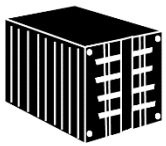
Secure ContainErisation

SLICE



Achim Eisele

Agenda

- Motivation
- Terminology
- Basic Architecture
- Docker Example App
- Threats
- Docker Security
  - Container
  - Container Engine
  - Host
  - Network
  - Secrets
- Conclusion

Motivation

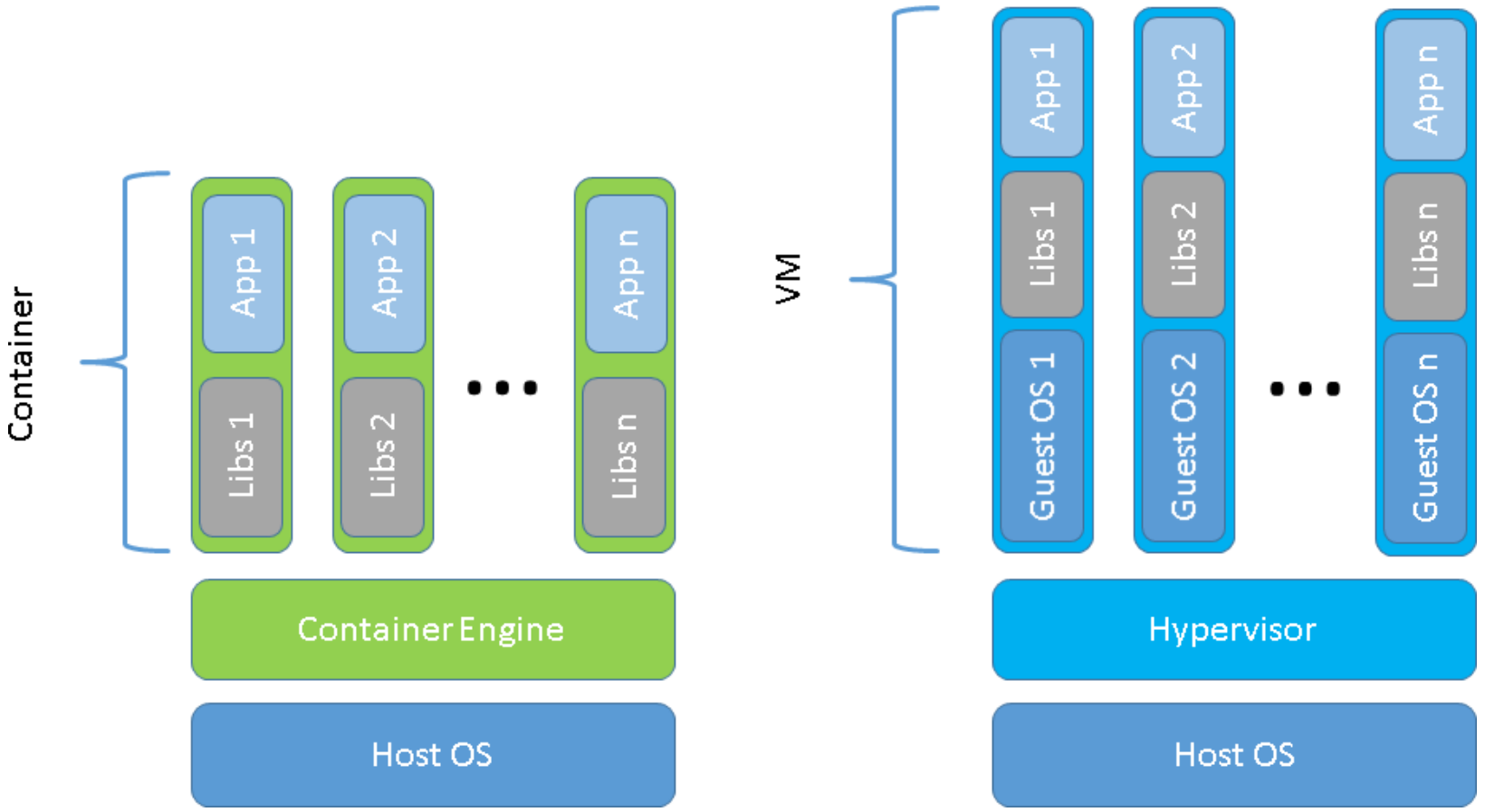
- Application Containerisation found its way from playground to production
- Easy and reliable provisioning of applications in different environments
  - application, libraries and (default) configuration are deployed as a whole
  - allow a fast and highly automated deployment routine
- 1&1 operates several environments
  - e.g. CaaS - Container-as-a-Service (Hosting Product)
  - First applications are deployed onto those platforms and more “traditional” ones to be shifted onto the new platforms in the future
- Document and share security best practices

Terminology

- **Docker** is a software technology providing operating-system-level virtualisation
  - Based on Kernel namespaces (2002) and cgroups (2008)
  - LXC, Solaris Zones, FreeBSD Jails existed prior to Docker, but Docker drives the containerisation hype by ease of use.
- **Kubernetes** (**OpenShift** is built around) offers Container Orchestration
- **OpenStack** – Infrastructure-as-a-Service platform

- An **image** is a file resp. a package, which contains everything to run an application, such as libraries, default configuration and code.
- A **container** is a runtime instance of image. It is isolated from the underlying host environment and other containers on the same host.
- Images are generally stored at a central place called **registry**.
- A **pod** is a group of one ore more containers with shared storage, network and configuration.

Basic Architecture



Docker Example WebApp

:$ cat Dockerfile

```
# base image
FROM ubuntu:16.04

# add apache
RUN apt-get update && apt-get install -y apache2

# add basic web page
COPY index.html /var/www/html/index.html

# do some configuration
ENV APACHE_RUN_USER www-data
ENV APACHE_RUN_GROUP www-data
ENV APACHE_LOG_DIR /var/log/apache2

# listen on port 80
EXPOSE 80

# run command on instantiation
CMD ["/usr/sbin/apache2", "-D", "FOREGROUND"]
```

:$ docker build –t test/mywebpage .
:$ docker run test/mywebpage
:$ curl http://192.168.122.2/

```
<html><head><title>Hallo World!</title></head><body /></html>
```

Threats

- Poisoned images
  - Images might be tampered
  - Outdated images might contain vulnerabilites.
- Container breakouts
  - An attacker with access to a container might break out and control the host
- Denial-of-service attacks
  - Container overconsuming resources
- Kernel exploits
  - One shared kernel among all containers on same host
- Compromising secrets
  - Secrets built into an image are not secret

Recommendations

- Down the stack
  - Container
  - Engine
  - Host

- Depending on strived level of security
  - Must Have (M)
  - Should Have (S)
  - Can Have (S)

Docker Security – Container I/II

- Base images must either be provided or created from trusted sources! (M)
- Regularly inspect images against new, previously unknown vulnerabilities! (M)
- Regularly update your images and exchange appropriate containers (M)
- Images shall only be loaded from Official Repositories! (S)
  - Disclaimer applies.
- Enable Docker Content Trust! (S)
- Drop root privileges as quickly as possible! (M)
- Only use and create small (base) images! (S)
- Reduce the number of layers to a minimum! (C)
- Aim at one process per container only! (S)

Docker Security – Container II/II

- Treat your images as immutable and do not store files inside a container (S)
- Ensure SSH is not run within containers! (M)
- Reduce the set of capabilities per container to the needed minimum! (C)
- Remove setuid and setgid flags off binaries unless needed! (C)
- Reduce the set of allowed syscalls! (C)
- Authenticate and authorise your clients! (M)
- Write logfiles to a central and tamper-proof log store under consideration of the GDPR! (S)

Docker Security – Engine

- Only trusted users shall be allowed to control your Docker daemon! (M)
- Docker's REST API shall only be accessible from trusted networks for authenticated and authorised users! (M)
- Make use of cgroups and define resource limits on container level! (S)
- Use appropriate measures to segregate containers with different data classification! (S)

Docker Security – Host

- Use a minimal operating system installation! (M)
- Do not use a container host for other services! (M)
- Patch the container host regularly! (M)
- Make sure the procfs and sysfs filesystems are mounted read-only into the container! (S)
- Introduce AppArmor or SELinux! (C)
- Apply grsecurity kernel patches! (C)
  - Not for free anymore ☹

Container Security – Network + Secrets

- Create and apply a firewall or ACL concept! (M)
- Ensure transport encryption over untrusted network segments! (M)
- Loadbalancer or reverse proxy shall be the only containers exposing a port to the outside world! (S)

- Do not store sensitive data in an image! (M)
  - Inject during instantiation
- Manage your secrets in a secure manner! (M)

```
[INFO] 2 - Docker daemon configuration
[WARN] 2.1  - Ensure network traffic is restricted between containers on the default bridge
[PASS] 2.2  - Ensure the logging level is set to 'info'
[PASS] 2.3  - Ensure Docker is allowed to make changes to iptables
[PASS] 2.4  - Ensure insecure registries are not used
[PASS] 2.5  - Ensure aufs storage driver is not used
```

Conclusion

- Many best practises do not differ from regular Linux hardening
- Head for defense in depth
- Follow the least privileges principle
- Only use trusted images
- It is not the question either VM or Docker – combine them
- Define the responsibilities for security – DevOps
- Done right, there's an additional layer of security

References

- Aaron Grattafiori, Understanding and Hardening Linux Containers, 2016
- Center for Internet Security, CIS Docker Community Edition Benchmark, 2017
- Docker Inc., Docker Security, 2017
- The Linux Foundation, Security Best Practices for Kubernetes Deployment, 2016
- https://github.com/docker/docker-bench-security