O'REILLY®

Certified Kubernetes Security Specialist (CKS) Crash Course

Kubernetes 1.23 Edition

About the trainer

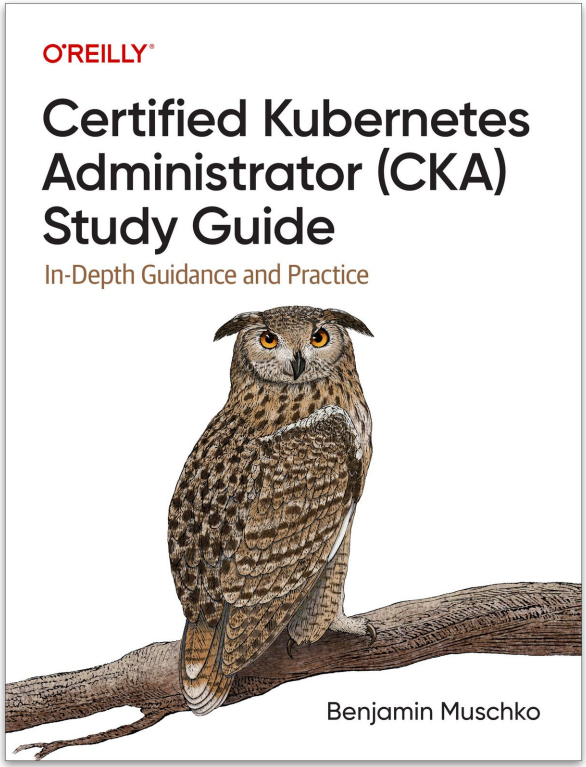
bmuschko

bmuschko

bmuschko.com

AUTOMATED ASCENT

automatedascent.com

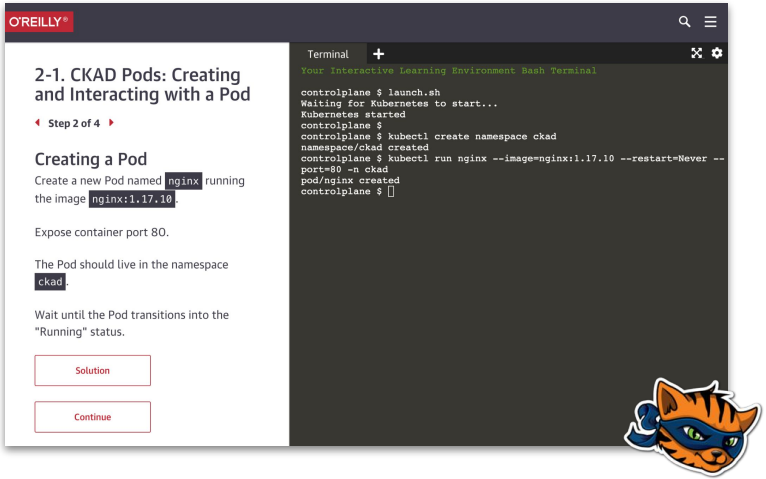


Companion study guide with practice questions

Released in June 2022

Online access on O'Reilly learning platform:

https://learning.oreilly.com/library/view/certified-kubernetes-administrator/9781098107215/



Interactive environment

Kubernetes sandbox

Online access on O'Reilly learning platform:

https://learning.oreilly.com/scenarios/kubernetes-sandbox/9781492062820/

Exam Details and Resources

Objectives, Environment, Time Management

Exam Objectives

“Demonstrate competence with securing container-based applications and Kubernetes platforms.”

The certification program allows users to demonstrate their competence in a hands-on, command-line environment.

https://www.cncf.io/certification/cks/

Exam Prerequisites

Requires CKA certification

You need to be certified at the time of signing up for the CKS exam.

https://www.cncf.io/certification/cka/

Exam Domains & Weights

Domain	Weight
Cluster Setup	10%
Cluster Hardening	15%
System Hardening	15%
Minimize Microservice Vulnerabilities	20%
Supply Chain Security	20%
Monitoring, Logging, and Runtime Security	20%

The Curriculum

10% - Cluster Setup

- Use Network security policies to restrict cluster level access
- Use CIS benchmark to review the security configuration of Kubernetes components (etcd, kubelet, kubedns, kubeapi)
- Properly set up Ingress objects with security control
- Protect node metadata and endpoints
- Minimize use of, and access to, GUI elements
- Verify platform binaries before deploying

15% - Cluster Hardening

- Restrict access to Kubernetes API
- Use Role Based Access Controls to minimize exposure
- Exercise caution in using service accounts e.g. disable defaults, minimize permissions on newly created ones
- Update Kubernetes frequently

15% - System Hardening

- Minimize host OS footprint (reduce attack surface)
- Minimize IAM roles
- Minimize external access to the network
- Appropriately use kernel hardening tools such as AppArmor, seccomp

20% - Minimize Microservice Vulnerabilities

- Setup appropriate OS level security domains e.g. using PSP, OPA, security contexts
- Manage kubernetes secrets
- Use container runtime sandboxes in multi-tenant environments (e.g. gvisor, kata containers)
- Implement pod to pod encryption by use of mTLS

20% - Supply Chain Security

- Minimize base image footprint
- Secure your supply chain: whitelist allowed image registries, sign and validate images
- Use static analysis of user workloads (e.g. kubernetes resources, docker files)
- Scan images for known vulnerabilities

20% - Monitoring, Logging and Runtime Security

- Perform behavioral analytics of syscall process and file activities at the host and container level to detect malicious activities
- Detect threats within physical infrastructure, apps, networks, data, users and workloads
- Detect all phases of attack regardless where it occurs and how it spreads
- Perform deep analytical investigation and identification of bad actors within environment
- Ensure immutability of containers at runtime
- Use Audit Logs to monitor access

Candidate Skills

kubernetes

Architecture & Concepts

`kubectl`

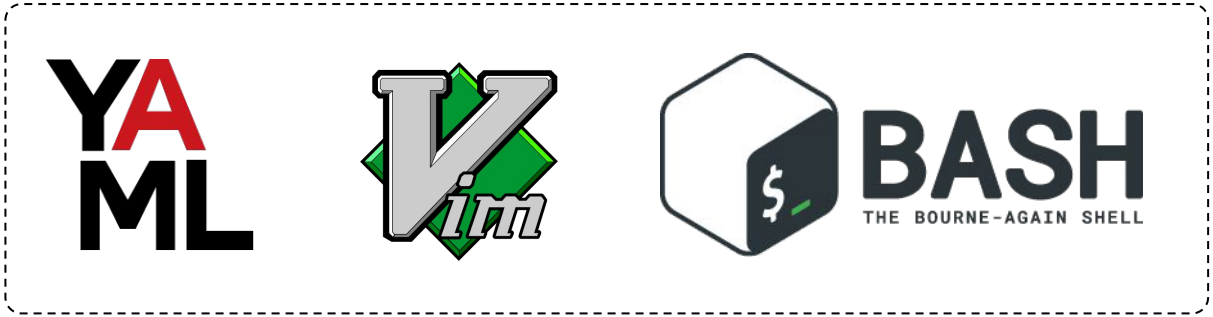
Running Commands

container runtime

Underlying Concepts

Exam Environment

Online and proctored exam



The trinity of tooling you need to be familiar with

Using Kubernetes Documentation

Kubernetes docs and subdomains (see FAQ)

- Docs: https://kubernetes.io/docs
- GitHub: https://github.com/kubernetes
- Blog: https://kubernetes.io/blog

Using Third-Party Documentation

Third-party tools are part of the exam (see FAQ)

- Trivy: https://github.com/aquasecurity/trivy
- Falco: https://falco.org/docs
- AppArmor:

  https://gitlab.com/apparmor/apparmor/-/wikis/Documentation

Bookmarking Documentation

You are allowed to use bookmarks during exam

- Create bookmarks of documentation you find important while practicing
- Only open “allowed” pages in a single tab

Getting Help on a Command

Render subcommands and options with `--help`

```
$ kubectl create --help
Create a resource from a file or from stdin.

JSON and YAML formats are accepted.

...
Available Commands:
...
  configmap           Create a configmap from a local file, directory or literal
value
  deployment          Create a deployment with the specified name.
...

Options:
...
```

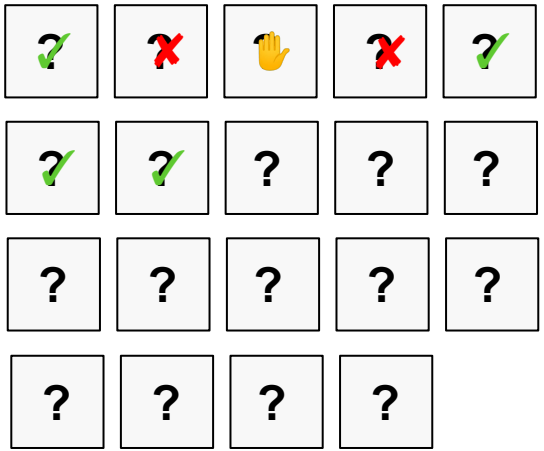
Zeroing in on Command Details

Drill into object details with the `explain` command

```
$ kubectl explain pods.spec
KIND:     Pod
VERSION:  v1

RESOURCE: spec <Object>

DESCRIPTION:
...

FIELDS:
...
```

Time Management

# of problems in 2 hours, use your time wisely!



Configuring Auto-Completion

Allowed during exam, configurable on-demand

```
$ kubectl cre<tab>
```

https://kubernetes.io/docs/tasks/tools/included/optional-kubectl-configs-bash-linux/

Using an Alias for kubectl

Already set up in exam environment (see FAQ)


```
$ alias k=kubectl
$ k version
...
```

Setting Namespace for a Context

Questions will ask you to run a command on a specific cluster - Make sure to execute it!

```
$ kubectl config set-context <context-of-question>↵
  --namespace=<namespace-of-question>
$ kubectl config use-context <context-of-question>
```

Internalize Resource Short Names

Some API resources provide a shortcut

```
$ kubectl get ns
```

Usage of `ns` instead of `namespaces`

```
$ kubectl describe pvc claim
```

Usage of `pvc` instead of `persistentvolumeclaim`

Deleting Kubernetes Objects

Don’t wait for a graceful deletion of objects...

```
$ kubectl delete pod nginx --force
```

Understand and Practice bash

Practice relevant syntax and language constructs

```
$ if [ ! -d ~/tmp ]; then mkdir -p ~/tmp; fi; while true;↲
do echo $(date) >> ~/tmp/date.txt; sleep 5; done;
```

Finding Object Information

Filter configuration with context from a set of objects

```
$ kubectl describe pods | grep -C 10 "author=John Doe"
$ kubectl get pods -o yaml | grep -C 5 labels:
```

`grep` is your friend!

How to Prepare

Practice, practice, practice!

The key to cracking the exam

Q & A

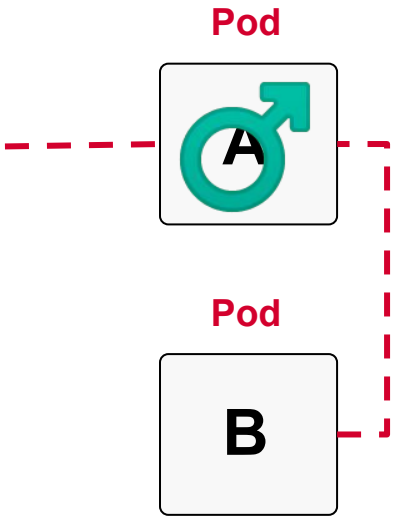
BREAK

Cluster Setup

Network Security Policies, CIS Benchmark, Protecting nodes and GUI, Ingress, Verifying Binaries

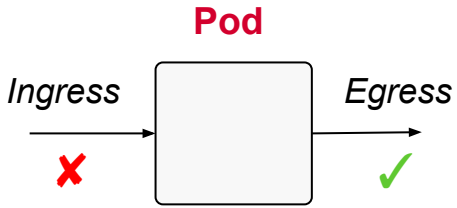
Scenario: Gaining Access to Pod

Once a Pod can be compromised, it can easy access other Pods



Network Policy Rules

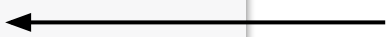
Which Pod does the rule apply to? ✔



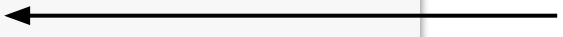
Which direction of traffic?
Who is allowed?

Creating a Network Policy

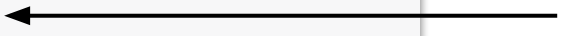
```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: my-network-policy
spec:
  podSelector:
    matchLabels:
      tier: frontend
  policyTypes:
  - Ingress
  - Egress
  ingress:
  - from:
    ...
  egress:
  - to:
    ...
```



Label selection for Pods



Inbound/outbound traffic



Who can connect to Pod?
Where can Pod connect to?

General Rule of Thumb

Start by denying all access and allowing access as needed

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny
spec:
  podSelector: {}
  policyTypes:
  - Ingress
  - Egress
```

Applies to all Pods

Inbound/outbound traffic is blocked

https://kubernetes.io/docs/concepts/services-networking/network-policies/#default-deny-all-ingress-and-all-egress-traffic

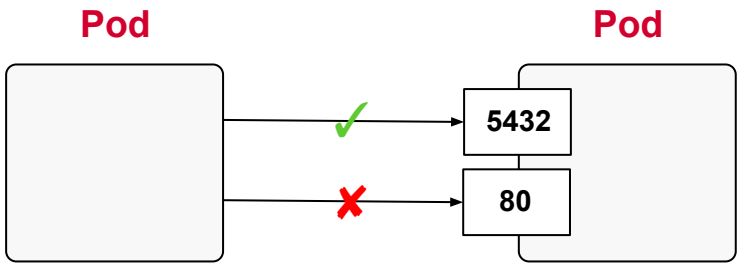
Behavior of `from`/`to` Selectors

Select by Namespace, Pod and IP address

```
...
ingress:
- from:
  - namespaceSelector:
      matchLabels:
        app: awesome
  - podSelector:
      matchLabels:
        tier: backend
...
```

Allow incoming traffic from Pod that matches the label `tier=backend` in the namespace `awesome`

Restricting Access to Ports

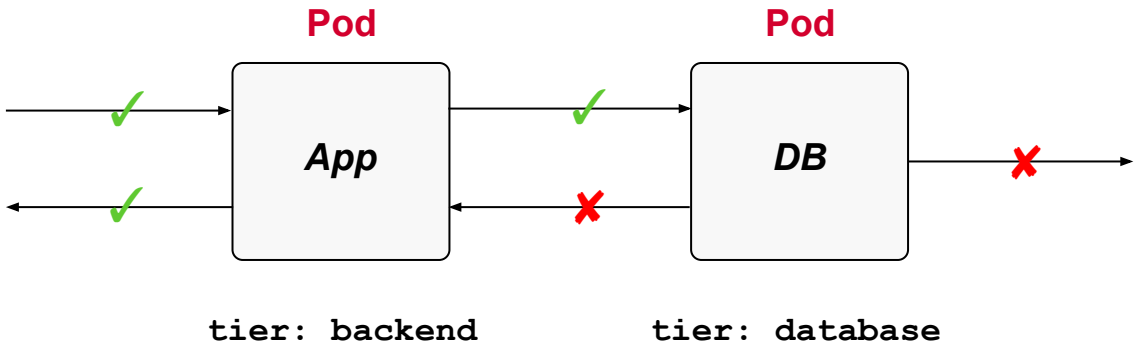


By default all ports are open

```
...
ingress:
- from:
  - podSelector:
      matchLabels:
        tier: backend
  ports:
  - protocol: TCP
    port: 5432
...
```

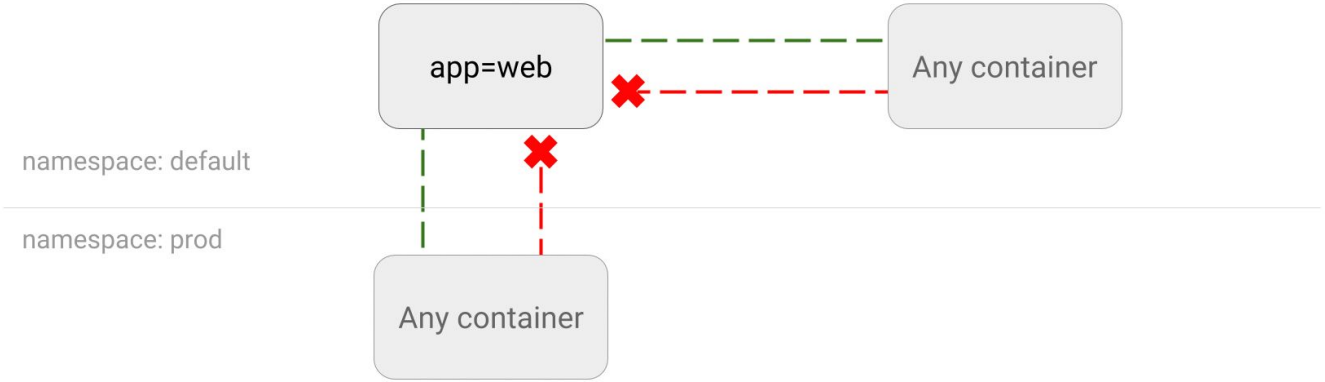
Representative Use Case

“Application makes request to database but database cannot make any outgoing requests.”



Additional Learning Resource

Network Policies explained by use case and visualization



https://github.com/ahmetb/kubernetes-network-policy-recipes

EXERCISE

Restricting
Communication
Between Pods

CIS Benchmarks

Guidelines and benchmark tests for best practices in securing code

- Published by non-profit Center of Internet Security (CIS)
- CIS Kubernetes Benchmark provides tests for Kubernetes environments

Using kube-bench

Tool by Aqua for identifying and reporting issues (based on CIS)

```
$ kubectl apply -f
https://raw.githubusercontent.com/aquasecurity/kube-benc
h/main/job.yaml
job.batch/kube-bench created

$ kubectl get pods
NAME                READY   STATUS      RESTARTS   AGE
kube-bench-8f6qh    0/1     Completed   0          45s

$ kubectl logs kube-bench-8f6qh
...
```

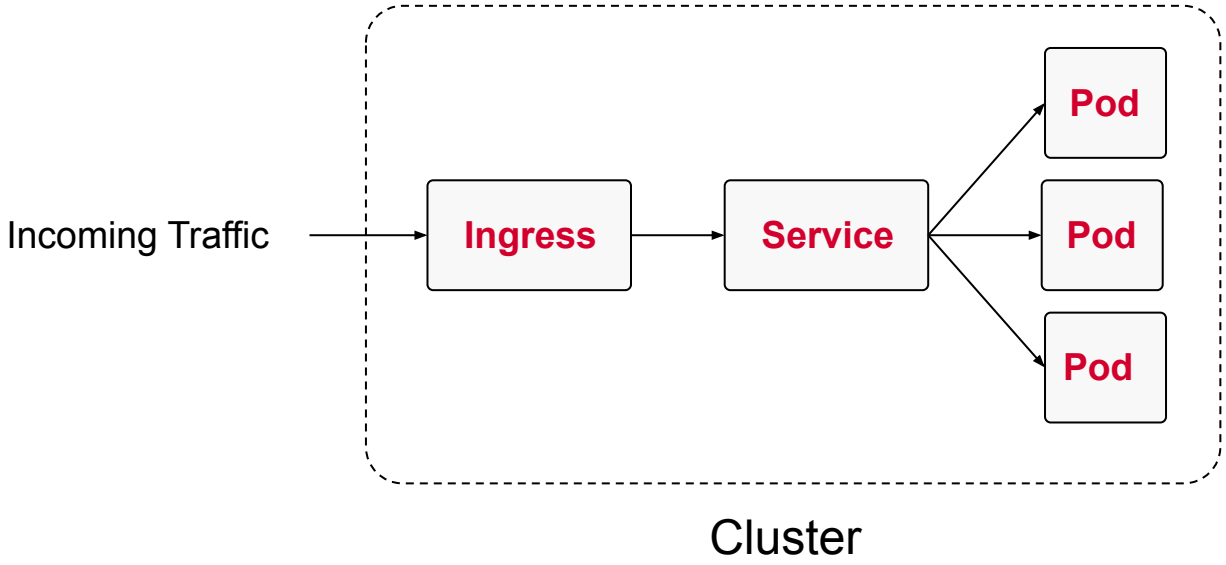
Results produced by kube-bench

Tests for different types of nodes and Pods running on them

```
[INFO] 1 Master Node Security Configuration
[INFO] 1.1 Master Node Configuration Files
[PASS] 1.1.1 Ensure that the API server pod specification file permissions are
set to 644 or more restrictive (Automated)
[INFO] 1.2 API Server
[FAIL] 1.2.5 Ensure that the --kubelet-certificate-authority argument is set as
appropriate (Automated)
...
== Summary total ==
61 checks PASS
14 checks FAIL
48 checks WARN
0 checks INFO
```

Understanding Ingress

Manages external access to the services in a cluster via HTTP(S)



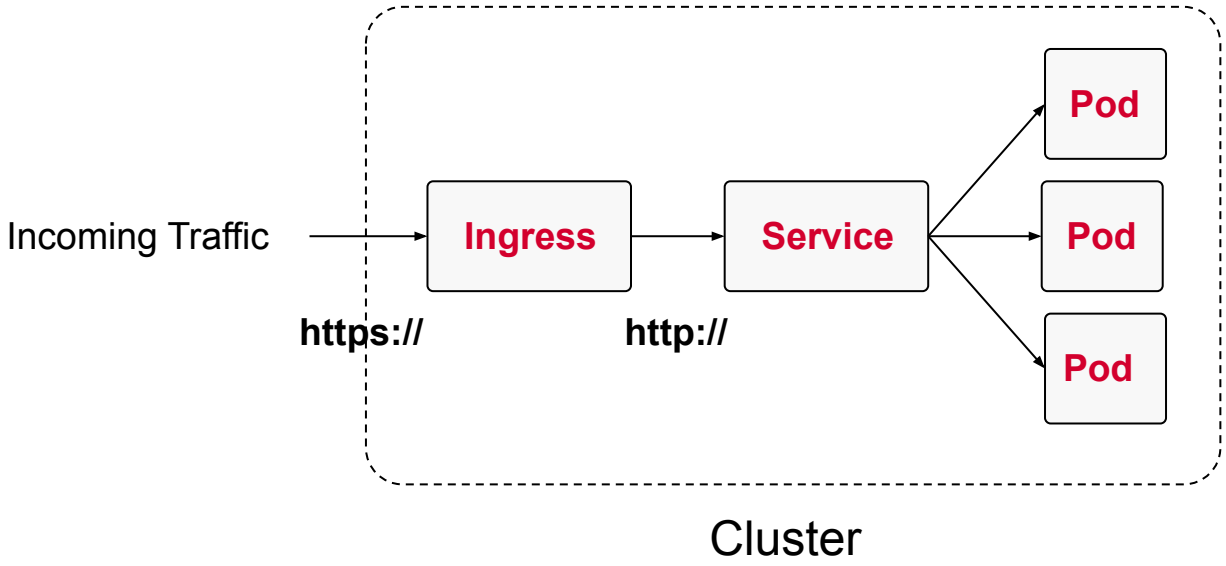
Detailed Coverage

The CKA exam lays the foundation for Ingress

- Certified Kubernetes Administrator (CKA) Study Guide
- Exercise: Creating an Ingress

Ingress with TLS termination

Use HTTPS for Ingress, route to Service with HTTP



Creating the Secret

Requires the type `kubernetes.io/tls`

Create TLS certificate and key

```
$ openssl req -nodes -new -x509 -keyout tls-ingress.key -out
  tls-ingress.crt -subj "/CN=ingress.tls"
```

Create the Secret

```
$ kubectl create secret tls tls-ingress-secret
  --cert=tls-ingress.crt --key=tls-ingress.key
```

Creating the Ingress

Assign Secret upon creation

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: tls-ingress
spec:
  tls:
  - hosts:
    - bar.foo.com
    secretName: tls-ingress-secret
  rules:
  - host: bar.foo.com
    ...
```

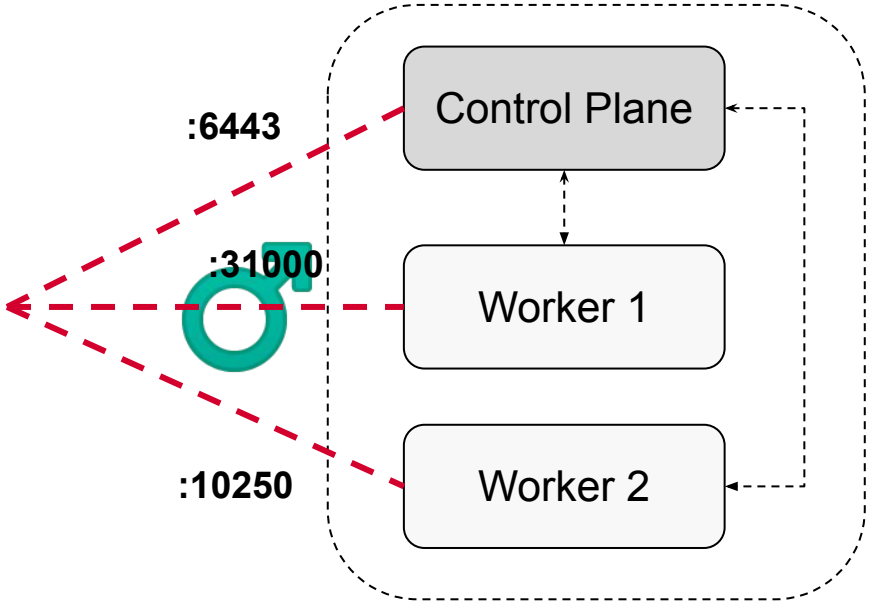
```
$ kubectl create ingress tls-ingress
--rule="bar.foo.com/=web-app-service
:8080,tls=tls-ingress-secret"
```

EXERCISE

Creating an Ingress
with TLS
termination

Scenario: Node Port Scans

Nodes need to expose ports for communication



Default Incoming Node Ports

Control Plane Node

Port Range	Purpose
6643	Kubernetes API server
2379-2380	etcd server client API
10250	Kubelet API
10259	kube-scheduler
10257	kube-controller-manager

Worker Node(s)

Port Range	Purpose
10250	Kubelet API
30000-32767	NodePort Services

Securing Node Endpoints

Only expose ports you need

- Minimize attack surface area
- Limit exposure of ports
- Instantiate firewall rules to protect against access of endpoints

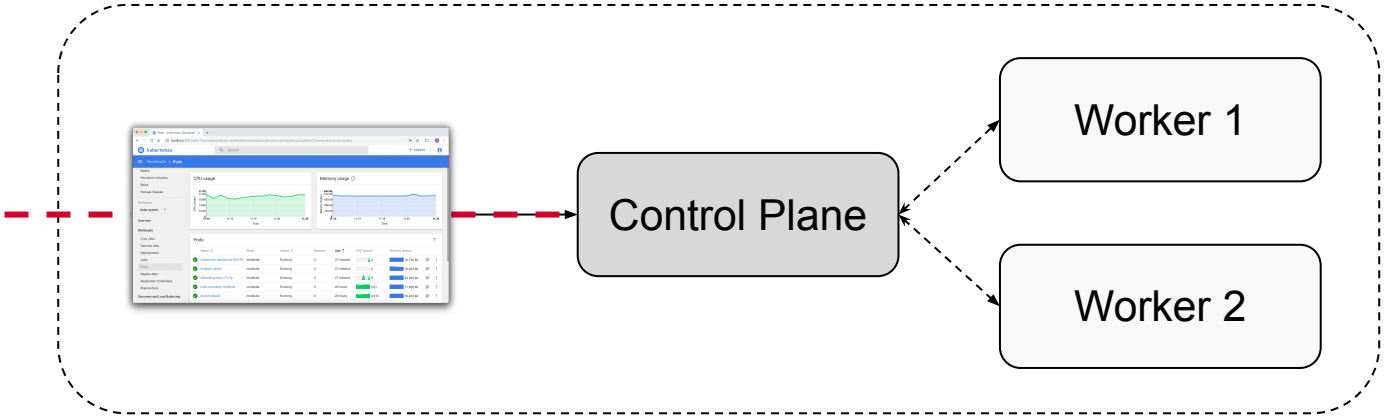
Securing Node Metadata

Retrieving cluster metadata via API endpoint

- Cloud provider Kubernetes environments can expose metadata via an API endpoint (including cloud credentials)
- Those endpoints are accessible by Pods running on the same node
- Limit permissions given to instance credentials and stand up network policies to restrict Pod access

Scenario: Attack on GUI

Dashboards grant access the control plane node



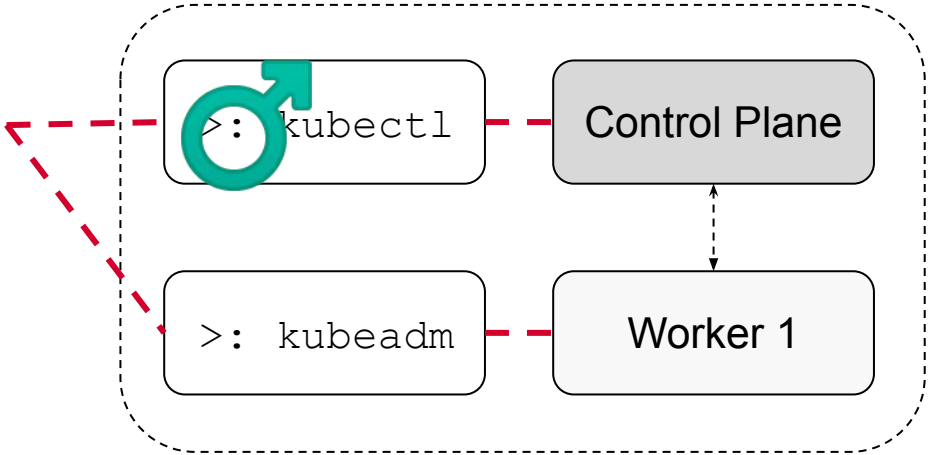
Securing GUI Elements

Convenient management of Kubernetes objects

- Exposing GUIs for Kubernetes provide attackers with a way to control the cluster
- Limit access to those GUIs (e.g. with firewall rules, RBAC permissions)
- Some organizations go as far as removing all GUI access to minimize attack surface

Scenario: Malicious Code Inject.

Gain control to cluster by modifying the byte code



Verifying Kubernetes Binaries

Use checksum to ensure that binary hasn't been tampered with

- Typical Kubernetes binaries are `kubectl`, `kubeadm`, and kubelet
- SHA256 checksum file should match the generated checksum of binary
- If both values don't match, then the file has been modified e.g. malicious code has been injected

Retrieving the Checksum File

Determine version, then use download link

Determine version of binary

```
$ kubectl version --client --short  # DO NOT RUN THIS!
Client Version: v1.23.5
```

Download corresponding checksum file

```
$ curl -LO
"https://dl.k8s.io/v1.23.5/bin/linux/amd64/kubectl.sha256"
```

List of Download URLs

Replace version, OS, and architecture to reflect your platform

- `kubectl`: https://dl.k8s.io/v1.23.5/bin/linux/amd64/kubectl.sha256
- `kubeadm`: https://dl.k8s.io/v1.23.5/bin/linux/amd64/kubeadm.sha256
- kubelet: https://dl.k8s.io/v1.23.5/bin/linux/amd64/kubelet.sha256
- kube apiserver: https://dl.k8s.io/v1.23.5/bin/linux/amd64/kube-apiserver.sha256

Verifying the Binary

Use sha256 tool to compare binary checksum (docs)

Linux

```
$ echo "$(cat kubectl.sha256)  kubectl" | sha256sum --check
```

MacOSX

```
$ echo "$(cat kubectl.sha256)  kubectl" | shasum -a 256 --check
```

EXERCISE

Verifying Existing
Kubernetes
Binaries

Cluster Hardening

Restricting API Access, RBAC for Minimizing Exposure, Service Accounts, Keeping Kubernetes Updated

RBAC High-Level Overview

Three key elements for understanding concept

Subject

Groups
Users
Service-Accounts

API Resources

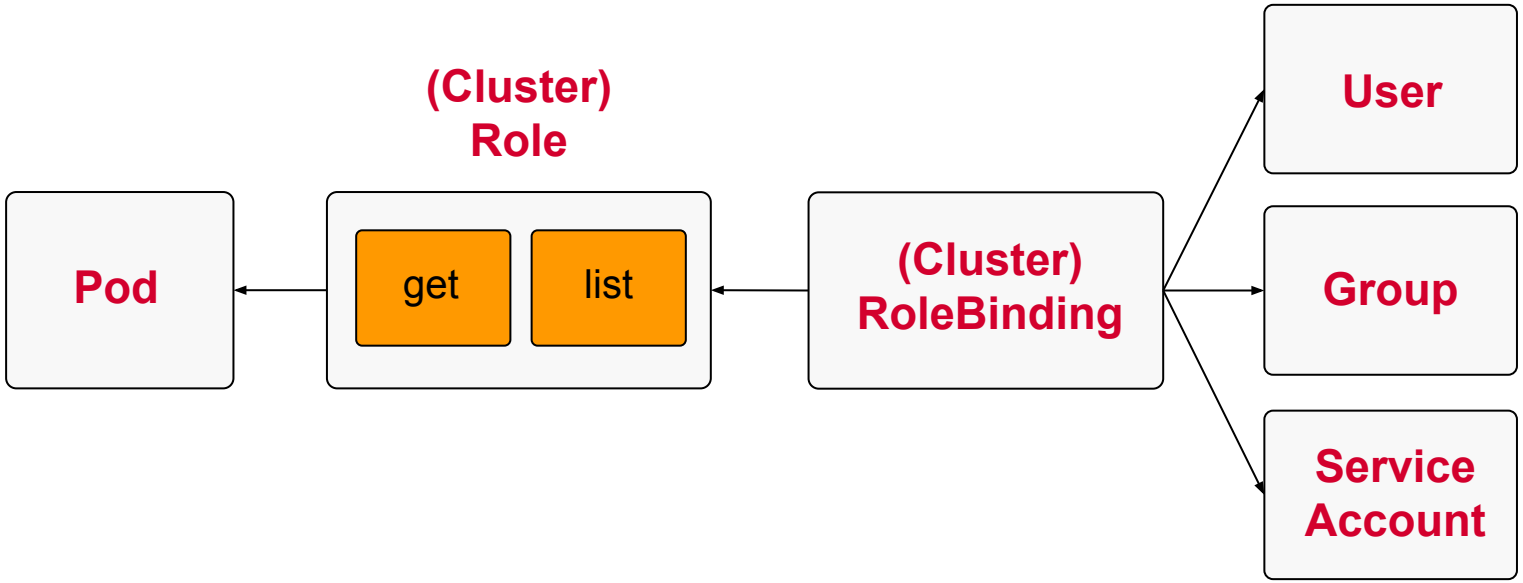
ConfigMap
Pod
Deployment
Node
...

Operations (Verbs)

create
list
watch
delete
...

Involved RBAC Primitives

Restrict access to API resources based on user roles



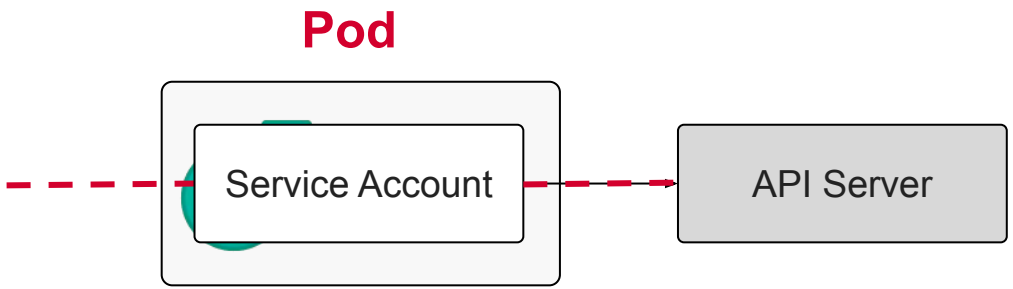
Detailed Coverage

The CKA exam lays the foundation for RBAC

- Certified Kubernetes Administrator (CKA) Study Guide
- Exercise: RBAC for a User

Scenario: Pod + Service Account

A Service Account with too many permissions that can be misused



Service Account Permissions

Restrict as much as possible

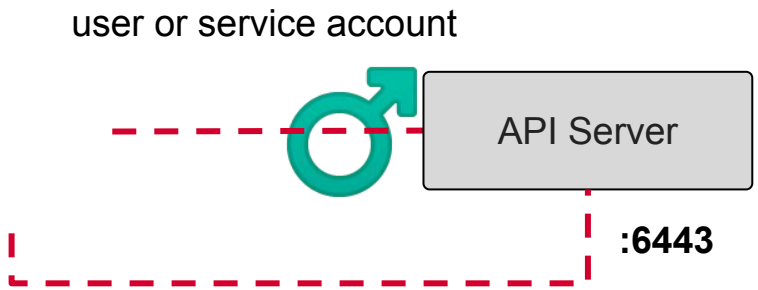
- Service Account should only allow be allowed the permissions needed for the process running in Pod
- Restricting those permissions is implemented via (Cluster)Role and (Cluster)RoleBinding (combination of verbs + API resources)
- An attacker of a compromised Pod using the Service Account gains access to operations executed against the API server (e.g. delete Pods)

EXERCISE

Minimizing Service Account Permissions and Usage

Scenario: Attack on API server

Used compromised user/service account or by port 6443



Securing API Endpoints

Restrict as much as possible

- Limit outside network access by defining firewall rules
- Only expose API endpoints via intranet
- User permissions should be limited and on point in case they get compromised

Scenario: Exploiting Vulnerability

Older Kubernetes versions can expose critical issues

1.7

https://www.cvedetails.com/vulnerability-list/vendor_id-15867/product_id-34016/Kubernetes-Kubernetes.html

Updating Kubernetes Frequently

Upgrade early and often!

- Versioning schema follows semantic versioning e.g. 1.23.5
- Check discovered vulnerabilities and upgrade to latest patch and minor version
- Make concerted effort to upgrade to next major version (they may contain breaking changes

Detailed Coverage

The CKA exam lays the foundation for cluster management

- Certified Kubernetes Administrator (CKA) Study Guide
- Exercise: Upgrading a cluster version

System Hardening

Minimizing OS Footprint, IAM Roles, and External Network Access, Kernel Hardening Tools

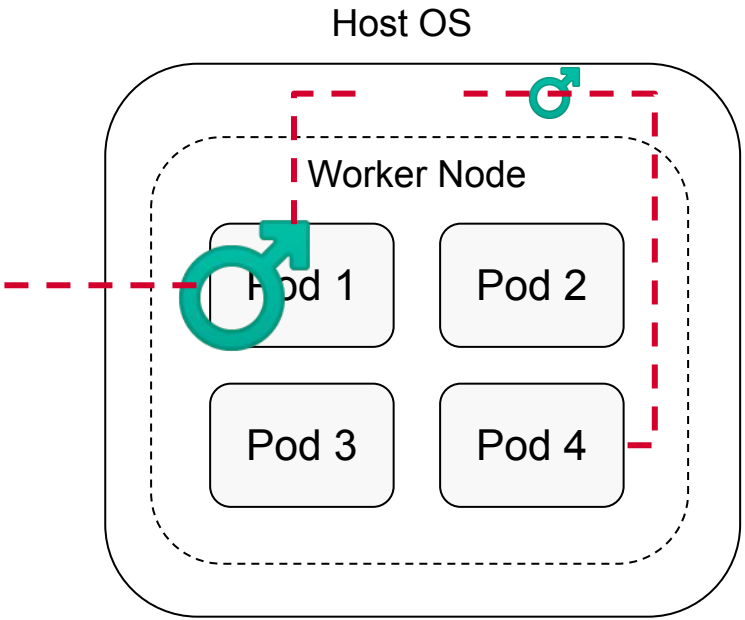
Container Namespaces

Containers only have limited access to OS namespace

- Regular processes run in OS namespace
- Containers run in container namespace isolated from OS namespace
- This leaves less room for attackers to gain access to host
- **Warning:** Containers can be configured to use OS namespace

Scenario: Pod to Host OS Attack

Compromised Pod gains access to other Pods via host OS



Enabling Host Namespace Usage

The Pod spec defined a handful of attributes for configuration

YAML Element	Purpose
`spec.hostIPC`	The host's inter-process communication namespace
`spec.hostNetwork`	The host's network namespace
`spec.hostPID`	The host's process ID namespace

```
apiVersion: v1
kind: Pod
metadata:
  name: alpine
spec:
  hostIPC: true
  hostNetwork: true
  hostPID: true
  containers:
  - name: alpine
    image: alpine/curl:3.14
```

Enabling Privileged Mode

Access to host-level resources can be granted via security context

```
apiVersion: v1
kind: Pod
metadata:
  name: alpine
spec:
  containers:
  - name: alpine
    image: alpine/curl:3.14
    securityContext:
        privileged: true
```

Equivalent to granting root access on host. Avoid if not absolutely necessary!

Granting Host Access to Pods

Only grant those permissions if needed

- By default, host access and privileged mode are disabled
- Enabling any of those settings opens an attack vector to the host
- Host access and privileged mode should only be enabled if really required

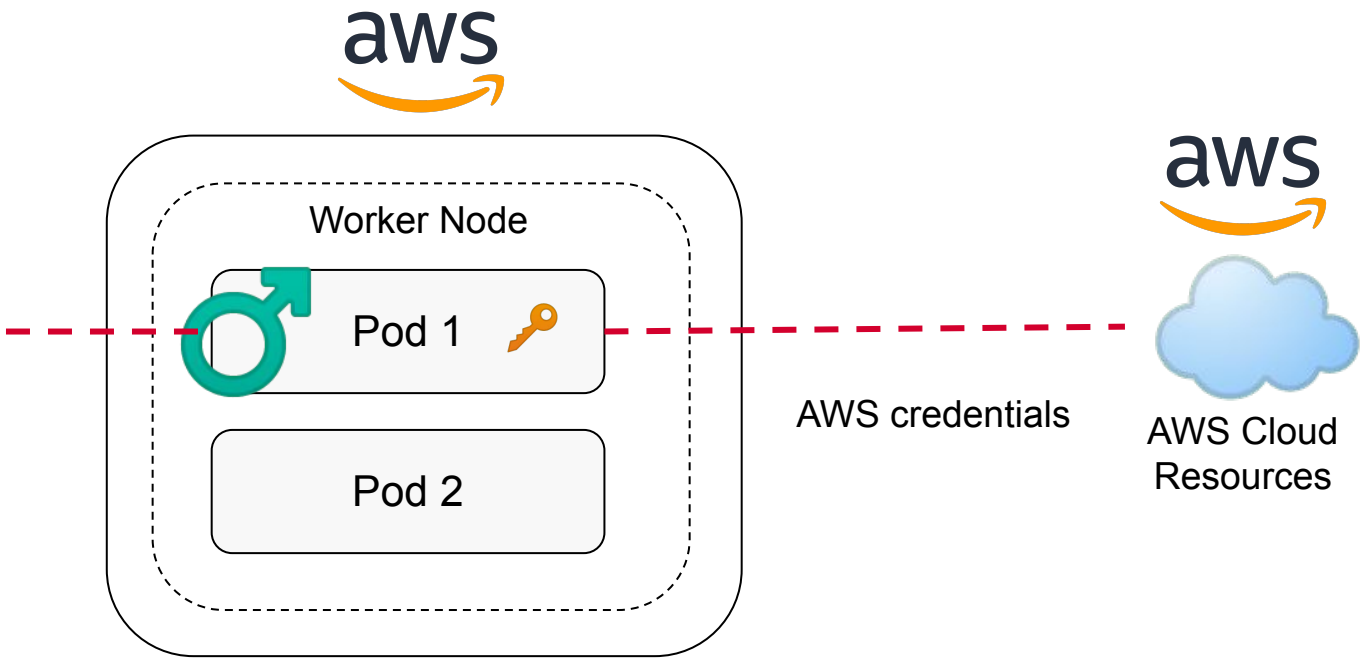
IAM Roles on aws

Obtain permissions/credentials to access AWS resources

- Only relevant if Kubernetes runs on AWS
- Containers can retrieve credentials if run on AWS and gain access to other resources

Scenario: Misusing AWS creds.

Compromised Pod can gain access to other AWS resources



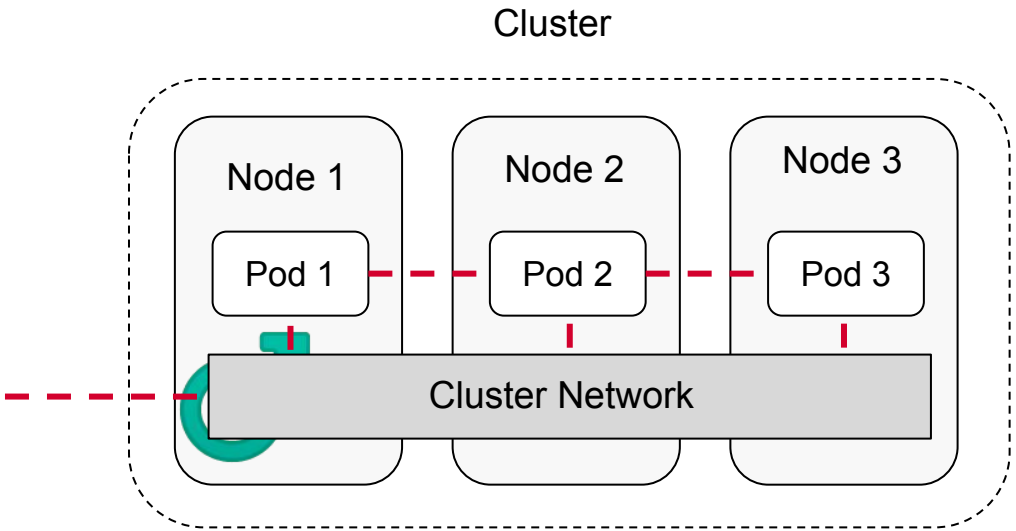
Handling IAM Roles

Follow those guidelines if you are running Kubernetes on AWS

- Only grant the minimal set of IAM roles required to do the job
- Block access from containers that don’t need IAM roles (e.g. via firewall, Network Policies…)

Scenario: Infiltr. Cluster Network

The cluster network spans across all nodes



Minimizing Ext. Network Access

By default, a Pod can talk to any other Pod in the cluster

- The cluster network spans across all nodes and enables Pods to communicate with each other
- Access to the cluster networks grants access to all Pods running on all nodes
- Limit network access from the outside

Linux Security Kernel Modules

Application that controls what programs can run on host OS

- AppArmor: Protects the OS from external or internal threats, even zero-day attacks, by enforcing good behavior and preventing flaws from being exploited

- seccomp: Restricts the system calls of an application to the kernel

AppArmor Installation Check

Usually exists on Linux system independent of Kubernetes

```
$ aa-status
apparmor module is loaded.
33 profiles are loaded.
33 profiles are in enforce mode.
...
```

Minikube does not support an AppArmor add-on at this time

AppArmor Profiles

A set of rules that define what a program is allowed/denied to do

```
#include <tunables/global>

profile k8s-deny-write flags=(attach_disconnected)
{
  #include <abstractions/base>

  file,

  # Deny all file writes.
  deny /** w,
}
```

Enforcing AppArmor Profile

Store the profile in a file and use `apparmor_parser` command

```
$ sudo apparmor_parser /etc/apparmor.d/k8s-deny-write

$ aa-status
apparmor module is loaded.
32 profiles are loaded.
32 profiles are in enforce mode.
   k8s-deny-write
   ...
```

The command uses enforce mode by default. Use `-C` option for complain mode. **Needs to be enabled on node that runs the Pod.**

AppArmor Profile in Pod

Assign annotation, profile needs to be activated on node

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: hello-apparmor
  annotations:
    container.apparmor.security.beta.kubernetes.io/hello:
    localhost/k8s-allow-write
spec:
  containers:
  - name: hello
    image: busybox:1.28
    command: ["sh", "-c", "echo 'Hello AppArmor!' && sleep 1h"]
```

EXERCISE

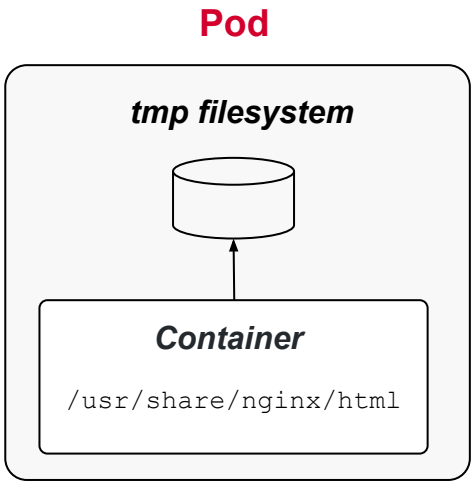
Preventing Network
Calls From a Pod
With AppArmor

Minimizing Microservice Vulnerabilities

OS-Level Security Domain, Secrets, Container Runtime Sandboxes, mTLS Encryption

Understanding Security Contexts

Privilege and access control settings for a Pod or container

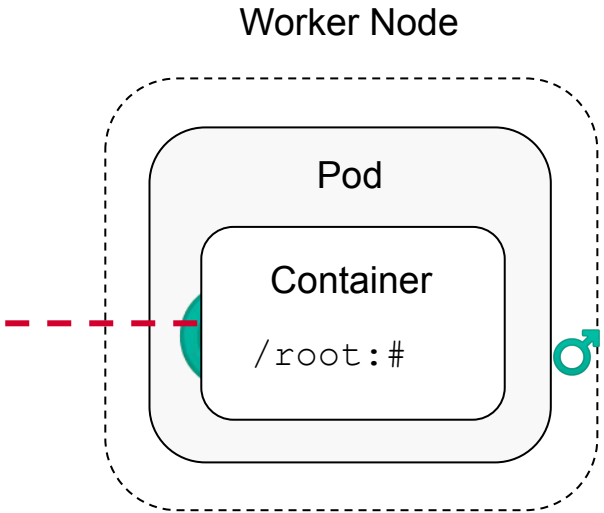


“Create files with a specific Unix group ID”

“Run this container with a specific Unix user ID”

Scenario: Using root Credentials

By default, containers run with root privileges



Defining a Security Context

Pod- vs. container-level definition

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: secured-pod
spec:
  securityContext:
    runAsUser: 1000
  containers:
  - image: nginx:1.18.0
    name: secured-container
    securityContext:
      runAsGroup: 3000
```

Defined on the Pod-level

Defined on the container-level

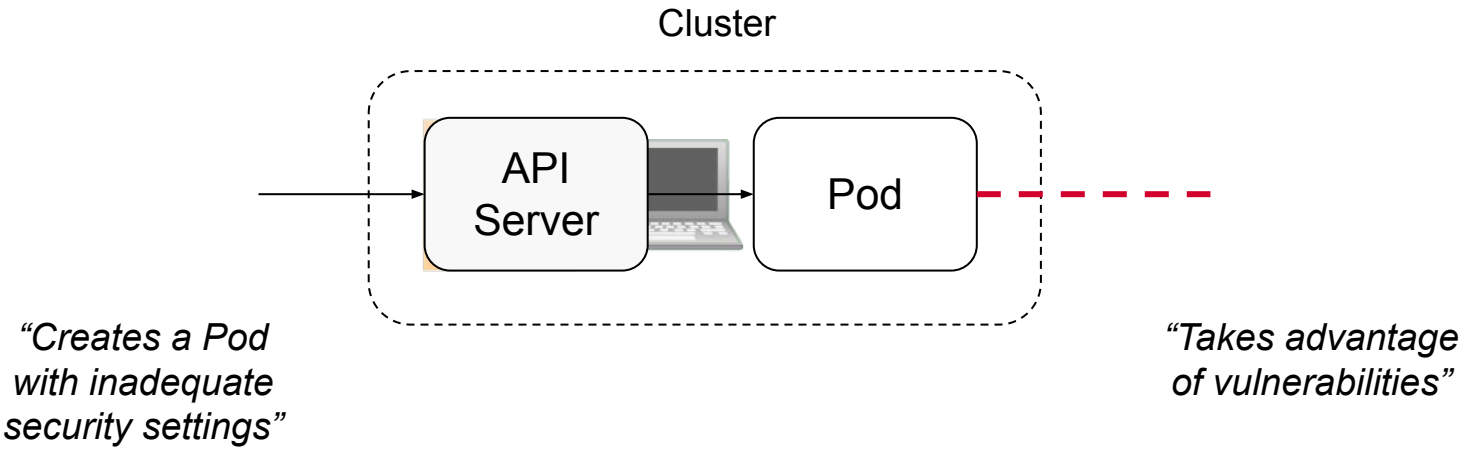
Security Context API

Only partial overlap for Pod- and container attributes

API	Description
PodSecurityContext	Defines Pod-level security attributes.
SecurityContext	Defines container-level security attributes.

Scenario: Governing Pod Security

Developers may be less familiar with Pod security best practices



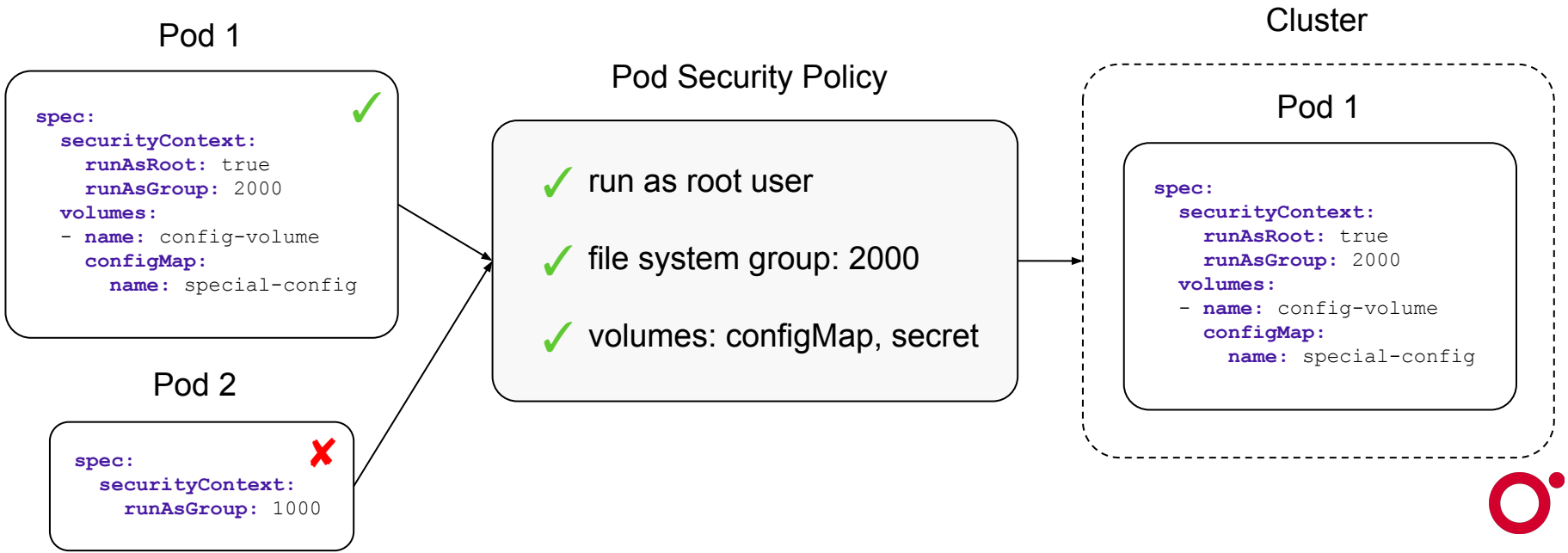
Pod Security Policy Overview

Administration-level security feature for Pods

- Controls which security settings should be used for Pods
- Can reject Pods that don’t follow the desired security settings
- Examples: Control which types of Volumes are allowed to be mounted, prevent running a container privileged mode or as root user

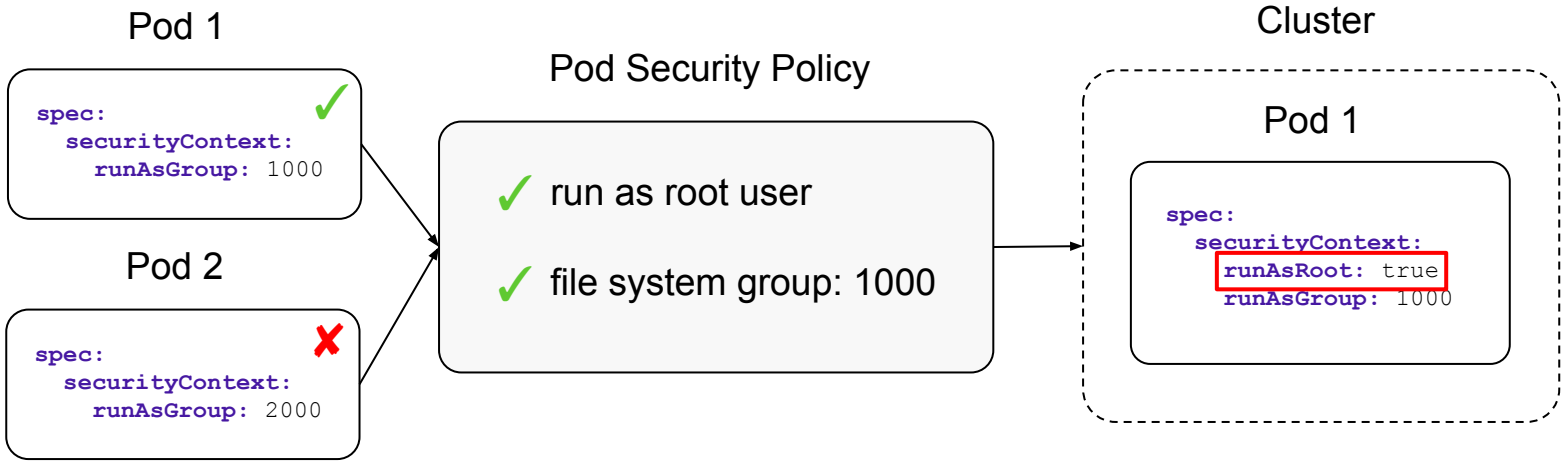
Pod Security Policy Enforcement

Requires Pods to set desired security configuration



Pod Security Policy Governance

Automatically assign default security configuration or reject



Deprecation Warning

Still relevant to CKS exam, PSA not in curriculum yet

Pod Security Policies

FEATURE STATE: `Kubernetes v1.21 [deprecated]`

> **Caution:** PodSecurityPolicy is deprecated as of Kubernetes v1.21, and **will be removed in v1.25**. We recommend migrating to Pod Security Admission, or a 3rd party admission plugin. For a migration guide, see Migrate from PodSecurityPolicy to the Built-In PodSecurity Admission Controller. For more information on the deprecation, see PodSecurityPolicy Deprecation: Past, Present, and Future.

PSP Admission Controller

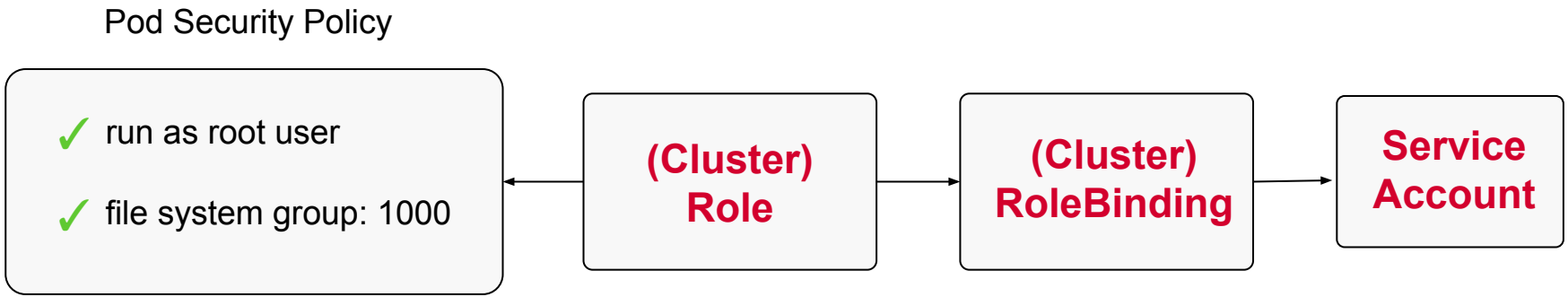
PodSecurityPolicy admission plugin needs to enabled

```
$ vim /etc/kubernetes/manifests/kube-apiserver.yaml
spec:
  containers:
  - command:
    - kube-apiserver
    - ...
    - --enable-admission-plugins=NodeRestriction,PodSecurityPolicy
```

```
$ minikube start
--extra-config=apiserver.enable-admission-plugins=PodSecurityPolicy
--addons=pod-security-policy
```

Pod Security Policy & RBAC

To let a policy take effect, you need to tie it in with RBAC



Defining a Pod Security Policy

No imperative command available, extensive configuration

```
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
  name: non-privileged
spec:
  privileged: false
  seLinux:
    rule: RunAsAny
  supplementalGroups:
    rule: RunAsAny
  runAsUser:
    rule: RunAsAny
  fsGroup:
    rule: RunAsAny
  volumes:
  - '*'
```

List of policies in reference

Defining the ClusterRole

Rule is pretty much set, assign the resource name

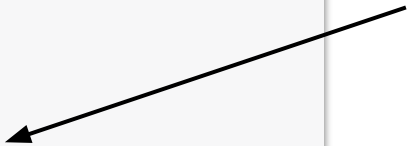
```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: psp-clusterrole
rules:
- apiGroups: ['policy']
  resources: ['podsecuritypolicies']
  verbs:     ['use']
  resourceNames:
  - non-privileged
```

Name of Pod Security Policy

Defining the (Cluster)RoleBinding

Apply service account via Group subject, ignore others

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: psp-clusterrolebinding
roleRef:
  kind: ClusterRole
  name: psp-clusterrole
  apiGroup: rbac.authorization.k8s.io
subjects:
- kind: Group
  apiGroup: rbac.authorization.k8s.io
  name: system:serviceaccounts:<authorized namespace>
```



Recommended service account configuration using the `Group` kind

EXERCISE

Enforcing a Pod Security Policy Upon Pod Creation

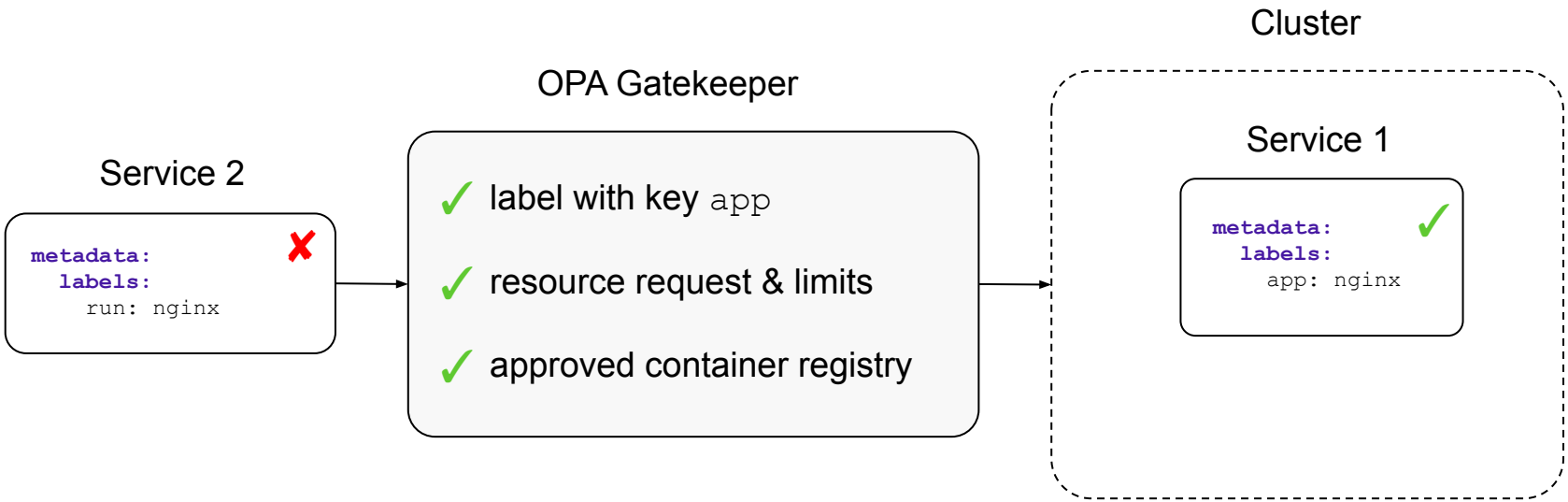
OPA Gatekeeper

Customization for Kubernetes objects at creation time

- Framework for defining constraints for any kind of objects, not just Pods
- Highly-flexible policy definition
- Examples: Ensure that certain labels are defined, which image registry are allowed to be used for pulling images

OPA Enforcement

Require the definition of a specific label key



OPA Key Elements

Three key elements important to understanding concept

- Constraint template: Defines the schema for the constraint with the help of Rego logic
- Constraint: Assigns the template to a set of primitives
- Gatekeeper: Components installed in the cluster that enforce the constraints.

OPA Constraint Template

No imperative command available, extensive configuration

```yaml
apiVersion: templates.gatekeeper.sh/v1beta1
kind: ConstraintTemplate
metadata:
  name: k8srequiredlabels
spec:
  crd:
    spec:
      names:
        kind: K8sRequiredLabels
      validation:
        # Schema for the `parameters` field
        openAPIV3Schema:
          properties:
            labels:
              type: array
              items: string
  targets:
    - target: admission.k8s.gatekeeper.sh
      rego: |
        package k8srequiredlabels

        deny[{"msg": msg, "details": {"missing_labels": missing}}] {
          provided := {label | input.review.object.metadata.labels[label]}
          required := {label | label := input.parameters.labels[_]}
          missing := required - provided
          count(missing) > 0
          msg := sprintf("you must provide labels: %v", [missing])
        }
```

Required parameters and their data types to be passed in

Constraint logic defined with the declarative language Rego

OPA Constraint

Maps constraint logic to specific primitives & defines params

```yaml
apiVersion: constraints.gatekeeper.sh/v1beta1
kind: K8sRequiredLabels
metadata:
  name: ns-must-have-hr
spec:
  match:
    kinds:
      - apiGroups: [""]
        kinds: ["Namespace"]
  parameters:
    labels: ["hr"]
```

Name of constraint template

Primitives the constraint applies to

Values for parameters

OPA Gatekeeper Installation

Deployable with a single command

Deploying a release using prebuilt image

```
$ kubectl apply -f
https://raw.githubusercontent.com/open-policy-agent/gatekeeper
/release-3.7/deploy/gatekeeper.yaml
```

Ensure that Gatekeeper Pods are running

```
$ kubectl get pods -n gatekeeper-system
```

OPA Gatekeeper Validation

Enforcement during object creation

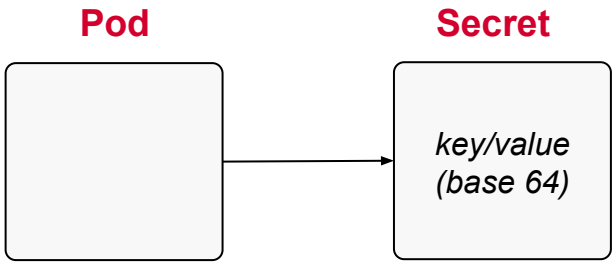
```
$ kubectl create ns hello
Error from server ([ns-must-have-hr] you must provide labels:
{"gatekeeper"}): admission webhook "validation.gatekeeper.sh"
denied the request: [ns-must-have-hr] you must provide labels:
{"hr"}
```

EXERCISE

OPA Gatekeeping for Pods

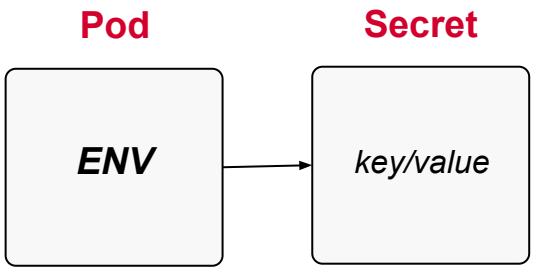
Centralized Configuration Data

Stores sensitive data, consumed by Pod

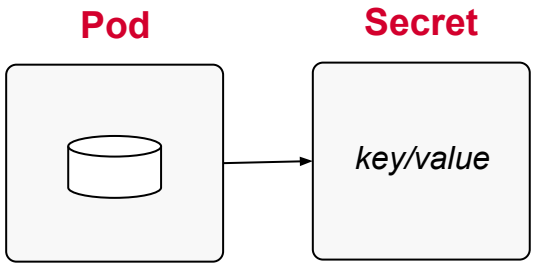


Mounting a Secret

Two options for consuming data



Injected as environment variables



Mounted as volume

Secret Options

Imperative command: `kubectl create secret`

Option	Description
`generic`	Creates a secret from a file, directory, or literal value.
`docker-registry`	Creates a secret for use with a Docker registry.
`tls`	Creates a TLS secret.

Creating Secrets (imperative)

Similar usage to creation of ConfigMap

```
# Literal values
$ kubectl create secret generic db-creds ↲
  --from-literal=pwd=s3cre!

# File containing environment variables
$ kubectl create secret generic db-creds ↲
  --from-env-file=secret.env

# SSH key file
$ kubectl create secret generic db-creds ↲
  --from-file=ssh-privatekey=~/.ssh/id_rsa
```

Creating Secrets (declarative)

Value has to be base64-encoded manually

```
$ echo -n 's3cre!' | base64
czNjcmUh
```

```
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
type: Opaque
data:
  pwd: czNjcmUh
```

Secret in Pod as Volume

Value has to be base64-encoded manually

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: backend
spec:
  containers:
    - name: backend
      image: nginx
      volumeMounts:
      - name: secret-volume
        mountPath: /etc/secret
  volumes:
    - name: secret-volume
      secret:
        secretName: mysecret
```

```
$ kubectl exec -it backend -- /bin/sh
# ls /etc/secret
pwd
# cat /etc/secret/pwd
s3cre!
```

Specialized Secret Types

The table only lists some, more in docs

Type	Description
`kubernetes.io/basic-auth`	Credentials for basic authentication
`kubernetes.io/ssh-auth`	Credentials for SSH authentication
`kubernetes.io/service-account-token`	ServiceAccount token

Plain-Text Secret Values

The `stringData` attribute allows for plain-text values

```
apiVersion: v1
kind: Secret
metadata:
  name: secret-basic-auth
type: kubernetes.io/basic-auth
stringData:
  username: admin
  password: t0p-Secret
```

Detailed Coverage

The CKAD exam lays the foundation for ConfigMaps/Secrets

- Certified Kubernetes Application Developer (CKAD) Study Guide
- Exercise: Creating and using a Secret

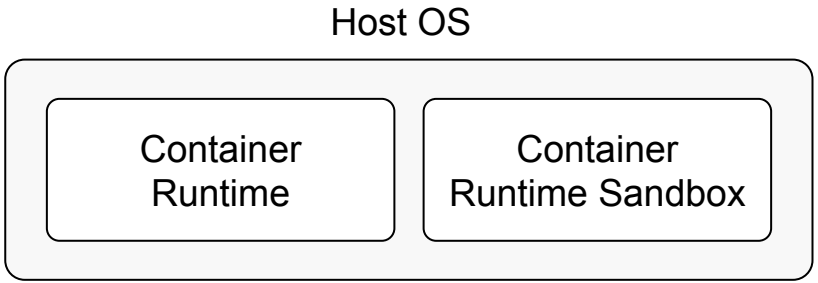
EXERCISE

Creating and
Consuming a Secret

Container Runtime Sandboxes

Provides extra layers of process isolation & security

- Does not execute in standard container runtime (for regular workload)
- The container runtime sandbox exists alongside the container runtime in isolation (for workload we are concerned about)



Container Runtime Sandboxes

Why would you want to use them?

- You don’t trust the workload as it could execute malicious operations
- Applications that don’t need host access
- Provide an extra level of security for workloads executed in a multi-tenant environment (e.g. different customers)

Implementation Options

Two solutions are relevant to the exam

gVisor/runsc: Linux application kernel by Google. Runs within host OS system with a less-lightweight solution than Virtual Machines. This blog post explains the motivation and some implementation details.

Kata containers: Containers run in lightweight Virtual Machines.

Using a gVisor Sandbox

Three-step process

- Install the container runtime sandbox implementation e.g. `runsc`
- Configure the container runtime (e.g. Docker Engine, containerd) to interact with the sandbox implementation (see docs for Docker Engine)
- Define a `RuntimeClass` object and reference its name in a Pod

Runtime Class Definition & Usage

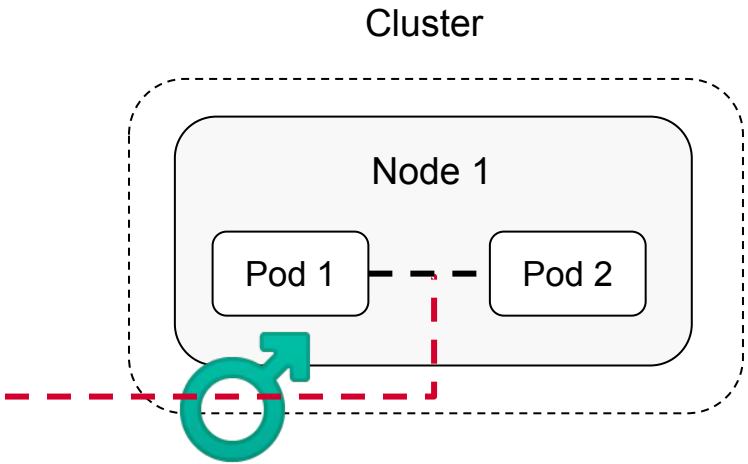
Defines the container runtime sandbox implementation

```yaml
apiVersion: node.k8s.io/v1
kind: RuntimeClass
metadata:
  name: gvisor
handler: runsc
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: alpine
spec:
  runtimeClassName: gvisor
  containers:
  - name: alpine
    image: alpine/curl:3.14
```

Scenario: Pod-to-Pod Communic.

Attacker can infiltrate unencrypted Pod to Pod messaging



Encryption with mTLS

Like HTTPS but both parties authenticate

- In the case of Pod-to-Pod communication each Pod has a certificate
- Makes it much harder to impersonate client- or server-side of communication
- Kubernetes API is able to programmatically retrieve a certificate from a Certificate Authority (CA)

Certificate Process

More information in documentation

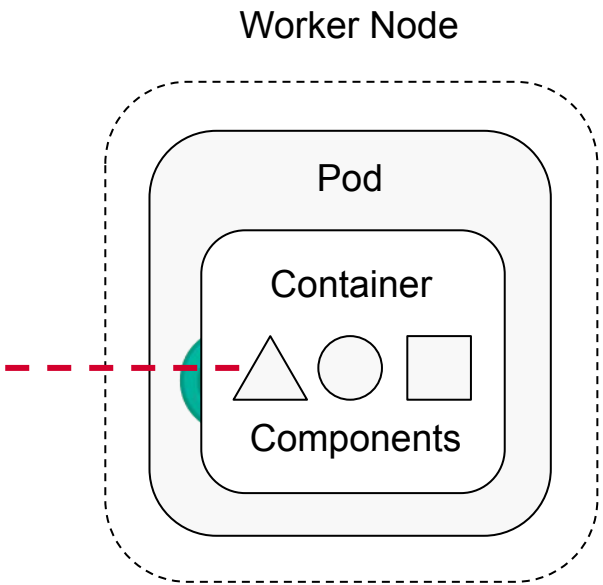
- Create a `CertificateSigningRequest` object to request certificate
- Approve or deny the certificate request via `kubectl certificate`
- The attribute `status.certificate` of the `CertificateSigningRequest` object will contain the approved certificate

Supply Chain Security

Base Image Footprint, Image Signing, Validation, and Registries, Static Analysis of Workloads, Scanning for Vulnerabilities

Scenario: Exploit Vulnerabilities

A base image can contain components with known issues



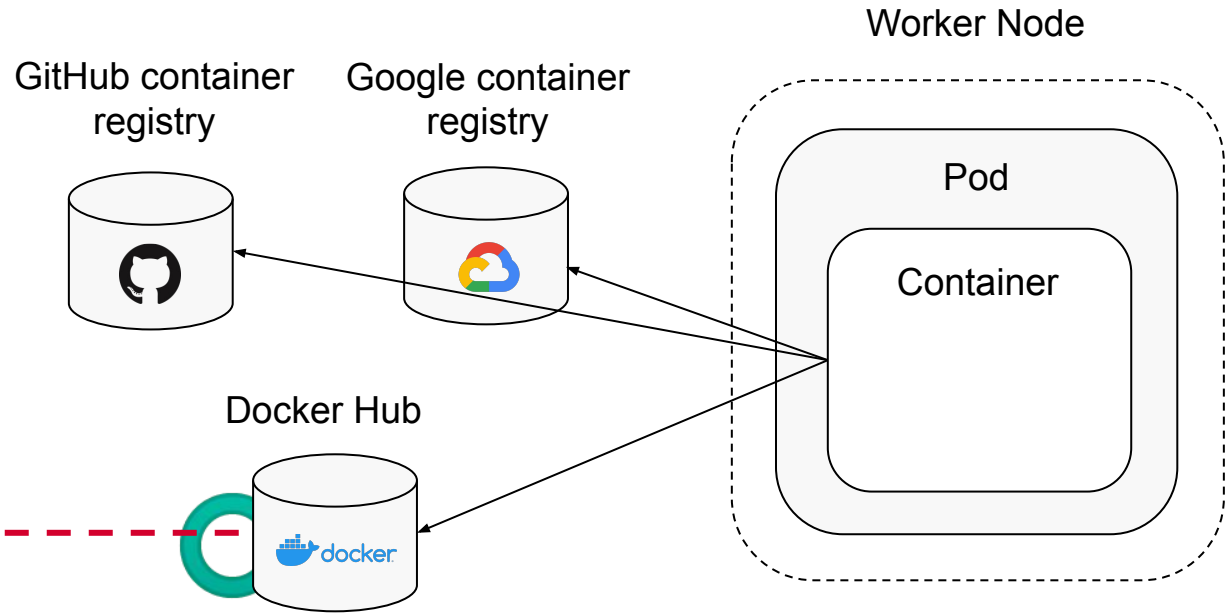
Minimizing Base Image Footprint

You can find helpful tips in this blog post

- Choose base images small in size (e.g. alpine uses a minimal OS footprint, distroless don't have a shell)
- Keep your images up to date including their dependencies + configs
- Avoid unnecessary components to be installed (which increases potential security risks) e.g. by using multi-stage builds

Scenario: Compromised Registry

Attacker can upload images with malicious instructions



Whitelisting Registries

Only trust an inhouse registry within the network

- All requests for retrieving container images should be funneled through a trusted registry e.g. Artifactory
- Images can be scanned automatically or upon request
- Enforce the use of a whitelisted registry on Kubernetes e.g. with OPA Gatekeeper policy

Validating Images

Many images have a corresponding hash in registry

- Hash (e.g. SHA256) can be used to validate the image against
- If validation fails then the image has been manipulated by an attacker
- Append `@sha256:<hash>` when declaring the image in a Pod

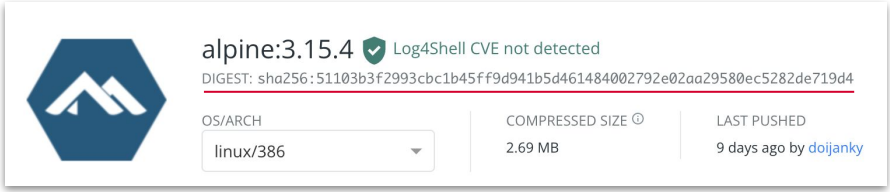


Image with SHA256

Definition with an appended hash for validation

```
apiVersion: v1
kind: Pod
metadata:
  name: alpine
spec:
  containers:
  - name: alpine
    image: alpine:3.15.4@sha256:a777c9c66ba177ccfea2...
```

Image Validation in Action

Node fails validation and doesn't download the image

```
$ kubectl get pods
NAME     READY   STATUS          RESTARTS   AGE
alpine   0/1     ErrImagePull    0          3m12s

$ kubectl describe pod alpine
Events:
  Type     Reason     Age   From               Message
  ----     ------     ----  ----               -------
  Warning  Failed     5s    kubelet            Failed to pull image
"alpine:3.15.4@sha256:a877c9c66ba177ccfea23f2a216ff6721e78a662cd17019488c417135299
cd89": rpc error: code = Unknown desc = Error response from daemon: manifest for
alpine@sha256:a877c9c66ba177ccfea23f2a216ff6721e78a662cd17019488c417135299cd89 not
found: manifest unknown:  manifest unknown
```

Creating a Container Image

Definition of instructions in a Dockerfile

- Writing a Dockerfile is “easy” but can introduce issues with sizing, and potential attack vectors
- There are a lot of best practices to adhere to that are not checked automatically

Some Best Practices

Definition of instructions in a Dockerfile

- The base image should use a concrete tag other than `latest`
- Avoid running the container with the `root` user
- Do not use sensitive data anywhere in the Dockerfile
- Use multi-stage builder to avoid having to ship with unnecessary tools

Creating a Kubernetes Object

Imperative or declarative approach

- The declarative approach requires a JSON or YAML manifest file via the `kubectl apply` or `kubectl create` command
- The YAML manifest applies to specific Kubernetes primitive and is very configurable
- Some primitives allow for granting permissions that can be used against you by an attacker

Some Best Practices

Restrict permissions, declare resource needs

- Avoid using `privileged` mode for Pods if you can avoid it
- Enforce running a container with the `root` user via a security context
- Don’t use the `latest` tags for images run in a container
- Define resource requests and limits for containers

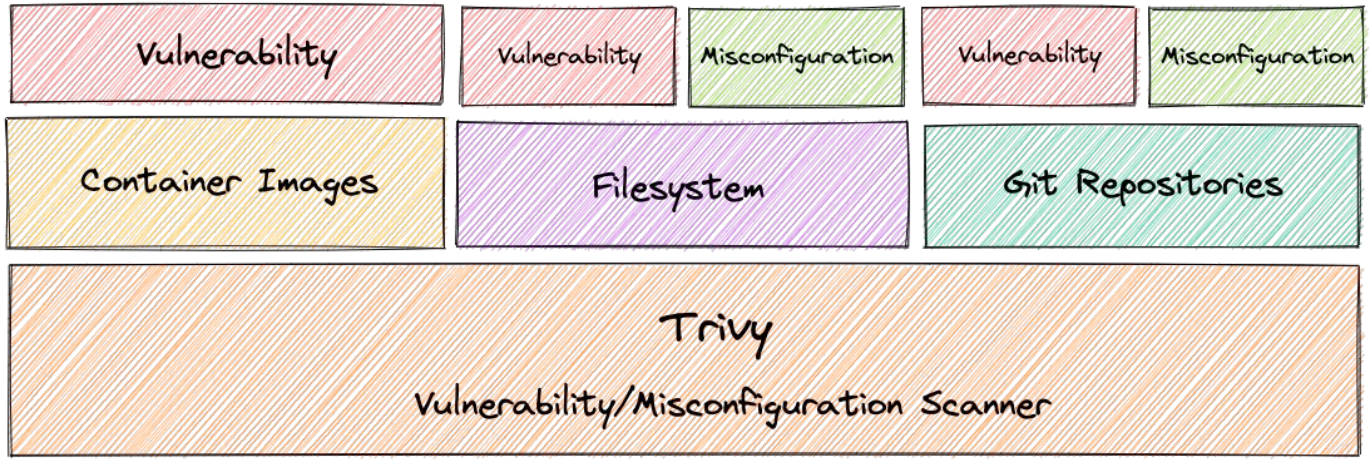
Scanning Images

Detecting vulnerabilities in an automated fashion

- Vulnerabilities are reported and stored in a central database
- The Common Vulnerability Scoring System (CVSS) assign the degree of severity
- Tools like Trivy can verify an image with the central database of vulnerabilities

Using Trivy

CLI tool for scanning images, Git repos, config files, FS



Installing & Using Trivy

Easy to install and use with different modes

- Installation: `apt-get install`, `yum install` and `brew install`
- Scanning an image: `$ trivy image nginx:1.18.0`
- Scanning a config file: `$ trivy config pod.yaml`
- Scanning a repo: `$ trivy repo https://github.com/bmuschko/cks-crash-course`

Trivy Vulnerability Report

Lists vulnerabilities per library used in an image

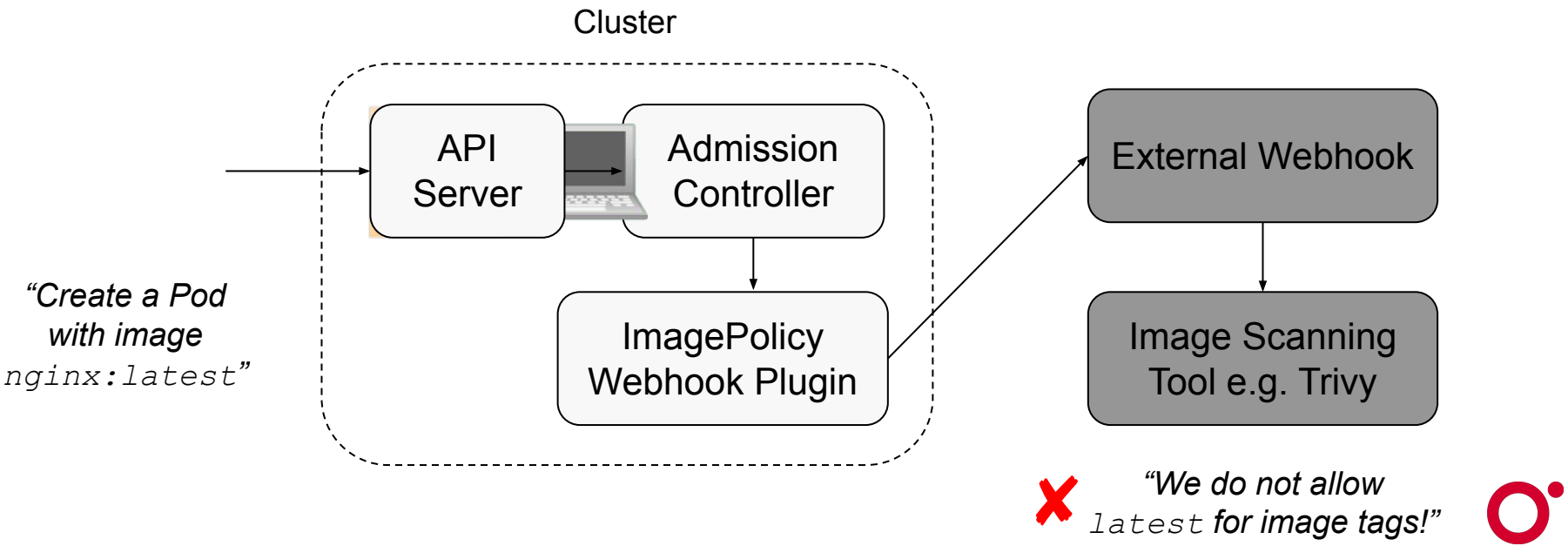
```
2022-04-20T11:24:24.060-0600	INFO	Table result includes only package filenames. Use '--format json' option to get the full path to the package file.

Node.js (node-pkg)
==================
Total: 35 (UNKNOWN: 0, LOW: 7, MEDIUM: 12, HIGH: 14, CRITICAL: 2)

+--------------------------------+--------------------+----------+-------------------+-------------------------------+-----------------------------------------------+
|            LIBRARY             |  VULNERABILITY ID  | SEVERITY | INSTALLED VERSION |         FIXED VERSION         |                     TITLE                     |
+--------------------------------+--------------------+----------+-------------------+-------------------------------+-----------------------------------------------+
| ajv (package.json)             | CVE-2020-15366     | MEDIUM   | 5.5.2             | 6.12.3                        | nodejs-ajv: prototype pollution               |
|                                |                    |          |                   |                               | via crafted JSON schema                       |
|                                |                    |          |                   |                               | in ajv.validate function                      |
|                                |                    |          |                   |                               | -->avd.aquasec.com/nvd/cve-2020-15366         |
+--------------------------------+--------------------+          +-------------------+-------------------------------+-----------------------------------------------+
```

EXERCISE

Image Scanning with
Trivy

Scenario: Intercepting API Calls

Image approval via admission controller and image policy webhook



Admission Controllers

Intercept calls via the Kubernetes API

- Decides if a request should be approved/denied or mutated before any changes take place to enforce aspects like security, governance, or configuration management
- Evaluation is based on information sent e.g. image name
- Example: Disallow containers from running with a non-root user when creation of a Pod is requested

ImagePolicyWebhook Plugin

Make admission decisions via backend

- One of the plugins for an admission controller
- Approve/deny usage of image when creating workloads e.g. scan image
- Example: Deny an image that contains known critical vulnerabilities

Admission Configuration

Deny Pod creation via ImagePolicyWebhook plugin

```
apiVersion: apiserver.config.k8s.io/v1
kind: AdmissionConfiguration
plugins:
- name: ImagePolicyWebhook
  configuration:
    imagePolicy:
      kubeConfigFile: /etc/kubernetes/admission/kubeconf
      allowTTL: 50
      denyTTL: 50
      retryBackoff: 500
      defaultAllow: false
```

Deny all Pods if backend is not available

Enabling ImagePolicyWebhook

Point to admission config file and enable plugin

```
$ vim /etc/kubernetes/manifests/kube-apiserver.yaml
spec:
  containers:
  - command:
    - kube-apiserver
    - ...
    - --admission-control-config-file=/etc/kubernetes/admission/admission-config.yaml
    - --enable-admission-plugins=NodeRestriction,ImagePolicyWebhook
```

```
$ minikube start
--extra-config=apiserver.admission-control-config-file=...
--extra-config=apiserver.enable-admission-plugins=ImagePolicyWebhook
```

The Kubeconfig File

Sets up communication with backend via TLS

```
clusters:
- name: name-of-remote-imagepolicy-service
  cluster:
    certificate-authority: /path/to/ca.pem
    server: https://images.example.com/policy

users:
- name: name-of-api-server
  user:
    client-certificate: /path/to/cert.pem
    client-key: /path/to/key.pem
```

HTTPS URL of remote service to query

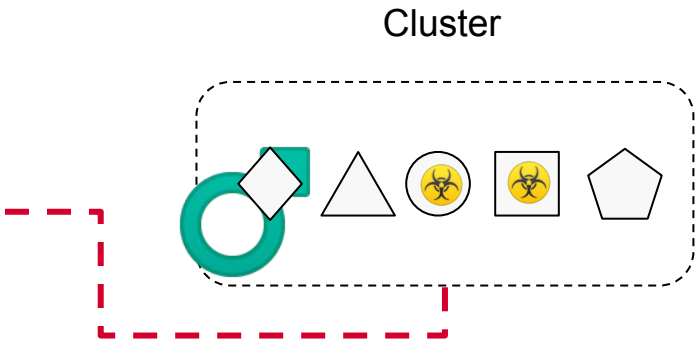
Cert and key for the webhook admission controller to use

Monitoring, Logging, and Runtime Security

Behavioral Analysis, Detecting Threats, Immutable Containers, Audit Logs

Scenario: Infiltrating Components

Detecting if and how cluster components are attacked



Behavioral Analytics

Process of observing abnormal & malicious events

- Answers questions like *“Are we currently under attack?”* or *“Was something changed after an attack?”*
- Can be performed in a manual and automated fashion
- Tools like Falco continuously detect unexpected behavior, configuration changes, intrusions, and data theft in real time

Installing & Using Falco

Only installable and executable on Linux systems

- Modes of operation
  - CLI tool
  - Background process
  - Web server

```
$ falco -r custom-rules.yaml -M 60
```

Length of execution time

Custom rules file

Falco Rules

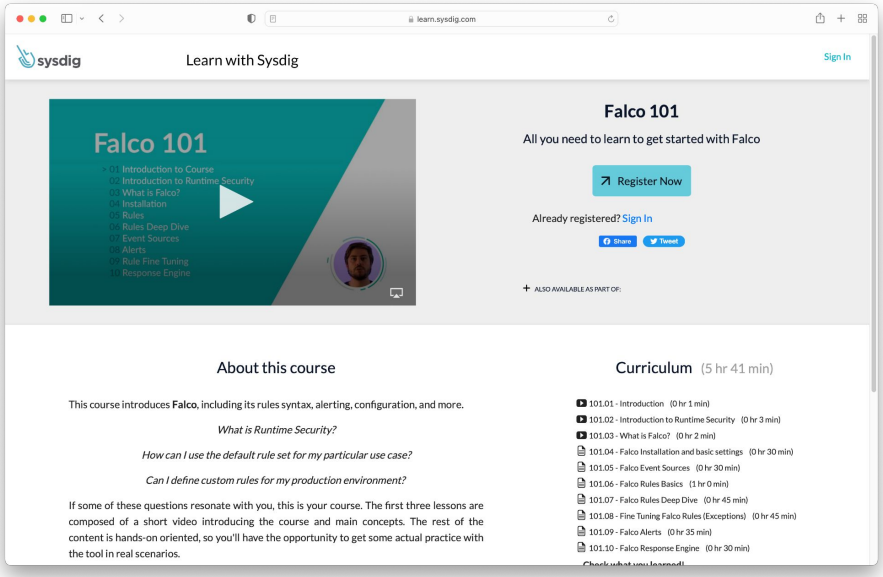
Define the conditions that trigger an alert

```
- rule: shell_in_container
  desc: notice shell activity within a container
  condition: evt.type = execve and evt.dir=< and container.id !=
             host and proc.name = bash
  output: shell in a container (user=%user.name
          container_id=%container.id
container_name=%container.name
          shell=%proc.name parent=%proc.pname
cmdline=%proc.cmdline)
  priority: WARNING
```

“Alert whenever a user opens a shell to a container”

Falco 101 Video Course

Free with online registration at [Sysdig webpage]



EXERCISE

Intrusion Detection
with Falco

Container Immutability

Cannot be changed during its lifespan

- Container's file system cannot be modified (no updates, no patches, no configuration changes)
- Updates can only be made via new container image
- Container configuration needs to be externalized e.g. environment variables, ConfigMaps, Secrets and can be used to control runtime behavior

Avoid Privileged Containers

No access to all the devices of the host machine

- Running container with root access opens door for attackers to gain full access to server
- Control host access in the Pod's security context configuration
- Use `PodSecurityPolicies` to prevent privileged access

Immutability Best Practices

No access to all the devices of the host machine

- Avoid running container with root user via `securityContext.runAsUser` with value `root` or `0`
- Avoid privileged mode via `securityContext.privileged` or `securityContext.allowPrivilegedEscalation` with value `true`
- Avoid access to host resources via `hostNetwork` with value `true`

Ensuring Container Immutability

Prevent writing to container file system

```
apiVersion: v1
kind: Pod
metadata:
  name: immutable
spec:
  securityContext:
    readOnlyRootFilesystem: true
  containers:
  - ...
```

Mount a Volume if you need to write data from the application

EXERCISE

Verifying Container Immutability

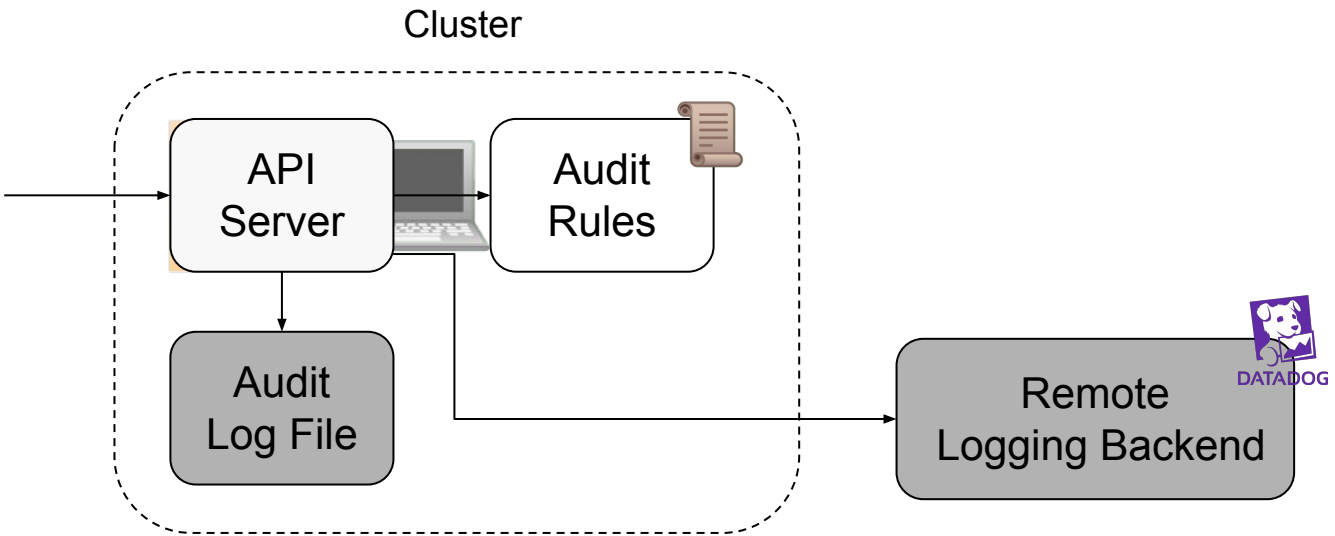
Audit Logging

Chronological records of actions in the cluster

- Records requests to Kubernetes API for internal and external components (e.g. a user creates a Pod with `kubectl`, or a change to CoreDNS)
- Structured in JSON format
- Helpful for real-time view on cluster events (e.g. threat detection) or why or what happened in the past (e.g. for debugging issues)

Audit Backends

Write to file system or external HTTP API



Audit Policy Configuration File

Rules about events and their data to be recorded

```
apiVersion: audit.k8s.io/v1
kind: Policy
rules:
- level: RequestResponse
  resources:
  - group: ""
    resources: ["pods"]
```

Audit level, in this case event metadata, request and response bodies

Kubernetes resource types to match including a potential API group

Audit Levels

Level	Runtime Behavior
`None`	Omit log events
`Metadata`	Log request metadata (requesting user, timestamp, resource, verb, etc.) but not request or response body
`Request`	Log event metadata and request body but not response body
`RequestResponse`	Log event metadata, request and response bodies

File System Audit Logging

Point to audit log config file and output log file

```
$ vim /etc/kubernetes/manifests/kube-apiserver.yaml
spec:
  containers:
  - command:
    - kube-apiserver
    - ...
    - --audit-policy-file=/etc/kubernetes/audit-policy.yaml
    - --audit-log-path=/var/log/kubernetes/audit/audit.log
```

```
$ minikube start
--extra-config=apiserver.audit-policy-file=/etc/ssl/certs/audit-policy.yaml
--extra-config=apiserver.audit-log-path=-
```

Workaround: Config dir/file needs to exist in `~/.minikube/files/`

Persisting Audit Log Records

Requires mounting the config file + produced audit log

Mount Volumes

```
volumeMounts:
  - mountPath: /etc/kubernetes/audit-policy.yaml
    name: audit
    readOnly: true
  - mountPath: /var/log/kubernetes/audit/
    name: audit-log
    readOnly: false
```

Configure host paths

```
volumes:
- name: audit
  hostPath:
    path: /etc/kubernetes/audit-policy.yaml
    type: File
- name: audit-log
  hostPath:
    path: /var/log/kubernetes/audit/
    type: DirectoryOrCreate
```

EXERCISE

Configuring and
Using Audit Logging

Summary & Wrap Up

Last words of advice...

Gaining confidence

- Run through practice exams as often as you can
- Read through online documentation start to end
- Know your tools (especially vim, bash, YAML)
- Pick time you are most comfortable, get enough sleep
- Take your first attempt easy but give it your best

O'REILLY®

Thank you