

# Containers and Kubernetes

Security is not an afterthought

---

October 2021



**Matt Colman**

Senior Cloud Security Architect, IBM Security

# Who am I?

Matt Colman

Senior Cloud Security Architect, IBM Security

13 years in IT

Very interested in the benefits containers can offer, but aware of the security concerns

My wife knows I do “cloud security” and my kids know I do something for IBM, “in the office”



# Agenda

The world of containers, Kubernetes (K8s) and the challenges of developing, deploying and managing secure applications in this environment.

- Introduction to containers and the role of Kubernetes.
- Some of the common security challenges.
- Strategies for securing containerised applications at the various stages of a DevSecOps pipeline.
- Best practices for addressing run-time security concerns for containerised applications and Kubernetes environments.

# Part 1

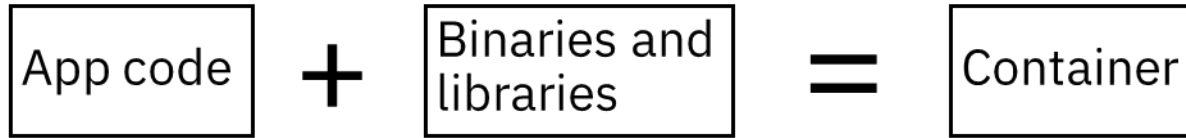
An introduction to containers and the role of Kubernetes.  
Having set the scene, the focus will move onto some of the common security challenges.

# Containers: How are they possible?

- Building blocks for containers:
  - **Namespaces:** kernel level isolation of processes
  - **Cgroups:** Control resource (CPU, memory, network, disk I/O) accessed/used by process/set of processes

# What actually is a container?

In a nutshell ...



...Containers are created from container images

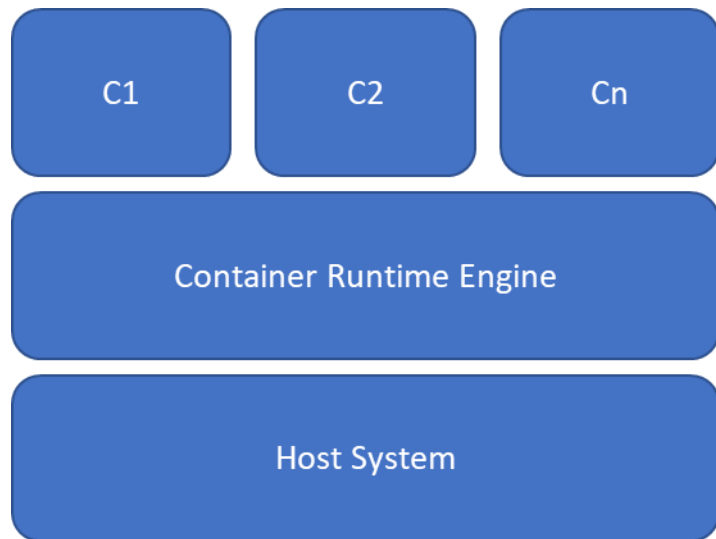
# Building a Container Image

- A container image can be created from a Dockerfile/Containerfile
- Various build tools available (Docker, Buildah, Kaniko)
- Be aware of image size (e.g., layers and bloating)
- Store it in local or remote image registry

# Running Containers

To execute a container, we use a container runtime engine. Commonly used ones are:

- CRI-O (used in Kubernetes)
- Containerd (from Docker)
- rkt





# The need for Container Orchestration

- Why Kubernetes came about...container orchestration
- Kubernetes is an open-source system for automating deployment, scaling, and management of containerised applications.

# Common Security Challenges

- Containers share the underlying host's kernel
- A container doesn't have security built in by default
- Using containers can help speed up our development lifecycle – security must keep up
- It's easy to take shortcuts to “just make it work”
- Container orchestration security doesn't just happen, we must get involved...

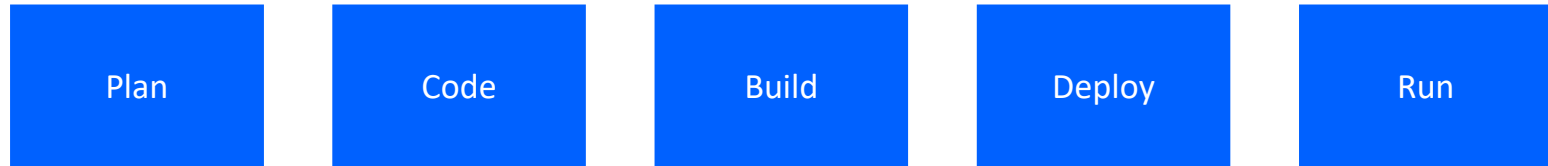
# Summary

- Namespaces and cgroups make containers possible
- A container image is the template from which a container is created & run
- A container image can be built from a configuration file – consider layers
- Container runtime engines abstract the complexity of creating and running containers
- Container Orchestrators bring scale and management of containers
- There are security challenges and we must proactively tackle them

# Part 2

Some strategies for securing containerised applications at the various stages of a DevSecOps pipeline.

# Key Lifecycle Phases for Container Security



...Throughout the lifecycle!

# Key things to shout about...

- Educate your team
- Consider your Host OS
- Deploy Kubernetes in a secure manner
- Separate and control image registries
- Configure the image securely
- Configure the workload object (deployment, statefulset etc) securely
- Check for vulnerabilities and weak configuration in pipeline
- Implement robust admission control for workloads
  - Get everything running with minimal privilege and permissions
- Monitor the runtime

# Plan

# NIST SP 800-190

- Title: “Application Container Security Guide”
- Contents includes:
  - Introduction to application containers
  - Major risks for core components of container technologies
  - Countermeasures for major risks
  - Container threat scenario examples
  - Container technology life cycle security considerations

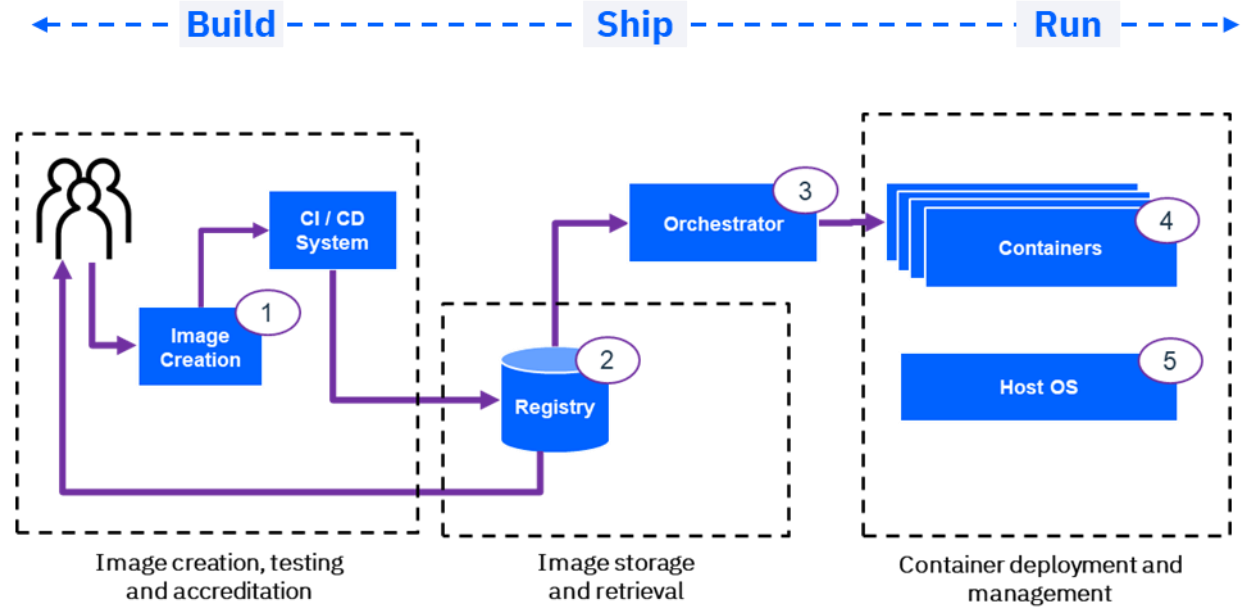
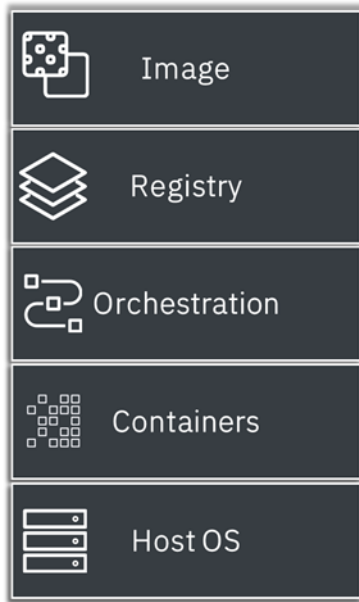


# NSA & CISA Kubernetes Hardening Guidance

The National Security Agency (NSA) and Cybersecurity and Infrastructure Security Agency (CISA) have put together a hardening guide for Kubernetes. It covers:

- Threat Model
- Pod Security
- Network separation and hardening
- Authentication and Authorisation
- Log Auditing
- Upgrading and application security practices

# Key Areas of Container Security



# HostOS Security

- Running a full blown OS vs a container-optimised distribution...
- Benefits of the latter:
  - Minimal OS image; only includes tools needed to run containers
  - Immutable filesystems (eliminate a lot of vulnerabilities)
  - Automated atomic upgrades
- For the former:
  - Consider why you are using it
  - Remove anything you don't need
  - Perform hardening!

# Orchestration: Things to consider...

- Secure cluster configuration
- Account use
- Authentication
- Authorisation
- Admission Control
- Resource Management
- Secrets Management (and encryption)
- Audit
- Workload placement (Taints & Tolerations)
- Exposing services
- Network Policies & Microsegmentation



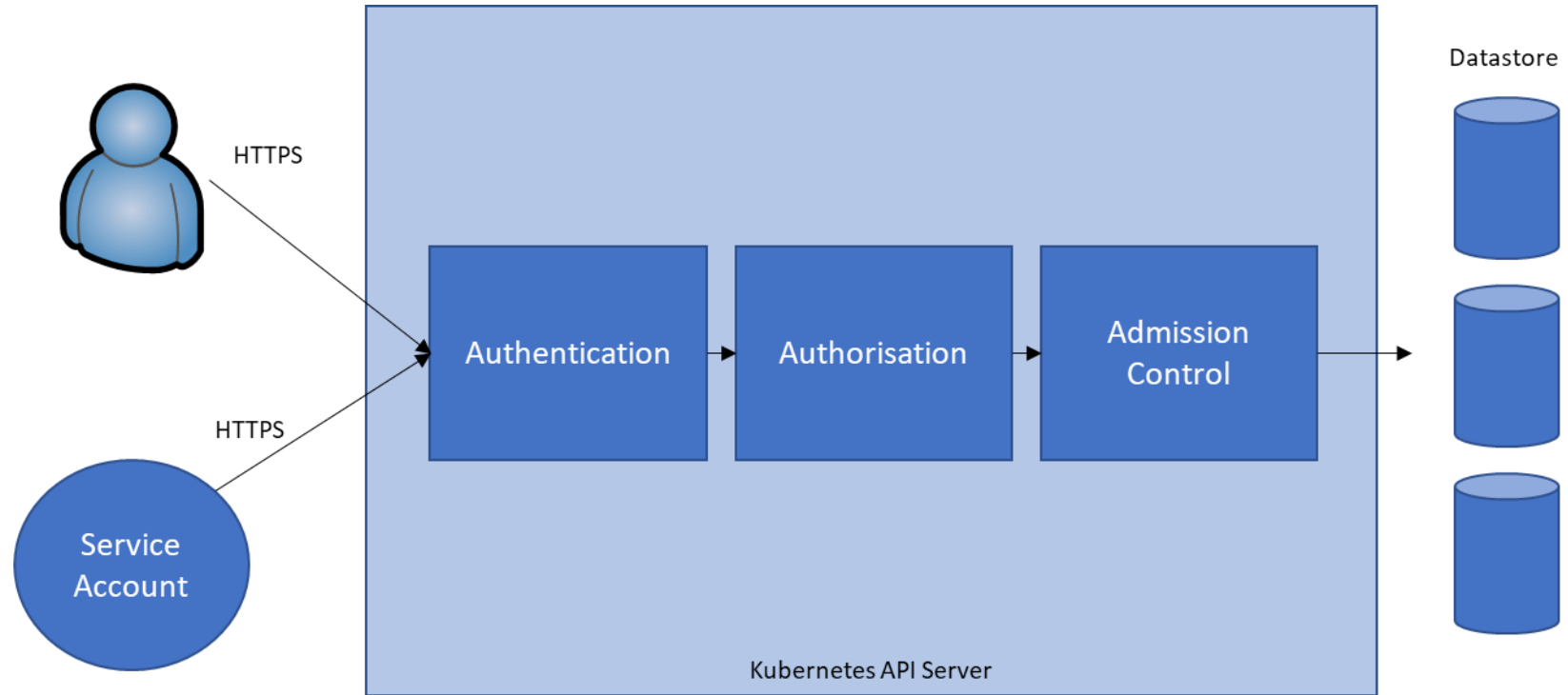
# Secure Cluster Configuration

CIS Kubernetes Benchmark v1.6.1 is available from Center for Internet Security, has sections on:

- Control Plane Components
- Etcd
- Control Plane Configuration
- Worker Nodes
- Policies

# Kubernetes Accounts

When you (a human) access the cluster (for example, using `kubectl`), you are authenticated by the apiserver as a particular User Account. Processes in containers inside pods can also contact the apiserver. When they do, they are authenticated as a particular Service Account (for example, default).



# Default ClusterRoles

Kubernetes provides some ClusterRoles that are intended to be user-facing roles:

- **Cluster-admin:** do anything on any resource, cluster-wide
- **Admin:** do anything on any resource within a namespace
- **Edit:** read/write to most objects in a namespace (can't modify roles or role bindings)
- **View:** read-only access to see most objects in a namespace (can't view roles or role bindings)

...Use these where possible. Base access on namespaces too.



# What are Admission Controllers?

Kubernetes admission controllers are plugins that govern and enforce how the cluster is used. They can be thought of as a gatekeeper that intercept (authenticated) API requests and may change the request object or deny the request altogether.

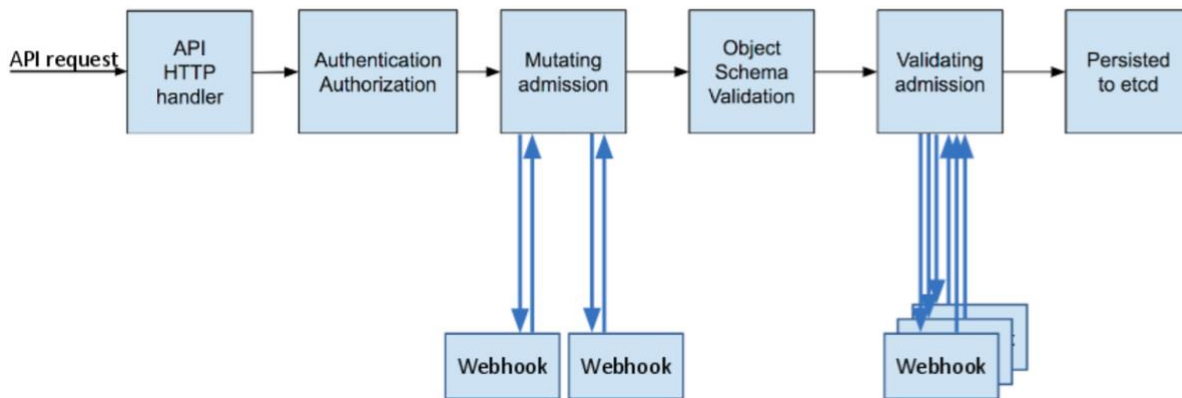


Image source: <https://kubernetes.io/blog/2019/03/21/a-guide-to-kubernetes-admission-controllers/>

# Admission Controller – Security Interest

- Pod Security Policies
- LimitRange
- ResourceQuota

# Pod Security

- PodSecurityPolicy – deprecated in v1.21 (1.22 is latest version)
- Being replaced: KEP-2579: Pod Security Admission Control
- “Replace PodSecurityPolicy with a new built-in admission controller that enforces the Pod Security Standards.”

Profile	Description
<b>Privileged</b>	Unrestricted policy, providing the widest possible level of permissions. This policy allows for known privilege escalations.
<b>Baseline</b>	Minimally restrictive policy which prevents known privilege escalations. Allows the default (minimally specified) Pod configuration.
<b>Restricted</b>	Heavily restricted policy, following current Pod hardening best practices.

# Pod Security

Has settings for:

- Access to host namespaces
- Running as privileged
- Running as root/non-root users and groups
- Kernel capabilities
- HostPath volumes
- Use of AppArmor and SELinux (access control)
- Seccomp profile (controlling syscalls)
- Sysctls (changing kernel parameters)

# Resource Control

- **LimitRange:**
  - Enforce minimum and maximum compute resource per pod or container
  - Set default/limit for compute resources in a namespace and automatically inject them to containers at runtime
- **ResourceQuota:**
  - Restrict resource consumption on a namespace basis

# Secrets Management

- Secret: An object that contains a small amount of sensitive data such as a password, a token, or a key
- Some potential gotchas:
  - Stored unencrypted by default
  - Anyone with API access and permission can view a secret
  - Anyone authorised to create a pod in a namespace can use that access to read any secret in the namespace
- So... RBAC, enable encryption at rest for secrets and consider use of a secrets vault.

# Audit

Auditing allows cluster administrators to answer the following questions:

- what happened?
- when did it happen?
- who initiated it?
- on what did it happen?
- where was it observed?
- from where was it initiated?
- to where was it going?

So, get it configured!

# Workload Placement

As per NIST SP 800-190 guidance:

*“Orchestrators should be configured to isolate deployments to specific sets of hosts by sensitivity levels.”*

**Node selector:** affects a single pod template and is useful when the pod needs something from the node, e.g., a kernel parameter setting.

**Taints and tolerations:** will affect all pods. Taints allow a node to repel a set of pods. Tolerations are applied to pods and allow (but do not require) the pods to schedule onto nodes with matching taints.



# Exposing Services

- By default, pods and their services are not exposed outside the cluster; it is a conscious decision to do so.
- Consider:
  - Who are you exposing it to?
  - Why are you exposing it? Does anything outside the cluster need it?
  - Have you secured the application being exposed?
  - Could port-forwarding be of use?
  - Use TLS

# Network Policies

- By default, pods are non-isolated; they accept traffic from any source.
- Network policies are additive
- Good ideas:
  - Block all traffic from outside the namespace
  - Permit the traffic that is actually required
- Consider a Service Mesh for production:
  - Facilities deployment strategies based on diverting traffic
  - Application layer aware
  - Additional controls and insights

# Code

# Securing Container Images: Best Practices

- Define and implement process for trusted base images
- Use minimal base image
- Reference base image by version tag or even better, its digest
- Don't run as root
- Only install required packages
- Don't run upgrades of packages
- Don't include secrets in configuration file
- Use COPY not ADD
- Don't open port 22 or include SSH unless required
- Remove set userID or set groupID permissions unless required

# Securing Container Images

Spot the difference...

```
FROM python:3.9.7-alpine3.14

ARG USER=myuser
ARG UID=1000
ARG GID=1000

COPY requirements.txt /tmp
RUN pip install --no-cache-dir -r /tmp/requirements.txt && \
    addgroup -S -g ${GID} ${USER} && \
    adduser -S -D -u ${UID} -G ${USER} ${USER}

COPY --chown=${UID}:0 my_app/the_api /my_app/my_api
COPY --chown=${UID}:0 my_app/ci_cd/ci_cd.py /my_app/
COPY --chown=${UID}:0 myscript.sh /my_app/

RUN chmod -R g=u /my_app/

USER ${UID}
```

```
FROM python:latest

COPY requirements.txt /tmp
RUN pip install --no-cache-dir -r /tmp/requirements.txt

COPY my_app/the_api /my_app/my_api
COPY my_app/ci_cd/ci_cd.py /my_app/
COPY myscript.sh /my_app/
```

# Securing Container Images

## Multi-stage

```
##### This is a multi-stage build #####
##### Below is for building app #####
FROM node:12.22.6 as installer
COPY ./nodejs /simple-page
WORKDIR /simple-page
RUN npm install --production --unsafe-perm && \
    npm dedupe

##### Below is for building image #####
FROM node:12.22.6-alpine3.13

# Arguments for use in image build
ARG USER=simple
ARG UID=1001
ARG GID=1001

WORKDIR /simple-page

RUN addgroup --system --gid ${GID} ${USER} && \
    adduser ${USER} --system --uid ${UID} --ingroup ${USER}

COPY --from=installer --chown=${USER} /simple-page .

# Remove any setuid or setgid bits from files to avoid permission elevation
RUN find / -xdev -perm /6000 -type f -exec chmod a-s {} \; || true

USER ${UID}
EXPOSE 8080
CMD ["npm", "start"]
```

# Securing Container Workloads

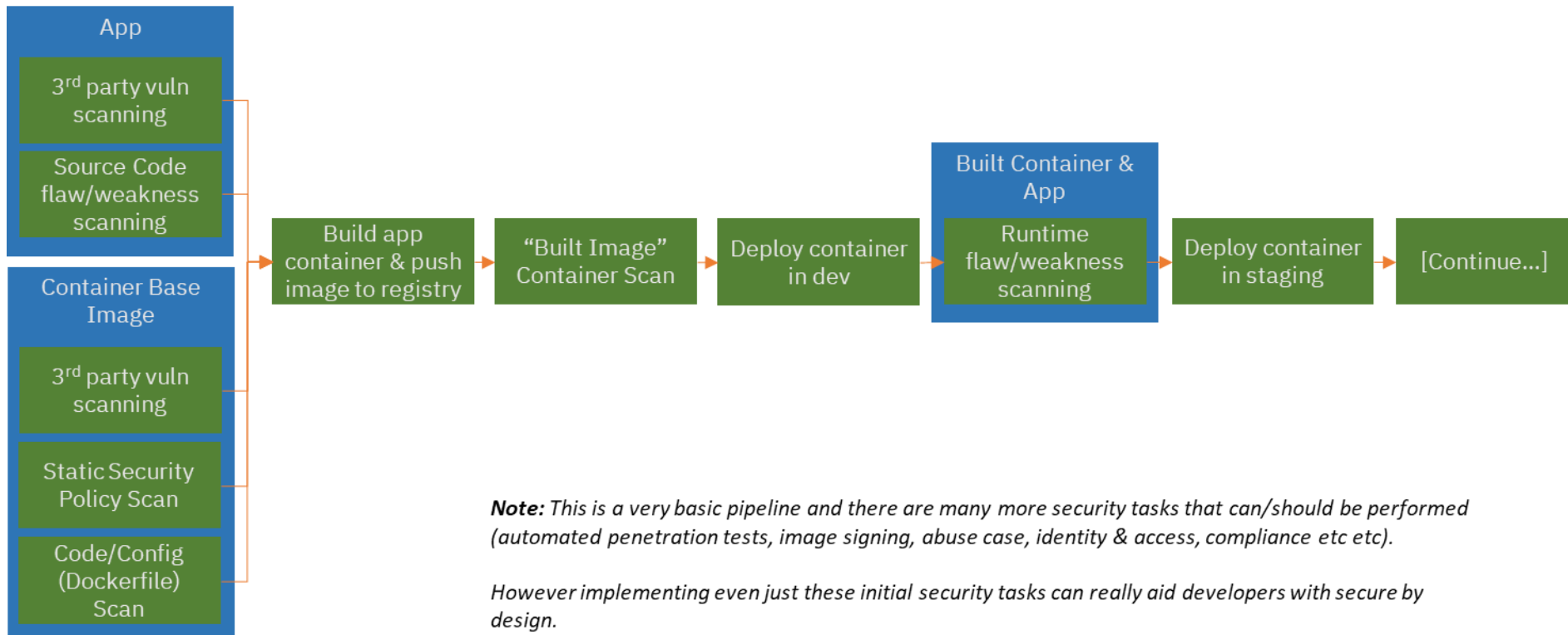
- Specify securityContexts!
- Some can be specified at both pod and container level (container level takes precedence)

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: <deployment-name>
...
...
containers:
- name: "<container-name>"
  image: "<trusted-registry>/<trusted-image>:<explicit-tag-or-hash>"
  imagePullPolicy: "IfNotPresent"
  resources:
    limits:
      cpu: <number>
      memory: <number>
    requests:
      cpu: <number>
      memory: <number>
  env:
  - name: <env-variable-name>
    value: <env-variable-value>
  securityContext:
    privileged: <false|true>
    allowPrivilegeEscalation: <false|true>
    runAsUser: <number>
    runAsGroup: <number>
    supplementalGroups: <number>
    fsGroup: <number>
    capabilities:
      add: ["<cap-l>", "<cap-n>"]
      drop: ["ALL", "NET_RAW"]
    seccompProfile:
      type: RuntimeDefault
    selinuxOptions:
      level: "<labels>"
    readOnlyRootFilesystem: <false|true>
...
...
```

# Build



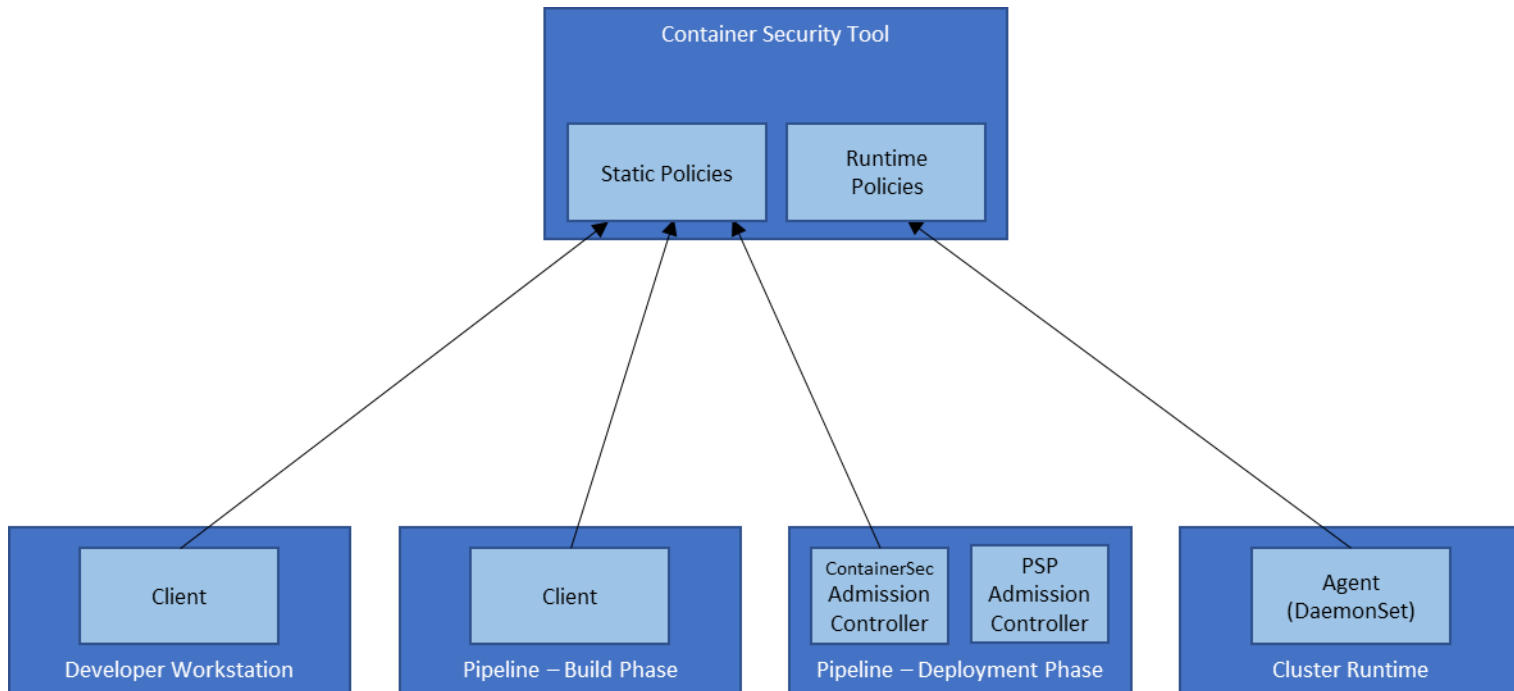
# Basic Pipeline



# Registry

- Access Control: consider human and programmatic
- Secure connections: TLS
- Separation between environments: control image use
- Stale images

# ContainerSec Tool Interactions



# Container Vulnerability & Configuration Flaw Checking

## **Some useful checks against the image filesystem that can be performed:**

- Checking the age of the vulnerability feeds
- Checking if unknown (i.e., non-official) NPM or Ruby feeds are used.
- OS and non-OS packages with vulnerabilities
  - Consider those with fixes vs those without

## **Useful Dockerfile checks:**

- The Dockerfile sets effective user as 'root'
- Dockerfile exposing port tcp/22 (SSH)
- Secrets, such as 'AWS\_SECRET\_KEY', 'password' being included in the image
- The Dockerfile includes secrets set as environment variables
- The Dockerfile contains an instruction to 'ADD' (vs 'COPY')

# Cluster Configuration Control

- Remember it's not just the container images that are changing
- What about cluster configuration?

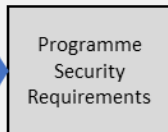
# Gatekeeper

OPA, pronounced "oh-pa"

Rego, pronounced "ray-go"

The audit loop will pick up future changes to policy and pick up "was ok, but now not" configurations

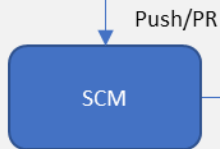
## Plan & Design



Convert to code = "Policy-as-code"

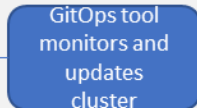
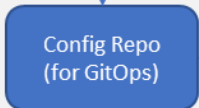
(Deployed via GitOps)

## Development Pipeline



Push/PR

Trigger Pipeline



Gatekeeper Operator

Installs

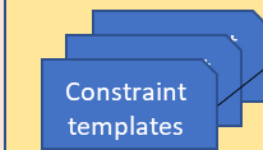
Namespace: gatekeeper-system



Watches

Gatekeeper OPA

Gatekeeper Policy Library



Defines Rego to enforce constraint + schema of constraint

Registers/invokes

## Deployment time

ValidatingWebhook

Object create/update/delete operation

Events

Monitor & remediate configuration drift (manual or automatic)

# Deploy

# Securing Container Deployments: Best Practices

- Use admission control, linked to container security tool policies
- Use securityContexts and associated admission control to:
  - Reduce/remove privilege and running as root
  - If root and privilege must be given; restrict (drop) capabilities to those truly needed
  - Utilise MAC (SELinux, AppArmor etc)
  - Utilise Seccomp where possible (Restricting system calls)
  - Restrict access to underlying host
- Also consider:
  - Image signature checking
  - Image registry source checking



# Summary

- Plan security into the application container lifecycle
- Ensure the 5 key risk areas are secured
- “Security Opportunities”:
  - Code & build = chance to shift-left
  - Deployment = last chance to stop something
  - Runtime = View of what has been deployed and ongoing security status

# Part 3

Best practices for addressing runtime security concerns for containerised applications and Kubernetes environments.

# Securing Container Runtime: Best Practices

- Segregation of privileged vs non-privileged containers by segregating worker nodes
- Only run containers with the same purpose, sensitivity, & threat posture on a single host OS kernel
- Specify resource requests and limits and project/namespace quotas
- Control network communication via microsegmentation

# Monitor Cluster and Workloads

- Monitor the cluster:
  - Network flows
  - Configuration compliance
  - Audit logs
- Monitor workloads:
  - Processes within containers
  - Interactions with the API
  - Vulnerabilities requiring attention

# Summary

We must consider best practices during:

- **Development:** base image – trusted, minimal?, component versions/ages, included packages, tags, configuration of the container image
- **Build:** malware, vulnerability, configuration checking
- **Release & Deployment:** vulnerability and malware, repository, privilege
- **Runtime:** malicious process monitoring, visibility of deployments vs vulnerabilities

And finally...

# Remember:

- Educate your team
- Consider your Host OS
- Deploy Kubernetes in a secure manner
- Separate and control image registries
- Configure the image securely
- Configure the workload object (deployment, statefulset etc) securely
- Check for vulnerabilities and weak configuration in pipeline
- Implement robust admission control for workloads
  - Get everything running with minimal privilege and permissions
- Monitor the runtime

# Useful Links

- Namespaces and cgroups:
  - <https://www.nginx.com/blog/what-are-namespaces-cgroups-how-do-they-work/>
  - <https://medium.com/@BeNitinAgarwal/understanding-the-docker-internals-7ccb052ce9fe>
- Container Image layers: <https://betterprogramming.pub/how-to-improve-docker-image-size-with-layers-3ad62be0da9b>
- Container Runtimes: <https://www.tutorialworks.com/difference-docker-containerd-runc-crio-oci/>
- NIST SP 800-190: <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-190.pdf>
- Kubernetes CIS Benchmark v1.6.1: <https://workbench.cisecurity.org/benchmarks/6083>
- PSP replacement:
  - <https://github.com/kubernetes/enhancements/tree/master/keps/sig-auth/2579-psp-replacement>
  - <https://kubernetes.io/docs/concepts/security/pod-security-standards/>
- Encrypt secrets at rest: <https://kubernetes.io/docs/tasks/administer-cluster/encrypt-data/>
- OPA Gatekeeper: <https://open-policy-agent.github.io/gatekeeper/website/docs/>



# Any questions?

# Thank you

Follow us on:

[ibm.com/security](https://ibm.com/security)

[securityintelligence.com](https://securityintelligence.com)

[ibm.com/security/community](https://ibm.com/security/community)

[xforce.ibmcloud.com](https://xforce.ibmcloud.com)

[@ibmsecurity](https://@ibmsecurity)

[youtube.com/ibmsecurity](https://youtube.com/ibmsecurity)

© Copyright IBM Corporation 2021. All rights reserved. The information contained in these materials is provided for informational purposes only, and is provided AS IS without warranty of any kind, express or implied. Any statement of direction represents IBM's current intent, is subject to change or withdrawal, and represent only goals and objectives. IBM, the IBM logo, and other IBM products and services are trademarks of the International Business Machines Corporation, in the United States, other countries or both. Other company, product, or service names may be trademarks or service marks of others.

Statement of Good Security Practices: IT system security involves protecting systems and information through prevention, detection and response to improper access from within and outside your enterprise. Improper access can result in information being altered, destroyed, misappropriated or misused or can result in damage to or misuse of your systems, including for use in attacks on others. No IT system or product should be considered completely secure and no single product, service or security measure can be completely effective in preventing improper use or access. IBM systems, products and services are designed to be part of a lawful, comprehensive security approach, which will necessarily involve additional operational procedures, and may require other systems, products or services to be most effective. IBM does not warrant that any systems, products or services are immune from, or will make your enterprise immune from, the malicious or illegal conduct of any party.