



Pentesting DevOps Environments

Matthias Luft, mluft@ernw.de, [@uchi_mata](https://twitter.com/uchi_mata)

ERNW

- Vendor-independent
- Established 2001
- 70 employees, 45 FTE consultants
- Continuous growth in revenue/profits
 - No venture/equity capital, no external financial obligations of any kind
- Customers predominantly large/very large enterprises
 - Industry, telecommunications, finance



whoami

- CEO of ERNW GmbH
- Technical background in hypervisor security
- From pentester to researcher to consultant to team lead
- Dealing with Virtualization Security since 2008



Agenda

- DevOps?
- Attack Surface
- Attacks & Countermeasures

DevOps?

- Developers & Operations? ... & Security?
 - SecDevOps?
 - DevSecOps?
 - DevOpsSec?
- Agile?
- CI/CD?
- Docker? Kubernetes? Marathon? CoreOS? ...
- Infrastructure-as-Code?
-



DevOps

Sounds Familiar?

- Increase flexibility while reducing costs
- Faster application deployments
- Compete with public cloud offerings



DevOps

“DevOps is the philosophy of unifying Development and Operations at the **culture**, **system**, **practice**, and **tool** levels, to achieve accelerated and more **frequent delivery** of value to the customer, by improving quality in order to increase velocity.”

Rob England, 2014



DevOps

- Culture & Practice
- Technology & Tools

Communication & Collaboration

- Embrace cross-functional roles/teams
- No silos of knowledge, language, goals

- Tooling:
 - Repositories of Code & Knowledge
 - Planning and Project Management
 - Analytics
 - Communication Tools



Difference to Agile Methods?

- Agile Methods focus on software development.
- DevOps focusses on software deployment.

- Both share many approaches, ideas and tools!
- Thoughts on Agile Development & Security

DevOps vs. Continuous Delivery

“Continuous Delivery (CD) is a software engineering approach in which teams produce software in **short cycles**, ensuring that the software can be **reliably released** at any time. It aims at **building, testing, and releasing** software **faster and more frequently**.”

DOI: [10.1109/MS.2015.27](https://doi.org/10.1109/MS.2015.27)



Continuous Deployment

... is often confused with Continuous Delivery and “means that every change goes through the pipeline and **automatically** gets put into production, resulting in **many production deployments every day**.”

<https://martinfowler.com/bliki/ContinuousDelivery.html>



Bringing it all together

- Agile Development produces software.
- Continuous Delivery is a paradigm for development that each sprint (or even more granular tasks) must result in deployable software.
- Continuous Deployment is the automated deployment of software produced by Continuous Delivery to production.
- DevOps is the approach to complement Agile Development with deployment aspects and provide the technology required to deploy fast and often.





ERNW
providing security.



DevOps Technology

From Culture to Technology

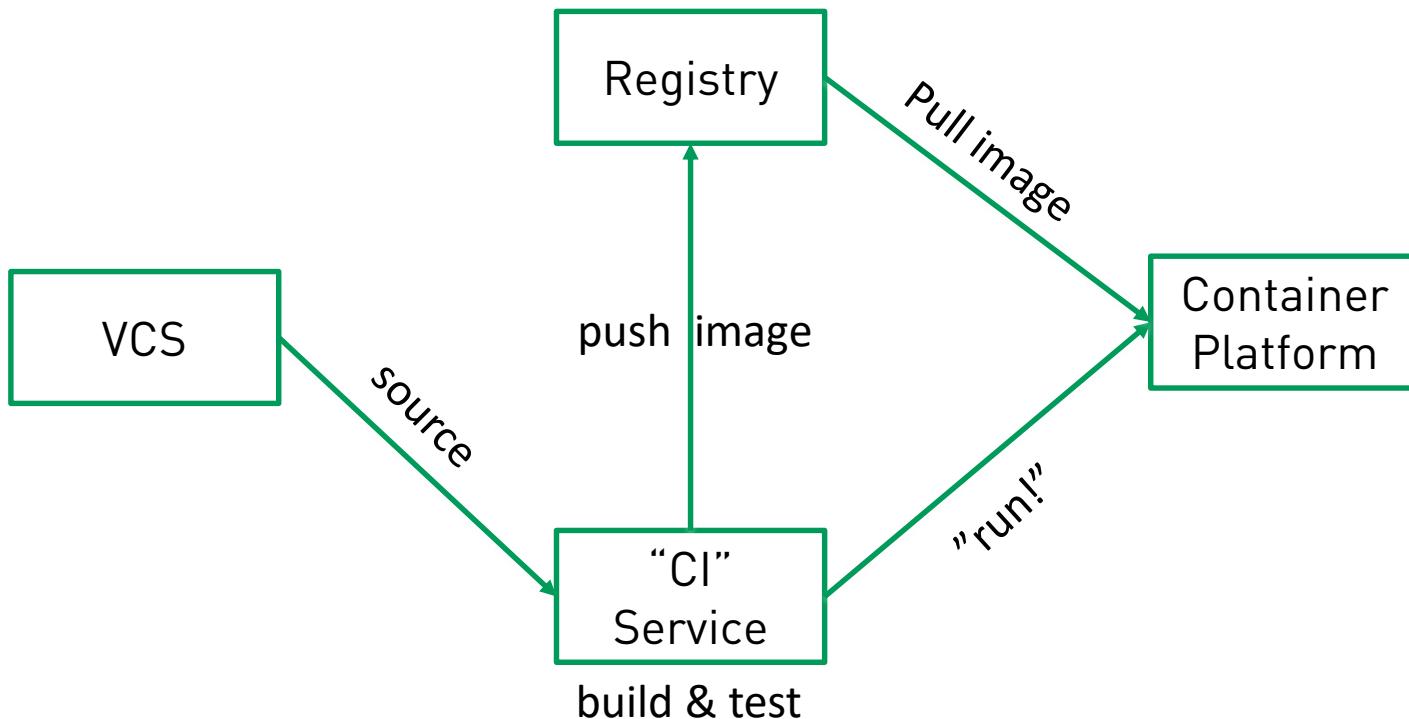
- We've seen the culture and approaches to continuously develop software that is deployable/does not hinder deployments.
- How to make deployments
 - ... faster?
 - ... reproducible?
 - ... automated?



Core DevOps Technologies

- VCS
- Build Pipeline
- Container Orchestration

Sample Build Pipeline



Why Container Orchestration?

- Containers provide an independent runtime environment
- For example, a Docker container runs (or is at least supposed to) the very same way on any Docker host
 - Ever tried that with a jar/war/egg file? ;-)
- => Every developer can have an environment identical to production on their own computer.

Why Container Orchestration?

- Container Orchestration solutions provide great platform features, for example:
 - High availability
 - APIs for automated deployment
 - SDN capabilities
 - Load balancing & Auto Scaling
 - Metrics
 - Logging
 - Secret Management
 - ...
- => Various features that each application was implementing on its own in the “old” world.

Docker?

- Docker?
- Docker Engine?
- Docker Swarm?
- Docker Machine?
- Docker Compose?
- CS Docker Engine? Docker CE/EE?
- Docker Cloud?
- Docker Registry?
- Docker Hub?
- ...
- ...
- Kubernetes, Rancher, Consul, Vault, LambdaCD, Zookeeper, Mesos ...?



Docker & Container



- Docker = Docker Engine
 - At least for most people as of 2017.
- What is a container?
 - For now: Group of processes managed/isolated via Linux kernel features
 - Core features:
 - cgroups
 - namespaces
 - Layered filesystem

cgroups



- v1 vs. v2:
 - v1 appeared first in 2008 2.6.24
 - v2 is re-written and appeared first in 2016 4.5
- <https://en.wikipedia.org/wiki/Cgroups>:
 - Resource limiting – groups can be set to not exceed a configured memory limit, which also includes the file system cache
 - Prioritization – some groups may get a larger share of CPU utilization or disk I/O throughput
 - Accounting – measures a group's resource usage, which may be used, for example, for billing purposes
 - Control – freezing groups of processes, their checkpointing and restarting
- [cgroups]

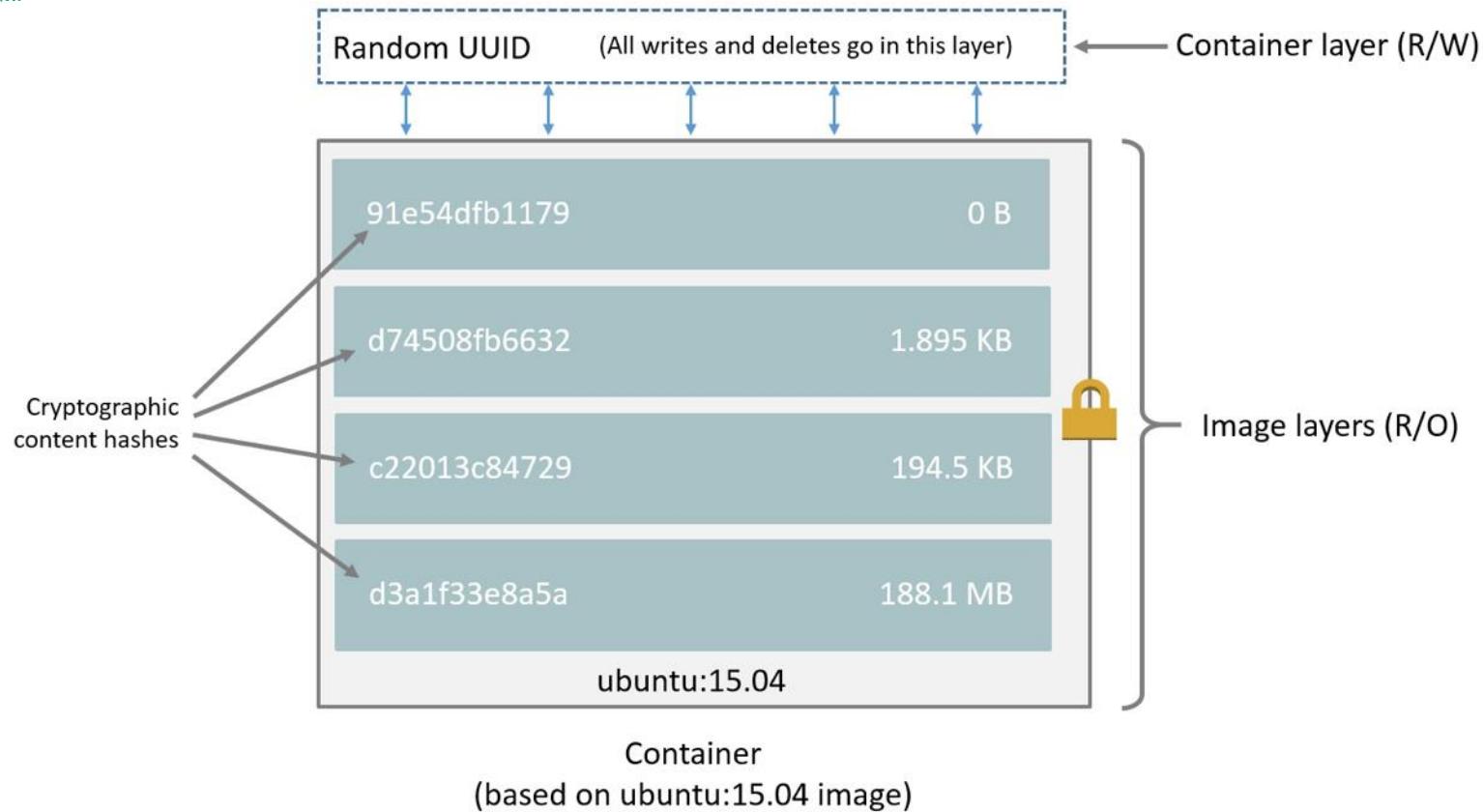
namespaces

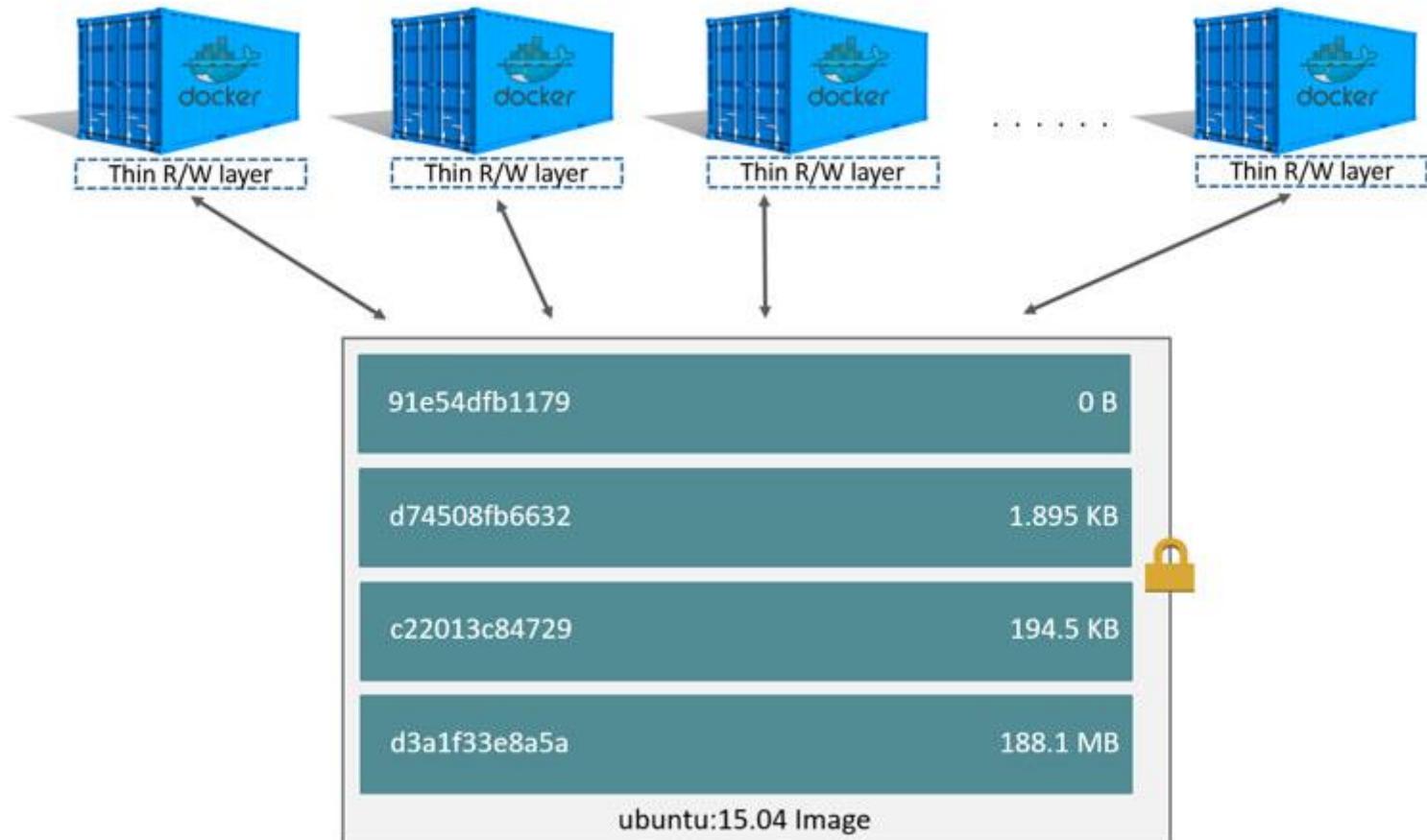


- Kernel feature for the isolation/virtualization of resources:
 - Mount
 - UTS
 - IPC
 - PID
 - Network
 - User
- [namespaces]

Layered
Filesystem

Source





Layered
Filesystem

Source



```
root@70b800f7f822: / 
<1> root@70b800f7f8...
root@docker:~#
root@docker:~# docker run -it --rm ubuntu bash
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
ae79f2514705: Pull complete
5ad56d5fc149: Pull complete
170e558760e8: Pull complete
395460e233f5: Pull complete
6f01dc62e444: Pull complete
Digest: sha256:506e2d5852de1d7c90d538c5332bd3cc33b9cbd26f6ca653875899c505c82687
Status: Downloaded newer image for ubuntu:latest
root@70b800f7f822:/#
root@70b800f7f822:/# id
uid=0(root) gid=0(root) groups=0(root)
root@70b800f7f822:/#
root@70b800f7f822:/# ls
bin  dev  home  lib64  mnt  proc  run  srv  tmp  var
boot  etc  lib  media  opt  root  sbin  sys  usr
root@70b800f7f822:/# |
```

```
ubuntu@docker: ~
```

```
<1> root@70b800f7f8...[*] <2> ubuntu@docker: ...
```

```
ubuntu@docker:~$ curl icanhazip.com  
34.229.145.56  
ubuntu@docker:~$ |
```

```
<1> root@70b800f7f8... <2> ubuntu@docker: ...
```

```
root@70b800f7f822:/# curl icanhazip.com  
34.229.145.56  
root@70b800f7f822:/#
```

<1> root@70b800f7f822... <2> ubuntu@docker: ...

```
root@70b800f7f822:/# id
uid=0(root) gid=0(root) groups=0(root)
```

```
root@70b800f7f822:/# ps aux
```

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
root	1	0.0	0.1	18240	3216	pts/0	Ss	13:45	0:00	bash
root	2846	0.0	0.1	34424	2796	pts/0	R+	13:48	0:00	ps aux

```
root@70b800f7f822:/# |
```

root	1033	0.2	3.8	414196	78364	?	Ssl	Oct19	8:23	/usr/bin/dockerd -H fd://
root	1238	0.0	0.5	292236	12124	?	Ssl	Oct19	1:15	_ docker-containerd -l unix:///v
root	14988	0.0	0.1	207328	2996	?	S1	13:45	0:00	_ docker-containerd-shim 70b
root	15003	0.0	0.1	18240	3216	pts/0	Ss+	13:45	0:00	_ bash
root	1036	0.0	0.1	28624	3100	?	Ss	Oct19	0:00	/lib/systemd/systemd-logind



```
❶ <1> root@70b800f7f8... ❷ <2> ubuntu@docker: ...
ubuntu@docker:~$ uname -a
Linux docker 4.4.0-96-generic #119-Ubuntu SMP Tue Sep 12 14:59:54 UTC 2017 x86_64 x86_64 x86_64 GNU
/Linux
ubuntu@docker:~$
```

```
❶ <1> root@70b800f7f8... ❷ <2> ubuntu@docker: ...
root@70b800f7f822:/# uname -a
Linux 70b800f7f822 4.4.0-96-generic #119-Ubuntu SMP Tue Sep 12 14:59:54 UTC 2017 x86_64 x86_64 x86_
64 GNU/Linux
root@70b800f7f822:/# |
```



① <1> root@70b800f7f8... ② <2> ubuntu@docker: ...

```
root@70b800f7f822:/# touch /i_was_here
root@70b800f7f822:/# ls /
bin  dev  home       lib   media   opt   root   sbin   sys   usr
boot etc  i_was_here lib64  mnt    proc   run    srv   tmp   var
root@70b800f7f822:/# rm etc/nsswitch.conf
root@70b800f7f822:/#
```



<1> root@70b800f7f8... <2> root@docker: /v...

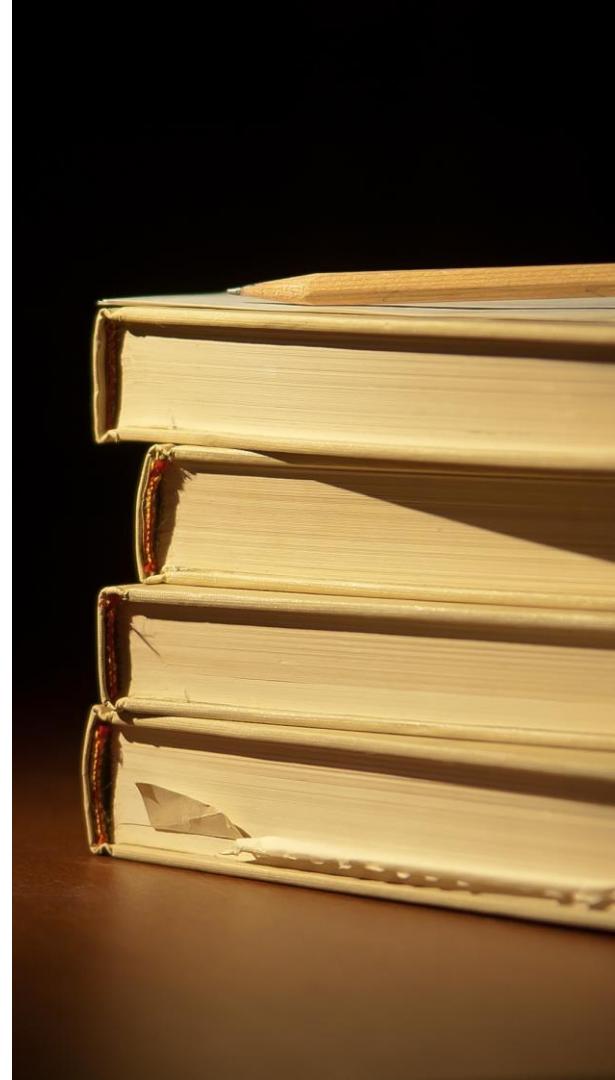
```
root@docker:/var/lib/docker/overlay2/f11b091eed6de9128513fc# 1 merged/
total 76K
drwxr-xr-x 2 root root 4.0K Oct  6 01:38 bin
drwxr-xr-x 2 root root 4.0K Apr 12 2016 boot
drwxr-xr-x 1 root root 4.0K Oct 21 13:45 dev
drwxr-xr-x 1 root root 4.0K Oct 21 13:53 etc
drwxr-xr-x 2 root root 4.0K Apr 12 2016 home
-rw-r--r-- 1 root root    0 Oct 21 13:53 i_was_here
drwxr-xr-x 1 root root 4.0K Sep 13 2015 lib
drwxr-xr-x 2 root root 4.0K Oct  6 01:38 lib64
```



```
<1> root@70b800f7fb... <2> root@docker: /v...
root@docker:/var/lib/docker/overlay2/f11b091eed6de912851325e9bb
fc# 1 diff/
total 20K
drwxr-xr-x 7 root root 4.0K Oct 21 13:53 etc
-rw-r--r-- 1 root root 0 Oct 21 13:53 i_was_here
drwxr-xr-x 3 root root 4.0K Sep 13 2015 lib
drwxrwxrwt 2 root root 4.0K Oct 21 13:47 tmp
drwxr-xr-x 7 root root 4.0K Oct 6 01:38 usr
drwxr-xr-x 5 root root 4.0K Oct 6 01:38 var
root@docker:/var/lib/docker/overlay2/f11b091eed6de912851325e9bb
fc# 1 diff/etc/
total 40K
drwxr-xr-x 3 root root 4.0K Oct 21 13:47 ca-certificates
-rw-r--r-- 1 root root 6.4K Oct 21 13:47 ca-certificates.conf
drwxr-xr-x 3 root root 4.0K Oct 21 13:47 gss
drwxr-xr-x 2 root root 4.0K Oct 21 13:47 ldap
-rw-r--r-- 1 root root 9.7K Oct 21 13:47 ld.so.cache
drwxr-xr-x 3 root root 4.0K Oct 21 13:47 logcheck
c----- 1 root root 0, 0 Oct 21 13:53 nsswitch.conf
drwxr-xr-x 4 root root 4.0K Oct 21 13:47 ssl
root@docker:/var/lib/docker/overlay2/f11b091eed6de912851325e9bb
fc# |
```

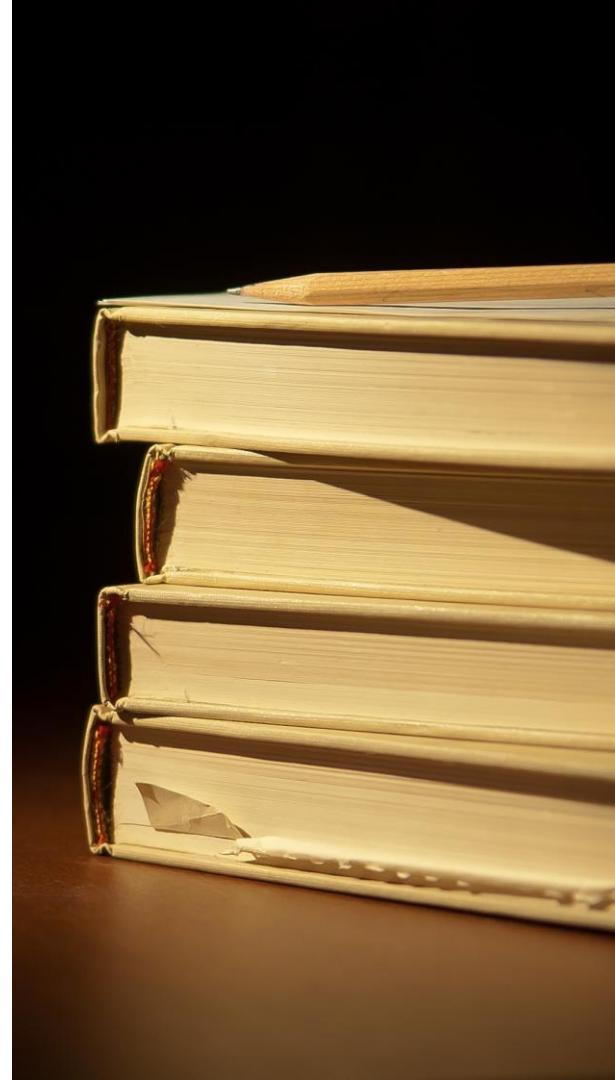
Container?

- OS Container
 - Group of processes isolated/controlled via OS functionality
- Application Container
 - One or more processes running in an OS container based on an Application Container Image
 - E.g. also “runtime instantiation of an application container image”
- Application Container Image
 - Portable base file system and configuration for an OS container.
 - Containing a self-contained application including all of its dependencies.



Container?

- Docker, rkt, LXC are (Application/OS) Container Management Solutions
 - Defining or using (e.g. the OCI format) Application Container Image Formats
 - Providing the filesystem layout for Application Containers
 - Which are then executed
 - as OS Containers containing the processes defined in the Image
 - using the created filesystem layout



Summary so far

- Containers are not virtual machines
- Shared resources
 - Storage/File system
 - NIC
 - Kernel



Docker Swarm

Motivation

- So far:
 - Isolated processes
 - Packaging/deployment
- But what about...
 - Scalability?
 - High availability?
 - Service discovery?
 - Ease of deployment?



Docker Swarm

- Swarm provides automatic scaling of containers across an arbitrary count of hosts
- Manager nodes handle cluster management tasks
 - Cluster state maintenance
 - Scheduling of services
 - API endpoints
- Worker nodes provide servicing and processing capabilities





```
<1> ubuntu@docker: ... <2> root@worker1: ~ Search

ubuntu@docker:~$ docker node ls
ID                      HOSTNAME        STATUS        AVAILABILITY      MANAGER STATUS
h3gcwbz5bxwpuambboitui1r8 *  docker          Ready        Active           Leader
ksyeqrklo88rne7wf0stncu34   worker1        Ready        Active
ubuntu@docker:~$ |
```

```
<1> ubuntu@docker: ... <2> root@worker1: ~ Search

ubuntu@docker:~$ docker service ls
ID                  NAME            MODE            REPLICAS      IMAGE
cy3wpc449lro      voting_worker    replicated     1/1          127.0.0.1:5000/worker:v0.01
ev91531amg64      voting_result    replicated     3/3          127.0.0.1:5000/result:v0.01
hzqw4xgaxxgo      voting_db       replicated     1/1          postgres:9.4
kpixt9gr7qbs      registry_registry replicated     1/1          registry:2
mc7hn2hiw4wz      voting_vote      replicated     5/5          127.0.0.1:5000/vote:v0.01
x886g8742441      voting_redis     replicated     1/1          redis:alpine
y82g1qe5acko      voting_sockpuppet replicated     0/0          127.0.0.1:5000/sockpuppet:v0.01
ubuntu@docker:~$ |
```

<1> ubuntu@docker: ... <2> root@worker1: ~

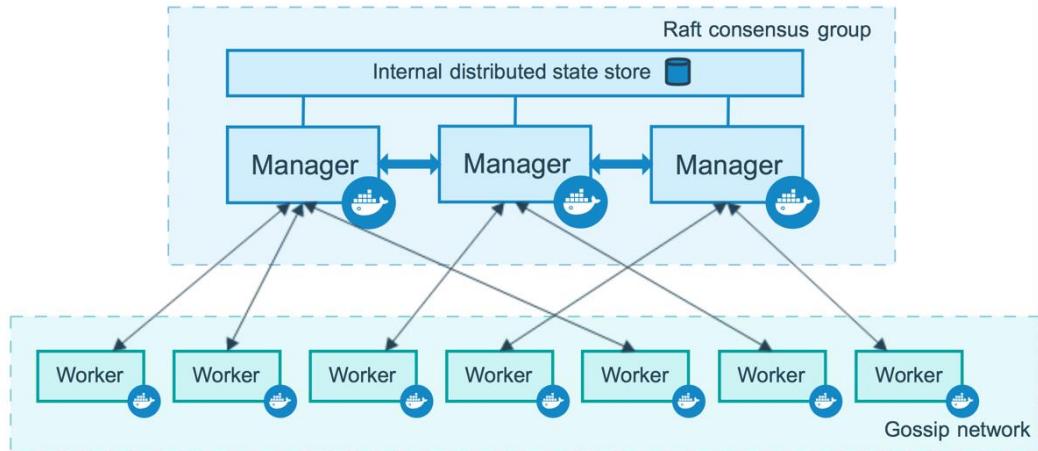
NAME	IMAGE
voting_db.1.0cs1nwpub7gaycpuqe7cmpend	postgres:9.4
voting_worker.1.lizilifrhbvs6f9yxyyyadc16	127.0.0.1:5000/worker:v0.01
voting_redis.1.bzu15g0k24iytrochzltij7h8	redis:alpine
voting_result.1.wy39fx0wcxtn8ht2j84sk8fur	127.0.0.1:5000/result:v0.01
voting_vote.5.vkxvwnw6r5960ch99g51utmkk	127.0.0.1:5000/vote:v0.01
voting_vote.2.1e22sdzke5aa1ymvrpp6gqo6c	127.0.0.1:5000/vote:v0.01
registry_registry.1.ys6rhtk34vsh3udr0s23m5qg1	registry:2

<1> ubuntu@docker: ... <2> root@worker1: ~

NAME	IMAGE
voting_result.3.0jr8k8jy291dp6u6zd8lq74aq	127.0.0.1:5000/result:v0.01
voting_result.2.yuohb5rq15ku50z37sggw9ykc	127.0.0.1:5000/result:v0.01
voting_vote.4.iywglyty8hjrnbbbe7my4n0c2i	127.0.0.1:5000/vote:v0.01
voting_vote.1.ubylppxddk3hasknh5b62k195	127.0.0.1:5000/vote:v0.01
voting_vote.3.mxzkeusjw93t9m86mc75bvjro	127.0.0.1:5000/vote:v0.01

Swarms from the inside

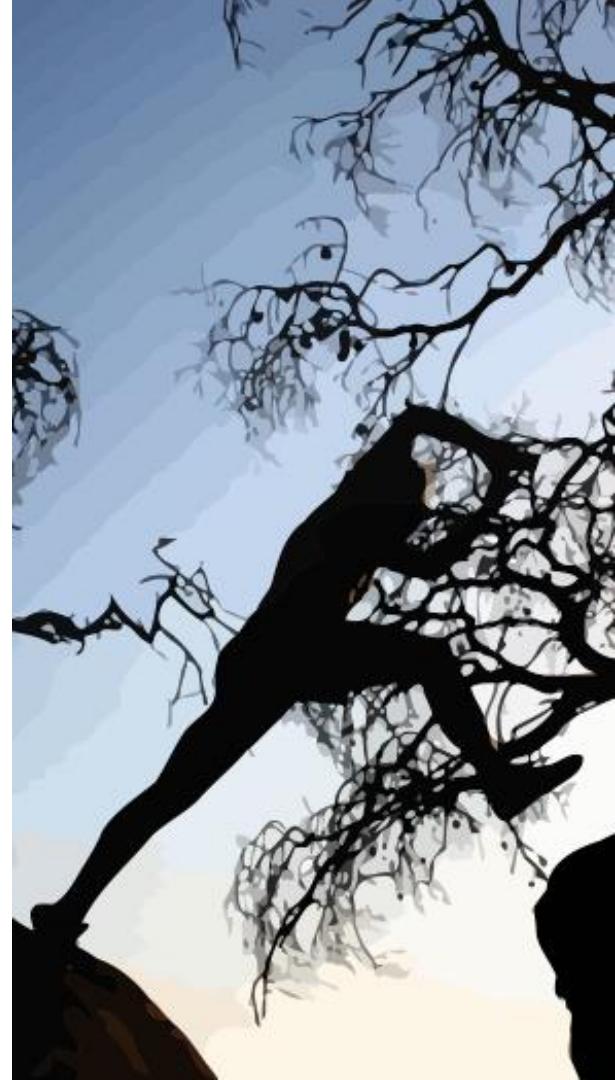
- Builds on the Raft Consensus Algorithm
- Manager nodes keep the cluster state consistent with the Raft log
- In case of failure:
 - majority of nodes needs to agree on values
 - $(N-1)/2$ failures tolerated, otherwise no more requests are processed



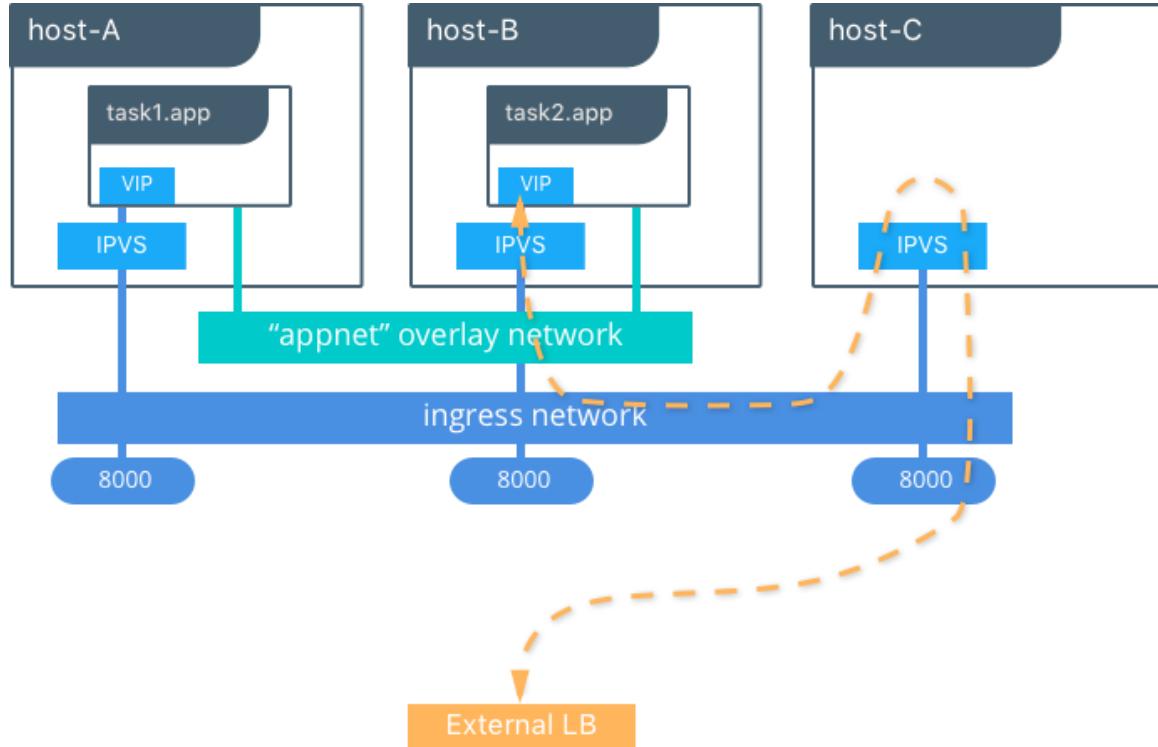
Source: [Docker](#)

Advantages

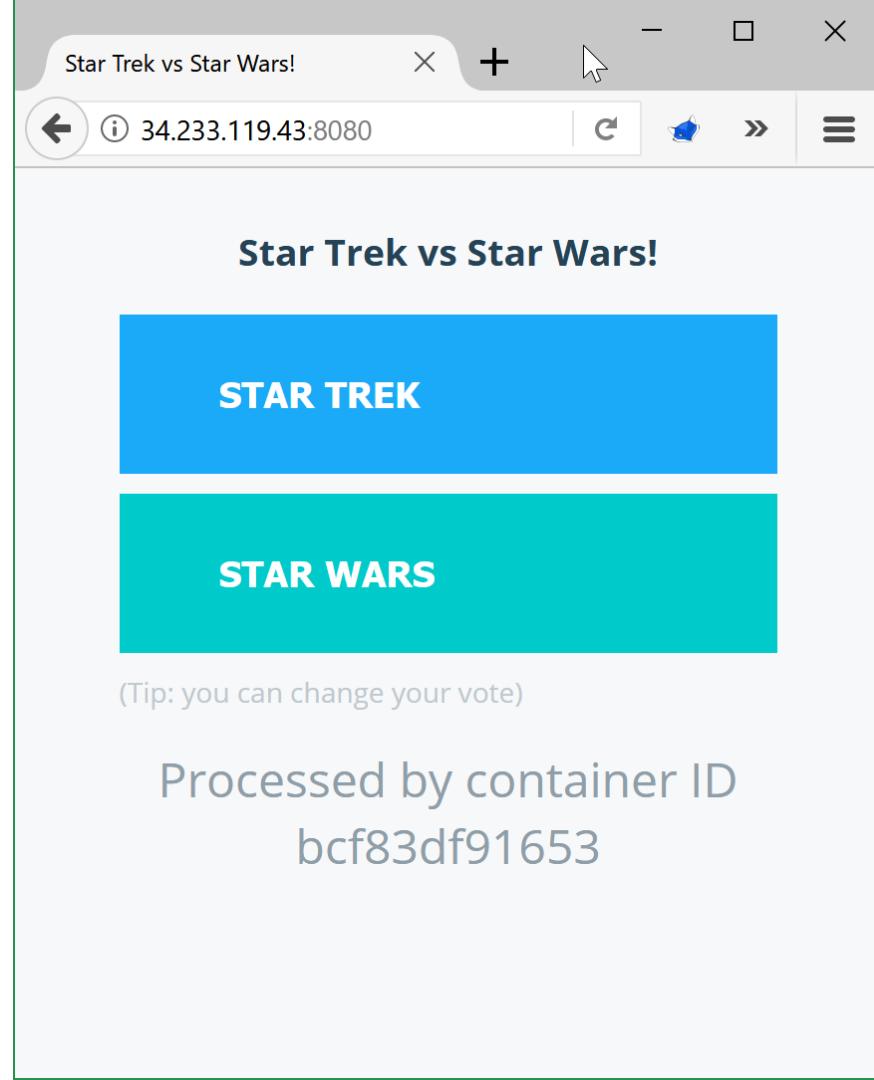
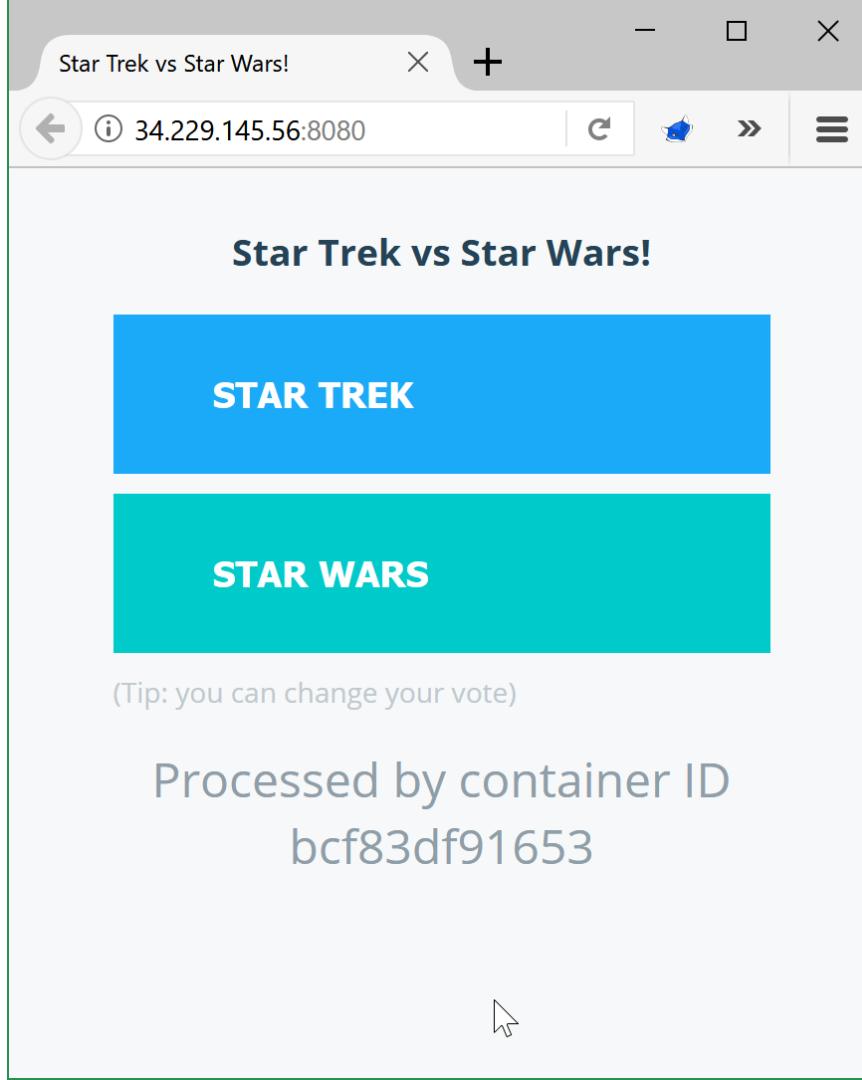
- Scalability
 - Additional nodes can easily be joined into the swarm
- High availability
 - Still and forever: Apps need to be designed for HA
 - If that is the case: Platform failover capabilities
- Service discovery
 - Internal DNS
 - Services exposed on *all* worker nodes (routing mesh/ingress networking)
- Ease of deployment
 - Remember demo



Routing Mesh



Source: [1]





Corollary: DevOps Complexity Kills The Pentester

Additions and Deletions per week

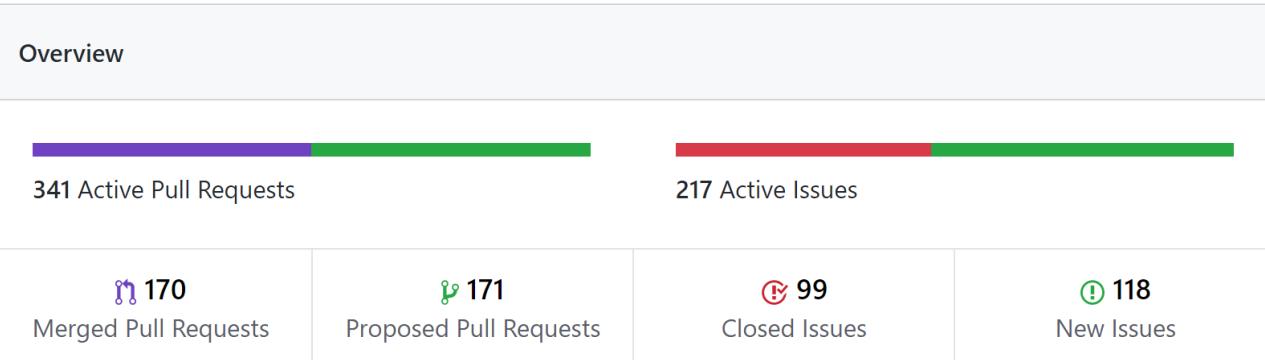
Docker Engine

Source

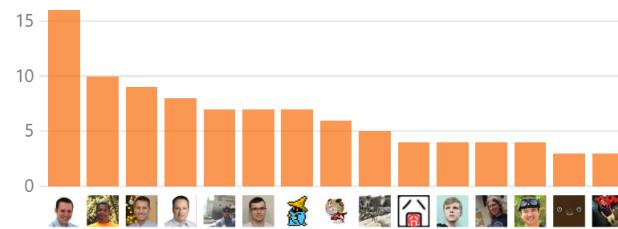


Kubernetes

Source



Excluding merges, **69 authors** have pushed **142 commits** to master and **161 commits** to all branches. On master, **2,984 files** have changed and there have been **23,580 additions** and **10,433 deletions**.



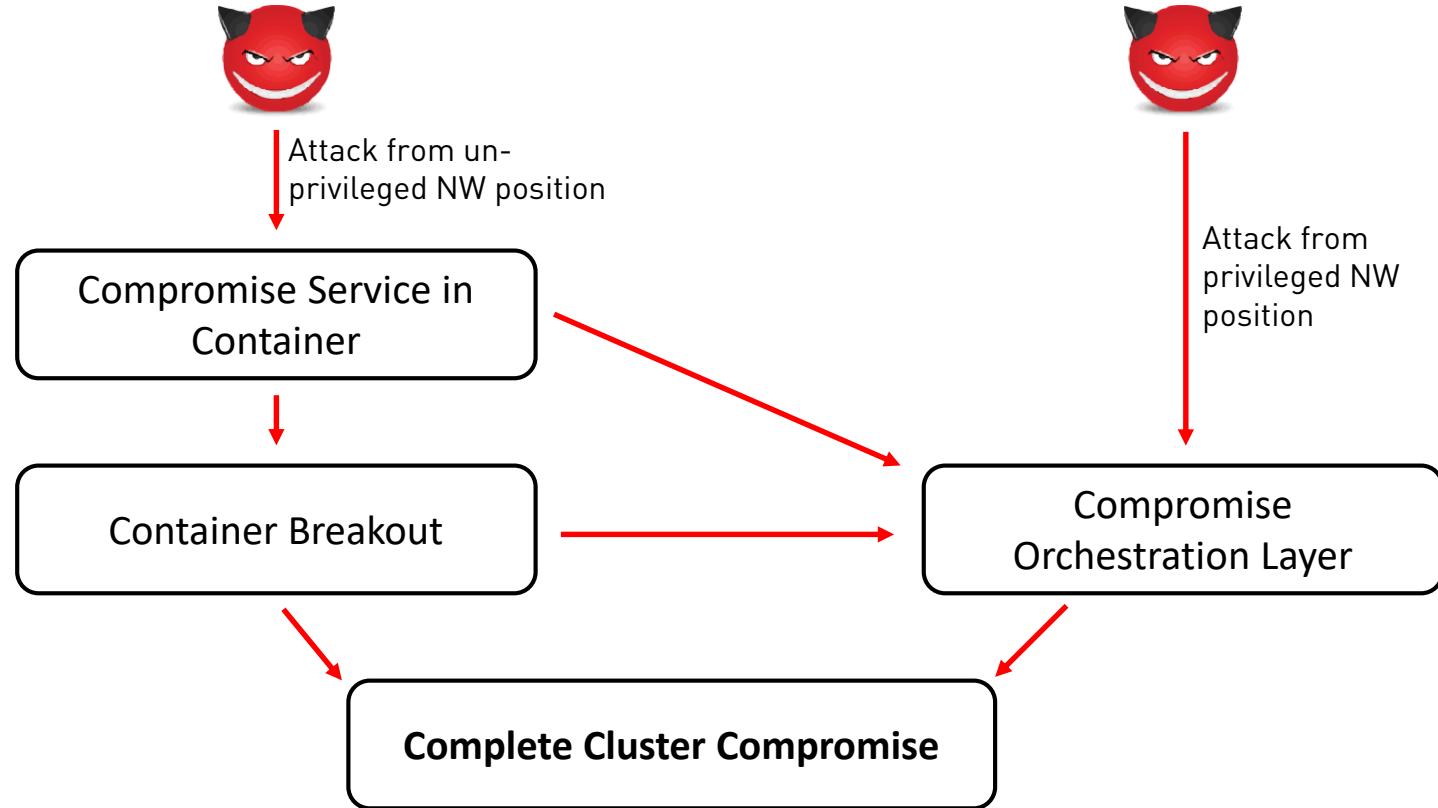


Attack Surface

Attack Surface

- Container breakout
- Cluster compromise/lateral movement
- Remote compromise orchestration layer





Remote Compromise of the Orchestration Layer

Remote Compromise

- containerd socket
 - No vulnerabilities yet
 - No authentication per default
 - access to containerd socket = root compromise.
- Docker image extraction vulnerabilities in the past
 - General note: Archive handling is tricky from a security perspective

Security Posture

- Known vulnerabilities in the DevOps technology space?
 - Few, differing severity.
- Generally good design and platform choices
- Differing focus on security resources
 - Positive: Docker
 - <https://www.docker.com/docker-cve-database>
 - <https://www.docker.com/docker-security>
 - Room for improvement: Kubernetes
 - <https://kubernetes.io/security/>
 - [Issue Tracker area:security](#)



Container Breakout



Container Breakout Comparison

Source

		<i>Physical Host</i>	<i>Virtual Machine</i>	<i>Container</i>
Container Breakout Comparison	<i>Shared Resources</i>	Sharing the network	Sharing the host's hardware	Sharing the kernel
	<i>Attack Scenario</i>	Attacks via network on open ports etc.	Attacks on the Hypervisor itself	Attacks via Syscall on the kernel isolation (Namespaces, Cgroups, ...)
	<i>Protection Measures</i>	Portfilter, firewalls, segmentation of networks	Robust hypervisor	Security controls within the container manager, SELinux, Capabilities
	<i>Operational Effort</i>	Easy, best practices	Complex, but centrally managable	Complex due to relatively big attack surface

Kernel Exploitation

- Shared kernel between container and host => Kernel vulnerability violates isolation
- Happen on a regular basis:
 - https://www.cvedetails.com/vulnerability-list/vendor_id-33/product_id-47/cvssscoremin-9/cvsscoremax-/Linux-Linux-Kernel.html
 - <http://seclist.us/list-of-linux-post-exploitation-enumeration-and-exploit-checking-tools.html>
 - <https://github.com/rebootuser/LinEnum>



Kernel Exploitation

- Identify containment:
 - <https://github.com/jessfraz/amicontained>
- Breaking out of namespaces:
 - <https://grsecurity.net/~spender/exploits/enlightenment.tgz>
 - Which is from 2013, btw.



Container Isolation

- Default docker containers run as root
- Pre-1.0 world: Containers do not contain.
 - E.g. Device/sysfs/procfs access as root from within container
- Post-1.0 (roughly):
 - Still running as root per default
 - Reduced capabilities
 - Compared to root, increased compared to regular user
 - Default AppArmor/Selinux profile
 - Default seccomp filter

Container Isolation

- Default Docker 17.06 container:
 - No immediate breakout possible.
- **Don't**
 - use `--privileged`
 - run containers as root
 - use volumes in an unmonitored way
- **Do**
 - Drop all privileges
 - Run containers as non-root
 - Not disable MAC/seccomp
 - Avoid volumes, monitor use of volumes

```
① <1> root@ecdd35d84b... ② <2> ubuntu@docker: ...
ubuntu@docker:~$ id
uid=1000(ubuntu) gid=1000(ubuntu) groups=1000(ubuntu),4(adm),20(dialout),24(cdrom),25(floppy),27(sudo),29
(audio),30(dip),44(video),46(plugdev),109(netdev),110(lxd),999(docker)
ubuntu@docker:~$
ubuntu@docker:~$
ubuntu@docker:~$
ubuntu@docker:~$ docker run -it --rm ubuntu bash
root@ecdd35d84b28:/#
root@ecdd35d84b28:/#
root@ecdd35d84b28:/# id
uid=0(root) gid=0(root) groups=0(root)
root@ecdd35d84b28:/#
root@ecdd35d84b28:/# |
```

```
root@887557865df0: /  
 ① <1> root@887557865d... ② <2> ubuntu@docker: ...  
ubuntu@docker:~$ docker run -it --rm -v /:/tmp ubuntu bash  
root@887557865df0:/#  
root@887557865df0:/# ls /tmp  
bin  etc          initrd.img.old  lost+found  opt    run    srv  usr  
boot home         lib           media      proc   sbin   sys  var  
dev  initrd.img  lib64        mnt       root   snap   tmp  vmlinuz  
root@887557865df0:/#  
root@887557865df0:/# cat /tmp/etc/shadow  
root:*:17156:0:99999:7:::  
daemon:*:17156:0:99999:7:::  
bin:*:17156:0:99999:7:::
```



```
<1> root@11d221b711... <2> ubuntu@docker: ...
ubuntu@docker:~$ id
uid=1000(ubuntu) gid=1000(ubuntu) groups=1000(ubuntu),4(adm),20(dialout),24(cdrom),25(floppy),27(sudo),29
(audio),30(dip),44(video),46(plugdev),109(netdev),110(1xd),999(docker)
ubuntu@docker:~$
ubuntu@docker:~$ getpcaps $$
Capabilities for `18687': =
ubuntu@docker:~$
ubuntu@docker:~$ docker run -it --rm ubuntu bash
root@11d221b71151:/#
root@11d221b71151:/# getpcaps 1
Capabilities for `1': = cap_chown, cap_dac_override, cap_fowner, cap_fsetid, cap_kill, cap_setgid, cap_setuid,
cap_setpcap, cap_net_bind_service, cap_net_raw, cap_sys_chroot, cap_mknod, cap_audit_write, cap_setfcap+eip
root@11d221b71151:/#
root@11d221b71151:/#
```

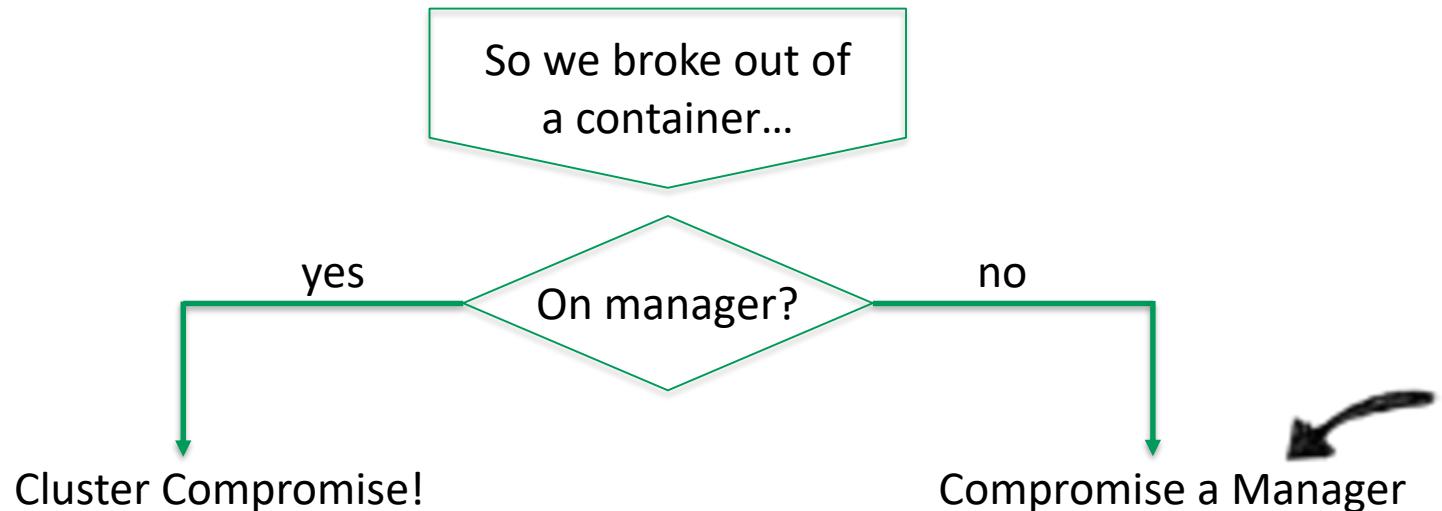
```
root@docker:~# getpcaps $$  
Capabilities for `24012': = cap_chown, cap_dac_override, cap_dac_read_search, cap_fowner, cap_fsetid, cap_kill  
, cap_setgid, cap_setuid, cap_setpcap, cap_linux_immutable, cap_net_bind_service, cap_net_broadcast, cap_net_admin,  
cap_net_raw, cap_ipc_lock, cap_ipc_owner, cap_sys_module, cap_sys_rawio, cap_sys_chroot, cap_sys_ptrace, cap_sys_pacct,  
cap_sys_admin, cap_sys_boot, cap_sys_nice, cap_sys_resource, cap_sys_time, cap_sys_tty_config, cap_mknod, cap_lease,  
cap_audit_write, cap_audit_control, cap_setfcap, cap_mac_override, cap_mac_admin, cap_syslog, cap_wake_alarm,  
cap_block_suspend, 37+ep  
root@docker:~#
```

```
ubuntu@docker:~$ docker run -it --rm --privileged ubuntu bash
root@f22d299b9453:#
root@f22d299b9453:# getpcaps 1
Capabilities for `1': = cap_chown,cap_dac_override,cap_dac_read_search,cap_fowner,cap_fsetid,cap_kill,cap_setgid,cap_setuid,cap_setpcap,cap_linux_immutable,cap_net_bind_service,cap_net_broadcast,cap_net_admin,cap_net_raw,cap_ipc_lock,cap_ipc_owner,cap_sys_module,cap_sys_rawio,cap_sys_chroot,cap_sys_ptrace,cap_sys_pacct,cap_sys_admin,cap_sys_boot,cap_sys_nice,cap_sys_resource,cap_sys_time,cap_sys_tty_config,cap_mknod,cap_lease,cap_audit_write,cap_audit_control,cap_setfcap,cap_mac_override,cap_mac_admin,cap_syslog,cap_wake_alarm,cap_block_suspend,37+eip
root@f22d299b9453:#
root@f22d299b9453:#
root@f22d299b9453:# mount /dev/xvda1 /mnt/
root@f22d299b9453:# ls /mnt/
bin  etc      initrd.img.old  lost+found  opt    run    srv    usr      vmlinuz.old
boot home     lib          media       proc   sbin   sys    var
dev   initrd.img lib64       mnt        root   snap   tmp    vmlinuz
root@f22d299b9453:# echo 1 > /proc/sys/net/ipv4/ip_forward
root@f22d299b9453:# |
```



Cluster Lateral Movement

Cluster Security

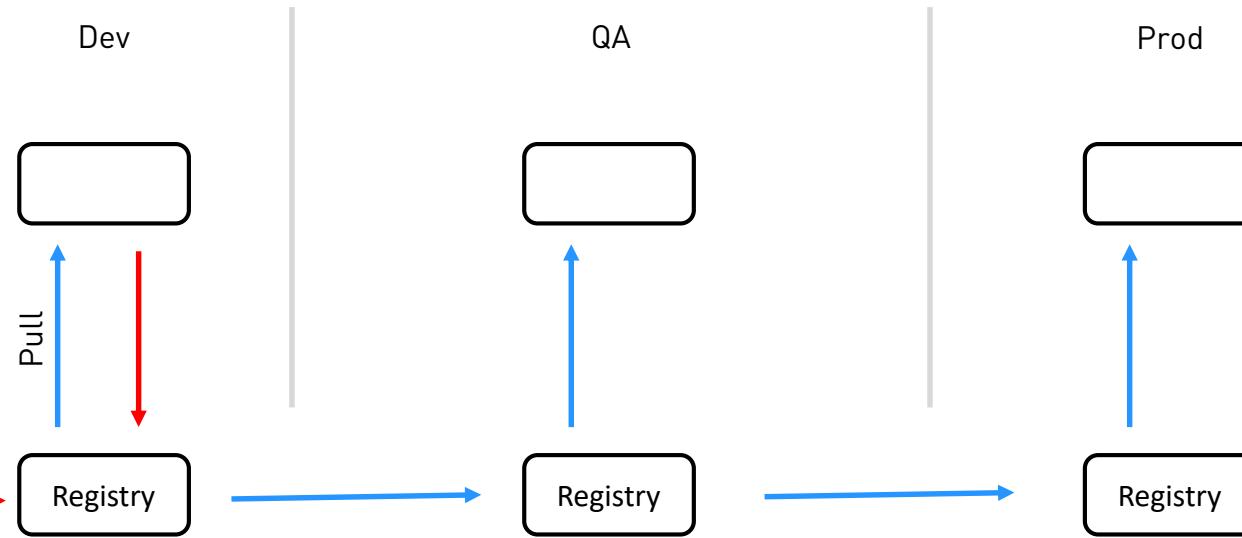


Becoming a Manager

- Wait ☺
- Kill infected container
 - => wait for respawn on manager
 - => break out again.
- Auto scaling?
 - => load on infected service
 - => wait for scale-up on manager
 - => break out again
- Push rights to registry?
 - => modify image
 - => put load on services/wait

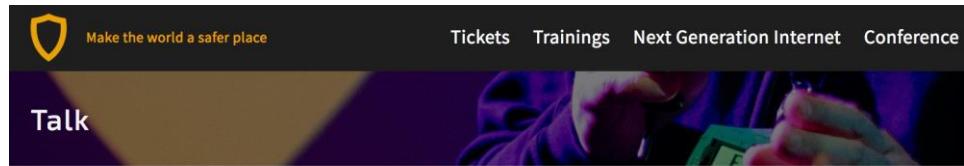


Registry Misconfiguration



Cluster Compromise via DCMS Backend

- DCMS often used in Container Orchestration
 - E.g. etcd in k8s
- Bottom line:
 - Via some stored configuration, code exec almost always possible.



How we hacked Distributed Configuration Management Systems

 March 23, 2017 (at 10:30) in Attack and Research

With increase in necessity of distributed applications, coordination and configuration management tools for these classes of applications have popped up. These systems might pop-up occasionally during penetration tests. The major focus of this research was to find ways to abuse these systems as well as use them for getting deeper access to other systems.

The talk deals with how we came across and exploited different configuration management systems during our pentests.

Cluster Lateral Movement – Vulnerabilities

Docker Swarm

Docker Swarm Lateral Movement

- Via Swarm control plane
 - No known vulnerabilities
- Relevant components:
 - dockerd/containerd, TCP 2375
 - not require for swarm, for sake of completeness – you don't want to see this anywhere ;-)
 - Dockerd - managers
 - TCP/UDP 7946: Gossip-based overlay network control plane
 - TCP 2377: RAFT sync
 - Dockerd – all nodes
 - TCP 4789: VXLAN data plane

Docker Swarm Hardening

- No traditional configuration hardening for Swarm
- Preferred solution:
 - One Swarm per application
- If not possible:
 - Kubernetes does not run workloads on the managers by default
 - Docker can be configured to do the same
- Implement access control on your registry
- Pay attention to container hardening against breakouts

Cluster Lateral Movement – Vulnerabilities

Kubernetes

K8s Vulnerabilities

- Kubelet port (TCP 10250) allowed unauthenticated command execution
 - ... e.g. also in central management containers
 - Fixed in 1.5
- Default service tokens grant cluster admin privileges
 - Fixed if RBAC is used, default from 1.6 on with kubeadm
- *Both vulnerabilities result in complete cluster compromise from within an unprivileged container!*

Enumeration K8s - Ports

- TCP 10250 kubelet
- TCP 10248 kubelet healthcheck
- TCP 10255 kubelet R/O w/o auth
- TCP 10256 kube-proxy healthcheck
- TCP 10251 kube-scheduler
- TCP 10252 kube-controller-manager
- TCP 6443 kube-apiserver

Enumeration Network Plugins

- Ports depend entirely on plugins
 - Calico for example has a BGP daemon listening, weave running (various) other daemons
- Multiple and independent attack surfaces!
 - Compromise results in cluster compromise as well!
- High number of relevant plugins:
 - Calico
 - Flannel
 - Weave
 - Contrail
 - ...

Enumeration Network Plugins

- Barely any known vulnerabilities
 - => No existing vulnerability research!
- No comprehensive security advisories/information on the project websites
 - Exception: OpenVSwitch



<1> root@k8s-master... <2> root@k8s-node1...

Search

```
root@k8s-master:~# kubectl get nodes
NAME           STATUS   ROLES      AGE     VERSION
ip-192-168-0-22   Ready    master     4h     v1.8.1
ip-192-168-0-91   Ready    <none>    4h     v1.8.1
root@k8s-master:~#
root@k8s-master:~# |
```

```
root@k8s-master:~# kubectl run my-shell2 --rm -i --tty --image ubuntu -- bash
If you don't see a command prompt, try pressing enter.
root@my-shell2-78d75fb694-mck4k:/#
root@my-shell2-78d75fb694-mck4k:/#
```

```
root@my-shell2-78d75fb694-mck4k:/# ip a s dev eth0
18: eth0@if19: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu
      link/ether be:ce:77:56:e3:22 brd ff:ff:ff:ff:ff:ff
      inet 10.44.0.2/12 scope global eth0
```



```
root@my-shell2-78d75fb694-mck4k:/#
root@my-shell2-78d75fb694-mck4k:/# curl -k https://192.168.0.22:6443/api
{
  "kind": "APIVersions",
  "versions": [
    "v1"
  ],
  "serverAddressByClientCIDRs": [
    {
      "clientCIDR": "0.0.0.0/0",
      "serverAddress": "192.168.0.22:6443"
    }
  ]
}root@my-shell2-78d75fb694-mck4k:/# |
```



```
root@my-shell2-78d75fb694-mck4k:/# curl -k https://192.168.0.22:6443/api/v1/namespaces/default/pods/
{
  "kind": "Status",
  "apiVersion": "v1",
  "metadata": {

  },
  "status": "Failure",
  "message": "pods is forbidden: User \\"system:anonymous\\" cannot list pods in the namespace \\"default\\"
,
  "reason": "Forbidden",
  "details": {
    "kind": "pods"
  },
  "code": 403
}root@my-shell2-78d75fb694-mck4k:/# |
```

```
root@my-shell2-78d75fb694-mck4k:/# cat /var/run/secrets/kubernetes.io/serviceaccount/token  
eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJrdWJlc...  
VsdCISImt1YmVybmV0ZXMuaw8vc2VydmljZWFjY291bnQvc2VydmljZS1hY2NvdW50LnVpZCI6IjMxNzFjYjQ5LWI2NDQtMTF1Ny1iYjE  
2LTBhYmE4Y2UwMGQ5YYIsInN1YiI6InN5c3R1bTpzZXJ2aN1YWNjb3VudDpkZWZhdWx0OmR1ZmF1bHQifQ.vuDH0LnTgrHHzoLKaPWKV  
05_zuz87BYeYkJkIUmQ00HIrnUJSjQ1grA-WX6cFWhmpykxe5JPTqbueDu2L9C5j_O_-5mRtx1CAQKPD...  
KUouVv5x2PA0ubLSw8uwCVoIPA1s7h7lrebpj4bkUcM0rQE...  
wvF5ArB0qFqrLoIj7leviHP7Fv8cWrgFyt06SkeeZTPMNj4HkrjngMuHYoYdLY61YERsyCVx8x2e64Pox5vvELilDMyNiUZMc...  
1cqg-Qroot@my-shell2-78d75fb694-mck4k:/#
```



```
root@my-shell2-78d75fb694-mck4k:/# KUBE_TOKEN=$(cat /var/run/secrets/kubernetes.io/serviceaccount/token)
root@my-shell2-78d75fb694-mck4k:/# curl -k -H "Authorization: Bearer $KUBE_TOKEN" https://192.168.0.22:6443/api/v1/namespaces/default/pods/
{
    "kind": "Status",
    "apiVersion": "v1",
    "metadata": {

    },
    "status": "Failure",
    "message": "pods is forbidden: User \\"system:serviceaccount:default:default\\" cannot list pods in the namespace \\"default\\",
    "reason": "Forbidden",
    "details": {
        "kind": "pods"
    },
    "code": 403
}root@my-shell2-78d75fb694-mck4k:/# |
```

```
root@my-shell2-78d75fb694-mck4k:/# kubectl config set-cluster test --server=https://192.168.0.22:6443
Cluster "test" set.
root@my-shell2-78d75fb694-mck4k:/#
root@my-shell2-78d75fb694-mck4k:/#
root@my-shell2-78d75fb694-mck4k:/# kubectl get all
Error from server (Forbidden): daemonsets.extensions is forbidden: User "system:serviceaccount:default:default" cannot list daemonsets.extensions in the namespace "default"
Error from server (Forbidden): deployments.extensions is forbidden: User "system:serviceaccount:default:default" cannot list deployments.extensions in the namespace "default"
Error from server (Forbidden): replicasesets.extensions is forbidden: User "system:serviceaccount:default:default" cannot list replicasesets.extensions in the namespace "default"
Error from server (Forbidden): deployments.apps is forbidden: User "system:serviceaccount:default:default" cannot list deployments.apps in the namespace "default"
Error from server (Forbidden): statefulsets.apps is forbidden: User "system:serviceaccount:default:default" cannot list statefulsets.apps in the namespace "default"
Error from server (Forbidden): daemonsets.apps is forbidden: User "system:serviceaccount:default:default" cannot list daemonsets.apps in the namespace "default"
Error from server (Forbidden): replicasesets.apps is forbidden: User "system:serviceaccount:default:default" cannot list replicasesets.apps in the namespace "default"
Error from server (Forbidden): horizontalpodautoscalers.autoscaling is forbidden: User "system:serviceaccount:default:default" cannot list horizontalpodautoscalers.autoscaling in the namespace "default"
Error from server (Forbidden): jobs.batch is forbidden: User "system:serviceaccount:default:default" cannot list jobs.batch in the namespace "default"
Error from server (Forbidden): cronjobs.batch is forbidden: User "system:serviceaccount:default:default" cannot list cronjobs.batch in the namespace "default"
Error from server (Forbidden): pods is forbidden: User "system:serviceaccount:default:default" cannot list pods in the namespace "default"
```

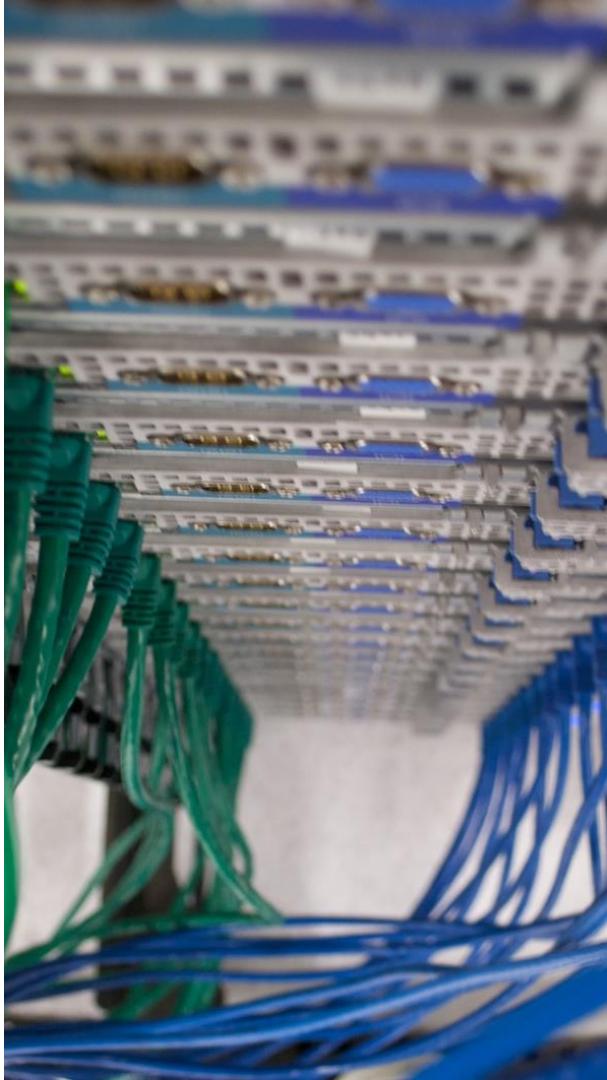
Post Exploitation

Network Isolation Issues

- Recap:
 - Containers share IP address of the host
- How to keep container from...
 - accessing non-container services on the host (e.g. SSH)?
 - accessing backend systems (e.g. NFS storage only to be used by host)?
 - accessing services on swarm members?
 - accessing the orchestration control plane?
- Network movement trivially possible

NW Isolation Container => Host

- Option 1: iptables
- Option 2: namespaces



iptables

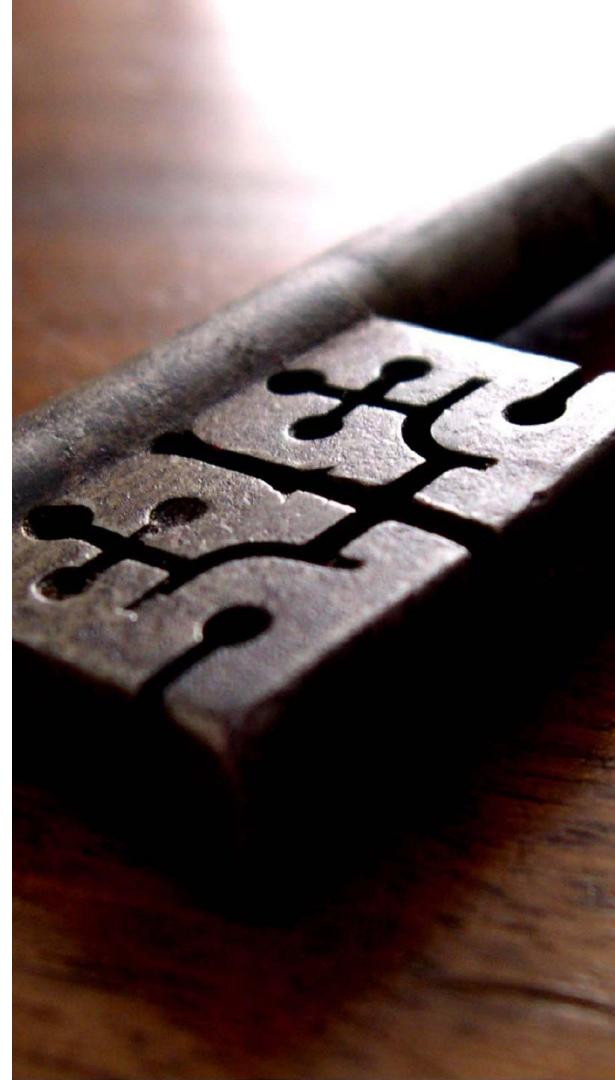
- Docker maintains dedicated iptables chains
 - Changes restricted to those chains as far as possible
 - Some FORWARD settings required.
- Modifications in non-Docker-chains should work
 - Our (non-comprehensive/-production) experience so far: Docker is quite well restrained to its own iptables entries
- `iptables -A INPUT -p tcp --dport 22 -i docker_gwbridge -j DROP`

Network Namespace

- Different network namespaces result in different routing tables
- Put SSH service in dedicated network namespace
 - Service will only be accessible via next layer 3 device
 - Unless you can use raw sockets or tamper with your devices/netmask/routing ;-)
 - Implement filtering there
 - Operational nightmare ☺

Credential Management

- Container Orchestration solutions offer built-in secret management
- Always check /var/run/secrets!



Conclusions

- Container Orchestration solutions are great hosting platforms from a functionality perspective
 - Not covered in this talk: How they can actually be used to improve security.
 - Also not covered image governance challenges
- Don't rely on isolation capabilities (yet).
- Don't treat them like a hypervisor



Conclusions

**Go do vuln research on the
DevOps/.io technologies!**



Obrigado pela atenção!

Questions?



mluft@ernw.de



@uchi_mata



www.ernw.de

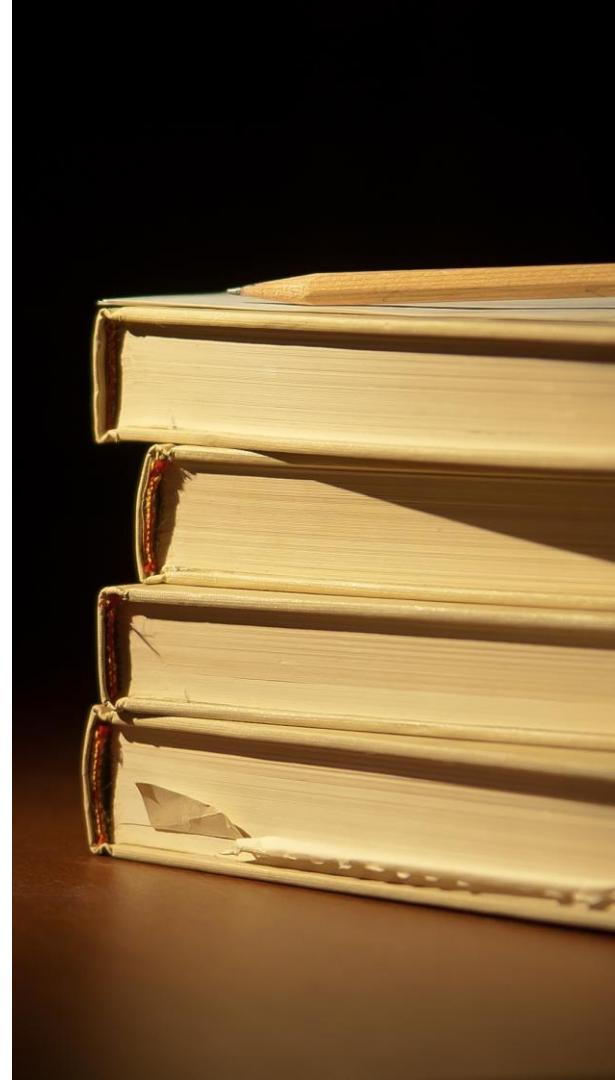


www.insinuator.net



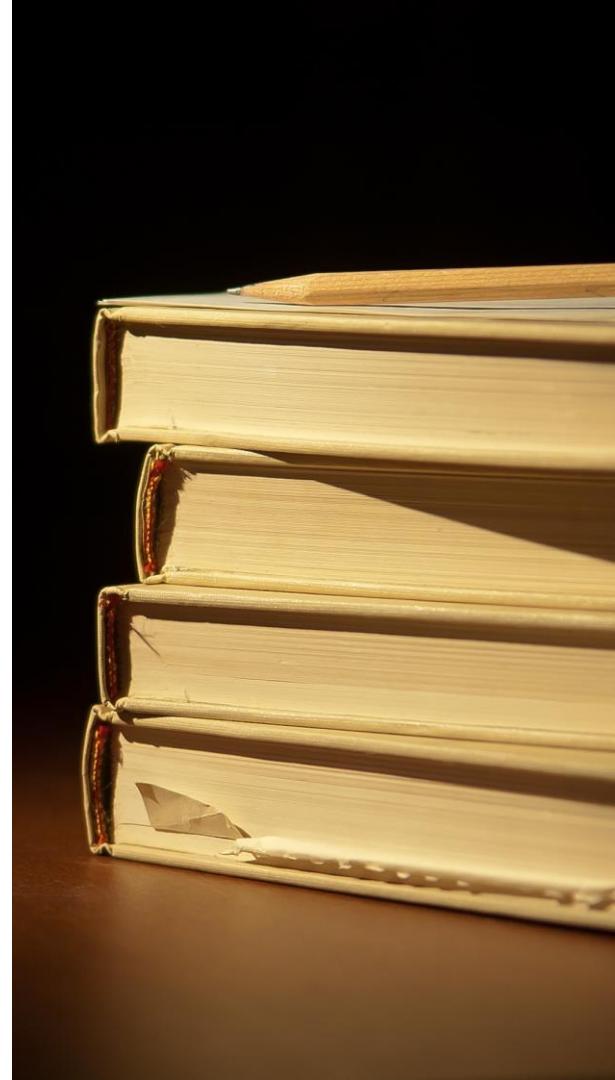
Sources

- [cgroups]
 - <https://lwn.net/Articles/524935/>
- [namespaces]
 - <https://lwn.net/Articles/259217/>
 - <https://lwn.net/Articles/524952/>
 - <https://lwn.net/Articles/531114/>
 - <https://lwn.net/Articles/531381/>
 - <https://lwn.net/Articles/531419/>
 - <https://lwn.net/Articles/532748/>
 - <https://lwn.net/Articles/532593/>
 - Hands-on network namespaces:
<https://www.howtoforge.com/linux-namespaces>



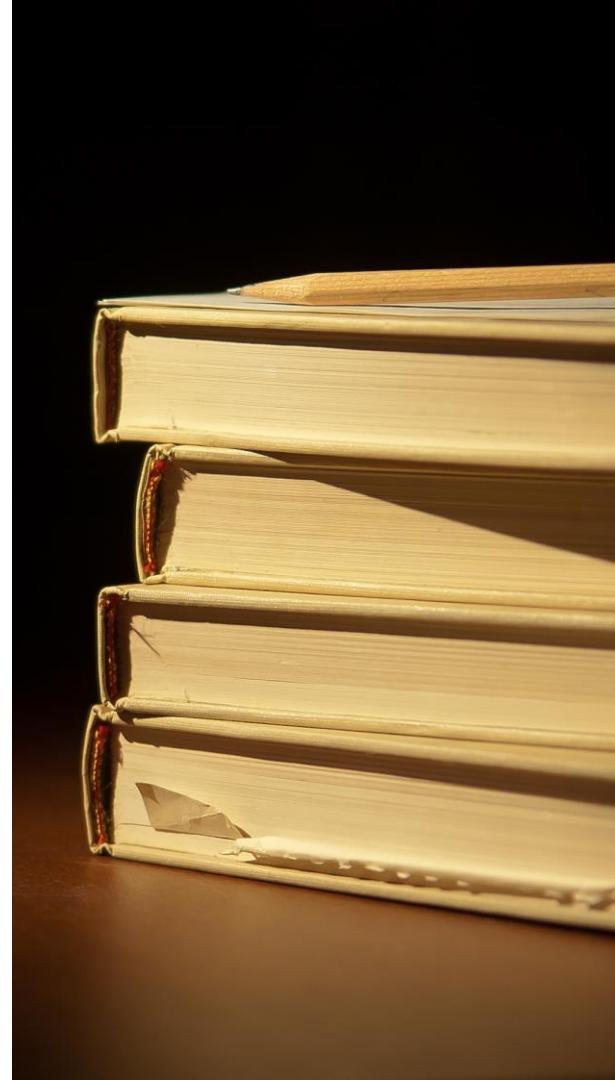
Sources

- [docker_images]
 - <https://docs.docker.com/engine/reference/builder/>
 - https://docs.docker.com/engine/userguide/eng-image/dockerfile_best-practices/
 - <https://docs.docker.com/engine/userguide/containers/dockerimages/>



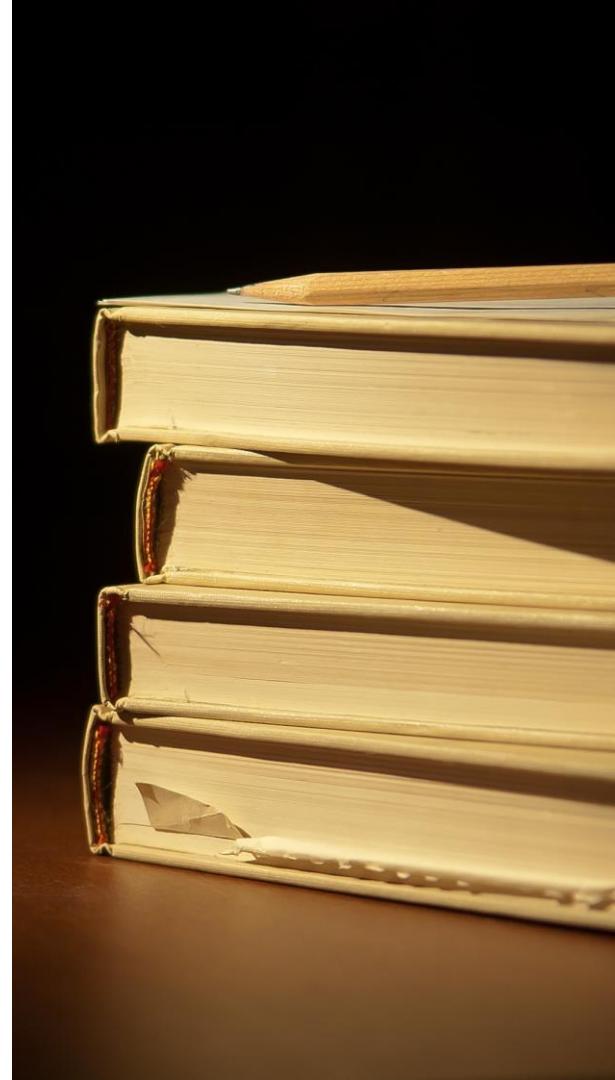
Sources

- [NCCSecuringContainers]
 - <https://www.nccgroup.trust/us/about-us/newsroom-and-events/blog/2016/april/understanding-and-hardening-linux-containers/>
- [DockerContainerSeclntro]
 - https://www.docker.com/sites/default/files/WP_IntrotoContainerSecurity_08.19.2016.pdf
- [CISDockerHardening]
 - https://benchmarks.cisecurity.org/tools2/docker/CIS_Docker_1.12.0_Benchmark_v1.0.0.pdf



Sources

- Docker Bench: Checking for best practices
 - <https://github.com/docker/docker-bench-security>
- Jérôme Petazzoni on Docker Security
 - E.g.: Containers, Docker, and Security: State of the Union
 - http://events.linuxfoundation.org/sites/events/files/slides/Containers,%20Docker,%20and%20Security_%20State%20of%20the%20Union.pdf
- <http://opensource.com/business/14/9/security-for-docker>
- <https://zeltser.com/security-risks-and-benefits-of-docker-application/>



Sources

- <http://devsecops.github.io/>
 - <https://github.com/devsecops/awesome-devsecops>
- <https://raesene.github.io/blog/2017/04/02/Kubernetes-Service-Tokens/>
- <https://raesene.github.io/blog/2016/10/08/Kubernetes-From-Container-To-Cluster/>

