back to basics

Anatomy of a container

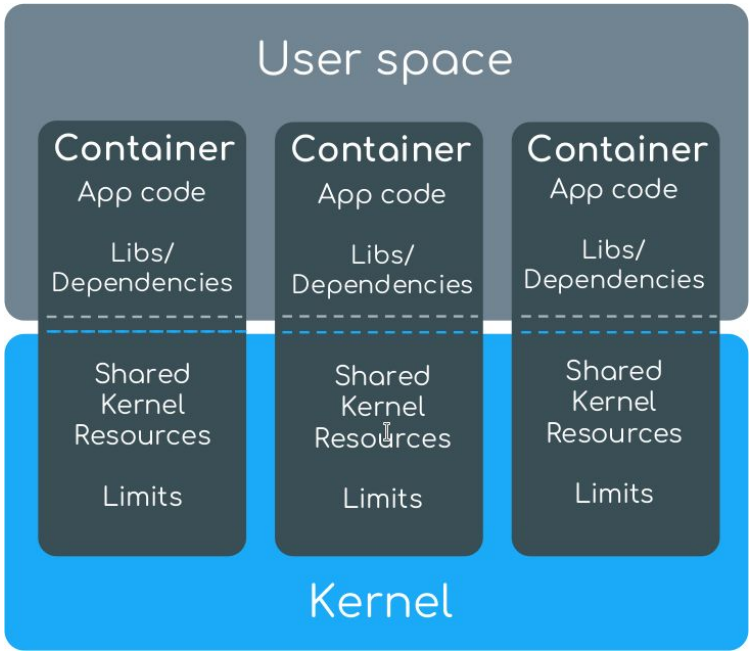
Anatomy of a container

User Space:

- Binaries
- Libraries
- Dependencies

Kernel Space:

- Process tree
- Filesystem root
- Network
- Limits on resource



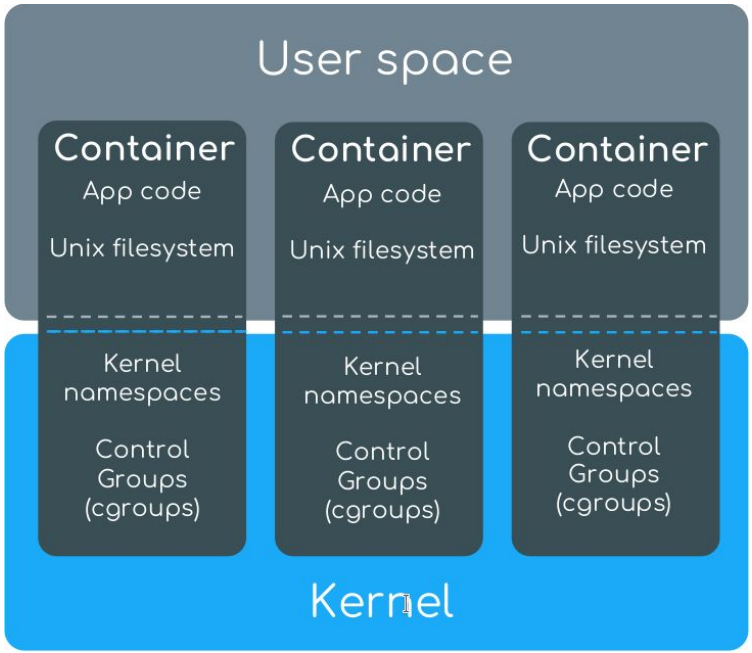
Anatomy of a container

Namespaces:

- The PID namespace stops processes in one container from seeing processes in another container (or on the host)
- The User namespace allows containers to run processes as root inside the container but as non-privileged users outside the container (on the host)

Cgroups:

- Can limit the amount of CPU or memory a container can use, and prevent them from consuming all system resource



Anatomy of a container

Namespaces

Cgroup	Cgroup root directory
IPC	System V IPC, POSIX message queues
Network	Network devices, stacks, ports, etc.
Mount	Mount points
PID	Process IDs
User	User and group IDs (*disabled by default*)
UTS	Hostname and NIS domain name

Cgroups

Memory
CPU
Blkio
Cpuacct
Cpuset
Devices
Net_prio
Freezer

Back to basics

Lab 1: cgroups

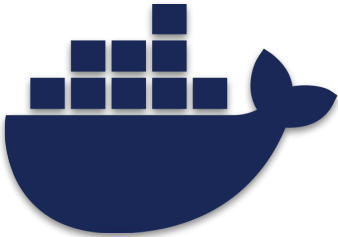
Lab1: Cgroups

$ git clone https://github.com/xinity/security-workshop.git

$ cd cgroups

$ vi README.md

Approximate time: 20 minutes



Docker images

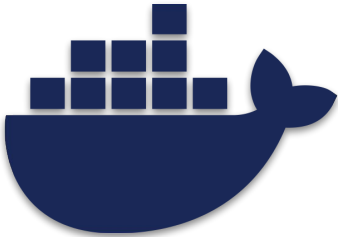
Build, Ship, Run

Docker images

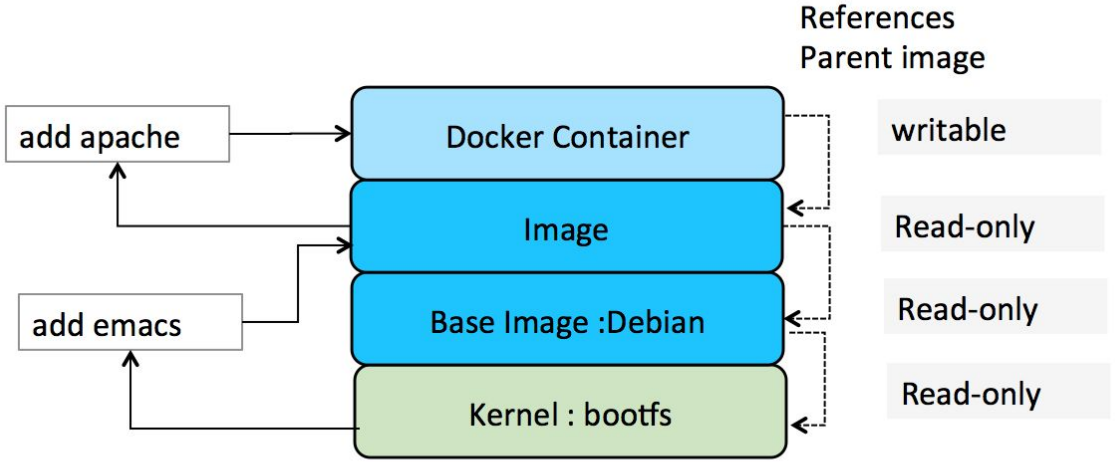
Image: several read-only layers build from a Dockerfile

Container: Image + read-write layer

I.e: `docker run -ti debian:stretch-slim bash`



Docker images



Docker images

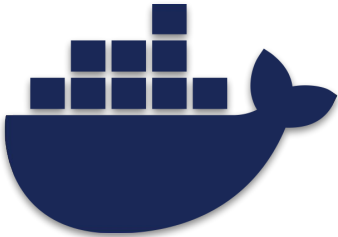
More layers mean :

- larger image
- longer build, push and pull from a registry

Small images mean :

- faster build and deploy
- small surface of attack

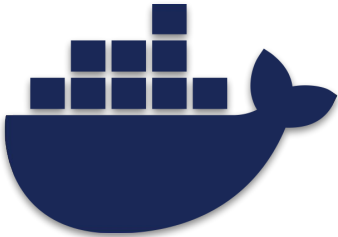
Q: Are small images always secure ?



Docker images

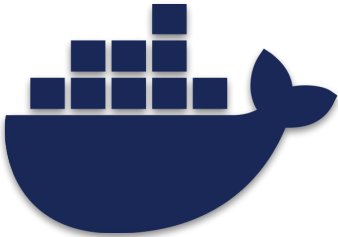
How to reduce layers number:

- Share base image as much as possible
- Limit data written to the layer (nothing unnecessary)
- Chain RUN statements



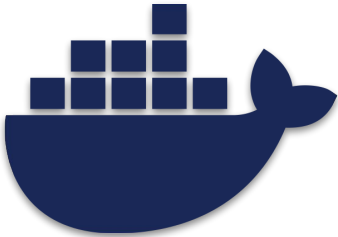
Docker images

```
FROM ubuntu:latest
LABEL maintainer abbyfull@amazon.com
RUN apt-get update -y && apt-get install -y python-pip
python-dev build-essential
COPY . /app
WORKDIR /app
RUN pip install –r requirements.txt
EXPOSE 5000
ENTRYPOINT ["python"]
CMD ["application.py"]
```



Docker images

```
FROM python:2.7-alpine
LABEL maintainer abbyfull@amazon.com
ONBUILD ADD requirements.txt /app
ONBUILD RUN pip install –r /app/requirements.txt
ONBUILD COPY . /app
WORKDIR /app
EXPOSE 5000
ENTRYPOINT ["python"]
CMD ["application.py"]
```



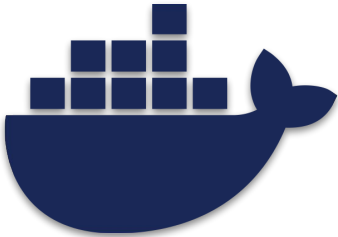
Docker images

Use minimalist base images

- Smaller images reduce the attack surface
- The official Alpine base image is <5MB‘

Use official images as base images

- All official images are scanned for vulnerabilities
- Usually follow best practices



Docker images

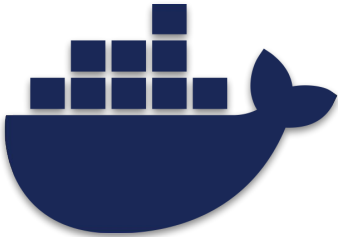
Pull images by digest ‘Image digests are a hash of the image’s config object)

- This makes them immutable
- If the contents of the image are changed/tampered with, the digest will be different

```
$ docker pull alpine@sha256:3dcdb92...b313626d99b889d0626de158f73a

sha256:3dcdb92d7432d...e158f73a: Pulling from library/alpine
e110a4a17941: Pull complete
Digest: sha256:3dcdb92d7432d56604...47b313626d99b889d0626de158f73a
```

⇒ If Docker Content Trust is enabled all images are pulled by digest



Docker images

Image signing

Docker Content Trust

Based on Notary

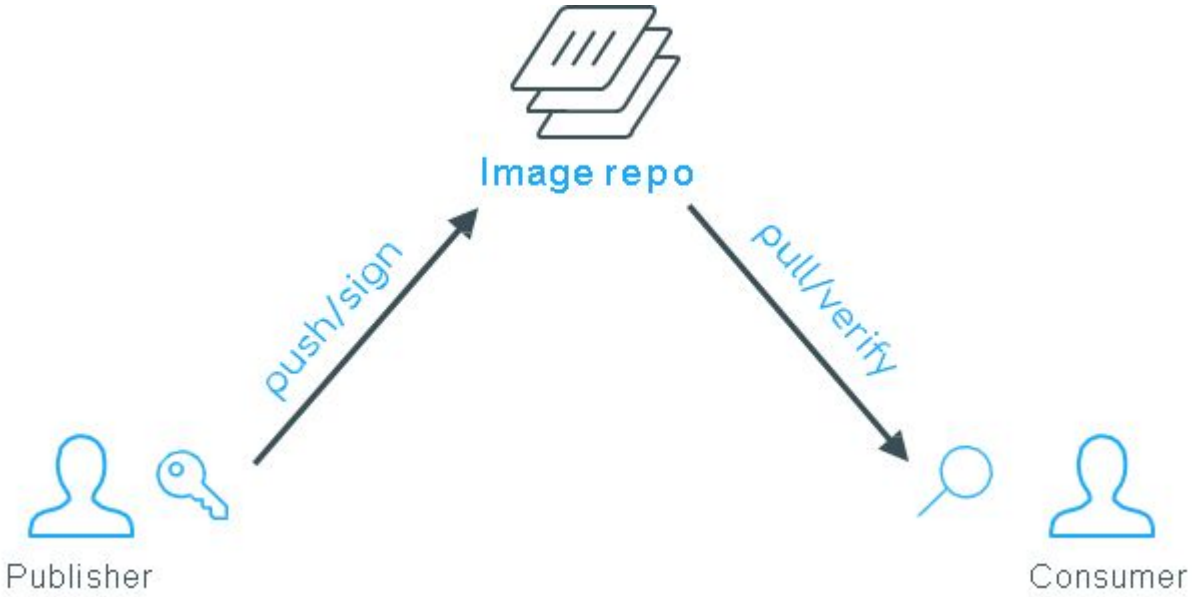
Uses a trust on first use model (TOFU)

Used to sign **and** verify an image

Secured Image Lifecycle:

- Push <-> Pull
- Build <-> Run

Docker Content Trust



Docker Content Trust

Provides:

- Digital signature mechanism
- Collaboration (signing delegation)
- Key expiration
- Collections signatures (multiple images with the same key)

Docker Content Trust

$ docker pull repo/image:unsigned

...

Error: No trust data for unsigned

⇒ Unsigned image, image pull denied

Docker Content Trust

$ docker pull repo/image:fakesignature

Warning: potential malicious behavior - trust data has insufficient signatures for remote repository docker.io/repo/image: valid signatures did not meet threshold

⇒ fake signature detected, image pull denied

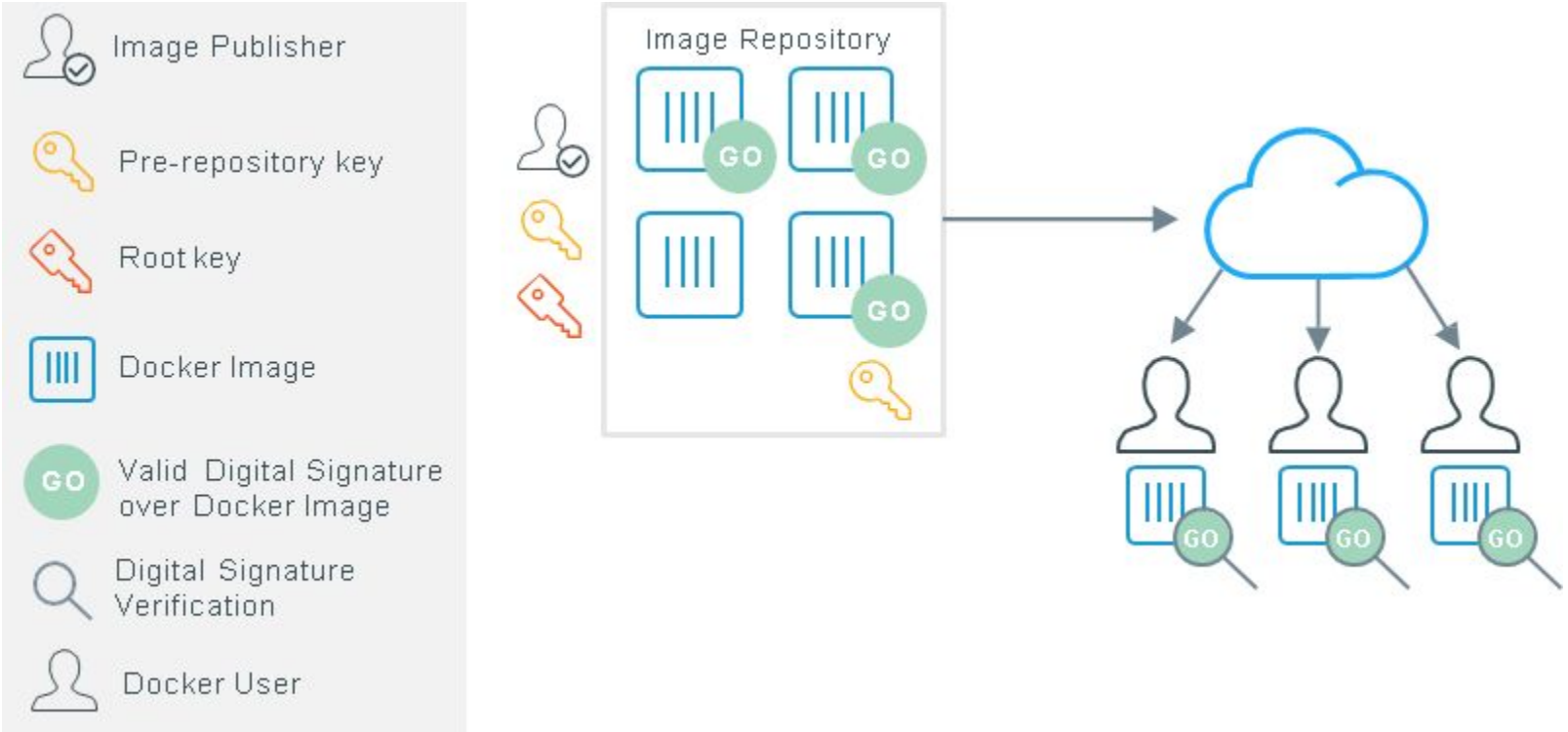
Docker Content Trust

$ docker pull repo/image:stale

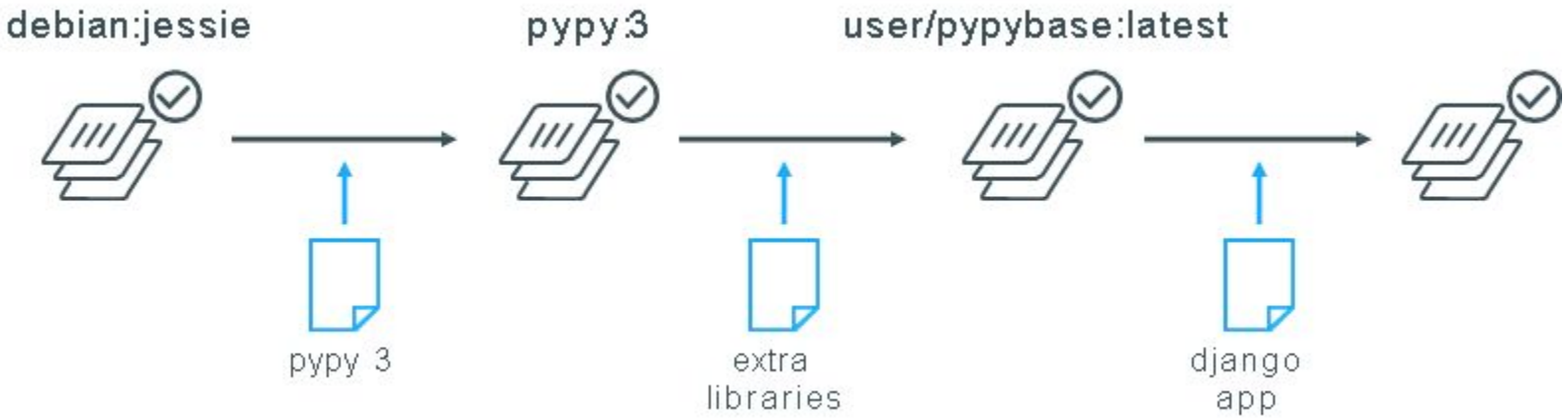
Error: remote repository docker.io/repo/image out-of-date: targets expired at Sun Mar 26 03:56:12 PDT 2017

⇒ signing key expired, image pull denied

Docker Content Trust



Docker Content Trust



Docker images

Lab 2: Image Signing

Lab2: Image Signing

$ git clone https://github.com/xinity/security-workshop.git

$ cd trust-basics

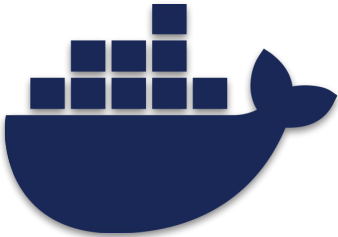
$ vi README.md

Approximate time: 15 minutes

$ cd ../trust

$ vi README.md

Approximate time: 40 minutes



Docker images

Security Scan

Security Scan

Tool/service that scans images for vulnerabilities

- Operates in the background
- Performs **deep** binary-level scanning of image layers
- Checks against database(s) of known vulnerabilities
- Provides detailed vulnerability report
- Uses CVE databases

Security Scan

Available hosted:

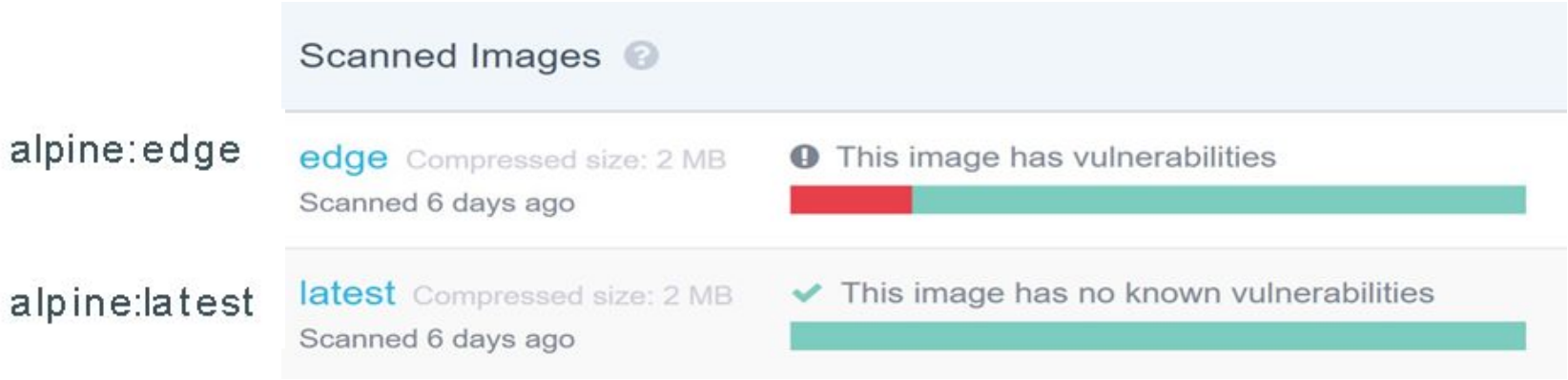
- Docker cloud
- Docker hub private repo

On premise:

- Supported: DockerEE (Advanced subscription)
- Opensource*: Harbor / Portus

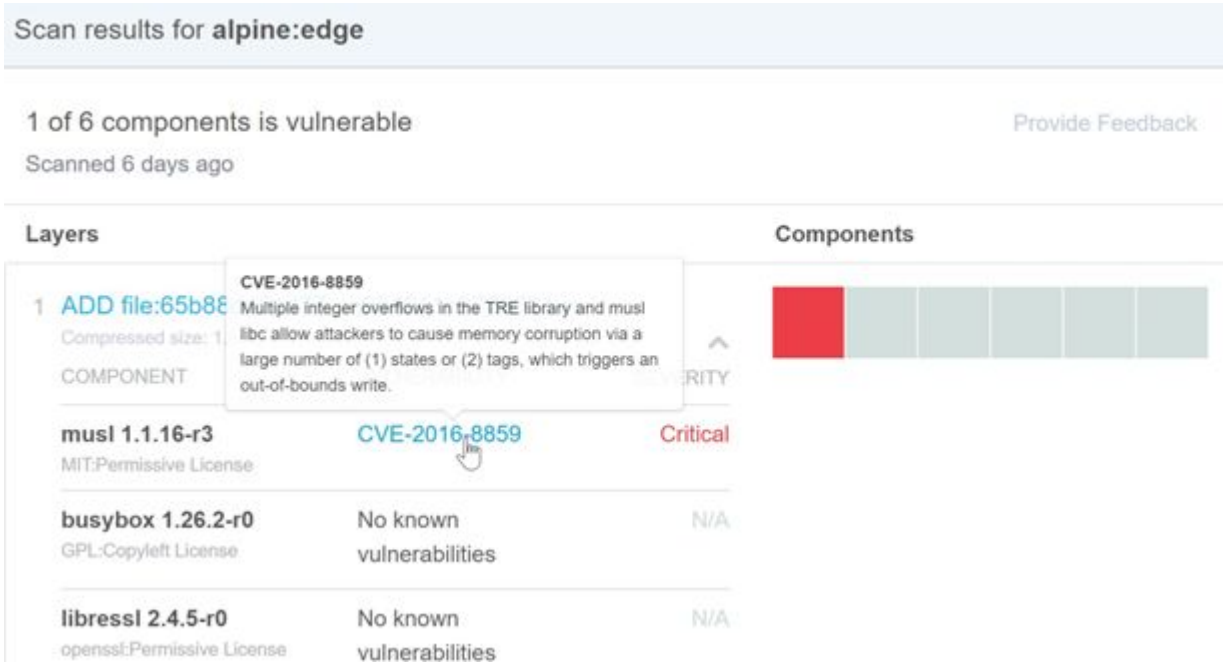
*: Uses CoreOS Clair : https://github.com/coreos/clair

Security Scan

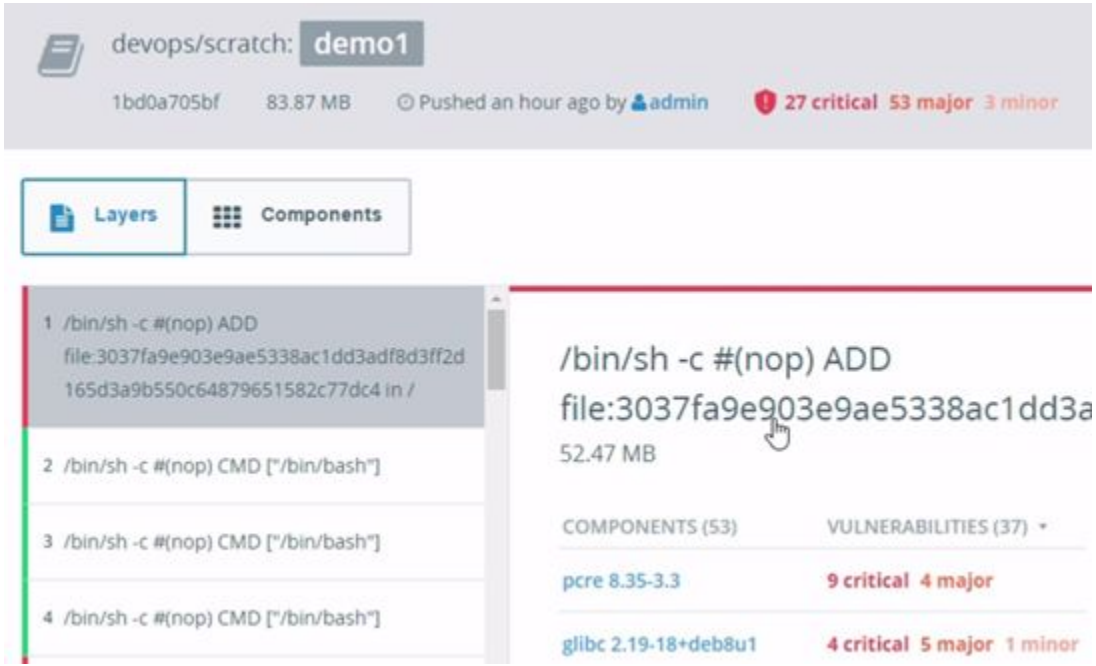


⇒ small image always secure ??

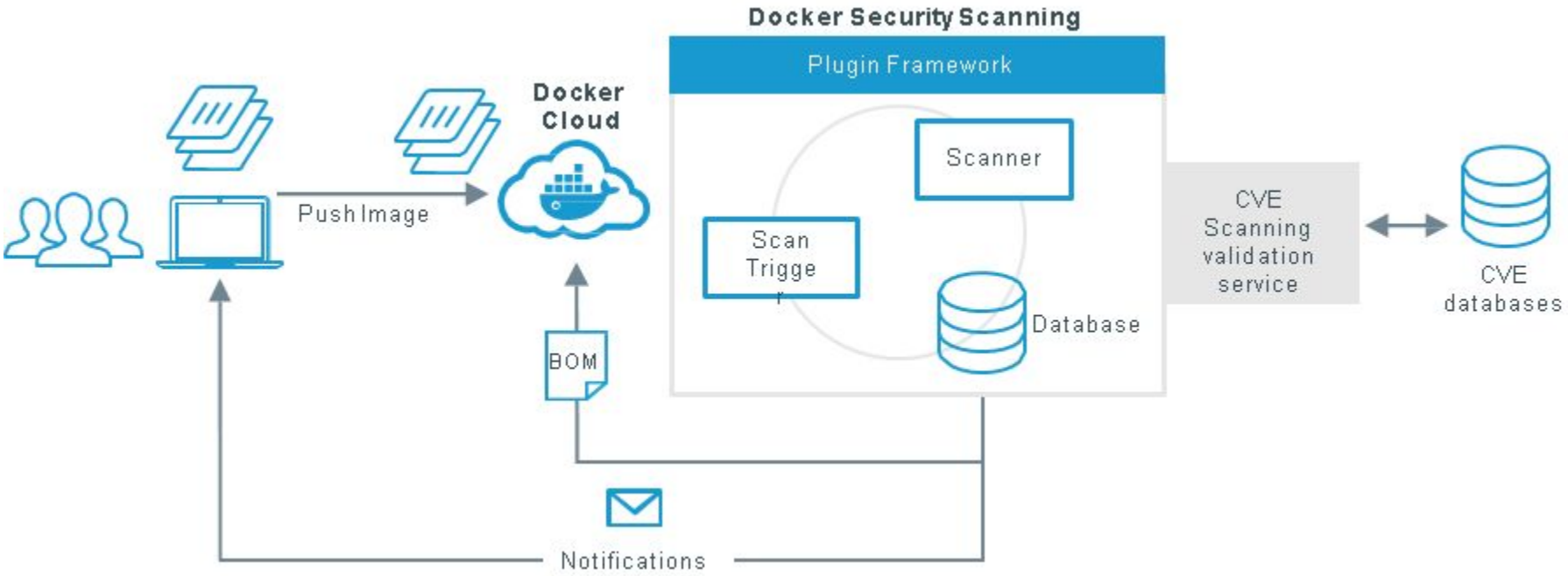
Security Scan



Security Scan



Security Scan



Docker images

Lab 3 : Security Scan

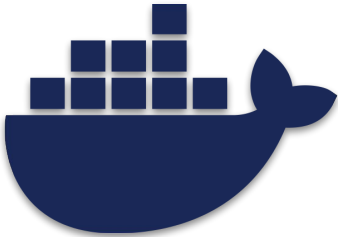
Lab3: Security Scan

$ git clone https://github.com/xinity/security-workshop.git

$ cd scanning

$ vi README.md

Approximate time: 15 minutes



Swarm

Swarm:

Security based on :

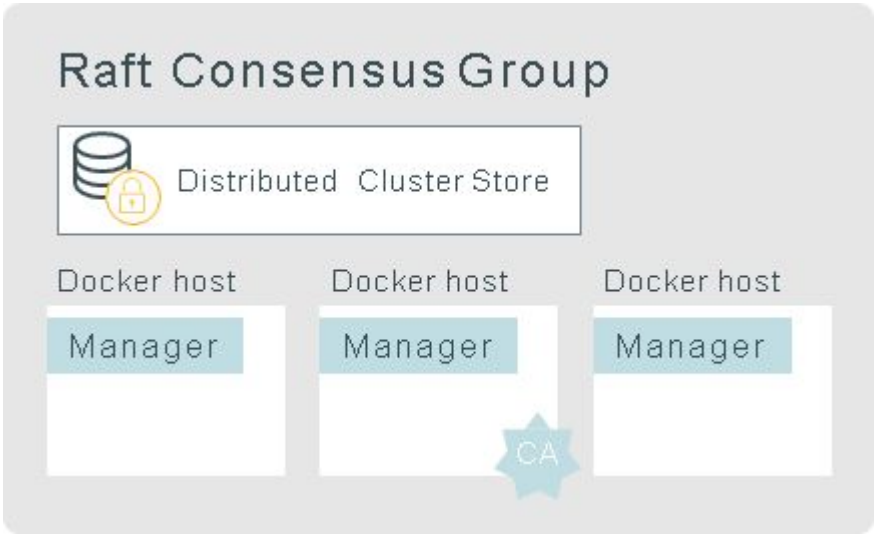
- Raft consensus
- MTLS mechanism
- Secure distributed cluster store

Swarm:

- Raft consensus algorithm
  - agreement on values in a fault tolerant system.
  - mutual exclusion through the leader election process
  - cluster membership management
  - globally consistent object sequencing and CAS (compare-and-swap) primitives

Swarm: Secure ClusterStore

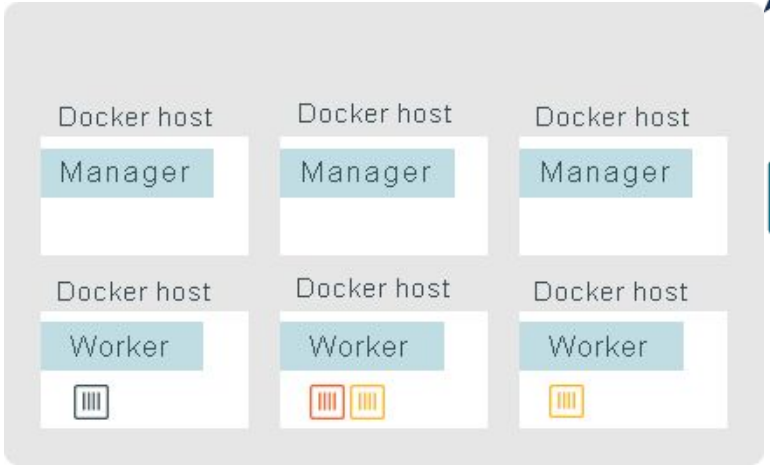
- The cluster store is encrypted
- Anything stored in the cluster store is encrypted (secrets etc.)
- The cluster store is distributed/replicated across all managers



Swarm:

- One or more Managers (control plane)
- One or more Workers (data plane)
  - Run user workloads

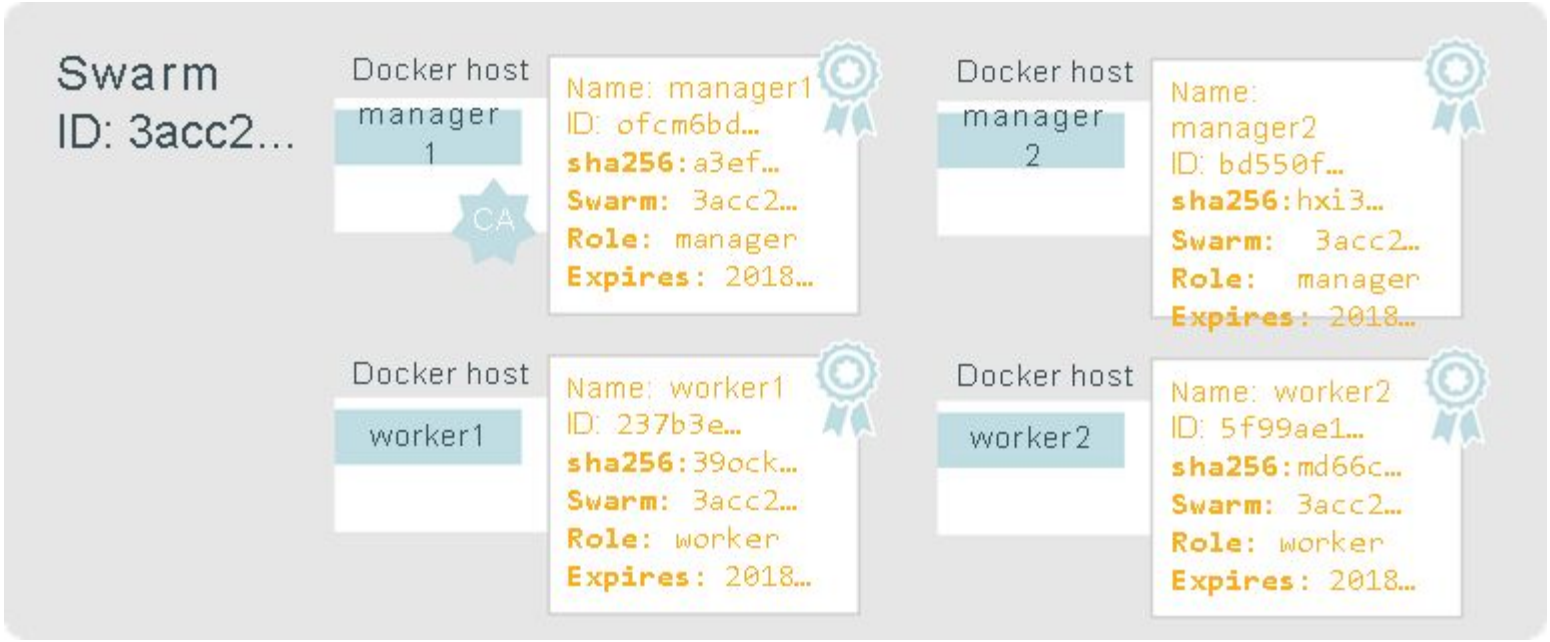
- Strong default security (out-of-the-box)



Swarm:

- Every node gets a Client cert that identifies:
  - The node
  - The Swarm that it's a member of
  - Its role in the Swarm

Swarm:


Swarm
ID: 3acc2...
Docker host
manager 1
CA
Name: manager1
ID: ofcm6bd...
sha256:a3ef...
Swarm: 3acc2...
Role: manager
Expires: 2018...
Docker host
manager 2
Name: manager2
ID: bd550f...
sha256:hxi3...
Swarm: 3acc2...
Role: manager
Expires: 2018...
Docker host
worker1
Name: worker1
ID: 237b3e...
sha256:39ock...
Swarm: 3acc2...
Role: worker
Expires: 2018...
Docker host
worker2
Name: worker2
ID: 5f99ae1...
sha256:md66c...
Swarm: 3acc2...
Role: worker
Expires: 2018...

Swarm:

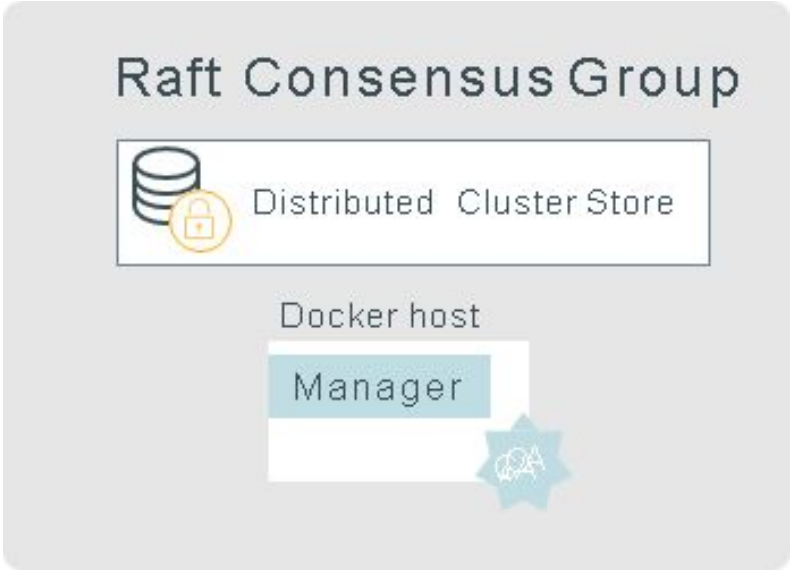
$ docker swarm init

Swarm initialized: current node (ofcm6bdy5qcrlievawsw9wqfp) is now a manager.

To add a worker to this swarm, run the following command:
docker swarm join --token SWMTKN-1-31fxss83n3puc6bd11wm8vxged2ul94fxfbckjdy0rj37agkko-bz14m6jyeakhzvccs7wnbmmof 172.31.45.44:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.

Swarm:

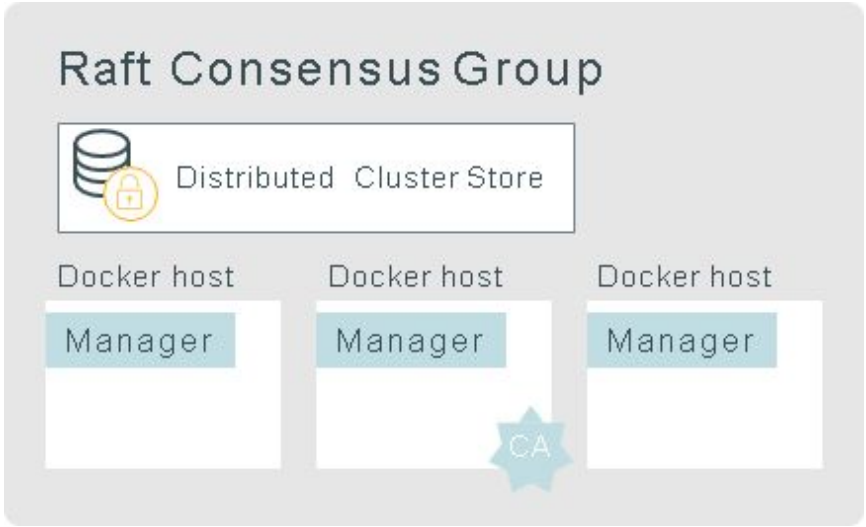

Raft Consensus Group
Distributed Cluster Store
Docker host
Manager

Swarm:

$ docker swarm join-token **manager**

....

$ docker swarm join --token --token SWMTKN-1-31fx-8z0l... \ 172.31.45.44:2377

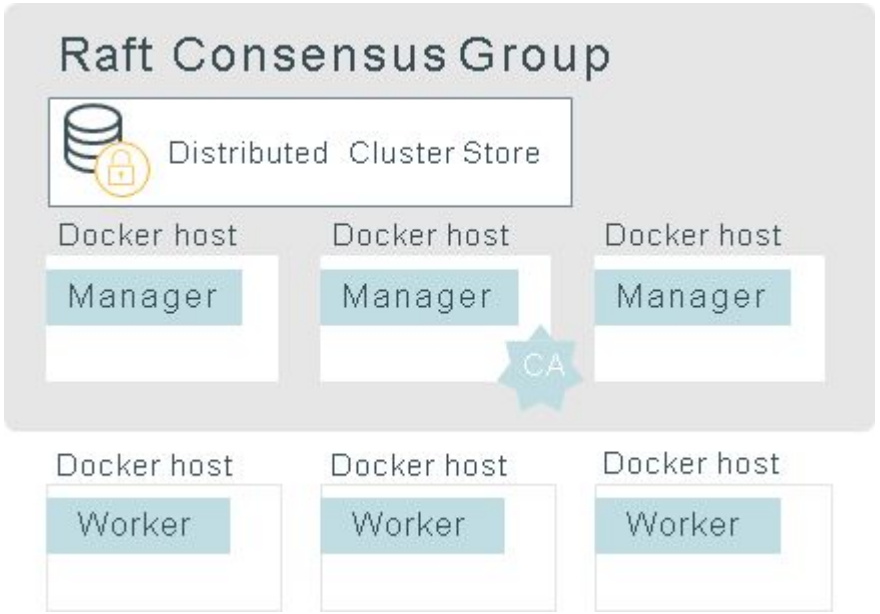


Swarm: Workers

$ docker swarm join-token **worker**

...

$ docker swarm join --token SWMTKN-1-31fx-bz14... \
172.31.45.44:2377



Swarm: !! Warning !!

Only approved nodes should be allowed to join your Swarm!

To join a Swarm as a **manager**, a node must specify the **manager join token**.

⇒ Keep it safe!

To join a Swarm as a **worker**, a node must specify the **worker join token**.

⇒ Keep it safe!

You can rotate join tokens with:

$ docker swarm join-token --rotate worker|manager

Swarm: Certificate Rotation

Automatic *client certificate* rotation:

- defaults to 90 days
- Customizable

Swarm operates a whitelist of valid certificates

Renewal times are randomized to prevent overloading the CA

Swarm: Certificate Rotation

Only client certificates can be rotated*

Use the *--cert-expiry* flag to change the rotation period

The following command will build a Swarm that rotates client certificates every 30 days

$ docker swarm init --cert-expiry 720h0m0s

The following command updates a Swarm to rotate client certificates every 60 days

$ docker swarm update --cert-expiry 1440h

Swarm:

Workers: User workload

Managers: management apps/UI

Don’t ever deploy production apps on a manager !!!

Swarm

Constraints

Swarm: Constraints

Constraints use the following:

- Built-in node attributes
  - node.id | node.hostname | node.role | ...
- Built-in Engine labels
  - engine.labels.operatingsystem | ...
- User-define node labels
  - node.labels.zone | node.labels.pcidss ...

Swarm: Constraints

$ docker service create --name svc1 **--constraint 'node.role == worker'** \

redis:latest

⇒ run redis container only on **workers** nodes

$ docker service create --name svc1 --constraint

'engine.labels.operatingsystem == ubuntu 16.04' \ redis:latest

⇒ run redis container only on **Ubuntu 16.04** hosts

Swarm: Constraints

$ docker node update --label-add zone=prod1 \ node1

$ docker service create --name svc1 \

--constraint ‘node.labels.zone == prod1’ \ redis:latest

⇒ deploys redis on zone: **prod1**, node : **node1**

Swarm: Constraints

$ docker node update --label-add zone=prod1 \ node1

$ docker service create --name svc1 \

--constraint ‘node.labels.zone != prod1’ \ redis:latest

⇒ does **not** deploys redis on zone: **prod1**, node : **node1**

Swarm: Constraints

$ docker node update --label-add

- Simple key/value pairs
- Great way to organize nodes
- Only apply within the Swarm

Swarm

Lab 4 : Building a Secure Swarm

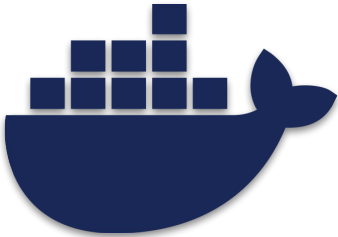
Lab4: Secure Swarm

$ git clone https://github.com/xinity/security-workshop.git

$ cd swarm

$ vi README.md

Approximate time: 20 minutes



Swarm

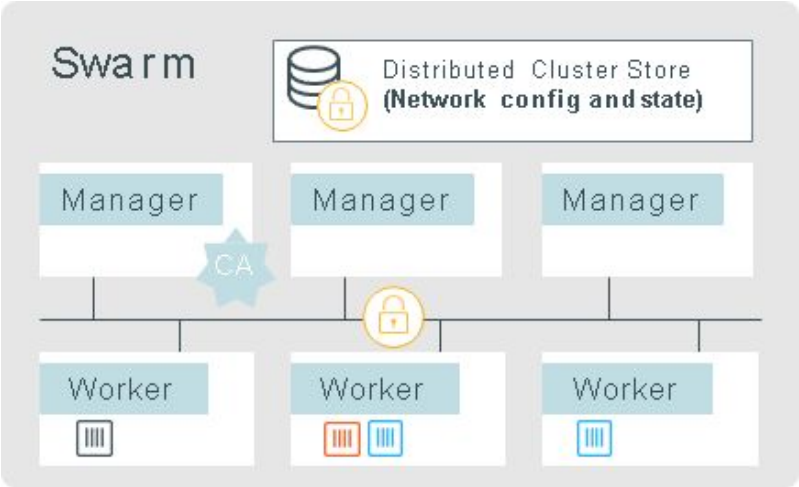
Networking

Swarm: Networking

Swarm: Networking

Every Swarm gets a distributed cluster store

- Encrypted by default
- Stores :
  - network config
  - Network state



⇒ All node-to-node communication is secured by mutual TLS

Swarm: Networking

Control Plane : Encrypted by default

- AES (GCM)
- Keys rotated every 12 hours

Data Plane : Can be easily encrypted ' *--opt encrypted* '

- AES (GCM)
- Keys rotated every 12 hours

⇒ compute overhead and possible latency when encrypted !

Swarm: Networking

$ docker network create -d overlay --opt encrypted my-net

- Control Plane encrypted (by default)
- Data Plane encrypted
- Keys automatically rotated
- Config in distributed secure cluster store

Swarm: Networking

$ docker container run --rm -it **--net=host** \ alpine sh

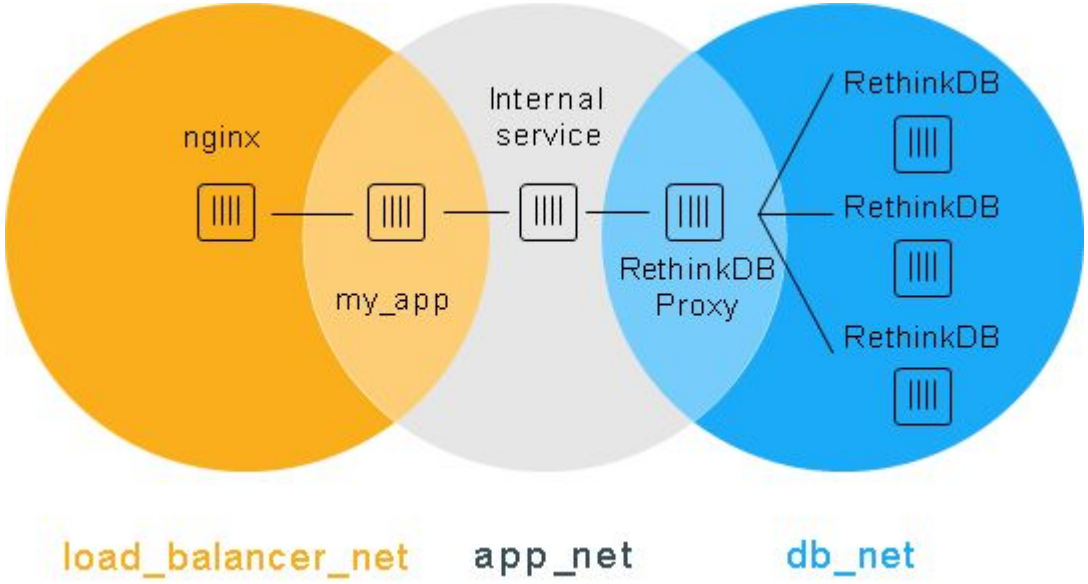
Don't !
--net=host will allow the container to see all networking traffic on the Docker host!

Swarm: Networking

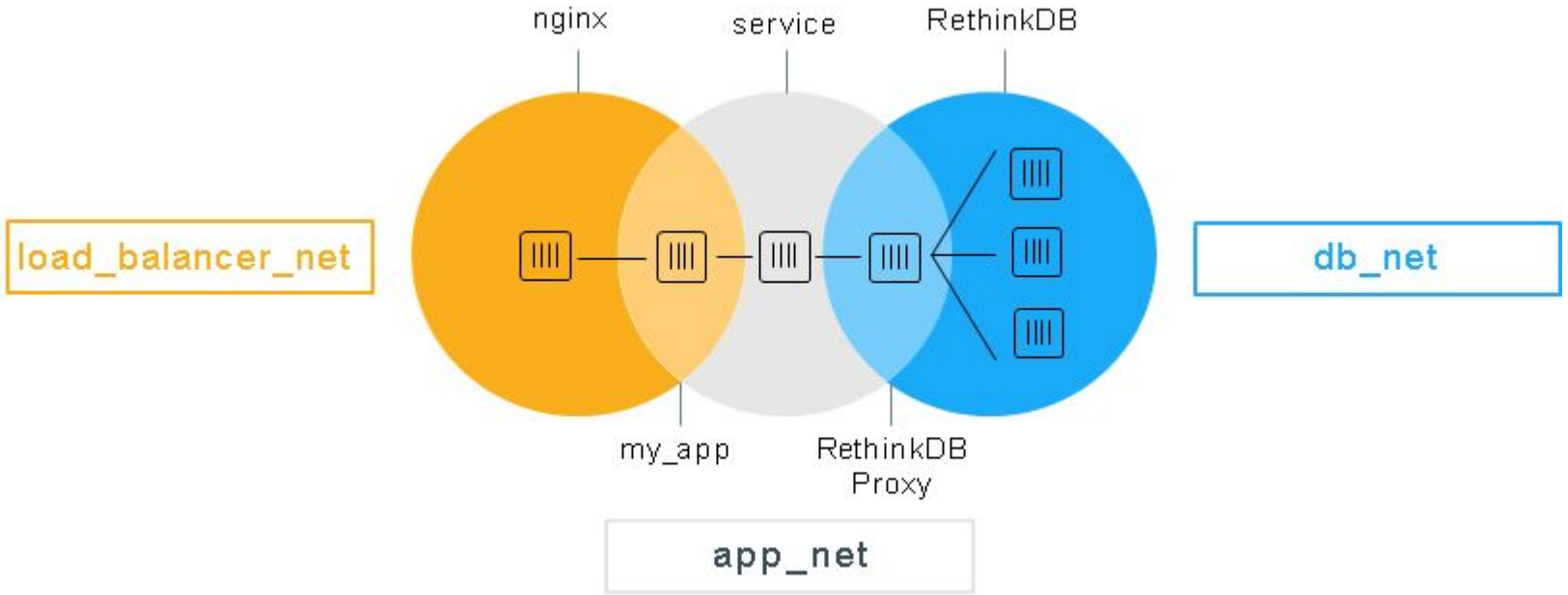
Networking in Swarm is “lazy”

- Newly created networks are only created on nodes that need them
- Nodes that do not need them do not get them (more secure)
- Reduces network chatter (more secure)

Swarm: Networking

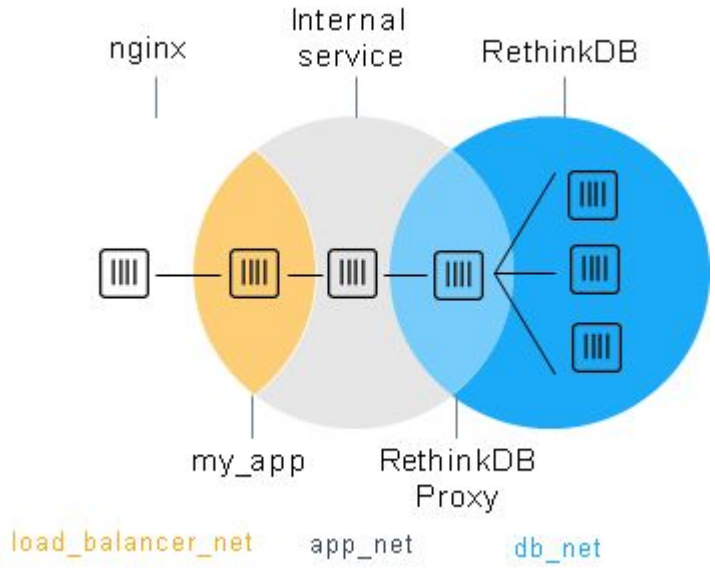


Swarm: Networking



Swarm: Networking

- Micro segmentation
- By default, containers can only talk to other containers on the same network
- Service Discovery is network-scoped
  - Containers cannot automatically discover services and containers on other networks



Swarm

Lab 5 : Secure Networking

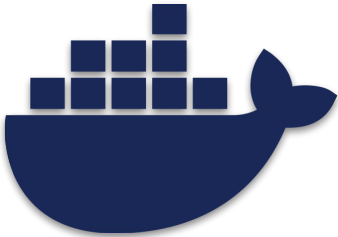
Lab5: Secure Networking

$ git clone https://github.com/xinity/security-workshop.git

$ cd networking

$ vi README.md

Approximate time: 20 minutes



Swarm

Secrets Management

Secrets Management

Standardized interface for :

- developers
- operations teams

Fits most existing methods of accessing secrets

Leverages existing security features of Swarm Mode

Secrets Management

- Compose and services support for secrets
  - Define services, secrets, networks and volumes in a single file
- Secrets available : /run/secrets/{my secret}

⇒ need to make the application “secrets enabled”

Secrets Management

- Encrypted at rest in the cluster store
- Encrypted in-flight over the network
- Only available to authorized apps/services
- Never persisted to disk in containers or on nodes

Secrets Management

Workflow Example:

Development environment:

- Secret: **password** stored in : /run/secrets/app-sec

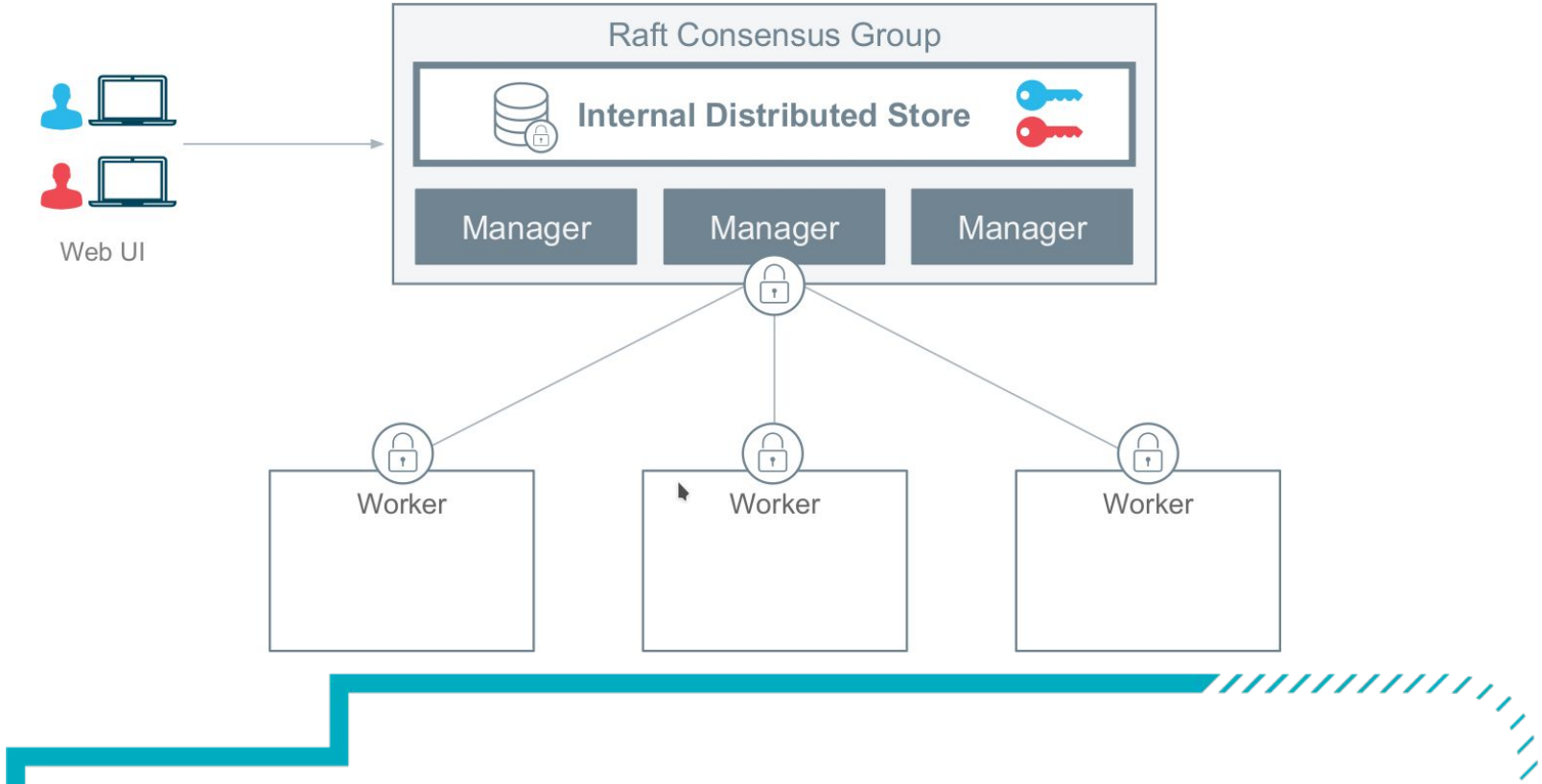
Test environment:

- Secret: **Password123** stored in **:** /run/secrets/app-sec

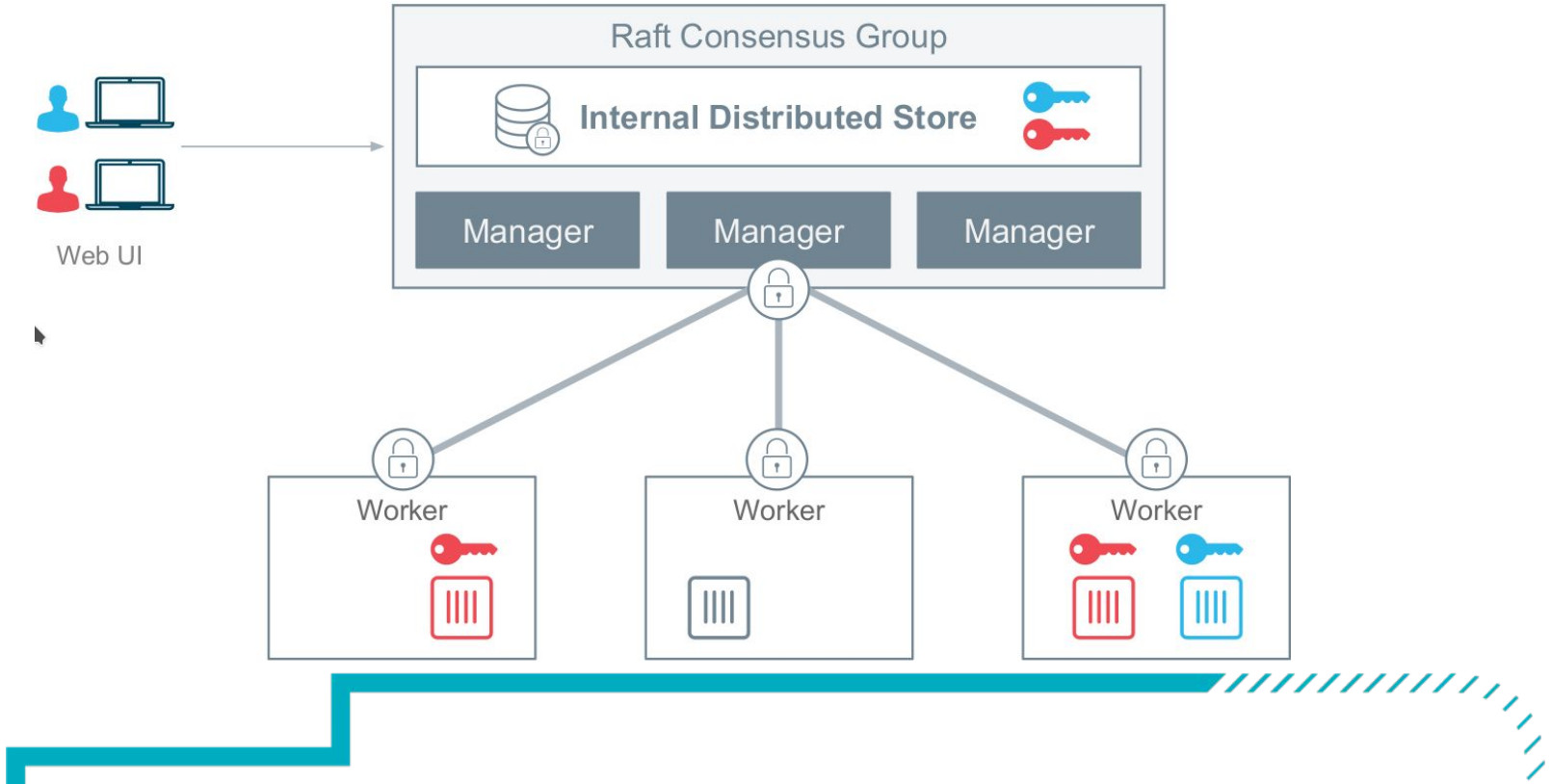
Production environment:

- Secret: **@e~£.#$$ed8kJ** stored in : /run/secrets/app-sec

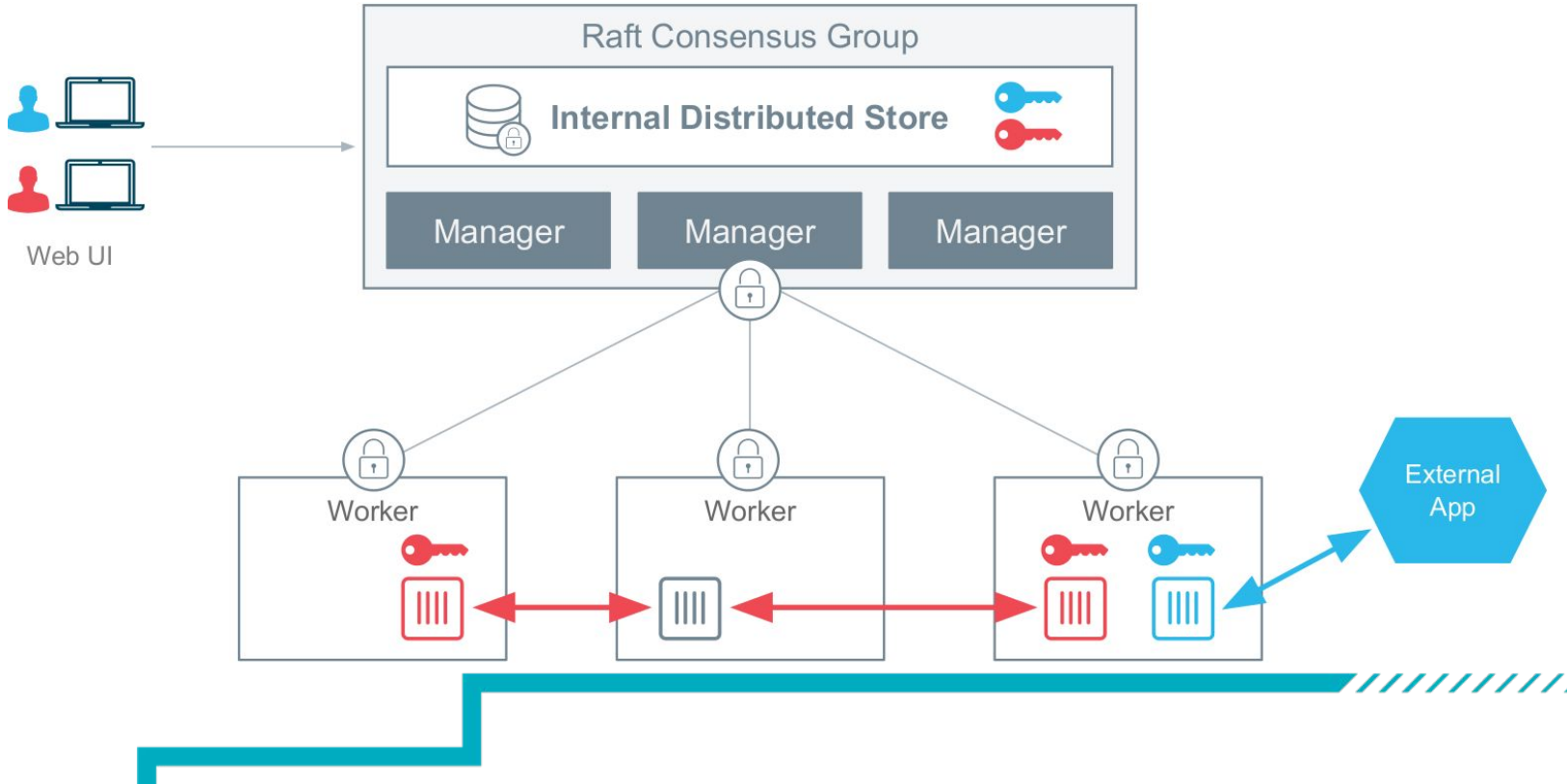
Secrets Management



Secrets Management



Secrets Management



Swarm

Lab 6 : Secrets

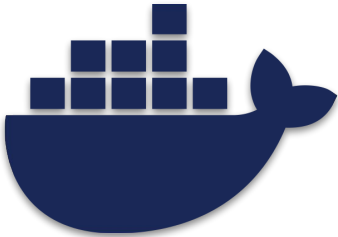
Lab6: Secrets

$ git clone https://github.com/xinity/security-workshop.git

$ cd secrets

$ vi README.md

Approximate time: 20 minutes



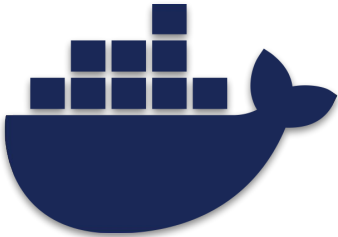
Lab6: Secrets-ddc

$ git clone https://github.com/xinity/security-workshop.git

$ cd secrets-ddc

$ vi README.md

Approximate time: 20 minutes



Advanced

User Management

User Management

Dockerd requires root access

- Uses kernel features like namespaces

Verify root access of dockerd:

$ ps ufaxw | grep dockerd

User Management

By default dockerd listens on /var/run/docker.sock

- Non-networked , local Unix socket
- Own by dockerd group
  - $ ls -l /var/run/docker.sock
  - srw-rw---- 1 root docker 0 Mar 30 09:15 /var/run/docker.sock

User Management

- Non root user should be member of docker group
  - $ sudo usermod -aG docker xinity
  - $ id xinity

User Management

By default containers runs as root /!\

$ docker container run -v /:/srv -it --rm alpine sh

```
# whoami
root

# id
uid=0(root) gid=0(root)

# rm -rf /srv/*
```

User Management

By default

Root inside a container == root outside of a container

- $ docker container run --user 1000:1000 \
  -v /:/srv -it --rm alpine sh
- / $ id
- uid=1000 gid=1000
- $ rm /srv/*
- rm: can't remove '/srv/*': **Permission denied**
- $ ps

User Management

User namespaces:

- Been in the Linux kernel for a while
- Supported in Docker since 1.10

How it works:

- Give a container its own isolated set of UIDs and GIDs
- These isolated UIDs and GIDs inside the container are mapped to non-privileged UIDs and GIDs on the Docker host.

User Management

User Management

User namespaces: example

- $ sudo systemctl stop docker
- $ sudo dockerd --userns-remap=default &
  INFO[0000] User namespaces: ID ranges will be mapped to subuid/subgid...
  .....
- $ docker run -v /:/srv -it --rm alpine sh
- # id
- uid=0(root) gid=0(root) ...
- / # rm /host/bin/sh
- rm: can't remove '/host/bin/sh': **Permission denied**

User Management

$ sudo dockerd --userns-remap=default &

The --userns-remp flag uses mappings defined in:

- /etc/subuid /etc/subgid

```
$ cat /etc/subuid
lxd:100000:65536
root:100000:65536
ubuntu:165536:65536
dockremap:231072:65536
```

```
cat /etc/subgid
lxd:100000:65536
root:100000:65536
ubuntu:165536:65536
dockremap:231072:65536
```

User Management

Mapping to the **default** user namespace uses the dockermap user and group

Mappings contain three fields:

- User or group name
- Starting subordinate UID/GID
- Number of subordinate UIDs/GIDs available

User Management

When you start Docker with the --userns-remap flag the daemon runs within

the confined user namespace.

- As part of the implementation a new Docker environment is created under /var/lib/docker
- The name of this new subdirectory the mapped UID and the mapped GID

```
$ sudo ls -l /var/lib/dockertotal 40
drwx------ 11 231072 231072 4096 Mar 30 11:17 231072.231072
```

This remapped daemon will operate inside of this **231072.231072** environment

- All of you previously pulled images etc will be inaccessible to this
- remapped daemon

User Management

You can verify the namespace that the daemon is running in

with the *docker info* command

It is not recommended to regularly

stop and restart the daemon

in **new user Namespaces**

```
$ docker info
Containers: 1
 Running: 1
 Paused: 0
 Stopped: 0
Images: 1
Server Version: 17.03.1-ce
Storage Driver: aufs
<Snip>
Docker Root Dir:

/var/lib/docker/231072.231072
...
```

• Mainly because you cannot access images etc. in other namespaces (including the global namespace)

User Management

You can verify the namespace that the daemon is running in with the **docker info** command

It is not recommended to regularly stop and restart the daemon in new user namespaces

• Mainly because you cannot access images etc. in other namespaces (including the global namespace)

User Management

```
$ docker info
Containers: 1
 Running: 1
 Paused: 0
 Stopped: 0
Images: 1
Server Version: 17.03.1-ce
Storage Driver: aufs
<Snip>
Docker Root Dir:

/var/lib/docker/231072.231072
...
```

User Management

Lab 7 : UserNS

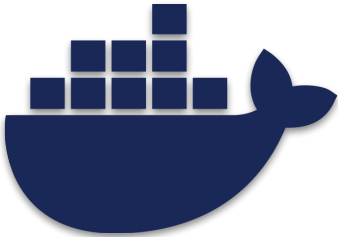
Lab7: User Management

$ git clone https://github.com/xinity/security-workshop.git

$ cd userns

$ vi README.md

Approximate time: 15 minutes



Linux Capabilities

Linux Kernel Capabilities

The Unix world has traditionally divided process into two categories:

- Privileged (root)
- Unprivileged (non-root)

Privileged processes bypass all kernel permission checks

Unprivileged process are subject to all kernel permission checks

Linux Kernel Capabilities

This **all or nothing** approach often led to processes running as root when they really only needed a small subset of the privileges assigned to root processes.

Modern Linux kernels slice root privileges into smaller chunks called **Capabilities**.

It is now possible to assign **some** root privileges to a process without assigning them all.

Linux Kernel Capabilities

A container running a web server that only needs to bind to a port below 1024 does not need to run as root! **Should not run as root!**

It might be enough to drop all capabilities for that container except CAP_NET_BIND_SERVICE.

If an intruder is able to escalate to root within the web server container they will be limited to binding to low numbered privileged ports.
They won't be able to bypass file ownership checks, kill processes, lock memory, create special files, modify routing tables, set promiscuous mode, setuid, load kernel modules, chroot, renice processes, ptrace, change the clock etc...

Net result = reduced attack surface!

Linux Kernel Capabilities

Docker operates a **whitelist** approach to implementing capabilities.

- If a capability isn't on the whitelist it is dropped.
- The list shows the current capabilities whitelist for the default profile.
- https://github.com/docker/docker/blob/master/oci/defaults_linux.go#L62-L77

```
s.Process.Capabilities = []string{
        "CAP_CHOWN",
        "CAP_DAC_OVERRIDE",
        "CAP_FSETID",
        "CAP_FOWNER",
        "CAP_MKNOD",
        "CAP_NET_RAW",
        "CAP_SETGID",
        "CAP_SETUID",
        "CAP_SETFCAP",
        "CAP_SETPCAP",
        "CAP_NET_BIND_SERVICE",
        "CAP_SYS_CHROOT",
        "CAP_KILL",
        "CAP_AUDIT_WRITE",
}
```

Linux Kernel Capabilities

You can use the **--cap-add** and **--cap-drop** flags to add an remove capabilities from a container.

To drop the **CAP_NET_BIND_SERVICE** capability form a container:

```
$ docker container run --rm -it --cap-drop NET_BIND_SERVICE alpine sh
```

Linux Kernel Capabilities

To drop all except the CAP_NET_BIND_SERVICE:

```
$ docker container run --rm -it \
  --cap-drop ALL --cap-add NET_BIND_SERVICE \
  alpine sh
```

To add the CAP_CHOWN capability to a container:

```
$ docker container run --rm -it \
  --cap-add CHOWN \
  alpine sh
```

Linux Kernel Capabilities

Docker cannot currently add capabilities to non-root users

- All of the examples shown in the slides have been adding and removing capabilities from containers running as root

Privilege escalation is difficult without file-related capabilities

- File-related capabilities are stored in a file's extended attributes
- Extended attributes are stripped out when Docker Images are built

Linux Kernel Capabilities

Lab 9 : Capabilities

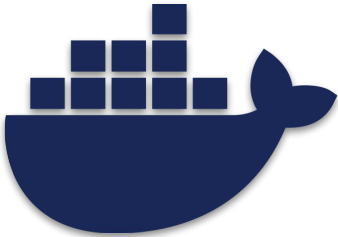
Lab9: Capabilities

$ git clone https://github.com/xinity/security-workshop.git

$ cd capabilities

$ vi README.md

Approximate time: 40 minutes



AppArmor

Mandatory Access Control

AppArmor

AppArmor is a Linux kernel security module.

You define profiles that control access to specific resources such as files and the network.

Sane defaults:

- Preventing writing to /proc/{num}, /proc/sys, /sys
- Preventing mount

You can apply these profiles to applications and containers.

Use the **docker info** command to see if AppArmor is installed and available

AppArmor

Docker creates and loads a default AppArmor profile for

containers called **docker-default**

- Sensible defaults
- Based on https://github.com/docker/docker/blob/master/profiles/apparmor/template.go

A profile for the Docker daemon exists but is not installed and used by default

AppArmor

```
# deny write for all files directly in /proc
deny @{PROC}/* w,
# deny write to files not in /proc/<number>/** or
/proc/sys/**
deny @{PROC}/{[^1-9],[^1-9][^0-9],[^1-9s][^0-
9y][^0-9s],[^1-9][^0-9][^0-9][^0-9]*}/** w,
# deny /proc/sys except /proc/sys/k* (effectively
/proc/sys/kernel)
deny @{PROC}/sys/[^k]** w,
# deny everything except shm* in /proc/sys/kernel/
deny @{PROC}/sys/kernel/{?,??,[^s][^h][^m]**} w,
deny @{PROC}/sysrq-trigger rwklx,
deny @{PROC}/mem rwklx,
deny @{PROC}/kmem rwklx,
deny @{PROC}/kcore rwklx,
deny mount,
deny /sys/[^f]*/** wklx,
deny /sys/f[^s]*/** wklx,
deny /sys/fs/[^c]*/** wklx,
deny /sys/fs/c[^g]*/** wklx,
deny /sys/fs/cg[^r]*/** wklx,
deny /sys/firmware/** rwklx,
deny /sys/kernel/security/** rwklx,
```

AppArmor

You can override the default container profile (docker-default) with the **--security-opt** flag

```
$ docker container run --rm -it /
  --security-opt apparmor=custom-profile hello-world
```

AppArmor

Use the **aa-status** command to see the status of AppArmor profiles

```
$ aa-status
apparmor module is loaded.
14 profiles are loaded.
14 profiles are in enforce mode.
   /sbin/dhclient
   /usr/bin/lxc-start
   ...
   docker-default
   <Snip>
0 profiles are in complain mode.

4 processes are in enforce mode.
   /sbin/dhclient (924)
   docker-default (26965)
   docker-default (27528)
   docker-default (27908)
```

AppArmor

Lab 8 : AppArmor

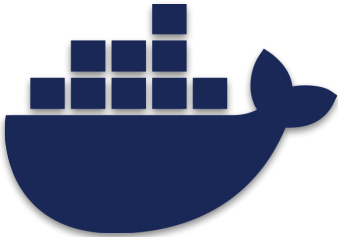
Lab8: AppArmor

$ git clone https://github.com/xinity/security-workshop.git

$ cd appamor

$ vi README.md

Approximate time: 40 minutes



SecComp

Syscalls Filtering

SecComp

seccomp is a Linux kernel module that acts like a firewall for syscalls

- In the mainline Linux kernel since 2005
- Supported in Docker since Docker 1.10

Using seccomp-bpf (Berkley Packet Filters) is an extension that makes seccomp more flexible and granular

- You can create policies that allow granular control of which syscalls are allowed and which are not

Docker allows you to associate seccomp policies with containers

- The aim is to control (limit) a containers access to the Docker host's kernel

SecComp

seccomp needs to be enabled in the Docker host's kernel **as well as** in the Docker Engine.

To check for seccomp in the kernel

```
$ cat /boot/config-`uname -r` | grep CONFIG_SECCOMP=
CONFIG_SECCOMP=y
```

To check for seccomp in Docker:

```
$ docker info | grep seccomp
seccomp
```

SecComp

Docker automatically applies the default seccomp policy to new

containers

The aim of the default policy is to provide a sensible out-of-the-box policy

You should consider the default policy as moderately protective while providing wide application compatibility

The default policy disables over 40 syscalls (Linux has over 300 syscalls)

The default policy is available here:

https://github.com/docker/docker/blob/master/profiles/seccomp/default.json

SecComp

You can use the --security-opt flag to force containers to run within a custom seccomp policy

```
$ docker run --rm -it \
  --security-opt seccomp=/path/to/seccomp/profile.json \
  hello-world
```

Docker seccomp profiles operate using a whitelist approach that specifies allowed syscalls. Only syscalls on the whitelist are permitted

SecComp

You can run containers without a seccomp policy applied

- This is call running a container unconfined

```
$ docker run --rm -it \
  --security-opt seccomp=unconfined \
  hello-world
```

⇒ It is not recommended to run containers unconfined!

Seccomp

Lab 10 : Seccomp

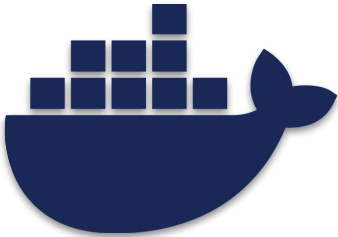
Lab10: SecComp

$ git clone https://github.com/xinity/security-workshop.git

$ cd seccomp

$ vi README.md

Approximate time: 40 minutes



Opensource Tools

Docker Bench Security

Open-source tool for running automated tests

- Inspired by the CIS Docker 1.13 benchmark
- Regularly updated

Checks Docker host

Runs against containers on same host

• Checks for AppArmor, read-only volumes, etc...

• https://dockerbench.com

Docker Bench Security

```
# ------------------------------------------------------------------------------
# Docker Bench for Security v1.3.0
#
# Docker, Inc. (c) 2015-
#
# Checks for dozens of common best-practices around deploying Docker containers in production.
# Inspired by the CIS Docker 1.13 Benchmark.
# ------------------------------------------------------------------------------

Initializing Thu Jan 26 08:58:33 UTC 2017


[INFO] 1 - Host Configuration
[WARN] 1.1  - Create a separate partition for containers
[INFO] 1.2  - Harden the container host
[PASS] 1.3  - Keep Docker up to date
[INFO]      * Using 1.13.0 which is current as of 2017-01-18
[INFO]      * Check with your operating system vendor for support and security maintenance for Docker
[INFO] 1.4  - Only allow trusted users to control Docker daemon
[INFO]      * docker:x:998:ubuntu
[WARN] 1.5  - Audit docker daemon - /usr/bin/docker
[WARN] 1.6  - Audit Docker files and directories - /var/lib/docker
[WARN] 1.7  - Audit Docker files and directories - /etc/docker
[WARN] 1.8  - Audit Docker files and directories - docker.service
[WARN] 1.9  - Audit Docker files and directories - docker.socket
[WARN] 1.10 - Audit Docker files and directories - /etc/default/docker
[INFO] 1.11 - Audit Docker files and directories - /etc/docker/daemon.json
[INFO]      * File not found
[WARN] 1.12 - Audit Docker files and directories - /usr/bin/docker-containerd
[WARN] 1.13 - Audit Docker files and directories - /usr/bin/docker-runc


[INFO] 2 - Docker Daemon Configuration
[WARN] 2.1  - Restrict network traffic between containers
[WARN] 2.2  - Set the logging level
[PASS] 2.3  - Allow Docker to make changes to iptables
[PASS] 2.4  - Do not use insecure registries
[WARN] 2.5  - Do not use the aufs storage driver
[WARN] 2.6  - Configure TLS authentication for Docker daemon
[WARN]      * Docker daemon currently listening on TCP with TLS, but no verification
[INFO] 2.7  - Set default ulimit as appropriate
[INFO]      * Default ulimit doesn't appear to be set
[WARN] 2.8  - Enable user namespace support
[PASS] 2.9  - Confirm default cgroup usage
[PASS] 2.10 - Do not change base device size until needed
[WARN] 2.11 - Use authorization plugin
[WARN] 2.12 - Configure centralized and remote logging
[WARN] 2.13 - Disable operations on legacy registry (v1)
[WARN] 2.14 - Enable live restore
[PASS] 2.15 - Do not enable swarm mode, if not needed
[PASS] 2.16 - Control the number of manager nodes in a swarm (Swarm mode not enabled)
[PASS] 2.17 - Bind swarm services to a specific host interface
[WARN] 2.18 - Disable Userland Proxy
[PASS] 2.19 - Encrypt data exchanged between containers on different nodes on the overlay network
[PASS] 2.20 - Apply a daemon-wide custom seccomp profile, if needed
[PASS] 2.21 - Avoid experimental features in production
```

Docker Bench Security

Run as a container

Runs with a lot of **privileges** !!
As It needs to run tests
Against the Docker host

```
$ docker run -it --net host --pid host \
    --cap-add audit_control \
    -e
DOCKER_CONTENT_TRUST=$DOCKER_CONTENT_TRUST
\
    -v /var/lib:/var/lib\
    -v
/var/run/docker.sock:/var/run/docker.sock \
    -v /usr/lib/systemd:/usr/lib/systemd \
    -v /etc:/etc --label
docker_bench_security \
    docker/docker-bench-security
```

Secure Supply Chain

Final Lab: let's put them all together