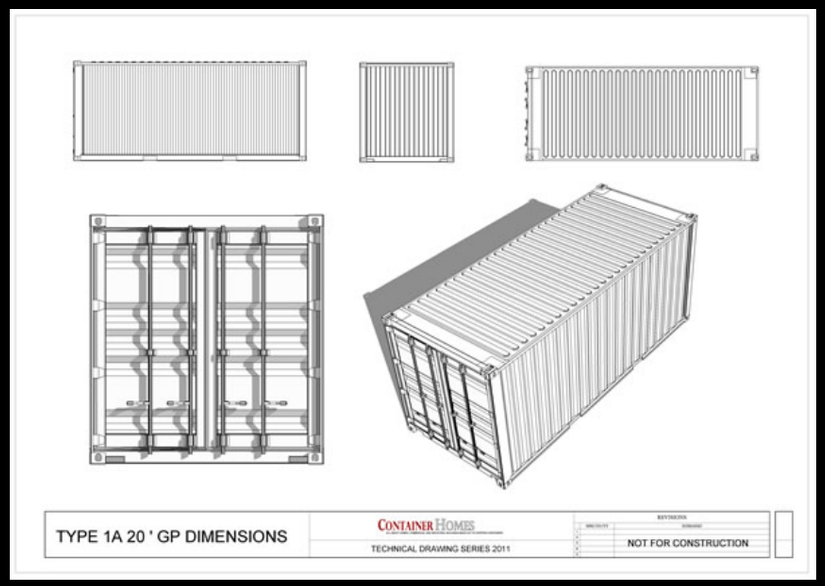
docker pull random-image

docker run —privileged —network host random-image

profit!!11

Fucking Mag^H^H^HContainers

How do they work?



Agenda

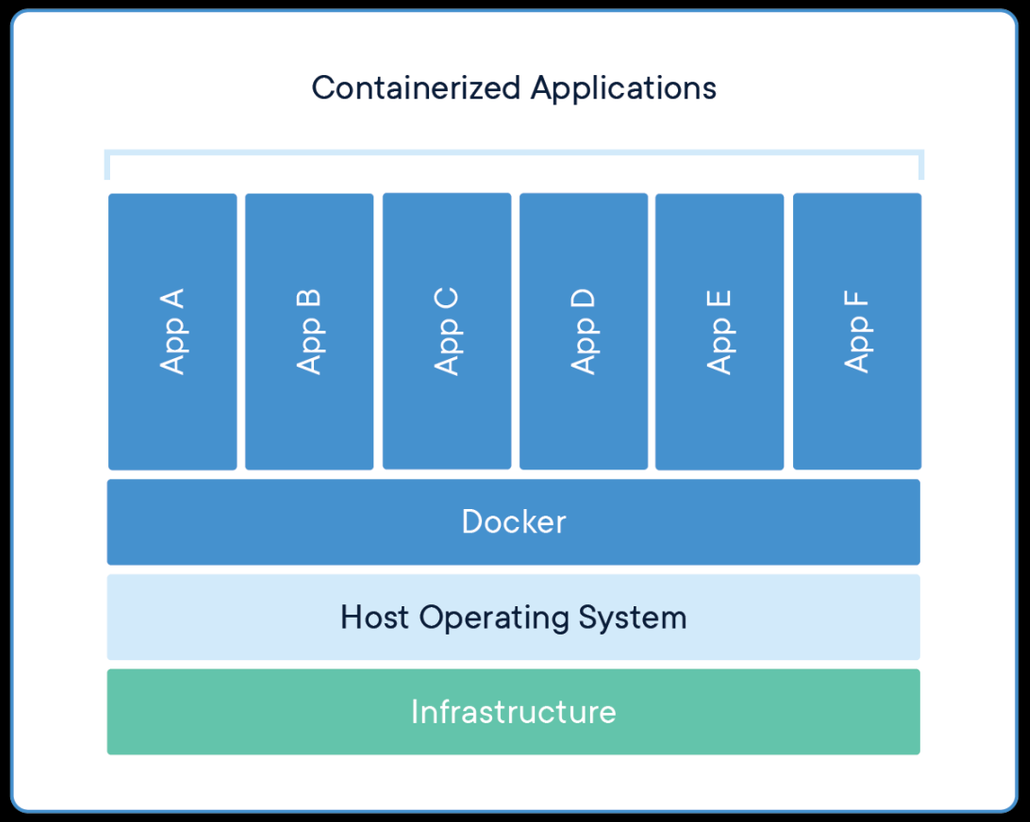
- Me - The Who
- Container overview - The Why
- Docker Engine internals - The How
- Container image internals - The How cont.
- Container security - The Cybers
- Q & A

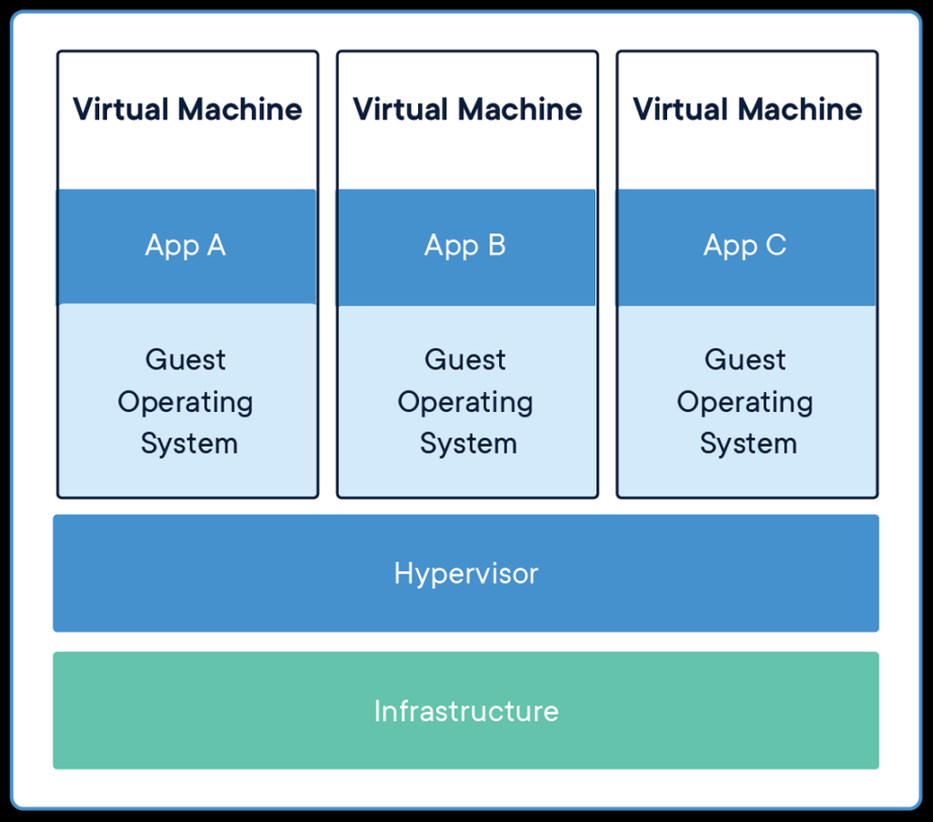
Who am I?

- Andreas Krebs aka ‘wintamute’
- Interested in way too many things
- Started playing with Sinclair ZX Spectrum in the mid 80ies
- First PC 1990
- Online since 1992
- Running my own business since 1999
  doing DevOps, virtualisation, automation
- Member of ccc, c-base, Geraffel and some other groups…
  - wintamute@wintamute.org
  - @wintamute

Container Overview

- Container, Docker, Kubernetes, etc.
- History: BSD Jails, Solaris Zones, Linux containers (lxc)
- Bundles everything in a container image - makes deployment easy, same image for testing, staging, production -> reproducible (yay!)
- APIs to make usage very easy, drives automation





Containers

- Share Host Hardware
- Share Host OS Kernel
- Virtualises OS
- Low resource overhead
- Startup time in milliseconds

VMs

- Share Host Hardware
- Each VM uses own OS
- Virtualises HW
- High resource overhead
- Startup time in minutes

Image Source: https://www.docker.com/

Docker Engine internals

- implemented in Go
- Linux Kernel features used:
  - Namespaces (pid, net, ipc, mnt, uts) - main feature for isolation
  - Iptables (with default network bridge)
  - Control groups (cgroups) - used to limit resources, manage processes, checkpointing
  - Union file systems (file system layering)
  - capabilities (default: drops all except needed ones, can be changed)
  - AppArmor/SELinux
- Components: server/daemon, REST API, docker CLI

REST Interface

Docker Engine

libcontainer | graph | libnetwork | Plugins

Containerd & runC

Control Groups
Job Objects

Namespaces
Process Namespace
Network Namespace
IPC, mnt, uts etc.

Layer Capabilities
AUFS, BRTFS, VFS, ZFS etc.

Other OS Functionality

Operating System

docker

Image Source: https://www.docker.com/

Networking

- Works out of the box (Single-Host/Multi-Host)
- Provides different options (None, Host, Bridge, overlay)
- Can be replaced by 3rd party network drivers
- Containers can be attached to multiple networks

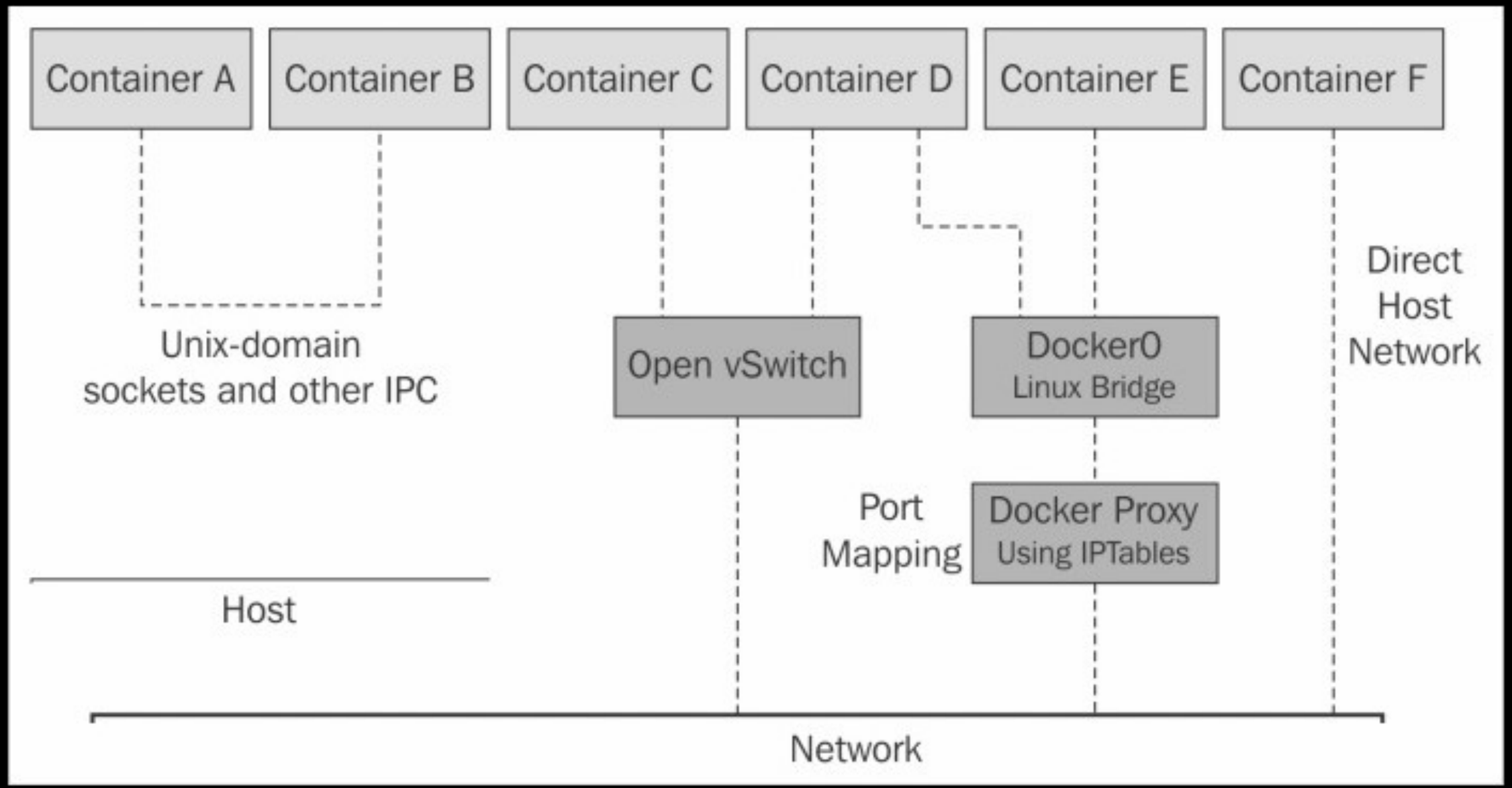


Image Source: the Internet

Networking contd.

- Default network created during Docker installation (creates bridge Docker0)
- Expose ports during image and/or container creation (available ports for incoming connections)
- Publish ports and create veth interface during container creation (ip table rules)
- Attach containers to multiple networks

Container Image Internals

- Docker Image Specification
- Format being standardised in OCI (Open Container Initiative)
- Series of layers, read-only
- Each layer represents one command from the Dockerfile

Union Filesystem

- Files and directories of separate systems
- Transparently overlaid
- Forming a single coherent file system
- Copy-on-write
- Different options AuFs, Btrfs, OverlayFS, DeviceMapper

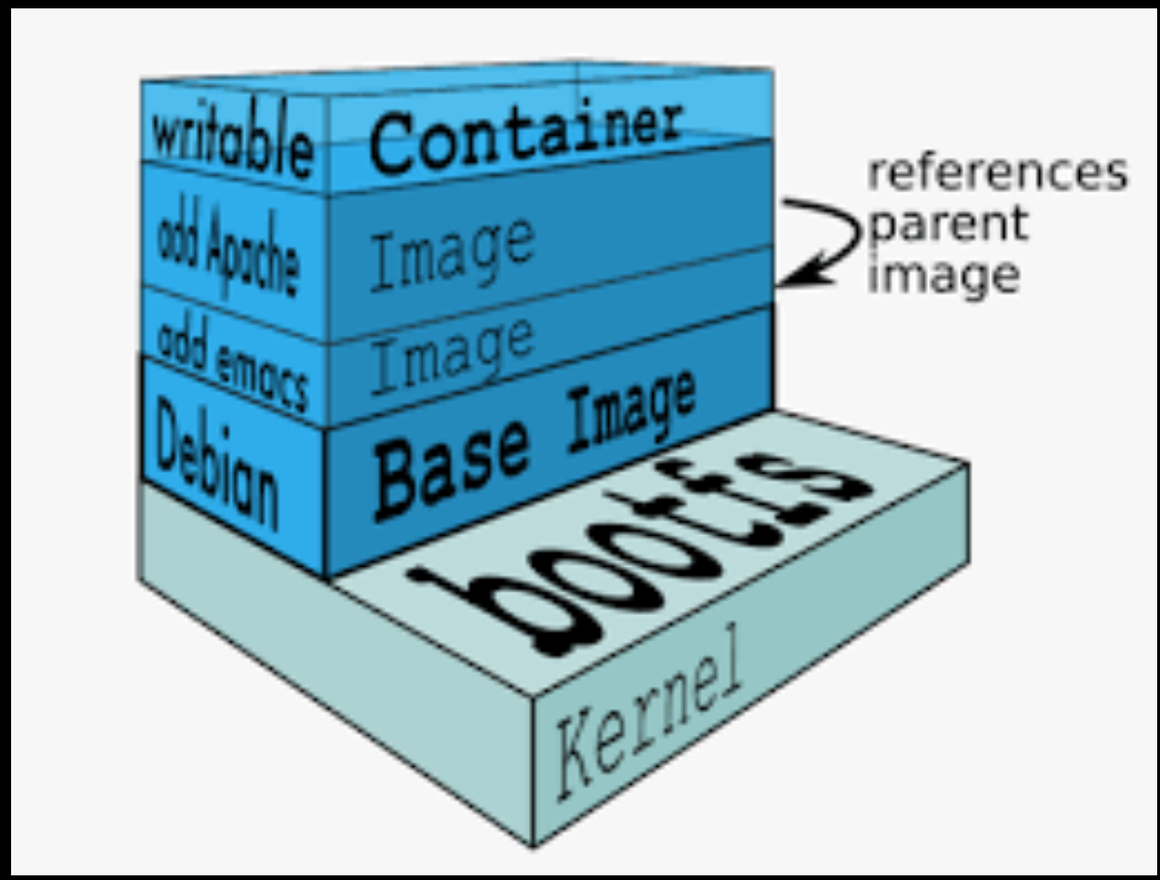


Image Source: https://www.docker.com/

Image from single binary to full OS - Dockerfile

```
FROM scratch
COPY hello /
CMD ["/hello"]
```

```
FROM buildpack-deps:sid-curl
RUN apt-get update && apt-get install -y --no-install-recommends \
       bzip2 \
       unzip \
       xz-utils \
   && rm -rf /var/lib/apt/lists/*
RUN echo 'deb http://deb.debian.org/debian experimental main' > /etc/apt/sources.list.d/
experimental.list
# Default to UTF-8 file.encoding
ENV LANG C.UTF-8
RUN { \
       echo '#!/bin/sh'; \
       echo 'set -e'; \
       echo; \
       echo 'dirname "$(dirname "$(readlink -f "$(which javac || which java)")")"'; \
   } > /usr/local/bin/docker-java-home \
   && chmod +x /usr/local/bin/docker-java-home
ENV JAVA_HOME /usr/lib/jvm/java-9-openjdk-amd64
ENV JAVA_VERSION 9~b149
ENV JAVA_DEBIAN_VERSION 9~b149-1
RUN set -x \
   && apt-get update \
   && apt-get install -y \
       openjdk-9-jre-headless="$JAVA_DEBIAN_VERSION" \
   && rm -rf /var/lib/apt/lists/* \
   && [ "$JAVA_HOME" = "$(docker-java-home)" ]
```

Container Security

- Run only one process per container
- Slim down container image as much as possible
- Use a Container Linux as Host
- Grant only necessary capabilities
- Avoid running process as root by using high ports when possible
- Remap root uid (userns-remap) if process has to run as root

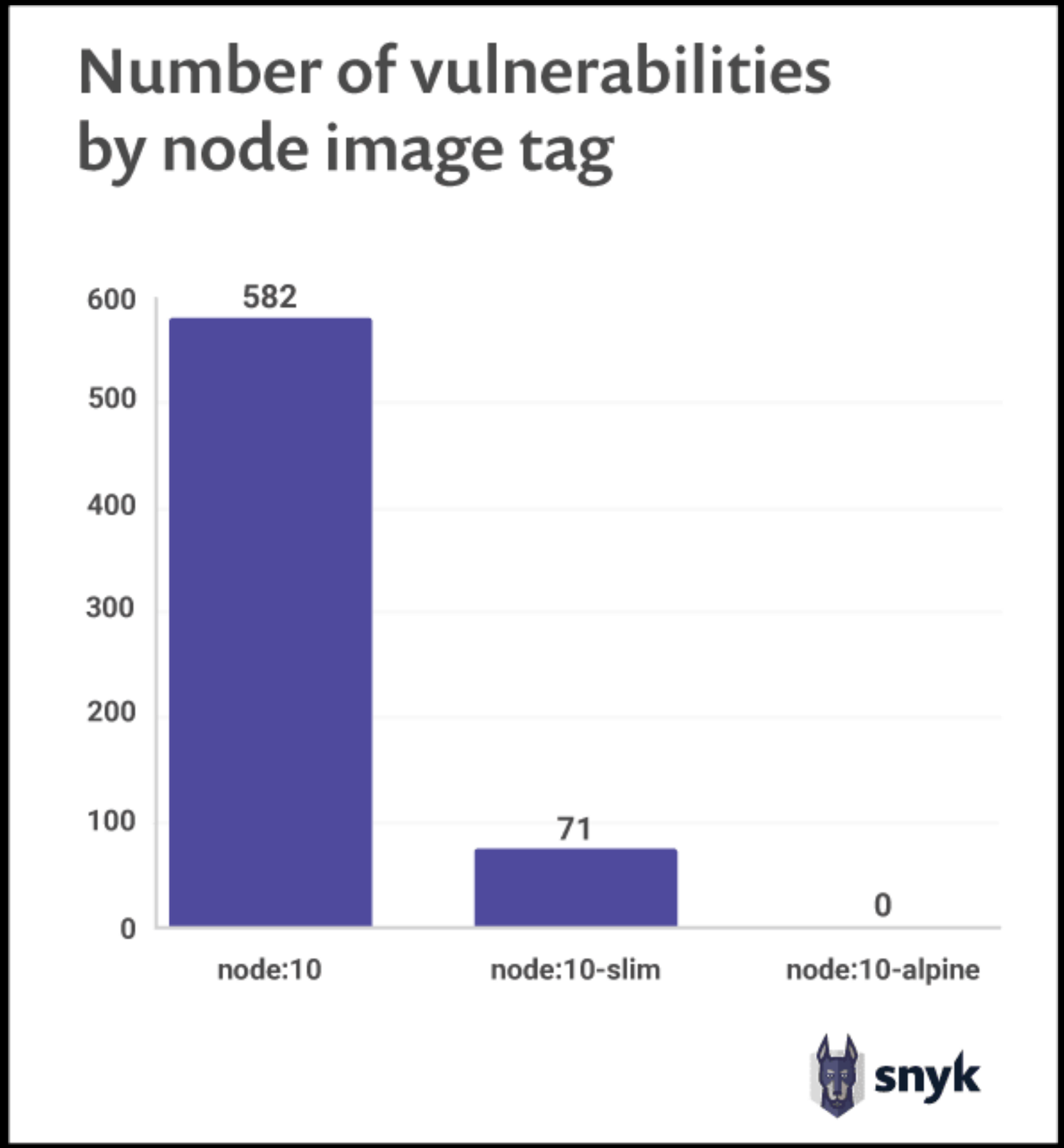


Image source: https://snyk.io

Container Security contd.

- Mount rootfs readonly, write only to mounted volumes
- Use separate networks to isolate services
- Expose only necessary ports
- Secure docker daemon
- Be careful when mounting host directories/sockets
- Use signed images (Docker Content Trust, notary server)

Static analysis

- Clair Container Image Security Analyzer
  - Index Images (based information from Dockerfile)
  - Correlate image features with vulnerability DBs
    - Debian/Ubuntu/Redhat/Oracle/Alpine/NIST

Forensics

- Containers complicate forensics (depends on environment) due to being ephemeral
- Containers can be paused (via cgroups freezer) for memory examination
- Depending on the options used to start the container, host system needs to be examined too.

More info:

- https://github.com/moby
- https://docs.docker.com
- https://www.opencontainers.org
- https://docs.docker.com/engine/security/security/
- https://github.com/coreos/clair
- https://github.com/docker/docker-bench-security
- https://success.docker.com/article/security-best-practices-17-06
- https://static.ernw.de/whitepaper/ERNW_Whitepaper64_IncidentForensicDocker_signed.pdf

More info contd:

- https://snyk.io/blog/top-ten-most-popular-docker-images-each-contain-at-least-30-vulnerabilities/
- https://firecracker-microvm.github.io
- https://katacontainers.io

- Linux kernel namespaces and cgroups http://www.haifux.org/lectures/299/netLec7.pdf
- http://www.netdevconf.org/1.1/proceedings/slides/rosen-namespaces-cgroups-lxc.pdf
- https://www.toptal.com/linux/separation-anxiety-isolating-your-system-with-linux-namespaces

Time for your questions

Thank you