O'REILLY®

Certified Kubernetes Security Specialist (CKS) Crash Course

Curriculum September 2024

About the Trainer

Social media handles and web pages

bmuschko

bmuschko

bmuschko.com

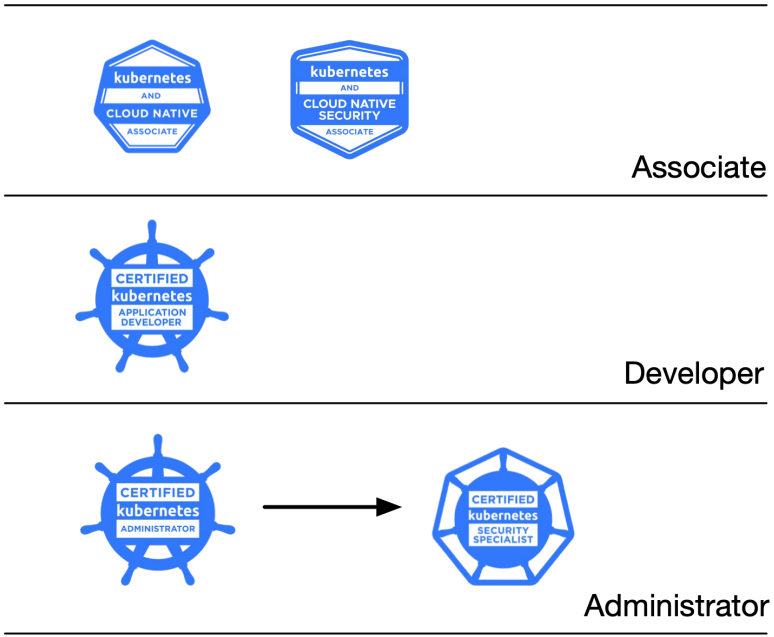
AUTOMATED ASCENT

automatedascent.com

Exam Objectives and Overview

Kubernetes Certification Path

Entry level certifications for beginners, hands-on certification for practitioners



Entry Level

Practitioner

Practitioner

Exam Objectives

"Securing container-based applications and Kubernetes platforms"

- The certification program allows users to demonstrate their competence in a hands-on, command-line environment.
- You need to be CKA-certified at the time of registration.
- You will have to use `kubectl`, cloud-native security projects, and OS-level security features.

Exam Domains

High-level topics and their weight that contribute to the total score

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%

Curriculum Details

Each domain is broken up into subtopics (find the latest outline on GitHub)

15% - 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 TLS
- Protect node metadata and endpoints
- Verify platform binaries before deploying

15% - Cluster Hardening

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

10% - System Hardening

- Minimize host OS footprint (reduce attack surface)
- Using least-privilege identity and access management
- Minimize external access to the network
- Appropriately use kernel hardening tools such as AppArmor, seccomp

20% - Minimize Microservice Vulnerabilities

- Use appropriate pod security standards
- Manage kubernetes secrets
- Understand and implement isolation techniques (multi-tenancy, sandboxed containers, etc.)
- Implement Pod-to-Pod encryption using Cilium

20% - Supply Chain Security

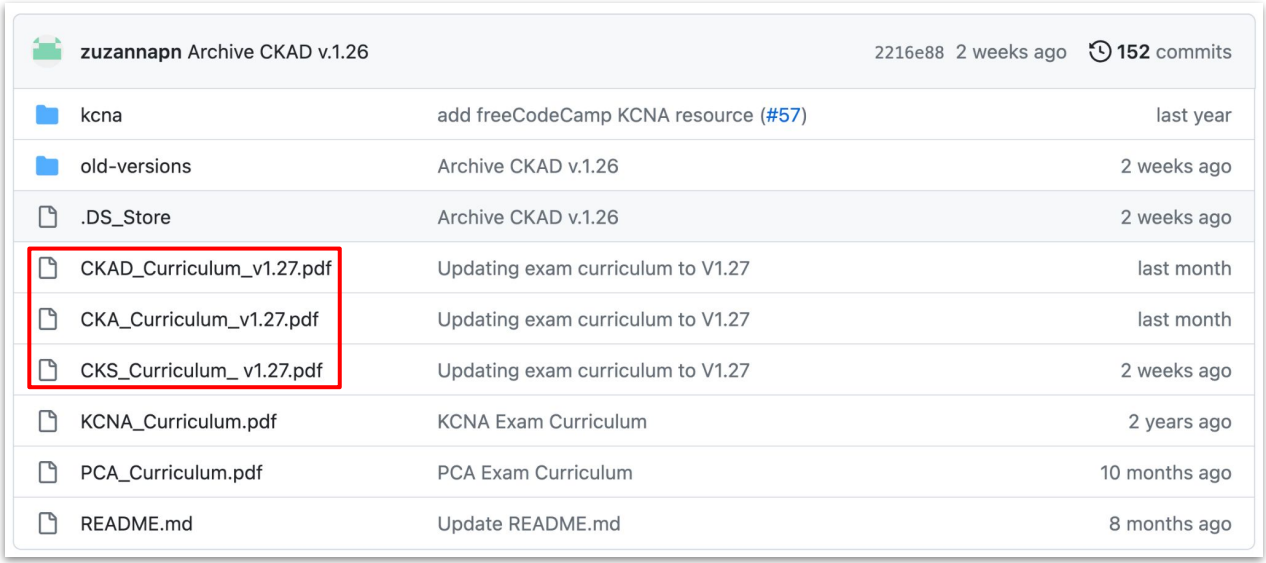
- Minimize base image footprint
- Understand your supply chain (e.g. SBOM, CI/CD, artifact repositories)
- Secure your supply chain (permitted registries, sign and validate artifacts, etc.)
- Perform static analysis of user workloads and container images (e.g. Kubesec, KubeLinter)

20% - Monitoring, Logging and Runtime Security

- Perform behavioral analytics to detect malicious activities
- Detect threats within physical infrastructure, apps, networks, data, users and workloads
- Investigate and identify phases of attack and bad actors within the environment
- Ensure immutability of containers at runtime
- Use Kubernetes audit logs to monitor access

Kubernetes Version Used in Exam

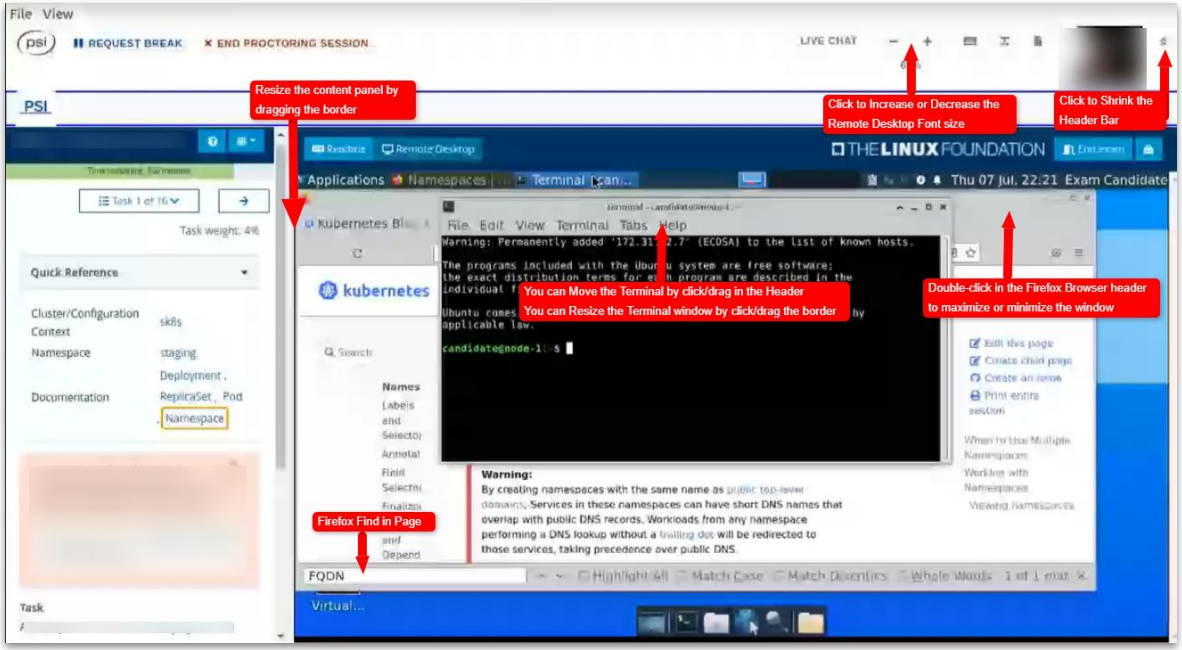
Suffix in curriculum file name indicates version, changes ~ every 3 months



zuzannapn Archive CKAD v.1.26	2216e88 2 weeks ago	**152** commits
kcna	add freeCodeCamp KCNA resource (#57)	last year
old-versions	Archive CKAD v.1.26	2 weeks ago
.DS_Store	Archive CKAD v.1.26	2 weeks ago
CKAD_Curriculum_v1.27.pdf	Updating exam curriculum to V1.27	last month
CKA_Curriculum_v1.27.pdf	Updating exam curriculum to V1.27	last month
CKS_Curriculum_ v1.27.pdf	Updating exam curriculum to V1.27	2 weeks ago
KCNA_Curriculum.pdf	KCNA Exam Curriculum	2 years ago
PCA_Curriculum.pdf	PCA Exam Curriculum	10 months ago
README.md	Update README.md	8 months ago

Exam Environment

PSI Remote desktop, single monitor, no bookmarks (announcement)



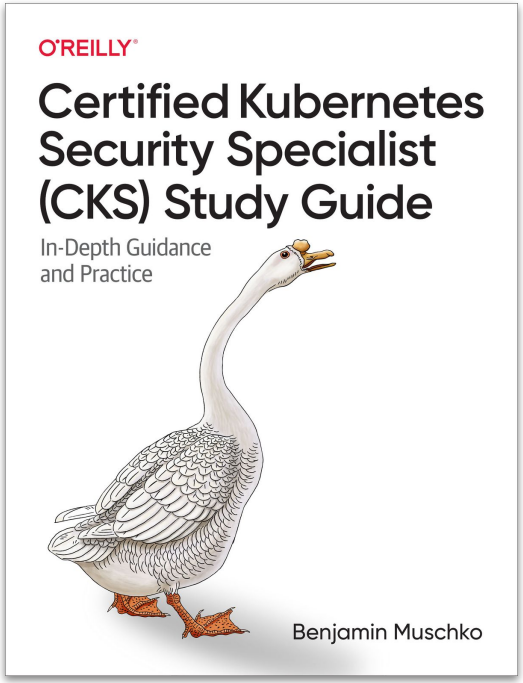
Using Kubernetes Documentation

You are allowed to open documentation pages during the exam (see FAQ)

- Docs: https://kubernetes.io/docs
- Blog: https://kubernetes.io/blog
- Trivy: https://aquasecurity.github.io/trivy
- Falco: https://falco.org/docs
- etcd: https://etcd.io/docs
- AppArmor: https://gitlab.com/apparmor/apparmor/-/wikis/Documentation

Companion Study Guide with Practice Questions

Available on Amazon and the O'Reilly learning platform



Amazon

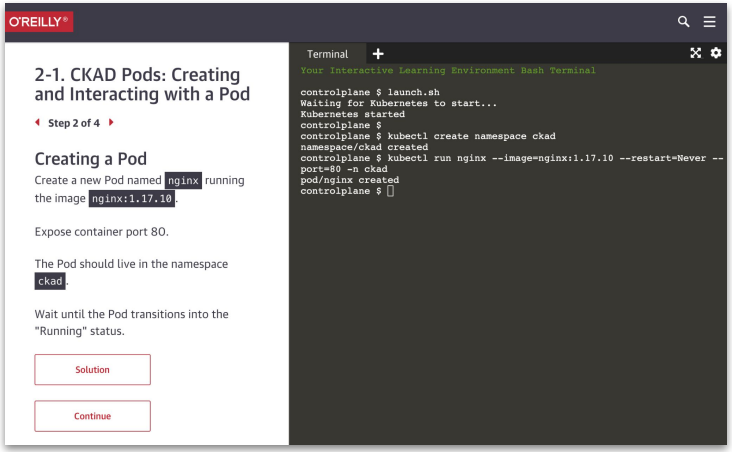
https://www.amazon.com/Certified-Kubernetes-Security-Specialist-Depth/dp/1098132971

Online access on O'Reilly learning platform

https://learning.oreilly.com/library/view/certified-kubernetes-security/9781098132965

Practice Exam on O'Reilly Learning

Interactive labs with automatic scenario and Kubernetes setup

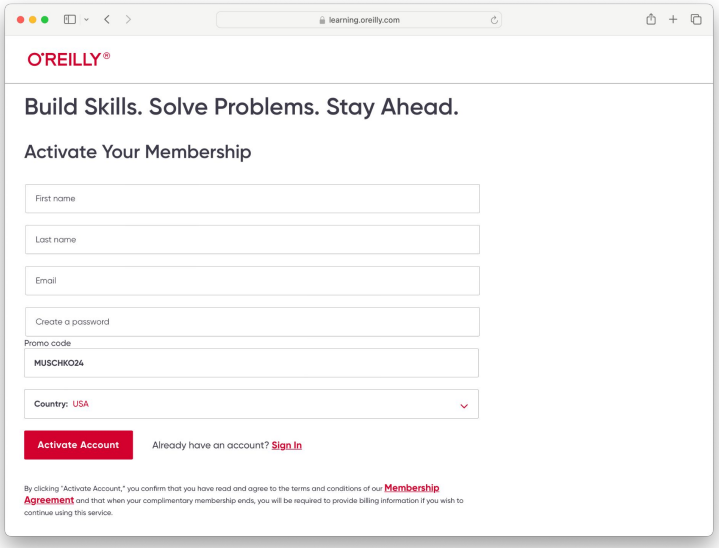


Online access on O'Reilly Learning

https://learning.oreilly.com/playlists/c94bd9b1-6277-4eb4-b442-9555ab6ad594/

30-Day Trial Access to the O'Reilly Learning Platform

Check out the content first before committing



Activate your membership with promo code **MUSCHKO2024**

https://oreillymedia.pxf.io/c/5266844/1962779/15173

Disclosure: This is an affiliate code.

Buying the Exam Voucher

There's no need to buy at full price!

Cost $395 | Online Exam

REGISTER FOR EXAM

Get a 20% with code **ASCENT20** at checkout

Disclosure: This is an affiliate code.

Preparing for the Exam

Practice is the key to cracking the exam

- Run through practice exams as often as you can
- Make yourself familiar with security-related Kubernetes documentation and where to find relevant information for external tools
- Know your tools (especially the external ones)
- Pick the time you are most comfortable with, get enough sleep
- Take it easy on your first attempt, but give it your best effort

Cluster Setup

Topics We'll Cover

Network Policies

- Defining firewall rules for Pod-to-Pod communication
- Network policy controller

CIS Benchmarks

- Best practices for configuring cluster components
- Using kube-bench
- Fixing cluster component misconfiguration

Ingress

- Recap of the Ingress primitive
- Configuring TLS termination

Verifying platform binaries

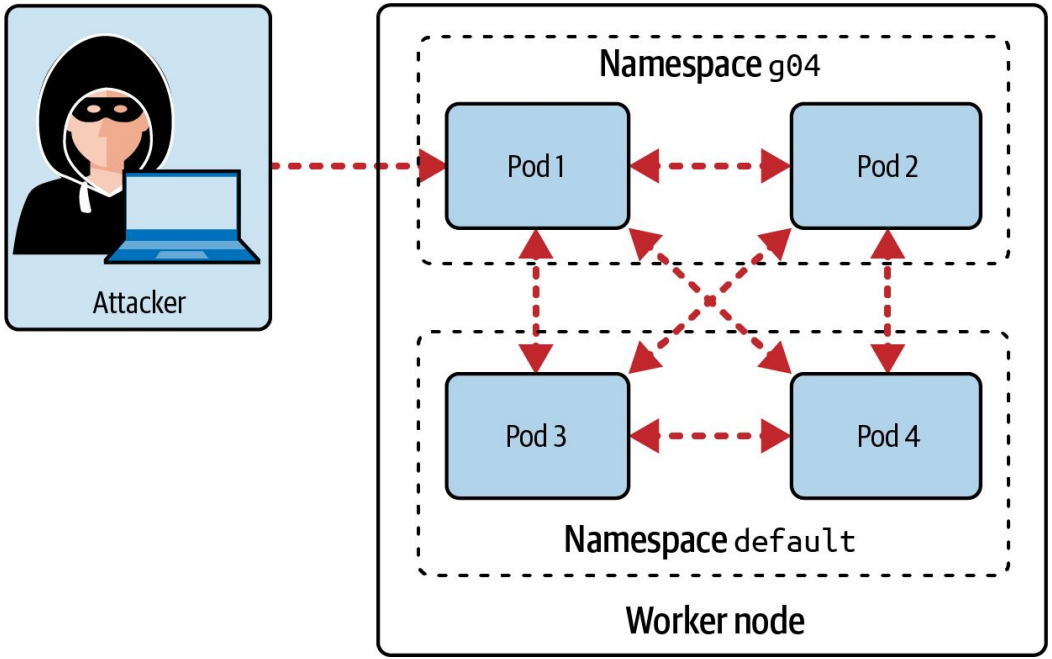
- Where do you download binaries?
- Checking contents against checksum

Network Policies

Restricting Pod-to-Pod communication

Scenario: Attacker Gains Access to a Pod

Unrestricted network access to other Pods in the cluster



What is a Network Policy?

Firewall rules for Pod-to-Pod communication

- The installed Container Network Interface (CNI) plugin leases and assigns a new virtual IP address to every Pod when it is created.
- Communication between Pods within the same cluster is unrestricted. You can use a Pod's IP address to communicate with it.
- Network policies implement the principle of least privilege. Only allow Pods to communicate if it is needed to fulfill the requirements of architecture.
- You can disallow and allow network communication with coarse-grained and fine-grained rules.

What is an Network Policy Controller?

Evaluating Network Policy rules

- A Network Policy cannot work without a Network Policy controller. The Network Policy controller evaluates the collection of rules defined by a Network Policy.

- One option for such a Network Policy controller is Cilium.

Network Policy YAML Manifest

Key-value pairs can be parsed from different sources

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: api-allow
spec:
  podSelector:
    matchLabels:
      app: payment-processor
      role: api
  ingress:
  - from:
    - podSelector:
        matchLabels:
          app: coffeeshop
```

Selects the Pod the policy should apply to by label selection

Allows incoming traffic from the Pod with matching labels within the same namespace

Network Policy Rules

Attributes that form the rules

Attribute	Description
`podSelector`	Selects the Pods in the namespace to apply the network policy to.
`policyTypes`	Defines the type of traffic (i.e., ingress and/or egress) the network policy applies to.
`ingress`	Lists the rules for incoming traffic. Each rule can define `from` and `ports` sections.
`egress`	Lists the rules for outgoing traffic. Each rule can define `to` and `ports` sections.

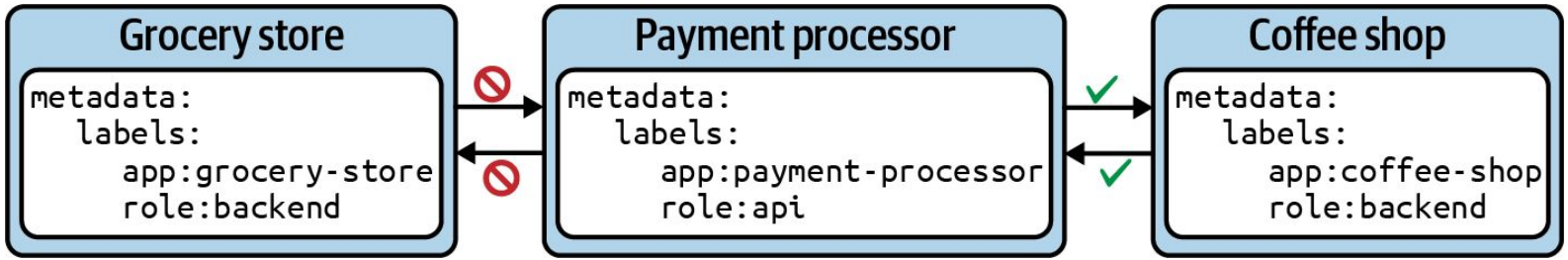
Behavior of `to` and `from` Selectors

Ingress and egress sections define the same attributes

Attribute	Description
`podSelector`	Selects Pods by label(s) in the same namespace as the Network Policy which should be allowed as ingress sources or egress destinations.
`namespaceSelector`	Selects namespaces by label(s) for which all Pods should be allowed as ingress sources or egress destinations.
`namespaceSelector` and `podSelector`	Selects Pods by label(s) within namespaces by label(s).

Example Network Policy

Restricting access to the payment processor Pod from other Pods



Listing Network Policies

Doesn’t provide much details apart from Pod selector labels

```
$ kubectl get networkpolicy
NAME          POD-SELECTOR                       AGE
api-allow     app=payment-processor,role=api     83m
```

Rendering Network Policy Details

Ingress and egress rules can be inspected in details

```
$ kubectl describe networkpolicy api-allow
Name:         api-allow
Namespace:    default
Created on:   2020-09-26 18:02:57 -0600 MDT
Labels:       <none>
Annotations:  <none>
Spec:
  PodSelector:     app=payment-processor,role=api
  Allowing ingress traffic:
    To Port: <any> (traffic allowed to all ports)
    From:
      PodSelector: app=coffeeshop
  Not affecting egress traffic
  Policy Types: Ingress
```

Verifying the Runtime Behavior

Communication from Pods not matching with labels will be blocked

```
$ kubectl run grocery-store --rm -it --image=busybox ↲
  -l app=grocery-store,role=backend -- /bin/sh
/ # wget --spider --timeout=1 10.0.0.51
Connecting to 10.0.0.51 (10.0.0.51:80)
wget: download timed out
/ # exit
pod "grocery-store" deleted

$ kubectl run coffeeshop --rm -it --image=busybox ↲
  -l app=coffeeshop,role=backend -- /bin/sh
/ # wget --spider --timeout=1 10.0.0.51
Connecting to 10.0.0.51 (10.0.0.51:80)
remote file exists
/ # exit
pod "coffeeshop" deleted
```

Default Deny Network Policies

Network policies are additive

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

Selects all Pods in the namespace

Applies in incoming and outgoing traffic

Restricting Access to Ports

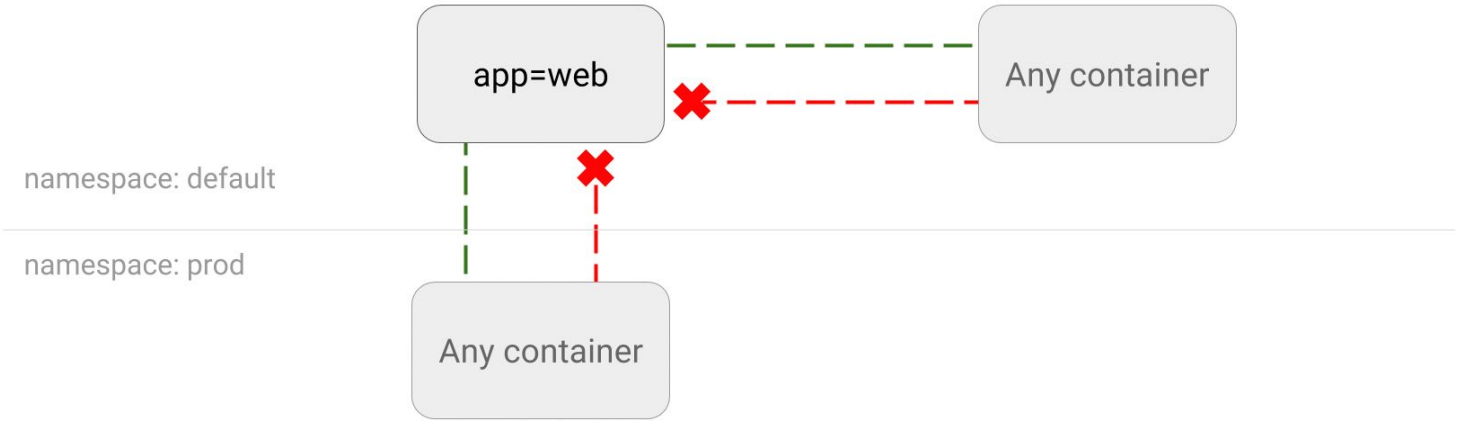
By default, all ports are accessible

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: port-allow
spec:
  podSelector:
    matchLabels:
      app: backend
  ingress:
  - from:
    - podSelector:
        matchLabels:
          app: frontend
    ports:
    - protocol: TCP
      port: 8080
```

Only allow incoming traffic to port 8080

Scenario-Based Learning Resource

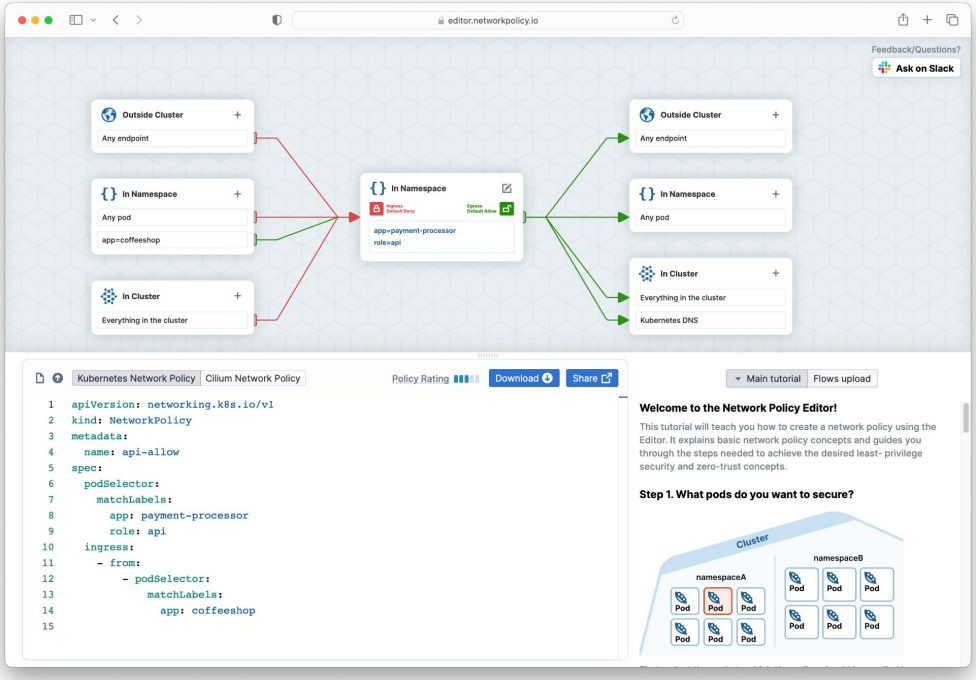
Exercising use cases for Network Policies is helpful with understanding concept



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

Visualizing and Analysing a Network Policy

The page networkpolicy.io provides policy editor



Exercise

Restricting access to and from a Pod with Network Policies

Exam Essentials

What to focus on for the exam

- By default, any Pod can talk any other Pod through its virtual IP address. Any container port is accessible. A Network Policy defines restrictions for Pod-to-Pod communication.
- It’s best practice to create a default deny policy first to allow as little access as possible to harden security.
- Given that Network Policies are additive, you can open access with additional definitions.
- Access can be defined for egress and ingress traffic, as well as ports. Network Policies work with label selection for namespaces and Pods.

CIS Benchmarks

Best practices for cluster component configuration

What is the CIS Benchmark?

Guidelines and benchmark tests for best practices in securing code

- Installing the cluster using `kubeadm` or provisioning it with a service provider doesn't necessarily apply security best practices from the get-go.
- Hardening the security measures of a cluster is a crucial activity for any Kubernetes administrator seeking to minimize attack vectors.
- The Center of Internet Security (CIS) is a not-for-profit organization that publishes cybersecurity best practices.
- The CIS Kubernetes Benchmark is a catalog of best practices for Kubernetes environments.

Using kube-bench

Checks against CIS Benchmark in an automated fashion

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

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

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

Rendering the Verification Result

The Pods logs show the results in textual form

```
[INFO] 1 Master Node Security Configuration
[INFO] 1.1 API Server
[FAIL] 1.1.1 Ensure that the --allow-privileged argument is set to false (Scored)
[FAIL] 1.1.2 Ensure that the --anonymous-auth argument is set to false (Scored)
[PASS] 1.1.3 Ensure that the --basic-auth-file argument is not set (Scored)
[PASS] 1.1.4 Ensure that the --insecure-allow-any-token argument is not set (Scored)
[FAIL] 1.1.5 Ensure that the --kubelet-https argument is set to true (Scored)
[PASS] 1.1.6 Ensure that the --insecure-bind-address argument is not set (Scored)
[PASS] 1.1.7 Ensure that the --insecure-port argument is set to 0 (Scored)
[PASS] 1.1.8 Ensure that the --secure-port argument is not set to 0 (Scored)
[FAIL] 1.1.9 Ensure that the --profiling argument is set to false (Scored)
[FAIL] 1.1.10 Ensure that the --repair-malformed-updates argument is set to false (Scored)
[PASS] 1.1.11 Ensure that the admission control policy is not set to AlwaysAdmit (Scored)
[FAIL] 1.1.12 Ensure that the admission control policy is set to AlwaysPullImages (Scored)
[FAIL] 1.1.13 Ensure that the admission control policy is set to DenyEscalatingExec (Scored)
[FAIL] 1.1.14 Ensure that the admission control policy is set to SecurityContextDeny (Scored)
[PASS] 1.1.15 Ensure that the admission control policy is set to NamespaceLifecycle (Scored)
[FAIL] 1.1.16 Ensure that the --audit-log-path argument is set as appropriate (Scored)
[FAIL] 1.1.17 Ensure that the --audit-log-maxage argument is set to 30 or as appropriate (Scored)
[FAIL] 1.1.18 Ensure that the --audit-log-maxbackup argument is set to 10 or as appropriate (Scored)
[FAIL] 1.1.19 Ensure that the --audit-log-maxsize argument is set to 100 or as appropriate (Scored)
[PASS] 1.1.20 Ensure that the --authorization-mode argument is not set to AlwaysAllow (Scored)
[PASS] 1.1.21 Ensure that the --token-auth-file parameter is not set (Scored)
[FAIL] 1.1.22 Ensure that the --kubelet-certificate-authority argument is set as appropriate (Scored)
```

Fixing Detected Security Issues

Edit the corresponding configuration file

- Key elements: the type of the inspected node, the inspected components, a list of passed checks, a list of failed checks, a list of warnings, and a high-level summary.
- Identify the cluster component configuration file and apply the recommended changes.
- The corresponding Pod of the cluster component in the `kube-system` namespace will be restarted automatically. The startup process can take a couple of seconds.

Common Configuration Files

Most cluster components run in a Pod

Cluster Component	Configuration File Location
API server	`/etc/kubernetes/manifests/kube-apiserver.yaml`
Etcd	`/etc/kubernetes/manifests/etcd.yaml`
Kubelet	`/var/lib/kubelet/config.yaml`

Exam Essentials

What to focus on for the exam

- There’s no need to memorize CIS benchmark rules and best practices. They are properly documented.
- Know how to run kube-bench against on-prem and cloud provider clusters.
- Be familiar with the interpreting the verification results provided by kube-bench.
- Practice editing the configuration files for different cluster components.

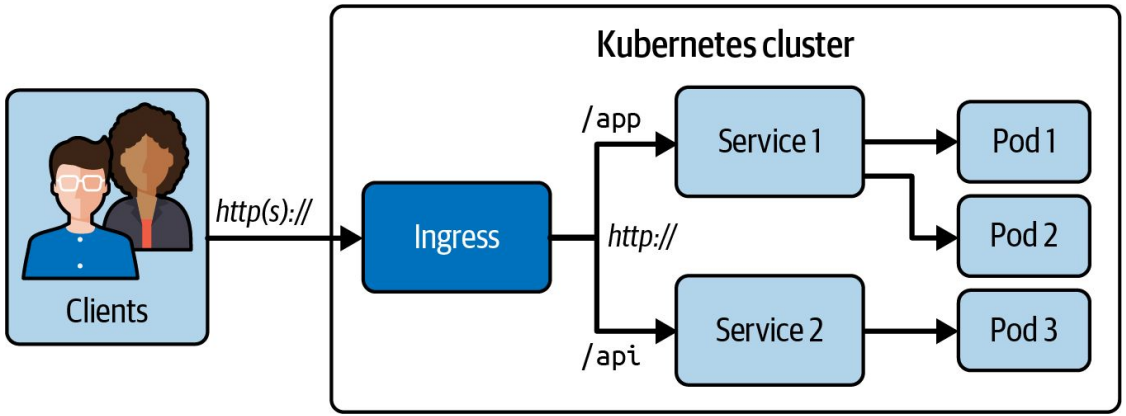
Ingress with TLS

Configuring encrypted communication via SSL

Configuring TLS termination for an Ingress

Ingress primitive has already been covered by the CKA exam

An Ingress routes HTTP and/or HTTPS traffic from outside of the cluster to one or many Services based on a matching URL context path.



Creating the TLS Certificate and Key

Need to configure a Secret that represents TLS certificate

- Generate a TLS certificate and key via OpenSSL.
- The exam will likely provide the corresponding files or the command.

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

Creating the TLS-Typed Secret Imperatively

Provide TLS certificate file and key file

- The easiest way to create a Secret is with the help of an imperative command.
- Does not require base64-encoding the file contents.

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

Creating the TLS-Typed Secret Declaratively

Requires more manual steps

Make sure to assign the values for the attributes `tls.crt` and `tls.key` as single-line, base64-encoded values.

```
apiVersion: v1
kind: Secret
metadata:
  name: tls-ingress-secret
type: kubernetes.io/tls
data:
  tls.crt: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk...
  tls.key: LS0tLS1CRUdJTiBQUklWQVRFIEtFWS0tLS0tCk...
```

Creating the Ingress Imperatively

The full command is hard to remember

The relevant portion of the command is the tls parameter in the `--rule` CLI flag.

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

Creating the Ingress Declaratively

Easy to copy and modify from the documentation

The attribute for defining the TLS information is `spec.tls[]`.

```
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
    ...
```

Exercise

Creating an Ingress with TLS termination

Exam Essentials

What to focus on for the exam

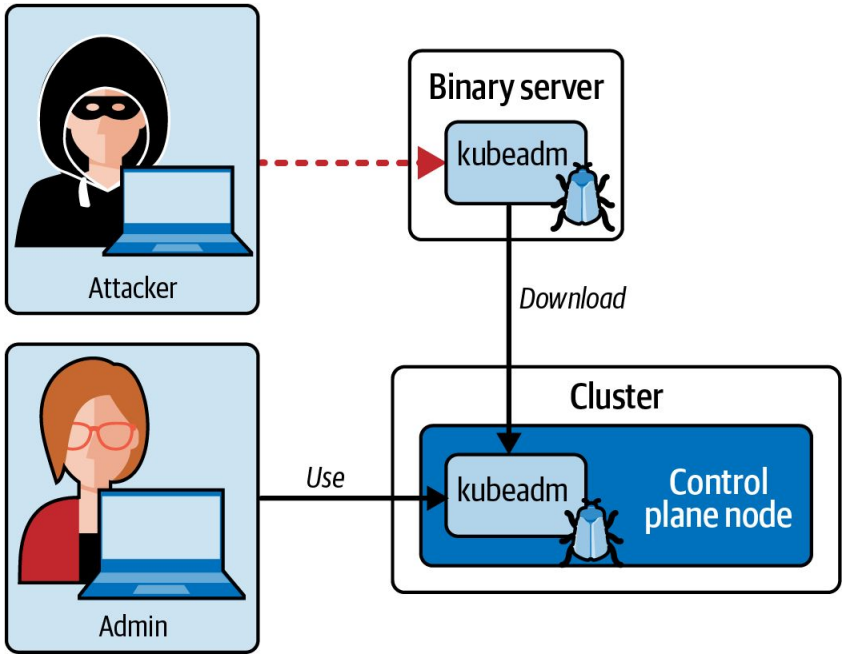
- Practice the creation of a TLS Secret. Pick the creation method that works best for you.
- Know how to assign the TLS Secret to the Ingress by name.
- You will not need to make a call the Ingress endpoint during the exam. I'd recommend trying it out to become more familiar with the concept.

Verifying Kubernetes Platform Binaries

Using checksums to validate the contents

Scenario: Attacker Injected Malicious Code

Gains control to cluster by modifying the byte code



What are Kubernetes Platform Binaries?

Needed to interact with cluster or to run the cluster

- The Kubernetes project publishes client and server binaries with every release. The client binary refers to the executable `kubectl`. Server binaries include `kubeadm`, as well as the executable for the API server, the scheduler, and the kubelet.

- The executables `kubectl` and `kubeadm` are essential for interacting with Kubernetes.

- You can verify the validity of a binary with the help of a hash code like MD5 or SHA. Kubernetes publishes SHA256 hash codes for each binary.

Downloading the Checksum Files

Available alongside the platform binaries

Binary	Download URL
`kubectl`	https://dl.k8s.io/v1.30.1/bin/linux/amd64/kubectl.sha256
`kubeadm`	https://dl.k8s.io/v1.30.1/bin/linux/amd64/kubeadm.sha256
`kubelet`	https://dl.k8s.io/v1.30.1/bin/linux/amd64/kubelet.sha256
`kube-apiserver`	https://dl.k8s.io/v1.30.1/bin/linux/amd64/kube-apiserver.sha256

Verifying the Binary with a Checksum

Needed to 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

# Windows using Powershell
$ $($(CertUtil -hashfile .\kubectl.exe SHA256)[1] -replace " ", "")↲
  -eq $(type .\kubectl.exe.sha256)
```

Exercise

Verifying existing Kubernetes binaries

Exam Essentials

What to focus on for the exam

- Know where to download the checksum files. The URL is available in the Kubernetes documentation.

- The exam environment will be Linux-based. It’s fine to simply practice the checksum validation command for Linux only.

Cluster Hardening

Topics We'll Cover

Interacting with the Kubernetes API

- API request processing phases
- Restricting Access to the API Server

Role-Based Access Control (RBAC)

- Understanding the Role and RoleBinding primitives
- Using a service account in the context of RBAC

Service Accounts

- Creation and usage of service accounts
- Changing security-specific configuration for service accounts

Upgrading Kubernetes

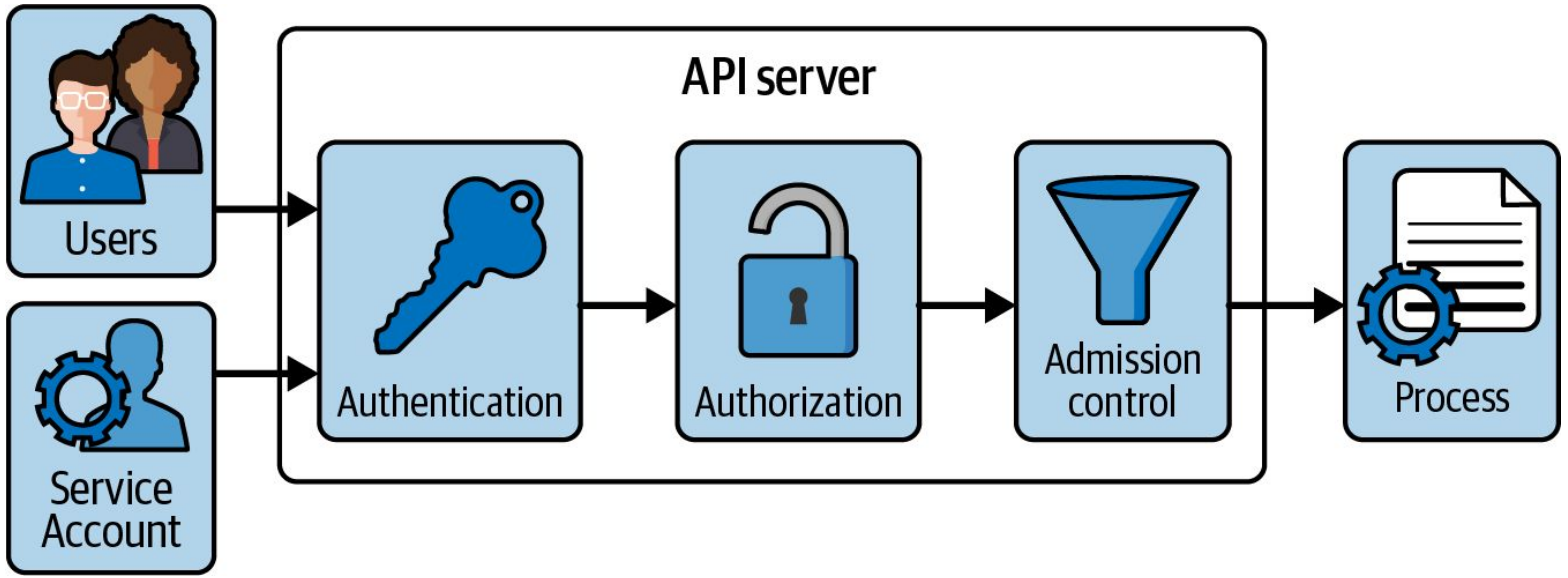
- Kubernetes release cadence
- Why is an periodic upgrade necessary?
- Upgrade process

Authentication and Authorization

Access control for the Kubernetes API

Processing a Request to the API Server

Any client request to the API server needs to pass four phases



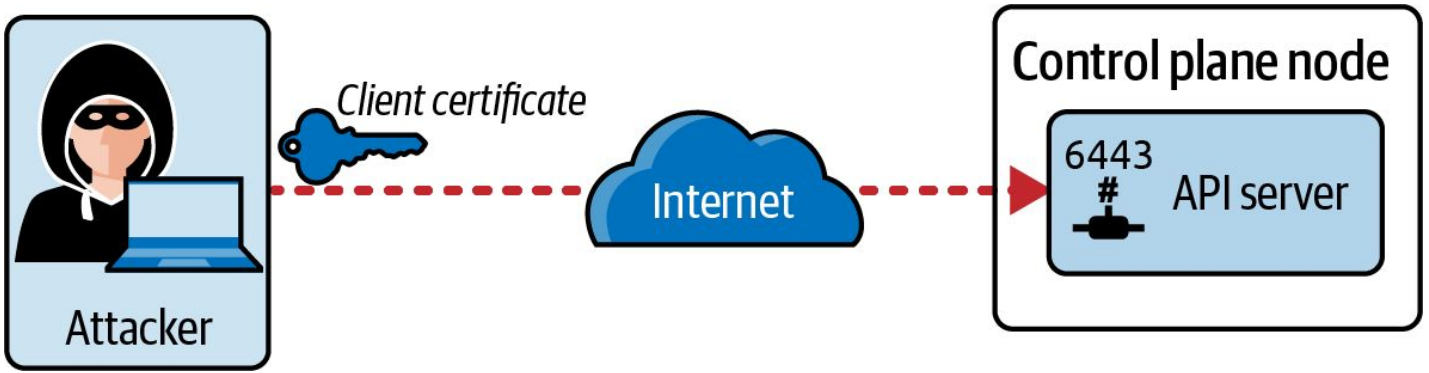
API Request Processing Phases

Every request has to pass those phases in the following order

- *Authentication:* Validates the identity of the caller using a supported authentication strategy, e.g. client certificates or bearer tokens.
- *Authorization:* Determines if the identity provided in the first stage can access the verb and HTTP path request. This is usually implemented with RBAC.
- *Admission Controller:* Verifies if the request is well-formed and potentially needs to be modified before the request is processed. Ensures that the resource included in the request is valid (could also be implemented as part of admission control).

Scenario: Attacker calls API Server from the Internet

Restrict admin permissions and instantiate firewall rules



Authentication via Credentials in Kubeconfig

The kubeconfig file at `$HOME/.kube/config` is evaluated by `kubectl`

```
apiVersion: v1
kind: Config
clusters:
- cluster:
    certificate-authority: /Users/bmuschko/.minikube/ca.crt
    extensions:
    - extension:
        last-update: Thu, 04 May 2023 08:48:09 MDT
        provider: minikube.sigs.k8s.io
        version: v1.30.1
      name: cluster_info
    server: https://127.0.0.1:58936
  name: minikube
contexts:
- context:
    cluster: minikube
    user: bmuschko
  name: bmuschko
current-context: bmuschko
users:
- name: bmuschko
  user:
    client-certificate-data: REDACTED
    client-key-data: REDACTED
```

API server endpoint for connecting with cluster

Groups access parameters under a convenient name

Client certificate for user

Managing Kubeconfig and Current Context

Interaction with `kubeconfig` configuration

```
$ kubectl config view
apiVersion: v1
kind: Config
current-context: bmuschko
...

$ kubectl config current-context
bmuschko

$ kubectl config use-context johndoe
Switched to context "johndoe".

$ kubectl config current-context
johndoe
```

Renders the contents of the kubeconfig file

Shows the currently-selected context

Switches to the context defined in the kubeconfig

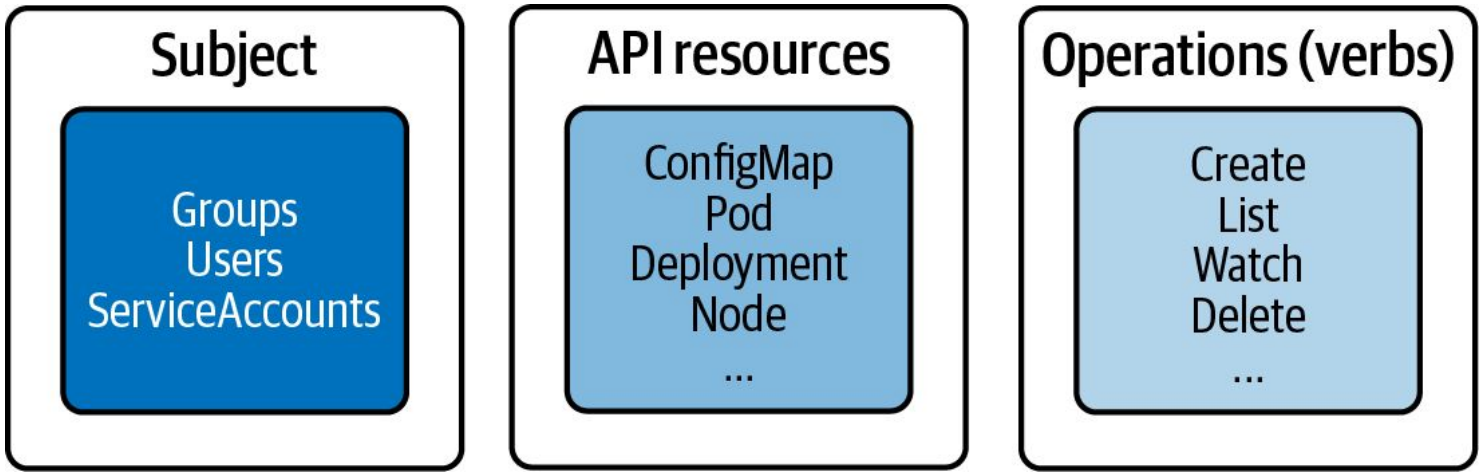
What is RBAC?

Restricting access control for clients interacting with API server

- Defines policies for users, groups, and processes by allowing or disallowing access to manage API resources.
- Enabling and configuring RBAC is mandatory for any organization with a strong emphasis on security.
- *Example:* “The human user John is only allowed to list and create Pods and Deployments, but nothing else.”

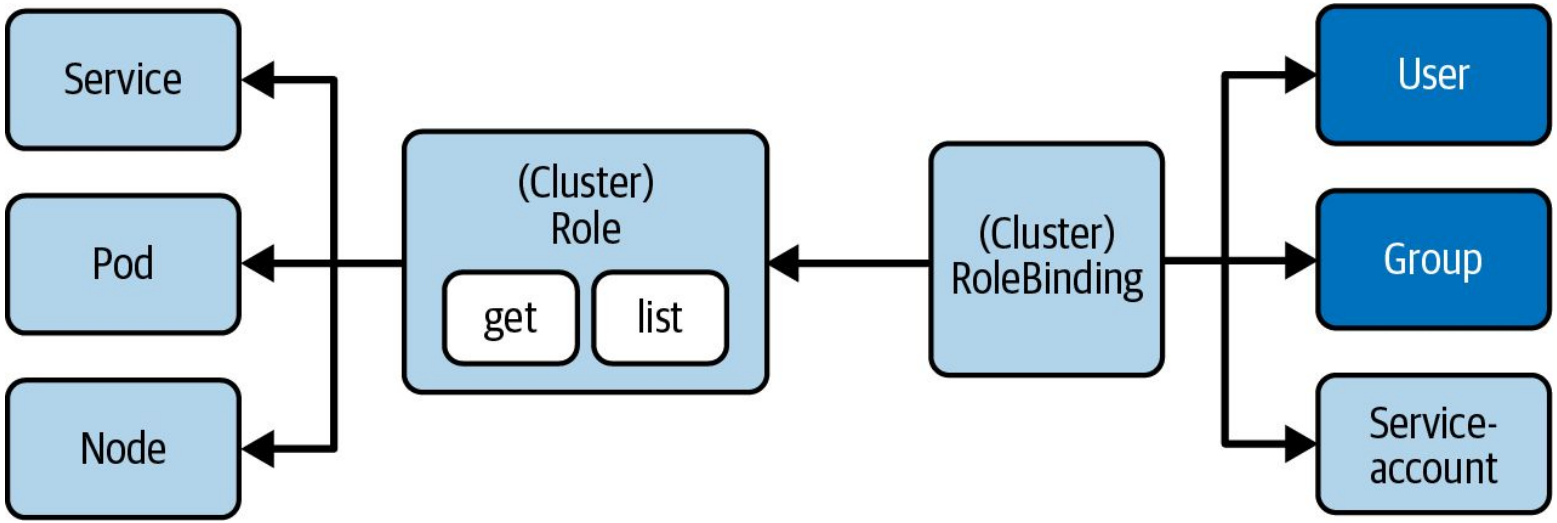
RBAC High-Level Overview

Three key elements for understanding concept



Involved RBAC Primitives

Restrict access to certain operations of API resources for subjects



Namespace and Cluster-Wide Permissions

Defining permission scopes

- The *Role* maps API resources to verbs for a single namespace.
- The *RoleBinding* maps a reference of a Role to one or many subjects for a single namespace.
- To apply permissions across all namespaces of a cluster or cluster-wide resources, use the corresponding API primitives *Cluster* and *ClusterRoleBinding*. The configuration options of the manifests are the same.

Creating a Role with Imperative Approach

Defines verbs and resources in comma-separated list

```
$ kubectl create role read-only --verb=list,get,watch ↲
  --resource=pods,deployments,services
role.rbac.authorization.k8s.io/read-only created
```

Resources: Primitives the operations should apply to

Operations: Only allow listing, getting, watching

Role YAML Manifest

Can define a list of rules in an array

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: read-only
rules:
- apiGroups: [""]
  resources: ["pods", "services"]
  verbs: ["list", "get", "watch"]
- apiGroups: ["apps"]
  resources: ["deployments"]
  verbs: ["list", "get", "watch"]
```

Resources with groups need to be spelled out explicitly (in this case `apps/Deployment`)

Getting Role Details

Maps objects of a Kubernetes primitive to verbs

```
$ kubectl describe role read-only
Name:         read-only
Labels:       <none>
Annotations:  <none>
PolicyRule:
  Resources         Non-Resource URLs  Resource Names  Verbs
  ---------         -----------------  --------------  -----
  pods              []                 []              [list get watch]
  services          []                 []              [list get watch]
  deployments.apps  []                 []              [list get watch]
```

Creating a RoleBinding with Imperative Approach

Maps subject to Role

```
$ kubectl create rolebinding read-only-binding --role=read-only ↲
  --user=johndoe
rolebinding.rbac.authorization.k8s.io/read-only-binding created
```

Subject: Binds a user to the Role

Role: Reference the name of the object

RoleBinding YAML Manifest

Roles can be mapped to multiple subjects if needed

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: read-only-binding
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: read-only
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: User
  name: johndoe
```

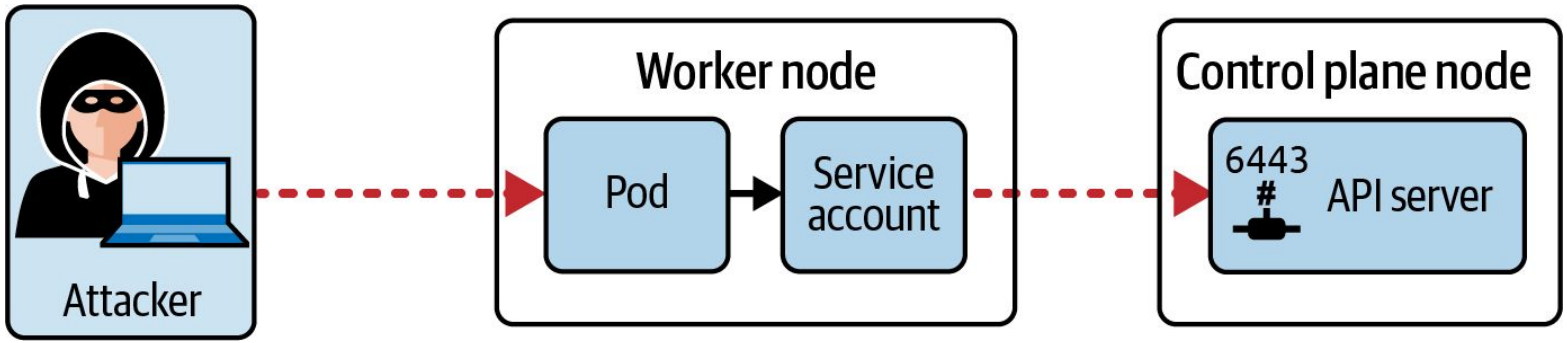
Getting RoleBinding Details

Only shows mapping between Role and subjects

```
$ kubectl describe rolebinding read-only-binding
Name:         read-only-binding
Labels:       <none>
Annotations:  <none>
Role:
  Kind:  Role
  Name:  read-only
Subjects:
  Kind  Name     Namespace
  ----  ----     ---------
  User  johndoe
```

Scenario: Attacker uses ServiceAccount

Gains access to Pod and misuses ServiceAccount with too much permissions



What is a Service Account?

Non-human request to Kubernetes API from a process

- Processes running outside of Kubernetes or processes running inside of a container may need to interact with the Kubernetes API. A Service Account allows for authenticating with the API server through an authentication token.
- Authorization is controlled through RBAC and assigned to the Service Account.
- If not assigned explicitly, a Pod uses the `default` Service Account. The `default` Service Account has the same permissions as an unauthenticated user.
- *Example:* “I have a CI/CD process that retrieves Pod information by making calls to the Kubernetes API.”

Service Account as Subject

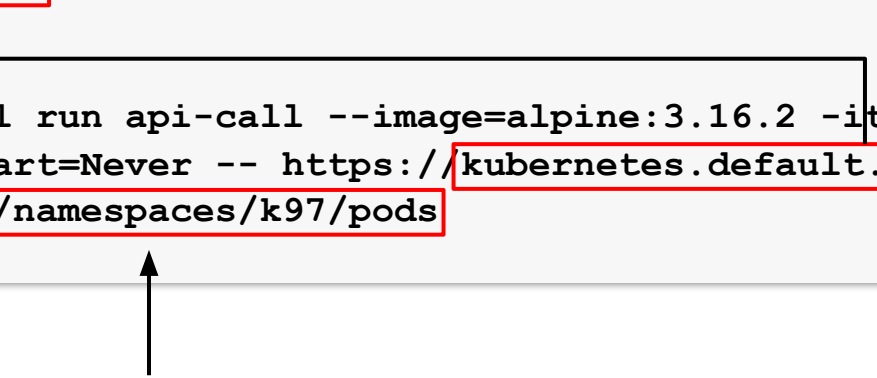
Pod assign the Service Account by name

Using the kubernetes Service

Convenient way to get API server endpoint from within the cluster

```
$ kubectl get service kubernetes
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   13h

$ kubectl run api-call --image=alpine:3.16.2 -it --rm ↲
  --restart=Never -- https://kubernetes.default.svc.cluster.local/↲
  api/v1/namespaces/k97/pods
```

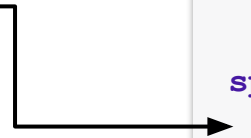


API endpoint for listing all Pods in namespace `k97`

Service Account YAML Manifest

In a Pod manifest, refer to the Service Account by name

```yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: sa-api
  namespace: k97
```



```yaml
apiVersion: v1
kind: Pod
metadata:
  name: list-pods
  namespace: k97
spec:
  serviceAccountName: sa-api
  ...
```

Accessing Service Account Authentication Token

Automounted at specific Volume mount path

```
$ kubectl describe pod list-pods -n k97
...
Containers:
  pods:
     ...
     Mounts:
       /var/run/secrets/kubernetes.io/serviceaccount from ↵
       kube-api-access-xnkwd (ro)
...

$ kubectl exec -it list-pods -n k97 -- /bin/sh
# cat /var/run/secrets/kubernetes.io/serviceaccount/token
eyJhbGciOiJSUzI1N...
# exit
```

Using Service Account in Pod

Enables authentication token auto-mounting by default

```
apiVersion: v1
kind: Pod
metadata:
  name: list-pods
  namespace: k97
spec:
  serviceAccountName: sa-api
  containers:
  - name: pods
    image: alpine/curl:3.14
    command: ['sh', '-c', 'while true; do curl -s -k -m 5 -H↵
              "Authorization: Bearer $(cat /var/run/secrets/kubernetes.io/↵
              serviceaccount/token)" https://kubernetes.default.svc.cluster.↵
              local/api/v1/namespaces/k97/pods; sleep 10; done']
```

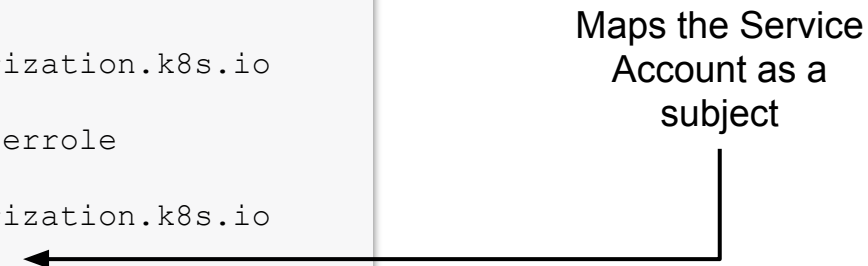
Assigned name of Service Account

List all Pods in the namespace `k97` via an API call

RoleBinding YAML Manifest

Default RBAC policies don't span beyond `kube-system`

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: serviceaccount-pod-rolebinding
  namespace: k97
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: list-pods-clusterrole
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: ServiceAccount
  name: sa-api
```



Exercise

Regulating access to API resources with RBAC

Exam Essentials

What to focus on for the exam

- All clients will make a call to the API server which will take care of authentication and authorization.
- Role-Based Access Control (RBAC) defines the permissions for permitted operations on specific API resources. RBAC kicks in everything the API server receives a request.
- A Role and RoleBinding define permissions for objects in a namespace. ClusterRole and ClusterRoleBinding define permissions across all namespaces.
- Process that need access to the Kubernetes API use a Service Account. The Service Account is subject that can be tied in with RBAC.

Upgrading Kubernetes

Fixing security vulnerabilities periodically

Why Upgrade?

Fixing security vulnerabilities and picking up new features and bug fixes

- Even if you used the latest Long-Term Support (LTS) Release at the time of installation, it does not guarantee that your cluster is without security vulnerabilities.
- An attacker can easily look up security vulnerabilities in the publicly disclosed Common Vulnerabilities and Exposures (CVE) database and exploit them.
- The upgrade process includes underlying operating system and the dependencies the cluster nodes run on, as well as the Kubernetes version.

Kubernetes Versioning Scheme & Release Cadence

Upgrade periodically to the next minor and patch versions

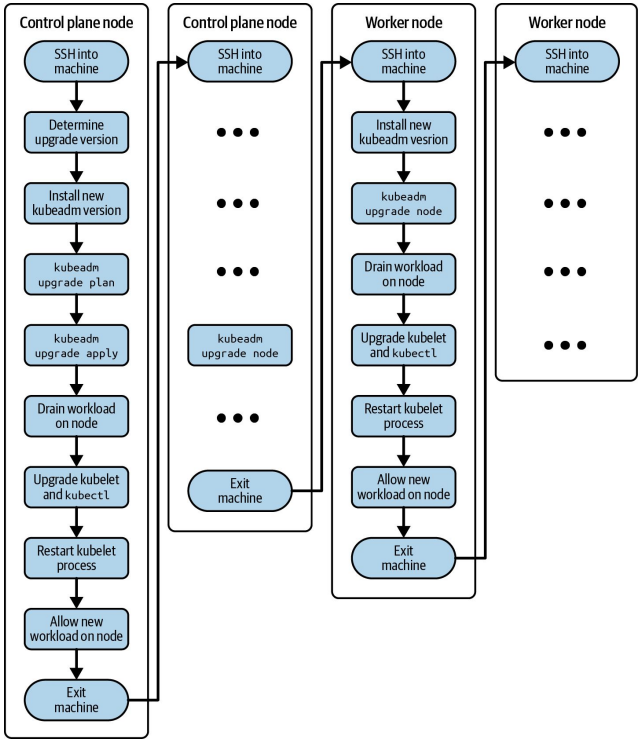
- Kubernetes follows the semantic versioning scheme. A Semantic Version consists of a major version, minor version, and patch version. For example, for the Kubernetes version 1.24.3, the major version is 1, the minor version is 24, and the patch version is 3.

- A change to the major version portion indicates a breaking change. Incrementing the minor version portion means that new functionality has been added in a backward-compatible manner. The patch version portion simply fixes a bug.

- You can expect a new minor version release of Kubernetes every three months. The release may include new features and additional bug fixes.

Kubernetes Upgrade Process

The usage of the `kubeadm` upgrade process has been tested in CKA



Exam Essentials

What to focus on for the exam

- This section is mainly about having a general awareness of Kubernetes' upgrade process.
- Understand that it's important to upgrade the operating system, its dependencies, as well as Kubernetes to fix security vulnerabilities.
- Keeping the Kubernetes platform upgraded makes it harder for attackers to infiltrate your system.

System Hardening

Topics We'll Cover

Minimizing System Attack Surface

- Minimizing the host OS footprint
- Minimizing IAM roles
- Minimizing external access to the network

Kernel Hardening Tools

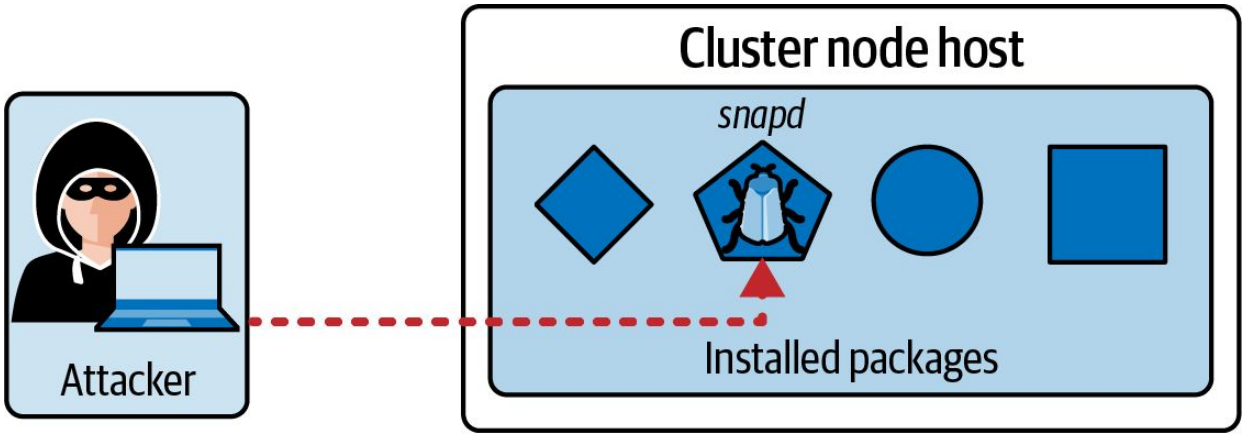
- Preventing unnecessary system calls
- Using tools like AppArmor and syscomp

Minimizing System Attack Surface

OS-level security risks and countermeasures

Scenario: Attacker Exploits a Package Vulnerability

Vulnerability of a package installed on the system



Minimizing the Host OS Footprint

Only install what you actually need

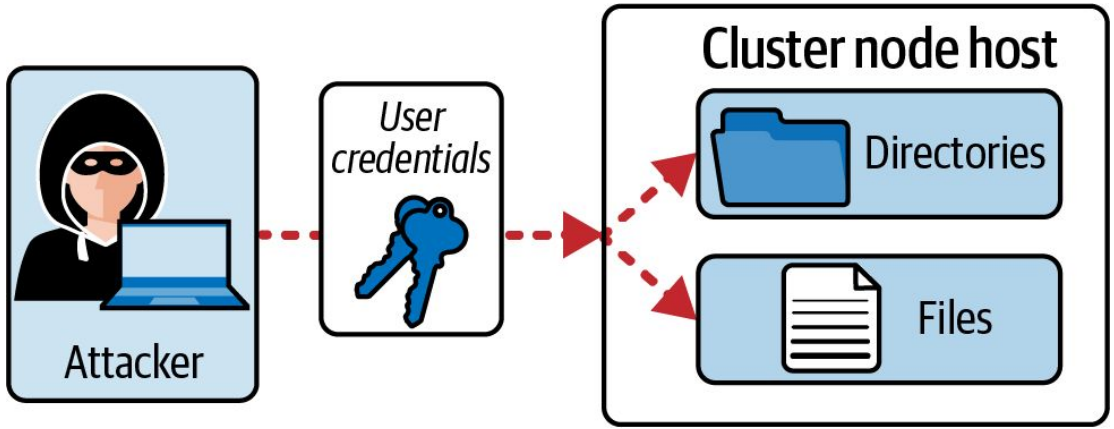
- Cluster nodes run on physical or virtual machines. In most cases, the operating system on those machines is a Linux distribution. Evidently, the operating system can expose security vulnerabilities.

- Many Linux distributions, such as Ubuntu, come with additional tools, applications, and services that are not necessarily required for operating the Kubernetes cluster

- It is your job as an administrator to identify security risks, disable or remove any operating system-specific functionality that may expose vulnerabilities, and keep the operating system patched to incorporate the latest security fixes.

Scenario: Attacker Uses Creds to Gain File Access

A security breach can lead to stolen user credentials



Minimizing IAM Roles

Only give users permission they need to perform their tasks

- Identity and access management (IAM) on the system level involves management of Linux users, the groups they belong to, and the permissions granted to them. Any directory and file will have file permissions assigned to a user.

- Every user must authenticate to a system to use it. The authenticated user has access to resources based on the assigned permissions. Minimize permissions as much as possible.

- Assigning the file permissions with as minimal access as possible is crucial to maximizing security.

Minimizing External Access to the Network

Close down ports that do not need to be open

- External access to your cluster nodes should only be allowed for the ports necessary to operate Kubernetes.
- Running applications that open network communication can expose a security risk. You can eliminate the risk by simply disabling the service and uninstalling the application.
- Another way to control ports is with the help of an operating-system-level firewall. On Linux, you could use the Uncomplicated Firewall (UFW).

Exam Essentials

What to focus on for the exam

- While mentioned in the curriculum, OS-level security features are likely not going to be touched on in the exam. It’s more about the general awareness.

- As a security specialist, you may also be responsible for maintaining the underlying machines that run Kubernetes. Make yourself familiar with the Linux tooling to manage users, file permissions, and firewalls.

Kernel Hardening Tools

Fine-grained control over system calls

What are Kernel Hardening Tools?

Controlling system calls from containers

- Applications or processes running inside of a container can make system calls. A typical example could be the curl command performing an HTTP request. A system call is a programmatic abstraction running in the user space for requesting a service from the kernel.

- We can restrict which system calls are allowed to be made with the help of kernel hardening tools. The CKS exam explicitly mentions two tools in this space, AppArmor and seccomp.

Using AppArmor

Controlling system calls from containers

- AppArmor provides access control to programs running on a Linux system. The tool implements an additional security layer between the applications invoked in the user space and the underlying system functionality.

- Many Linux distributions (e.g., Debian, Ubuntu, openSUSE) already ship with AppArmor. Therefore, AppArmor doesn't have to be installed manually. Linux distributions that do not support AppArmor use Security-Enhanced Linux (SELinux) instead, which takes a similar approach to AppArmor.

Inspecting Loaded AppArmor Profiles

AppArmor loads with existing profiles

The rules that define what a program can or cannot do are defined in an AppArmor profile. Every profile needs to be loaded into AppArmor before it can take effect.

```
$ sudo aa-status
apparmor module is loaded.
31 profiles are loaded.
31 profiles are in enforce mode.
   /snap/snapd/15177/usr/lib/snapd/snap-confine
   ...
0 profiles are in complain mode.
14 processes have profiles defined.
14 processes are in enforce mode.
   /pause (11934) docker-default
   ...
0 processes are in complain mode.
0 processes are unconfined but have a profile defined.
```

AppArmor Profiles Modes

Determines treatment of rules at runtime should a matching event happen

- *Enforce:* The system enforces the rules, reports the violation, and writes it to the syslog. You will want to use this mode to prevent a program from making specific calls.

- *Complain:* The system does not enforce the rules but will write violations to the log. This mode is helpful if you want to discover the calls a program makes.

Example AppArmor Profile

File should be placed in the directory `/etc/apparmor.d` of every worker node

```
#include <tunables/global>

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

  file,

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

Setting an AppArmor Custom Profile

Import the profile before it can be used

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

```
$ 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
   ...
```

Applying an AppArmor Profile to a Pod

Requires setting a security context

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: hello-apparmor
  securityContext:
    appArmorProfile:
      type: Localhost
      localhostProfile: k8s-deny-write

spec:
  containers:
  - name: hello
    image: busybox:1.28
    command: ["sh", "-c", "echo 'Hello AppArmor!' && sleep 1h"]
```

Exam Essentials

What to focus on for the exam

- AppArmor and seccomp are just some kernel hardening tools that can be integrated with Kubernetes to restrict system calls made from a container.
- Practice the process of loading a profile and applying it to a container.
- In order to expand your horizons, you may also want to explore other kernel functionality that works alongside Kubernetes, such as SELinux or sysctl.

Minimizing Microservice Vulnerabilities

Topics We'll Cover

Pod Security Standards

- Security context
- Pod Security Admission (PSA)

Managing Secrets

- Creating and using Secrets
- Encrypting data-at-rest in etcd

Container Runtime Sandboxes

- Purpose and use cases
- gVisor and Kata Containers

Pod-to-Pod communication encryption

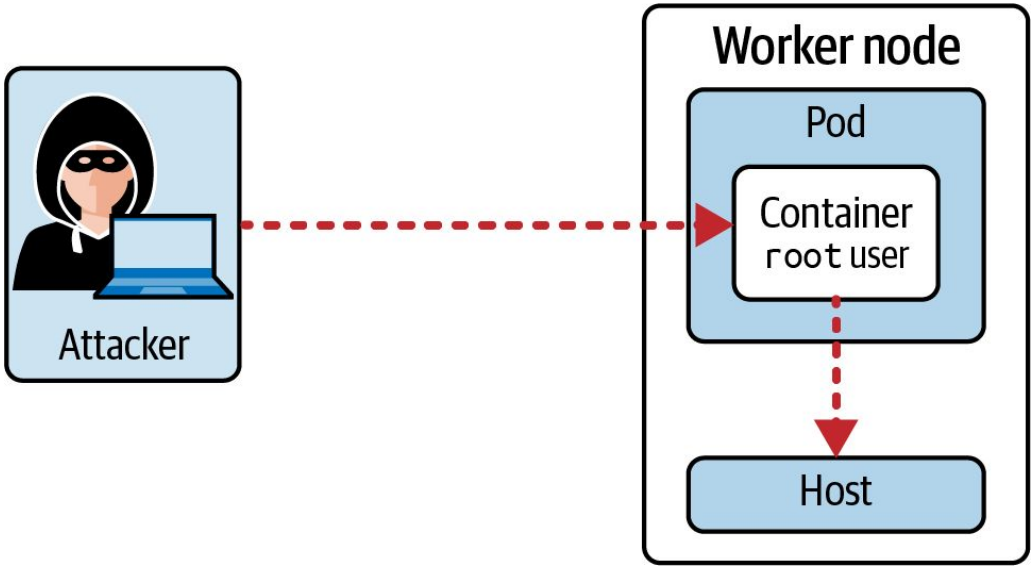
- Mutual Transport Layer Security (TLS)
- Implementing encryption with Cilium

Security Context

Privilege and access control for a Pod

Scenario: Attacker Misuses root User Access

An application vulnerability could grant attacker root access to the container



What is a Security Context?

Defines security settings for containers of a Pod

- Modeled as a set of attributes within the Pod specification, e.g. for configuring Linux capabilities, file access control, privileges to parent process or host system.

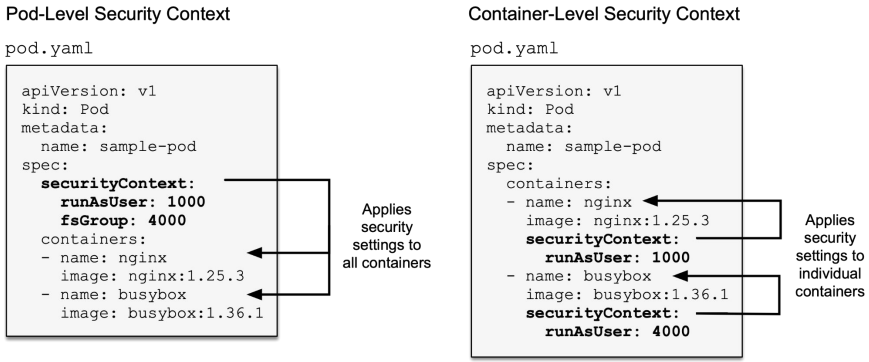
- Can be applied to all containers in Pod with `spec.securityContext` or for individual containers with `spec.containers[].securityContext`

Securing Pods

Securing settings are available through the Pod spec

- The Pod spec offers the security context concept to enforce access and privilege control.

- Configuration example: “Set the `runAsNonRoot` attribute to `true` to prevent executing the container with the root user.”



Security Context API

All attributes can be discovered in documentation for the Kubernetes version

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

Defining a Security Context

Pod- and/or container-level definition

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: secured-pod
spec:
  securityContext:
    fsGroup: 3000
  volumes:
  - name: data-vol
    emptyDir: {}
  containers:
  - image: nginx:1.18.0
    name: secured-container
    volumeMounts:
    - name: data-vol
      mountPath: /data/app
```

Defined on the Pod-level (applies to all containers)

Inspecting the Runtime Behavior

Rendering the file system Linux group ID

```
$ kubectl exec secured-pod -it -- /bin/sh
# cd /data/app
# touch hello.txt
# ls -l
total 0
-rw-r--r-- 1 root 3000 0 Apr 30 18:27 hello.txt
# exit
```

File system group ID is assigned automatically

Overriding a Value on the Container-Level

Container-level definition wins

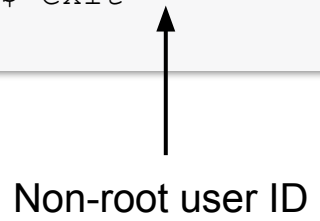
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: non-root-success
spec:
  securityContext:
    runAsNonRoot: false
  containers:
  - image: bitnami/nginx:1.23.1
    name: nginx
    securityContext:
      runAsNonRoot: true
```

Takes precedence

Inspecting the Runtime Behavior

Rendering the current user ID and group ID

```
$ kubectl exec non-root-success -it -- /bin/sh
$ id
uid=1001 gid=0(root) groups=0(root)
$ exit
```



Exercise

Defining a security context for a Pod

Exam Essentials

What to focus on for the exam

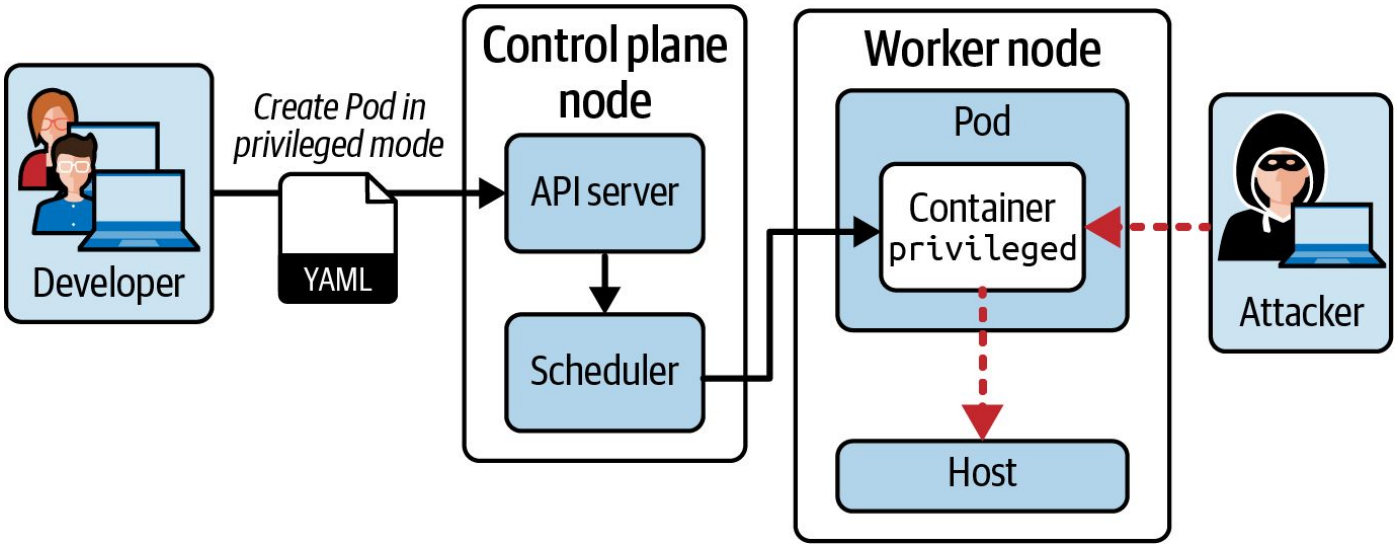
- The security context lets you provide privilege and access control to containers.
- A security context can be set on the Pod- or container-level. The Pod-level applies to all containers in a Pod. The container-level takes precedence should the same attribute be defined already on the Pod-level.
- Know where to find the security context attributes that can be set in the Kubernetes documentation for quick reference.

Pod Security Standards and Admission

Governing Pod security settings

Scenario: Developers Don’t Follow Security Practices

Can open the backdoor for attack vectors



Pod Security Standard (PSS)

Pod-specific security policies

- A PSS defines a range of security policies from highly restrictive to highly permissive.
- Most security policies describe desired security context configurations applicable to Pods.
- The different levels of policies help with gradually introducing security standards within an organization without being too disruptive.

Policy Profiles

What level of security standards do you want to apply?

Profile	Description
`privileged`	Fully unrestricted policy.
`baseline`	Minimally restrictive policy that covers crucial standards.
`restricted`	Heavily restricted policy following best practices for hardening Pods from a security perspective.

The Kubernetes documentation describe the settings per level in more detail.

Pod Security Admission (PSA)

A admission controller that governs Pods upon creation

- The PSA determines which Pod Security Standard (PSS) to follow.
- It does so by assigning the desired PSS to a namespace and the runtime treatment for Pods created in the namespace if they don't follow the standards.

> **Removed feature**
>
> PodSecurityPolicy was deprecated in Kubernetes v1.21, and removed from Kubernetes in v1.25.

PSA Modes

How should a violation be handled

Profile	Description
`enforce`	Violations will cause the Pod to be rejected.
`audit`	Pod creation will be allowed. Violations will be appended to the audit log.
`warn`	Pod creation will be allowed. Violations will be rendered on the console.

Pod Security Admission (PSA)

A admission controller that governs Pods upon creation

- The PSA functionality is enabled by default.
- As a developer, you just have to assign a reserved label to the namespace for the PSA to kick in.

```yaml
apiVersion: v1
kind: Namespace
metadata:
  name: psa
  labels:
    pod-security.kubernetes.io/enforce: restricted
```

Reserved label | Mode | Policy profile

A Pod Violating Security Standards

Does not define the expected security context configuration

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: busybox
  namespace: psa
spec:
  containers:
  - image: busybox:1.35.0
    name: busybox
    command: ["sh", "-c", "sleep 1h"]
```

A Pod Violating Security Standards

Error message is rendered, Pod creation is denied

```
$ kubectl create -f psa-namespace.yaml
namespace/psa created
$ kubectl apply -f psa-violating-pod.yaml
Error from server (Forbidden): error when creating "psa-pod.yaml": pods \
"busybox" is forbidden: violates PodSecurity "restricted:latest": \
allowPrivilegeEscalation != false (container "busybox" must set \
securityContext.allowPrivilegeEscalation=false), unrestricted \
capabilities (container "busybox" must set securityContext. \
capabilities.drop=["ALL"]), runAsNonRoot != true (pod or container \
"busybox" must set securityContext.runAsNonRoot=true), seccompProfile \
(pod or container "busybox" must set securityContext.seccompProfile. \
type to "RuntimeDefault" or "Localhost")
$ kubectl get pod -n psa
No resources found in psa namespace.
```

Exercise

Enforcing a Pod Security Standard upon Pod creation

Exam Essentials

What to focus on for the exam

- Pod Security Standards define best practice security settings for Pods. Introduce them gradually to avoid disrupting workload.
- Read up on the different levels for Pod Security Standards in the Kubernetes documentation and what security settings they apply.
- Practice the use of different levels and the effect they have on scheduled workload.

Managing Secrets

Defining and consuming sensitive data

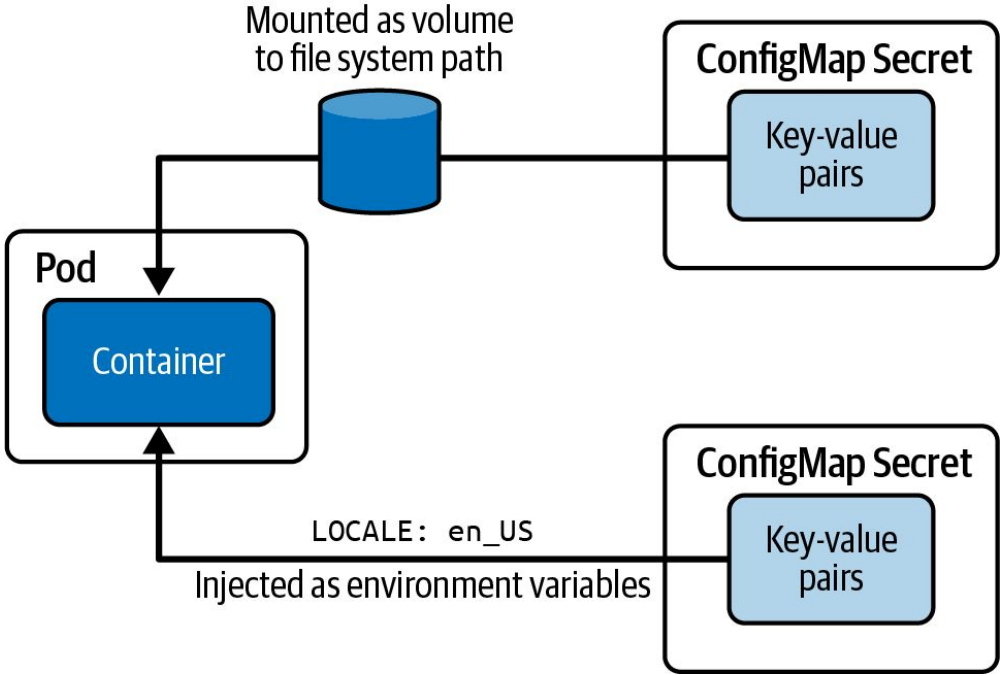
Defining Configuration Data

Control runtime behavior with centralized information

- Stored as key-value pairs in dedicated Kubernetes primitives with a lifecycle decoupled from consuming Pod.
- *ConfigMap:* Plain-text values suited for configuring application e.g. flags, or URLs.
- *Secret:* Base64-encoded values suited for storing sensitive data like passwords, API keys, or SSL certificates. **Values are not encrypted!**
- ConfigMap and Secret objects are stored in etcd in unencrypted form by default. Encryption of data in etcd is configurable.

Consuming Configuration Data

Applicable to ConfigMap and Secret



Creating a Secret with Imperative Approach

Parsed values are base64-encoded automatically

```
$ kubectl create secret generic db-creds --from-literal=pwd=s3cre!
secret/db-creds created

$ kubectl create secret docker-registry my-secret ↲
  --from-file=.dockerconfigjson=~/.docker/config.json
secret/my-secret created

$ kubectl create secret tls accounting-secret --cert=accounting.crt ↲
  --key=accounting.key
secret/accounting-secret created
```

Imperative CLI Options for Creating a Secret

You will need to provide one of the options

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

Source Options for Data Parsed by a Generic Secret

Configuration options and example values

Option	Description	Example Value
`--from-literal`	Literal values, which are key-value pairs as plain text.	`--from-literal=password=secret`
`--from-env-file`	A file that contains key-value pairs and expects them to be environment variables.	`--from-env-file=config.env`
`--from-file`	Huge page resource type.	`--from-file=id_rsa=~/.ssh/id_rsa`
`--from-file`	A directory with one or many files.	`--from-file=config-dir`

Specialized Secret Types

Can be set with `--type` CLI option, or the `type` attribute in manifest (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
`bootstrap.kubernetes.io/token`	Node bootstrap token data

Creating a Secret from Literal Values

Imperative command will base64-encode values automatically

```
$ kubectl create secret↲
  generic↲
  db-creds↲
  --from-literal=pwd=s3cre!
secret/db-creds created
```

Plain-text value

Secret YAML Manifest

Assigned values need to be provided in base64-encoded form

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

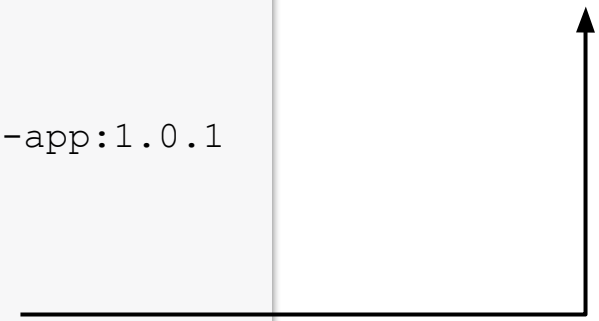
```
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
type: Opaque
data:
  pwd: czNjcmUh
```

Consuming a Secret as Environment Variables

Accessed values are base64-decoded

```
apiVersion: v1
kind: Pod
metadata:
  name: backend
spec:
  containers:
  - image: bmuschko/web-app:1.0.1
    name: backend
    envFrom:
    - secretRef:
        name: mysecret
```

```
$ kubectl exec -it backend -- env
pwd=s3cre!
```



Creating a Secret from a SSH Private Key File

Imperative creation by pointing to file

```
$ kubectl create secret generic secret-ssh-auth
  --type=kubernetes.io/ssh-auth --from-file=ssh-privatekey
secret/secret-ssh-auth created
```

ssh-privatekey

```
-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info:
AES-128-CBC,8734C9153079F2E8497C8075289EBBF1
...
-----END RSA PRIVATE KEY-----
```

Consuming a Secret as a Volume

A good choice for processing a machine-readable configuration file

```
apiVersion: v1
kind: Pod
metadata:
  name: backend
spec:
  containers:
    - name: backend
      image:
bmuschko/web-app:1.0.1
      volumeMounts:
      - name: ssh-volume
        mountPath: /var/app
        readOnly: true
  volumes:
    - name: ssh-volume
      secret:
        secretName:
secret-ssh-auth
```

```
$ kubectl exec -it backend -- /bin/sh
# ls -l /var/app
ssh-privatekey
# cat /var/app/ssh-privatekey
-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info:
AES-128-CBC,8734C9153079F2E8497C8075289E
BBF1
...
-----END RSA PRIVATE KEY-----
```

Plain-Text Secret Values

The `stringData` attribute allows for plain-text values

secret.yaml

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

```
kubectl create -f
secret.yaml
```

Live object

```
apiVersion: v1
kind: Secret
metadata:
  name: secret-basic-auth
type: kubernetes.io/basic-auth
stringData:
  username: YWRtaW4=
  password: dDBwLVNlY3JldA==
```

Secrets Are Not Really Secret

Techniques that help with making Secrets more secure

- Secret objects are stored in etcd in unencrypted form by default. Encryption of data in etcd is configurable.

- Define RBAC permissions to only allow developers to create, view, and modify Secret objects in dedicated namespaces. An even stricter policy could only allow administrators to manage Secrets.

- Use Kubernetes-external, third-party Secrets services for managing sensitive data, e.g. AWS Secrets Manager or HashiCorp Vault. You can consume the data with the help of the External Secrets Operator.

Exercise

Creating and consuming a Secret

Exam Essentials

What to focus on for the exam

- ConfigMaps store plain-text key-value pairs. They are a good fit for simple strings, e.g. connection URLs, and theme names.

- Secrets are meant to represent sensitive data, however, they are not encrypted and therefore not “secret”. The values of the key-value pairs are only base64-encoded.

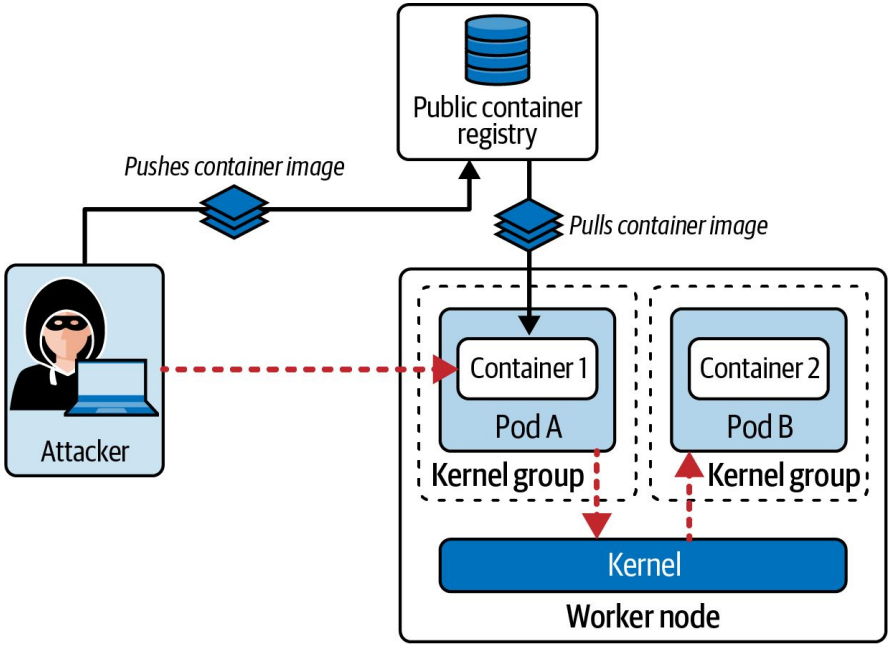
- ConfigMaps and Secrets can be consumed by Pods as environment variables or Volumes. Choose the appropriate method based on your application’s implementation. Practice both approaches.

Container Runtime Sandboxes

Stronger isolation of workload running containers

Scenario: Attacker Gains Access to Another Container

A container image that has been tampered with can break out of container



What is a Container Runtime Sandbox?

Stronger isolation of workloads running in a container

- Generally speaking, it’s not a good idea to blindly trust public container images. One way to ensure that such a container image runs with more isolation is the container runtime sandbox.

- Why would you want to use them?
  - You don’t trust the workload as it could execute malicious operations.
  - For 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)

Container Runtime Sandbox Implementations

Other implementations are available

- Kata containers achieves container isolation by running them in a lightweight virtual machine.

- gVisor takes a different approach. It effectively implements a Linux kernel that runs on the host system. Therefore, syscalls are not shared anymore across all containers on the host system.

- Container runtime sandboxes have already been installed and configured with the container runtime. See the corresponding installation instructions for more details.

Creating and Using a Runtime Class

Reference RuntimeClass by name in Pod

```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
```

Exam Essentials

What to focus on for the exam

- Container runtime sandboxes help with adding stricter isolation to containers. You will not be expected to install a container runtime sandbox, such as Kata Containers or gVisor.

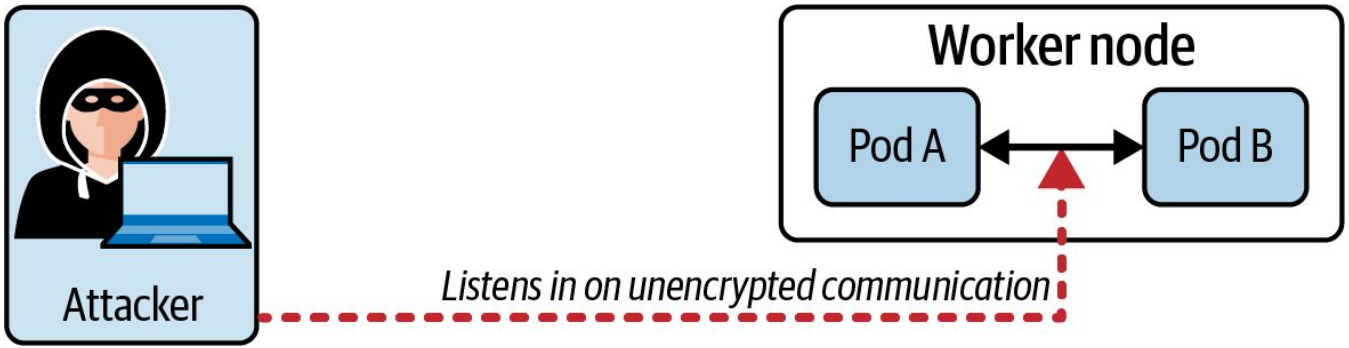
- You do need to understand the process for configuring a container runtime sandbox with the help of a RuntimeClass object and how to assign the RuntimeClass to a Pod by name.

Pod-to-Pod Encryption with mTLS

Enabling encrypted network communication

Scenario: Attacker Listens to Pod Communication

Impersonating the sending or the receiving side to extract sensitive data



What is Mutual TLS?

Enables encrypted Pod-to-Pod communication

- Every Pod can talk to any other Pod by targeting its virtual IP address unless you put a more restrictive network policy in place. The communication between two Pods is unencrypted by default.

- Mutual TLS (mTLS) is like TLS, but both sides have to authenticate. This approach has the following benefits. First, you achieve secure communication through encryption. Second, you can verify the client identity. An attacker cannot easily impersonate another Pod.

Adopting mTLS in Kubernetes

Enables encrypted Pod-to-Pod communication

- In most cases, Kubernetes administrators rely on a Kubernetes service mesh to implement mTLS instead of implementing it manually. A Kubernetes service mesh, such as Linkerd or Istio, is a tool for adding cross-cutting functionality to your cluster, like observability and security.

- Another option is to use transparent encryption to ensure that traffic doesn’t go on the wire unencrypted. Some of the popular CNI plugins, such as Calico and Cilium, have added support for WireGuard.

Exam Essentials

What to focus on for the exam

- Setting up mTLS for all microservices running in a Pod can be extremely tedious due to certificate management. For the exam, understand the general use case for wanting to set up mTLS for Pod-to-Pod communication.

- You are likely not expected to actually implement it manually, though. Production Kubernetes clusters use services meshes to provide mTLS as a feature.

Supply Chain Security

Topics We'll Cover

Minimizing Base Image Footprint

- Picking a small base image
- Multi-stage Dockerfiles
- Reducing the number of layers

Securing the Supply Chain

- Signing container images
- Validating container images
- Whitelisting allowed image registries

Static Analysis of Workload

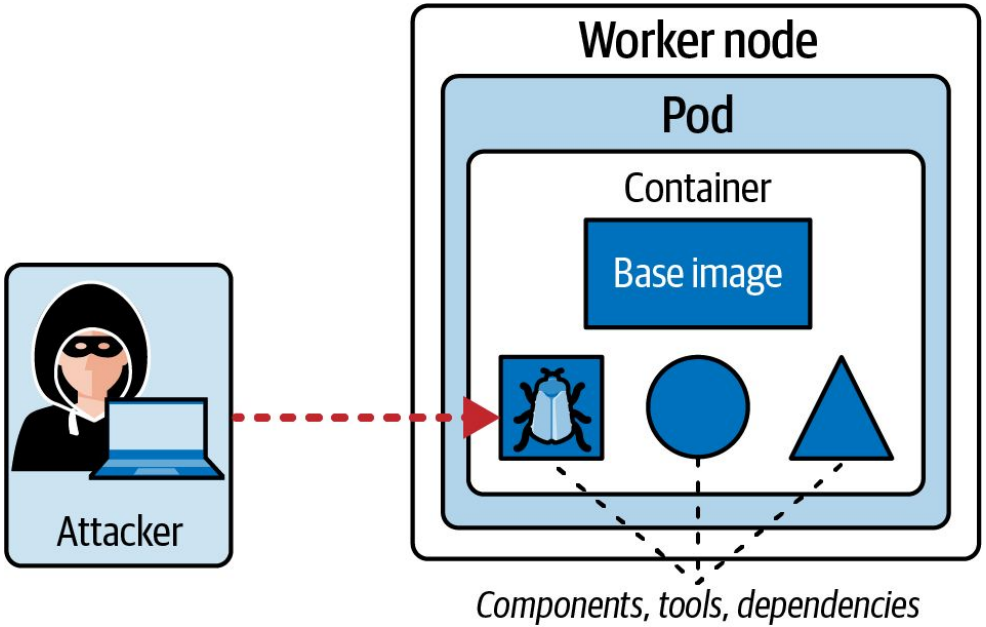
- Analyzing Dockerfiles
- Analyzing Kubernetes Manifests
- Scanning Images for Known Vulnerabilities

Minimizing Base Image Footprint

Techniques for reducing image size

Scenario: Attacker Exploits Container Vulnerabilities

Vulnerabilities can now be used as a launching pad for more advanced attacks



Picking a Base Image Small in Size

Start by looking at the size before deciding to use it

- It is recommended to use a base image with a minimal set of functionality and dependencies. The less is included, the less potential vulnerabilities do you ship. Upgrade to newer base image versions to pull in security vulnerability fixes.

- Many container producers upload a wide range of variations of their container images for the same release. One of those variations is an `alpine` image, a small, lightweight, and less vulnerable Linux distribution.

- You can further reduce the container image size and the attack surface by using a distroless image offered by Google, an image without a shell.

Building Multi-Stage Container Images

Stripping tools used to build binary from container image

- Some developers prefer building the application binary directly as part of the container image creation process.
- Separate the build stage from the runtime stage. As a result, all dependencies needed in the build stage will be discarded after the process has been performed and therefore do not end up in the final container image.
- This approach automatically leads to a much smaller container image size by removing all the unnecessary cruft.

Example Multi-Stage Dockerfile

Output of named stage is copied to final container image

```
FROM golang:1.19.4-alpine AS build
RUN apk add --no-cache git
WORKDIR /tmp/go-sample-app
COPY go.mod .
COPY go.sum .
RUN go mod download
COPY . .

RUN CGO_ENABLED=0 go test -v
RUN go build -o ./out/go-sample-app .

FROM alpine:3.17.0
RUN apk add ca-certificates
COPY --from=build /tmp/go-sample-app/out/go-sample-app /app/go-sample-app
CMD ["/app/go-sample-app"]
```

Reducing the Number of Layers

Roll multiple commands into one line of instruction

- Every Dockerfile consists of an ordered list of instructions. Only some instructions create a read-only layer in the resulting container image. Those instructions are `FROM`, `COPY`, `RUN`, and `CMD`.

- All other instructions will not create a layer as they create temporary intermediate images.

- It’s common practice to execute multiple commands in a row to avoid creating extra layers.

Example Combining Commands

Use `&&` to execute multiple commands in a single layer

```
FROM ubuntu:22.10
RUN apt-get update -y
RUN apt-get upgrade -y
RUN apt-get install -y curl
```

vs.

```
FROM ubuntu:22.10
RUN apt-get update -y && apt-get upgrade -y && apt-get install -y curl
```

Container Image Optimization Tools

Inspecting a produced container image for further optimization potential

- *DockerSlim* will optimize and secure your container image by analyzing your application and its dependencies. You can find more information in the tool's GitHub repository.

- *Dive* is a tool for exploring the layers baked into a container image. It makes it easy to identify unnecessary layers, which you can further optimize on. The code and documentation for Dive are available in a GitHub repository.

Exam Essentials

What to focus on for the exam

- I would suggest you read the best practices mentioned on the Docker web page and try to apply them to sample container images. Compare the size of the produced container image before and after applying a technique.

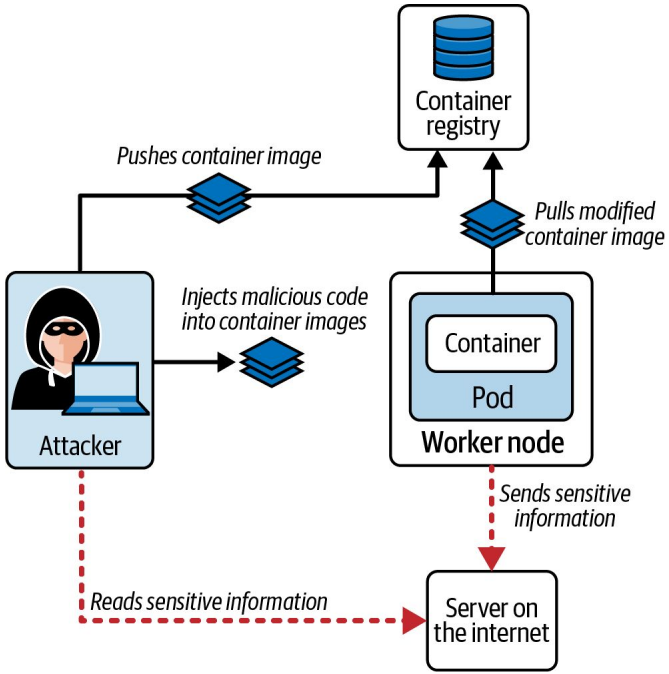
- You can try to challenge yourself by reducing a container image to the smallest size possible while at the same time avoiding the loss of crucial functionality.

Securing the Supply Chain

Techniques for automating the process of producing a container image and operating it in a runtime environment

Scenario: Attacker Injects Malicious Code

Avoid automatically pulling the latest version of a container image

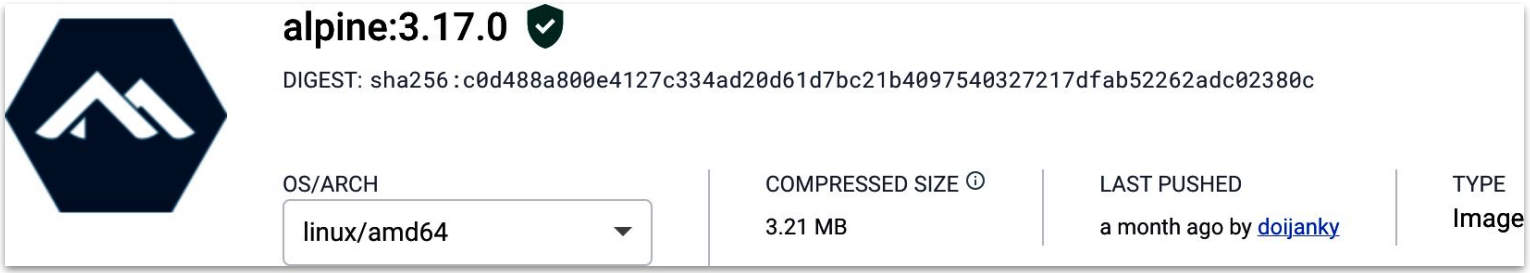


Validating Container Images

Many images provide a corresponding image digest in registry

- Image digest validation is an opt-in functionality in Kubernetes. When defining Pods, make sure you spell out the image digest explicitly for all container images.

- Append the SHA256 image digest to the specification of a container. For example, this can be achieved with the attribute `spec.containers[].image`



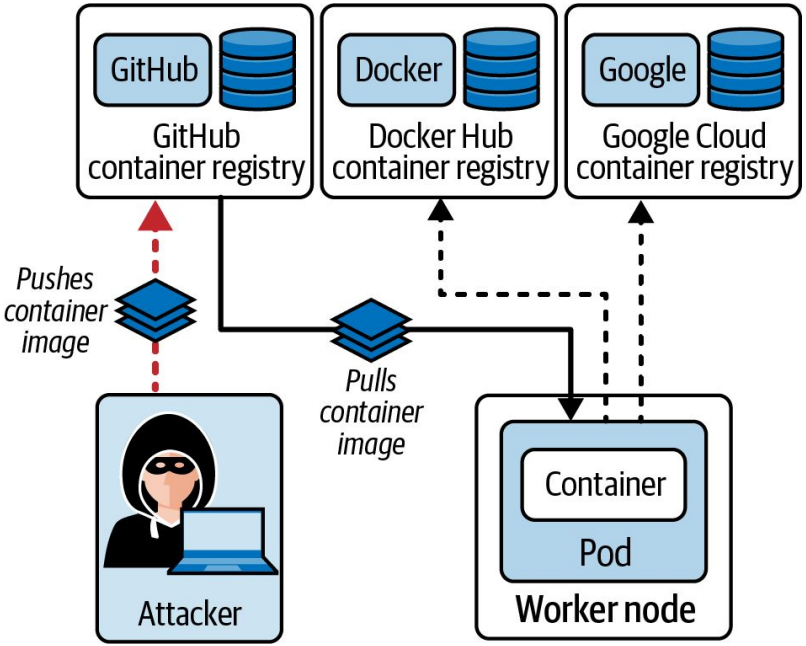
Example Pod Using an Image Digest

The container runtime does not pull image if digest doesn't match

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

Scenario: Attacker Uploads a Malicious Image

Avoid pulling images from public registries



Only Allow the Use of Internal Image Registries

Set up and control your own registry without the risk of pulling public images

- On an enterprise level, you need to control which container registries you trust. It's recommended to set up your own container registry within your company's network, which you can fully control and govern. Alternatively, you can set up a private container registry in a cloud provider environment, not accessible by anyone else.

- One of the prominent binary repository managers you can choose from is JFrog Artifactory. The product fully supports storing, scanning, and serving container images.

- One way to govern container registry usage is with Kyverno and OPA Gatekeeper, or an admission controller plugin.

Example Pod With Registry Reference

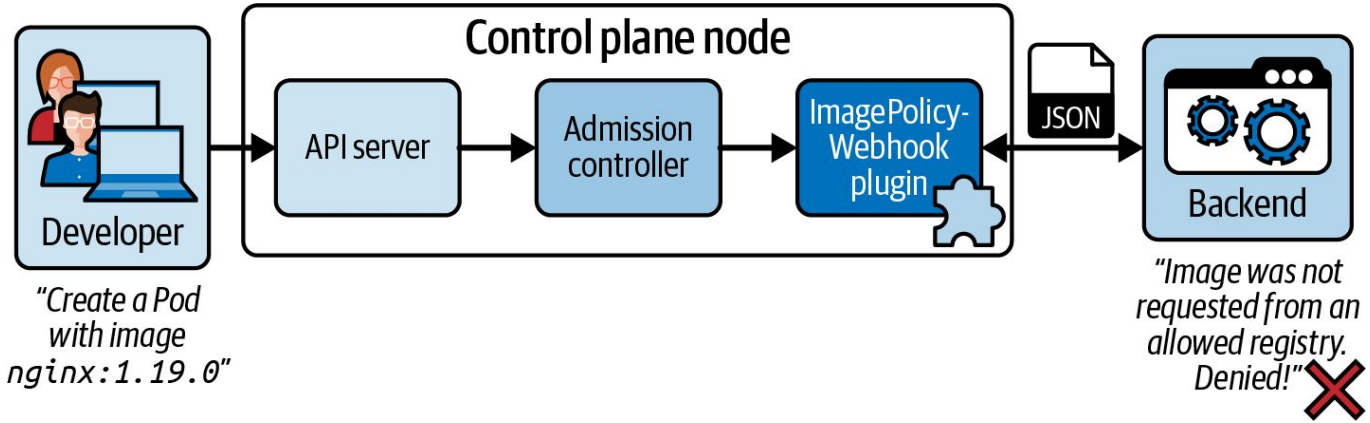
The registry will very likely want you to authenticate

```
apiVersion: v1
kind: Pod
metadata:
  name: alpine
spec:
  containers:
  - name: alpine
    image: my-company-docker-local.jfrog.io/my-app:1.0.0
  imagePullSecrets:
  - name: regcred
```

Put credentials into a Secret
and reference it here

Using an Admission Controller Plugin

Validation logic lives in a backend service external to Kubernetes



ImagePolicyWebhook admission controller plugin

The plugin defines policies for requested images

- The admission controller provides a way to approve, deny, or mutate a request before the request takes effect.
- The ImagePolicyWebhook admission controller plugin is one of the plugins we can configure for intercepting Kubernetes API calls. The backend implements a policy for all defined container images in a Pod.
- The admission controller plugin needs to be registered with the API server. Communication between the admission controller and the backend needs to happen over HTTPS.

Admission Controller Configuration File

Configures all plugins you want to register with API server

/etc/kubernetes/admission-control/image-policy.yaml

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

Kubeconfig File

Sets up communication with backend via TLS

/etc/kubernetes/admission/imagepolicywebhook.kubeconfig

```
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

Configuring the API Server

Enables plugin and points to configuration file

/etc/kubernetes/manifests/kube-apiserver.yaml

```
spec:
  containers:
  - command:
    - kube-apiserver
    - --enable-admission-plugins=NodeRestriction,ImagePolicyWebhook
    - --admission-control-config-file=/etc/kubernetes/admission-control/image-policy.yaml
    ...
    volumeMounts:
    ...
    - name: admission-control
      mountPath: /etc/kubernetes/admission-control
      readonly: true
  volumes:
  ...
  - name: admission-control
    hostPath:
      path: /etc/kubernetes/admission-control
      type: DirectoryOrCreate
```

Exam Essentials

What to focus on for the exam

- After building a container image, make sure to also create a digest for it. Publish the container image, as well as the digest, to a registry. Practice how to use the digest in a Pod definition and verify the behavior of Kubernetes upon pulling the container image.

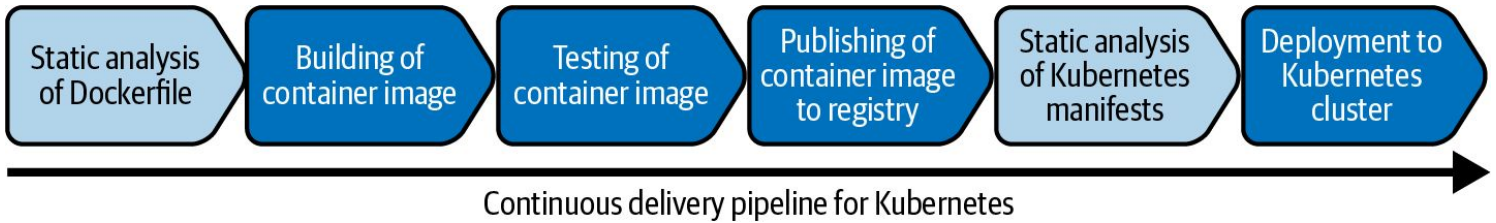
- You are not expected to write a backend for an ImagePolicyWebhook. That's out of scope for the exam and requires knowledge of a programming language. You do need to understand how to enable the plugin in the API server configuration, though. I would suggest you practice the workflow even if you don't have a running backend application available.

Static Analysis of Workload

Automated scanning of container images and Kubernetes workload

Scenario: CI/CD Pipelines Integrate Scanning Tools

Scanning manifests and container images should happen automatically



Using Hadolint for Analyzing Dockerfiles

Check against best practices

Haskell Dockerfile Linter, also called hadolint in short, is a linter for Dockerfiles. The tool inspects a Dockerfile based on best practices listed on the Docker documentation page.

```
$ hadolint Dockerfile
Dockerfile:1 DL3006 warning: Always tag the version of an image
explicitly
Dockerfile:2 DL3045 warning: `COPY` to a relative destination
without `WORKDIR` set.
```

Using Kubesec for Analyzing Kubernetes Manifests

Check against best practices

Kubesec is a tool for analyzing Kubernetes manifests. It can be executed as a binary, Docker container, admission controller plugin, and even as a plugin for `kubectl`.

```
$ docker run -i kubesec/kubesec:512c5e0 scan /dev/stdin < pod-initial-kubesec-test.yaml
[
  {
    "object": "Pod/kubesec-demo.default",
    "valid": true,
    "message": "Passed with a score of 1 points",
    "score": 1,
    "scoring": {
      "advise": [
        {
          "selector": ".spec .serviceAccountName",
          "reason": "Service accounts restrict Kubernetes API access and \
                     should be configured with least privilege"
        },
 ...
```

Scanning Images for Known Vulnerabilities

Finding embedded security vulnerabilities

- One of the top sources for logging and discovering security vulnerabilities is CVE Details. The page lists and ranks known vulnerabilities (CVEs) by score. One of the open source tools with this capability explicitly mentioned in the CKS curriculum is Trivy.

- Trivy can run in different modes of operation: as a command line tool, in a container, as a GitHub Action configurable in a continuous integration workflow, as a plugin for the IDE VSCode, and as a Kubernetes operator.

Example Trivy

Renders detailed information on vulnerabilities

```
bmuschko@ascent: ~                                                    ⌥⌘1
~ trivy image python:3.4-alpine                          ✓ 10119  18:11:18
2023-04-04T18:11:24.515-0600    INFO    Need to update DB
2023-04-04T18:11:24.515-0600    INFO    DB Repository: ghcr.io/aquasecurity/trivy-db
2023-04-04T18:11:24.515-0600    INFO    Downloading DB...
36.16 MiB / 36.16 MiB [-------------------------------------------------------------------------] 100.00% 8.29 MiB p/s 4.6s
2023-04-04T18:11:30.483-0600    INFO    Vulnerability scanning is enabled
2023-04-04T18:11:30.483-0600    INFO    Secret scanning is enabled
2023-04-04T18:11:30.483-0600    INFO    If your scanning is slow, please try '--security-checks vuln' to disable secret scanning
2023-04-04T18:11:30.483-0600    INFO    Please see also https://aquasecurity.github.io/trivy/v0.36/docs/secret/scanning/#recommendation for faster secret detection
2023-04-04T18:11:31.922-0600    INFO    Detected OS: alpine
2023-04-04T18:11:31.922-0600    INFO    Detecting Alpine vulnerabilities...
2023-04-04T18:11:31.923-0600    INFO    Number of language-specific files: 1
2023-04-04T18:11:31.923-0600    INFO    Detecting python-pkg vulnerabilities...
2023-04-04T18:11:31.926-0600    WARN    This OS version is no longer supported by the distribution: alpine 3.9.2
2023-04-04T18:11:31.926-0600    WARN    The vulnerability detection may be insufficient because security updates are not provided

python:3.4-alpine (alpine 3.9.2)

Total: 37 (UNKNOWN: 0, LOW: 4, MEDIUM: 16, HIGH: 13, CRITICAL: 4)
```

Library	Vulnerability	Severity	Installed Version	Fixed Version	Title
expat	CVE-2018-20843	HIGH	2.2.6-r0	2.2.7-r0	expat: large number of colons in input makes parser consume high amount... https://avd.aquasec.com/nvd/cve-2018-20843
	CVE-2019-15903			2.2.7-r1	expat: heap-based buffer over-read via crafted XML input https://avd.aquasec.com/nvd/cve-2019-15903
libbz2	CVE-2019-12900	CRITICAL	1.0.6-r6	1.0.6-r7	bzip2: out-of-bounds write in function BZ2_decompress https://avd.aquasec.com/nvd/cve-2019-12900
libcrypto1.1	CVE-2019-1543	HIGH	1.1.1a-r1	1.1.1b-r1	openssl: ChaCha20-Poly1305 with long nonces https://avd.aquasec.com/nvd/cve-2019-1543
	CVE-2020-1967			1.1.1g-r0	openssl: Segmentation fault in SSL_check_chain causes denial of service https://avd.aquasec.com/nvd/cve-2020-1967

Exercise

Container image scanning with Trivy

Exam Essentials

What to focus on for the exam

- During the exam, you may be asked to run a specific command for a given static analysis tool that will produce a list of error and/or warning messages. Understand how to interpret the messages, and how to fix them in the relevant resource files.

- You will need to understand the different ways to invoke Trivy to scan a container image. The produced report will give a you clear indication on what needs to be fixed and the severity of the found vulnerability. Given that you can’t modify the container image easily, you will likely be asked to flag Pods that run container images with known vulnerabilities.

Monitoring, Logging, and Runtime Security

Topics We'll Cover

Behavior Analytics

- Why track events in cluster?
- Using Falco
- Falco configuration options

Container Immutability

- Why is immutability important?
- What makes a container immutable?
- Pod configuration options

Audit Logging

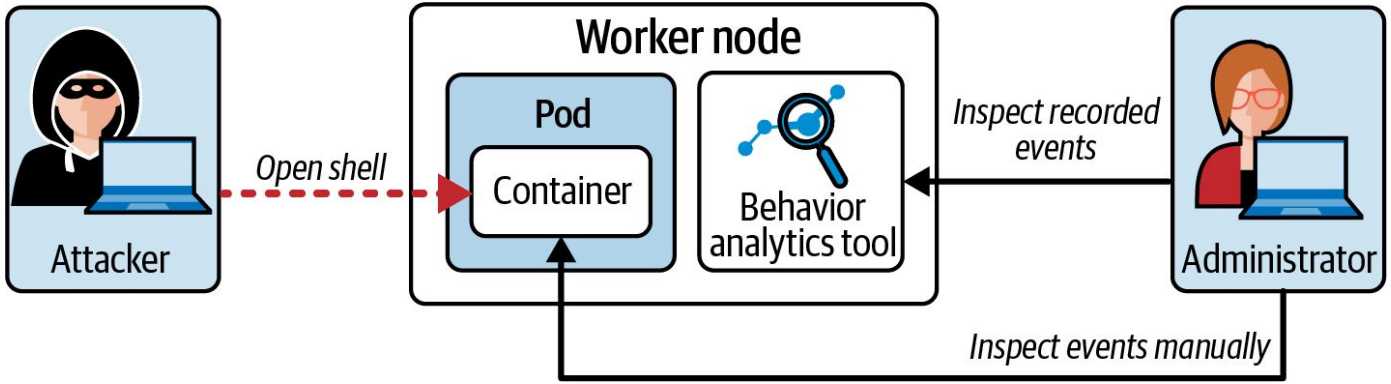
- Information captured
- Configuring auditing rules
- Example audit log entries

Behavior Analytics

Observing cluster nodes for unusual events

Scenario: Administrator Observes Attacker Actions

You need to be able to identify malicious intent



What is Behavior Analytics?

Observing unusual activity

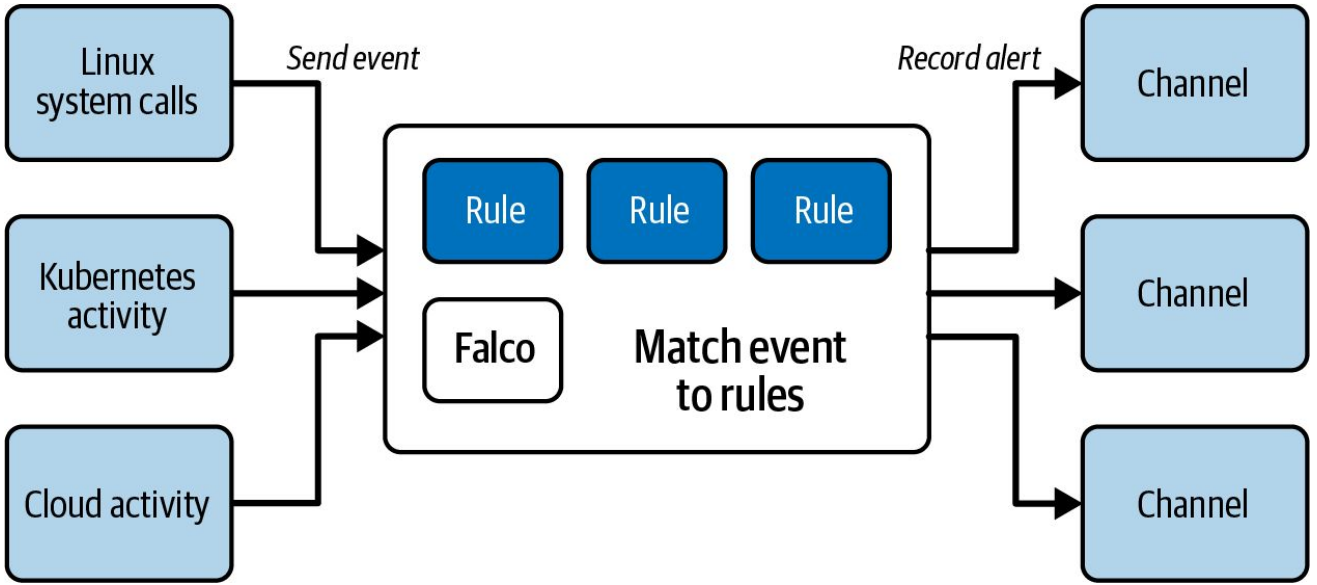
- Apart from managing and upgrading a Kubernetes cluster, the administrator is in charge of keeping an eye on potentially malicious activity. While you can perform this task manually by logging into cluster nodes and observing host- and container-level processes, it is a horribly inefficient undertaking.

- *Behavior analytics* is the process of observing the cluster nodes for any activity that seems out of the ordinary. An automated process helps with filtering, recording, and alerting events of specific interest.

- Among the behavior analytics tools relevant to the exam are Falco, Tracee, and Tetragon.

Falco Architecture

Detects threats by observing host- and container-level activity



Falco Components

Terminology

- Falco deploys a set of *sensors* that listen for the configured events and conditions. Each sensor consists of a set of *rules* that map an event to a data source.

- An *alert* is produced when a rule matches a specific *event*.

- Alerts will be sent to an output *channel* to record the event, such as standard output, a file, or an HTTPS endpoint. Falco allows for enabling more than one output channel simultaneously.

Configuring Falco

Allow for configuring the Falco runtime, and its rules

```
$ tree /etc/falco
/etc/falco
├── aws_cloudtrail_rules.yaml
├── falco.yaml
├── falco_rules.local.yaml
├── falco_rules.yaml
├── k8s_audit_rules.yaml
├── rules.available
│   └── application_rules.yaml
└── rules.d
```

Default Falco Logs

Inspect logged events by using the `journalctl` command

```
$ sudo journalctl -fu falco
...
Jan 24 18:03:37 kube-worker-1 falco[8718]: 18:03:14.632368639:
Notice A shell ↲
was spawned in a container with an attached terminal (user=root
user_loginuid=0 ↲
nginx (id=18b247adb3ca) shell=bash parent=runc cmdline=bash
pid=47773 ↲
terminal=34816 container_id=18b247adb3ca
image=docker.io/library/nginx)
```

Falco Rule

Falco comes with a default rule set

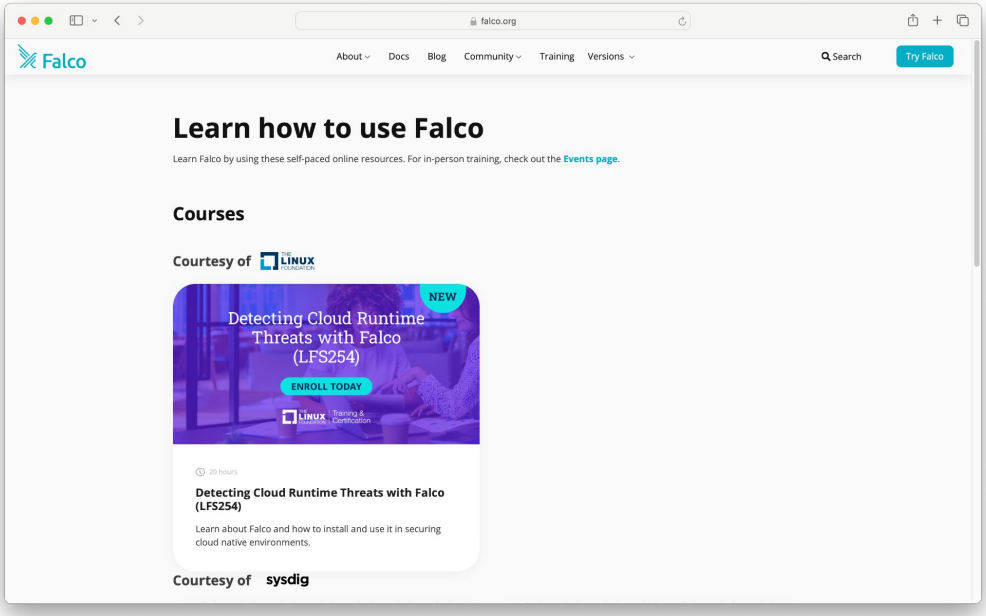
/etc/falco/falco_rules.yaml

```
- 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 Video Course

Deep dive with a free online trainings



Exercise

Intrusion detection with Falco

Exam Essentials

What to focus on for the exam

- Falco is definitely going to come up as a topic during the exam. You will need to understand how to read and modify a rule in a configuration file. I would suggest you browse through the syntax and options in more detail in case you need to write one yourself.

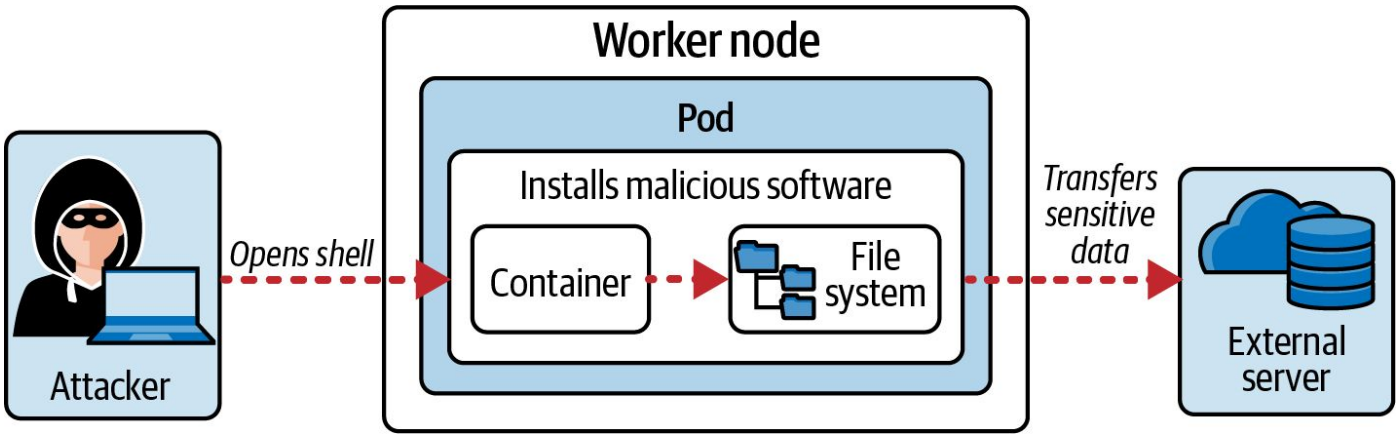
- The main entry point for running Falco is the command line tool. It’s fair to assume that it will have been pre-installed in the exam environment.

Container Immutability

Logging API server events

Scenario: Attacker Installs Malicious Software

Container(s) allow file write operation



What Makes a Container Immutable?

Do not allow modifications to the container once started

- *Distroless container images* are not just smaller in size; they are also more secure. An attacker cannot shell into the container, and therefore the filesystem cannot be misused to install malicious software.

- It is best practice to use the same container image for different deployment environments, even though their runtime configurations may be different. You can *inject configuration data* into a container with the help of a ConfigMap or Secret as environment variables or files mounted via Volumes.

- Configuring a *read-only container filesystem* by setting `spec.containers[].securityContext.readOnlyRootFilesystem` to `true`.

Example of an Immutable Container

Nginx needs to write to file system, but you should provide a mount instead

```
spec:
  containers:
  - name: nginx
    image: nginx:1.21.6
    securityContext:
      readOnlyRootFilesystem: true
    volumeMounts:
    - name: nginx-run
      mountPath: /var/run
    - name: nginx-cache
      mountPath: /var/cache/nginx
    - name: nginx-data
      mountPath: /usr/local/nginx
  volumes:
  - name: nginx-run
    emptyDir: {}
  - name: nginx-data
    emptyDir: {}
  - name: nginx-cache
    emptyDir: {}
```

Exercise

Verifying container immutability

Exam Essentials

What to focus on for the exam

- Immutable containers are a central topic to this exam domain. You will have to either create a Pod with immutable configuration or be able to identify an existing Pod for immutability.

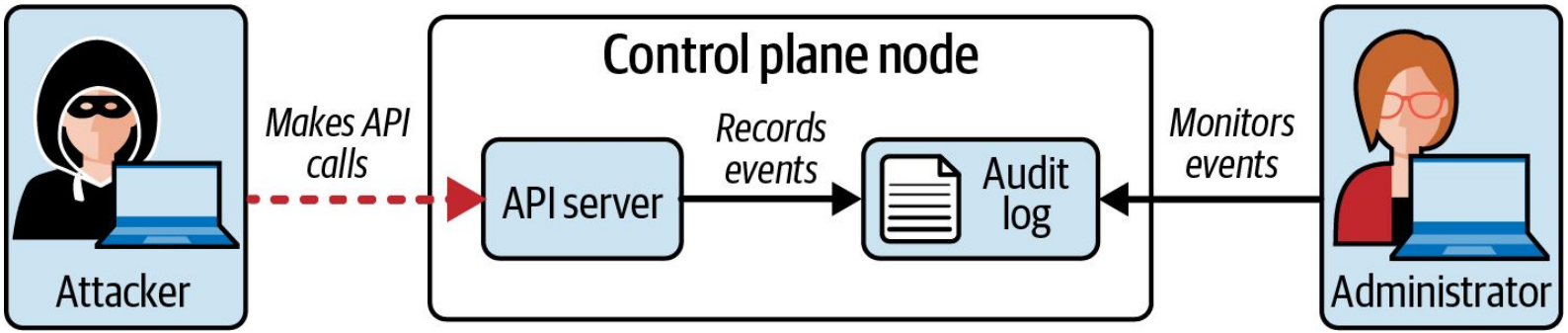
- Understand how to set the `spec.containers[].securityContext.readOnlyRootFilesystem` attribute for a Pod and how to mount a Volume to a specific path in case a write operation is required by the container process.

Audit Logging

Logging API server events

Scenario: Monitoring Malicious Events in Real Time

Observing unauthorized access and historical tracking of API requests



Information Captured by Audit Logging

Events emitted by API server or by client requests to the API server

- What event occurred?
- Who triggered the event?
- When was it triggered?
- Which Kubernetes component handled the request?

Audit Logging Terminology

Events emitted by API server or by client requests to the API server

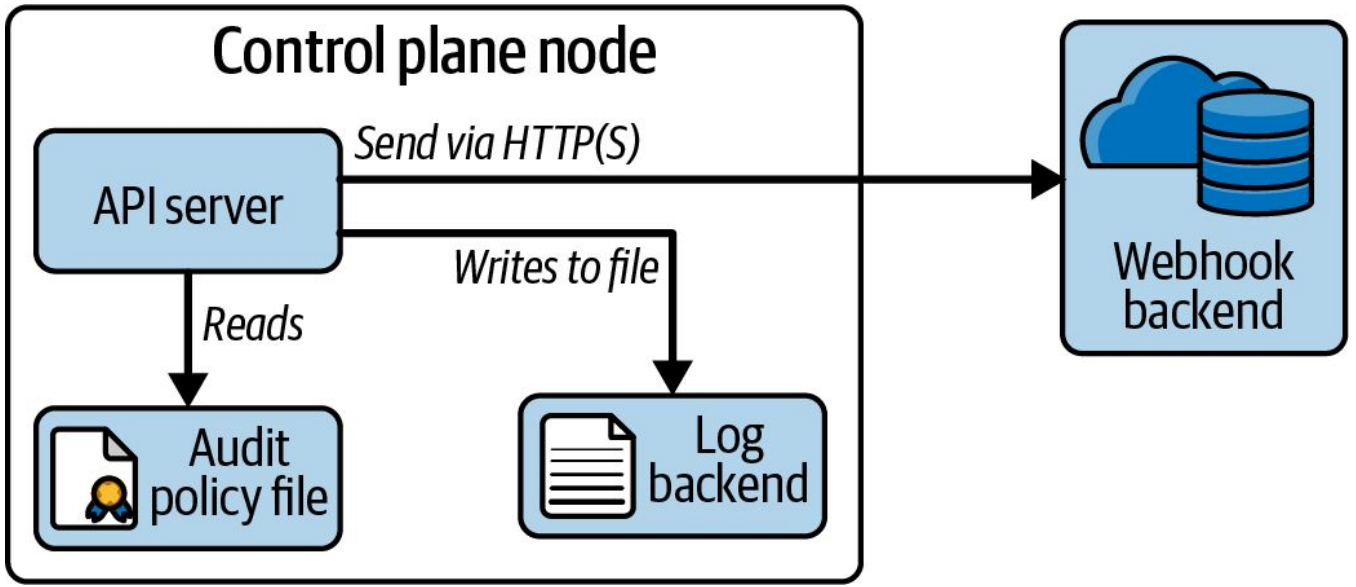
- *Audit policy:* Defines the type of event and the corresponding request data to be recorded. The audit policy is a YAML manifest specifying those rules and has to be provided to the API server process.

- *Audit backend:* Responsible for storing the recorded audit events, as defined by the audit policy.

- You have two configurable options for a backend:
  - A log backend, which write the events to a file.
  - A webhook backend, which sends the events to an external service via HTTP(S) - for example, for the purpose of integrating a centralized logging and monitoring system

Audit Logging Architecture

Involved components



Example Audit Policy File

Events emitted by API server or by client requests to the API server

```
apiVersion: audit.k8s.io/v1
kind: Policy
omitStages:
  - "RequestReceived"
rules:
  - level: RequestResponse
    resources:
    - group: ""
      resources: ["pods"]
  - level: Metadata
    resources:
    - group: ""
      resources: ["pods/log", "pods/status"]
```

Prevents generating logs for all requests in the `RequestReceived` stage

Logs Pod changes at `RequestResponse` level

Logs specialized Pod events, e.g., log and status requests, at the `Metadata` level

Audit Levels

Logging on a high level requires more storage space

Level	Description
`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.

Configuring Audit Logging

Modify the API server configuration file

```
spec:
  containers:
  - command:
    - kube-apiserver
    - --audit-policy-file=/etc/kubernetes/audit-policy.yaml
    - --audit-log-path=/var/log/kubernetes/audit/audit.log
    ...
    volumeMounts:
    - mountPath: /etc/kubernetes/audit-policy.yaml
      name: audit
      readOnly: true
    - mountPath: /var/log/kubernetes/audit/
      name: audit-log
      readOnly: false
  ...
  volumes:
  - name: audit
    hostPath:
      path: /etc/kubernetes/audit-policy.yaml
      type: File
  - name: audit-log
    hostPath:
      path: /var/log/kubernetes/audit/
      type: DirectoryOrCreate
```

Provides the location of the policy file and log file to the API server process.

Mounts the policy file and the audit log directory to the given paths.

Defines the Volumes for the policy file and the audit log directory.

Example Audit Logging File Output

Produces log entries in JSON format

```
$ sudo grep 'audit.k8s.io/v1' /var/log/kubernetes/audit/audit.log
...
{"kind":"Event","apiVersion":"audit.k8s.io/v1","level":"RequestResponse", \
"auditID":"285f4b99-951e-405b-b5de-6b66295074f4","stage":"ResponseComplete", \
"requestURI":"/api/v1/namespaces/default/pods/nginx","verb":"get", \
"user":{"username":"system:node:node01","groups":["system:nodes", \
"system:authenticated"]},"sourceIPs":["172.28.116.6"],"userAgent": \
"kubelet/v1.26.0 (linux/amd64) kubernetes/b46a3f8","objectRef": \
{"resource":"pods","namespace":"default","name":"nginx","apiVersion":"v1"}, \
"responseStatus":{"metadata":{},"code":200},"responseObject":{"kind":"Pod", \
"apiVersion":"v1","metadata":{"name":"nginx","namespace":"default", \
...
```

Exercise

Configuring and using audit logging

Exam Essentials

What to focus on for the exam

- Setting up audit logging consists of two steps. For one, you need to understand the syntax and structure of an audit policy file. The other aspect is how to configure the API server to consume the audit policy file, provide a reference to a backend, and mount the relevant file system Volumes.

- Make sure to practice all of those aspects hands-on. Configuring audit logging can take a little bit of time. Budget for it during the exam.


O'REILLY®