Purple Containers

Attack and defence across the entire container stack

David Grice

Slides: https://git.io/fjoX7

I am Dave

- Technical Security Consultant & previously in governance roles
- Work in financial services
- I like motorbikes

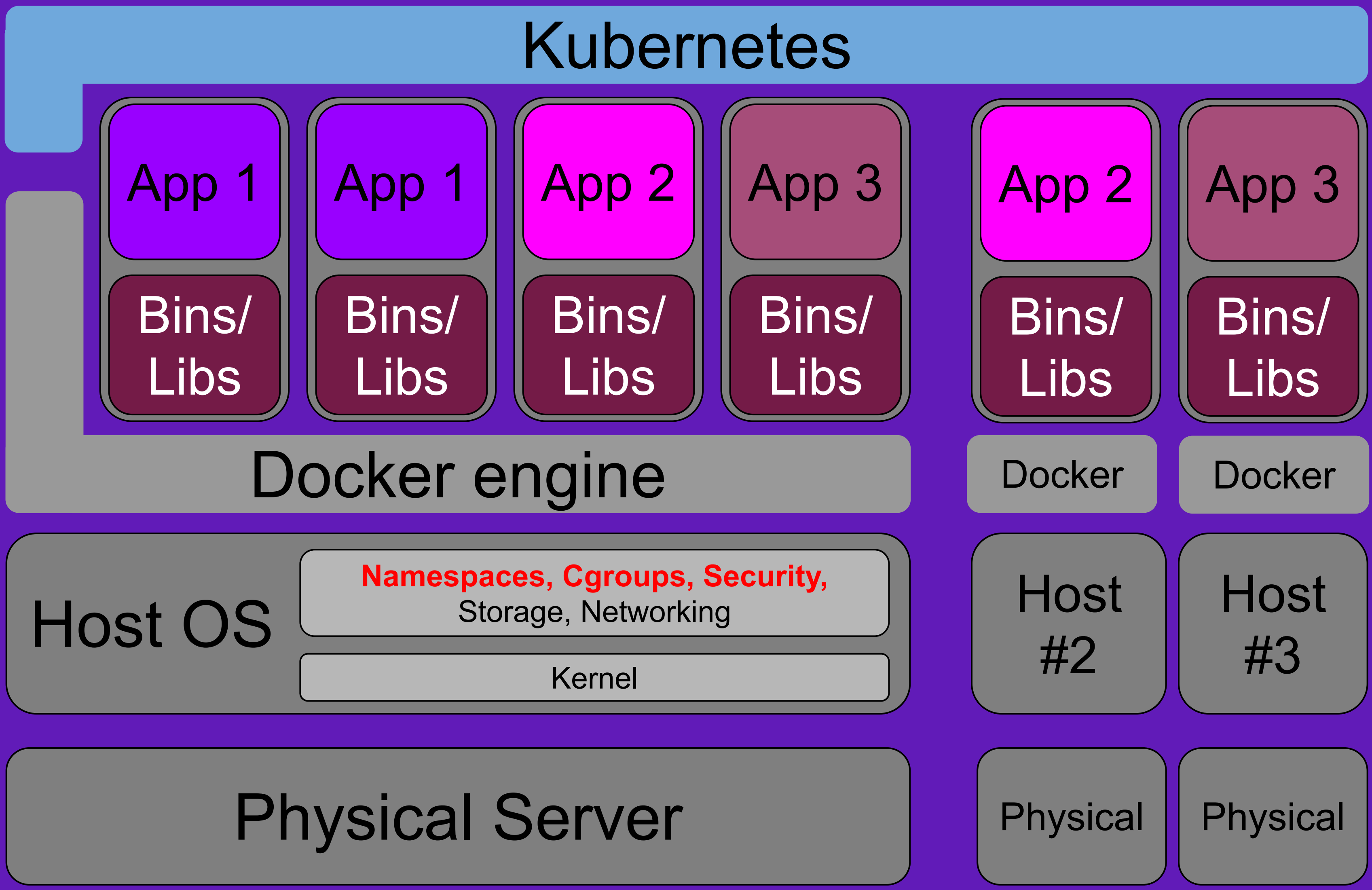

BEGINNING, MIDDLE, END. FACTS, DETAILS,
CONDENSE, PLOT, TELL IT.

Containers have arrived...

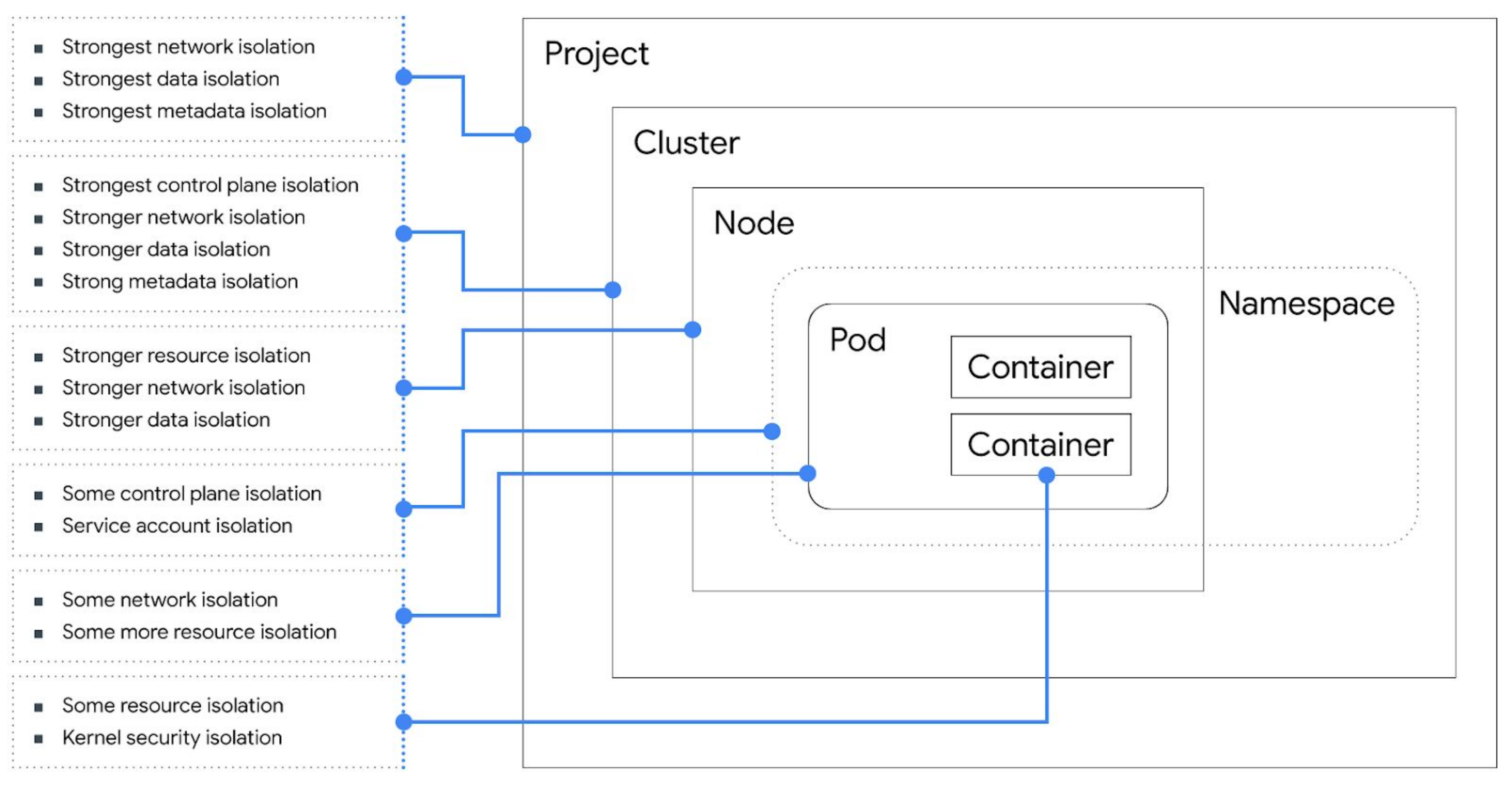
App teams:
“We want containers”

Security:
“Yeah, but...”

Anatomy of a typical container stack



Anatomy from within Kubernetes



Source: https://cloud.google.com/blog/products/gcp/exploring-container-security-isolation-at-different-layers-of-the-kubernetes-stack

It’s not just the tech that is different, so are the processes to build & run it.

The risks are real (FUD page)

Vulnerability Details : CVE-2016-9962

RunC allowed additional container processes via 'runc exec' to be ptraced by the pid 1 of the container. This allows the main processes of the container, if running as root, to gain access to file-descriptors of these new processes during the initialization and can lead to container escapes or modification of runC state before the process is fully placed inside the container.

Publish Date : 2017-01-31 Last Update Date : 2018-01-04

A Dirty Cow Container Exploit Persists

NeuVector → Container Security → A Dirty Cow Container Exploit Persists

CONTAINER SECURITY

A Dirty Cow Container Exploit Persists

By Andson Tung

We have seen a lot of reports on how the Linux kernel can be compromised by the Dirty Cow (CVE-2016-5195) exploit. One technique that attackers use is to exploit this kernel bug to overwrite a so-called **setuid** program in the system. A setuid program allows the user to temporarily elevate the privilege in order to perform a certain task. By replacing the setuid program, the attacker can gain root access privilege when the program is executed, and be able to do whatever he/she wants.

SEARCH

Tweets

Tweets by @NeuVector

NeuVector
@NeuVector

While at #DockerCon, here are some interesting takeaways on how you can deploy @NeuVector in dev, staging or production. #ContainerSecurity neuvector.com/run-time-conta…

Docker » Docker : Security Vulnerabilities

CVSS Scores Greater Than: 0 1 2 3 4 5 6 7 8 9
Sort Results By : CVE Number Descending CVE Number Ascending CVSS Score Descending Number Of Exploits Descending
Copy Results Download Results

#	CVE ID	CWE ID	# of Exploits	Vulnerability Type(s)	Publish Date	Update Date	Score
1	CVE-2017-14992	20		DoS	2017-11-01	2017-11-22	4.3
Lack of content verification in Docker-CE (Also known as Moby) versions 1.12.6-0, 1.10.3, 17.03.0, 17.03.1, 17.03. Service via a crafted image layer payload, aka gzip bombing.							
2	CVE-2017-7297	264			2017-03-28	2017-04-04	6.5
Rancher Labs rancher server 1.2.0+ is vulnerable to authenticated users disabling access control via an API call. Thi rancher/server:v1.5.3.							
3	CVE-2016-9962	362			2017-01-31	2018-01-04	4.4
RunC allowed additional container processes via 'runc exec' to be ptraced by the pid 1 of the container. This allows processes during the initialization and can lead to container escapes or modification of runC state before the process							
4	CVE-2016-8867	264		Bypass	2016-10-28	2017-07-27	5.0
Docker Engine 1.12.2 enabled ambient capabilities with misconfigured capability policies. This allowed malicious ima							
5	CVE-2016-6595	399		DoS	2017-01-04	2017-08-15	4.0
** DISPUTED ** The SwarmKit toolkit 1.12.0 for Docker allows remote authenticated users to cause a denial of ser disputes this issue, stating that this sequence is not "removing the state that is left by old nodes. At some point the both for Docker swarm and for Docker Swarmkit nodes are *required* to provide a secret token (it's actually the on resources. We can't do anything about a manager running out of memory and not being able to add new legitimate vulnerability."							
6	CVE-2016-3697	264		+Priv	2016-06-01	2017-06-30	2.1
libcontainer/user/user.go in runC before 0.1.0, as used in Docker before 1.11.2, improperly treats a numeric UID as password file in a container.							
7	CVE-2015-3631	264			2015-05-18	2017-01-02	3.6
Docker Engine before 1.6.1 allows local users to set arbitrary Linux Security Modules (LSM) and docker_t policies vi							
8	CVE-2015-3630	264		+Info	2015-05-18	2017-01-02	7.2
Docker Engine before 1.6.1 uses weak permissions for (1) /proc/asound, (2) /proc/timer_stats, (3) /proc/latency_st perform protocol downgrade attacks via a crafted image.							
9	CVE-2015-3627	59		+Priv	2015-05-18	2017-01-02	7.2
Libcontainer and Docker Engine before 1.6.1 opens the file-descriptor passed to the pid-1 process before performing							
10	CVE-2014-9358	20			2014-12-16	2014-12-30	6.4
Docker before 1.3.3 does not properly validate image IDs, which allows remote attackers to conduct path traversal communications."							
11	CVE-2014-9357	264		Exec Code	2014-12-16	2014-12-30	10.0
Docker 1.3.2 allows remote attackers to execute arbitrary code with root privileges via a crafted (1) image or (2) bu							
12	CVE-2014-6408	264		Bypass	2014-12-12	2014-12-15	5.0
Docker 1.3.0 through 1.3.1 allows remote attackers to modify the default run profile of image containers and possib							

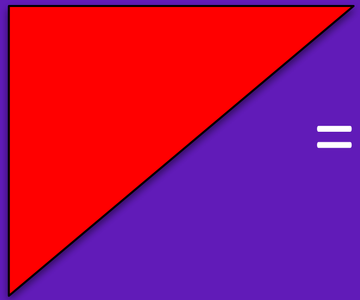
Dec 2018: CVE-2018-1002105 - Kubernetes API server auth bypass

Feb 2019: CVE-2019-5736 - RunC container breakout to host as root

So, “we want containers” becomes a security review of:

1. The build environment (CI/CD Pipeline)
2. Underlying Operating System Security
    2a. Cloud Services Hardening and Security
3. Container Image (inc. Docker Daemon)
4. Runtime security
5. Orchestration layer (e.g. Kubernetes)
6. Network
7. Logging & Auditing

Concerned?



= Attack / Threats

= Defence / Controls*

*Adopting all controls would be overkill…pick & choose as appropriate.

1. The build environment

- Malicious or ‘sub-optimal’ source code
- Malicious or mistaken alterations to automated build controllers/policies
- Configuration scripts with errors, or that expose credentials
- Supply chain provenance
- The addition of insecure libraries or down-rev/insecure versions of existing code

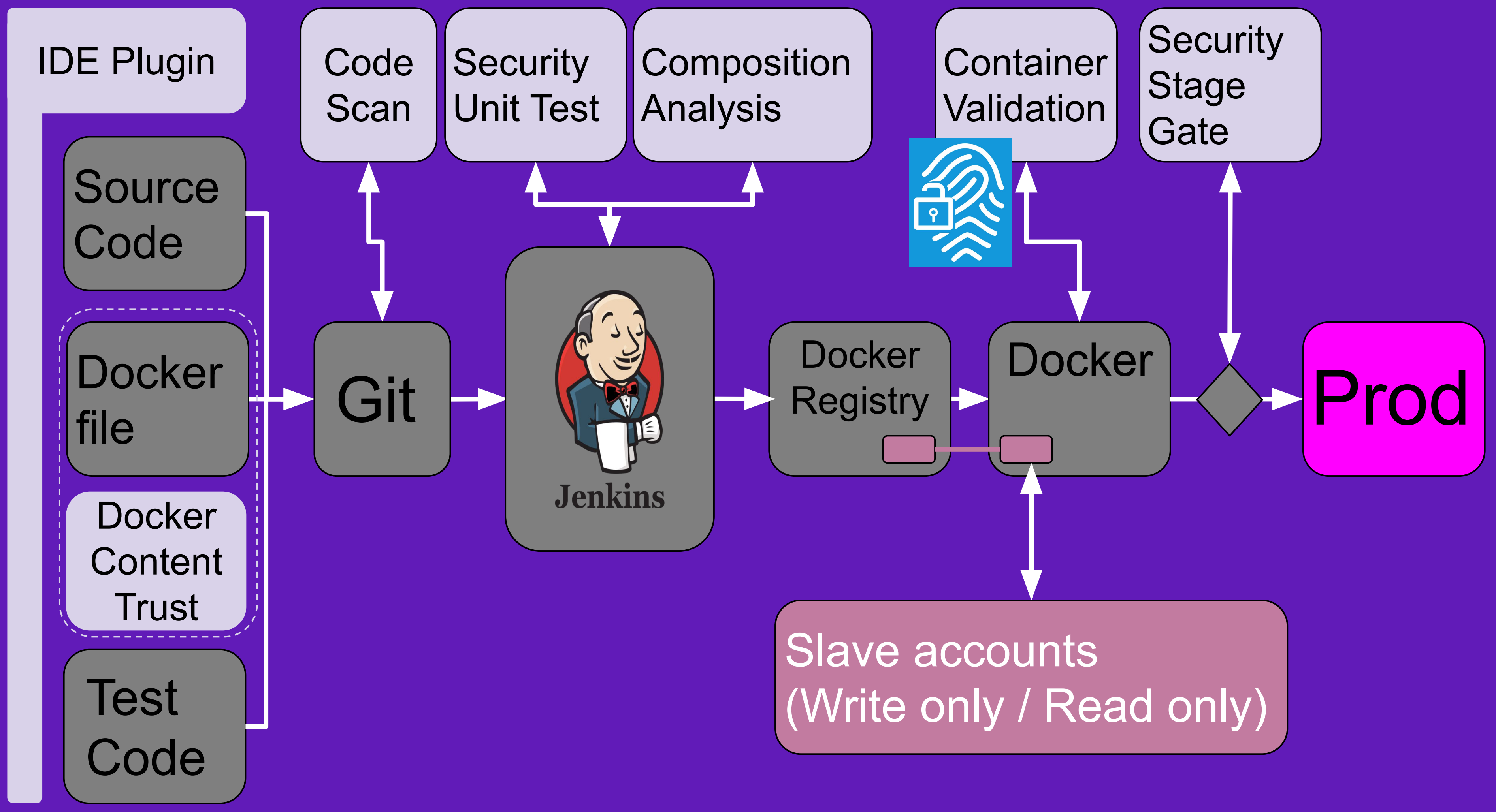
1. The build environment

- Protect your build environment like your app
- Governance over code changes
- Principle of least privilege:
  - who has access to code?
  - who has access to deploy?
- Use IDE security testing plugins

1. The build environment

- Integrate security tools into your build pipeline (SAST, Security Unit Tests)
- Implement policies to prevent bad code
- Hash your binaries
- Use a pattern of pre-agreed ‘safe code’ libraries

So maybe it looks like this….



2. Host Operating System

- Underlying kernel vulns
- Containers share OS resources
- Quarantine each container
- Container break-out

2. Underlying Operating System

- Patch often
- Run a container specific OS
- gVisor
- Enforce strict access control

or drop the ops….

- AWS, Azure & GCP all have managed k8s

3. Container Image

- Images need hardening
- Defaults are not exhaustive, particularly in older versions
- Provenance over supply chain: is this the container I was expecting?

3. Container Image

- CIS Benchmark
- Docker Bench Security
- Container Signing and Chain of Custody
  - Docker Content Trust is part of it
- Remove all unneeded modules, packages and files
- Fail builds with vulnerable images
  - Have a whitelisting strategy

4. Runtime security

- Port scanning
- Hijacked processes
- Data exfiltration
- Backdoor
- Network lateral movement
- Brute force attacks
- Container breakout

4. Runtime security

- Enforce whitelist with Mandatory Access Controls (MAC) - AppArmour, SELinux
  – Shout-out: RCE Guard
- Separate PID namespaces per tenant (not shared); default in current Kubernetes
- Remove SSH access from images
- Reduce your risk by running containers in read-only/non-persistent mode

4. Runtime security

- Enable Docker's seccomp profile
- V1.8+ defaults block a lot of nasty stuff: add_key, keyctl, request_key, clone, unshare
- Implement NO_NEW_PRIVS (Default in k8s)
- Enforce time-to-live thresholds
- Continue to vuln scan against running instances
- Super Paranoid? Hardware root of trust

5. Orchestration Layer

- Steal secrets
- Service account access to API layer
- Access cloud providers metadata API (e.g. AWS 169.254.169.254)
- Enumeration of services
- k8s API server attacks via token replay
- Container breakout and hijack kubelet
- Container MITM

5. Orchestration Layer

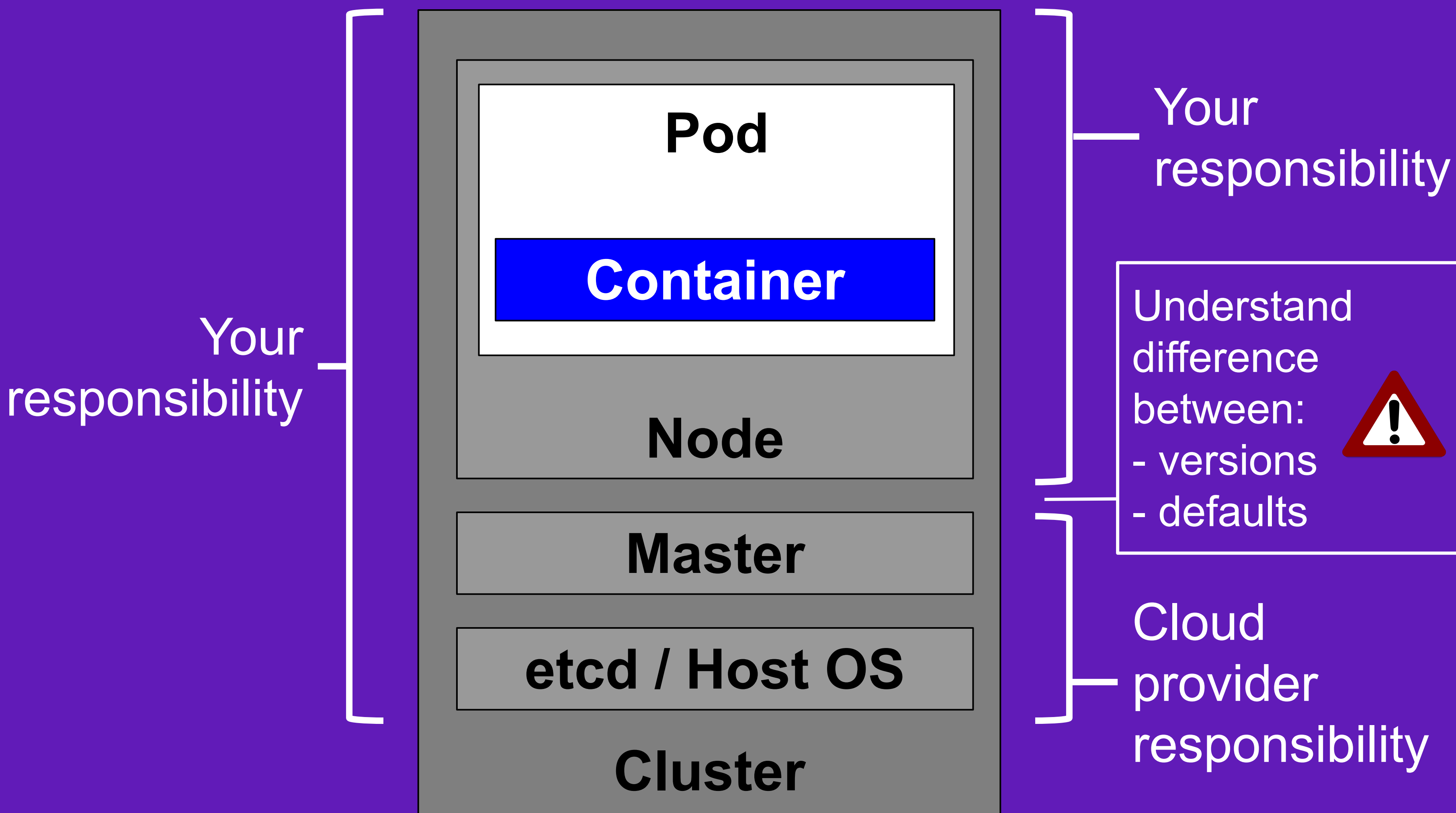
- CIS benchmarks for k8s
- RBAC & Cluster role binding
- Disable mounting of service account token
- Restrict access to the Kubernetes API
- Enable pod security policy option & assign it a policy (BETA)
- Restrict access to the Kubernetes Web UI (Dashboard); or turn it off!
- Implement a Network Policy using pod labels

5. Orchestration Layer

- Encrypt your K8s secrets - this is two things!
- Restrict Kubelet permissions so you only know pods on that node
- Use a service mesh (e.g. Istio)
- Add a specific metadata proxy
- Log everything outside the cluster
- Rotate Kubelet Certificates
- Run Kubernetes V1.8 at a minimum, but fresher is better (current is V1.13)

5. Orchestration Layer

Self Managed Vs. Managed



6. Network Security

- More surface area now that microservices are talking to each other
- Need to isolate discrete workloads
- Restrict traffic to unintended services
- Prevent egress to the internet
- Is logical isolation enough for a DMZ?

6. Network Security

- Not much has changed since the 80’s, same concepts apply.
- Orchestration layer:
  - Use a service mesh (e.g. Istio)
  - Mutually Authenticated Proxy connections & TLS your traffic
- Container layer:
  - Calico
  - Principle of least privilege (nano-segmentation)

6. Network Security

- Cloud Service:
  - Separate cloud accounts/VPCs/projects/resource groups for different workload groups
  - 0.0.0.0/0 CIDR ranges should be avoided
  - Limit privilege to change network policies
  - Segregation of duties
- Run regular checks against ‘known good’
- Limit your blast radius
- Log

7. Logging & Auditing

- Need to meet operational requirements
- Logging via the application may not be enough
- Containers are transient
- Need to meet forensic requirements
- Logs are not sufficiently protected
- Logs are not sanitised
- Signal to noise ratio
- Cloud services features keep changing

7. Logging & Auditing

- Small & Simple - use a dedicated logging container
- Big & Complex - consider sidecar approach
- Practice end-to-end logging - app, k8s, container, OS, cloud services

7. Logging & Auditing

- OWASP Logging cheat sheet
- Log into a separate immutable store under a dedicated cloud account
- Alert & alarm when logs do not arrive
- Sysdig & Prometheus
- Have processes to monitor change in cloud services
- Your systems are talking, but are you listening?

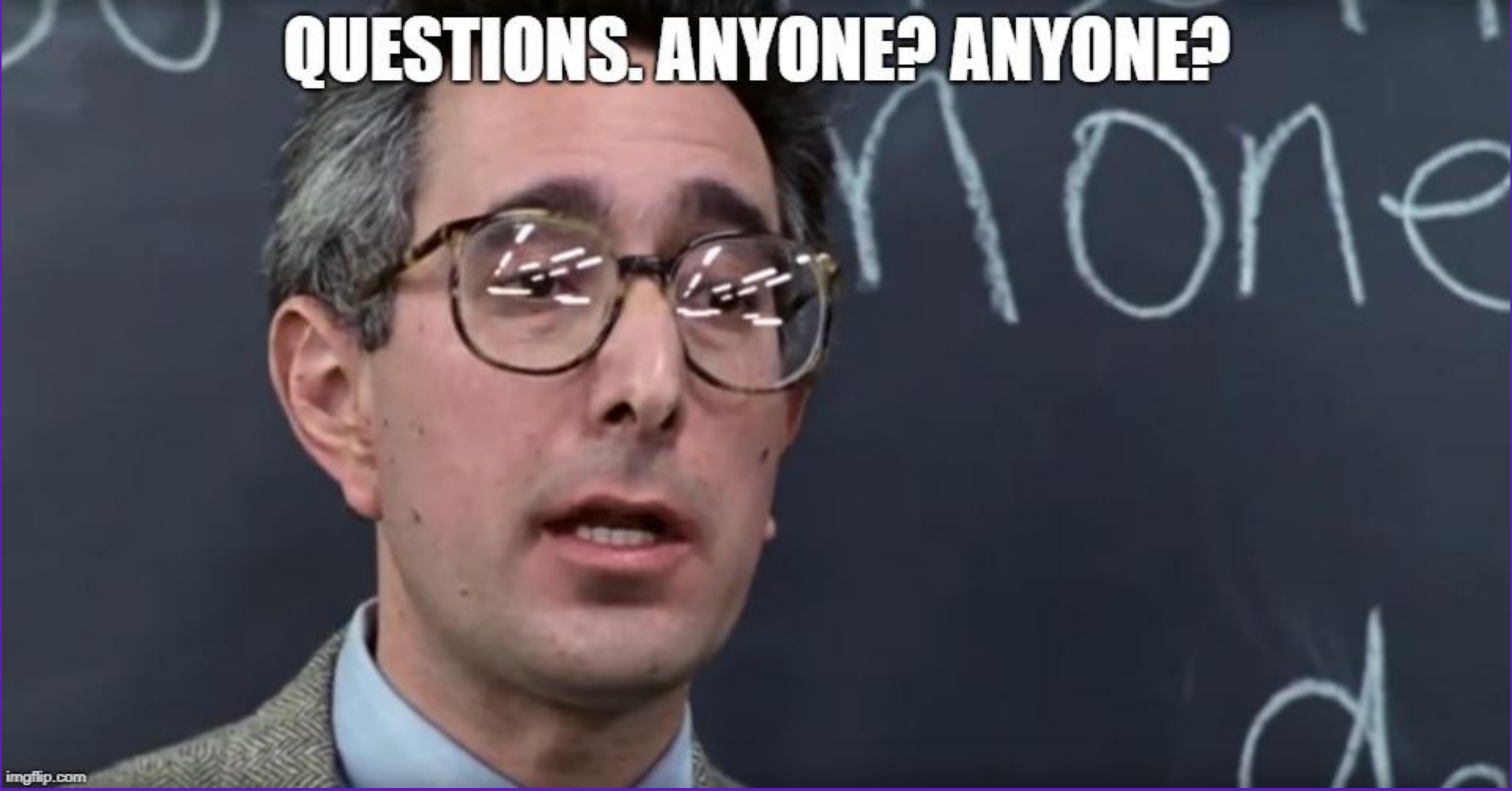
Too Much?

Let’s focus on the key ones

Essential Ingredients

- Pod Security Policy: non-root
- Docker Bench & Kube Bench Security
  - RBAC essential
- Run a minimal OS with no SSH; 400MB is large
- Restrictive Network Policy
- Cloud Account hardening: blast radius, segregation, CIDR ranges, metadata API proxy
- Enforce run time protection
- Protect your build environment like your app

- Defaults
- Automation
- Control Ownership
- Lean in



www.linkedin.com/in/davidagrice

@Shh_Dontell Slides: https://git.io/fjoX7