MANICODE

SECURE CODING EDUCATION

Cloud-Native Security

A little background dirt…

@jimmesta

- 10 years of penetration testing, teaching, and building security programs
- OWASP AppSec California organizer and Santa Barbara chapter founder
- Conference speaker
- Been on both sides of the InfoSec fence
- Loves Clouds

Introduction to Cloud-Native

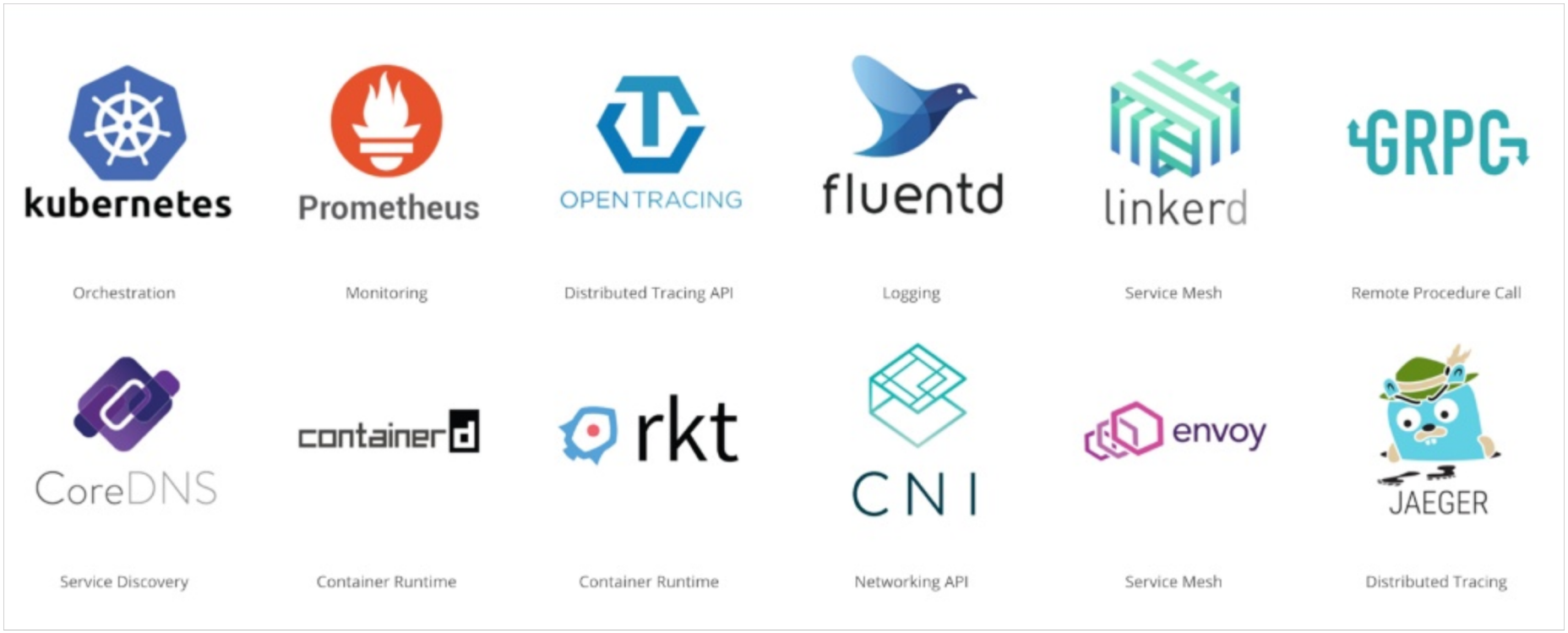
CLOUD NATIVE COMPUTING FOUNDATION

Create and drive the adoption of a new computing paradigm that is **optimized for modern distributed systems environments** capable of scaling to tens of thousands of self healing multi-tenant nodes.

Fast · Open · Fair

- Provide stewardship for projects
- Foster growth and evolution of ecosystems
- Promote of the underlying technologies
- Make the technology accessible and reliable

CNCF Projects

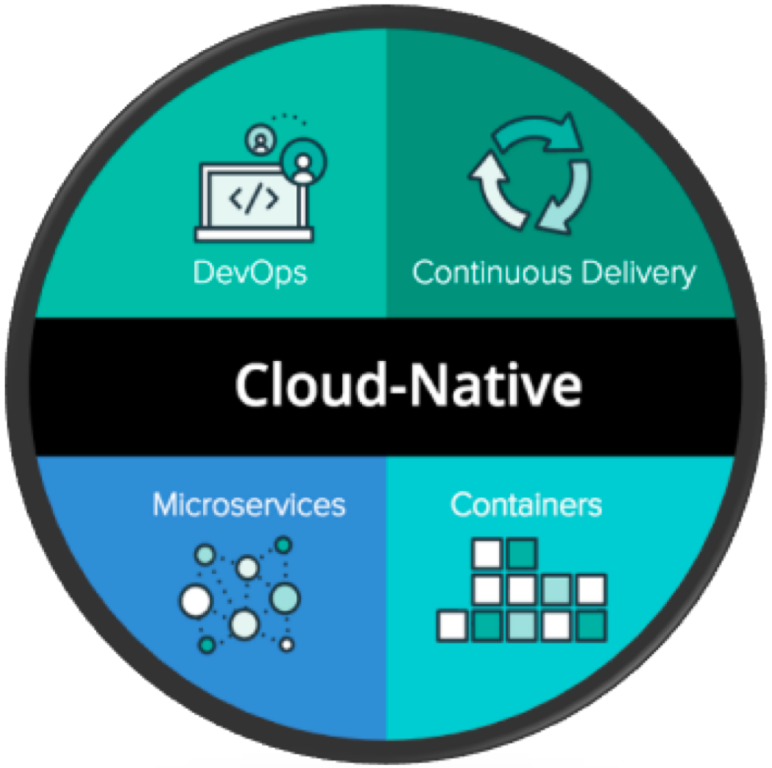


CNCF Working Groups

Continuous Integration	Networking	Storage	Serverless
Provides infrastructure to hosted projects. Looks to offer integration testing between projects.	Providing a Container Networking Interface (**CNI**) specification. Aims for connectivity and portability in cloud native application networking.	Providing a Container Storage Interface (**CSI**) specification. Aims for portability across cloud orchestration systems.	Educate cloud native developers on serverless architectures. Determine what the CNCF should do in this space. Recommend involvement in specifications and projects.

Cloud Native Overview

- Microservice-centric
- CI/CD Support
- Portable
- Infrastructure as Code
- Monitoring and Logging
- IaaS / PaaS



Cloud Native Security Challenges

- New stuff, new problems
- Network and infrastructure security still matter
- Microservices add networking, authZ / authN complexity
- Attack detection models change drastically
- New tooling and mindset
- Automation takes upfront work

Introduction to Serverless Security

The Promise…

Serverless architecture

High-availability

Event-driven

Zero administration

Serverless Overview

- **Servers go away?!**
  - Kind of but no…we are offloading server admin to a cloud provider.
- **Ops go away?!**
  - Not really…we still do networking and sysadmin.
- **Vulnerabilities go away?!**
  - Definitely not.

- **Then why are we doing this?!**

¯_(ツ)_/¯

Serverless Top Ten

SAS-1: Function Event Data Injection

SAS-2: Broken Authentication

SAS-3: Insecure Serverless Deployment Configuration

SAS-4: Over-Privileged Function Permissions & Roles

SAS-5: Inadequate Function Monitoring and Logging

SAS-6: Insecure 3rd Party Dependencies

SAS-7: Insecure Application Secrets Storage

SAS-8: Denial of Service & Financial Resource Exhaustion

SAS-9: Serverless Function Execution Flow Manipulation

SAS-10: Improper Exception Handling and Verbose Error Messages

Hands-On Serverless Hacking

OWASP Serverless Goat

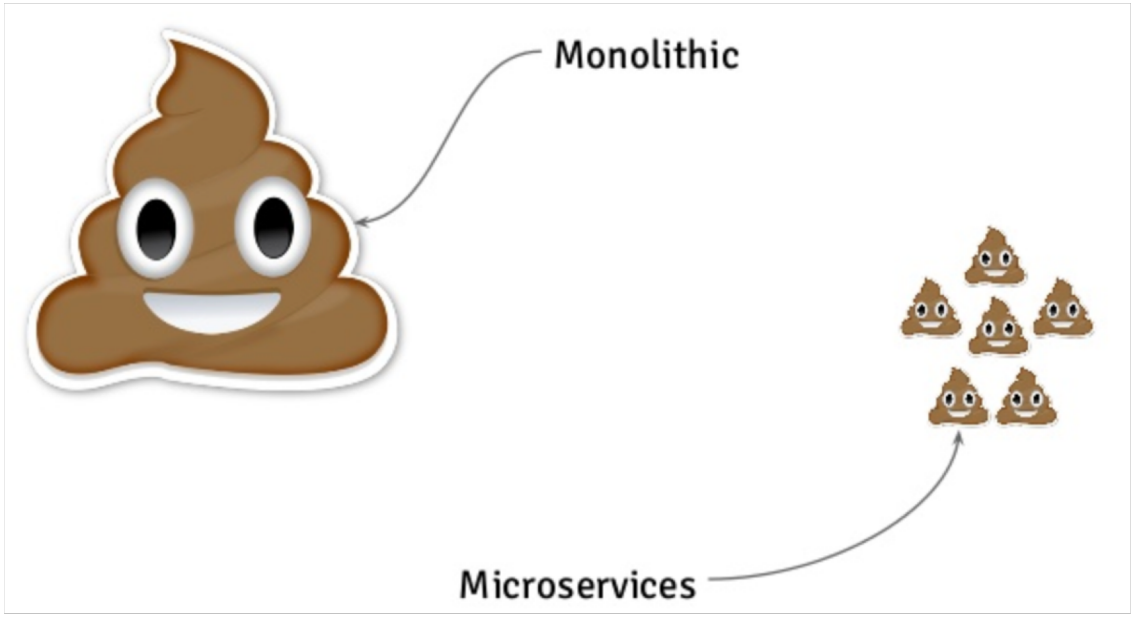
Introduction to Containers

Software Deployment is Changing

- Massive shift toward cloud computing
- Increased demand for application and infrastructure portability across environments
- Avoid vendor “lock in” when possible
- Increase in microservices AKA loosely coupled services

Modern Applications

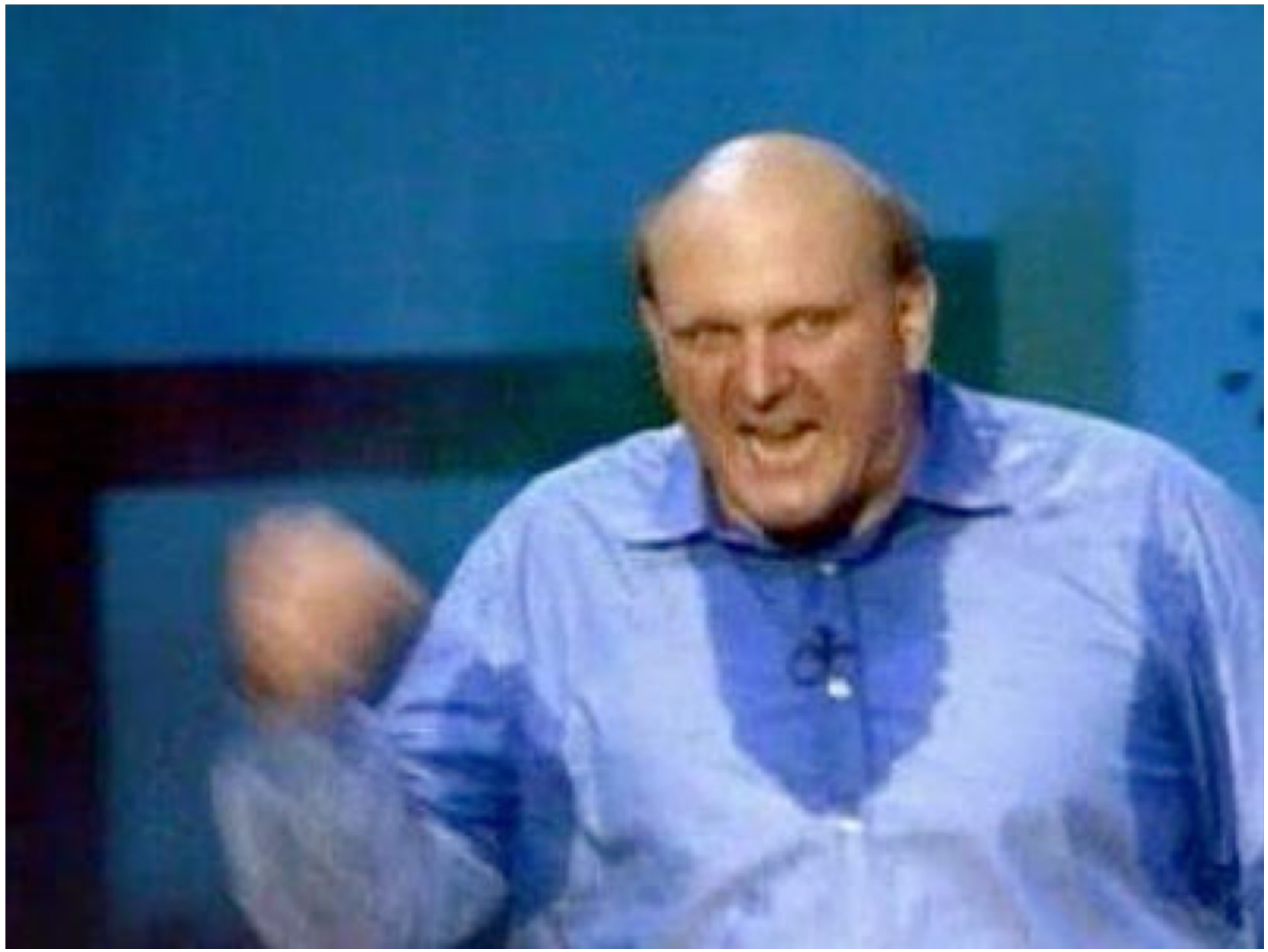
- Breaking monolithic applications into smaller services offers several advantages:
  - Scale independently
  - Stateless
  - High Availability
  - API-Driven
  - Faster iteration times

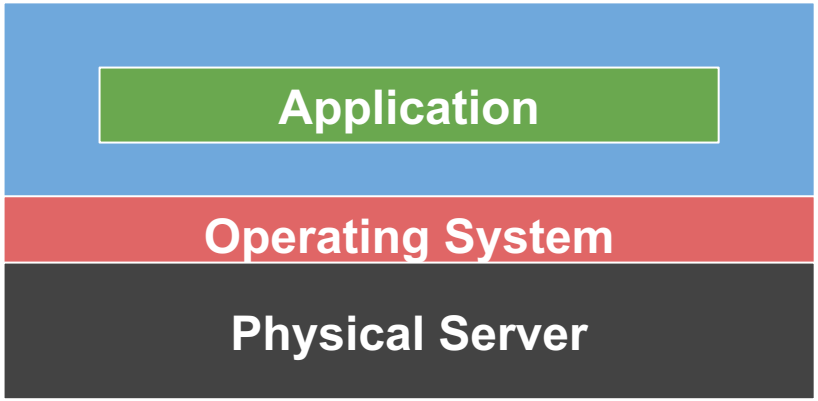


Issues with Modern Applications

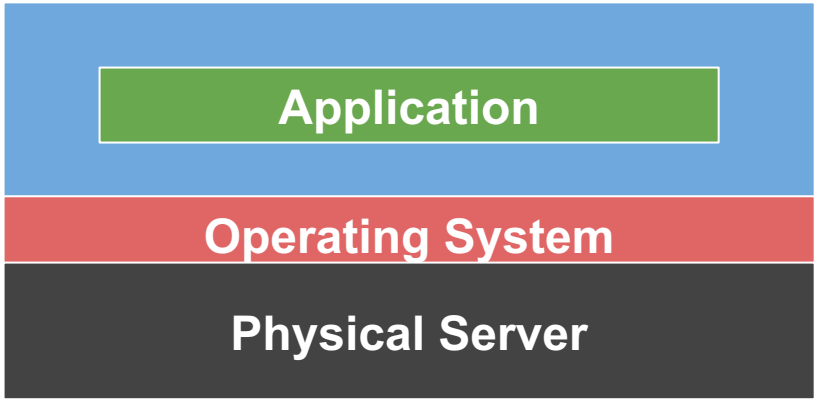
- Organizations often operate in an Ops vs. Dev vs. Sec world
- Applications and microservices are written in a variety of languages and frameworks
- Applications need to run on different technology stacks:
  - Virtual Machines
  - Windows Server
  - Bare Metal Servers
  - Cloud Environments
  - On-Prem Environments
  - Developer Laptops

Containers, Containers, Containers, Containers…

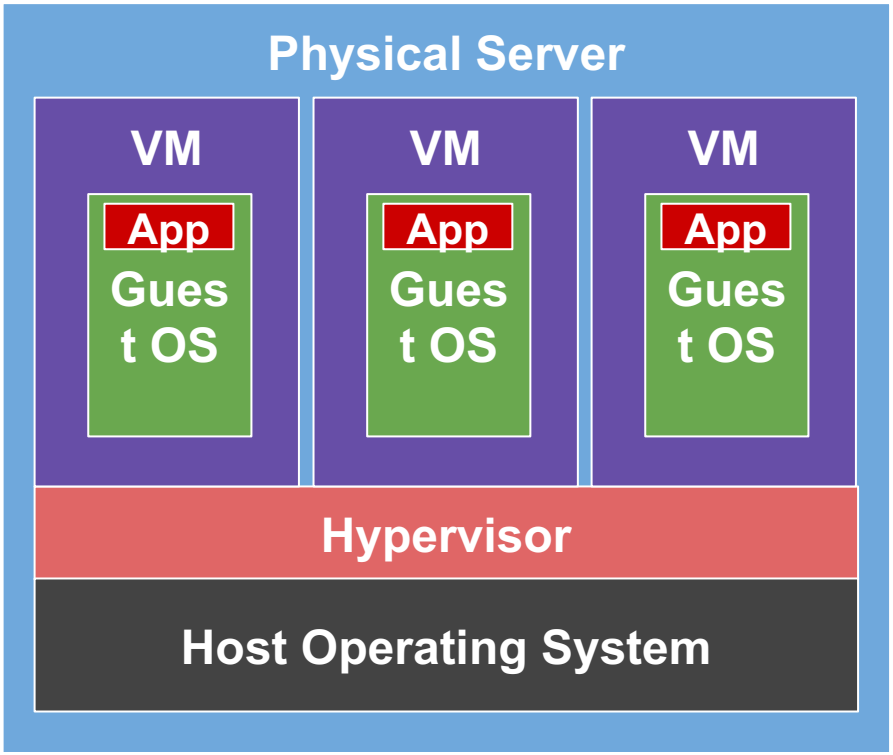




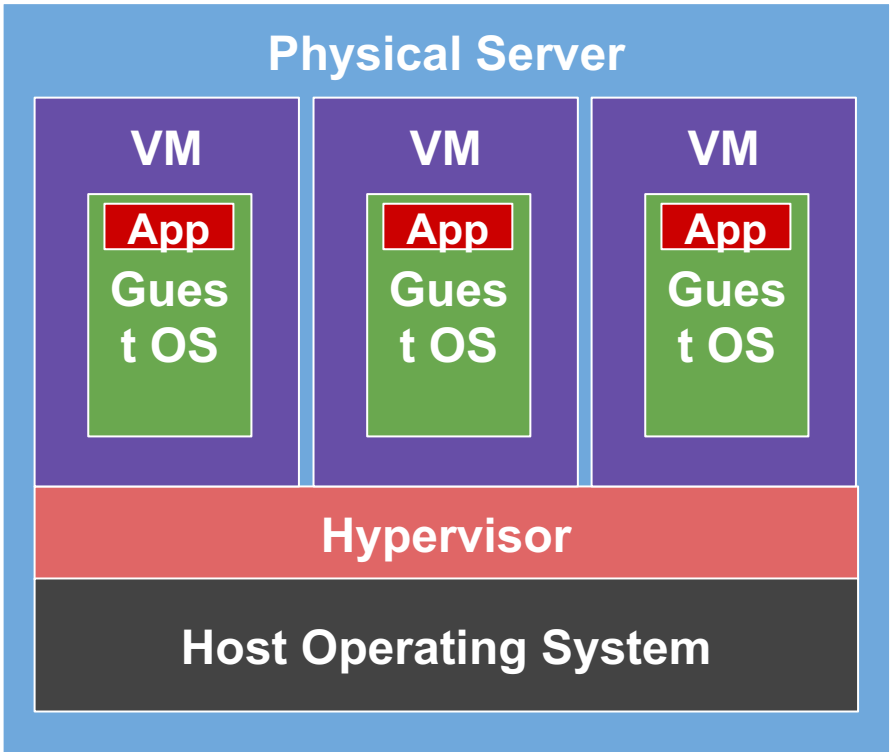
Physical Host



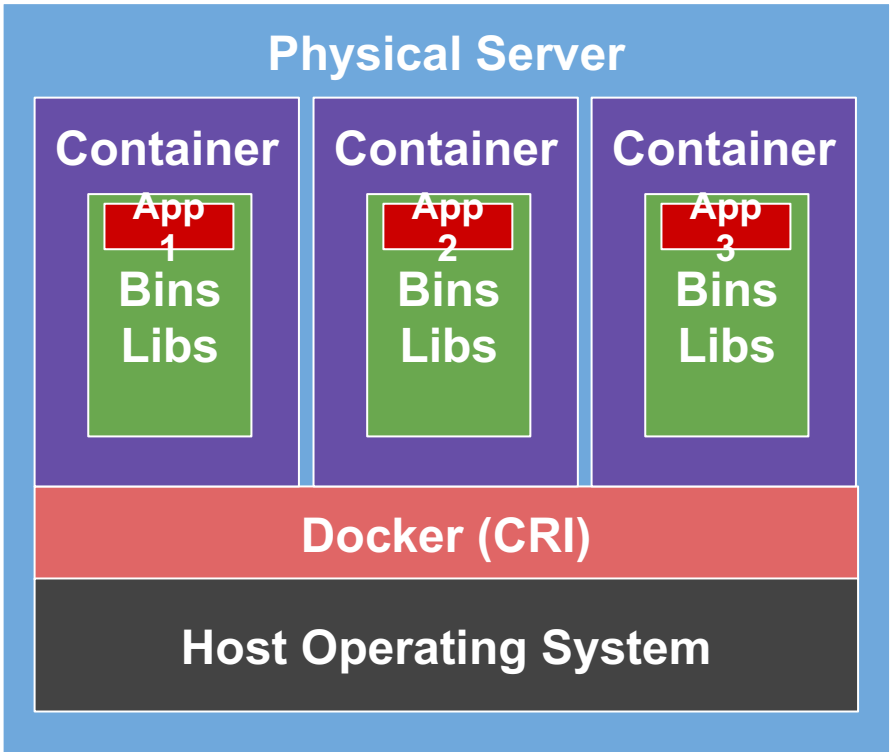
- One application per server
- Slow deployment times
- Low resource utilization
- Scaling challenges
- Migration challenges
- $$$
- Difficult to replicate locally



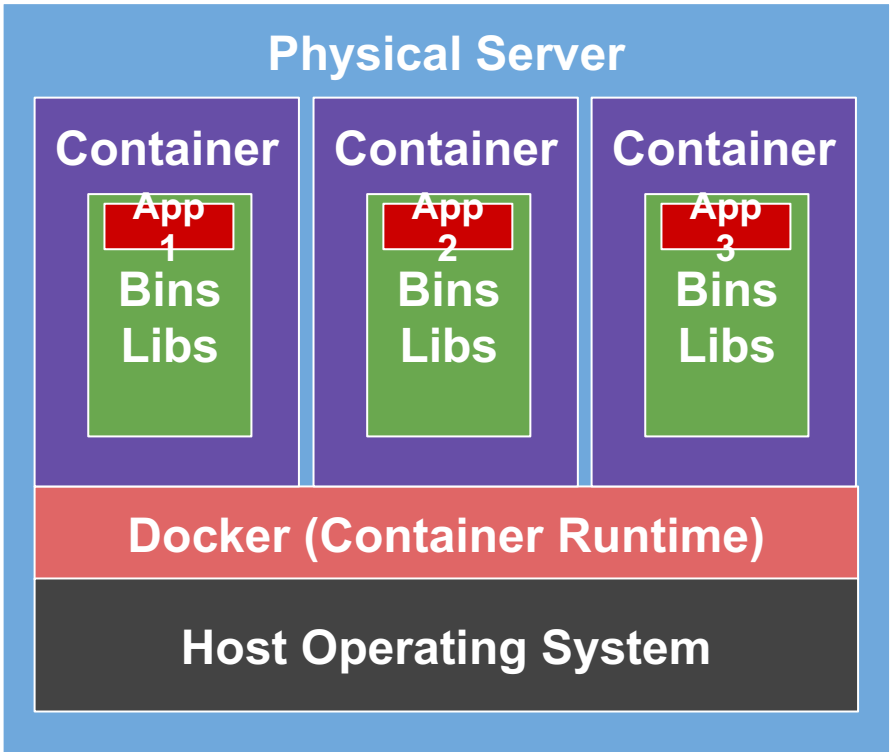
VM



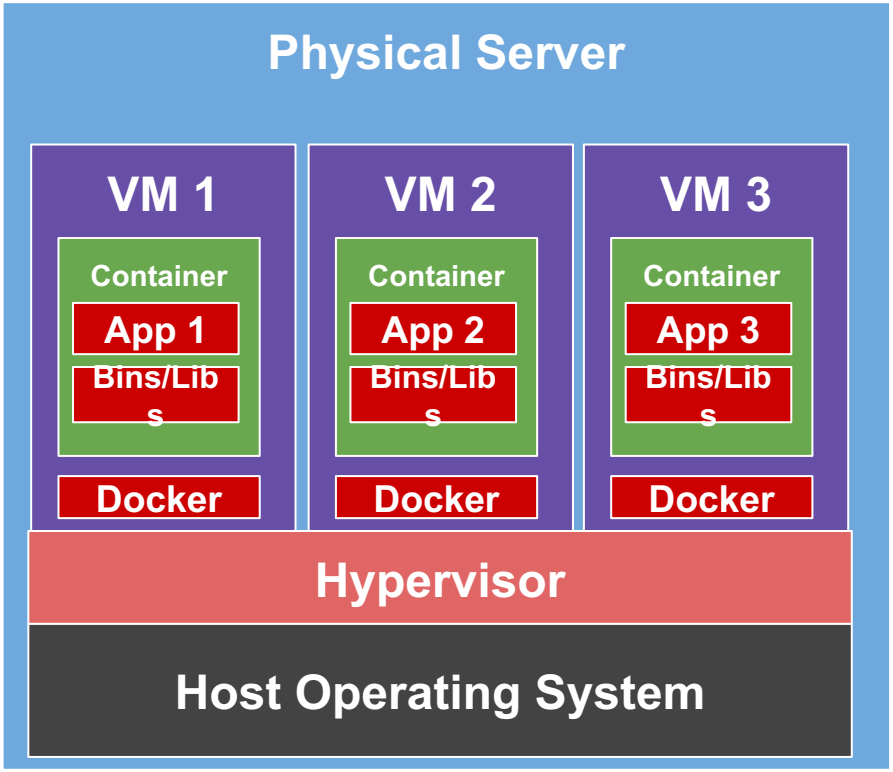
- One physical server and multiple applications
- Each application runs in a Virtual Machine
- Better resource utilization
- Easier to scale
- VMs live in the Cloud
- Still requires complete guest Operating Systems
- Application portability not guaranteed



Container



- Containers are an application layer construct
- VMs allow us to convert one physical machine into many servers
- No Operating System to boot (fast!)
- Most portable out of all options
- Less OS overhead using shared kernel model



Containers and VMs are Happy Together

Containers 101

Image

The basis of a Docker container. The content at rest.

Container

The image when it is 'running.' The standard unit for app service

Engine

The software that executes commands for containers. Networking and volumes are part of Engine. Can be clustered together.

Registry

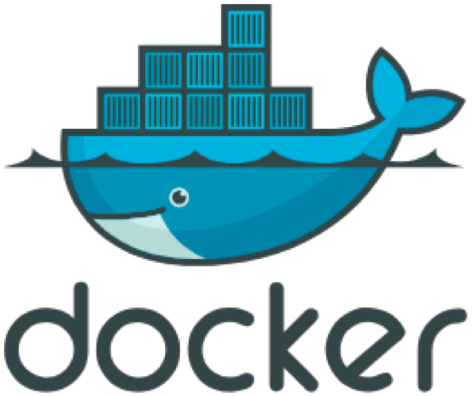
Stores, distributes and manages Docker images

Control Plane

Management plane for container and cluster orchestration

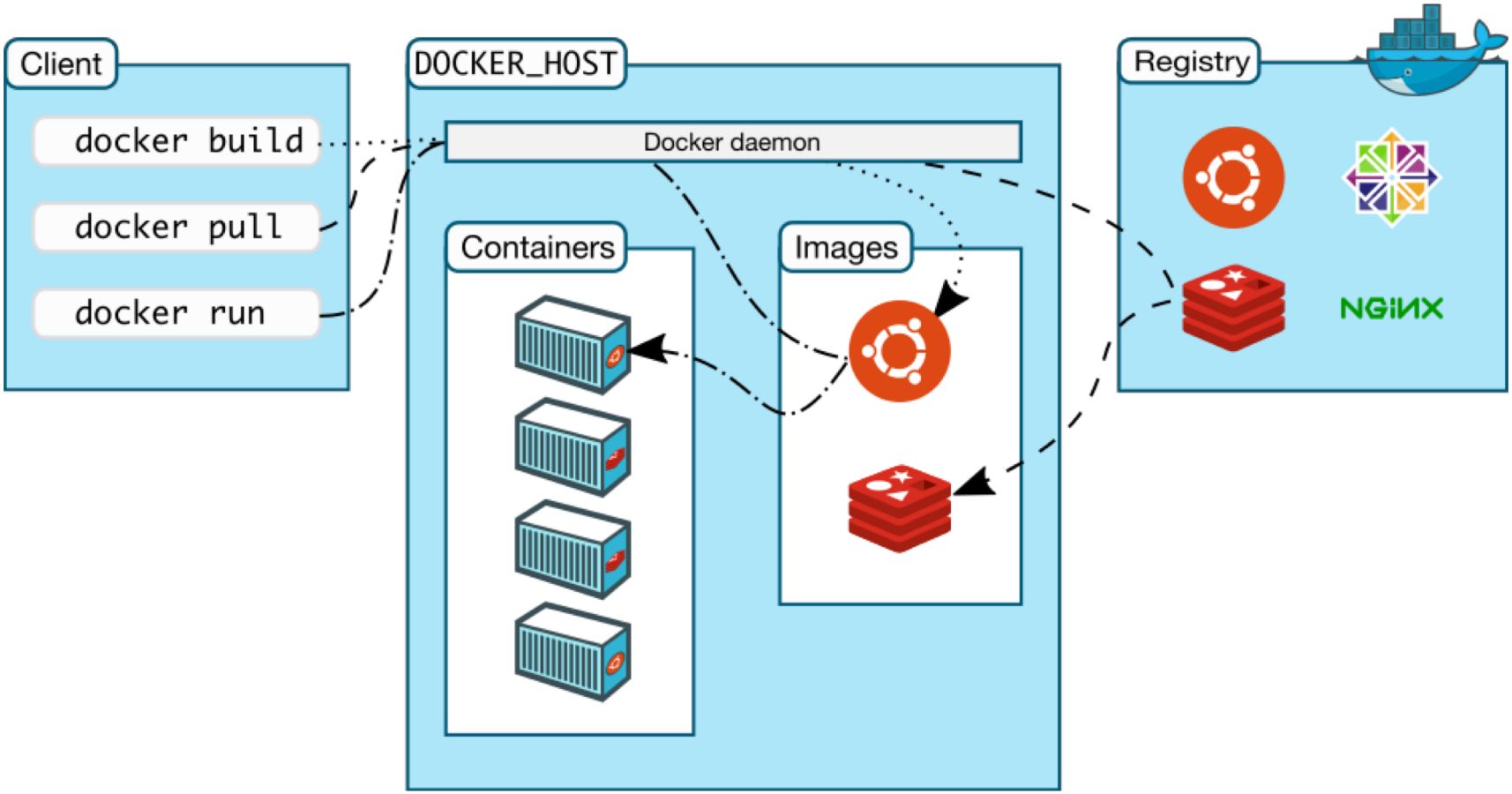
Docker Engine

Client-Server application that includes a few key components



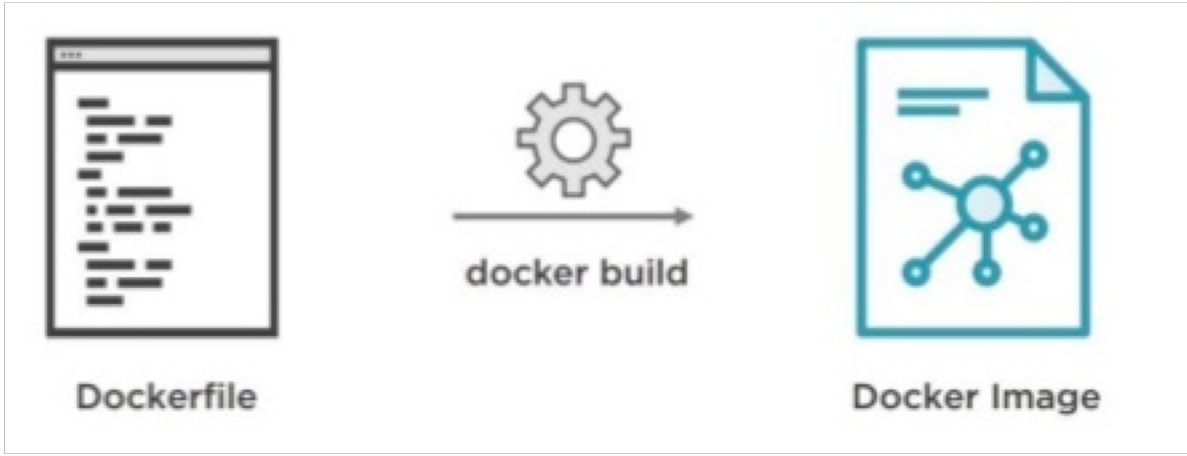
- **Docker Daemon** (dockerd)
  - Responsible for container orchestration
- **REST API**
  - Used to talk to the Docker daemon
- **Docker Client** (CLI)
  - Interface to interact with the Docker daemon

Docker Engine



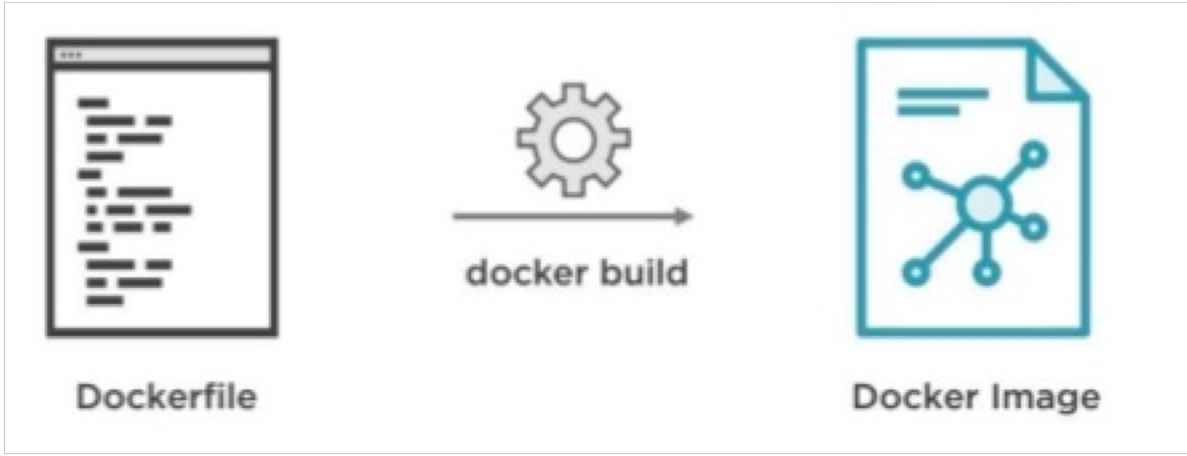
Dockerfile

- Text document that is used to build images
- Contains all of the commands that could be used in the CLI to assemble an image
- The docker build command creates the command-line instructions



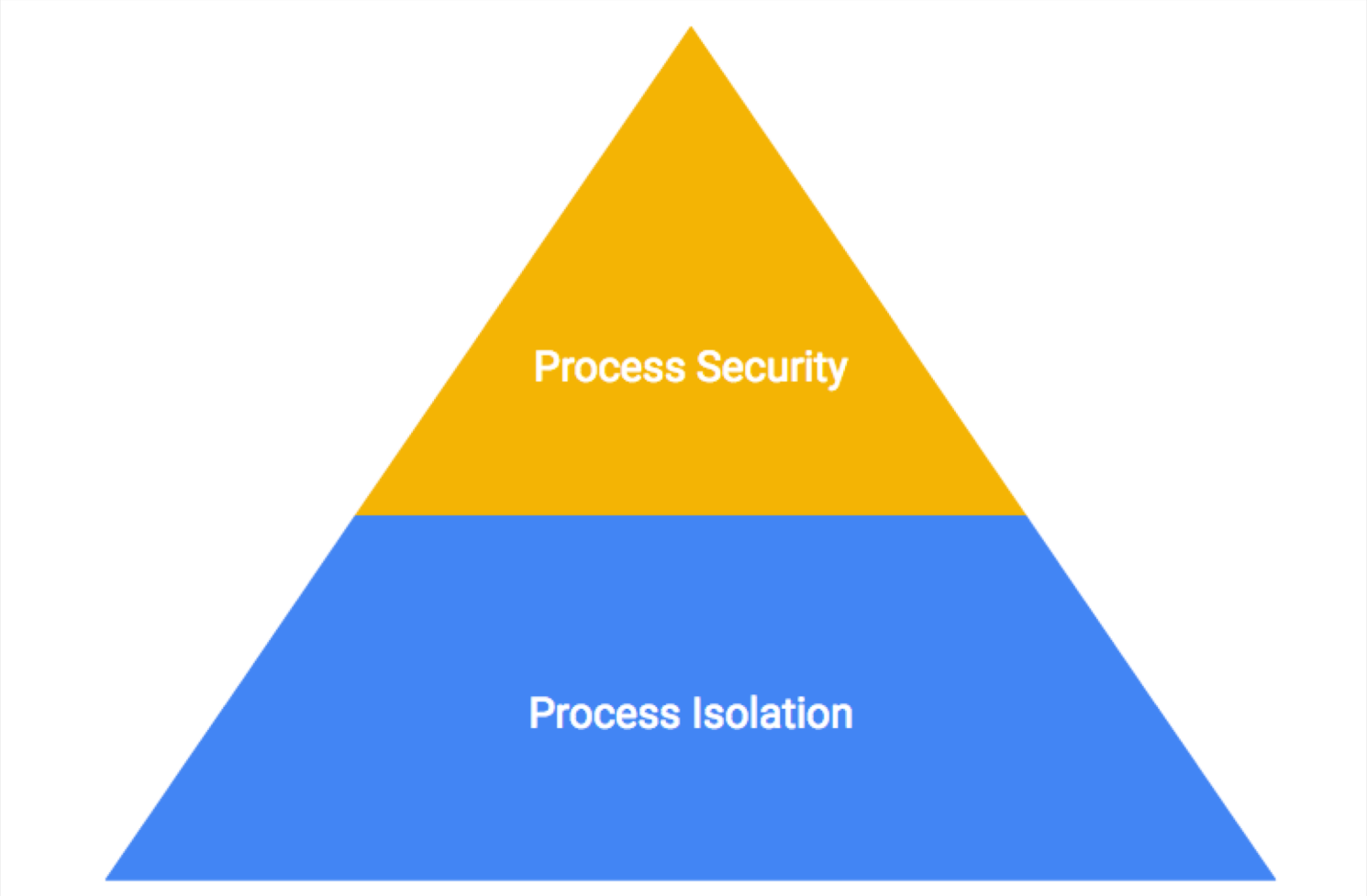
Docker Images

- Read only templates from which containers are launched from
- Each image consists of layers
- When you change an image a new layer is created



Container Security

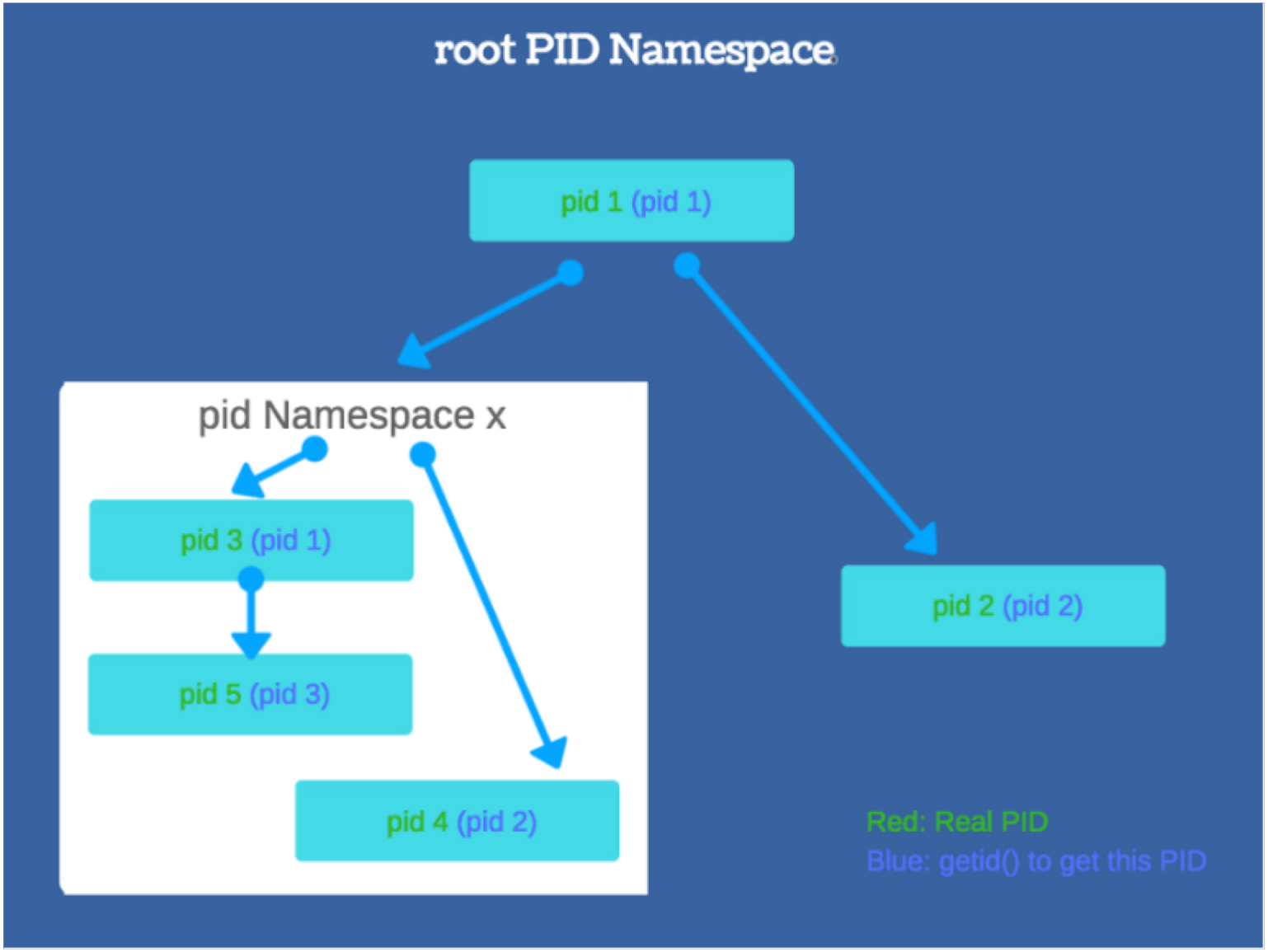
OS Virtualization Security Building Blocks



Kernel Namespaces

- Limits what a process can see
  - The **pid namespace** partitions kernel resources such that one set of processes may be provided with an independent set of process IDs (PIDs). Each container gets its own network stack
  - **Network namespaces** create virtual networking interfaces to allow programs to run on any port without conflict
  - **Mount namespaces** enable the mounting and un-mounting of filesystems without affecting the host filesystem
- No privileged access to the sockets or interfaces of another container

PID Namespace

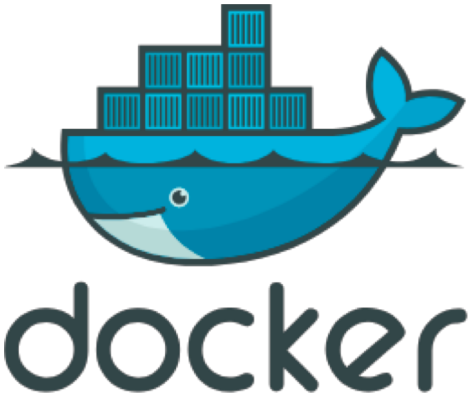


Control Groups

- Ensures each container is provided with its fair share of memory, CPU, disk I/O and more
- DoS anyone?
- Released in 2006 in kernel 2.6.24

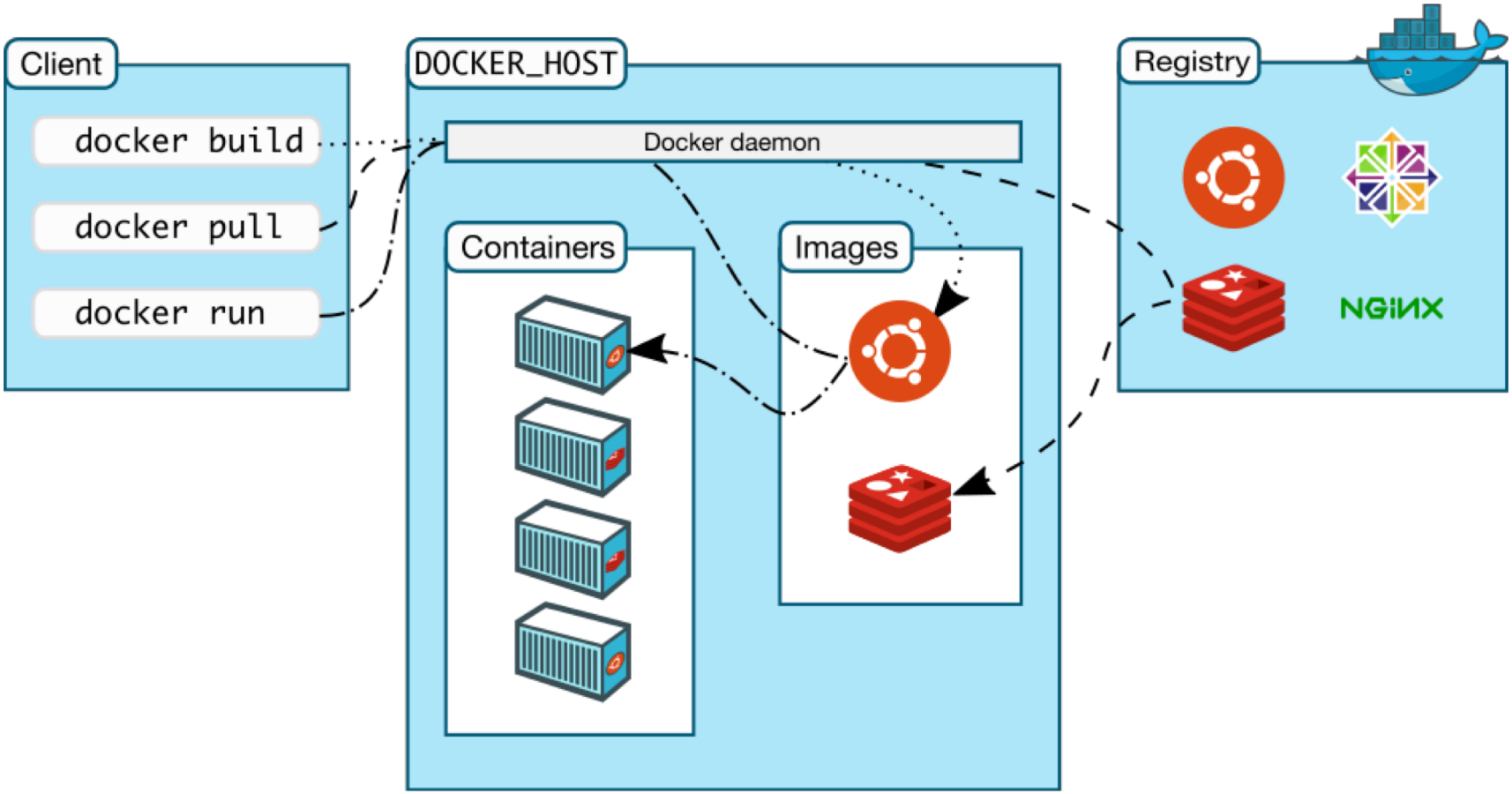
Docker Engine

Client-Server application that includes a few key components

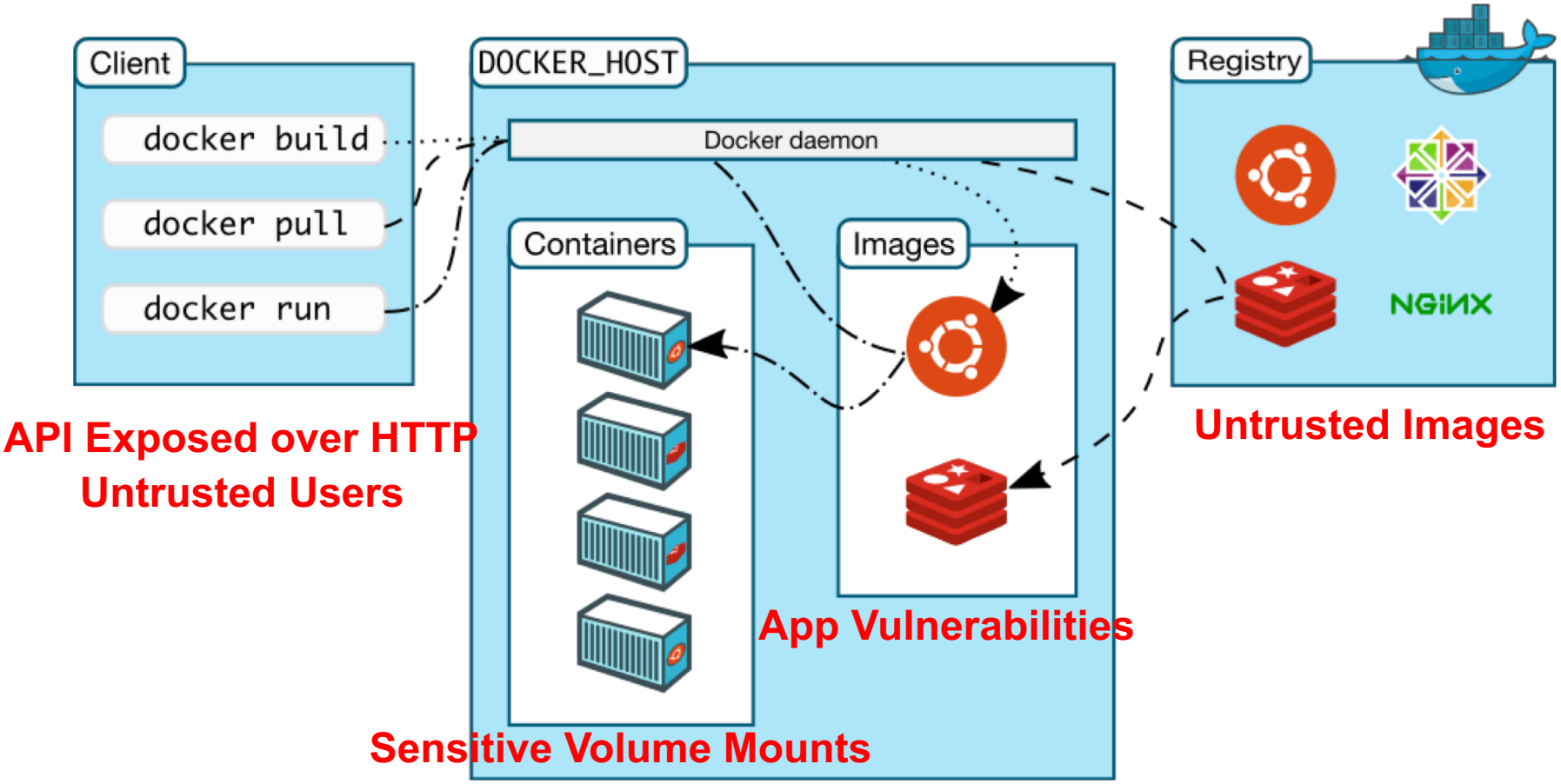


- **Docker Daemon** (dockerd)
  - Responsible for container orchestration
- **REST API**
  - Used to talk to the Docker daemon
- **Docker Client** (CLI)
  - Interface to interact with the Docker daemon

Docker Engine



Docker Security Gotchas



Container Security Benefits

- Patching Simplicity
- Typically Short Lifespans
- One Process Per Container (Ideally)
- Isolation from Others

Docker is a daemon running as root

Docker daemon attack surface

Running containers (and applications) with Docker implies running the Docker daemon. This daemon currently requires `root` privileges, and you should therefore be aware of some important details.

First of all, **only trusted users should be allowed to control your Docker daemon**. This is a direct consequence of some powerful Docker features. Specifically, Docker allows you to share a directory between the Docker host and a guest container; and it allows you to do so without limiting the access rights of the container. This means that you can start a container where the `/host` directory will be the `/` directory on your host; and the container will be able to alter your host filesystem without any restriction. This is similar to how virtualization systems allow filesystem resource sharing. Nothing prevents you from sharing your root filesystem (or even your root block device) with a virtual machine.

From https://docs.docker.com/engine/security/security/

Docker Images Running as Root

```
FROM ubuntu:latest
RUN apt-get update --fix-missing && \
   apt-get install -y redis-server && \
   rm -rf /var/lib/apt/lists/*
EXPOSE 6379
CMD redis-server
```

```
$ docker run --rm example whoami
root
```

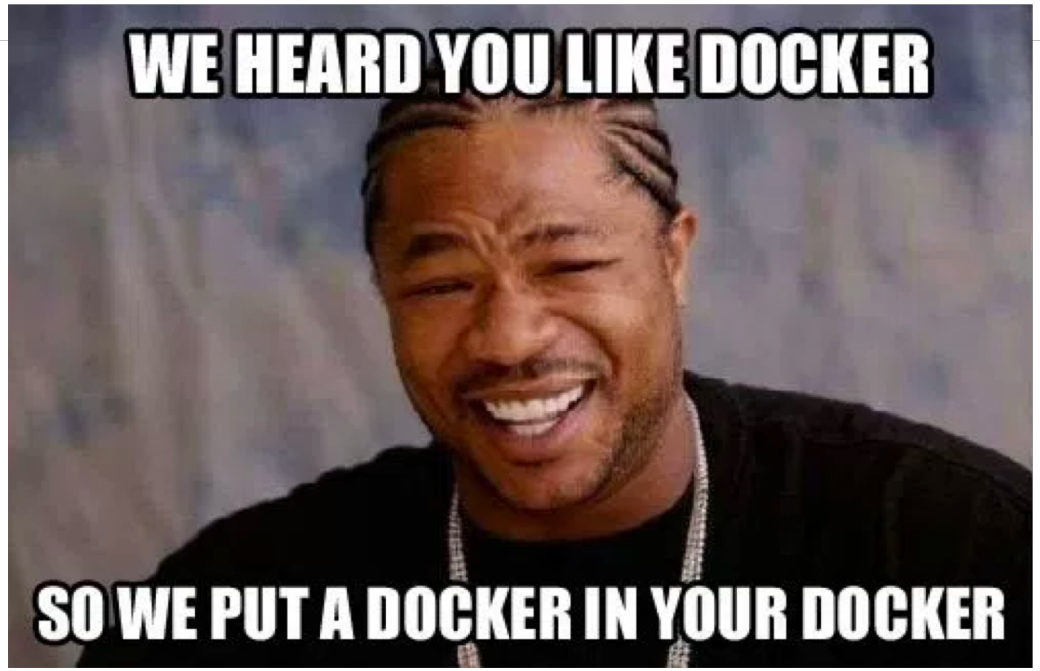
Docker Images Running as Root

- Declare a non-root user in our Dockerfile

```
FROM ubuntu:latest
RUN apt-get update --fix-missing && \
   apt-get install -y redis-server && \
   rm -rf /var/lib/apt/lists/*
USER 9000
EXPOSE 6379
CMD redis-server
```

A House of Cards: An Exploration of Security When Building Docker Containers

MARCH 08, 2018 · POSTED BY ETIENNE STALMANS

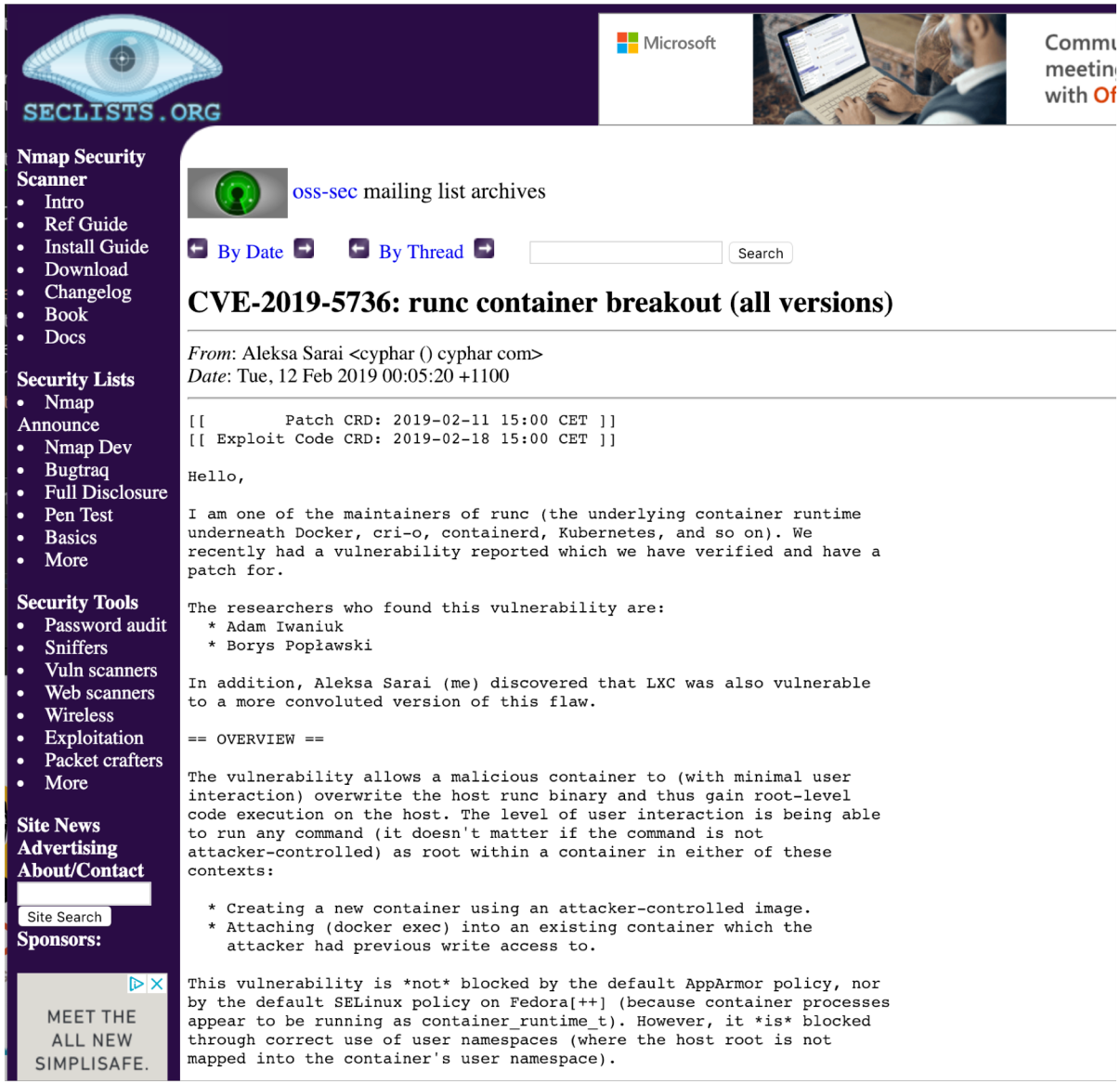


https://blog.heroku.com/exploration-of-security-when-building-docker-containers

It is possible to break out of a Docker container

```
root@precise64:~# docker run gabrtv/shocker
[***] docker VMM-container breakout Po(C) 2014             [***]
[***] The tea from the 90's kicks your sekurity again.     [***]
[***] If you have pending sec consulting, I'll happily     [***]
[***] forward to my friends who drink secury-tea too!      [***]
[*] Resolving 'etc/shadow'
[*] Found vmlinuz
[*] Found vagrant
[*] Found lib64
[*] Found usr
[*] Found ...
[*] Found shadow
[+] Match: shadow ino=3935729
[*] Brute forcing remaining 32bit. This can take a while...
[*] (shadow) Trying: 0x00000000
[*] #=8, 1, char nh[] = {0xf1, 0x0d, 0x3c, 0x00, 0x00, 0x00, 0x00, 0x00};
[!] Got a final handle!
[*] #=8, 1, char nh[] = {0xf1, 0x0d, 0x3c, 0x00, 0x00, 0x00, 0x00, 0x00};
[!] Win! /etc/shadow output follows:
root:!:15597:0:99999:7:::
daemon:*:15597:0:99999:7:::
bin:*:15597:0:99999:7:::
```

Even in 2019…



SECLISTS.ORG

Microsoft

Nmap Security Scanner
- Intro
- Ref Guide
- Install Guide
- Download
- Changelog
- Book
- Docs

Security Lists
- Nmap Announce
- Nmap Dev
- Bugtraq
- Full Disclosure
- Pen Test
- Basics
- More

Security Tools
- Password audit
- Sniffers
- Vuln scanners
- Web scanners
- Wireless
- Exploitation
- Packet crafters
- More

Site News
Advertising
About/Contact

Site Search

Sponsors:

MEET THE ALL NEW SIMPLISAFE.

oss-sec mailing list archives

By Date By Thread Search

CVE-2019-5736: runc container breakout (all versions)

From: Aleksa Sarai <cyphar () cyphar com>
Date: Tue, 12 Feb 2019 00:05:20 +1100

```
[[        Patch CRD: 2019-02-11 15:00 CET ]]
[[ Exploit Code CRD: 2019-02-18 15:00 CET ]]

Hello,

I am one of the maintainers of runc (the underlying container runtime
underneath Docker, cri-o, containerd, Kubernetes, and so on). We
recently had a vulnerability reported which we have verified and have a
patch for.

The researchers who found this vulnerability are:
  * Adam Iwaniuk
  * Borys Popławski

In addition, Aleksa Sarai (me) discovered that LXC was also vulnerable
to a more convoluted version of this flaw.

== OVERVIEW ==

The vulnerability allows a malicious container to (with minimal user
interaction) overwrite the host runc binary and thus gain root-level
code execution on the host. The level of user interaction is being able
to run any command (it doesn't matter if the command is not
attacker-controlled) as root within a container in either of these
contexts:

  * Creating a new container using an attacker-controlled image.
  * Attaching (docker exec) into an existing container which the
    attacker had previous write access to.

This vulnerability is *not* blocked by the default AppArmor policy, nor
by the default SELinux policy on Fedora[++] (because container processes
appear to be running as container_runtime_t). However, it *is* blocked
through correct use of user namespaces (where the host root is not
mapped into the container's user namespace).
```

Yes. Docker Images Have Vulnerabilities

Docker vulnerability scanning

anchore

clair

QUAY

K8S - A Gentle Introduction

K8S - A Gentle Introduction

Kubernetes is an open-source platform built to automate **deployment**, **scaling** and **orchestration** of containers.

K8S is **portable**. Clusters can be deployed on a public/private cloud, on prem, and even on your laptop.

K8S is **customizable**. It is modular and extensible to fit a variety of use-cases.

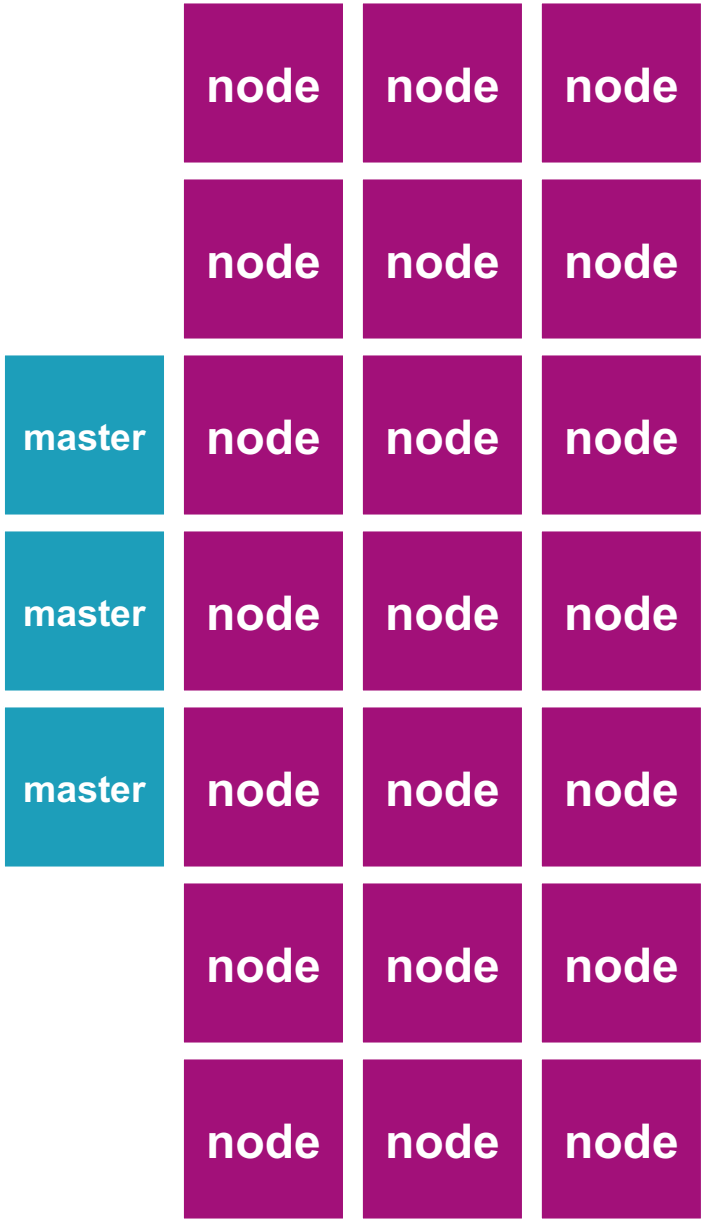
K8S is **scalable**. It provides self-healing, auto scaling, and replication out of the box.

cluster

virtual machines that Kubernetes manages

cluster

cluster

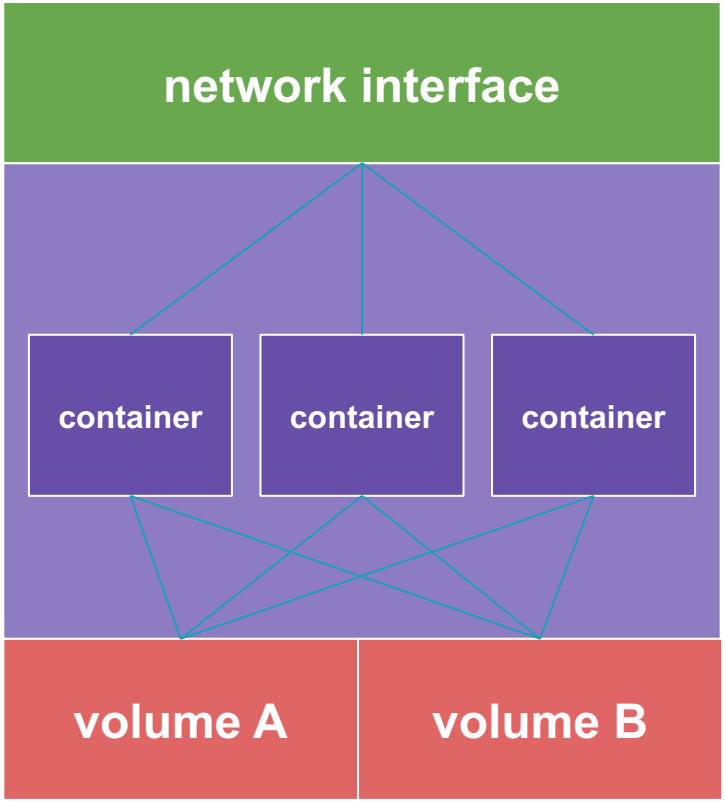


cluster

pod

group of containers sharing storage and network

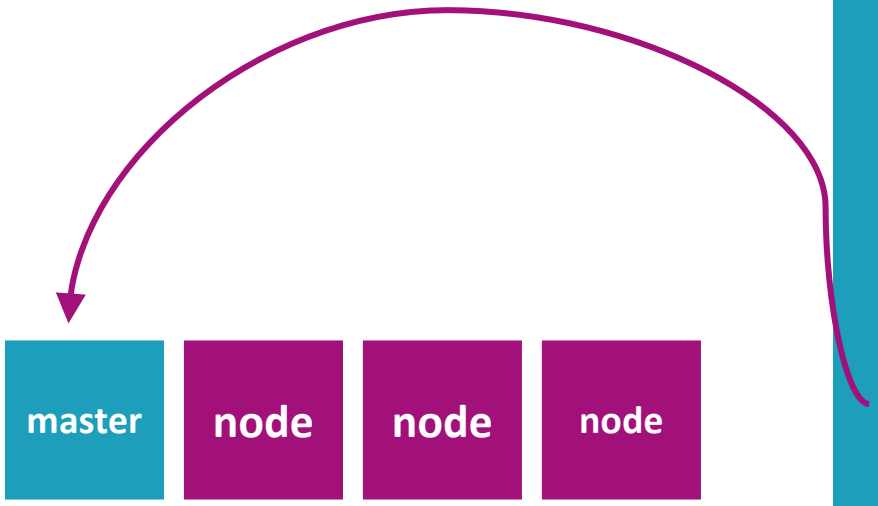
pod



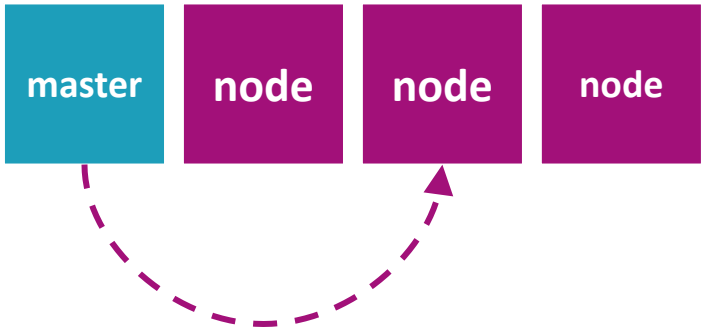
pod

```
apiVersion: v1
kind: Pod
metadata:
  name: redis-rails
spec:
  containers:
  - name: key-value
    image: redis
    ports:
    - containerPort: 6379
  - name: rails-frontend
    image: rails
    ports:
    - containerPort: 3000
```

pod.yaml



pod.yaml



pod.yaml

pod.yaml

deployment

ensure *N* pods are up and running

deployment

```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: rails-deployment
  labels:
    app: rails
spec:
  replicas: 4
  selector:
  matchLabels:
    app: rails
  template:
    metadata:
      labels:
        app: rails
    spec:
      containers:
      - name: key-value
        image: redis
        ports:
        - containerPort: 6379
      - name: rails-frontend
        image: rails
        ports:
        - containerPort: 3000
```

deploy.yaml

```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: rails-deployment
  labels:
    app: rails
spec:
  replicas: 4
  selector:
  matchLabels:
    app: rails
  template:
    metadata:
      labels:
        app: rails
    spec:
      containers:
      - name: key-value
        image: redis
        ports:
        - containerPort: 6379
      - name: rails-frontend
        image: rails
        ports:
        - containerPort: 3000
```

deploy.yaml

```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: rails-deployment
  labels:
    app: rails
spec:
  replicas: 4
  selector:
  matchLabels:
    app: rails
  template:
    metadata:
      labels:
        app: rails
  spec:
    containers:
    - name: key-value
      image: redis
      ports:
      - containerPort: 6379
    - name: rails-frontend
      image: rails
      ports:
      - containerPort: 3000
```

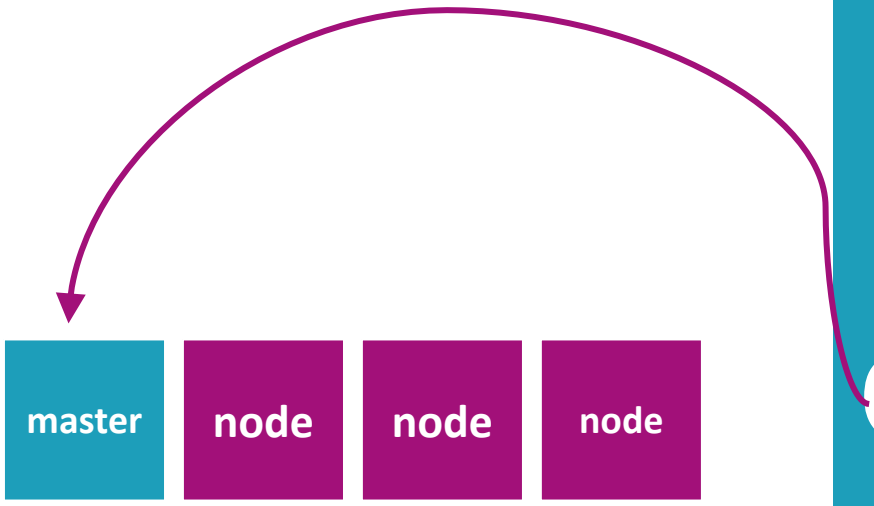
deploy.yaml

```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: rails-deployment
  labels:
    app: rails
spec:
  replicas: 4
  selector:
  matchLabels:
    app: rails
  template:
    metadata:
      labels:
        app: rails
    spec:
      containers:
      - name: key-value
        image: redis
        ports:
        - containerPort: 6379
      - name: rails-frontend
        image: rails
        ports:
        - containerPort: 3000
```

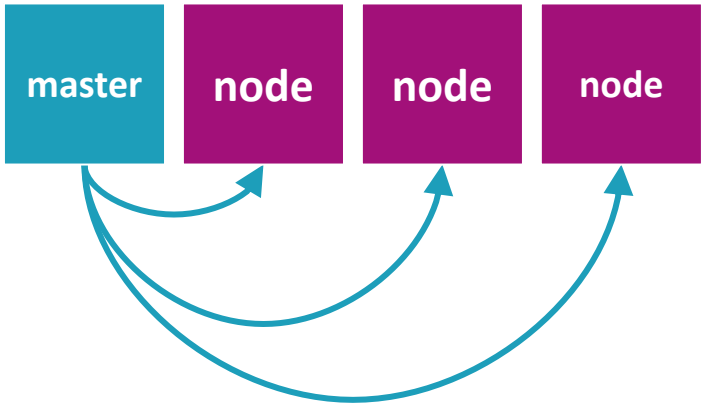
deploy.yaml

```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: rails-deployment
  labels:
    app: rails
spec:
  replicas: 4
  selector:
  matchLabels:
    app: rails
  template:
    metadata:
      labels:
        app: rails
    spec:
      containers:
      - name: key-value
        image: redis
        ports:
        - containerPort: 6379
      - name: rails-frontend
        image: rails
        ports:
        - containerPort: 3000
```

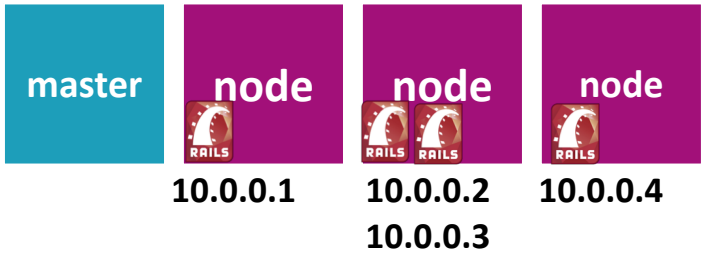
deploy.yaml



deploy.yaml



deploy.yaml

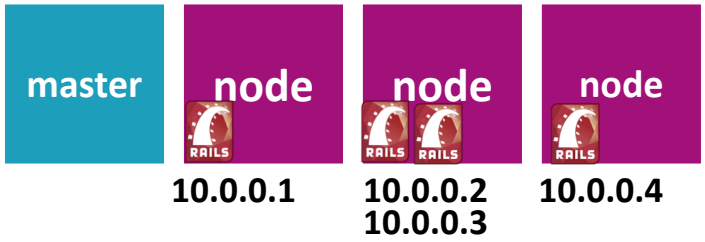


deploy.yaml

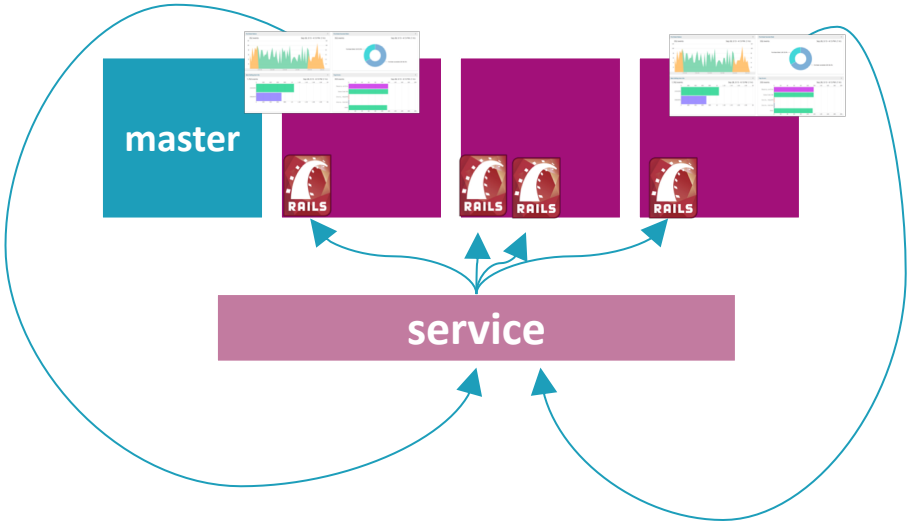
abstraction layer that enables pod communication

service

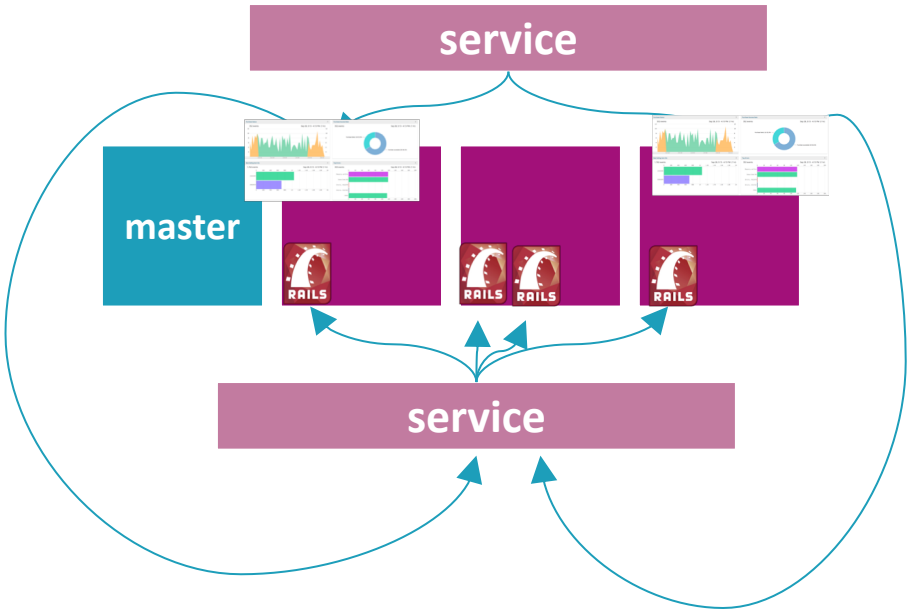
service



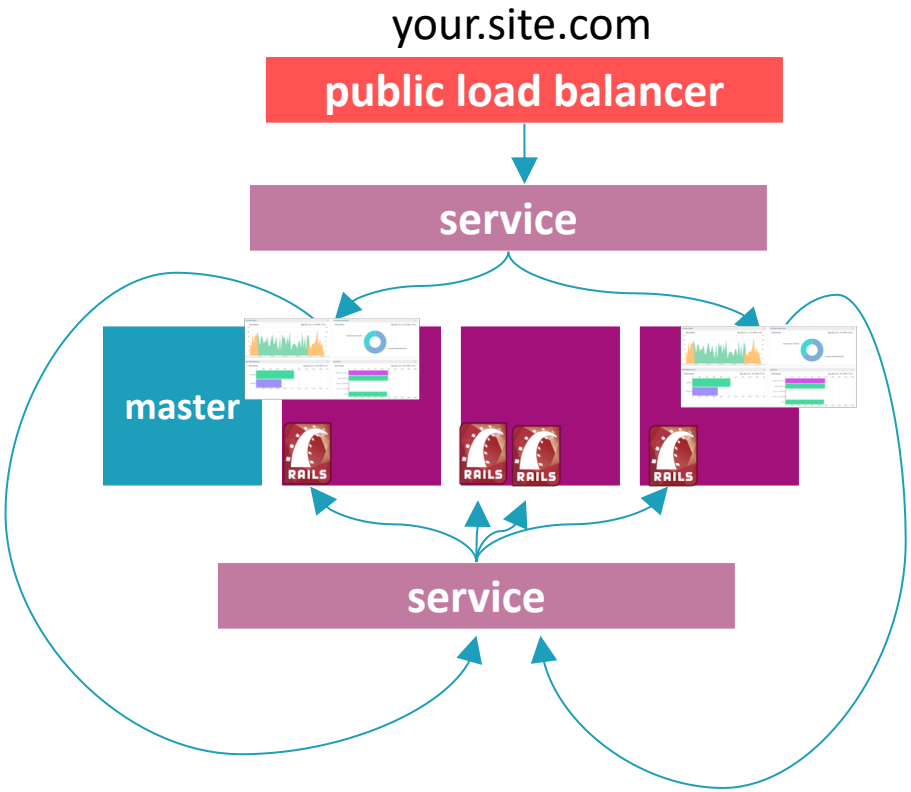
service



service



service



service

```
kind: Service
apiVersion: v1
metadata:
  name: web-frontend
spec:
  ports:
  - name: http
    port: 80
    targetPort: 3000
    protocol: TCP
  selector:
    app: rails
  type: LoadBalancer
```

svc.yaml

```
kind: Service
apiVersion: v1
metadata:
  name: web-frontend
spec:
  ports:
  - name: http
    port: 80
    targetPort: 80
    protocol: TCP
  selector:
    app: rails
  type: LoadBalancer
```

svc.yaml

```
kind: Service
apiVersion: v1
metadata:
  name: web-frontend
spec:
  ports:
  - name: http
    port: 80
    targetPort: 80
    protocol: TCP
  selector:
    app: rails
  type: LoadBalancer
```

svc.yaml

```
kind: Service
apiVersion: v1
metadata:
  name: web-frontend
spec:
  ports:
  - name: http
    port: 80
    targetPort: 80
    protocol: TCP
  selector:
    app: rails
  type: LoadBalancer
```

svc.yaml

Labels and Selectors

Metadata (key-value) which can be attached to a resource

Labels

Labels

Used for identification such as app name, tier, environment

```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: rails-deployment
  labels:
    app: rails
spec:
  replicas: 4
  selector:
  matchLabels:
    app: rails
  template:
    metadata:
      labels:
        app: rails
    spec:
      containers:
      - name: key-value
        image: redis
        ports:
        - containerPort: 6379
      - name: rails-frontend
        image: rails
        ports:
        - containerPort: 3000
```

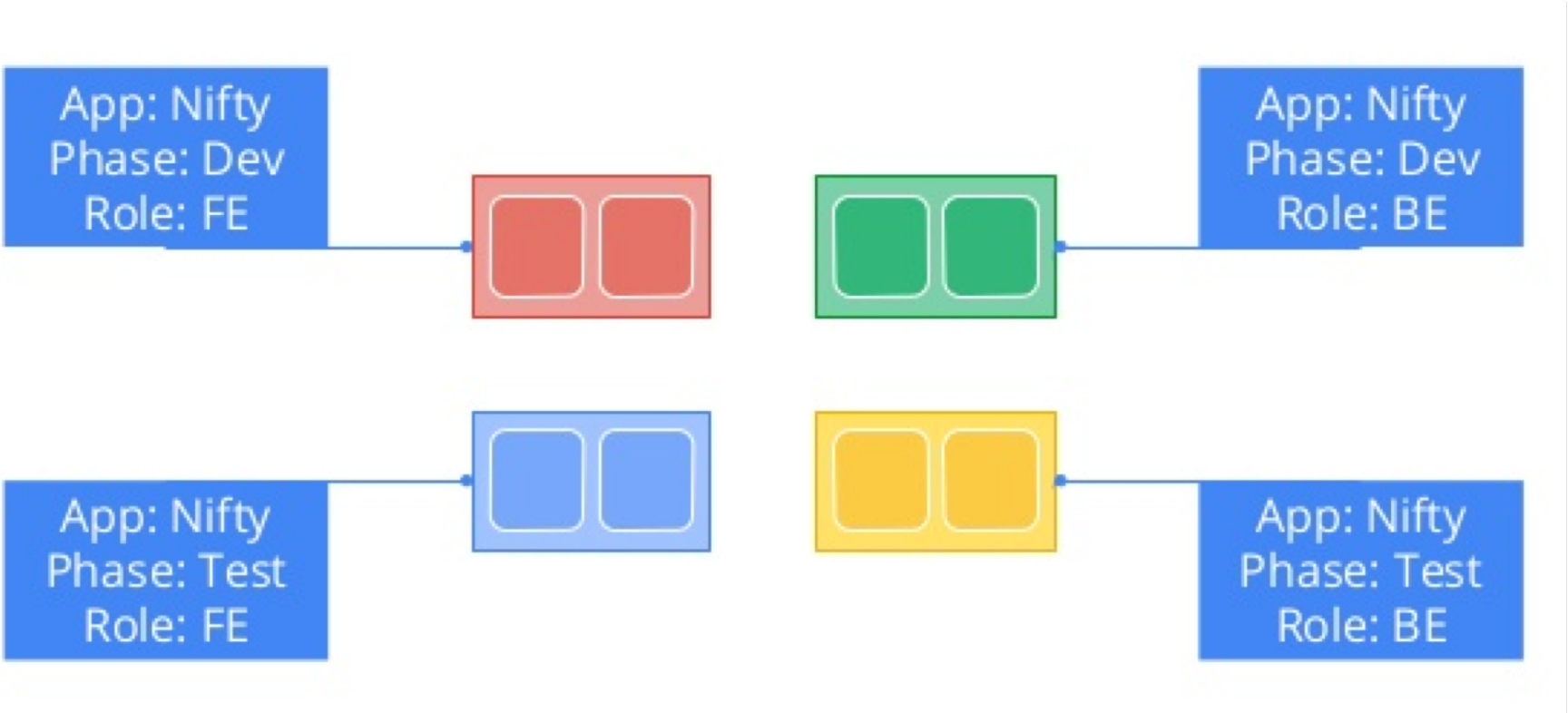
deploy.yaml

Provides loose coupling between objects

Selectors

```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: rails-deployment
  labels:
    app: rails
spec:
  replicas: 4
  selector:
  matchLabels:
    app: rails
  template:
    metadata:
      labels:
        app: rails
    spec:
      containers:
      - name: key-value
        image: redis
        ports:
        - containerPort: 6379
      - name: rails-frontend
        image: rails
        ports:
        - containerPort: 3000
```

deploy.yaml



Ingress

configure external access to your cluster

ingress.yaml

```
kind: Ingress
apiVersion: extensions/v1beta1
metadata:
  name: web-ingress
spec:
  backend:
    serviceName: web-frontend
    servicePort: 80
```

ingress.yaml

```
kind: Ingress
apiVersion: extensions/v1beta1
metadata:
  name: web-ingress-vhosts
  rules:
  - host: sub.domain.com
    http:
      paths:
      - backend:
          serviceName: web-frontend-1
          servicePort: 80
 - host: other.domain.com
    http:
      paths:
      - backend:
          serviceName: web-frontend-2
          servicePort: 80
```

ingress.yaml

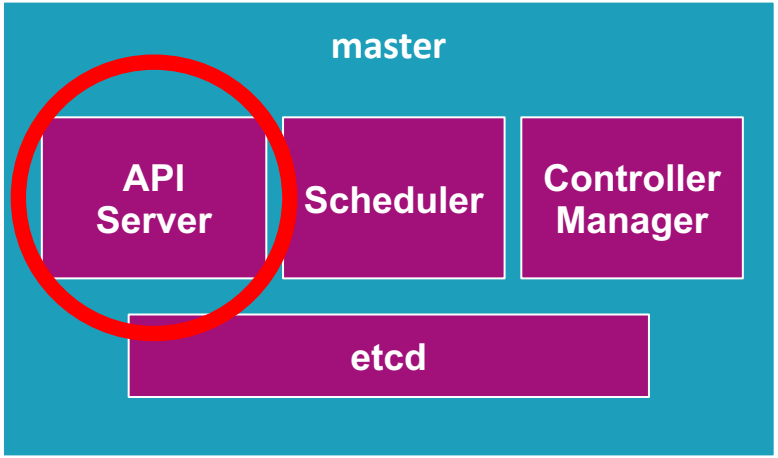
manage different environments in the same cluster

namespace

```
kind: Namespace
apiVersion: v1
metadata:
   name: development
```

ns.yaml

Kubernetes Security Model



- The REST API is the fundamental fabric of Kubernetes
- All operations and communications between components, and external user commands are REST API calls that the API Server handles
- Everything in the Kubernetes platform is treated as an API object and has a corresponding entry in the API

K8S Security Model

kubectl

https://

apiserver

Authentication (*Who* can access the cluster?

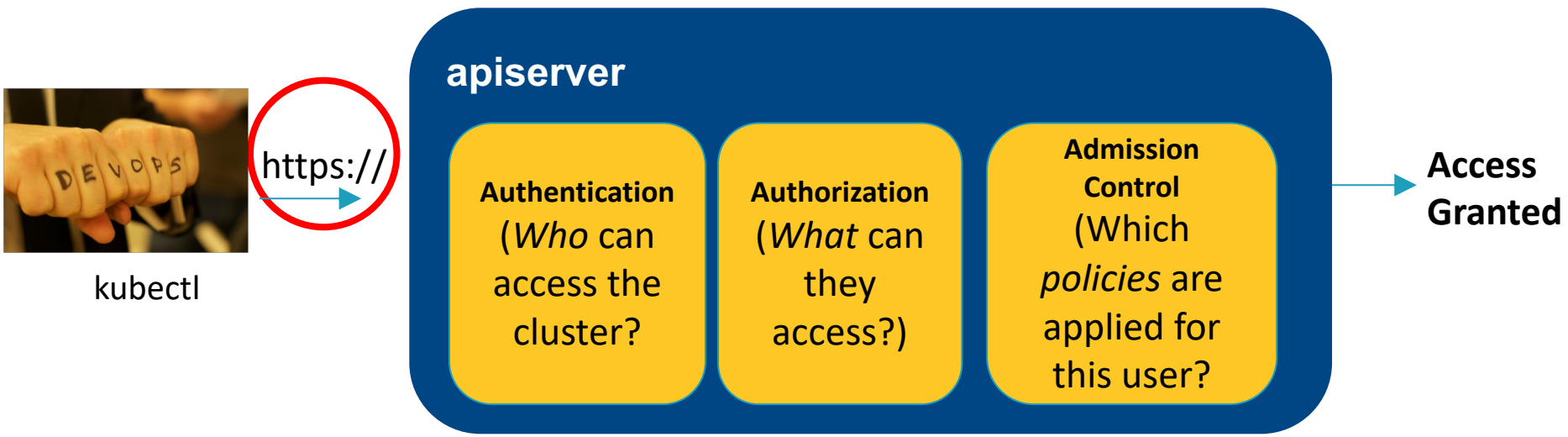
Authorization (*What* can they access?)

Admission Control (Which *policies* are applied for this user?

Access Granted

Transport Security

- K8S API typically serves traffic over TLS
- Self-Signed Cert provisioned on operators laptop in $USER/.kube/config



Authentication

- Supports many authentication modules:
  HTTP Basic, OpenID, Tokens, Client Cert, Keystone
- Multiple modules can be specified

kubectl

https://

apiserver

Authentication (*Who* can access the cluster?

Authorization (*What* can they access?)

Admission Control (Which *policies* are applied for this user?

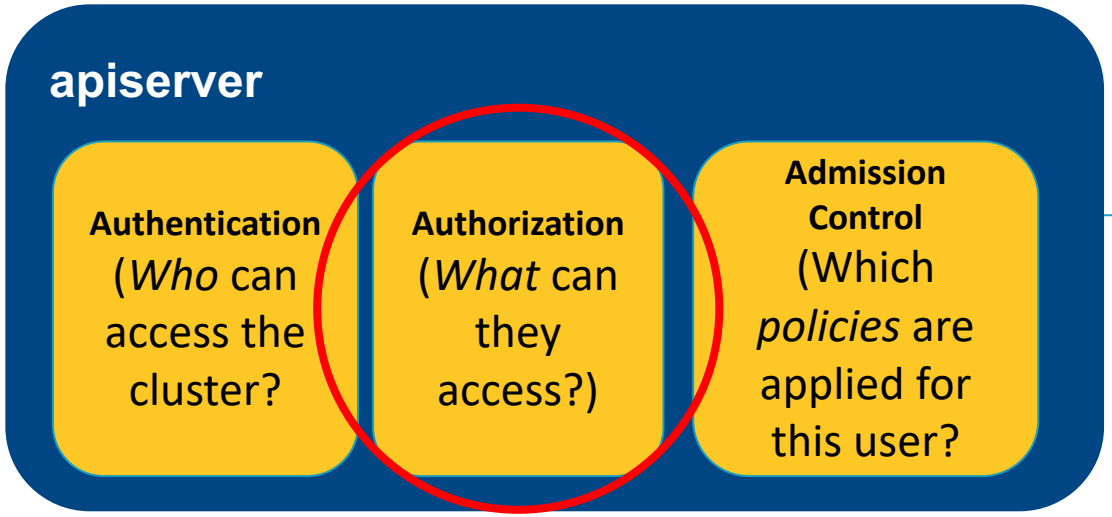
Access Granted

Authorization

- Every HTTP request is authorized

  get, list, create, update, etc.
- Request attributes are checked against policy

https://



Authorization

`--authorization-mode=AlwaysAllow` allows all requests; use if you don't need authorization

`--authorization-mode=ABAC` allows for a simple local-file-based user-configured authorization policy

`--authorization-mode=RBAC` allows for authorization to be driven by the Kubernetes API

Admission Controllers

- Intercept requests prior to object creation
- May mutate incoming request to apply system defaults

kubectl

https://

apiserver

Authentication (*Who* can access the cluster?

Authorization (*What* can they access?)

Admission Control (Which *policies* are applied for this user?

Access Granted

Admission Controllers

AlwaysPullImages

DenyEscalatingExec

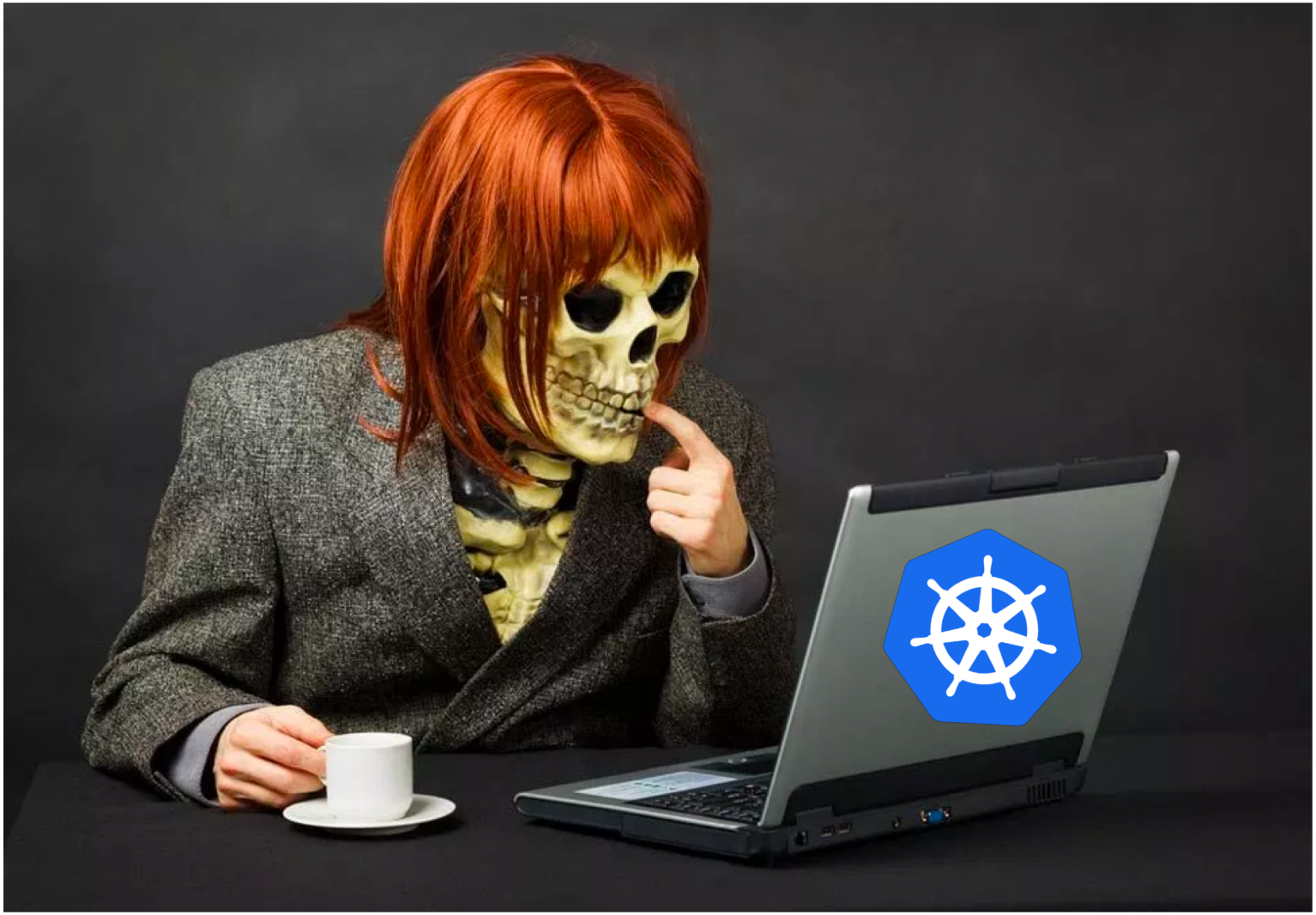
ResourceQuota

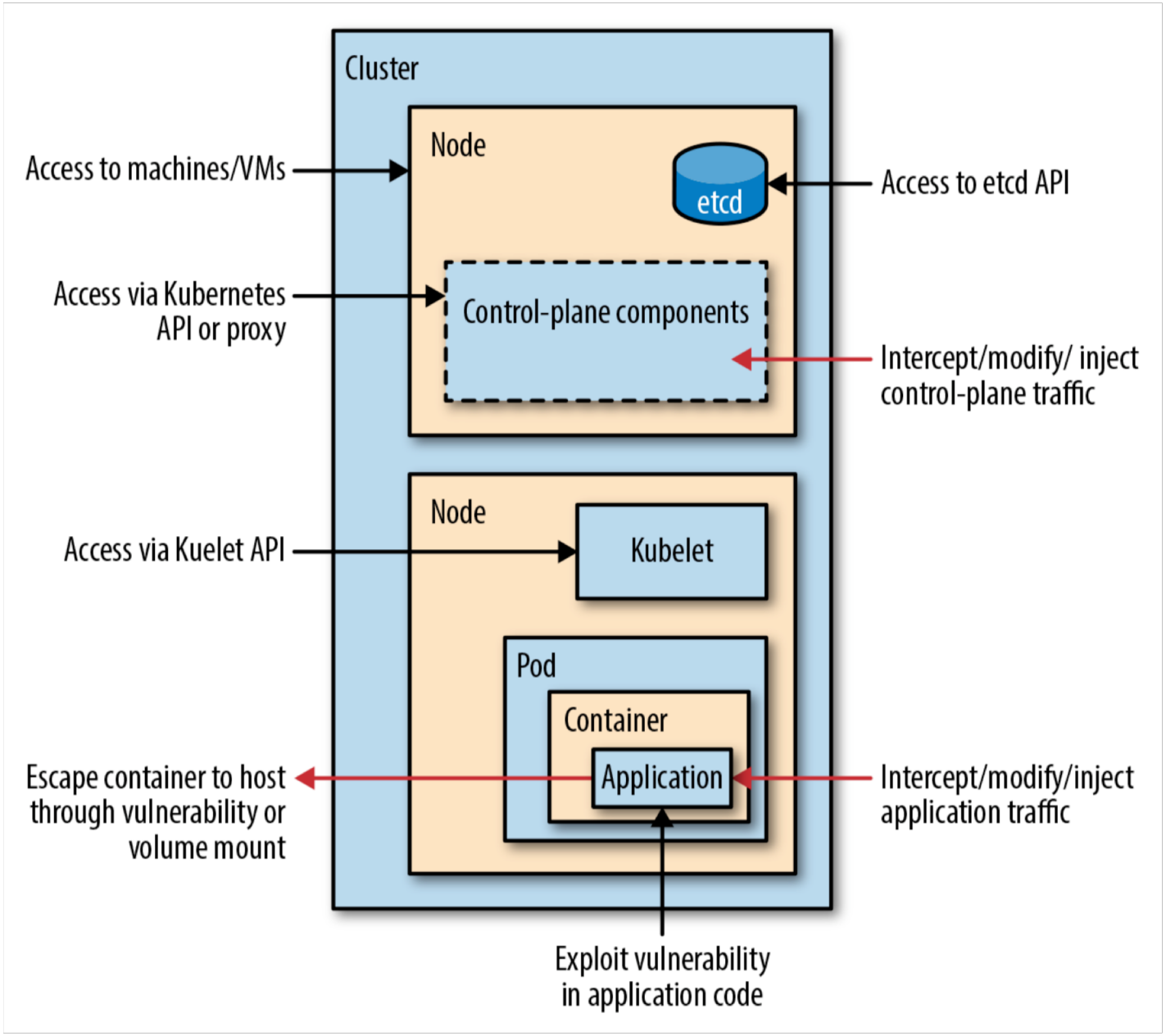
NamespaceExists

http://kubernetes.io/docs/admin/admission-controllers/

Attacking and Defending Kubernetes

Let’s Play a Game - Kubernetes Threat Model





Source: Kubernetes Security - Operating Kubernetes Clusters and Applications Safely

Kubernetes Threat Model

User Compromise and Insider Threats

- Cluster admin account compromise
- Rogue Employee
- Tenant account compromise leads to the application compromise

Application Vulnerabilities

- Lack of authentication and authorization, both k8s internal and external
- Weak or incorrect usage of cryptography
- Application and API vulnerabilities - remote code execution (RCE), web vulnerabilities (XSS, CSRF, SSRF, SQL Injection etc.)
- Unsecured third party components

Kubernetes Threat Model

Network and Infrastructure

- Network snooping, ARP spoof attacks
- Compromising infrastructure services (etc. NTP, DNS, SSH)
- Kernel and other operating system vulnerabilities

Application Containers

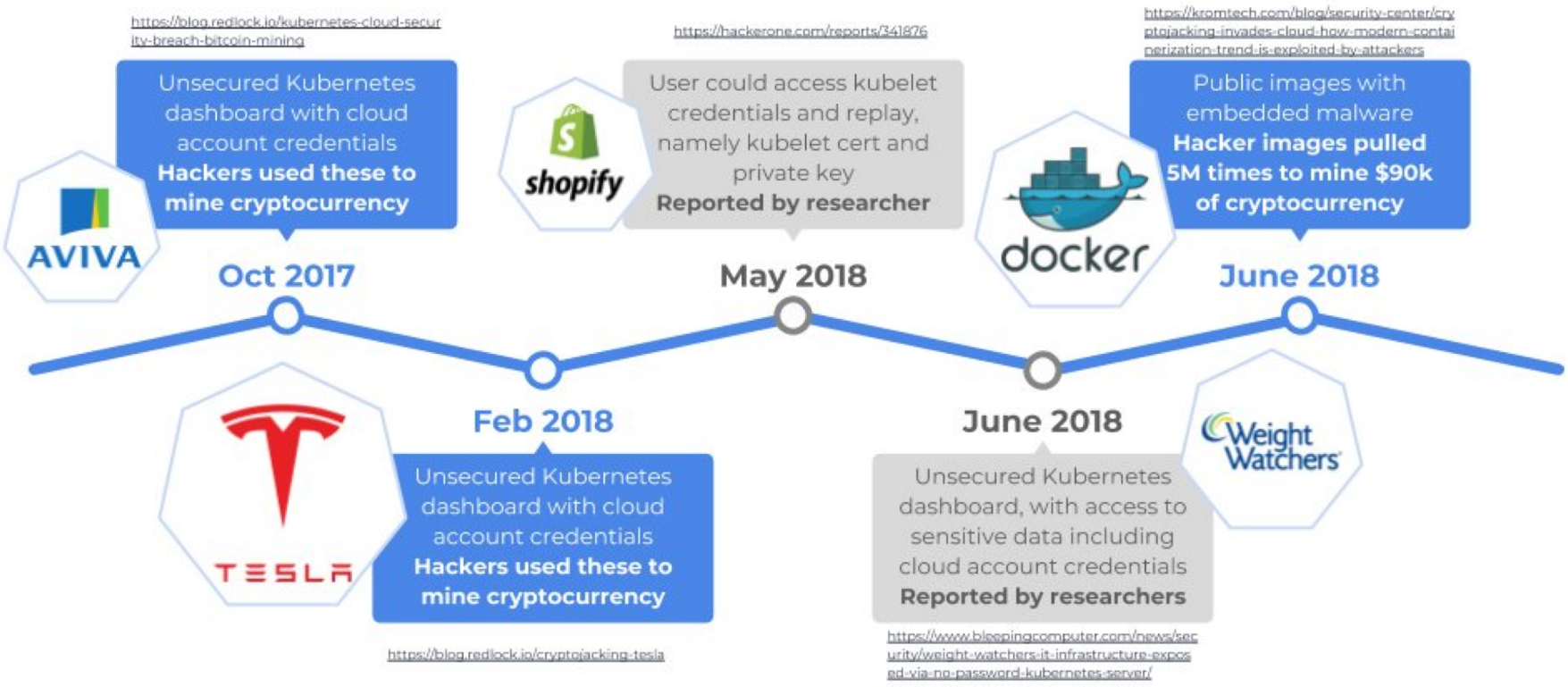
- Container breakout and unauthorized access control plane and other containers
- Denial of Service - resource hogging, eating up CPU/Mem/Disk/IO to impact or even crash other containers
- Compromised or malicious image or pipeline

Kubernetes Threat Model

Misconfiguration

- Insecure default configurations - unused open ports, services, not enforcing system/application limits, failing to implement security features
- Misuse of passwords, passphrases, TLS private keys (*cough* checking them into git *cough*. Bad handling include key reuse, insecure handling of keys, no key rotation, weak passwords, not using MFA etc.
- Lack of network segmentation - exposing critical systems to various network attacks

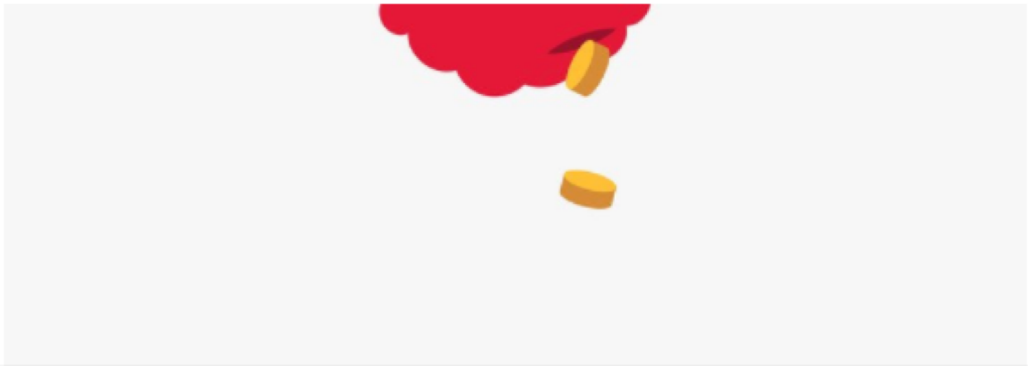
Kubernetes security threats in the wild



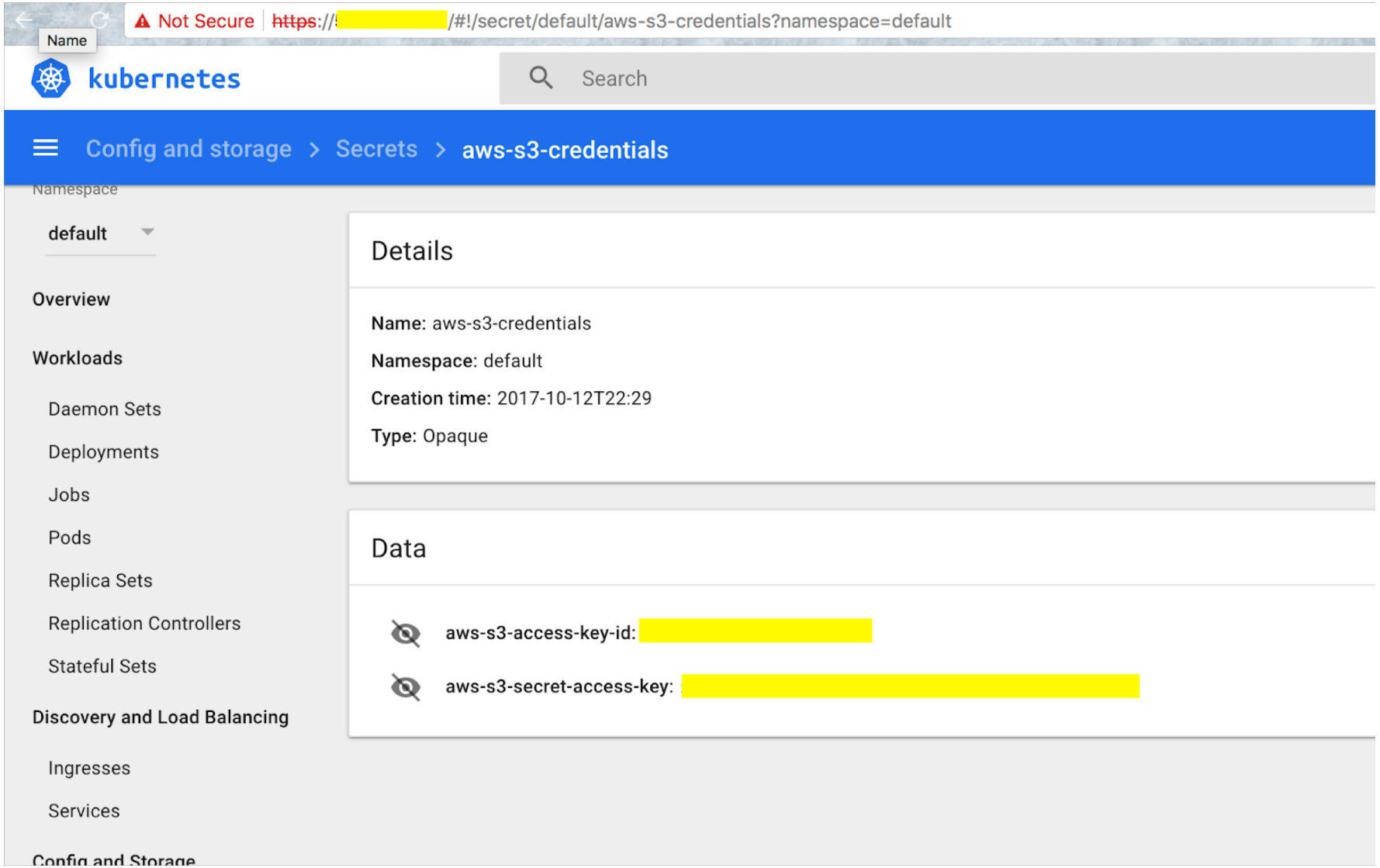
O boy.

LILY HAY NEWMAN SECURITY 02.20.18 05:06 PM

HACK BRIEF: HACKERS ENLISTED TESLA'S PUBLIC CLOUD TO MINE CRYPTOCURRENCY



Attack: Unauthorized Dashboard Access



Defense: Unauthorized Dashboard Access

- ***Always*** run RBAC on your cluster
- By default, the Dashboard ServiceAccount has very limited privileges. Do not grant the Kubernetes dashboard service account elevated privileges such as root!
- If access is needed, create SAs per user with limited permissions
- Don’t expose to the internet
- Don’t be Tesla

Attack: Elevated Pod Privileges

- Pods may be deployed with containers that require elevated privileges:
  - "privileged mode" grants containers the ability to manipulate the network stack or access devices
  - Containers may run as root (User ID = 0)
  - Containers may request to mount sensitive volumes or request write access to volumes
  - Containers may request to bind to host ports
  - Containers may request elevated Linux capabilities

- Compromised containers can take full advantage of these privileges to attack the cluster and cloud infrastructure

Pod Security Context

- Pod security context is defined in the pod or deployment manifest
- Defines the the privilege and access control for a pod
- The security context defined in a pod applies to all containers within the pod
- Examples include:
  - Defining seccomp, SELinux, or AppArmor profiles
  - Defining users and groups containers use to run
  - Whitelisting certain Linux privileges to the container

#KubernetesSecurityTip: Pod Security Context should be used along with Pod Security Policies to enforce strict security admission controls

```
apiVersion: v1
kind: Pod
metadata:
  name: priv-pod
spec:
  securityContext:
    privileged: true
  securityContext:
    runAsUser: 1001
  containers:
    - name:  pause
      image: k8s.gcr.io/pause
      securityContext:
        capabilities:
          add: ["NET_ADMIN", "SYS_TIME"]
```

priv-pod.yaml

Defense: Pod Security Policies

- Pod security policies are represented by the **PodSecurityPolicy** resource
- Defines conditions a pod must meet to be scheduled
- Examples include:
  - Disallow privileged containers from running
  - Disallow containers that require root privileges
  - Disallow containers that access certain volume types
  - Disallow containers that access certain host ports

#KubernetesSecurityTip: Use the PodSecurityPolicy admission controller to restrict the use of privileged pods in your cluster

```
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
  name: my-psp
spec:
  privileged: false
  seLinux:
    rule: RunAsAny
  supplementalGroups:
    rule: RunAsAny
  runAsUser:
    rule: 'MustRunAsNonRoot'
  volumes:
    - 'configMap'
    - 'emptyDir'
    - 'secret'
    - 'persistentVolumeClaim'
```

psp.yaml

Attack: Unauthorized Network Access

- If you run an API endpoint in your cluster such as Redis without authentication, other pods may have unrestricted access to the pod
- A compromised pod may be able to read, alter, or delete data from another pod in the cluster

- It is important to isolate these workloads using granular Network Policies as well as mTLS where appropriate

#KubernetesSecurityTip: Third-party technologies such as Istio and Linkerd offer proxy services or ”sidecar” containers which can help deploy mTLS / proxying throughout your cluster

Attack: Unauthorized Network Access

- If you run an API endpoint in your cluster such as Redis without authentication, other pods may have unrestricted access to the pod
- A compromised pod may be able to read, alter, or delete data from another pod in the cluster

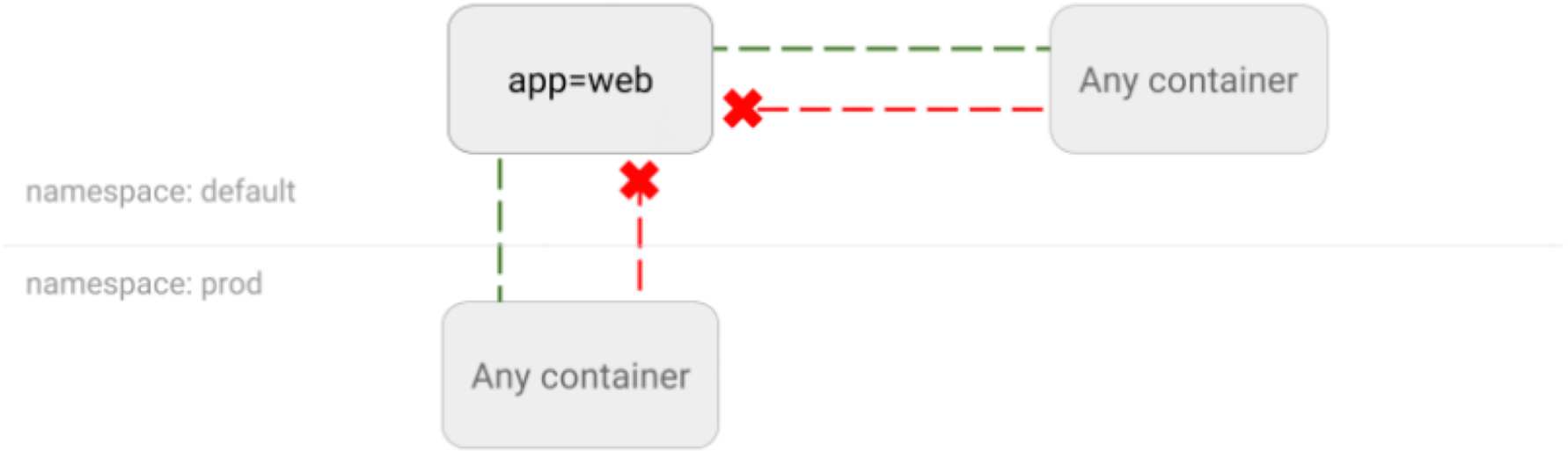
#KubernetesSecurityTip: Third-party technologies such as Istio and Linkerd offer proxy services or ”sidecar” containers which can help deploy mTLS / proxying throughout your cluster

Defense: Network Policies

- The Kubernetes object NetworkPolicy allows you to block traffic to pods
- Acts as a ”pod firewall” where rules are administered by cluster admins
- Best practice is to start with a default “deny all” and only add what you need
- Default Deny – You must build the whitelist

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: web-deny-all
spec:
  podSelector:
    matchLabels:
      app: web
  ingress: []
```

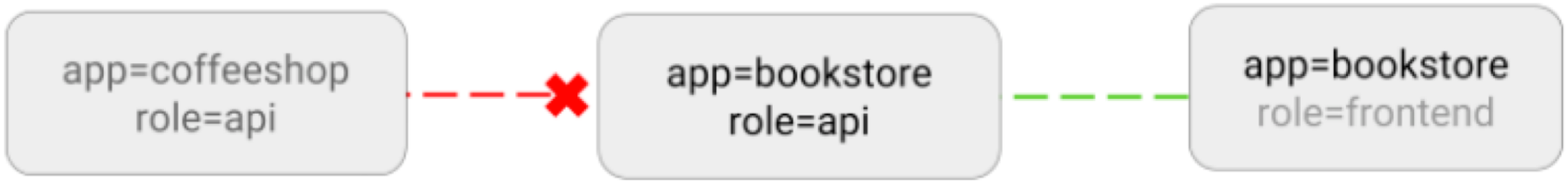
np-deny-all.yaml



https://github.com/ahmetb/kubernetes-network-policy-recipes/blob/master/01-deny-all-traffic-to-an-application.md

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: api-allow
spec:
  podSelector:
    matchLabels:
      app: bookstore
      role: api
  ingress:
  - from:
      - podSelector:
          matchLabels:
            app: bookstore
```

np-limit-traffic.yaml



https://github.com/ahmetb/kubernetes-network-policy-recipes/blob/master/02-limit-traffic-to-an-application.md

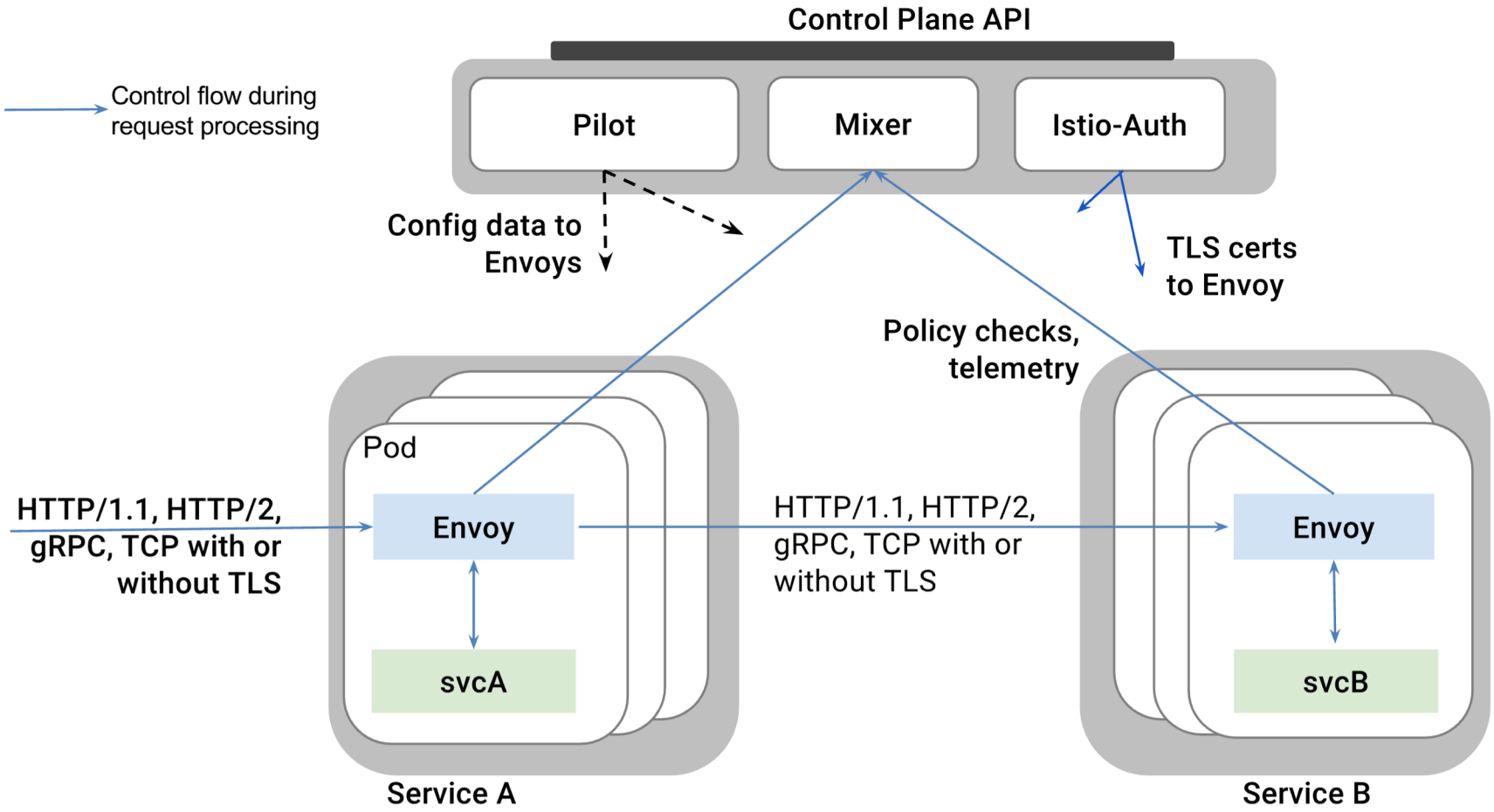
```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: limit-egress
spec:
  podSelector:
    matchLabels:
      app: foo
  policyTypes:
  - Egress
  egress:
  - ports:
    - port: 53
      protocol: UDP
    - port: 53
      protocol: TCP
  - to:
    - namespaceSelector: {}
```

limit-egress.yaml

Defense: Istio Service Mesh

- Istio is a service mesh for microservices (not just Kubernetes)
- Offers:
  - Monitoring
  - Metrics
  - Traffic Management and Routing
  - Security
  - Tracing

Defense: Istio Service Mesh

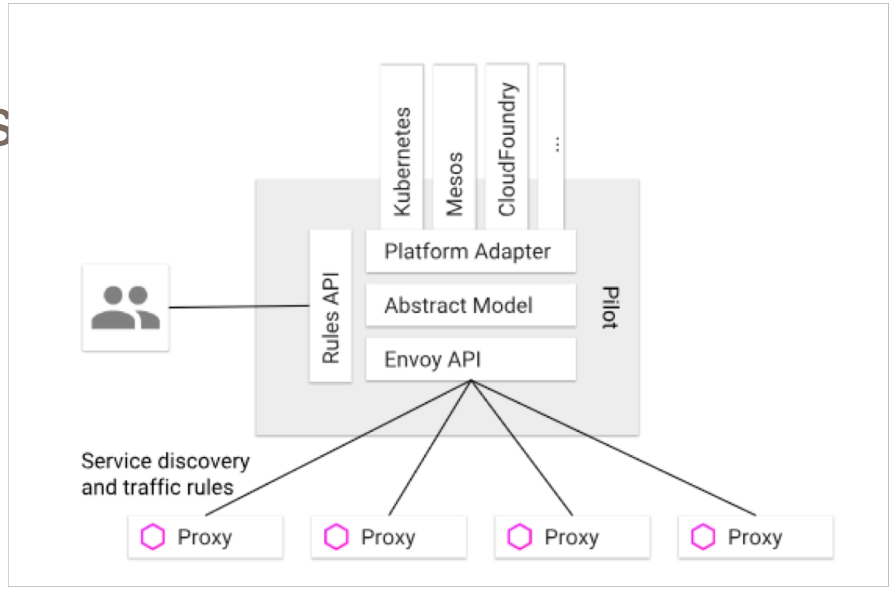


Istio: Envoy Proxy

- High performance load balancer
- Config management via API
- L7 Visibility
- Rate-limiting, health checks, retries, etc.
- In Kubernetes…
  - Envoy container is injected as a “sidecar” container
  - Controls pod ingress / egress routing
  - Config is via Pilot

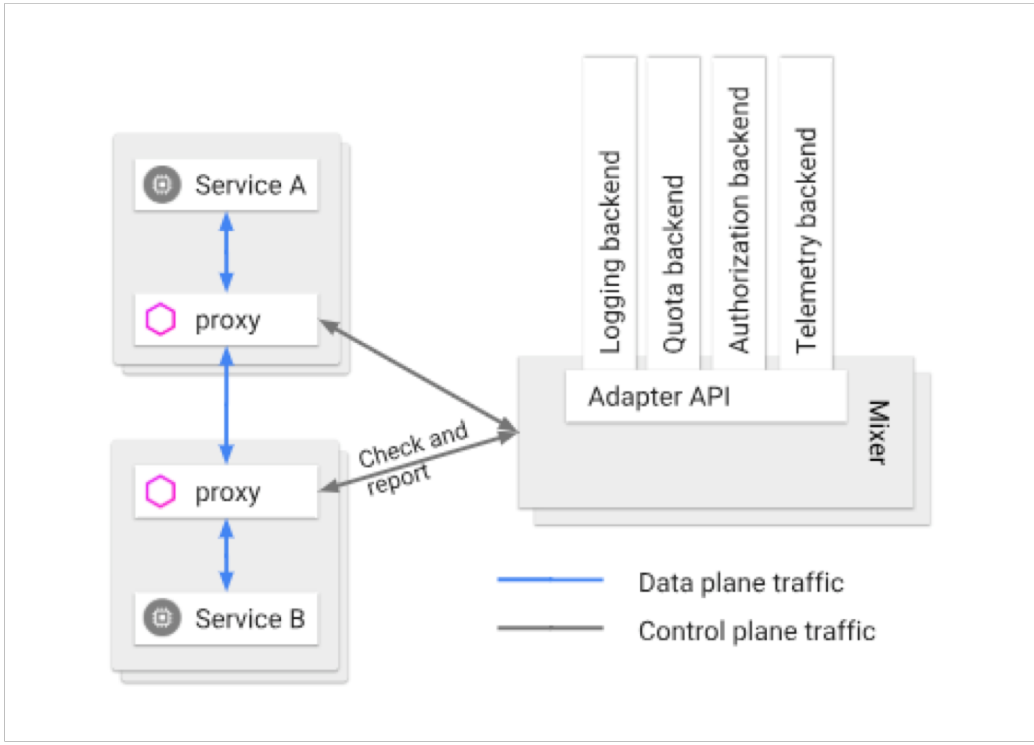
Istio: Pilot

- Control plane for distributed Envoy instances
- Configures Istio configurations and pushes to other system components
- System of record for the service mesh
- Exposes API for service discovery, load balancing, etc.



Istio: Mixer

- Responsible for providing policy controls
- Handles telemetry collection (Grafana, Prometheus)
- Envoy sidecar calls Mixer before each request to perform precondition checks and report telemetry



Access Cloud Metadata

- Simple SSRF can lead to Cloud Metadata leak
- Using curl we can hit the AWS Metadata API endpoint from a pod and depending on the configuration, sensitive data may be returned
- http://169.254.169.254/latest/meta-data/iam/security-credentials/IAM_USER_ROLE_HERE

#KubernetesSecurityTip: Use a tool like KIAM or Kube2IAM to limit access to the AWS Metadata API. Better yet, apply a NetworkPolicy to stop traffic outbound.

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
 name: deny-all
 namespace: default
spec:
 podSelector: {}
 egress:
 - to:
   - podSelector:
       matchLabels:
         k8s-app: kube-dns
   - ports:
     - protocol: UDP
     port: 53
   policyTypes:
   - Ingress
   - Egress
```

deny-all.yaml

Attack: Unprotected Kubelet API

- The Kubelet handles Master <-> Node communication
- By default, the Kubelet API allows for unauthenticated access to ports **10255** (read-only) and **10250** (read / write)
- If a user has network access to your nodes the Kubelet API may be exposed

#KubernetesSecurityTip: This is a big deal and is not trivial to address. Some bootstrap tools enable certificate authentication between the master and nodes by default. Some don't. YMMV.

Kubernetes Secrets

Kubernetes Secrets

- Kubernetes Secret objects are designed to store small amounts of sensitive data such as API keys, tokens, or passwords
- Secrets are only sent to a node if a pod on that node requires it
- Secrets may be exposed to a Pod as a mounted volume or as an Environment Variable

Kubernetes Secrets

- Secret data on nodes is stored in tmpfs volumes and not stored at rest on disk (technically)
- Communication between api-server to Kubelet is encrypted with TLS
- Secrets are tied to a particular namespace and must be encoded using base64

```
$ echo -n "admin" | base64
YWRtaW4=
$ echo -n "1f2d1e2e67df" | base64
MWYyZDFlMmU2N2Rm
```

Kubernetes Secrets Risks

- Secrets are stored in plain text by default in etcd
- Very little separation of duties
- During etcd replication, secrets are sent in plaintext
- People ***still*** love pushing secrets to version control
- Modifying secrets requires rolling out new objects

Kubernetes Secrets

Which is the most secure way to pass secrets to a pod?

1. Pass secrets as an environment variable
2. Mount volume in container that has secrets in a file
3. Build the secrets into the container image
4. Query a ”Secrets API” over your network
5. Other

Building Secrets into Container Images

- Access to image == access to secrets
  - Who has access to your images?
- Rotation becomes a new image build
- Secrets are likely stored in source code control ending up on laptops, cloud environments, etc.
- Chance of accidentally making the secrets “public” increases

Pass Secrets as Environment Variables

- Twelve-Factor App suggests this mechanism
- Passed into containers at runtime
- Can still end up checked in to source control via hardcoding in YAML
- Native Secrets in Kubernetes support this out of the box
- In-cluster RBAC needs to be tight to prevent misuse
- Watch out for secrets in logs and error messages
- Accessible using `docker inspect` or `kubectl describe`

Pass Secrets as Files

- Mount a volume in the pod that has a file with secret values usually as key-value pairs
- Your app needs to support this
- Writing to a temporary filesystem prevents secrets from being written to disk (auditors <3 this)
- Make sure your app doesn't just rewrite this file elsewhere
- Not accessible using `docker inspect` or `kubectl describe`

Rotating and Revoking Secrets

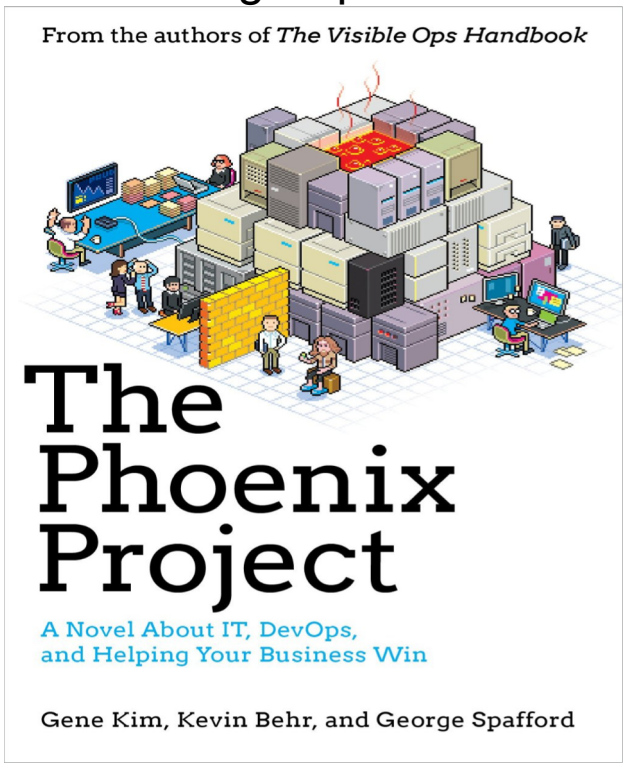
- Rotation and revocation depend on your threat model and internal security policies
- You need a mechanism in place no matter what
- Pods may need restarted for app to recognize new secrets
- If using mounted volumes for secrets, pods do not need to be restarted
- **Your app should know how to handle rotation and revocation gracefully**

Exploring container security: Encrypting Kubernetes secrets with Cloud KMS

Where do we go from here?

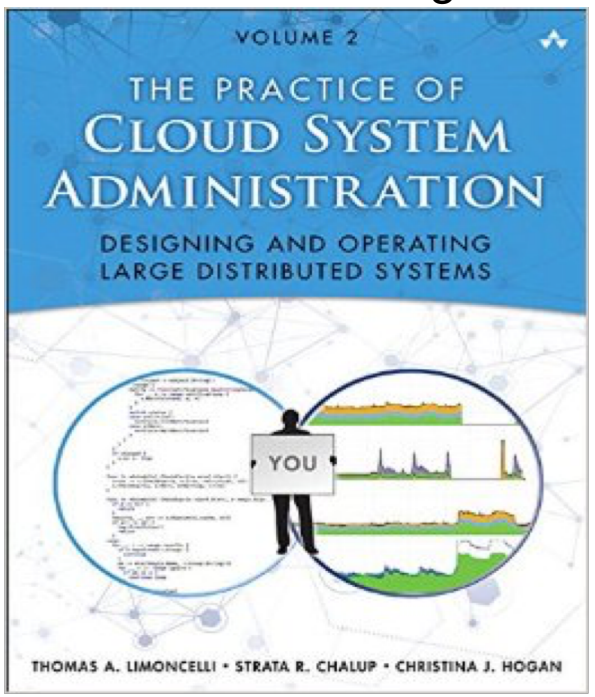
The Phoenix Project

Gene Kim, Kevin Behr and George Spafford



The Practice of Cloud System Administration

Thomas A. Limoncelli, Strata R. Chalup, Christina J. Hogan



It's been a pleasure.

jmesta@manicode.com

Jimmy Mesta Secure Coding Instructor *www.manicode.com*