

O'REILLY®

Certified Kubernetes Security Associate (KCSA) Crash Course

Curriculum February 2024



About the Trainer

Social media handles and web pages



bmuschko



bmuschko



bmuschko.com



automatedascent.com

Exam Objectives and Overview





Kubernetes Certification Path

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



Associate



Developer



Administrator

Entry Level

Practitioner

Practitioner

Exam Objectives

"Understand baseline security aspects of Kubernetes and cloud-native apps"

- The [certification program](#) allows users to demonstrate their competence as part of a multiple-choice test.
- Test takers **do not** need to solve problems on a real Kubernetes cluster.
- Documentation is not going to be available during the exam.
- While you may think that this is a beginner-level certification from the name, it is **not**. Advanced knowledge is required to pass the exam.





Exam Domains

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

DOMAIN	WEIGHT
Overview of Cloud Native Security	14%
Kubernetes Cluster Component Security	22%
Kubernetes Security Fundamentals	22%
Kubernetes Threat Model	16%
Platform Security	16%
Compliance and Security Frameworks	10%



Curriculum Details

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

14% - Overview of Cloud Native Security

- The 4Cs of Cloud Native Security
- Cloud Provider and Infrastructure Security
- Controls and Frameworks
- Isolation Techniques
- Artifact Repository and Image Security
- Workload and Application Code Security

22% - Kubernetes Security Fundamentals

- Pod Security Standards
- Pod Security Admissions
- Authentication
- Secrets
- Isolation and Segmentation
- Audit Logging
- Network Policy

16% - Platform Security

- Supply Chain Security
- Image Repository
- Observability
- Service Mesh
- PKI
- Connectivity
- Admission Control

22% - Kubernetes Cluster Component Security

- API Server
- Controller Manager
- Scheduler
- Kubelet
- Container Runtime
- KubeProxy
- Pod
- Etcd
- Container Networking
- Client Security
- Storage

16% - Kubernetes Threat Model

- Kubernetes Trust Boundaries and Data Flow
- Persistence
- Denial of Service
- Malicious Code Execution and Compromised Applications in Containers
- Attacker on the Network
- Access to Sensitive Data
- Privilege Escalation

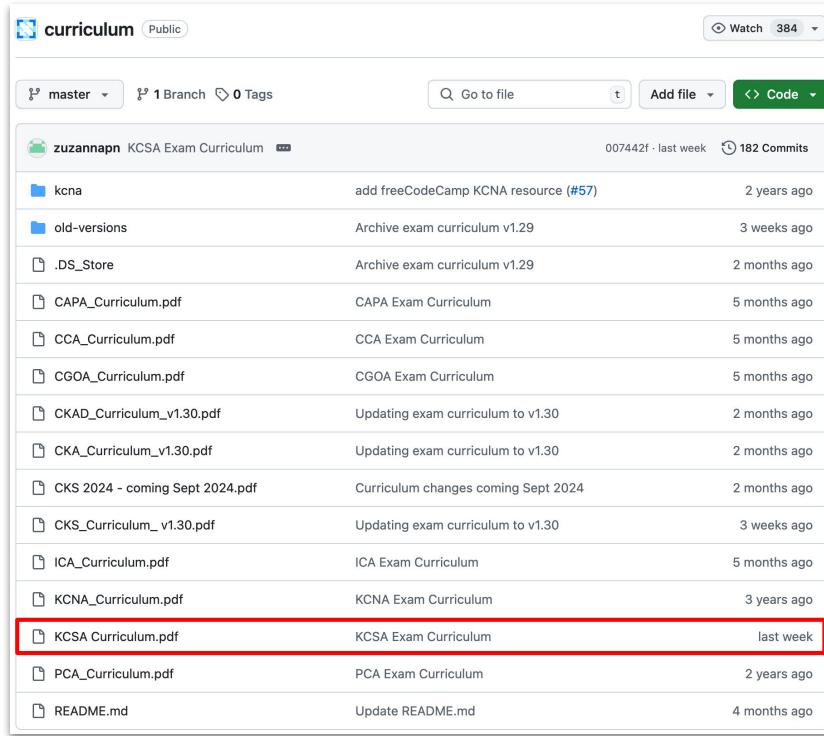
10% - Compliance and Security Frameworks

- Compliance Frameworks
- Threat Modeling Frameworks
- Supply Chain Compliance
- Automation and Tooling

Covered along the way as scenarios and examples

Kubernetes Version Used in Exam

The curriculum file name does not indicate version, assume latest



A screenshot of a GitHub repository page for 'curriculum'. The repository is public and has 384 watchers. It contains 1 branch and 0 tags. The master branch has 182 commits. The repository owner is zuzannapn. The files listed are:

File	Description	Last Commit
kcna	add freeCodeCamp KCNA resource (#57)	2 years ago
old-versions	Archive exam curriculum v1.29	3 weeks ago
.DS_Store	Archive exam curriculum v1.29	2 months ago
CAPA_Curriculum.pdf	CAPA Exam Curriculum	5 months ago
CCA_Curriculum.pdf	CCA Exam Curriculum	5 months ago
CGOA_Curriculum.pdf	CGOA Exam Curriculum	5 months ago
CKAD_Curriculum_v1.30.pdf	Updating exam curriculum to v1.30	2 months ago
CKA_Curriculum_v1.30.pdf	Updating exam curriculum to v1.30	2 months ago
CKS 2024 - coming Sept 2024.pdf	Curriculum changes coming Sept 2024	2 months ago
CKS_Curriculum_v1.30.pdf	Updating exam curriculum to v1.30	3 weeks ago
ICA_Curriculum.pdf	ICA Exam Curriculum	5 months ago
KCNA_Curriculum.pdf	KCNA Exam Curriculum	3 years ago
KCSA Curriculum.pdf	KCSA Exam Curriculum	last week
PCA_Curriculum.pdf	PCA Exam Curriculum	2 years ago
README.md	Update README.md	4 months ago

Exam Environment

PSI Remote desktop, single monitor, no bookmarks ([announcement](#))

The screenshot shows the PSI Secure Browser 2.4.10 interface during an exam session. At the top, there are navigation menus (File, View), a PSI logo, and buttons for REQUEST BREAK and END PROCTORING SESSION. On the right side of the header, there are icons for LIVE CHAT, zoom controls (144%), and a video feed window. Below the header, a blue bar displays the text "PSI". The main content area shows "Question: 2 of 10" and a question: "What is the highest mountain in the world?". Four options are listed with radio buttons: Mount Everest, Alaska Range, Smoky Mountains, and Adironback Mountains. In the top right corner of the content area, there is a red box highlighting the "Exam Timer" which shows "00:18:10" and "English". In the bottom right corner, there is another red box highlighting a "Flag for review" button with the sub-instruction "Click to Flag this item for review". At the very bottom of the browser window, there is a red box highlighting a navigation bar with "Previous" and "Next" buttons.

PSI Secure Browser 2.4.10

REQUEST BREAK END PROCTORING SESSION

LIVE CHAT - + 144%

PSI

Question: 2 of 10

What is the highest mountain in the world?

- Mount Everest
- Alaska Range
- Smoky Mountains
- Adironback Mountains

00:18:10 English

Exam Timer

Flag for review

Click to Flag this item for review

Navigate between items by clicking Previous or Next

← Previous Next →



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

Check out the content first before committing

The screenshot shows a web browser window for learning.oreilly.com. The page title is "O'REILLY®". Below it, the tagline "Build Skills. Solve Problems. Stay Ahead." is displayed. A "Activate Your Membership" section contains fields for "First name", "Last name", "Email", "Create a password", and a "Promo code" field which contains "MUSCHKO24". Below these fields is a dropdown menu set to "Country: USA". At the bottom of the form are two buttons: a red "Activate Account" button and a smaller "Already have an account? [Sign In](#)" link. A small legal notice at the bottom states: "By clicking 'Activate Account,' you confirm that you have read and agree to the terms and conditions of our [Membership Agreement](#) and that when your complimentary membership ends, you will be required to provide billing information if you wish to continue using this service."



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 \$250 | Online Exam

REGISTER FOR EXAM

Get a 20% with code
ASCENT20 at checkout

Disclosure: This is an affiliate code.



Preparing for the Exam

Supplement theory with practice

- Read through online documentation for relevant topics
- Apply knowledge by getting hands-on with Kubernetes
- Pick the time you are most comfortable with, get enough sleep
- Take it easy on your first attempt, but give it your best effort

Overview of Cloud Native Security





Topics We'll Cover

Cloud-native security features

- Overview
- Cloud providers and on-premises
- Established best practices

Isolation techniques

- Cloud in general
- Kubernetes-specific

4Cs of cloud-native security

- The 4 layers
- Security practices for each layer

Application/container image security

- Security aspects for code
- Security aspects for container images
- Artifact repositories
- Workload security



Cloud Provider and Infrastructure Security

Definition and outlook



Cloud Provider Security

Many security features and tools are built-in

- Cloud providers like Amazon, Google, and Azure offer Kubernetes hosted in the cloud (EKS, GCP, and AKS).
- Each of those cloud providers offer security features.
 - Runtime monitoring, e.g. Azure Security Center
 - Identity and access management, e.g. Amazon IAM
 - Container registry, e.g. Google Cloud Registry
- Many cloud providers offer service API endpoints. In most cases, you'll likely not want to expose those endpoints to the internet.

Infrastructure Security

Guidelines for securing cluster nodes and components

- **Network access to the API server:** Only allow clients to interact with the cluster from within your company network. Set up firewall rules to prevent unauthorized access.
- **Network access to cluster nodes:** Control plane and worker nodes should not be accessible from the public internet. Only those ports should be open that are needed to operate the cluster plus NodePort and LoadBalancer ports.
- **Etcd encryption:** Data-at-rest should not be stored in plain text so that compromised data cannot be read.
- **Access to etcd:** Only the control plane node(s) should be able to access etcd.



Controls and Frameworks

Codified security best practices

- **Controls:** Codify security best practices developed by researchers and domain expert. Measure to be to be taken or avoid to prevent a security breach.
- **Frameworks:** A set of security guidelines, best practices, or standards. Fairly often, tools have been developed to execute the controls in an automated manner. An example of a framework in the Kubernetes space is the [CIS Kubernetes Benchmarks](#).

Common Security Frameworks

A wide range of options published by research organizations

- [Center for Internet Security \(CIS\)](#) Kubernetes Benchmark: A comprehensive guide for hardening Kubernetes environments.
- [MITRE ATT&CK framework](#): A comprehensive knowledge base built on various hacking mechanisms and tactics based on real-world events.
- [PCI DSS Compliance for Kubernetes](#): A compliance framework outlines the technical and operational requirements to enable security and data protection for the payment industry.
- [NIST Application Container Security Framework](#): A risk management framework for containers and containerized environments.

Security Framework Comparison

Automated tools can help with finding misconfiguration and security issues

Security Frameworks					
	CIS	MITRE	PCI DSS	NIST	NSA-CISA
GOAL	Provide configuration benchmarks	Enumerate all real-world attack tactics, techniques, and procedures	Protect cardholder data	Issue guidelines for a risk management framework	Enable the hardening of Kubernetes Cluster
WHEN TO USE	Pre-production	In production	Development, testing, and production	Testing and vulnerability scanning	Pre-production
FOCUS AREAS	Auditing configuration management	Threat modeling	Access control, vulnerability management	Risk management	Configuration management
HOW TO USE	Compare CIS standards with system configuration	Use the prescribed techniques for pen tests and threat management	Use guidelines to create policies for data protection	Use the RMF to establish a baseline for security posture	Use example configurations of hardened clusters
TOOLS	Kubescape	Kubescape	Twistlock	NIST SRE Tools	Kubescape

Source: <https://www.armosec.io/blog/kubernetes-security-frameworks-and-guidance/>

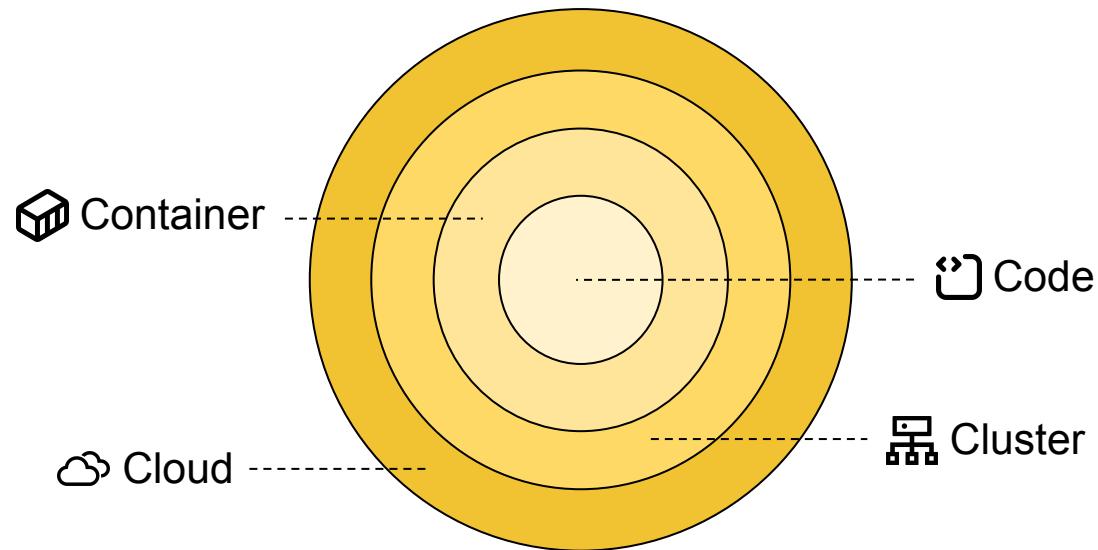


The 4Cs of Cloud Native Security

Cloud, clusters, containers, and code

The 4Cs

Each Layer is shielded from the others, and provides unique security capabilities





The Cloud Layer

Layer includes on-prem and hybrid environments

- The trusted computing base of your Kubernetes cluster.
- If the cloud layer becomes vulnerable, no guarantees can be made about the security of components running on top of this layer.
- Default cloud settings may expose your cluster to outside world.



Cloud Layer Security Best Practices

Requires experts to configure due to its complexity and variability

- Follow security best practices published by cloud providers to take security measures.
- **Networking:** Configure firewall rules, port access, and encryption.
- **Principle of least privilege:** Only provide access to human users and processes that need to have access. Deny access to anyone/anything else.

The Cluster Layer

Cluster components and networking

- Each cluster consists of control plane nodes and worker nodes.
- Control plane nodes and worker nodes run dedicated cluster components (e.g. API server, etcd).
- Secure cluster component settings are well documented (e.g. CIS benchmarks).
- Pod-to-Pod communication can be restricted (network policies) and encrypted (mTLS).



Cluster Layer Security Best Practices

Typically the responsibility of the Kubernetes administrator

- Verify and adjust secure configuration of cluster components according to [CIS benchmark](#) criteria.
- **Authorization:** Only allow stakeholders permissions to cluster operations that they actually need via Role-based access control (RBAC).
- **Encryption:** Encrypt network communication to API server, and between Pods. Encrypt data at rest in etcd (e.g. Secret data).
- **Networking:** Close unnecessary ports.



The Container Layer

Containerized workload executed in Kubernetes

- The central Kubernetes resource for executing processes and applications in a container is the Pod.
- A Pod can run a single container (the default) or multiple containers if you need helper functionality.
- The container image should follow best practices when built. Pods can be configured to apply security measures.



Container Layer Best Practices

Primary responsibility of application developers with governance of admins

- Every container image is built from a base image. The smaller the image and the less functionality included, the smaller the potential security risk. Distroless container image do not provide a shell. Sign container images to ensure their authenticity and integrity.
- Automated tools, e.g. [Trivy](#), [Snyk](#), can help find known security vulnerabilities before they are consumed. Use them as part of your development and automated delivery process.
- Configure Pods with security in mind. For example, do not run the container with the root user, make container immutable (configurable via [security context](#)), and govern the creation of Pods via [Pod Security Admission](#) rules.



The Code Layer

Application implementation written by developers

- Every program starts with the source code used to build it.
- Business application code usually incorporates dependencies in the form of reusable libraries, or external services.
- On the code layer, security issues can arise due to vulnerabilities in dependencies, leaking credentials, or insecure coding practices that allow cross-site scripting or SQL injection.

Code Layer Best Practices

Responsibility of application developers and their team

- Involve and enforce a code review process (e.g. as part of a pull request) by experienced peers to identify and correct poor coding practices.
- Employ security scanning tools incorporated in automated CI process to detect vulnerabilities.
- Manual QA process: Try to find security holes that feel through the cracks via acceptance testing by humans.



Isolation Techniques

Categories supported by Kubernetes



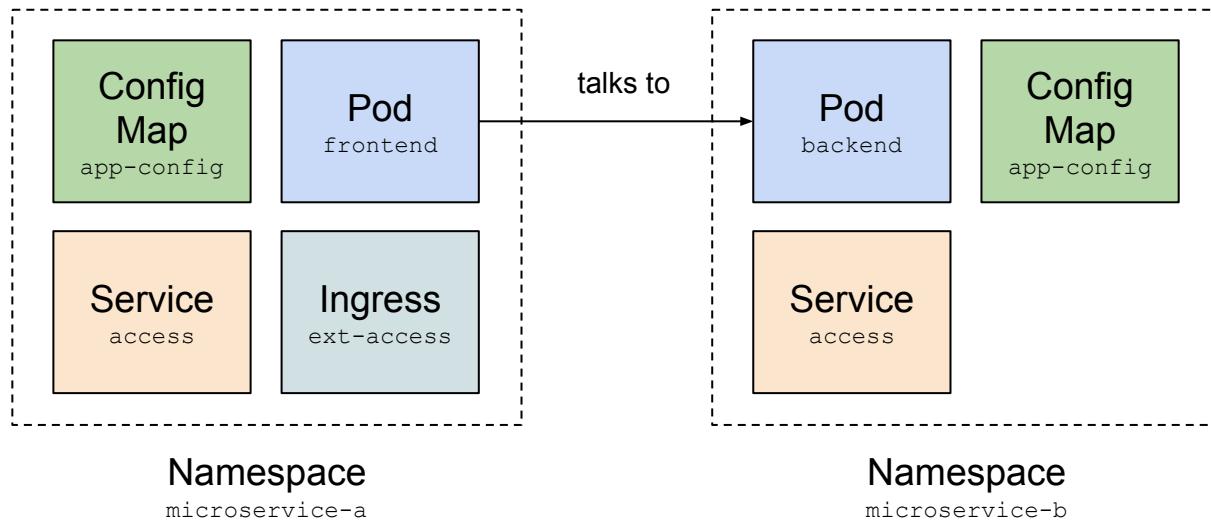
Kubernetes Isolation Techniques

Built-in capabilities

Type	Description
Namespace	Logical separation of objects into named buckets
Role-based access control (RBAC)	Authorization concerns - “Who’s allowed to do what within the cluster?”
Network Policies	“Firewall rules” for Pod-to-Pod communication
Policy Enforcement	Enforces security settings for data sent with an API request via core Kubernetes concepts or Kubernetes extension mechanism

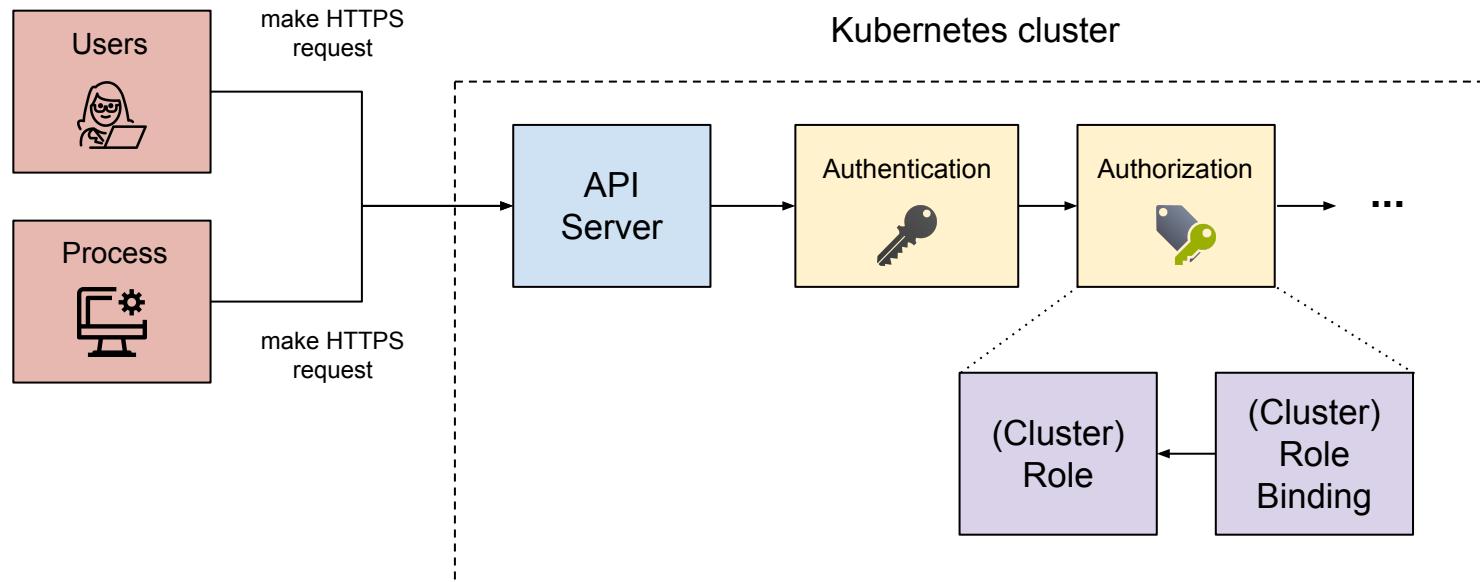
Namespace

Isolates Kubernetes objects logically



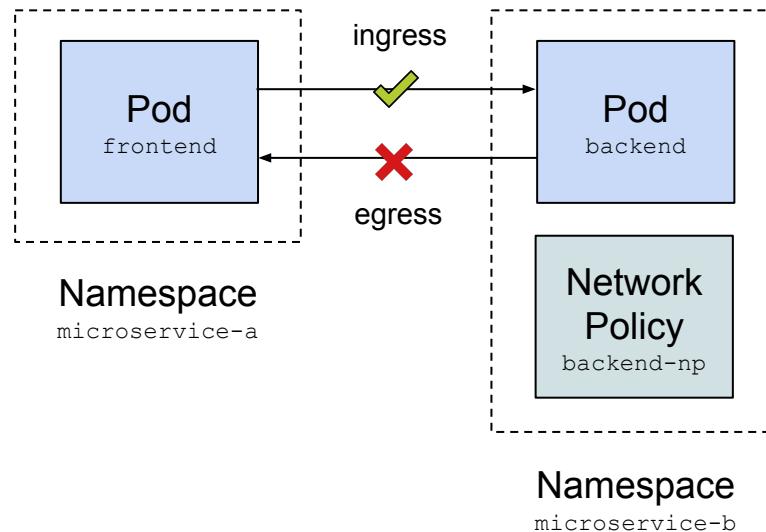
Role-Based Access Control (RBAC)

Defines permissions for authenticated users and processes



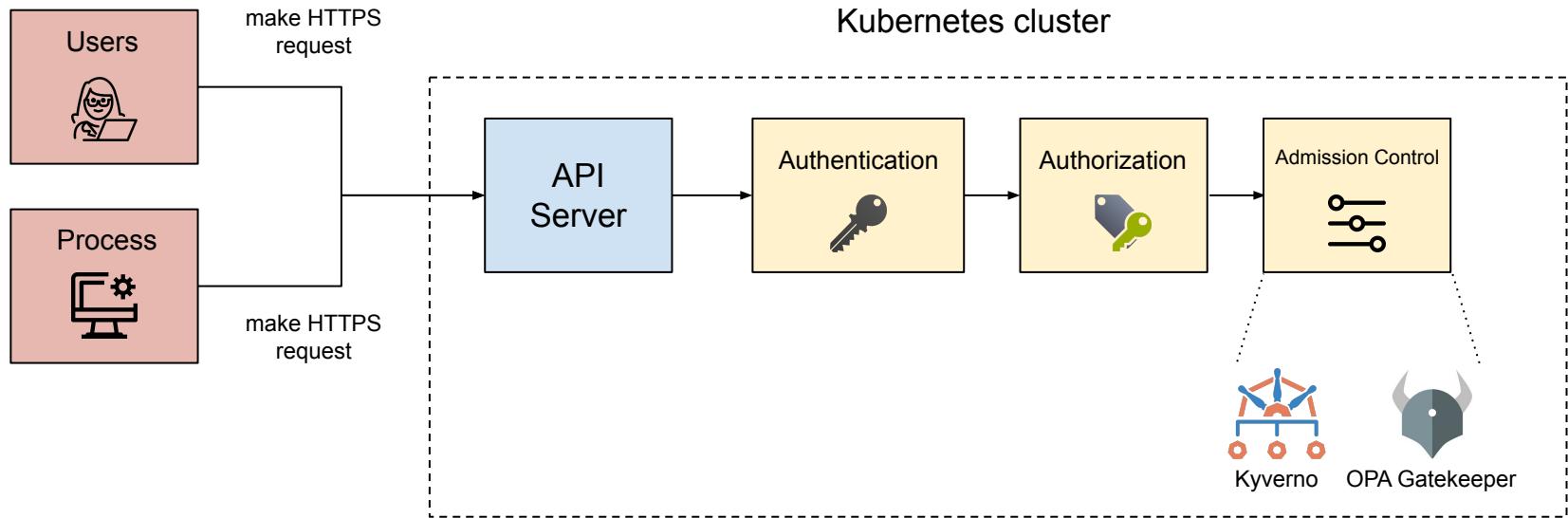
Network Policies

By default, Pod-to-Pod network communication is not restricted



Policy Enforcement

Admission Controllers can reject or manipulate configuration sent with request





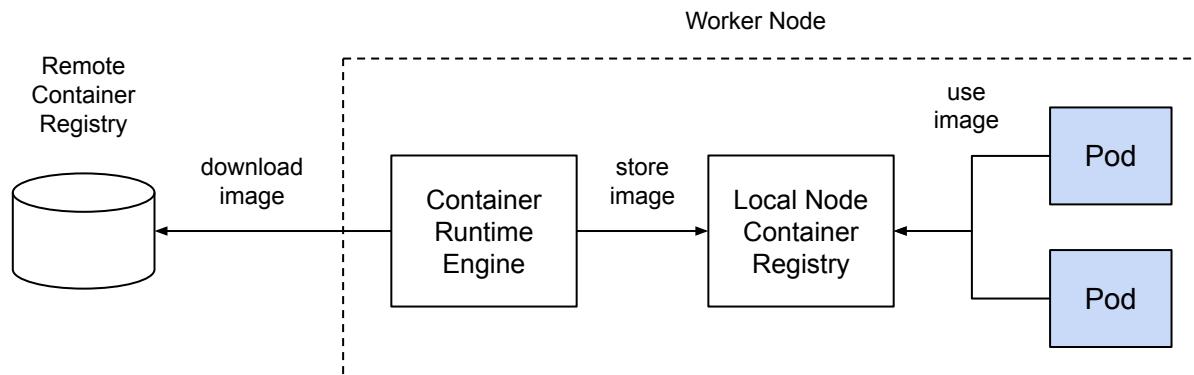
Artifact Repository and Image Security

Controlling container image storage and vulnerability scanning

Artifact Repositories

Remote container image storage location

- Application developers build container images and share them for consumption in an artifact repository.
- Examples for artifact repositories are Docker Hub, Google Cloud Registry (GCR), Amazon Elastic Container Registry (ECR), and JFrog Artifactory.





Using Private Artifact Repositories

Do not use public artifact repositories to safeguard from potential security risks

- To ensure that container images aren't tempered with, you should host a private artifact repository for your company-internal container images.
- You can configure Kubernetes to only pull images from private artifact repositories. Tools like Kyverno and OPA Gatekeeper can further govern the behavior.



Container Image Security

Consumers shouldn't blindly use container images

- Container images, their base image, and used dependencies can contain security vulnerabilities.
- Commercial artifact repositories provide built-in feature for scanning container images to detect known vulnerabilities published in [CVE database](#).
- You can prevent consumers from pulling container images containing detected vulnerabilities based on CVSS score.



Workload and Application Code Security

Runtime and code security

Workload vs. Application Code Security

Runtime security and the code executed by the runtime

- **Workload security:** Configuring and maintaining monitoring, logging, and networking tools to ensure workload security. For example, you will want to enable audit logging in your cluster, and employ monitoring tools like Falco to watch for abnormal behavior, potential security threats, and compliance violations in containers by inspecting kernel events.
- **Application code security:** This security concern is less about Kubernetes itself. It focuses on security best practices used by the code itself. You will want to employ automated tools scanning for issues as part of your delivery process even before the container image is built and beyond.



Application Code Security

Automated scanning tools can help a lot

- **Scan the application code:** Use code reviews and run automated security tools like [SonarQube](#).
- **Scan the container image:** Run automated tools like [JFrog XRay](#) to find vulnerabilities and configuration issues.
- **Scan Pod configuration:** Ensure that containers stick to principle of least privilege via tools by inspecting manifests via tools like [Kubesec](#).
- **Only allow required operations:** Pods running processes that need to interact with the Kubernetes API need to be restricted by RBAC bound to the assigned ServiceAccount.

Exam Essentials

What to focus on for the exam

- Know the 4Cs of cloud-native security and what they entail. Most importantly, understand that the different layers build on top of each other.
- Understand Kubernetes isolation techniques on a technical level. You should not stop at concept basics. You will need to know how to apply it. That means, knowing spec details of a Role/RoleBinding for RBAC for example.
- Become familiar with some container image security best practices
 - Techniques for building container images (e.g. choosing a small base image, multi-stage builds)
 - Container image scanning for vulnerabilities
 - Consumption of container image from private registries

Kubernetes Cluster Component Security





Topics We'll Cover

High-level architecture

- Overview
- Node types
- Cluster components

Securing cluster components

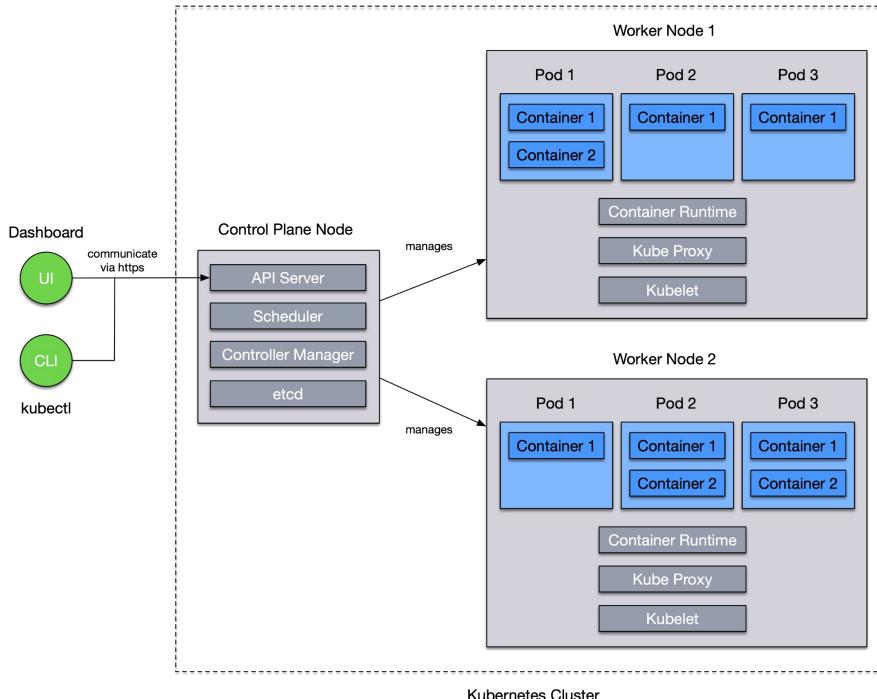
- Responsibility of components
- How to secure the components

Securing workload

- Pod security settings
- Container networking
- Storage security

Kubernetes Architecture

Cluster components in control plane node and worker nodes



Scanning Cluster Component Configuration

Use automation tools to identify deficiencies and fix them

- Research organization publish recommendations for cluster configuration. The most prominent example is the [CIS Kubernetes benchmark](#).
- You can use the tool [kube-bench](#) to check Kubernetes cluster components against the CIS Benchmark best practices in an automated fashion.

```
[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)
[PASS] 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 --kubernetes-certificate-authority argument is set as appropriate (Scored)
```

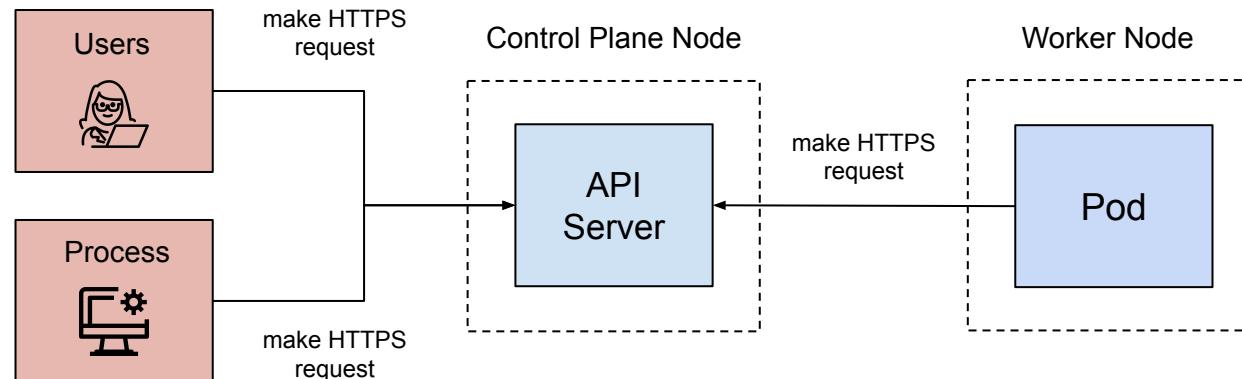


API Server

Securing the cluster component

API Server

Exposes the API endpoints clients use to communicate with cluster



Securing the API Server

Configurable in dedicated configuration file

- The `kube-apiserver` process running in a Pod allows for configuring a variety of security-related settings.
- Configurable through the file
`/etc/kubernetes/manifests/kube-apiserver.yaml` on control plane node.
- **Configuration example:** “Make sure that anonymous requests to the secure port of the API server are disabled by setting `--anonymous-auth=false`.”
- Automatically restarts the API server Pod in the `kube-system` namespaces after making a change to configuration file.



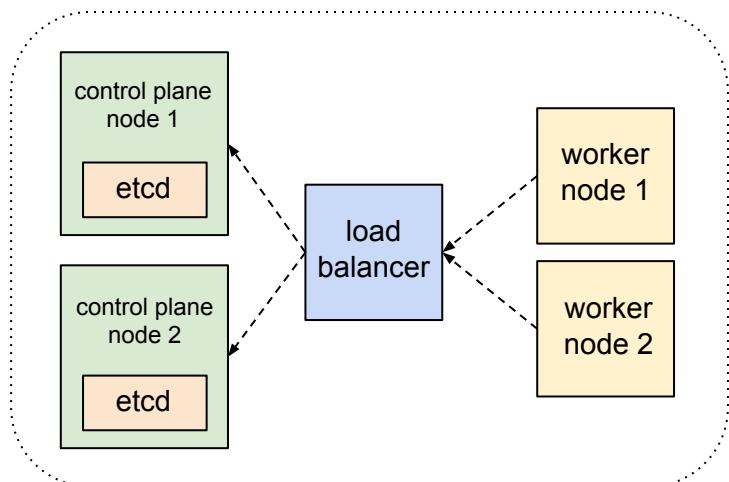
Etcd

Securing the cluster component

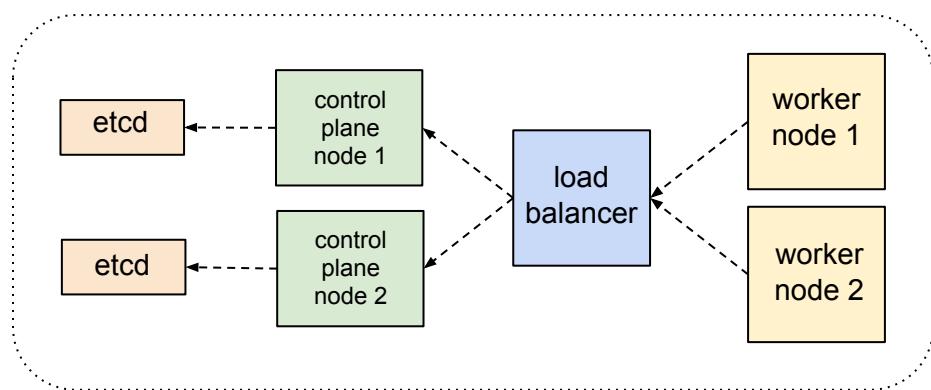
Etcd

Persists the cluster state in plain-text

Stacked etcd topology



External etcd topology



[Detailed installation instructions](#)



Securing Etcd

Configurable in dedicated configuration file

- The `etcd` process allows for configuring a variety of security-related settings.
- Configurable through the file `/etc/kubernetes/manifests/etcd.yaml` on the host that runs etcd.
- **Configuration example:** “Ensure TLS communication with flags like `--cert-file`, `--key-file`, `--client-cert-auth`, and `--trusted-ca-file`”



Encrypting and Backing Up Etcd Data

Stored in plain-text by default

- Sensitive data like Secrets stored in etcd should be encrypted. You'd do so by assigning the encryption configuration via the API server configuration flag
`--encryption-provider-config`
- Alternatively, you can integrate external secrets manager like HashiCorp Vault or AWS Secrets Manager using the External Secrets Operator.
- Make sure to back up etcd periodically in case the data gets compromised or corrupted. One way to do this is with the help of etcdctl.

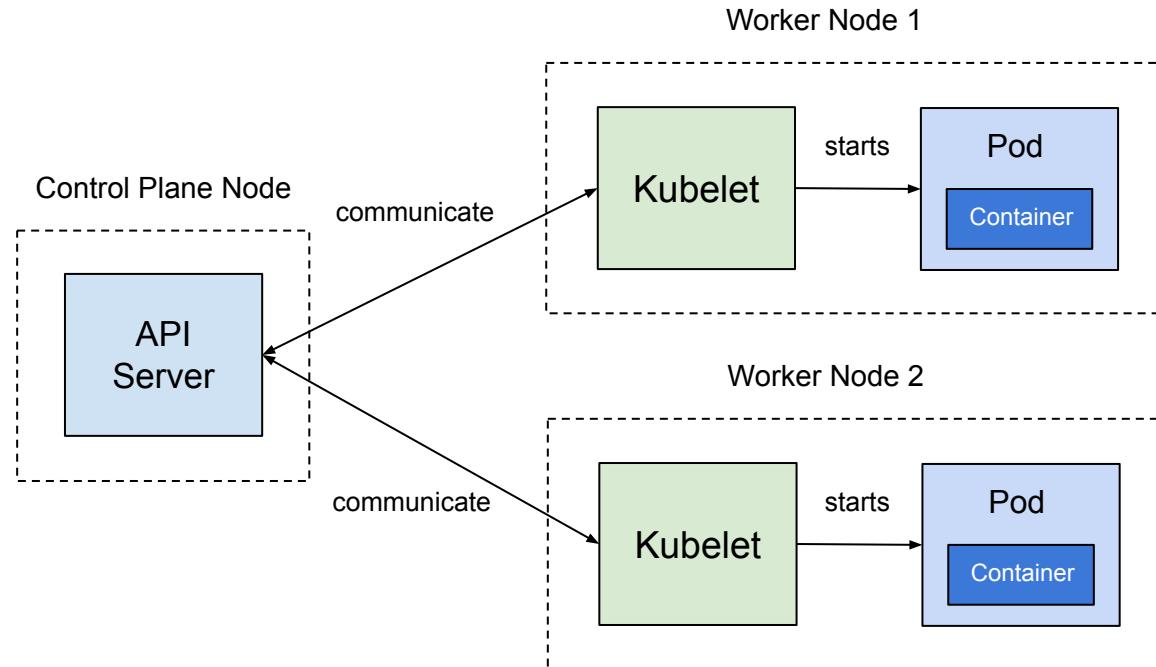


Kubelet

Securing the cluster component

Kubelet

Manages and coordinates Pods and nodes





Securing the Kubelet

Configurable in dedicated configuration file

- The `kubelet` process allows for configuring a variety of security-related settings.
- Configurable through the file `/var/lib/kubelet/config.yaml` on the node that runs a kubelet Pod.
- **Configuration example:** “Set `--authorization-mode` to a value other than `AlwaysAllow` to verify that requests are authorized.”

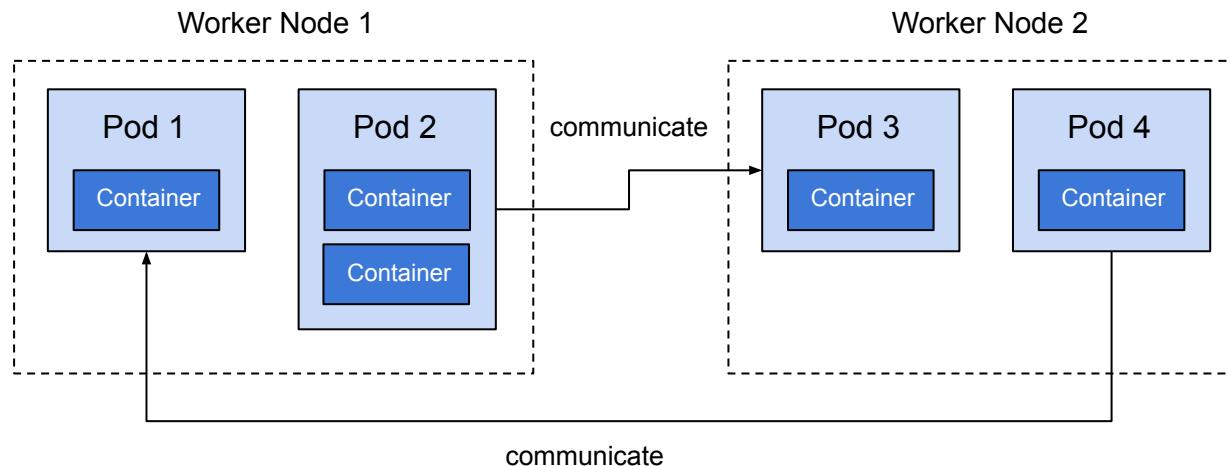


Workload

Securing Pods, network communication, and storage

Pods

Runs an application or process inside of one or many containers



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.”

Pod-Level Security Context

pod.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: sample-pod
spec:
  securityContext:
    runAsUser: 1000
    fsGroup: 4000
  containers:
    - name: nginx
      image: nginx:1.25.3
    - name: busybox
      image: busybox:1.36.1
```

Applies security settings to all containers

Container-Level Security Context

pod.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: sample-pod
spec:
  containers:
    - name: nginx
      image: nginx:1.25.3
      securityContext:
        runAsUser: 1000
    - name: busybox
      image: busybox:1.36.1
      securityContext:
        runAsUser: 4000
```

Applies security settings to individual containers



Securing Pod-to-Pod Communication

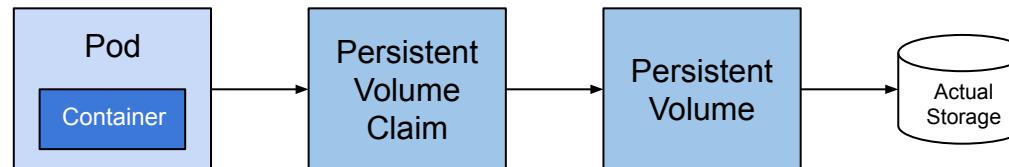
CNI assigns virtual IP address to Pods and doesn't restrict communication

- A Network Policy defines ingress and egress traffic for Pods. You can also restrict access to ports with Network Policy rules.
- **Configuration example:** “The frontend Pod should be allowed to talk to the backend Pod on port 8080. All other network traffic should be denied.”

Securing Storage

Volumes can be mounted to containers to provide persistent data storage

- The Persistent Volume is a primitive that allows for storing data beyond a Pod, node, or even cluster restart.
- Data is usually stored unencrypted. Make sure to back up data in case you need to restore it. Encrypt the data to prevent intruders from reading sensitive information.





Exam Essentials

What to focus on for the exam

- Understand the role of each cluster component and know how to configure them. I would recommend reading through the configuration options as well.
- Certain cluster components need to talk to each other. You will have to understand how to secure the network communication between them.
- Dive deeper into how etcd stores data and how to configure encryption of data-at-rest.
- Learn about Pod spec security options like the security context, as well as privilege mode, and access to the host namespace.

Kubernetes Security Fundamentals





Topics We'll Cover

Pod Security Standards and Admission

- Overview
- Policy profiles
- Governing Pod security settings

Secrets and Network Policies

- Defining and consuming sensitive configuration data
- Restricting Pod-to-Pod communication

Authentication and Authorization

- Overview
- Authentication with `kubectl`
- High-level RBAC

Audit Logging

- Information captured by audit logging
- Configuring audit logging
- Example audit logging output

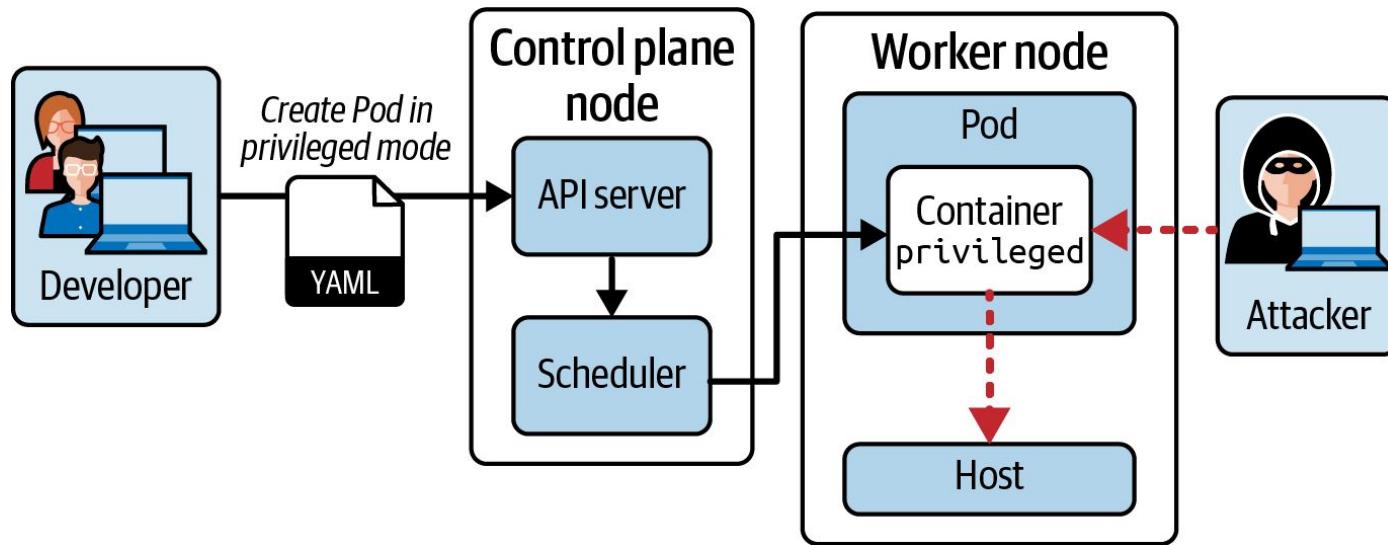


Pod Security Standards and Admission

Governing Pod security settings

Why Pod Security Standards?

Governing poor security practices in Pods



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.

```
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

```
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.
```

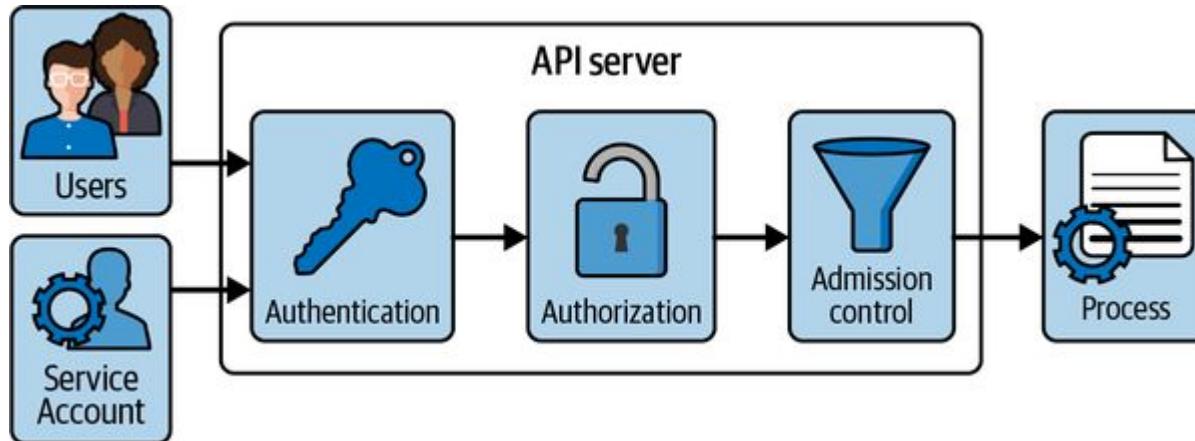


Authentication and Authorization

Processing a request to the Kubernetes API

Processing a Request to the Kubernetes API

Multi-phases approach which includes authentication and authorization



Why Authentication and Authorization?

Prevent authorized access to cluster operations

- **Authentication:** Authentication validates the identity of the caller by inspecting the client certificates or bearer tokens. If the bearer token is associated with a service account, then it will be verified here.

Prevent anonymous requests and only allow identities with proper credentials.

- **Authorization:** Verifies if authenticated client should be allowed to execute requested operation via RBAC. For example, is the request allowed to list Pods or create a new Service object.

Do not allow any authenticated client to run all operations.

Authentication

User management + credentials

- User management is handled by the cluster administrator. The administrator creates a user representing the developer and hands the relevant information (username and credentials) to the human wanting to interact with the cluster via `kubectl`.
- Alternatively, it is also possible to integrate with external identity providers for authentication purposes, e.g., via OpenID Connect.



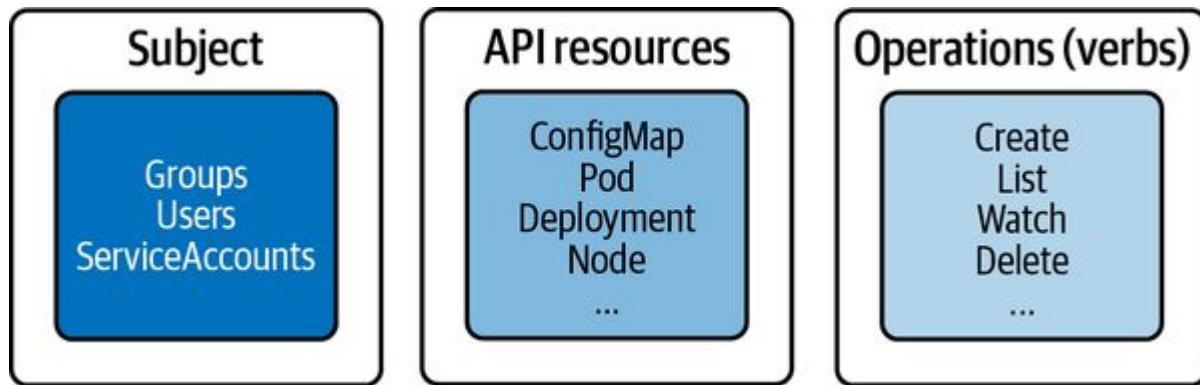
Authenticating with kubectl

Command line tool automatically sends credentials with every request

- Whenever you execute a command with `kubectl`, the underlying HTTPS call to the API server needs to authenticate.
- Credentials for the use of `kubectl` are stored in the file `$HOME/.kube/config`, also known as the `kubeconfig` file.
- `Kubectl` uses the currently selected context to know which cluster to talk to and which credentials to use.

RBAC Overview

Three key building blocks for understanding the concept



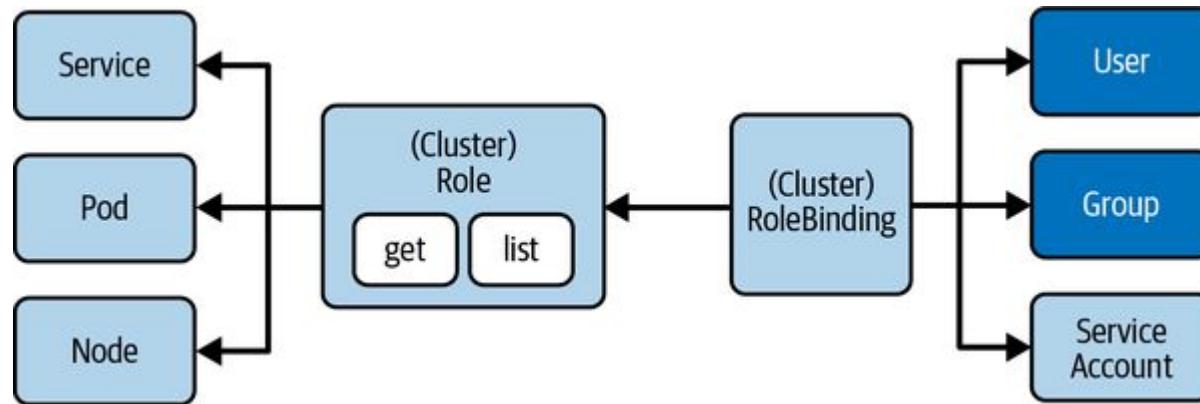
RBAC API Primitives

Role and RoleBinding

- **Role:** The Role API primitive declares the API resources and their operations this rule should operate on in a specific namespace. For example, you may want to say “allow listing and deleting of Pods,” or you may express “allow watching the logs of Pods,” or even both with the same Role. Any operation that is not spelled out explicitly is disallowed as soon as it is bound to the subject.
- **RoleBinding:** The RoleBinding API primitive binds the Role object to the subject(s) in a specific namespace. It is the glue for making the rules active. For example, you may want to say “bind the Role that permits updating Services to the user John Doe.”

Relationships between RBAC API Primitives

A Role is bound by a RoleBinding



Role YAML Manifest

Can define a list of rules in an array

```
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)

RoleBinding YAML Manifest

Roles can be mapped to multiple subjects if needed

```
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
```

← Reference to Role

← Reference to User

Working with Service Accounts

Service Accounts by default have very little permissions

- Some applications running inside of a Pod may have to communicate with the API server as well. For example, the application may ask for specific cluster node information or available namespaces.
- Pods can use a service account to authenticate with the API server through an authentication token. A Kubernetes administrator assigns rules to a service account via RBAC to authorize access to specific resources and actions.



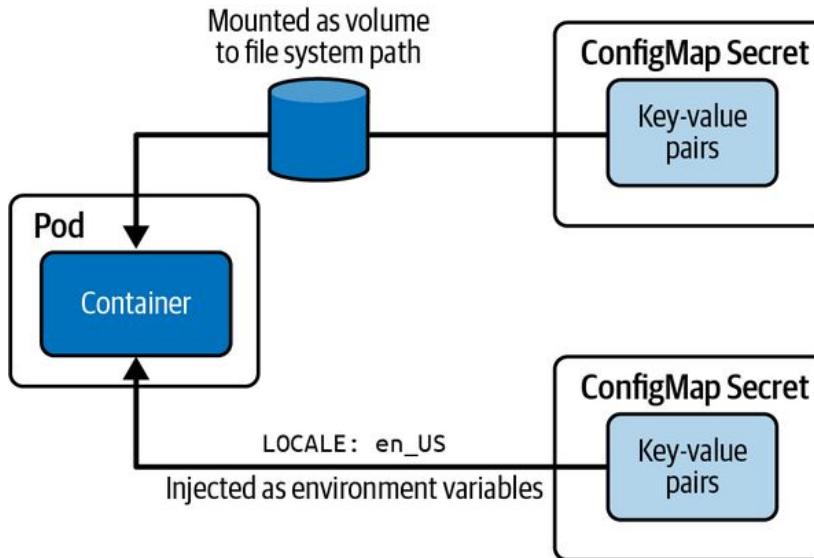


Secrets

Defining and consuming sensitive configuration data

Why do Pods need Sensitive Configuration Data?

Controlling runtime behavior for applications running in the container



Secrets

A Secret is meant for storing sensitive data

- The Secret is a Kubernetes primitive for storing key-value pairs with a lifecycle decoupled from the consuming Pod.
- Syntactically meant storing sensitive data like passwords, API keys, or SSL certificates
- Values are only base64-encoded values, not encrypted!

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
```

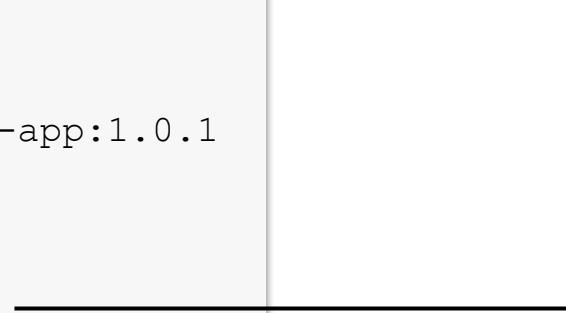


Consuming a Secret as Environment Variables

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!
```



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:
        bmuschnko/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
data:
  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](#).

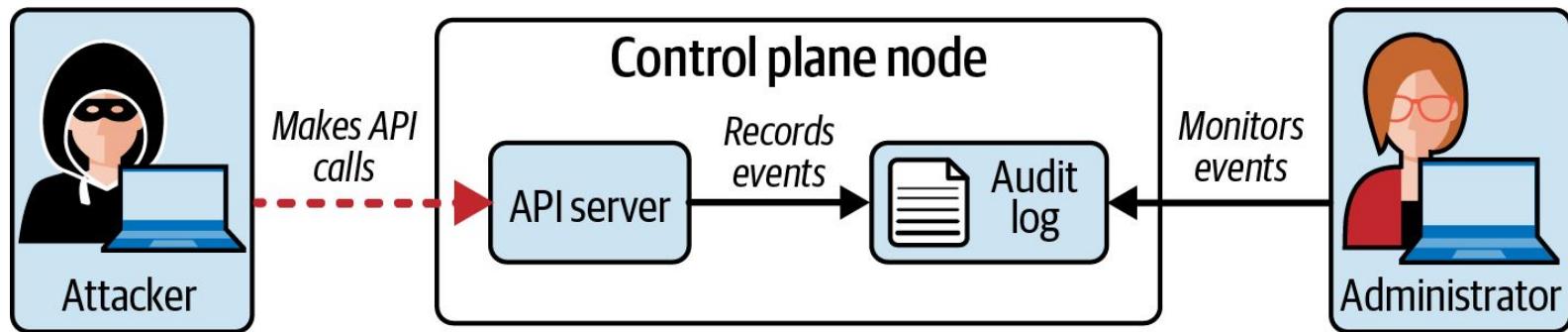


Audit Logging

Logging API server events

Why Audit Logging?

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?

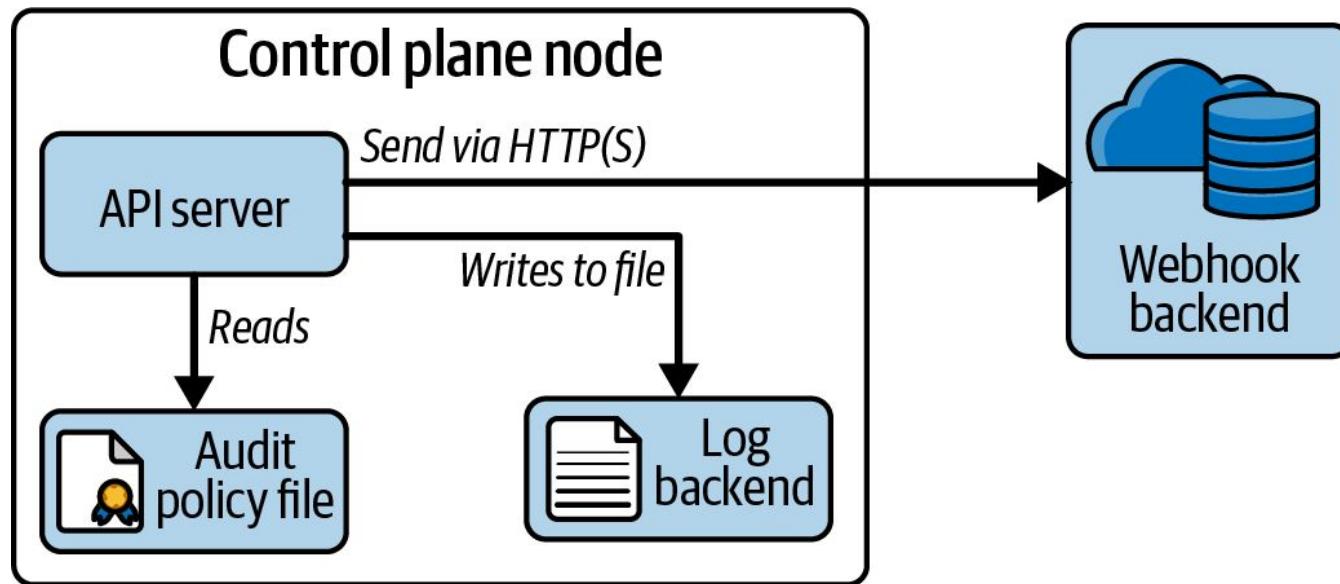
Audit Logging Terminology

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 writes 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", \
"...
```

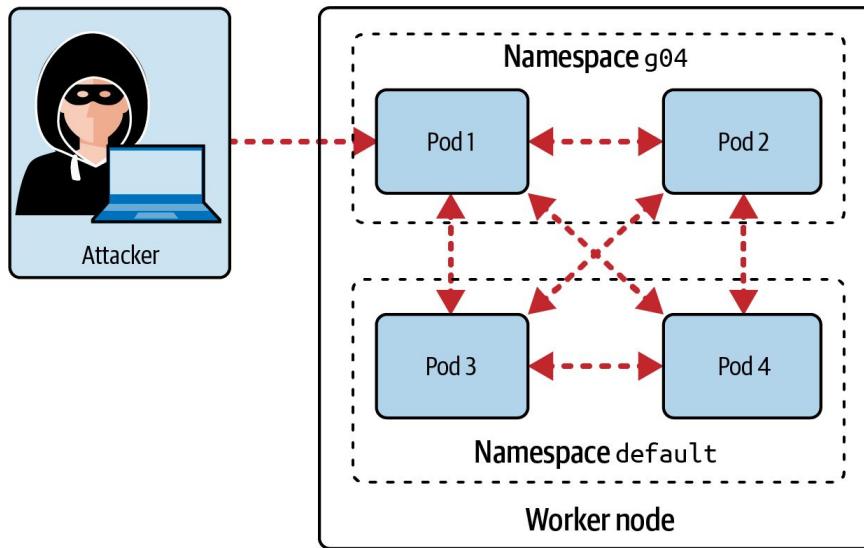


Network Policies

Restricting Pod-to-Pod communication

Why Network Policies?

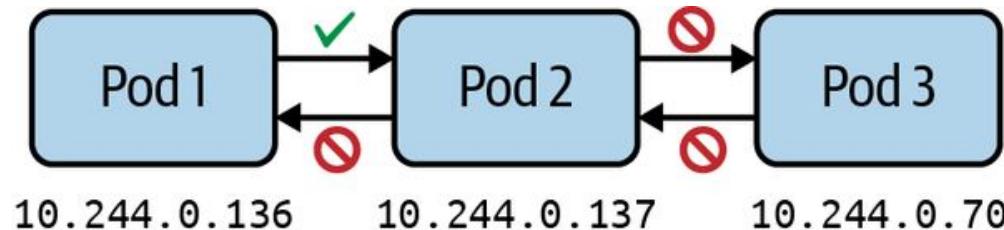
Unrestricted Pod-to-Pod communication opens new attack vectors for intruders



Pod-to-Pod Communication

Microservices running in Pods need to be able to communicate

- Pod-to-Pod communication is unrestricted by default, even across namespaces. Any Pod can talk to any other Pod in the cluster via the virtual IP address.
- The Network Policy is a Kubernetes primitive for restricting the network communication.



General Principles of Network Policies

Restrict as much as possible, only open up necessary communication

- You will want to start by establishing the principle of least privilege. That means instantiating a network policy that disallows all communication.
- Network policies are additive. You will want to open up network communication between Pods based on your communication requirements. For example, you may want that a frontend microservice can talk to a backend microservice.

Installing an Network Policy Controller

A network policy cannot work without a network policy controller

- The network policy controller evaluates the collection of rules defined by a network policy. You can find instructions for a wide range of network policy controllers in the [Kubernetes documentation](#).
- [Cilium](#) is a CNI that implements a network policy controller. You can install Cilium on cloud provider and on-prem Kubernetes clusters. Refer to the [installation instructions](#) for detailed information.

Denying Directional Traffic

The Kubernetes documentation offers a list of [default policies](#)

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny-ingress
  namespace: g04
spec:
  podSelector: {}
  policyTypes:
  - Ingress
```

Opening up Directional Traffic

You can define ingress and egress traffic, as well as ports

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: backend-ingress
  namespace: g04
spec:
  podSelector:
    matchLabels:
      tier: backend
  policyTypes:
  - Ingress
  ingress:
  - from:
    - podSelector:
        matchLabels:
          tier: frontend
  ports:
  - protocol: TCP
    port: 3000
```

Exam Essentials

What to focus on for the exam

- Understand the following Kubernetes primitives and their spec in detail: Namespace, Pod, NetworkPolicy, Secret, Role, RoleBinding. It's not enough to know responsibilities of those primitives.
- Deepen your knowledge by understanding use cases and scenarios. For example, "What's the impact of using base64-encoding in a Secret for security?"
- Audit Logging is a cluster-level configuration option. Learn about its purpose and its configuration options.

Platform Security





Topics We'll Cover

Supply Chain Security

- Scenarios
- Tools and approaches

Service Mesh and PKI

- Drivers for using a service mesh
- Service mesh implementation
- Usage of PKI in Kubernetes

Observability

- Overview
- A quick tour on common tooling

Admission Control

- The purpose of admission control
- Admission controller plugins
- CNCF projects



Supply Chain Security

Reduce security risks in creation and delivery process of your software



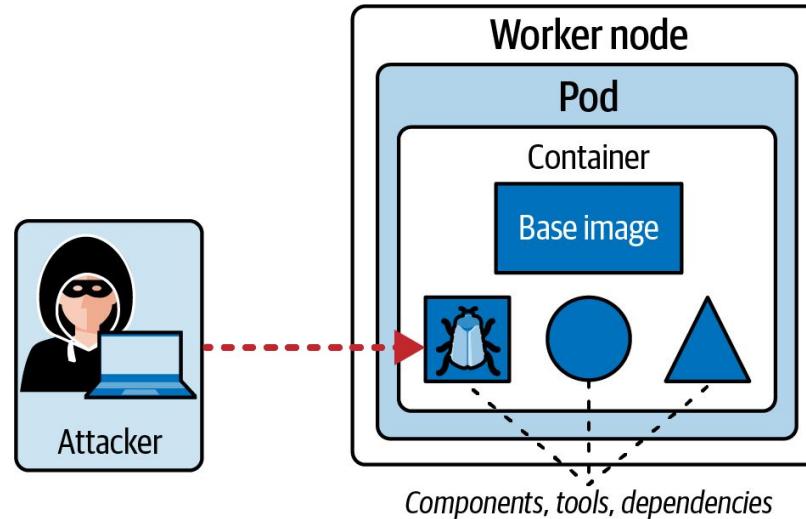
What is Supply Chain Security?

Security as an important aspect of your automation pipelines

- The creation and delivery of software is optimally automated by employing CI/CD pipelines, as well as the consumption of the software in the runtime environment.
- Some best practices include in the context of containerized applications include:
 - Minimizing base image footprint
 - Scanning images for known vulnerabilities
 - Signing container images and software bill of materials (SBOM) attestation
 - Governing the use of public container registries
 - Performing static analysis of user workload

Why Minimize Container Image Footprint?

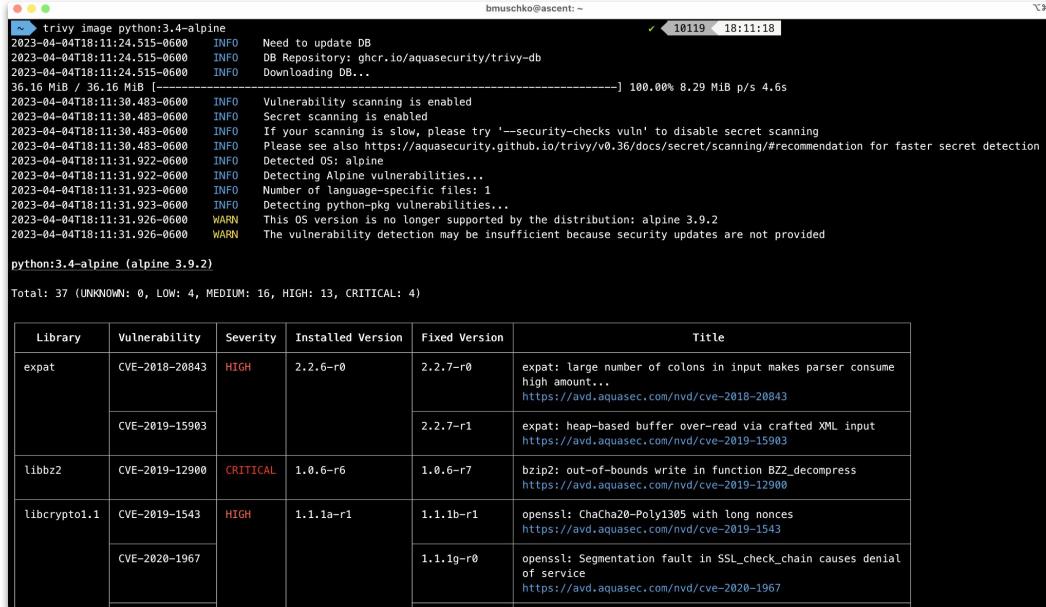
Reducing security risk by including less



Example: <https://docs.docker.com/build/building/best-practices/>

Why Scan Images for Vulnerabilities?

Preventing the use of known vulnerabilities in dependencies



The terminal window shows the Trivy command running on a python:3.4-alpine Docker image. It outputs various logs about the database update and scanning process, including a warning about the Alpine 3.9.2 OS version being unsupported.

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

python:3.4-alpine (alpine 3.9.2)

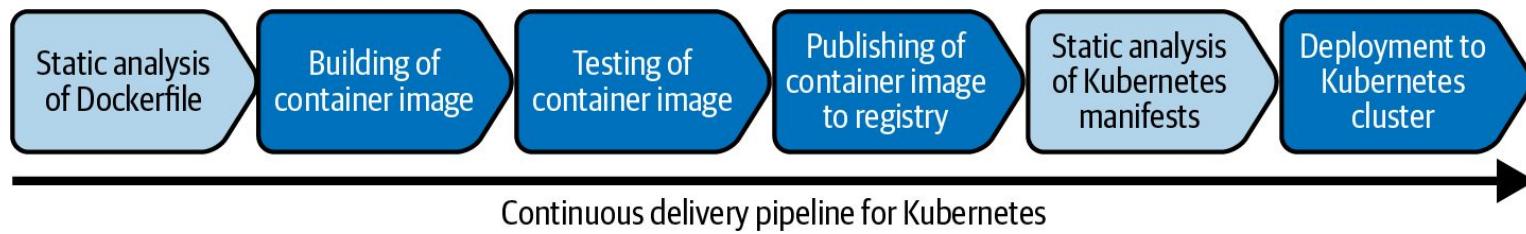
Total: 37 (UNKNOWN: 0, LOW: 4, MEDIUM: 16, HIGH: 13, CRITICAL: 4)
```

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

Example: [Trivy](#)

Why use Static Analysis of Workload?

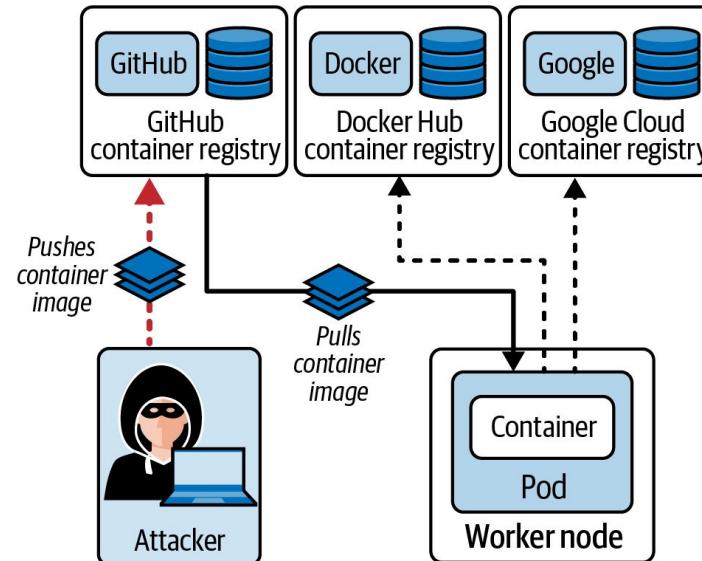
Use tools to enforce the use of best practices



Examples: [Haskell Dockerfile Linter](#), [Kubesec](#)

Why avoid the use of Public Registries?

Only rely on a private registry you can control and govern



Examples: [OPA Gatekeeper](#), [Kyverno](#)



Observability

Monitoring and visualizing the health of your cluster and applications

What is Observability?

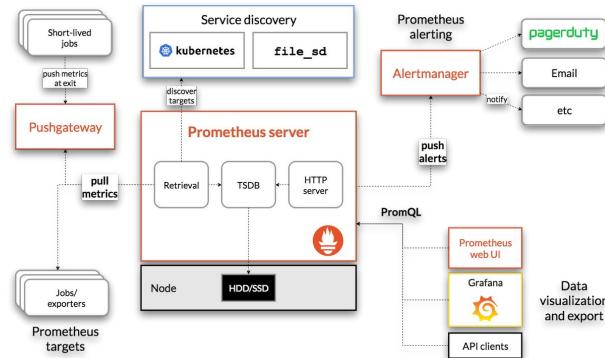
Insights into the real-time performance of Kubernetes clusters and applications

- Infrastructure monitoring for your cluster nodes.
- Application resource monitoring (e.g. CPU, memory, storage, networking metrics).
- Service component metrics for discovering failures.
- Alerting and notifications based on events.
- Support for capturing custom metrics.

Prometheus

Open-source monitoring and alerting tool

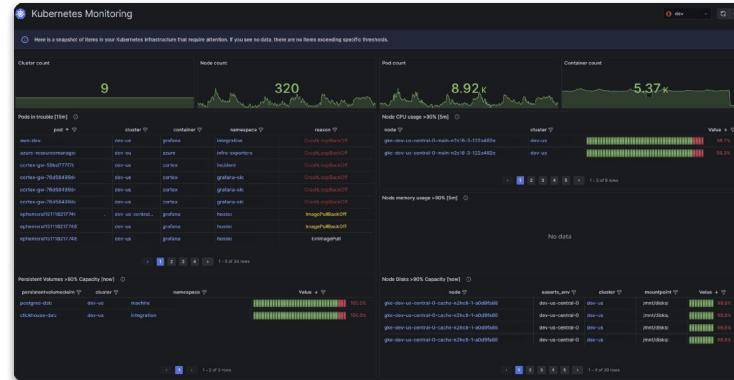
- Cloud-native time-series data store with a built-in query language for metrics.
- Built-in support for Kubernetes and containers.
- Can be installed as Kubernetes operator.



Grafana

Open-source tool to visualize observability data

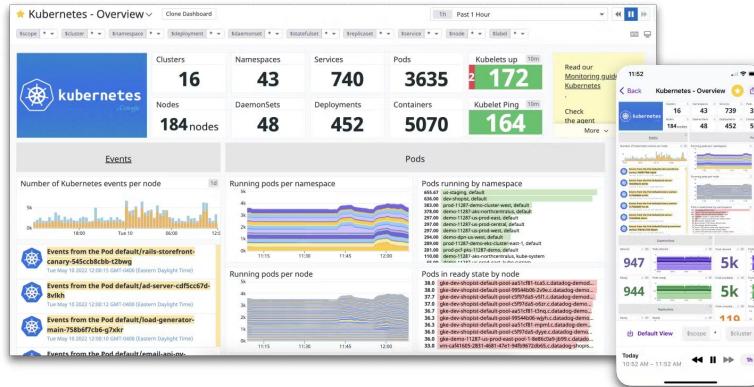
- Data visualization and analytics tool that supports alerting and notifications.
- Integrates with major time-series databases (Prometheus, InfluxDB), Elasticsearch, SQL databases, cloud monitoring service.



Datadog

Commercial observability tool

- Cloud-based monitoring and analytics platform.
- Can be used to visualize data, explore metrics, manage logs, and perform various other tasks.





Service Mesh

Encrypted network communication between cluster components via
TLS

What is a Service Mesh?

Simplify and abstract microservices communication

- Organizations that build their system in a microservices fashion break down their business functionality into distinct, decoupled pieces of software. Each part of the system can be developed and deployed independently.
- In Kubernetes, each microservice would live in a Pod, often times scaled out by a ReplicaSet. Pods need to talk to each other to be able to fulfill the business needs.
- A service mesh is a dedicated infrastructure layer designed to manage, observe, and control communication between microservices within a Kubernetes cluster.



Benefits of Using a Service Mesh

Microservices communication, management, and security

- **Traffic management:** Ensuring that application traffic is routed efficiently and reliably.
- **Security:** Providing end-to-end encryption via mutual TLS.
- **Service-level observability:** Monitor and analyze the performance and behavior of services in real time to diagnose issue.
- **Policy enforcement:** Support access controls, rate limits, and quotas.

Examples for Service Mesh Implementation

Compare feature lists when deciding for a product

- Istio: Provides features such as load balancing, authentication, authorization, rate limiting, and observability.
- Linkerd: Provides features such as load balancing, automatic retries, circuit breaking, and observability.
- Consul service mesh: Provide features such as service segmentation, mutual TLS encryption, and fine-grained access control for microservices. Developed by HashiCorp that can be integrated with Kubernetes.



PKI

Encrypted network communication between cluster components via
TLS

What is Public key infrastructure (PKI)?

Responsible for generating TLS certificates

- Refers to tools used to create and manage public keys for encryption, which is a common method of securing data transfers on the internet.
- When you install Kubernetes using `kubeadm`, certificates used by cluster components are generated automatically.
- You can also generate your own certificates.
- Most certificates are stored in `/etc/kubernetes/pki`. `Kubeadm` places user account certificates in `/etc/kubernetes`.



What does Kubernetes use PKI for?

Primarily for cluster component communication

Kubernetes requires PKI for the following operations:

- Client certificates for the kubelet to authenticate to the API server
- Kubelet [server certificates](#) for the API server to talk to the kubelets
- Server certificate for the API server endpoint
- Client certificates for administrators of the cluster to authenticate to the API server
- Client certificates for the API server to talk to the kubelets
- Client certificate for the API server to talk to etcd
- Client certificate/kubeconfig for the controller manager to talk to the API server
- Client certificate/kubeconfig for the scheduler to talk to the API server.
- Client and server certificates for the [front-proxy](#)

<https://kubernetes.io/docs/setup/best-practices/certificates/#how-certificates-are-used-by-your-cluster>



Admission Control

Validation and mutations of request to the API server

What is an Admission Controller?

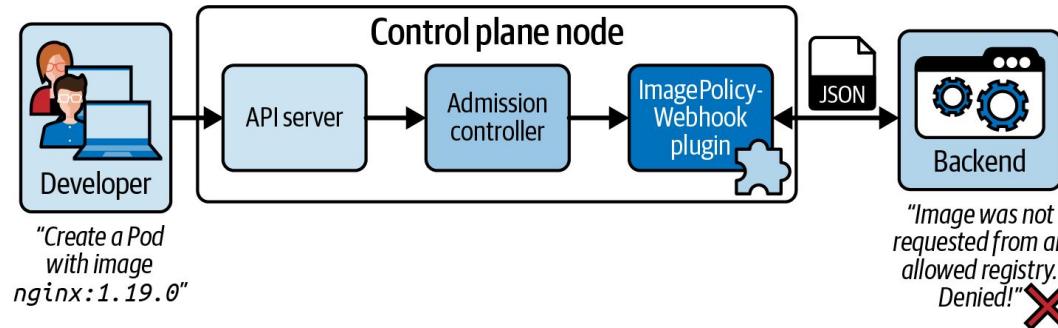
Piece of code that intercepts requests to the Kubernetes API server

- Admission control is the third phase when processing a Kubernetes API request.
- Authentication and authorization needs to pass before admission control comes into play.
- An admission controller provides a way to approve, deny, or mutate a request before the request takes effect.

Admission Controller Plugins

You can choose from a wide range of built-in plugins

- You can configure different [admission controller plugins](#) in the API server.
- For example, the `ImagePolicyWebhook` admission controller plugin defines a policy for all container images used by Pod upon creation of the object.



Admission Control Phases

If any of the phases fails, then the whole request will be rejected

- The admission control process proceeds in two phases. In the first phase, mutating admission controllers are run. In the second phase, validating admission controllers are run.
- **Mutation:** Runs the `MutatingAdmissionWebhook` plugin. Modifies the data for the object requested. This may confuse end users if functionality isn't communicated properly.
- **Validation:** Runs the `ValidatingAdmissionWebhook` plugin. Validates the data for the object requested.

Default Admission Controller Plugins

Configured in the API server via --enable-admission-plugins option

```
$ kubectl get pod kube-apiserver -n kube-system -o yaml | grep enable-admission-plugins
  - --enable-admission-plugins=CertificateApproval, CertificateSigning,
  CertificateSubjectRestriction, DefaultIngressClass, DefaultStorageClass,
  DefaultTolerationSeconds, LimitRanger, MutatingAdmissionWebhook,
  NamespaceLifecycle, PersistentVolumeClaimResize, PodSecurity, Priority,
  ResourceQuota, RuntimeClass, ServiceAccount, StorageObjectInUseProtection,
  TaintNodesByCondition, ValidatingAdmissionPolicy, ValidatingAdmissionWebhook
```

Those are the default plugins enabled in Kubernetes 1.30.

Cloud-Native Governance Projects

CNCF projects that help with rolling out governance on an enterprise level

- Implementing custom solutions in Kubernetes is not necessarily easy.
- You can choose from official CNCF that solve the project declaratively, e.g. OPA Gatekeeper or Kyverno.



Exam Essentials

What to focus on for the exam

- You will only need to understand concepts on a higher level. Details are usually not asked about.
- In this domain, strongly focus on admission control. That includes its purpose, phases, and plugins.

The background features a vibrant red-to-yellow gradient. Overlaid on this gradient are several semi-transparent, overlapping circular shapes in shades of red, orange, and yellow, creating a sense of depth and motion.

O'REILLY®