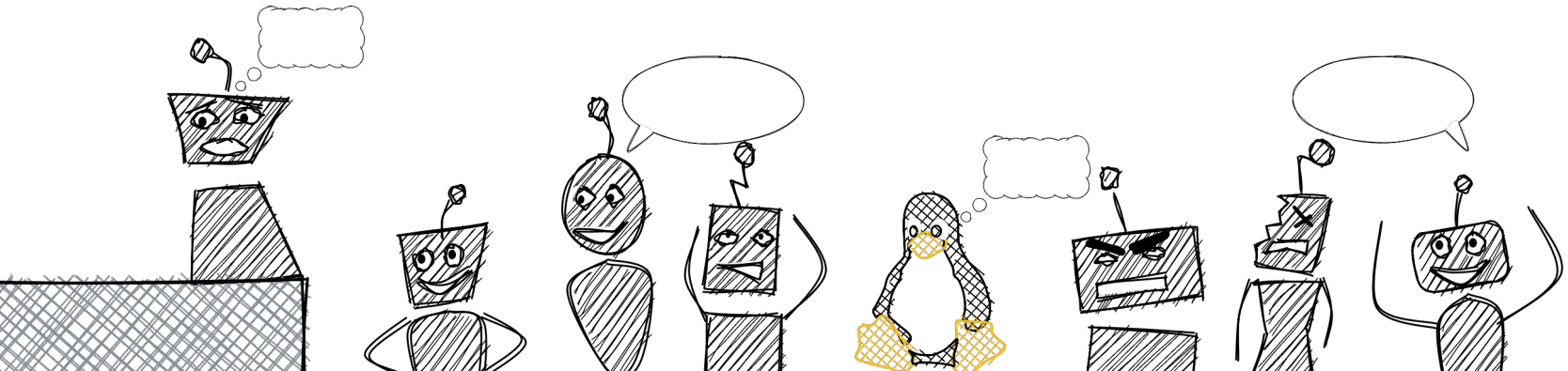
The 4C's of Cloud Native Security



Valentin Hristev
Staff Site Reliability Engineer


The 4C's of Cloud Native Security


Container and Image security


Runtime security


Kubernetes pod security


Kubernetes network security

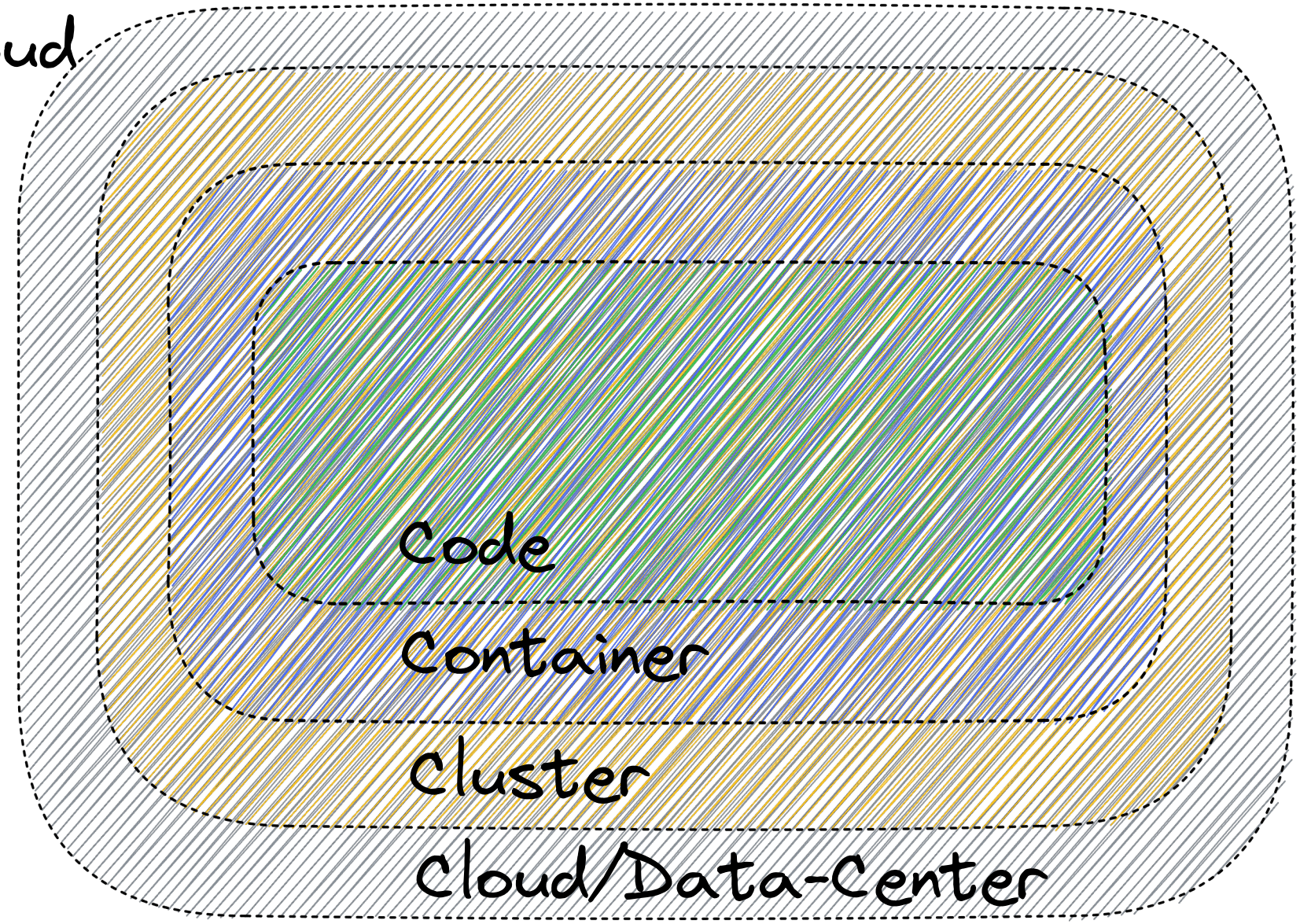

Behavioral Analysis


Supply Chain Security

The 4C's of Cloud Native Security

Security is a complex multi-dimensional beast, and it must be contained on different levels.

The 4C's of Cloud Native Security



Cloud / Datacenter security

- The 6 Layers of Google Data Center Security
  - Layer 1: Signage and Fencing
  - Layer 2: Secure Perimeter
  - Layer 3: Building Access
  - Layer 4: Security Operations Center
  - Layer 5: Data Center Floor
  - Layer 6: Secure Hard Drive Destruction

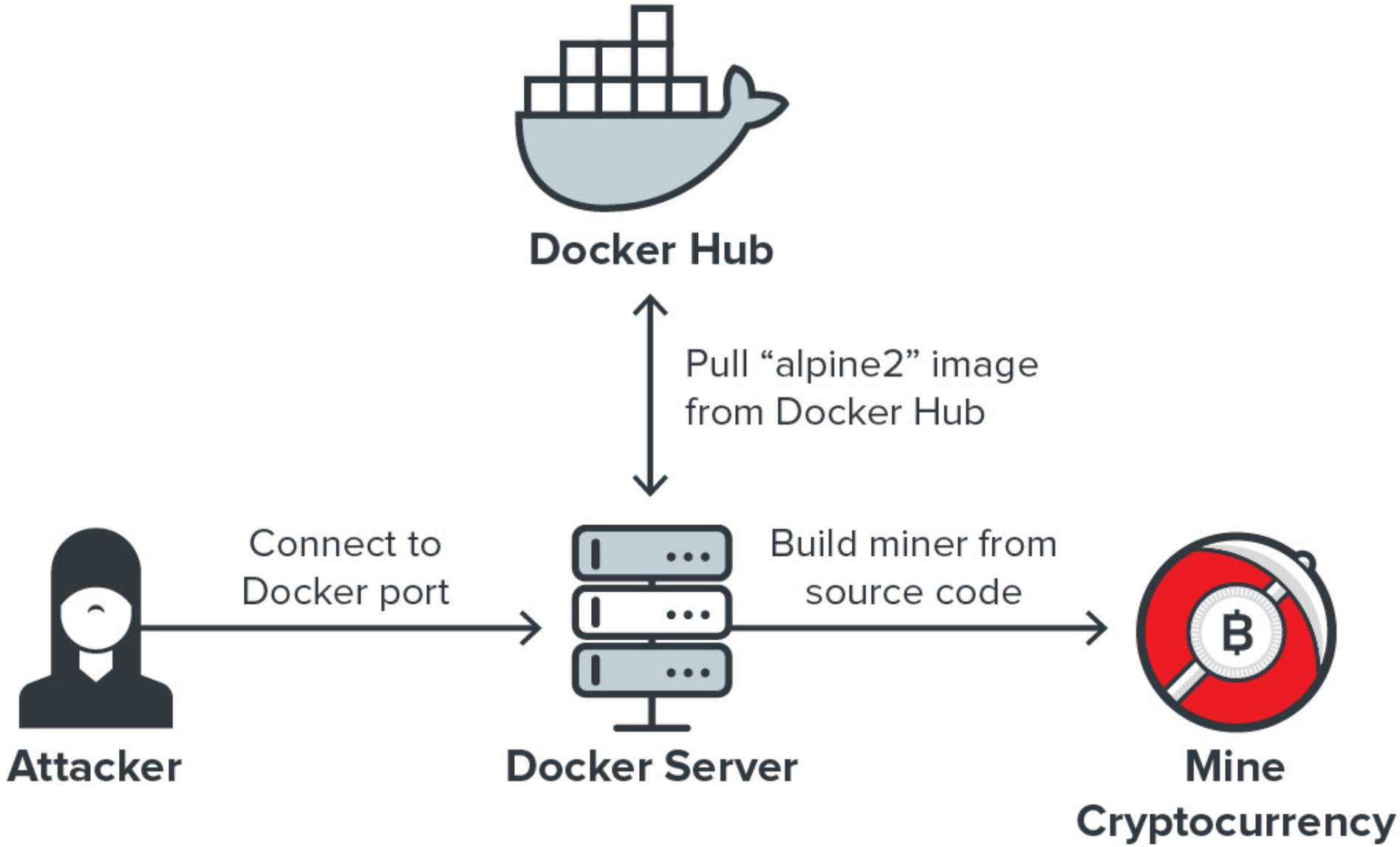
https://www.youtube.com/watch?v=kd33UVZhnAA&t=37s

Container and Image security

Public registries

- Can we trust them?
  - Yes, you can trust me 

"By pushing malicious images to a Docker Hub registry and pulling it from the victim's system, hackers were able to mine 544.74 Monero, which equals $90,000."

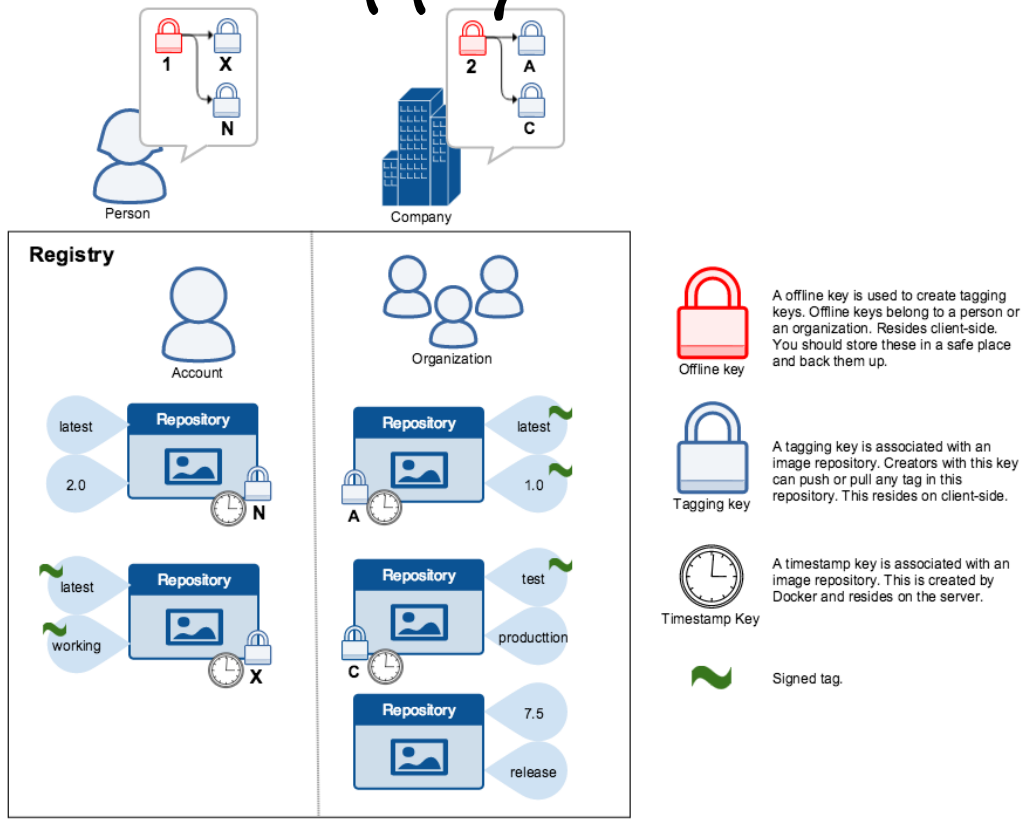


Build images best practices

- Multi-stage build
  - Drop build tools/libs from the final image
- Run as a non-root user
  - Don't run as root just because it's easy
  - I need port below 1024 that's why I need root
- Old images like 'Ubuntu:12.04'
- tag images
  - Are tags mutable or immutable?
- source code at the last step (use cache)

Sign and validate images

- Image scanning included in the supply chain
  - Trivy
- Sign image
  - Gnu Privacy Guard (GPG)
  - Docker Content Trust (DCT)
- Timestamp key



Runtime security

Open container Initiative

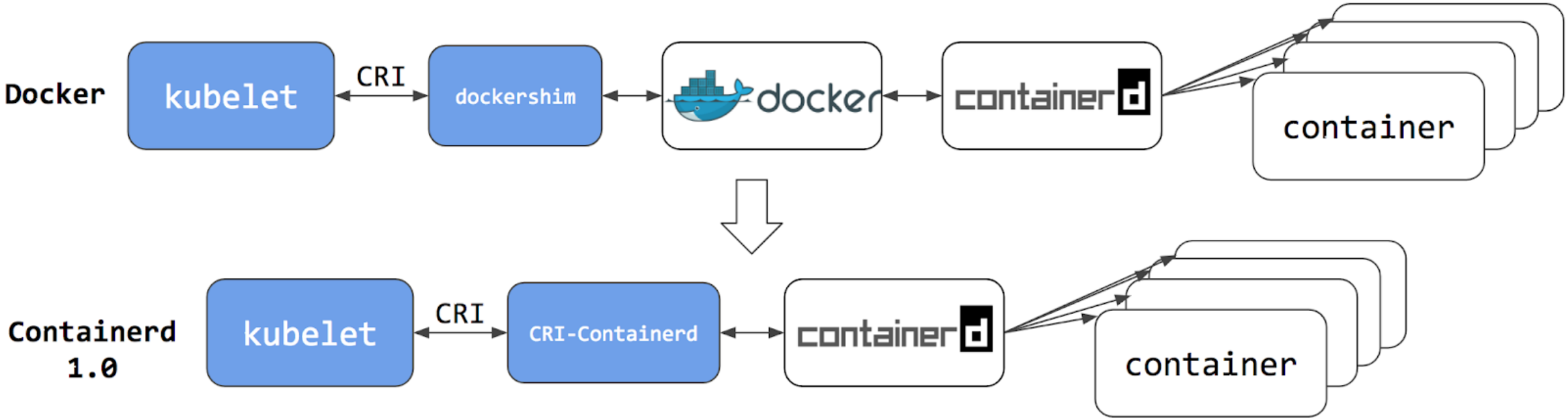
- Runtime specification
  - Specify execution env and container lifecycle
- Image specification
  - Download, unpack and run it specification
- Distribution specification
  - Standardize container image distribution based on Dock Registry HTTP API v2

Lets run our image

- Container Runtime Interface (CRI)
- OCI runners - low level container exec
  - runc - written in Go (Docker is using it)
  - crun - redhat written in C
- CRI shims - pull and runs images per OCI
  - cri-dockerd
  - containerd
  - cri-o
- Container Engines
  - Docker - all in one building, publishing, running ...

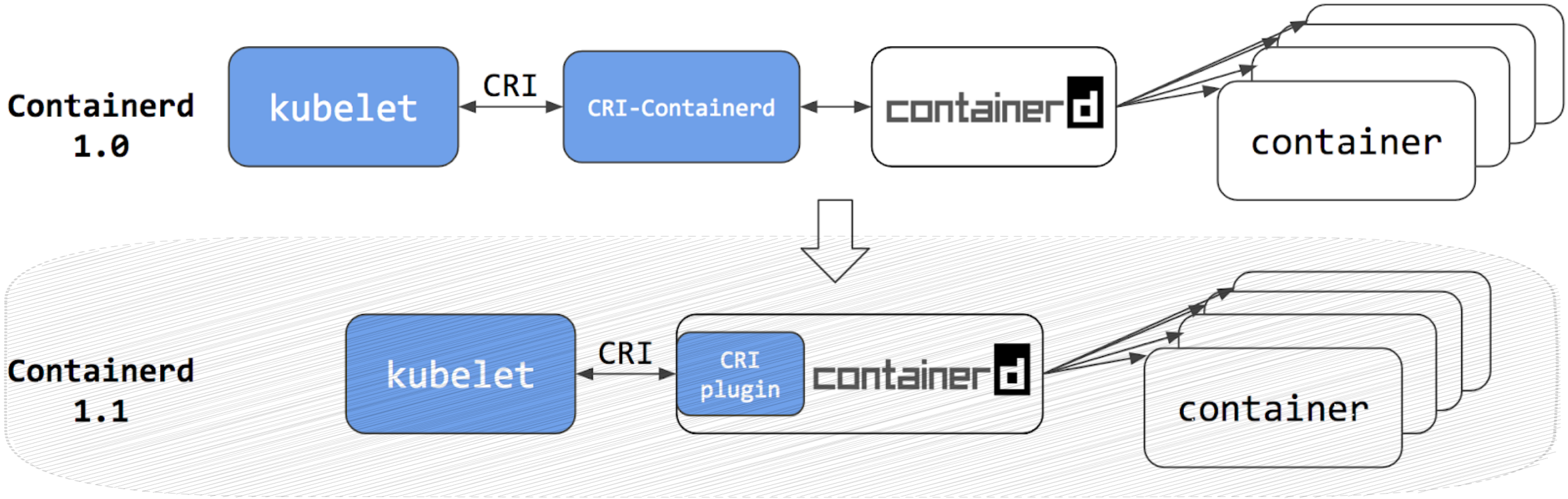
Lets run our image

- Container Runtime Interface (CRI)



Lets run our image

- Container Runtime Interface (CRI)

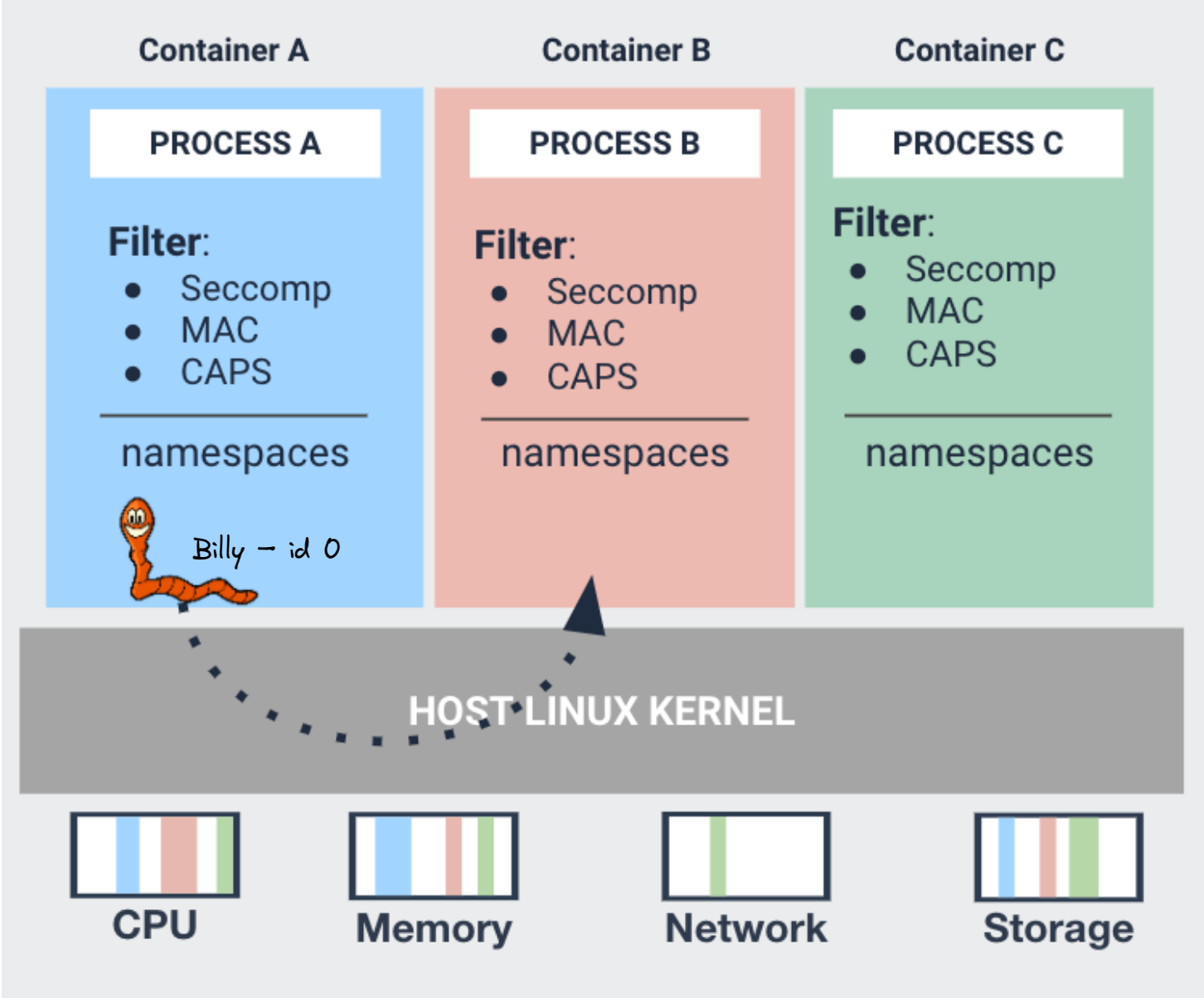


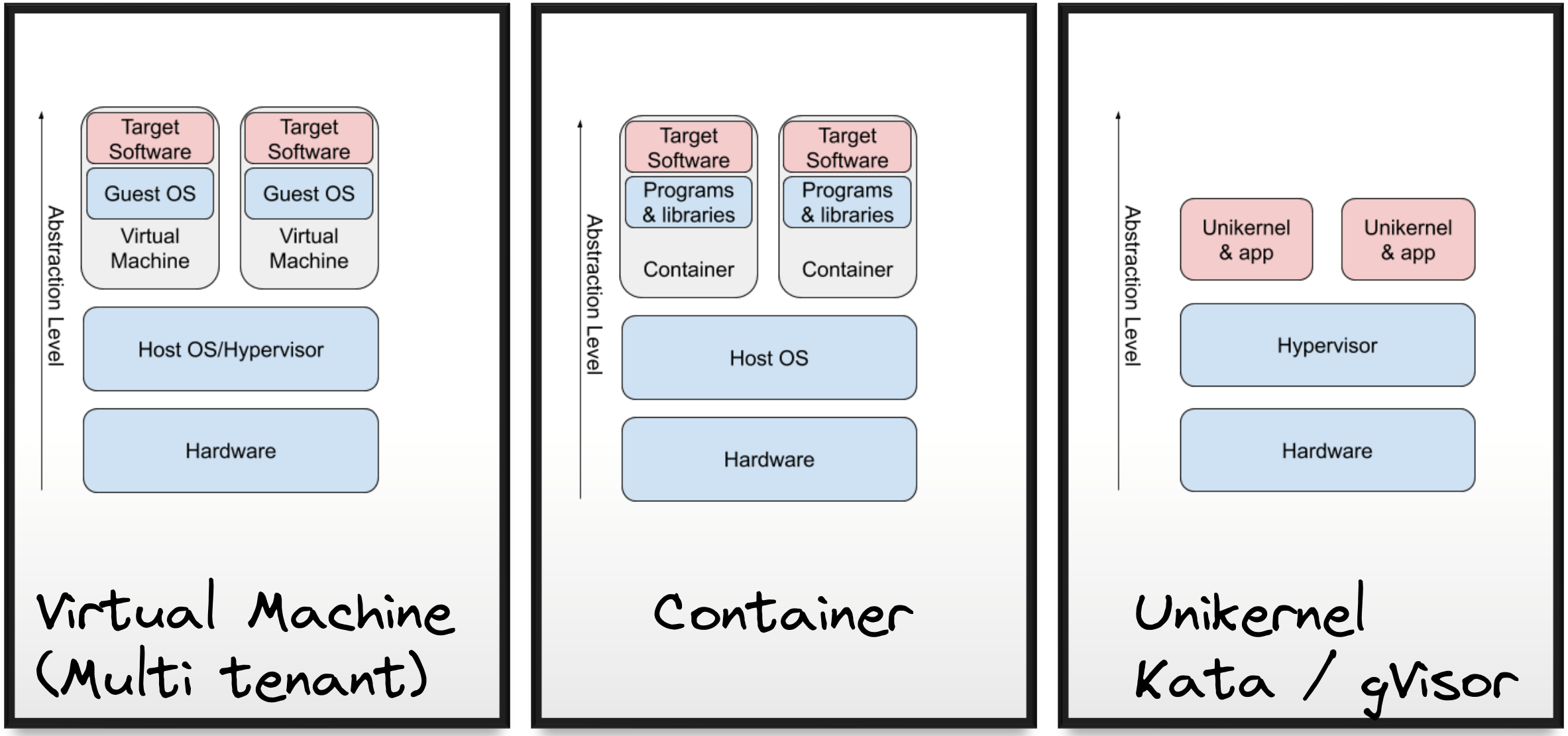
The OCI is the standard not Docker

Docker is OCI compliant

Container vs VM vs Lightvisor

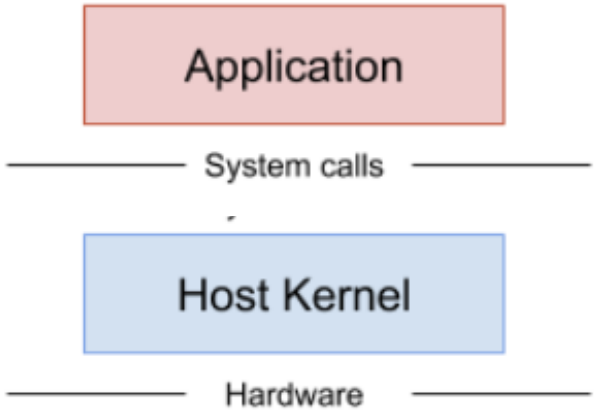
- Linux containers – There is NO such thing as a 'container'
- Known the limits of the namespace isolation
- Do not run different customers on the same physical host


Container A
Container B
Container C
PROCESS A
PROCESS B
PROCESS C
Filter:
Seccomp
MAC
CAPS
namespaces
Billy – id 0
HOST LINUX KERNEL
CPU
Memory
Network
Storage

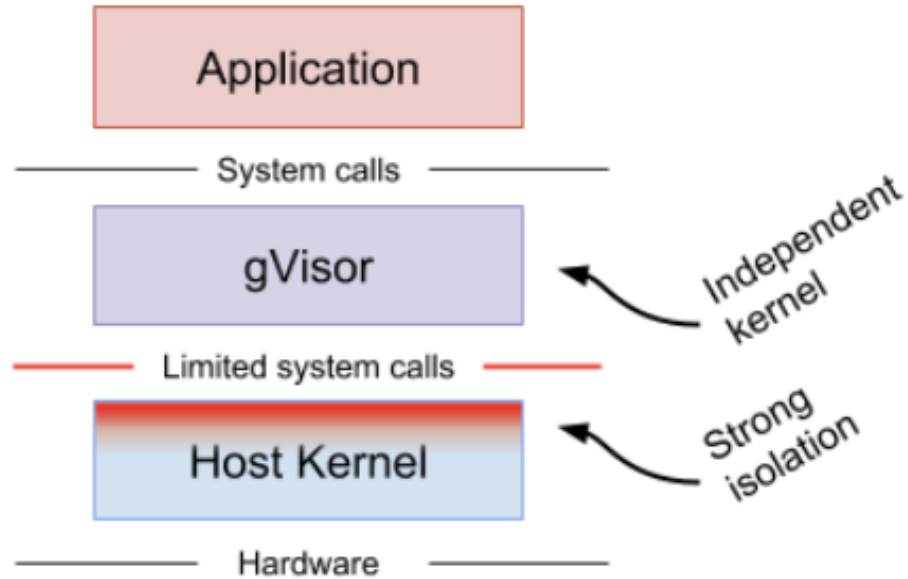

Abstraction Level
Target Software
Guest OS
Virtual Machine
Target Software
Guest OS
Virtual Machine
Host OS/Hypervisor
Hardware
Virtual Machine (Multi tenant)
Abstraction Level
Target Software
Programs & libraries
Container
Target Software
Programs & libraries
Container
Host OS
Hardware
Container
Abstraction Level
Unikernel & app
Unikernel & app
Hypervisor
Hardware
Unikernel Kata / gVisor

```
~$ sudo crictl inspect c86389dc7fa48 | grep -iA 20 namespaces"
"namespaces": [
            {
            "type": "ipc",
            "path": "/proc/10413/ns/ipc"
            }, {
            "path": "/proc/10413/ns/uts" },
            {
            "path": "/proc/10413/ns/net" }
            ],
$ ps -ef | grep 10413
nobody 10413 10375 0 10:06 ? 00:00:00 runsc-sandbox --
root=/run/containerd/runsc/k8s.io --
log=/run/containerd/io.containerd.runtime.v2.task/k8s.io/63df904cf22f2644adf5
a79c69631f80aabe54aa9f53ae60351687114fce416a/log.json --log-format=json --
log-fd=3 boot --
bundle=/run/containerd/io.containerd.runtime.v2.task/k8s.io/63df904cf22f2644a
df5a79c69631f80aabe54aa9f53ae60351687114fce416a -- controller-fd=4 --
mounts-fd=5 --spec-fd=6 --start-sync-fd=7 --io-fds=8 --io-fds=9 --stdio-fds=10
--stdio-fds=11 --stdio-fds=12 -- cpu-num 2 --user-log-fd 13
63df904cf22f2644adf5a79c69631f80aabe54aa9f53ae60351687114fce416a

nobody 10484 10413 0 10:06 ? 00:00:00 [exe]
```



```
~$ sudo crictl inspect ef947d7856c12 | grep -iA 20 namespaces
"namespaces": [ {
            {
            "type": "ipc",
            "path": "/proc/9665/ns/ipc"
            }, {
            "type": "uts",
            "path": "/proc/9665/ns/uts" },
            },
            {
            "type": "network",
            "path": "/proc/9665/ns/net" }
            ],
user@worker-node:~$ ps -ef | grep 9665
root 9665 1 0 09:25 ? 00:00:00 /usr/bin/containerd-shim-kata-v2 -namespace
k8s.io -address /run/containerd/containerd.sock -publish-binary
/usr/bin/containerd -id  a808b7b498fc088e228215fe4c4e6f74d40
```

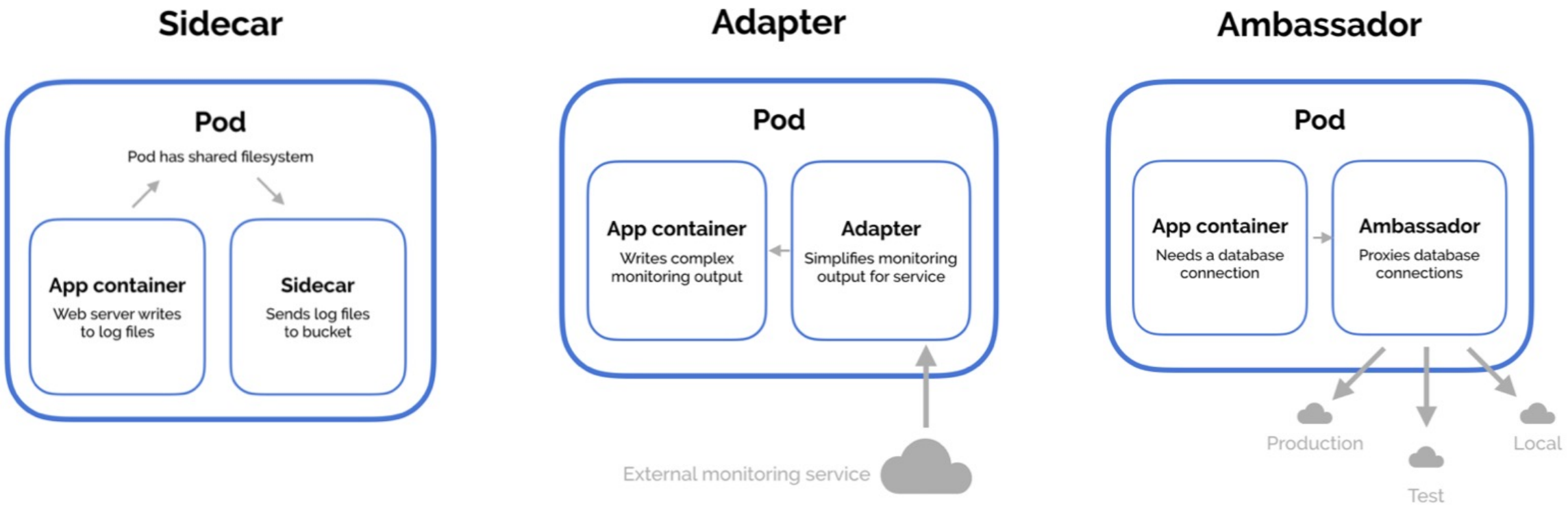


Kubernetes pod security

Cloud security principles

- Zero trust - AutheNtication
  - Who are you? (I'm Billy)
  - Everything must be authN
- Least privileges - AuthoriZation
  - Hey Billy let me check what you can do
  - AuthZ for the fewest privileges
  - Use init containers running for short amount of time
- Anti patterns:
  - I don't need verification if it comes from inside

Multi container pod



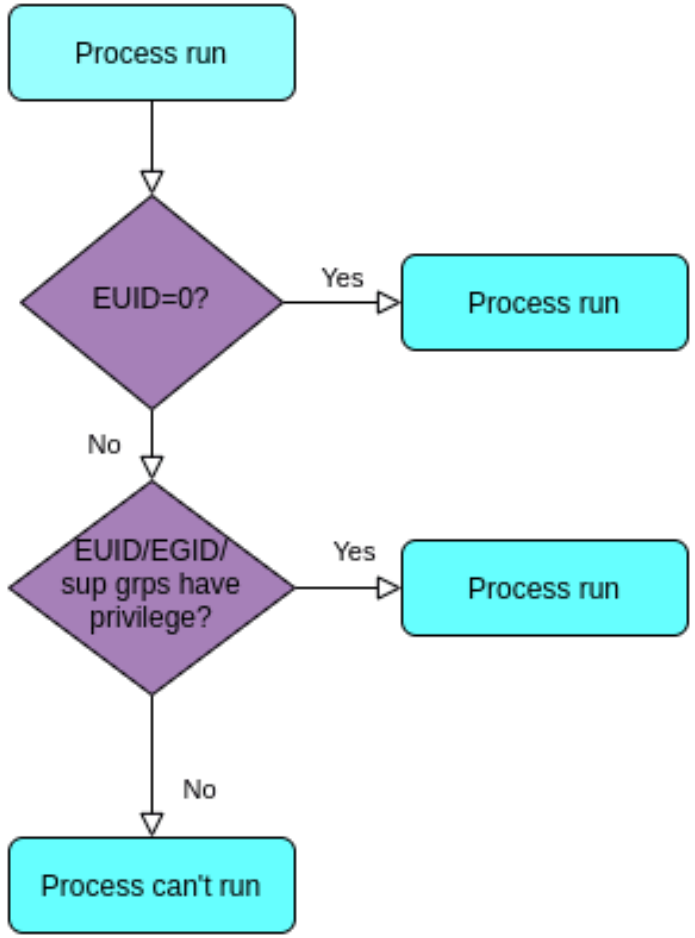
Capabilities

```
root# grep Cap /proc/self/status
CapInh: 0000000000000000
CapPrm: 000000 1fffffffff
CapEff: 000000 1fffffffff
CapBnd: 000000 1fffffffff
cncf$ grep Cap /proc/self/status
CapInh: 0000000000000000
CapPrm: 0000000000000000
CapEff: 0000000000000000
CapBnd: 000000 1fffffffff
```

Capabilities

```
root# man capabilities
- privileged processes - user ID is 0
- unprivileged processes are subject to full permission checking based on the process's credentials

Example:
CAP_CHOWN
CAP_NET_ADMIN
CAP_SYS_TIME
```



More security methods

- SELinux – Linux kernel security module
  - Control access to the application
- AppArmor - Linux kernel security module
  - Applied to a Pod by specifying an AppArmor profile
- SecComp - security facility
  - Restrict System calls like a firewall

Kubernetes network security

Network policies

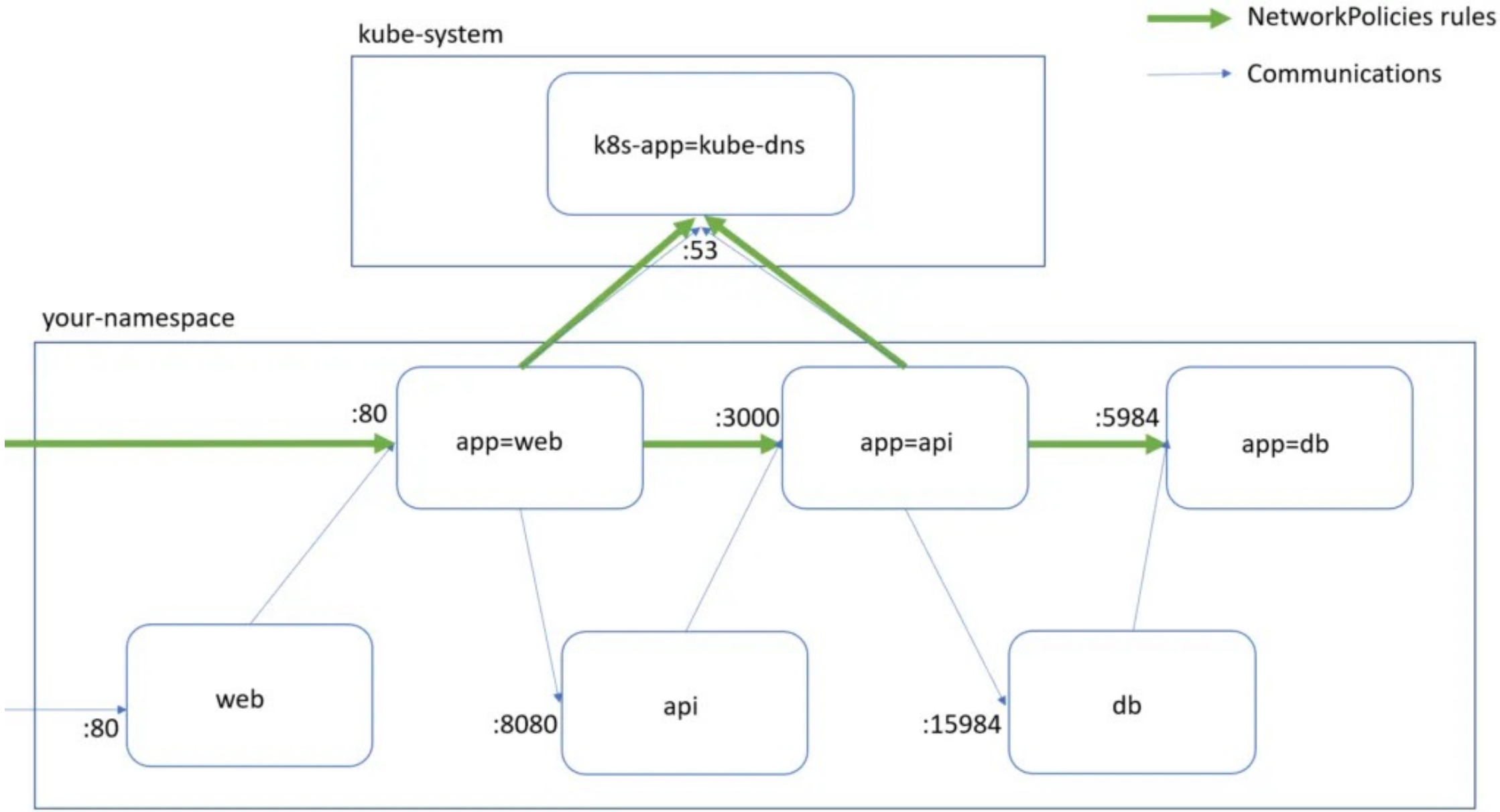
Kubernetes Network Model

- Pods can communicate with all other pods on any other node without NAT
- Agents on a node can communicate with all pods on that node

- Control traffic flow at the IP address or port level (OSI layer 3 or 4)

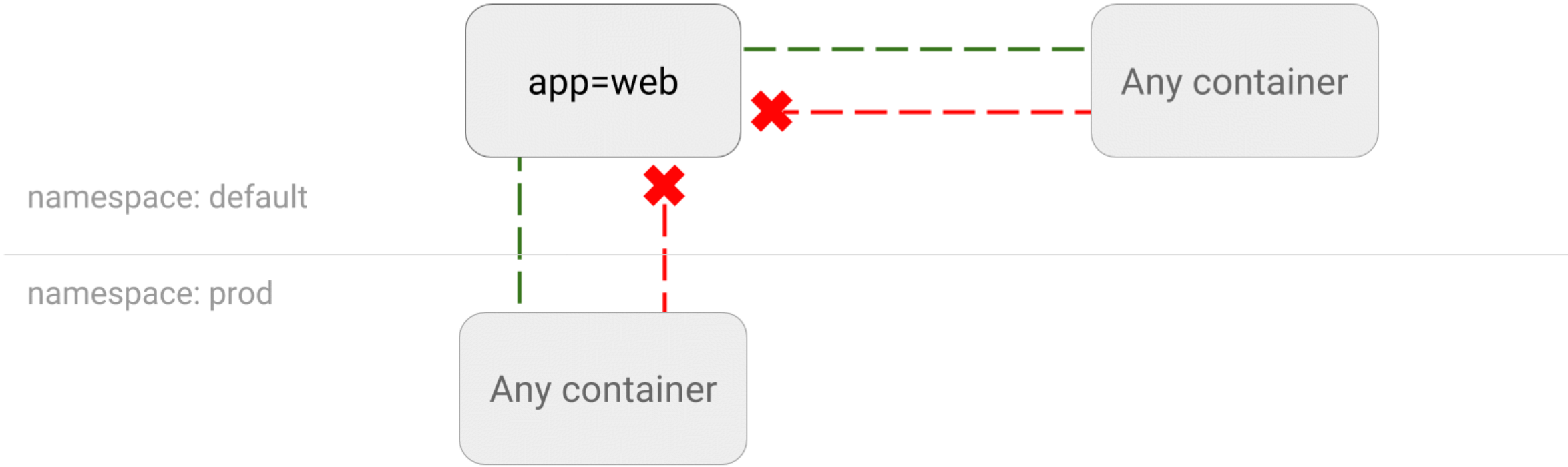
Network policies

- Container Network Interface Plugins:
  - Canal
  - Calico - popular
  - Weave Net - easy to install
  - Cilium - New generation CNI leverage eBPF
  - VMware NSX
  - Flannel - Do not support Network Policies


NetworkPolicies rules
Communications
kube-system
k8s-app=kube-dns
:53
your-namespace
:80
app=web
:3000
app=api
:5984
app=db
web
:80
api
:8080
db
:15984

Network policies restrictions

Control Ingress and Egress traffic

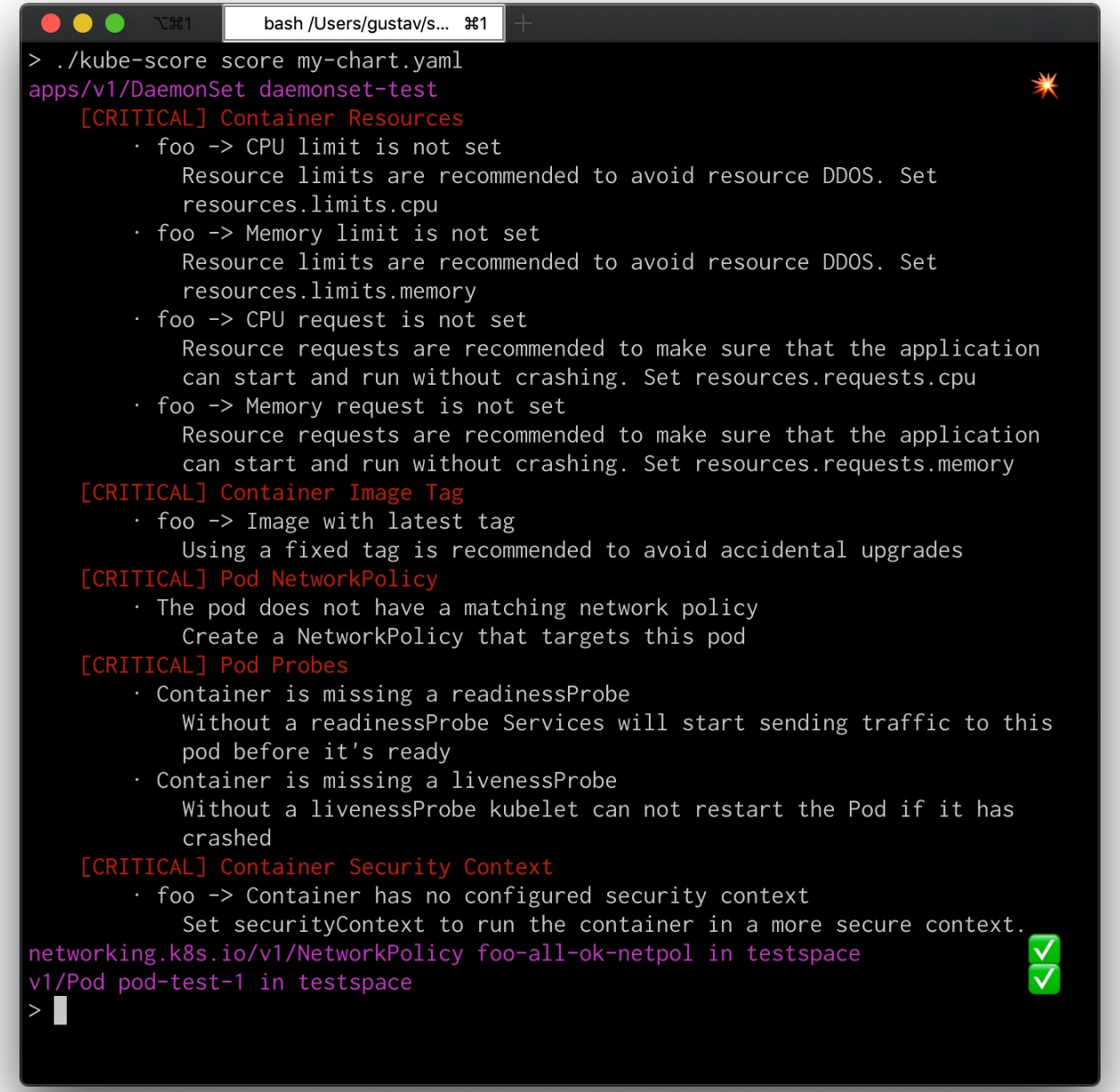


Behavioral Analysis

Static Analysis

- Analyze your Infra as code
- Scan manifests for bad practices like:
  - Miss configuration
  - Escalation privileges
  - Hardcoded secrets (tokens, keys, creds...)
- Your computer editor > pre-commit > CICD
  - Linters - PyLint
  - Formatters - Black
  - Tools - pre-commit

- Kube-Score
  - Implement it in your CICD
- Analyze manifests
  - Static YAML
  - Helm charts
  - Existing k8s resources



Running inside the cluster

- Do not touch it if it works
- I checked the image is NOT vulnerable
  - When ?
  - [1.021462] Kernel panic()

- Did you check your currently running images in your cluster ?
  - Remember security is tight with time


Monitoring
Logging
Observability
Visualization
Tracing

Observability usage

- Potential threat detection
  - Honeypots
- Behavior analysis
  - What an attacker is doing

- MITRE ATT & CK Matrix
  - Table of known attack techniques and behaviors documented by the enterprise security community

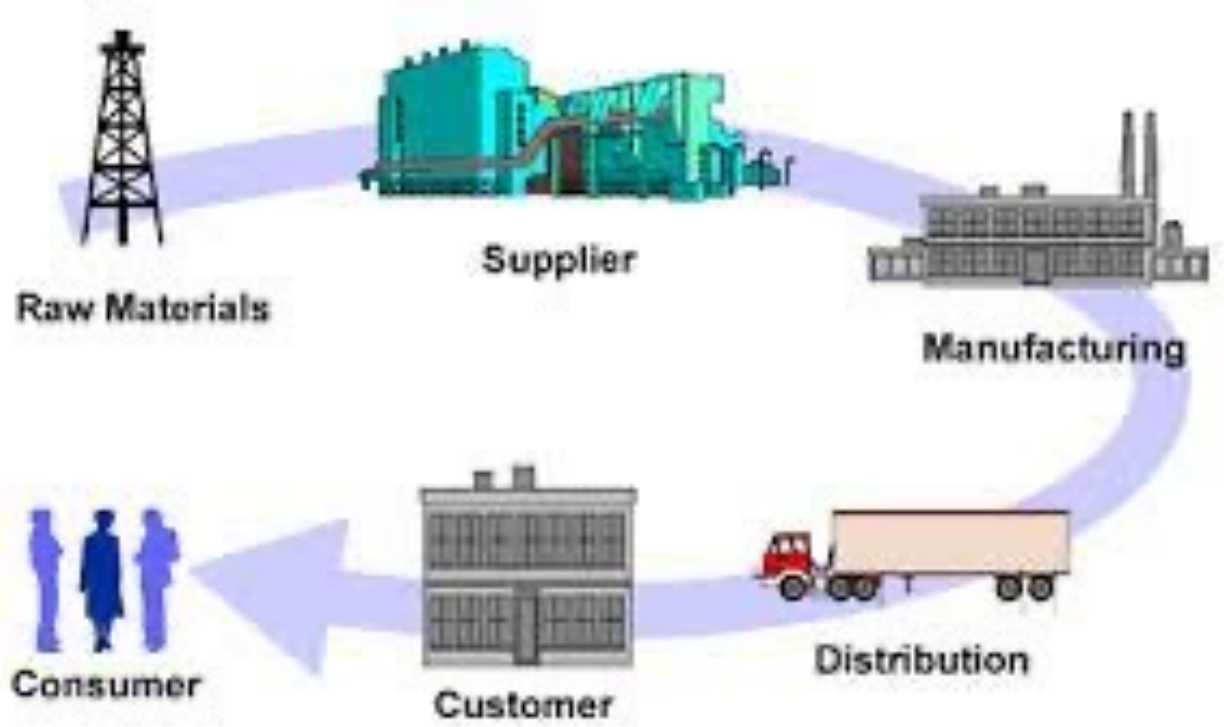
Observability usage

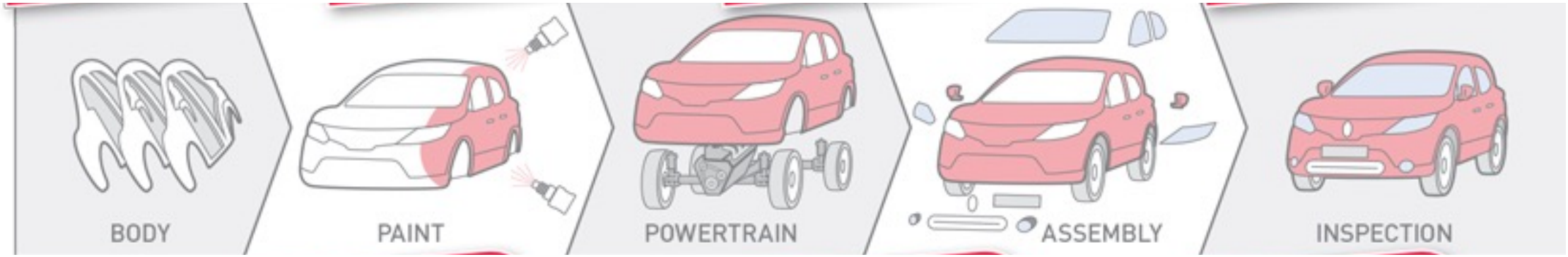
- STRIDE
  - Spoofing
  - Tampering
  - Repudiation
  - Information disclosure (privacy breach or data leak)
  - Denial of service
  - Elevation of privilege

Supply Chain Security

Supply chain

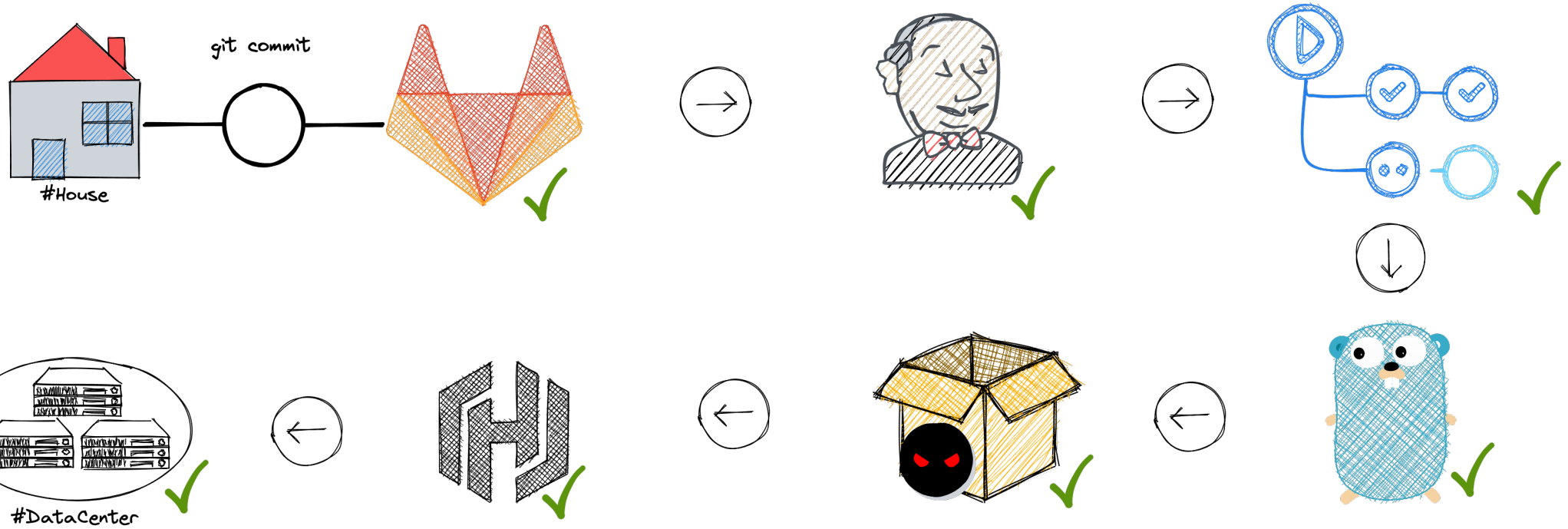
What is actually a supply chain ?





Supply chain tools

- You use 'Trivy' to scan your images
  - Guest what we hacked 'Tviry'
  - Guest what now our malware is inside the image and you don't know because 'Tviry' can't catch it
  - Use multiple scanners bypass if 3 of 5 are meeting your policies


git commit
#House
#DataCenter
Download update
Upgrade
Customer private net

It's not a job it's a passion

Q&A


01000001 01101110 01111001 00100000
01110001 01110101 01100101 01110011
01110100 01101001 01101111 01101110 01110011
00111111

Sources

- Kata containers: https://thenewstack.io/the-road-to-kata-containers-2-0/
- Alpine malware miner - https://www.trendmicro.com/vinfo/fr/security/news/virtualization-and-cloud/malicious-docker-hub-container-images-cryptocurrency-mining
- Monero mining: https://unit42.paloaltonetworks.com/cryptojacking-docker-images-for-mining-monero/
- Google 6 layers of security
- https://www.youtube.com/watch?v=kd33UVZhnAA
- sidecar: https://matthewpalmer.net/kubernetes-app-developer/articles/multi-container-pod-design-patterns.html
- network policies: https://cloudogu.com/en/blog/k8s-app-ops-part-1, https://github.com/ahmetb/kubernetes-network-policy-recipes , https://cloudblogs.microsoft.com/opensource/2019/10/17/tutorial-calico-network-policies-with-azure-kubernetes-service/
- CNI plugins: https://github.com/containernetworking/cni#3rd-party-plugins
- Observability: https://linkedin.github.io/school-of-sre/level101/metrics_and_monitoring/observability/