DevOps and Container Security

Mike Bartock
IT Specialist
NIST

Paul Cichonski
Cloud Architect
Lancop

John Morello
Chief Technology Officer
Twistlock

Raghu Yeluri
Principal Engineer
Intel

Certain commercial entities, equipment, or materials may be identified in this document in order to describe an experimental procedure or concept adequately. Such identification is not intended to imply recommendation or endorsement by NIST, nor is it intended to imply that the entities, materials, or equipment are necessarily the best available for the purpose.

Agenda

- Introductions
- Purpose of the Panel
- Panelist Container Work
  - Intel
  - Twistlock
  - Lancope
- Discussion on container security and applications
- Questions from the Audience

Purpose

- Introduction to containers and their uses
- Different methods of security for containers
- Discussion of what industry is doing

Trusted Containers

Raghu Yeluri

Principal Engineer, Lead Cloud Security Architect

Datacenter Group, Intel Corporation

Legal Information

Containers

Lightweight, fast, disposable virtual environments

- App portability, maintenance and deployment.

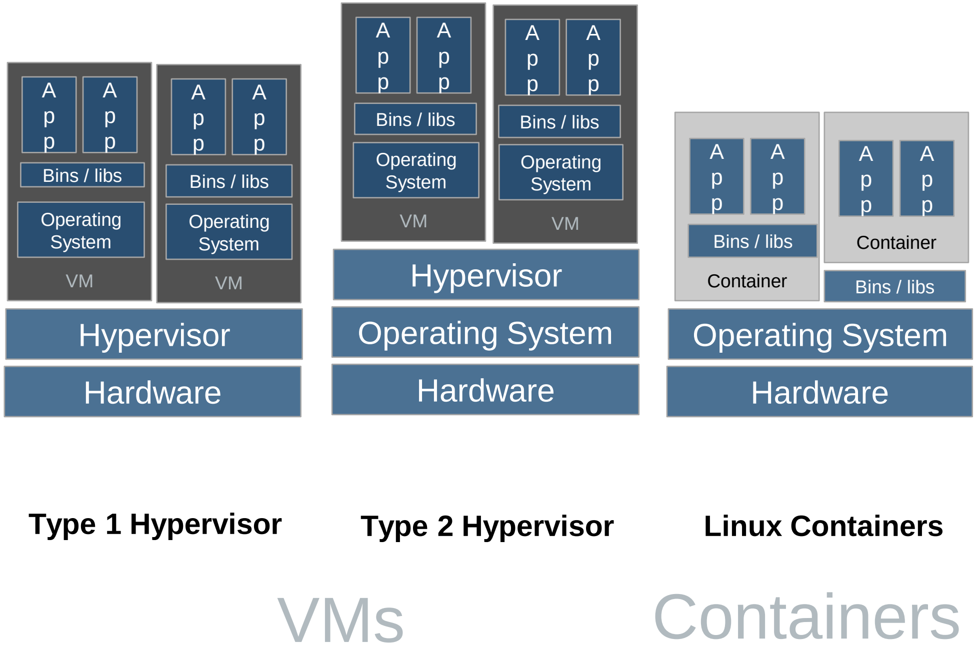
Technically:

- Set of processes running atop shared kernel
- Isolated from rest of the system (limitations)

From a distance… looks like a VM (SSH, root access, eth0, mount file systems)

Have been around for 10+ years (Solaris* containers, Linux* Containers..)

Efficient way to build, ship, run, deliver apps



Docker* Containers commoditized Linux Containers

What is Docker*?

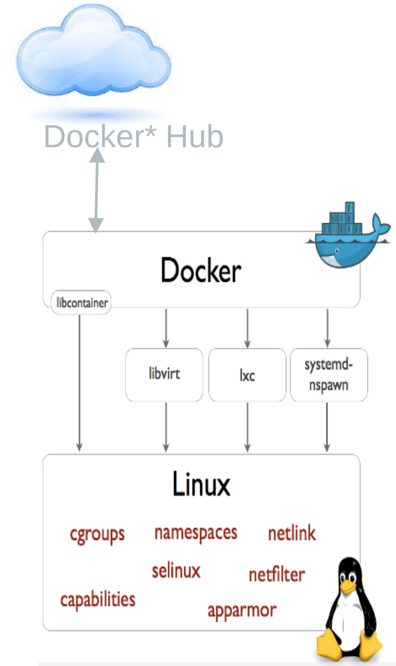
Lightweight, open source engine for creating, deploying containers

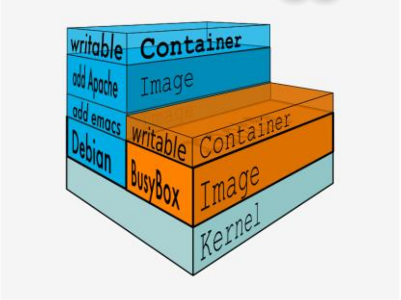
- Provides work flow for running, building and containerizing apps.
- Separates apps from where they run; enables Micro-services; scale by composition
- Underlying building blocks: Linux* kernel's namespaces (isolation) + cgroups (resource control) + ..

Components of Docker*

- Docker Engine: Runtime for running, building Docker containers
- Docker Repositories(Hub): SaaS for sharing/managing images
- Docker Images (layers)
  – Images hold Apps. Shareable snapshot of software. Container is a running instance of image.

Orchestration: OpenStack*, Docker Swarm, Kubernetes*, Mesos*, CoreOs Tectonic, Fleet





Docker* Layers

Container Security – Key Customer Asks

1. Docker* Host Integrity
   - Do you **trust** the Docker daemon?
   - Do you trust the Docker host has booted with Integrity?
2. Docker Container Integrity verification
   - Who wrote the container image? Do you **trust** the image? Did the right Image get launched?
3. Runtime Protection of containers & Enhanced Isolation
   - How can Intel help with runtime Integrity, Isolation?
4. Intelligent orchestration – OpenStack as singular control plane for Trusted VMs and Containers

Intel's Focus: Hardware-based Integrity Assurance for Containers – Trusted Docker Containers

Trusted Docker* Containers – 3 Focus Areas

Launch Integrity of Docker* Host & Docker Engine

Integrity of Docker Images & Containers

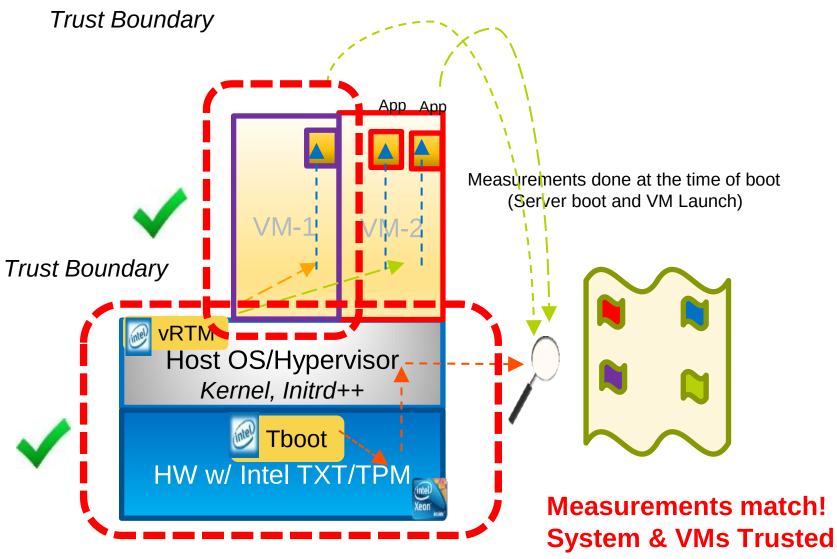
Looking ahead…Runtime Integrity of Docker Host, H/w-based enhanced Isolation

Trusted VMs - Summary

Launch VMs on Servers with demonstrated Boot Integrity – **Trusted Boot**

Chain of Trust to VMs – **Trusted VMs**

Control where Trusted VMs are launching and migrating: **Boundary Control of VMs**



Trusted Platform Module (TPM)
Intel® Trusted Execution Technology (Intel® TXT)

Enable same model and use-cases for Trusted Containers

Trusted Docker* Containers - 1

Ensure Docker* Containers are launched on **Trusted Docker Hosts**

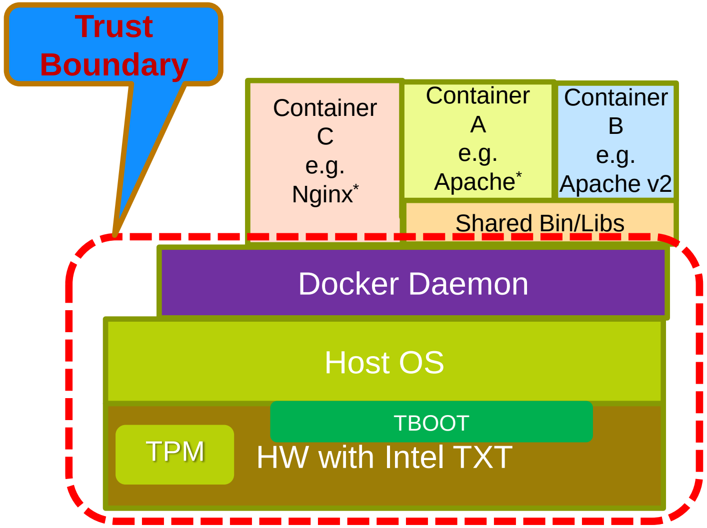
Boot-time integrity of the Docker Host

- **Measured** Launch of Boot Process and components with Intel® Trusted Execution Technology (Intel® TXT)

Docker daemon and associated components added to TCB and Measured

Chain of Trust: **H/W → FW → BIOS → OS → Docker Engine**

Remote attestation using Intel Cloud Integrity Technology (Attestation Authority)



Docker Host Platform Integrity

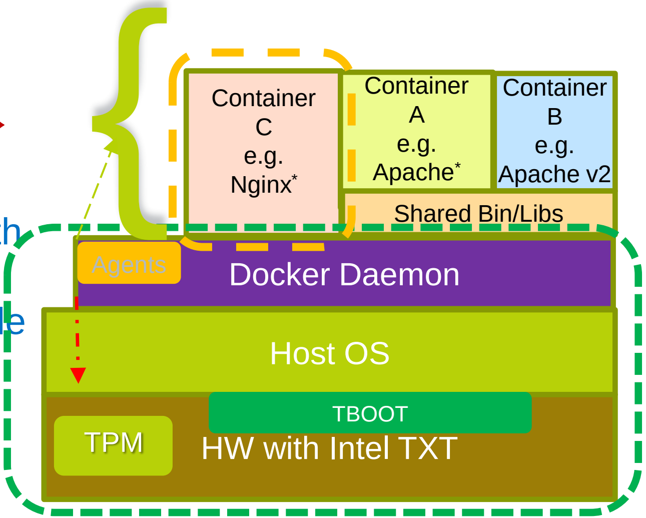
Assure and attest the Integrity of Docker host/platform

Trusted Docker* Containers - 2

Ensure that Docker* Images not tampered prior to Launch

Two Models:

1. Measure and verify Docker images, Chain of Trust: H/W → FW → BIOS → OS → Docker Engine → Docker image layers
2. Sign images in Docker Hub. Verify images signature prior to launch with root Cert signature that is 'Sealed' to Intel® Trusted Execution Technology (Intel® TXT) measurements in the Trusted Platform Module (TPM). – *Can work with Notary* - Docker Content Trust Model.*



Docker Host & Container Launch Integrity

Boundary Control/Geo-Tagging applies equally to Docker Containers as well for compliance needs

- Orchestrator determines location/boundary at launch time

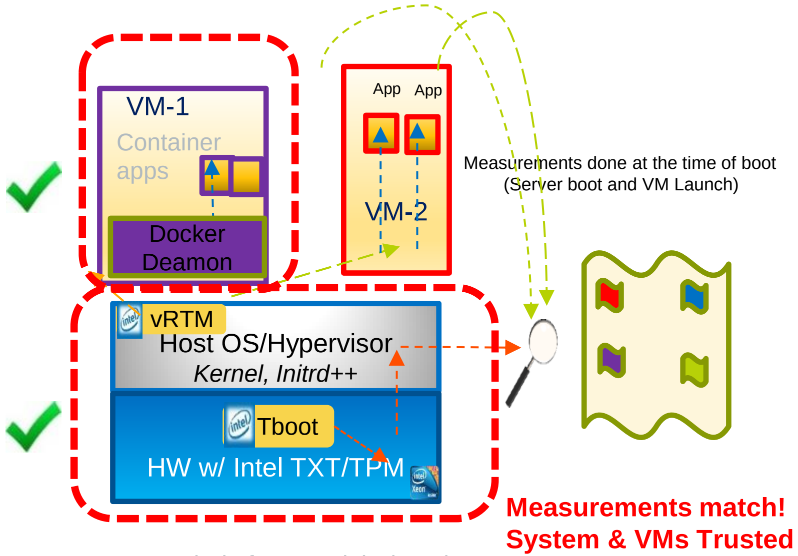
Assure and attest the Integrity of Docker images/containers

How about Docker* Containers in VMs?

Leverage Trusted VMs for asserting trust of the host, and the VMs.

Include Docker* Daemon as part of VMs TCB – **measure** and verify Docker Daemon as part of VM launch attestation.

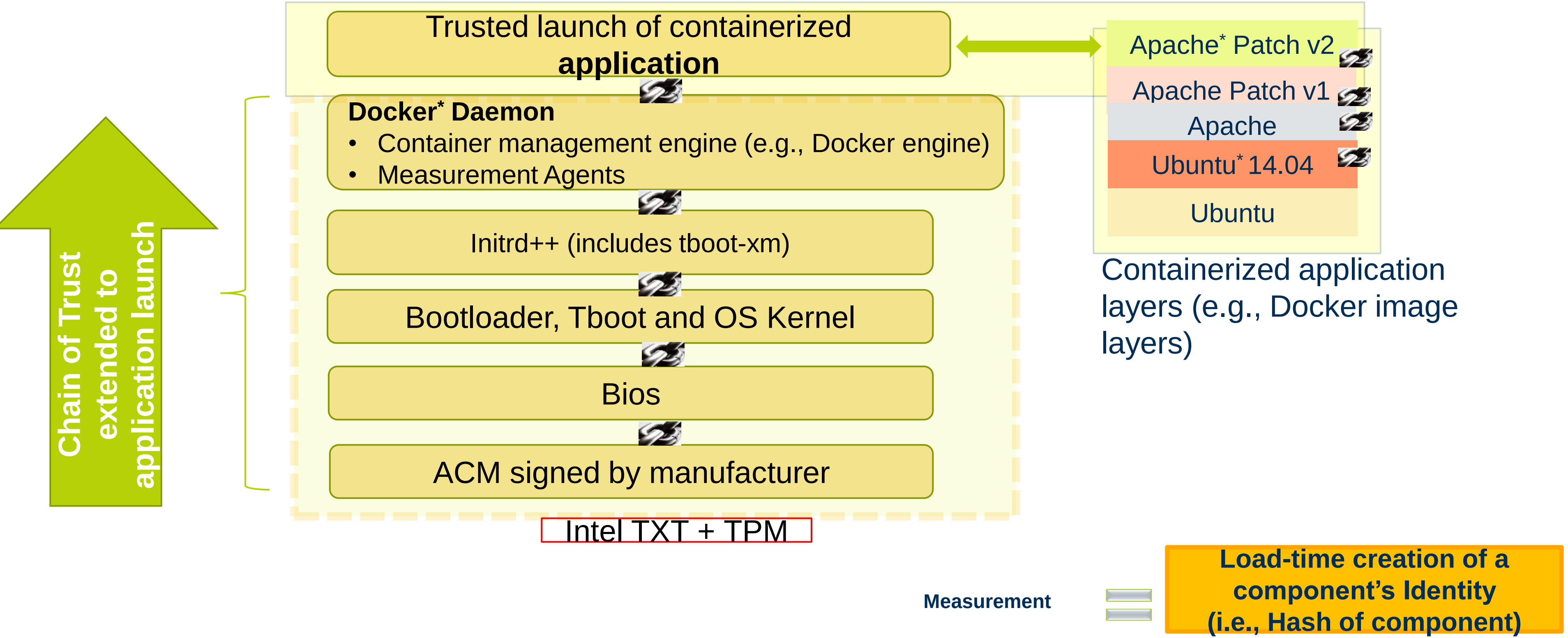
- Boot-time integrity of Host + VMM
- Integrity assurance of VM and Docker Daemon
- Chain of Trust: **H/W → FW → BIOS-OS/VMM-VM → Docker Engine**



Trusted Platform Module (TPM)
Intel® Trusted Execution Technology (Intel® TXT)

Assure and attest the Integrity of Host and the VM w/ Docker Engine

What is Measured for Trusted Containers



Intel® Trusted Execution Technology (Intel® TXT) chain of trust extended up the stack

Looking Ahead: Hardware-based Runtime Integrity

Intel® Kernel Guard Technology (Intel® KGT)

- Policy specification and enforcement framework
- Ensuring runtime integrity of kernel and platform assets

Extends launch-time integrity to run-time integrity

Based on a thin Intel VT-x (VMX-root) layer software component called **xmon**

- De-privileges OS
- Monitors/controls access to critical assets (CRs, MSRs, Memory Pages..)

Allows specification of policy from user-mode via *configfs*

- Policy describes assets to be monitored and actions to be taken when monitoring events occur

Policy can be locked down until next reboot

Intel KGT: Flexible, low overhead integrity framework; open source

Twistlock

“We’re going to build a software layer to make the internet programmable”

- Docker, DockerCon 2015

Docker is the best known example…

… but these trends are being driven by the growth of software across all industries and the need to rapidly build, iterate, and improve it

… all major IT players are investing

Some Growth Statistics

	June 2014	June 2015	Growth
Contributors	*460*	*1,300*	*183%*
Projects on GitHub	6,500	40,000	515%
Docker Job Openings (Indeed)	*2,500*	*43,000*	*1720%*
Dockerized Applications	14,500	150,000	934%
Boot2Docker Downloads	*225,000*	*3,500,000*	*1,456%*
Container Downloads	2,750,000	500,000,000	18,082%

dockercon 15

Fork me on GitHub

PHOTON™

VMware Photon OS™

Minimal Linux Container Host

Microsoft Azure

SALES 1-800-867-1389 | MY ACCOUNT

Why Azure | Products | Documentation | Pricing | Downloads | Partners | Blog | Community | Support

BLOG > ANNOUNCEMENTS , VIRTUAL MACHINES

New Windows Server containers and Azure support for Docker

Google Cloud Platform Blog

Product updates, customer stories, and tips and tricks on Google Cloud Platform

Kubernetes V1 Released - Cloud Native Computing Foundation to Drive Container Innovation

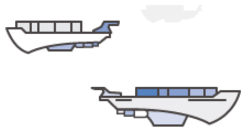
Amazon Web Services

PRODUCTS & SERVICES

Amazon EC2 Container Service ›

Product Details ›

Getting Started ›



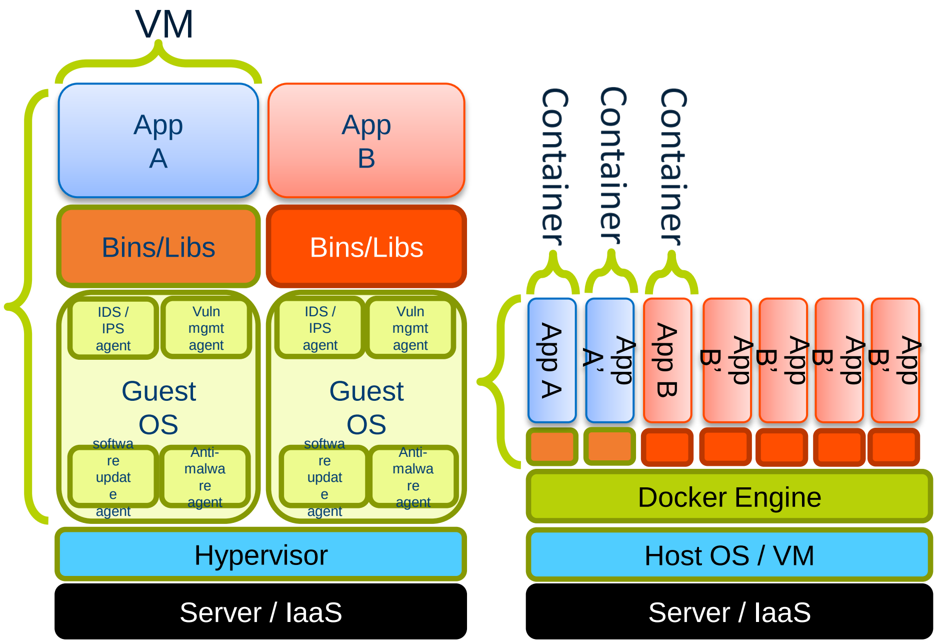
What are the challenges?

Containers don’t keep themselves up to date

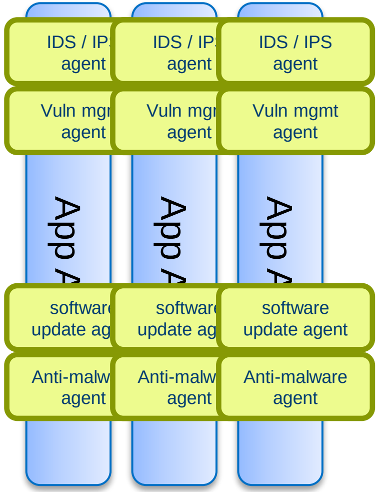
Many more containers but fewer tools for protecting them

Many more, and more diverse, places where your containers run

‘All or nothing’ administrative model



Why not existing solutions?



Containers are portable and minimal

Deployment is frictionless

Cramming containers full of agents and tools is antithetical to the model

What is Twistlock?

The first security solution built for containerized computing

… that secures the entire lifecycle of containerized apps…

… across all the environments they run in

A company that contributes back to the open source community

Defend your containers

Vulnerability management, with an intelligence stream of the latest CVEs and proactive defense

Advanced authorization capabilities, including Kerberos support and role based access control

Runtime defense, monitoring container memory space, storage, and networking to detect and block anomalous behaviors

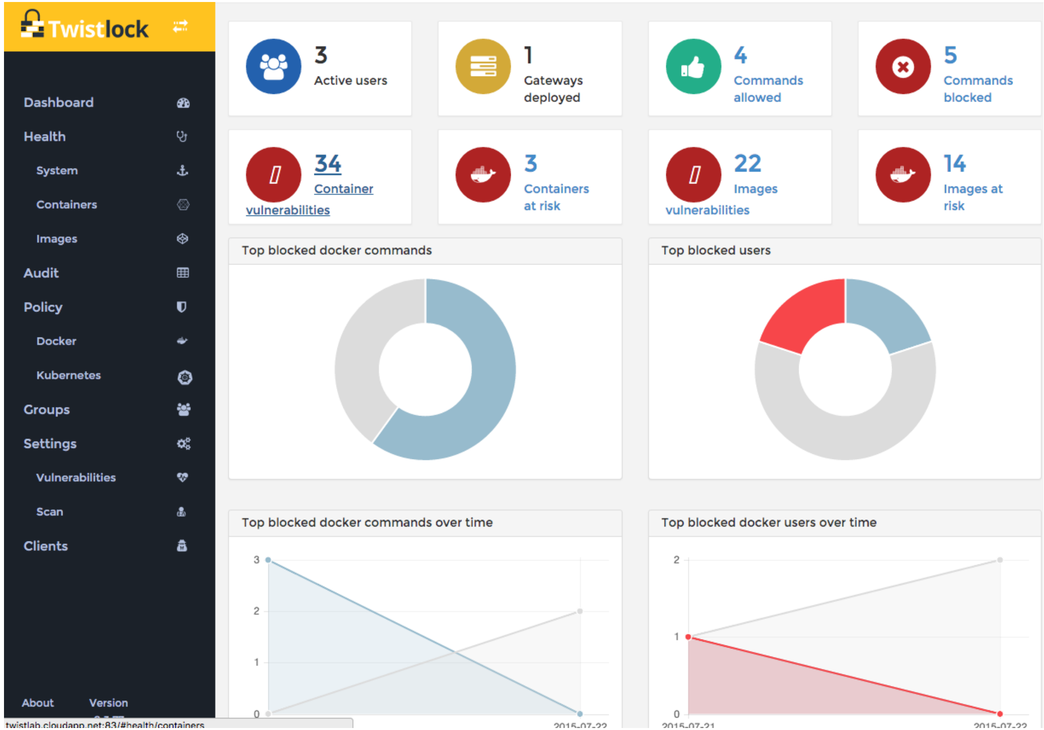
Purpose built

Agentless

Runs anywhere your containers run

API driven for continuous integration

Container Security Console



Configure

Monitor

Visualize risk

Vulnerability management “demo”

twisttest	Debian GNU/Linux 7 (wheezy)	index.docker.io	mongo	2.6.9	b4fa7b9347198d73	3	⚠	true	🔍
twisttest	Ubuntu 14.04.2 LTS	index.docker.io	docker/whalesay	latest	fb43[illegible]fc77c965	21	⚠	false	🔍

Block deployment of vulnerable images

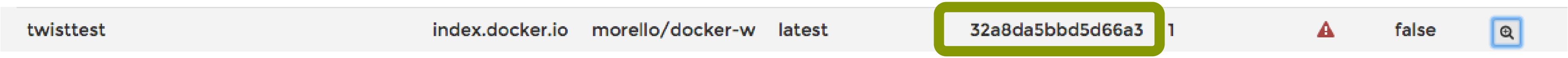
Tag resources and apply granular policies

Containers | Images | Host configuration | Daemon configuration | Daemon configuration files | Security Operations

Vulnerability id	Description	Block	Suppress alert
41	Image should be created with a user (CIS 4.1)	Off	Off
42	Use trusted base images for containers (CIS 4.2)	Off	Off
43	Image is not updated to latest	Off	Off
44	Image contains banned processes	Off	Off
45	Image contains process linked with a vulnerable dynamic library	Off	Off
46	Image contains vulnerable package versions	On	Off
47	Image contains vulnerable jar versions	Off	Off
48	Image contains vulnerable ruby gem versions	Off	Off

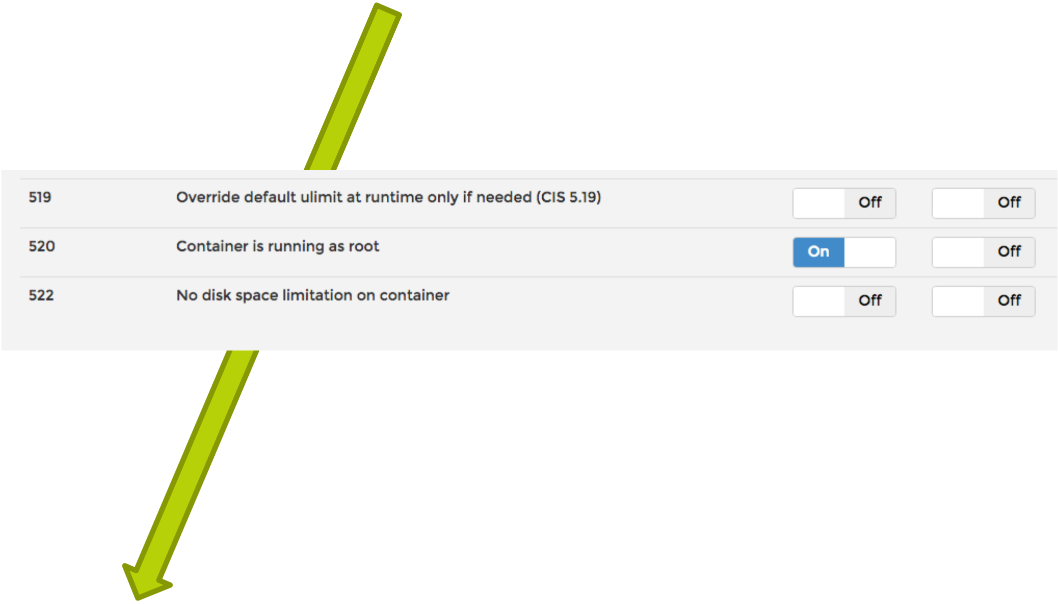
```
test2@twisttest:~/.docker$ docker -H :9998 --tlsverify run -[illegible] b4fa7b9347198d73
Error response from daemon: [Twistlock] Operation blocked. N[illegible]7ac80a0109' violates the policy '[46] Distro package 'openssl' version '1.0.1e-2+deb7u16'
is used (has vulnerabilities identified in 'CVE-2015-1791')'
test2@twisttest:~/.docker$
```

Security hardening “demo”

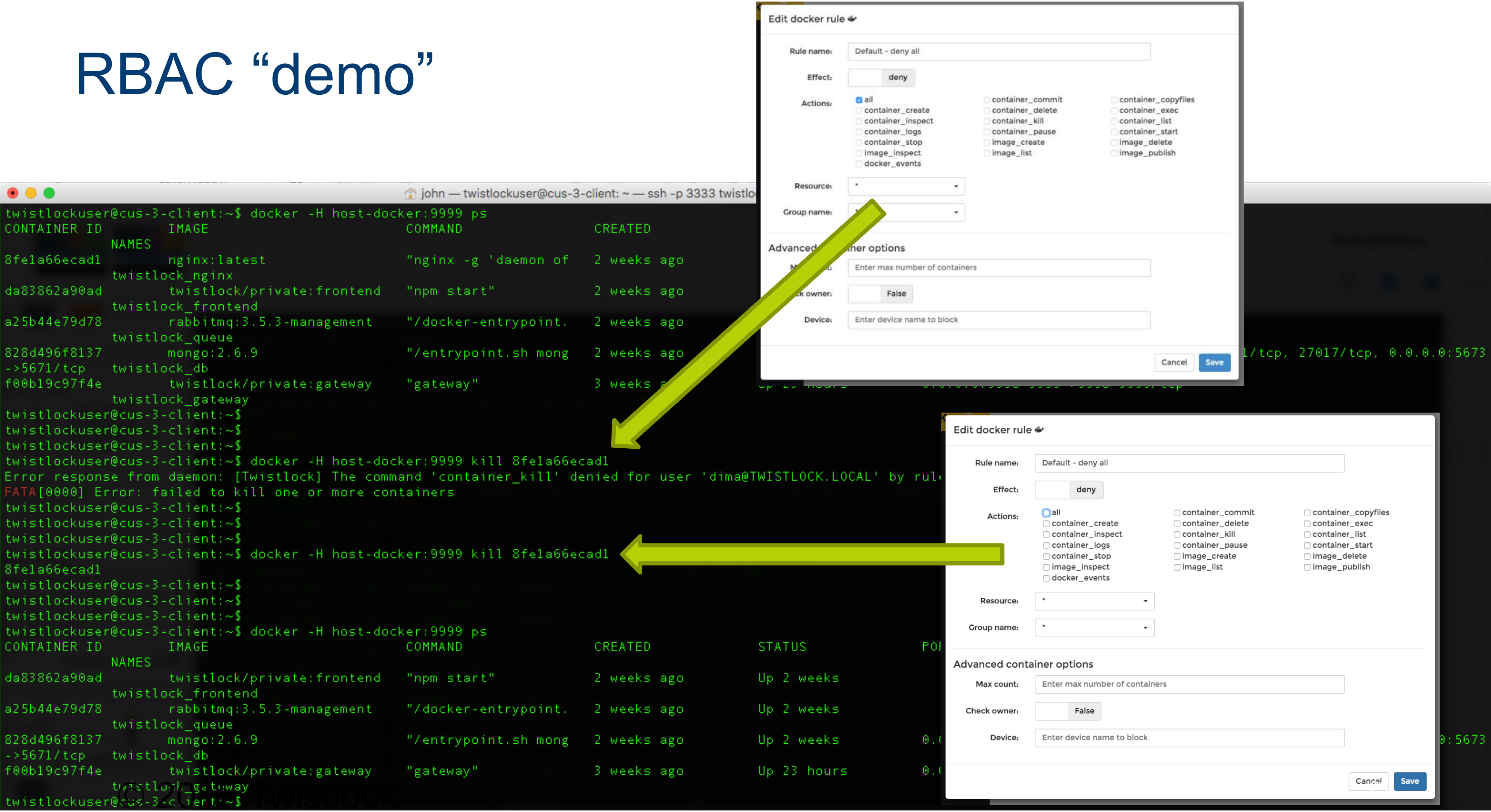


Ensure regulatory compliance

Prevent configuration drift

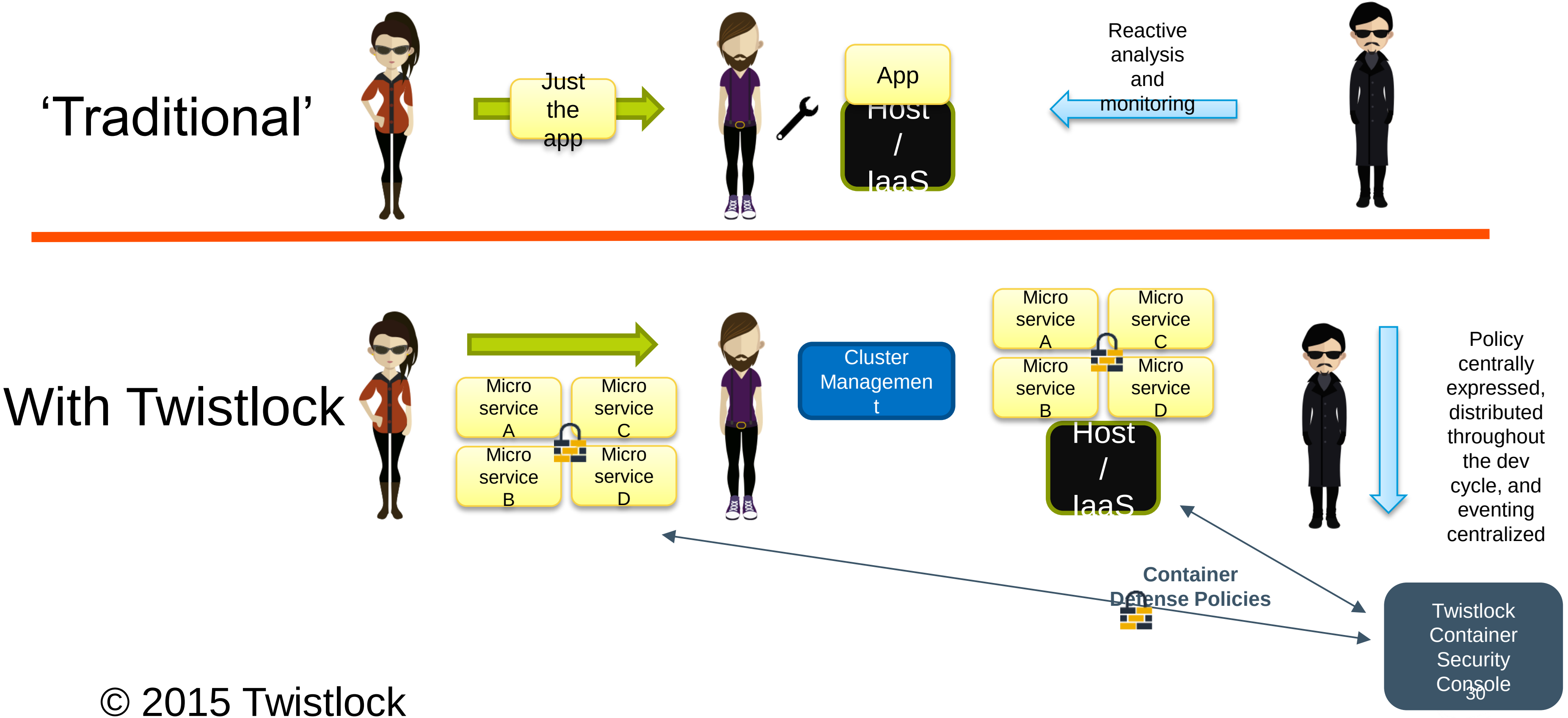


RBAC “demo”



Edit docker rule
Rule name: Default - deny all
Effect: deny
Actions:
all
container_create
container_inspect
container_logs
container_stop
image_inspect
docker_events
container_commit
container_delete
container_kill
container_pause
image_create
image_list
container_copyfiles
container_exec
container_list
container_start
image_delete
image_publish
Resource: *
Group name: *
ner options
Enter max number of containers
k owner: False
Device: Enter device name to block
Cancel
Save
john — twistlockuser@cus-3-client: ~ — ssh -p 3333 twistlo
twistlockuser@cus-3-client:~$ docker -H host-docker:9999 ps
CONTAINER ID IMAGE COMMAND CREATED
NAMES
8fe1a66ecad1 nginx:latest "nginx -g 'daemon of 2 weeks ago
twistlock_nginx
da83862a90ad twistlock/private:frontend "npm start" 2 weeks ago
twistlock_frontend
a25b44e79d78 rabbitmq:3.5.3-management "/docker-entrypoint. 2 weeks ago
twistlock_queue
828d496f8137 mongo:2.6.9 "/entrypoint.sh mong 2 weeks ago
->5671/tcp twistlock_db
f00b19c97f4e twistlock/private:gateway "gateway" 3 weeks a
twistlock_gateway
twistlockuser@cus-3-client:~$
twistlockuser@cus-3-client:~$
twistlockuser@cus-3-client:~$
twistlockuser@cus-3-client:~$ docker -H host-docker:9999 kill 8fe1a66ecad1
Error response from daemon: [Twistlock] The command 'container_kill' denied for user 'dima@TWISTLOCK.LOCAL' by rul
FATA[0000] Error: failed to kill one or more containers
twistlockuser@cus-3-client:~$
twistlockuser@cus-3-client:~$
twistlockuser@cus-3-client:~$
twistlockuser@cus-3-client:~$ docker -H host-docker:9999 kill 8fe1a66ecad1
8fe1a66ecad1
twistlockuser@cus-3-client:~$
twistlockuser@cus-3-client:~$
twistlockuser@cus-3-client:~$
twistlockuser@cus-3-client:~$ docker -H host-docker:9999 ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PO
NAMES
da83862a90ad twistlock/private:frontend "npm start" 2 weeks ago Up 2 weeks
twistlock_frontend
a25b44e79d78 rabbitmq:3.5.3-management "/docker-entrypoint. 2 weeks ago Up 2 weeks
twistlock_queue
828d496f8137 mongo:2.6.9 "/entrypoint.sh mong 2 weeks ago Up 2 weeks
->5671/tcp twistlock_db
f00b19c97f4e twistlock/private:gateway "gateway" 3 weeks ago Up 23 hours
twistlock_gateway
twistlockuser@cus-3-client:~$
1/tcp, 27017/tcp, 0.0.0.0:5673
Edit docker rule
Rule name: Default - deny all
Effect: deny
Actions:
all
container_create
container_inspect
container_logs
container_stop
image_inspect
docker_events
container_commit
container_delete
container_kill
container_pause
image_create
image_list
container_copyfiles
container_exec
container_list
container_start
image_delete
image_publish
Resource: *
Group name: *
Advanced container options
Max count: Enter max number of containers
Check owner: False
Device: Enter device name to block
Cancel
Save

Evolution of roles with containers



The containers are coming

… if they’re not already on your network

Balance security and capability with tools purpose built for the new model

Twistlock

Lancope®

Container Security

Paul Cichonski
Cloud Architect

@paulcichonski

Why Containers? Two Areas of Focus

1. Software Delivery:

- Build pipelines now produce consistent, immutable artifacts
- Immutable artifacts offer many benefits for security

2. Software Deployment:

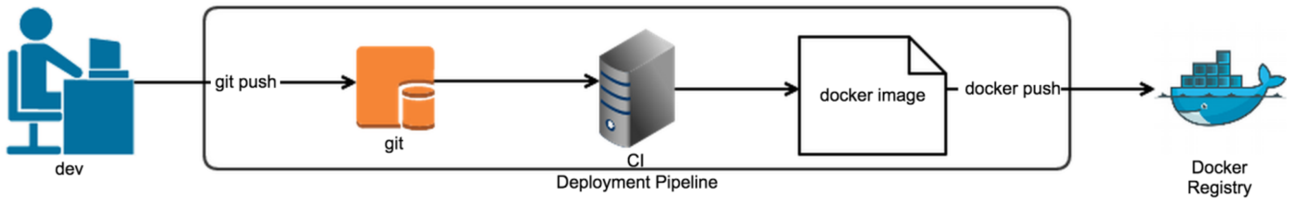
- Software deployment mechanism is common across all technologies (e.g., python, JVM, c, perl)
- If it can go into a container, you can deploy it
- Incredible for devs, but creates many challenges for security

Evolution of Software Delivery

Era*	Characteristics
Custom Bash Scripts (1990s – late 2000s)	• Mutable infrastructure (e.g., send EAR to server) • Servers are pets, we even gave them names • Many differences between environments • Deployed a few times per month (if lucky)
Configuration Management (Late 2000s – current)	• Immutable infrastructure • Servers start becoming cattle • Still many differences between environments • Deployments happen more, but still slow
Containers (now)	• Immutable infrastructure • Servers become more like cattle, OS provides bare minimum to run container • Software systems now fully reproducible in all environments • Build/Deployment pipeline is the center of the universe • Deploy as frequently as your build pipeline can produce new image

**time periods are rough estimates, they change depending on who you ask.*

Deployment Pipelines with Containers



Steps taken inside CI (fail fast between steps):

1. Standup dependent services (using containers) for testing
2. Run unit and integration tests on code
3. Create final Docker Image of tested code
4. Start container using newly created image
5. Run black box functional tests on container
6. **Run security scans on container**
    1. Examples: SCAP scan, GAUNTLT tests, CIS Docker Benchmark
7. Push validated image to registry IFF all previous steps pass

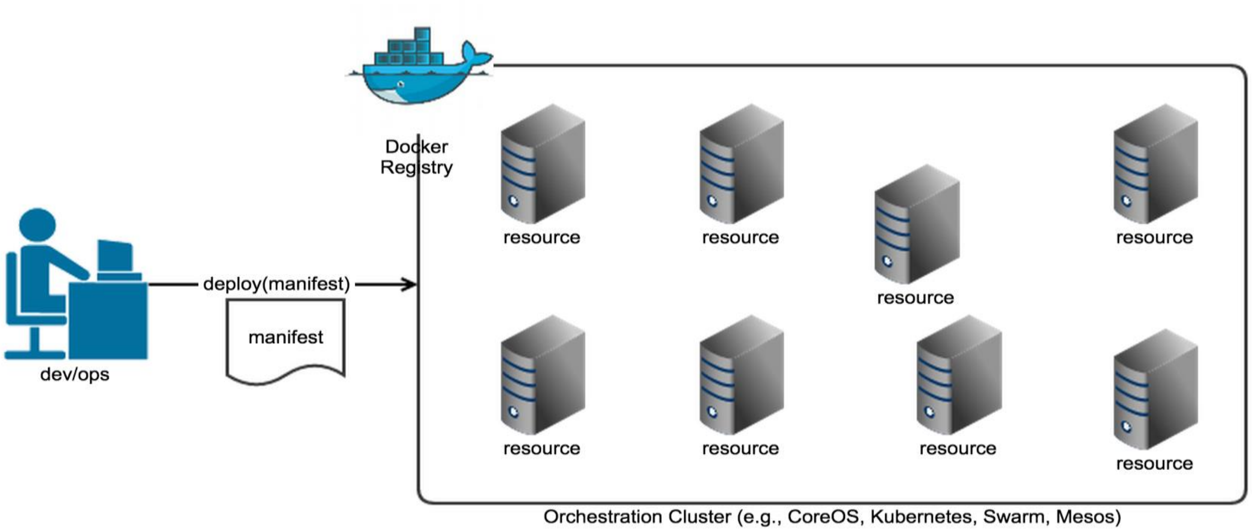
Why this is good for security?

- A successful run of deployment pipeline gives us an immutable image for deploying to production
  - Never run anything not validated by pipeline
- Before ever getting to prod, we have already instantiated the container and run:
  - Blackbox functional tests
  - Full security scans (both blackbox and whitebox)

This means we can catch security issues before ever releasing software into the wild

Side bonus: devs can run all this from their laptop

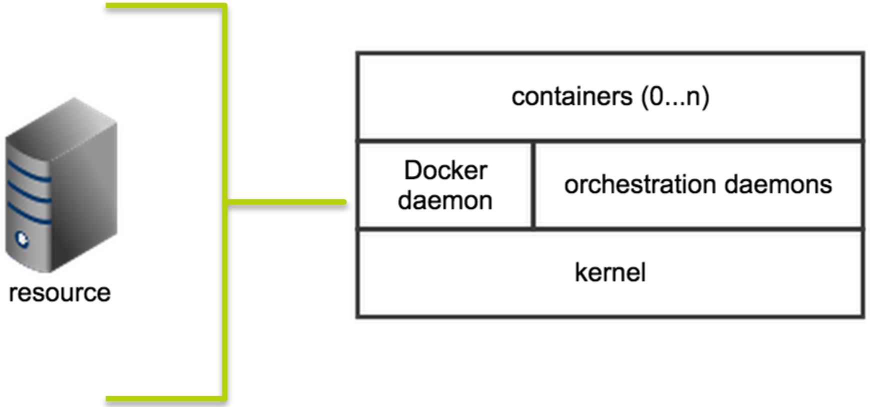
Deploying Containers (high level)



Manifest encodes:

- Docker image to launch on cluster
- Number of instances to deploy (e.g., run 3 instances of nginx container)
- Resource requirements (e.g., each container needs 2 cores and 8gb memory)
- Custom rules (e.g., don’t run container X and container Y on same host)

Deploying Containers (high level)



- Each server (or resource) is only there to run containers
- Stripped down kernel (lower attack surface)
- Orchestration tooling required to help schedule containers across a cluster

Benefits for Dev/Ops

Containers provide a common point of abstraction for deploying any arbitrary software stack

Great for microservices and polyglot infrastructures

We can start thinking about creating infrastructure-level patterns and sharing them via GitHub (think: package manager for your datacenter)

What about security? Here be dragons

Orchestration layer adds new set of distributed communication protocols that must be secured

Host-level isolation for different workloads still required until container isolation is on par with OS isolation

Storing secrets becomes more complex in a dynamic world

Image validation tooling required to forbid untrusted images (it is not enabled by default)

Burden of patching software shifts to devs

Questions?