DevSecOps Kubernetes Pipeline Workshop

From @ibm, @tetrateio, and @controlplaneio

IBM Cloud

Michael Hough

Developer, IBM Cloud Container Registry

Maintainer, Portieris

🐦 @molepigeon

Innovate with IBM’s Cloud-Native Platform

Functions

Kubernetes

CLOUD FOUNDRY

Istio

Event Streams

Watson

Cloudant

Databases

Continuously delivering innovation to your application

Client Success Stories

MAERSK

Always-On & Automation

IBM Cloud builds on open-source to relieve the pains of security, scale, software, & infrastructure management

Tetrate

Liam White

Software Engineer, Tetrate

Core contributor, Portieris & Istio

@liamandrewwhite

controlplane

Pi Unnerup

Infrastructure Engineer, ControlPlane

OS work on Netassert, Kubesec.io

@piunnerup

controlplane

Andrew Martin

Security Engineer, ControlPlane

OS work on Kubernetes & Istio

@sublimino

Preflight Checks

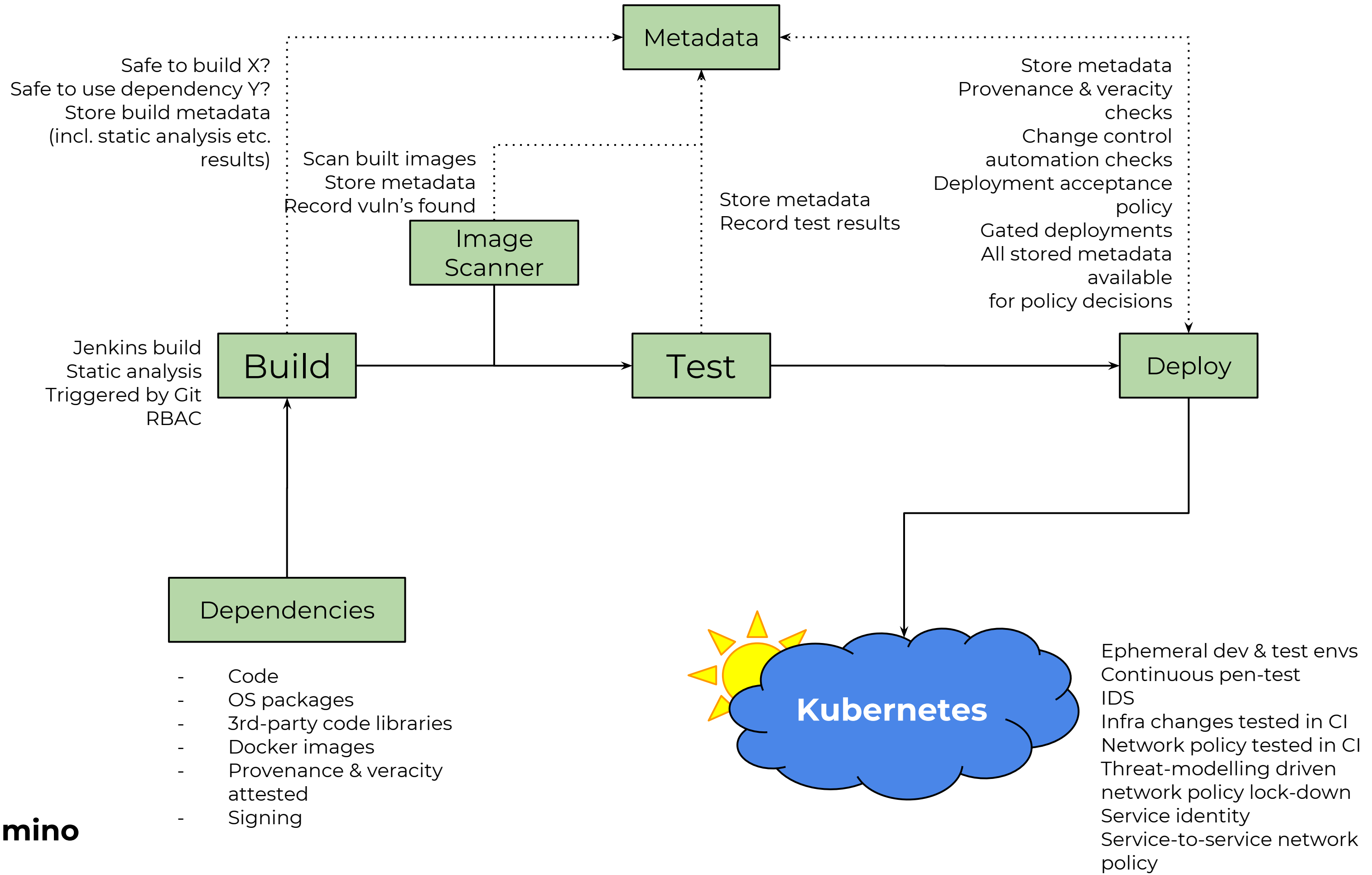
Preparation

- Navigate to https://goo.gl/DKXRnb and follow the instructions
  - Ensure minikube is running (tested on v1.10+)
  - `docker pull sublimino/alpine-base:insecure`
  - Ensure you can `kubectl get pods`
  - Clone the repo locally
  - Run through **00-Prerequisites** and **01-Installing-Harbor**

Secure Pipelines

/ROOT:#
STOP
#HaCk3r
RIGHT
WRONG
FRAGILE
#YOU HAVE BEEN#
HACKED!!
TO BE
FREE
I read your email

Secure Cloud-Native Delivery



Safe to build X?
Safe to use dependency Y?
Store build metadata
(incl. static analysis etc. results)

Scan built images
Store metadata
Record vuln's found

Store metadata
Record test results

Store metadata
Provenance & veracity checks
Change control automation checks
Deployment acceptance policy
Gated deployments
All stored metadata available
for policy decisions

Jenkins build
Static analysis
Triggered by Git
RBAC

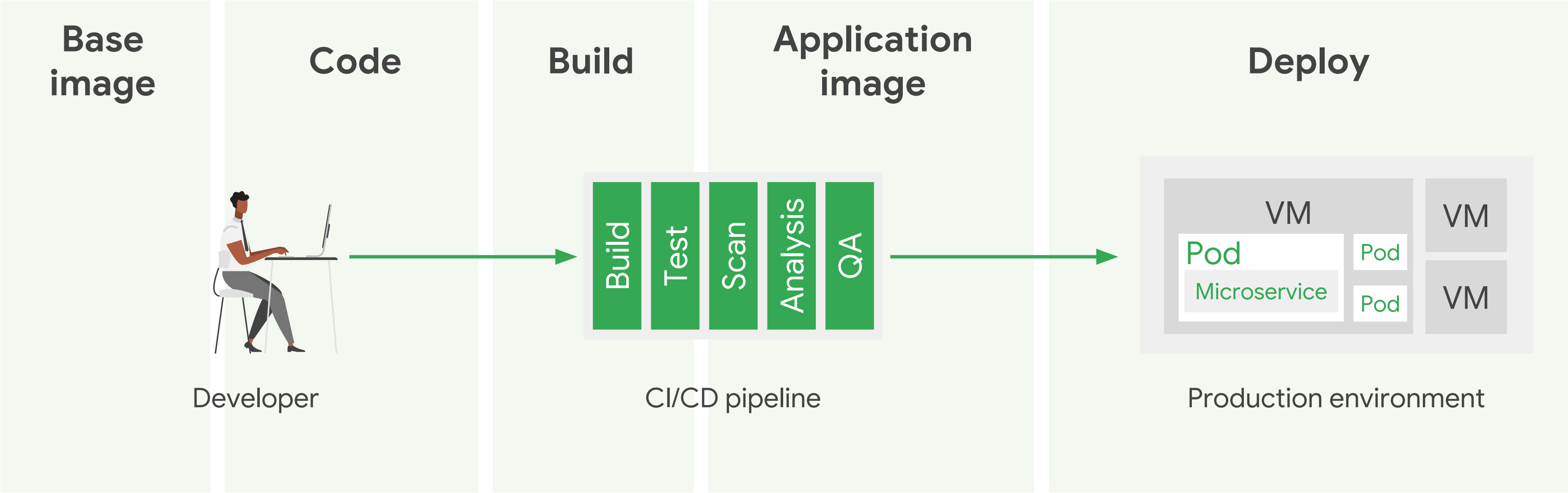
- Code
- OS packages
- 3rd-party code libraries
- Docker images
- Provenance & veracity attested
- Signing

Ephemeral dev & test envs
Continuous pen-test
IDS
Infra changes tested in CI
Network policy tested in CI
Threat-modelling driven network policy lock-down
Service identity
Service-to-service network policy

@sublimino

controlplane

Stages of the CDLC (Container Delivery Lifecycle)



Base image

Code

Build

Application image

Deploy

Build

Test

Scan

Analysis

QA

VM

Pod

Microservice

Pod

Pod

VM

VM

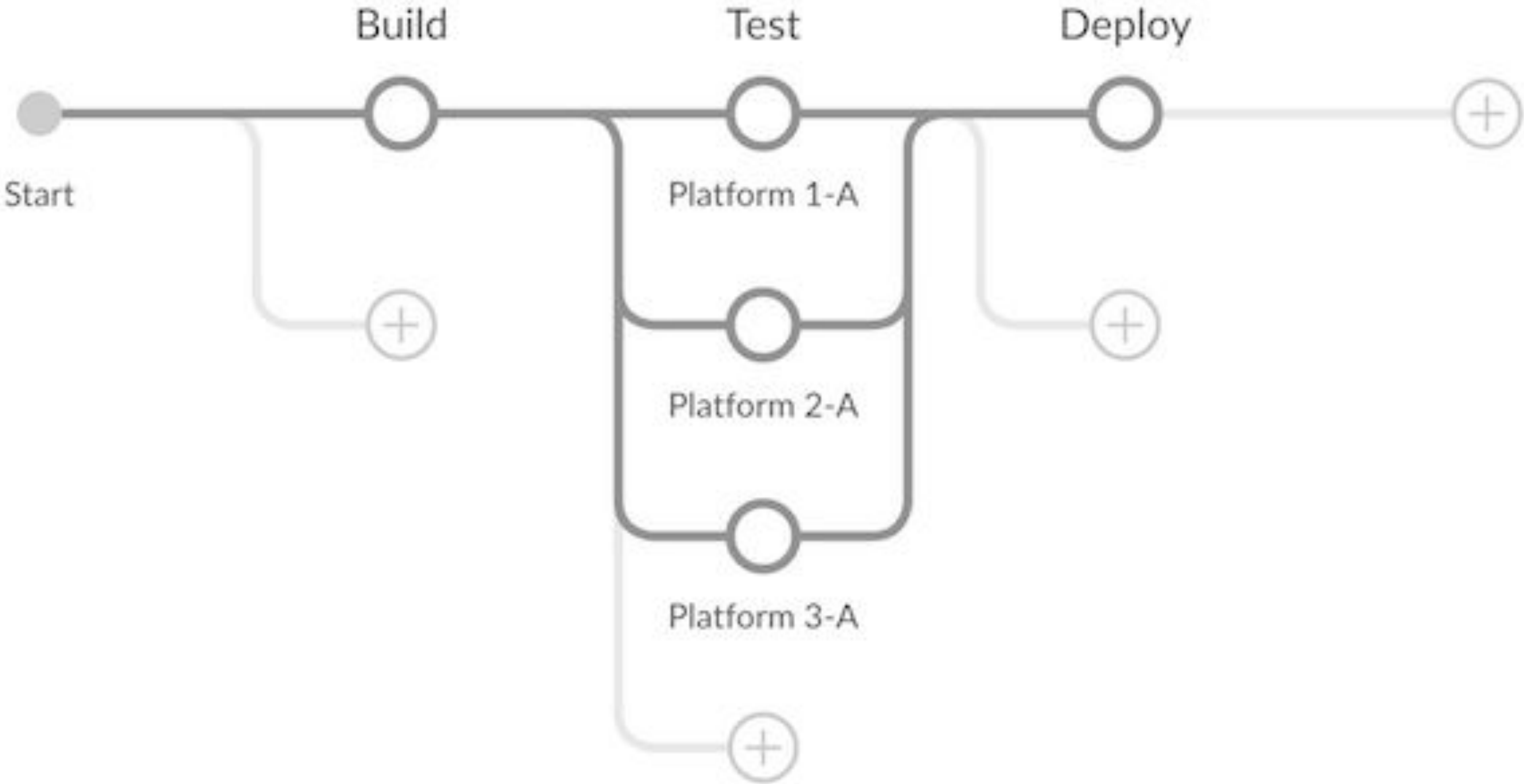
Developer

CI/CD pipeline

Production environment

“I find your lack of security disturbing.”

https://www.slideshare.net/FranklinMosley/devsecops-a-new-hope-for-security-in-cicd


Build
Test
Deploy
Start
Platform 1-A
Platform 2-A
Platform 3-A

Open-source supply chain today

Base image

Images: Docker Distribution (Hub)

Code

Updates: TUF, Notary

Build

Pipeline metadata: Grafeas, in-toto

Application image

Vulnerability scanning: Clair, Micro Scanner, Anchore Open Source Engine

Deploy

Admission control: K8s admission controllers, Kritis, Portieris

@sublimino

Open-source supply chain today

Base image

Images: Docker Distribution (Hub)

Code

Updates: TUF, Notary

TUF

notary

Build

Pipeline metadata: Grafeas, in-toto

Grafeas

Application image

Vulnerability scanning: Clair, Micro Scanner, Anchore Open Source Engine

clair

aqua MicroScanner

Deploy

Admission control: K8s admission controllers, Kritis, Portieris

PORTIERIS

KUBESEC.IO

Build Flow

- Build image (base image from Docker Hub)
- Assert absence of vulnerabilities in image (Harbor)
- Cryptographically sign image for later verification
- Push image to container registry
- Attempt to deploy image to cluster
- Verify image has been signed with an admission controller
- Reject images that have not followed due process and organisational policy

01 - Installing Harbor

Harbor

- Container image registry (a “self-hosted Docker hub”)
- New CNCF project
- Capable of running inside a cluster for inception-esque self-referential image pulls

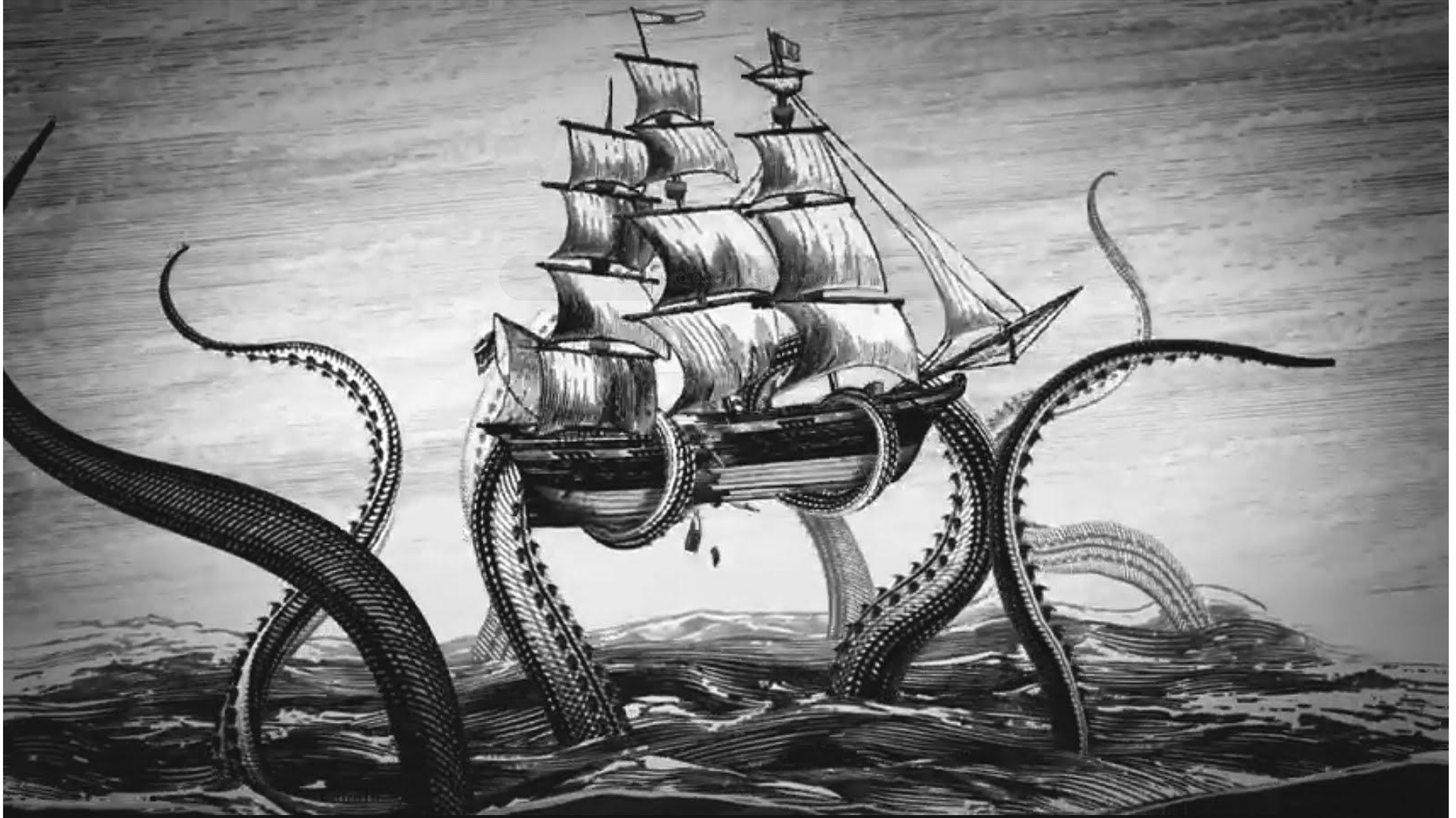
Harbor

Notary
Cryptographic image signing

Docker Distribution
Container registry

Clair
Image vulnerability scanning

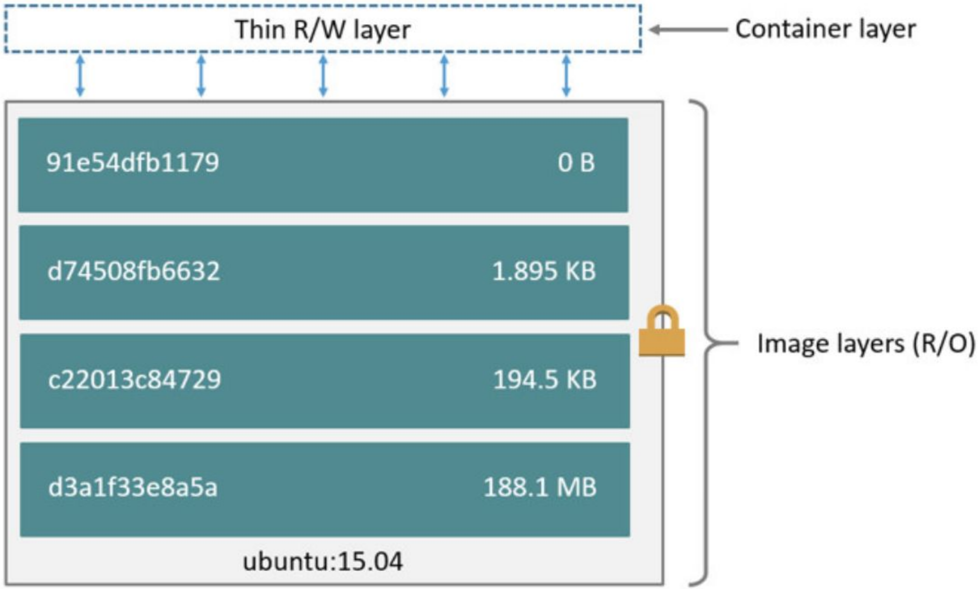
Vulnerable Images



What Can Image Scanning Detect?

- This depends upon the depth of the tool
- Some will just scan installed operating system package manager versions
- Others will check filesystem permissions for all entities, extra binaries, secrets, policies etc.

Image vulnerability scanning approaches



https://sysdig.com/blog/container-security-docker-image-scanning/

- Components to scan: package-level vs. code-level
  - OS packages
  - App library packages
  - JARs, WARs, TARs, etc.
  - Malware
  - Misconfigurations, e.g., secrets
- Scan type
  - Layer-by-layer
  - UnionFS top layer only

Clair vs. MicroScanner vs. Anchore

	Scanning depth	OS covered	Maintainer
clair	Packages	Alpine, CentOS, Debian, Oracle Linux, RHEL, Ubuntu	CoreOS
aqua MicroScanner	Packages		Aqua Security
anchore	Packages, files, software artifacts		Anchore

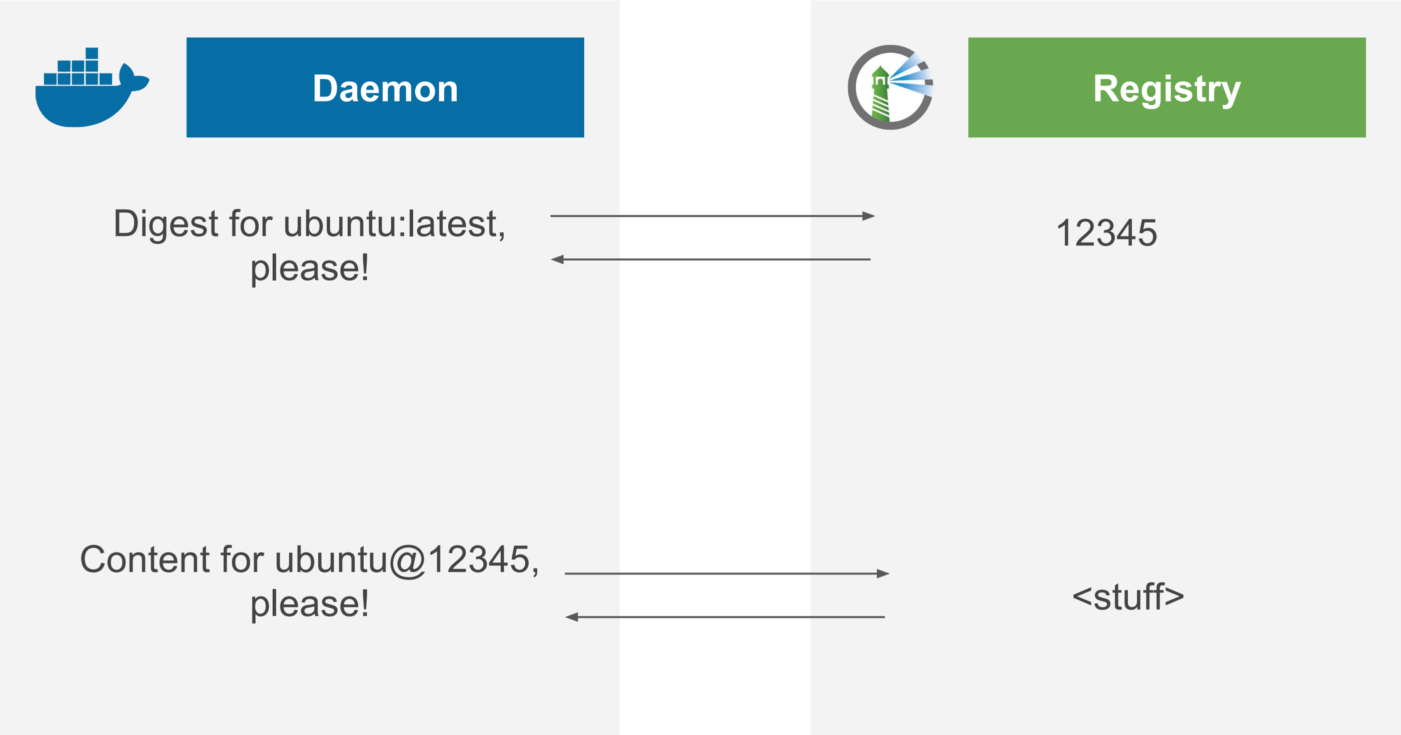
02 - Vulnerability Scanning

Part 2 - recap

- Firstly - we couldn’t deploy anything! Harbor will not allow us to pull vulnerable images
- This was the intended consequence of attempting to ship a CVE-laden image to production
- CVEs are a likely way for an attacker to being their assault on your systems
- Never ship CVEs to production

Notary

Notary



Daemon

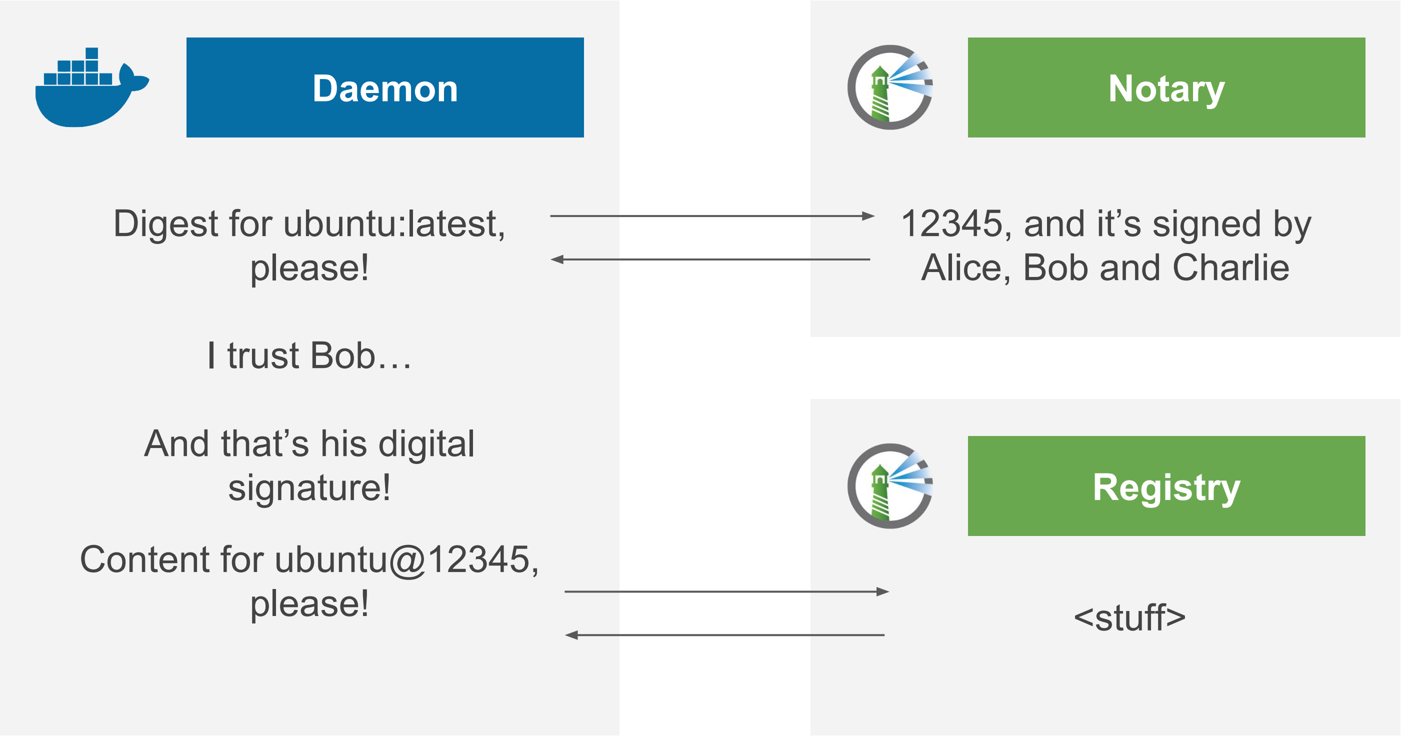
Registry

Digest for ubuntu:latest, please!

12345

Content for ubuntu@12345, please!

<stuff>



Daemon

Notary

Registry

Digest for ubuntu:latest, please!

12345, and it’s signed by Alice, Bob and Charlie

I trust Bob…

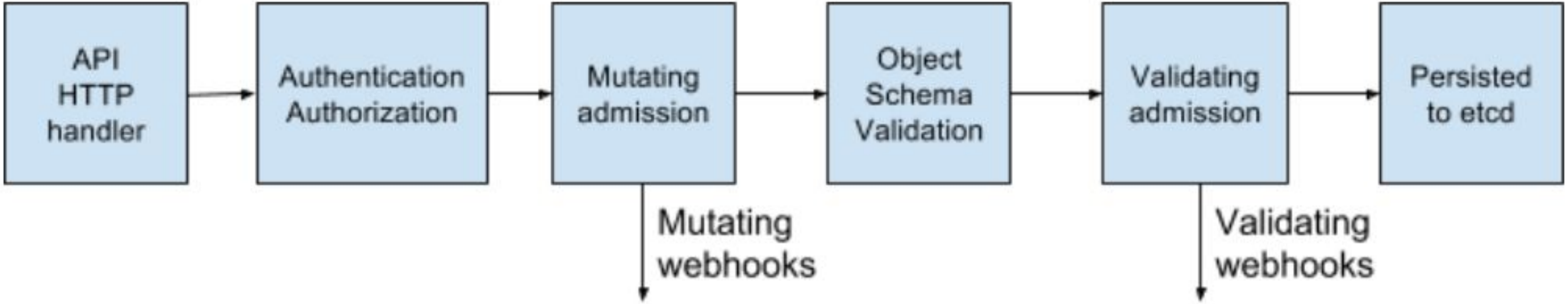
And that’s his digital signature!

Content for ubuntu@12345, please!

<stuff>

Admission control

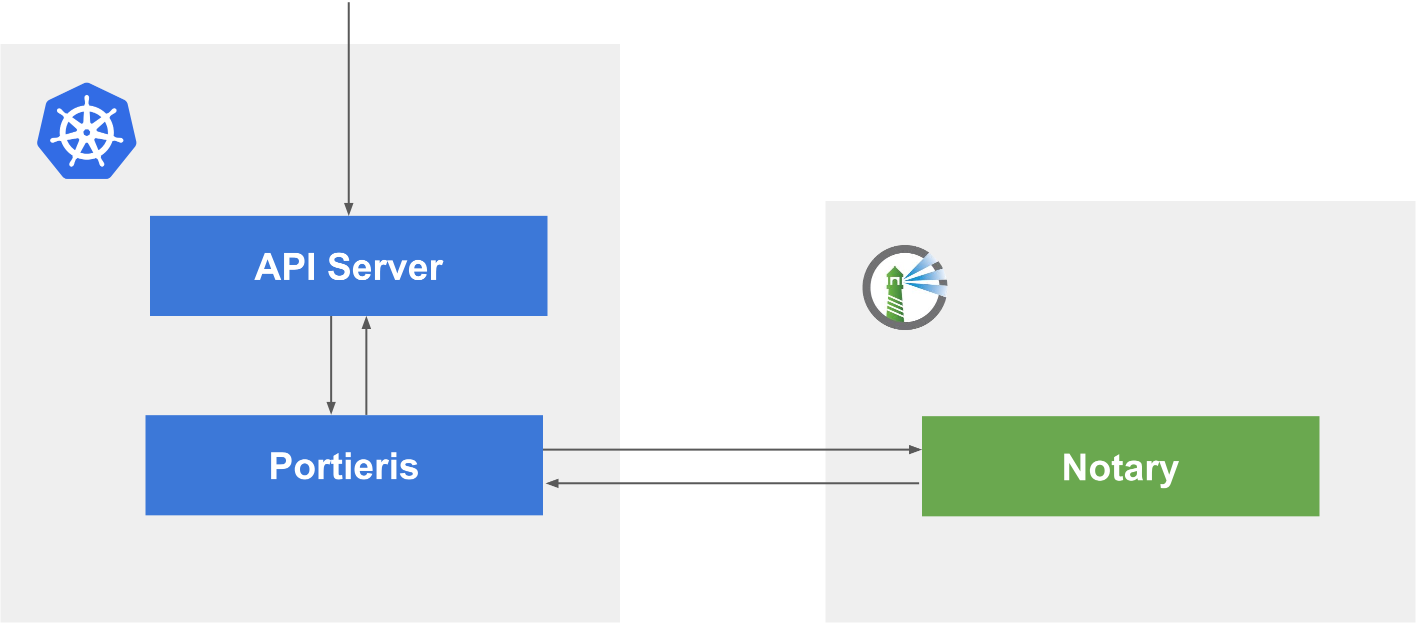
Extensible Admission Controllers



http://blog.kubernetes.io/2018/01/extensible-admission-is-beta.html

controlplane

PORTIERIS


API Server
Portieris
Notary

```
image: ibmcom/portieris:0.5.1
```

```
image: ibmcom/portieris@sha256:19b6e9df327....
```

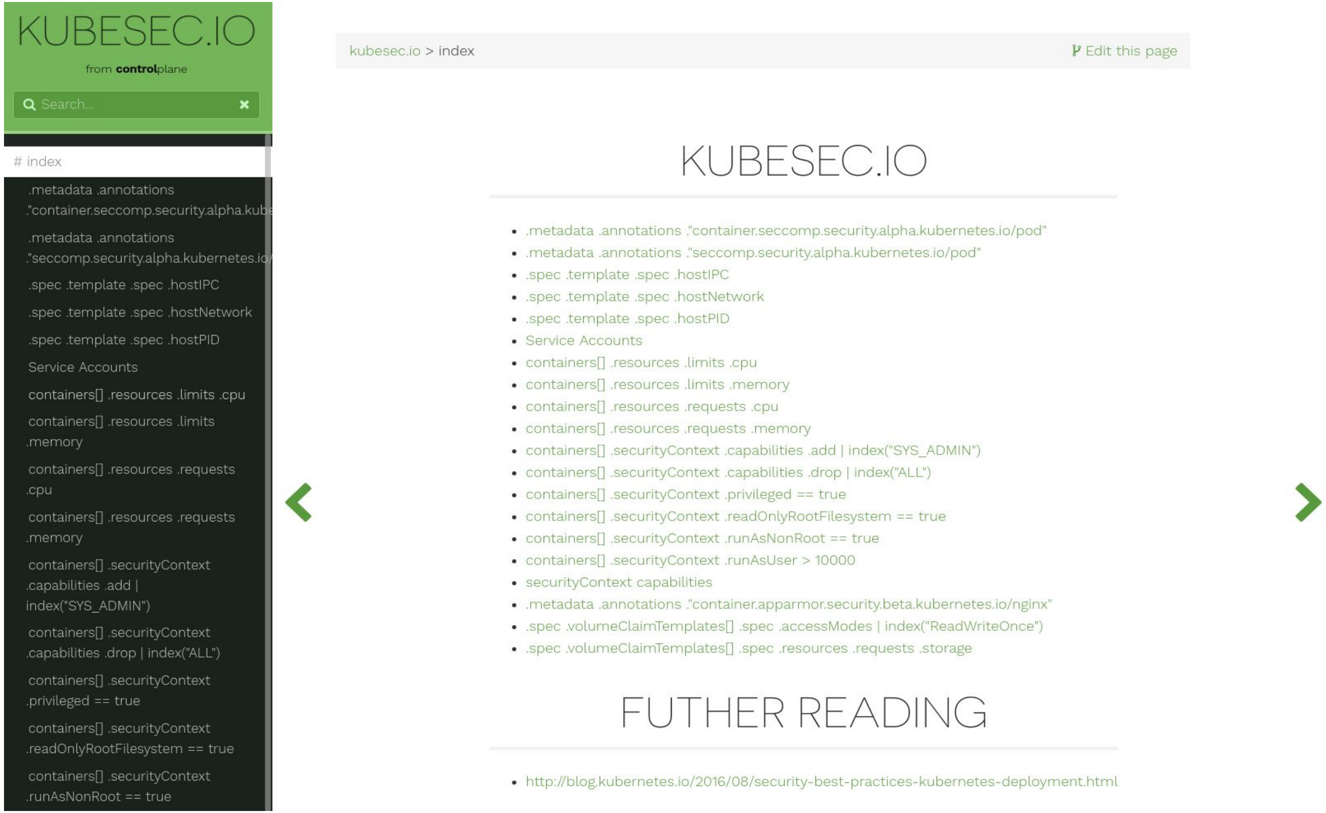
03 - Image Signing

Part 3 - was that enough?

- Scanning for vulnerabilities is important - but only makes any sense if that same image is deployed to production
- Asserting that the image that runs in production contains what you think it does is another basic security precaution that is too-often overlooked
- This security measure can prevent the compromise of access to your container registry from compromising production

Kubesec

kubesec.io - risk score for K8S YAML



KUBESEC.IO

from controlplane

Search...

index

.metadata .annotations ."container.seccomp.security.alpha.kube

.metadata .annotations ."seccomp.security.alpha.kubernetes.io/

.spec .template .spec .hostIPC

.spec .template .spec .hostNetwork

.spec .template .spec .hostPID

Service Accounts

containers[] .resources .limits .cpu

containers[] .resources .limits .memory

containers[] .resources .requests .cpu

containers[] .resources .requests .memory

containers[] .securityContext .capabilities .add | index("SYS_ADMIN")

containers[] .securityContext .capabilities .drop | index("ALL")

containers[] .securityContext .privileged == true

containers[] .securityContext .readOnlyRootFilesystem == true

containers[] .securityContext .runAsNonRoot == true

kubesec.io > index

Edit this page

KUBESEC.IO

- .metadata .annotations ."container.seccomp.security.alpha.kubernetes.io/pod"
- .metadata .annotations ."seccomp.security.alpha.kubernetes.io/pod"
- .spec .template .spec .hostIPC
- .spec .template .spec .hostNetwork
- .spec .template .spec .hostPID
- Service Accounts
- containers[] .resources .limits .cpu
- containers[] .resources .limits .memory
- containers[] .resources .requests .cpu
- containers[] .resources .requests .memory
- containers[] .securityContext .capabilities .add | index("SYS_ADMIN")
- containers[] .securityContext .capabilities .drop | index("ALL")
- containers[] .securityContext .privileged == true
- containers[] .securityContext .readOnlyRootFilesystem == true
- containers[] .securityContext .runAsNonRoot == true
- containers[] .securityContext .runAsUser > 10000
- securityContext capabilities
- .metadata .annotations ."container.apparmor.security.beta.kubernetes.io/nginx"
- .spec .volumeClaimTemplates[] .spec .accessModes | index("ReadWriteOnce")
- .spec .volumeClaimTemplates[] .spec .resources .requests .storage

FUTHER READING

- http://blog.kubernetes.io/2016/08/security-best-practices-kubernetes-deployment.html

controlplane

kubesec.io - example insecure pod

```
{
  "score": -30,
  "scoring": {
    "critical": [{
      "selector": "containers[] .securityContext .privileged == true",
      "reason": "Privileged containers can allow almost completely unrestricted host access"
    }],
    "advise": [{
      "selector": "containers[] .securityContext .runAsNonRoot == true",
      "reason": "Force the running image to run as a non-root user to ensure least privilege"
    }, {
      "selector": "containers[] .securityContext .capabilities .drop",
      "reason": "Reducing kernel capabilities available to a container limits its attack surface",
      "href": "https://kubernetes.io/docs/tasks/configure-pod-container/security-context/"
    },
   ...
```

04 - More Admission Control

Part 4 - minimum viable security

- We have
  - Verified the contents of an image are not insecure
  - Signed the image to confirm we have tested it
  - Prevented unsigned images from being deployed to production
- These are the building blocks of a secure pipeline
  - But only focus on the contents of the image and not its runtime configuration
- PodSecurityPolicy and NetworkPolicy should be use to limit the behaviour of the application at runtime
- Further admission controllers can be added to enhance security

Threat Model

- Attacks wholly or partially mitigated:
  - Container image and application supply chain with known CVEs
  - Theft of users' container registry credentials
  - Some build server compromises
- Extant risk:
  - Compromised user or insider threat
  - Zero day vulnerabilities
  - ...the rest of the Kubernetes attack surface!

Summary