Kubernetes Security

Karthikeyan Vaiyapuri

Agenda

01

Securing Kubernetes
(RBAC, PodSecurity, NetworkPolicies)

02

Secrets Management and Image Scanning

03

TLS and Admission Controllers

01

Kubernetes Security Fundamentals

The Security Challenge

01

Default Kubernetes installations are **not secure**

02

Multiple attack vectors: API server, etcd, nodes, pods, network

03

Security must be implemented at **every layer**

Defense in Depth Strategy

Cluster Security: Secure the control plane and nodes

Network Security: Control traffic flow between components

Pod Security: Restrict container capabilities and privileges

Application Security: Secure the applications themselves

Security Principles

Least Privilege: Grant minimum necessary permissions

Zero Trust: Verify everything, trust nothing

Continuous Monitoring: Monitor and audit all activities

Understanding Authentication vs Authorization

Authentication - "Who are you?"

X.509 Client Certificates: Most common for users

Service Account Tokens: For pods and applications

Static Token Files: Simple but less secure

OIDC/LDAP Integration: Enterprise authentication

Authorization - "What can you do?"

RBAC: Role- Based Access Control (recommended)

ABAC: Attribute-Based Access Control

Node Authorization: For kubelet access

Webhook: Custom authorization logic

The Flow

User Request → Authentication → Authorization → Admission Control → API Server

RBAC Core Concepts

Key Components

Subjects: Users, Groups, Service Accounts

Resources: Pods, Services, ConfigMaps, etc.

Verbs: get, list, create, update, patch, delete

API Groups: core, apps, extensions, etc.

RBAC Objects

Role: Permissions within a namespace

ClusterRole: Permissions across the entire cluster

RoleBinding: Binds Role to Subjects in a namespace

ClusterRoleBinding: Binds ClusterRole to Subjects cluster- wide

Permission Model

01

Additive Only: Permissions are granted, never denied

02

Explicit: Must explicitly grant each permission

03

Namespace Scoped vs Cluster Scoped

RBAC Role Definition

Role Structure

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: development
  name: pod-reader
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "watch", "list"]
```

Common Verbs

- **Get:** Retrieve a specific resource
- **List:** Retrieve all resources of a type
- **Watch:** Watch for changes to resources
- **Create:** Create new resources
- **Update:** Update existing resources
- **Patch:** Partially update existing resources
- **Delete:** Delete resources

RBAC RoleBinding

Binding Roles to Subjects

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: read-pods
  namespace: development
subjects:
- kind: User
  name: jane
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: pod-reader
  apiGroup: rbac.authorization.k8s.io
```

Subject Types

User: Human users (external to Kubernetes)

Group: Collections of users

ServiceAccount: Applications and pods

ClusterRole and ClusterRoleBinding

Cluster-Wide Permissions

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: secret-reader
rules:
- apiGroups: [""]
  resources: ["secrets"]
  verbs: ["get", "watch", "list"]
```

When to Use ClusterRole

- **Cluster- scoped resources:** nodes, persistentvolumes
- **Non-resource endpoints:** /healthz, /version
- **Cross-namespace access:** monitoring, logging systems
- **Administrative tasks:** cluster management

Service Accounts in RBAC

Service Account Basics

Default Service Account: Every namespace has one

Pod Association: Every pod runs with a service account

Token Mounting: Automatically mounted at */var/run/secrets/kubernetes.io/serviceaccount/*

Creating Custom Service Accounts

01

```yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: my-app-sa
  namespace: production
```

Best Practices

Create dedicated service accounts for each application

Use principle of least privilege

Regularly audit service account permissions

Disable token auto- mounting when not needed

Recommendations

RBAC Best Practices

Security Guidelines

Start with Minimal Permissions: Grant only what's needed

Use Namespaces: Segregate resources and permissions

Regular Audits: Review and clean up unused permissions

Avoid Wildcards: Be specific with resources and verbs

Common Patterns

Application- Specific Roles: One role per application type

Environment- Based Roles: Different permissions per environment

Team- Based Roles: Align with organizational structure

Troubleshooting

Use kubectl auth can-i to test permissions

Check RBAC events in cluster logs

Use --dry-run to test configurations

Pod Security Evolution

Historical Context

01

Pod Security Policies (PSP): Original security mechanism (deprecated in v1.25)

02

Pod Security Standards: New simplified approach

03

Migration Path: PSP → Pod Security Standards

Why the Change?

Reasons

Complexity: PSPs were difficult to configure and debug

Usability: Many edge cases and confusing behaviors

Maintenance: High operational overhead

Pod Security Standards Benefits

Advantages

01 **Simplicity:** Three predefined security profiles

02 **Transparency:** Clear understanding of what's allowed/denied

03 **Flexibility:** Can be applied at namespace level

Pod Security Standards Profiles

Privileged

No restrictions: Equivalent to no Pod Security Policy

Use case: Trusted workloads, system components

Risk: High - allows all potentially dangerous configurations

Baseline

“

01

Minimally restrictive: Prevents known privilege escalations

02

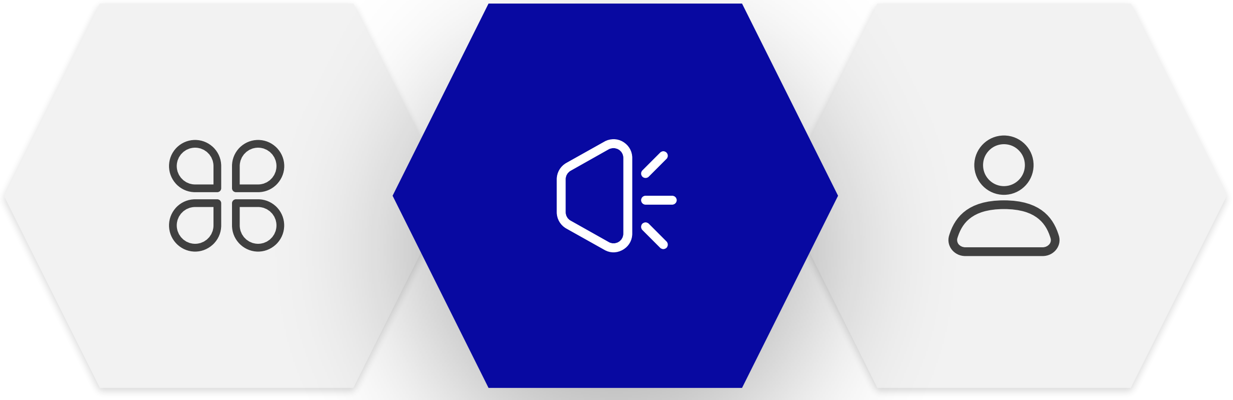
Blocked capabilities: Privileged containers, host networking, host filesystem access

03

Use case: Non- critical applications

Restricted

Heavily restricted: Current Pod hardening best practices



Use case: Security- critical applications

Additional restrictions: Non- root containers, read- only root filesystem, restricted volume types

Pod Security Standards Implementation

Enforcement Modes

Enforce: Violating pods are rejected

Audit: Violations are logged but pods are allowed

Warn: Violations trigger user-facing warnings

Namespace Labels

```yaml
apiVersion: v1
kind: Namespace
metadata:
  name: production-apps
  labels:
    pod-security.kubernetes.io/enforce: restricted
    pod-security.kubernetes.io/audit: restricted
    pod-security.kubernetes.io/warn: restricted
```

Version Pinning

Pin to specific Kubernetes version for consistency

Example: pod-security.kubernetes.io/enforce-version: v1.28

Security Context Configuration

Pod Security Context

```yaml
apiVersion: v1
kind: Pod
spec:
  securityContext:
    runAsNonRoot: true
    runAsUser: 1000
    runAsGroup: 3000
    fsGroup: 2000
    seccompProfile:
      type: RuntimeDefault
```

Container Security Context

```yaml
containers:
- name: app
  securityContext:
    allowPrivilegeEscalation: false
    readOnlyRootFilesystem: true
    runAsNonRoot: true
    capabilities:
      drop:
      - ALL
```

Network Security Fundamentals

Default Kubernetes Networking

Flat Network Model: All pods can communicate with all other pods

No Network Segmentation: By default, no traffic restrictions

Security Risk: Lateral movement, data exfiltration

Network Policy Benefits

Micro- segmentation: Isolate workloads at network level

Zero Trust Networking: Explicit allow model

Compliance: Meet regulatory requirements

Incident Containment: Limit blast radius of security incidents

CNI Requirement

Plugins

01

Network policies require CNI plugin support

02

Supported: Calico, Cilium, Weave Net, Antrea

03

Not Supported: Flannel (basic), basic kubenet

Network Policy Structure

Policy Components

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: deny-all
  namespace: production
spec:
  podSelector: {}  # Applies to all pods in namespace
  policyTypes:
  - Ingress
  - Egress
```

Traffic Direction

Directions

Egress: Traffic leaving pods

Ingress: Traffic coming into pods

Policy Types: Must specify which directions to control

Selection Mechanisms

Types

PodSelector: Select pods within same namespace

NamespaceSelector: Select entire namespaces

IpBlock: Select IP address ranges

Network Policy Examples

Default Deny All Traffic

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

Allow Specific Pod Communication

```yaml
spec:
  podSelector:
    matchLabels:
      app: web
  ingress:
  - from:
    - podSelector:
        matchLabels:
          app: frontend
    ports:
    - protocol: TCP
      port: 8080
```

Advanced Network Policies

Cross-Namespace Communication

```yaml
ingress:
- from:
  - namespaceSelector:
      matchLabels:
        name: monitoring
  - podSelector:
      matchLabels:
        app: prometheus
```

External Traffic Control

```yaml
egress:
- to:
  - ipBlock:
      cidr: 10.0.0.0/8
      except:
      - 10.0.1.0/24
  ports:
  - protocol: TCP
    port: 443
```

DNS Policy

- Always allow DNS traffic for service discovery

- Typically allow egress to kube-dns/coredns

Network Policy Best Practices

Implementation Strategy

Start with Audit Mode: Monitor existing traffic patterns

Implement Gradually: Begin with non- critical applications

Default Deny: Create baseline deny- all policies

Explicit Allow: Add specific allow rules as needed

Common Patterns

01

Three-Tier Architecture: Web → App → Database tiers

02

Microservices: Service- to- service communication rules

03

External Access: Ingress controller and egress gateway rules

Monitoring and Troubleshooting

Enable CNI logging for policy violations

Use network monitoring tools

Test policies in staging environments first

02

Secrets Management and Image Scanning

Kubernetes Secrets Fundamentals

What are Secrets?

Purpose: Store and manage sensitive information

Base64 Encoding: Data is base64 encoded (not encrypted!)

etcd Storage: Stored in cluster's etcd database

Access Control: Integrated with RBAC system

Types of Secrets

- **Opaque:** Arbitrary user- defined data
- **kubernetes.io/dockerconfigjson:** Docker registry credentials
- **kubernetes.io/tls:** TLS certificates and keys
- **kubernetes.io/service-account-token:** Service account tokens

Security Considerations

01

Secrets are **NOT encrypted by default**

02

Base64 encoding is easily reversible

03

Enable encryption at rest for production

Secret Creation and Management

Creating Secrets

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: app-secrets
  namespace: production
type: Opaque
data:
  username: YWRtaW4=  # admin in base64
  password: MWYyZDFlMmU2N2Rm  # password in base64
```

Using Secrets in Pods

```yaml
containers:
- name: app
  env:
  - name: DB_USER
    valueFrom:
      secretKeyRef:
        name: app-secrets
        key: username
```

Volume Mounting

```yaml
volumes:
- name: secret-volume
  secret:
    secretName: app-secrets
```

Secrets Best Practices

Security Guidelines

Enable Encryption at Rest: Configure etcd encryption

Limit Access: Use RBAC to control secret access

Separate Secrets: Don't combine unrelated secrets

Regular Rotation: Implement secret rotation policies

Operational Practices

External Secret Management: Use tools like HashiCorp Vault, AWS Secrets Manager

GitOps Considerations: Never commit secrets to Git

Backup and Recovery: Include secrets in backup strategies

Monitoring: Audit secret access and modifications

Alternative Solutions

01

External Secrets Operator: Sync from external secret stores

02

Sealed Secrets: Encrypt secrets for GitOps workflows

03

CSI Secret Store Driver: Mount secrets from external systems

Container Image Security Overview

Image Security Challenges

01

Vulnerable Base Images: Outdated OS packages and libraries

02

Embedded Secrets: Hardcoded credentials and API keys

03

Malicious Code: Compromised or malicious container images

04

Configuration Issues: Insecure default configurations

Security Scanning Approaches

Static Analysis: Scan images before deployment

Dynamic Analysis: Monitor running containers

Continuous Scanning: Regular rescanning of stored images

Policy Enforcement: Block vulnerable images from deployment

Industry Standards

01 **CVE Database:** Common Vulnerabilities and Exposures

02 **CVSS Scoring:** Severity rating system

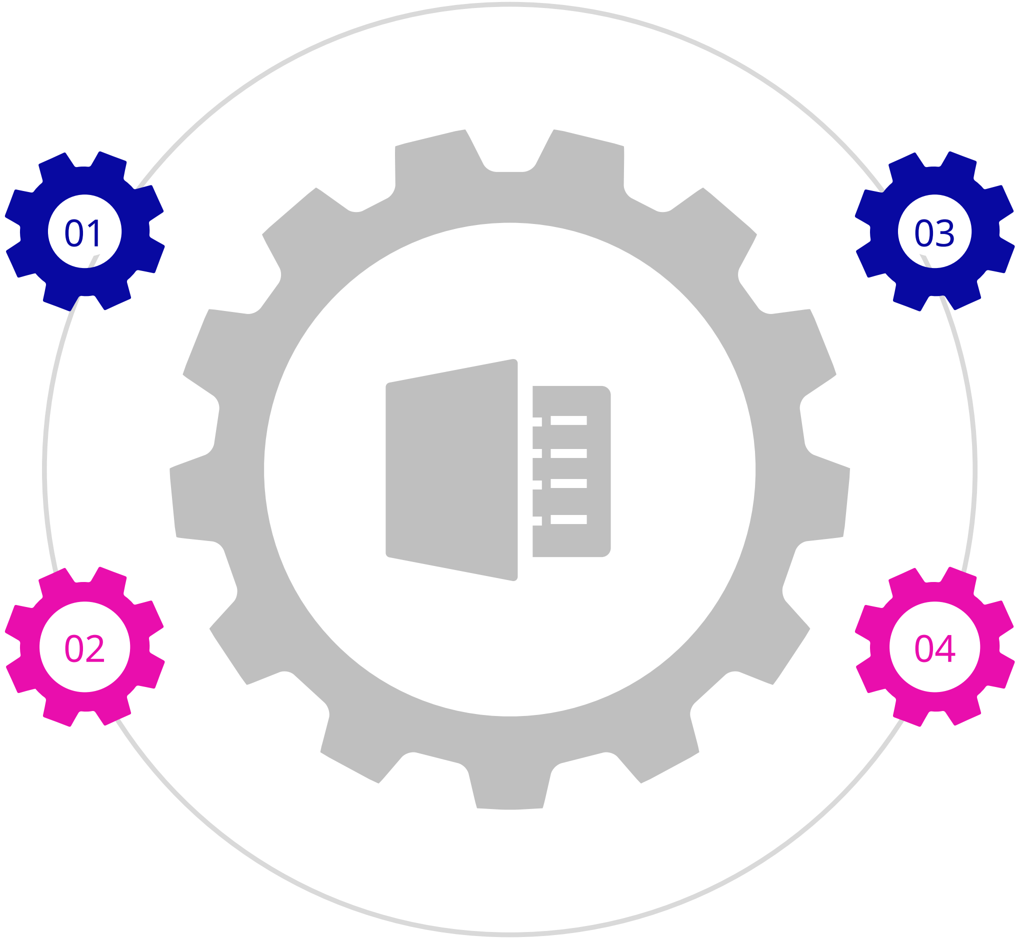
03 **CIS Benchmarks:** Security configuration guidelines

Image Scanning Techniques

Vulnerability Scanning Methods

Pattern Matching: Regular expressions for known vulnerability signatures

Database Comparison: Compare against known vulnerability databases



Package Analysis: Analyze installed packages and versions

Behavioral Analysis: Monitor runtime behavior for anomalies

Secret Detection Methods

Regular Expression Based: Pattern matching for common secret formats

Dictionary Based: Compare against known secret patterns

Entropy Analysis: Detect high-entropy strings

Machine Learning: AI-based detection of potential secrets

Scanning Scope

01

OS Packages: System-level vulnerabilities

02

Application Dependencies: Language-specific packages (npm, pip, gem)

03

Configuration Files: Insecure configurations

04

Embedded Secrets: API keys, passwords, certificates

Image Scanning Tools and Integration

Popular Scanning Tools

Trivy: Comprehensive vulnerability scanner

Clair: Open-source vulnerability scanner

Snyk: Commercial security platform

Anchore: Enterprise container security

Twistlock/Prisma Cloud: Comprehensive security platform

Integration Points

CI/CD Pipelines: Scan during build process

Container Registries: Scan on push/pull

Kubernetes Admission Controllers: Scan before deployment

Runtime Scanning: Continuous monitoring of running containers

Registry Integration

01

Harbor: Open- source registry with built- in scanning

02

Docker Hub: Vulnerability scanning for public images

03

AWS ECR: Image scanning service

04

Google Container Registry: Vulnerability scanning

03

TLS and Admission Controllers

TLS in Kubernetes Overview

TLS Requirements in Kubernetes

01

API Server: Client-server communication encryption

02

Etcd: Cluster data store encryption

03

Kubelet: Node-to-control plane communication

04

Service Mesh: Pod-to-pod communication encryption

Certificate Types

Client Certificates: User and component authentication

Server Certificates: Service identity and encryption

CA Certificates: Certificate authority for trust chain

PART 01

PART 02

PART 03

PKI Architecture

Cluster CA: Root certificate authority

Component Certificates: Individual service certificates

Certificate Rotation: Automatic renewal and rotation

TLS Configuration Components

API Server TLS

```
# API Server TLS Configuration
--tls-cert-file=/etc/kubernetes/pki/apiserver.crt
--tls-private-key-file=/etc/kubernetes/pki/apiserver.key
--client-ca-file=/etc/kubernetes/pki/ca.crt
--tls-cipher-suites=TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
```

etcd TLS

```
# etcd TLS Configuration
--cert-file=/etc/kubernetes/pki/etcd/server.crt
--key-file=/etc/kubernetes/pki/etcd/server.key
--trusted-ca-file=/etc/kubernetes/pki/etcd/ca.crt
--peer-cert-file=/etc/kubernetes/pki/etcd/peer.crt
--peer-key-file=/etc/kubernetes/pki/etcd/peer.key
```

Certificate Locations

01
Standard Path:
/etc/kubernetes/pki/

02
etcd Certificates:
/etc/kubernetes/pki/etcd/

03
Service Account Keys:
/etc/kubernetes/pki/sa.key

Certificate Management

Certificate Lifecycle

1. Generation: Create certificates with proper SANs

2. Distribution: Securely distribute to components

3. Rotation: Regular renewal before expiration

4. Revocation: Handle compromised certificates

Tools for Certificate Management

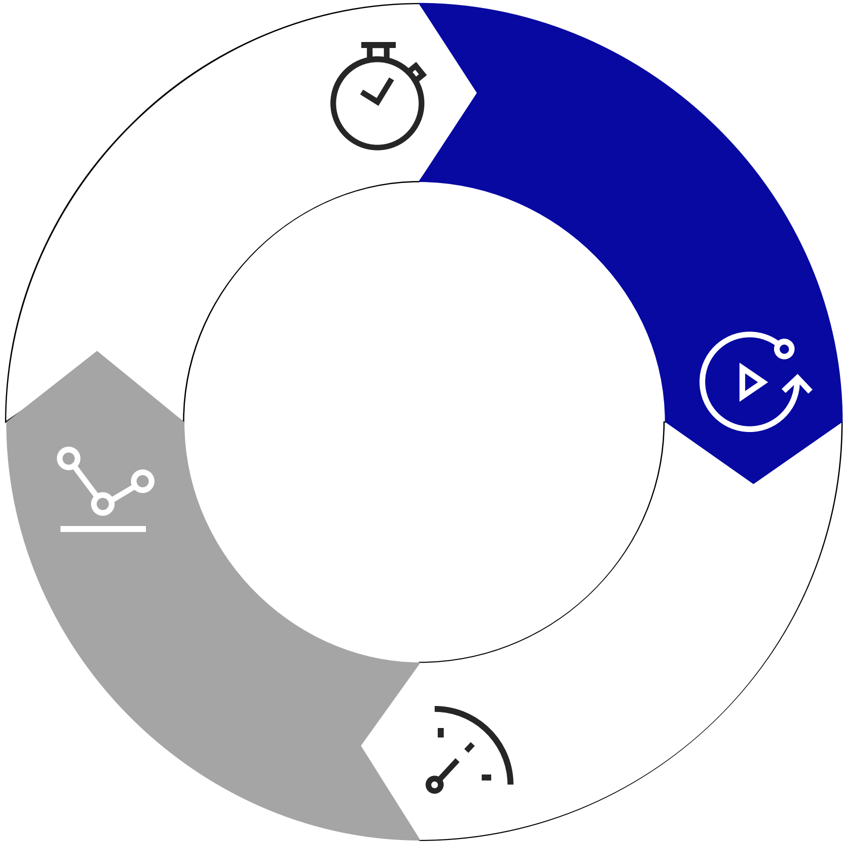
Kubeadm: Automatic certificate generation and renewal

cert-manager: Kubernetes certificate management operator

external-secrets: Integration with external PKI systems

Manual Management: Custom scripts and processes

Best Practices



01 **Short Certificate Lifetimes:** Reduce exposure window

02 **Automated Rotation:** Minimize manual intervention

03 **Monitoring:** Alert on certificate expiration

04 **Backup:** Secure storage of CA private keys

Admission Controllers Overview

What are Admission Controllers?

Gatekeepers: Intercept API requests after authentication/authorizat ion

Request Modification: Can modify or reject requests

Policy Enforcement: Implement cluster-wide policies

Security Controls: Additional security layer

Admission Controller Chain

01

Authentication → Authorization → Admission Controllers → API Server

Types of Admission Controllers

01

Mutating: Modify incoming requests

02

Validating: Validate requests (allow/deny)

03

Both: Can both mutate and validate

Built-in Admission Controllers

Essential Security Controllers

01

NamespaceLifecycle:

Prevents operations on terminating namespaces

02

LimitRanger:

Enforces resource limits

03

ServiceAccount:

Automatically adds service accounts to pods

04

DefaultStorageClass:

Adds default storage class to PVCs

05

ResourceQuota:

Enforces resource quotas

06

PodSecurityPolicy:

Enforces pod security policies (deprecated)

Configuration

-enable- admission-
plugins=*NamespaceLifecycle,LimitRanger,ServiceAccount,DefaultStorageClass,ResourceQuota,MutatingAdmissionWebhook*
,ValidatingAdmissionWebhook

Recommended Controllers

01

Always enable:
NamespaceLifecycle,
LimitRanger, ServiceAccount

02

Security:
PodSecurity, ResourceQuota

03

Webhooks:
MutatingAdmissionWebhook,
ValidatingAdmissionWebhook

Custom Admission Controllers

Admission Webhooks

Mutating Admission Webhooks

Modify incoming requests

Validating Admission Webhooks

Validate requests

Dynamic Configuration

Register webhooks dynamically

Custom Logic

Implement organization-specific policies

Webhook Configuration

```yaml
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingAdmissionWebhook
metadata:
  name: security-policy-webhook
webhooks:
- name: validate-security.example.com
  clientConfig:
    service:
      name: security-webhook
      namespace: kube-system
      path: "/validate"
  rules:
  - operations: ["CREATE", "UPDATE"]
    apiGroups: [""]
    apiVersions: ["v1"]
    resources: ["pods"]
```

Popular Admission Controllers

Open Policy Agent (OPA) Gatekeeper

01

Policy as Code: Define policies using Rego language

02

Constraint Framework: Template-based policy definitions

03

Validation and Mutation: Support for both types

04

Audit: Continuously validate existing resources

Falco

Runtime Security: Monitor running containers

Behavioral Analysis: Detect anomalous activities

Rule Engine: Flexible rule definition system

Integration: Works with Kubernetes admission controllers

Polaris

Configuration Validation: Validate Kubernetes configurations

Best Practices: Enforce security and reliability best practices

Dashboard: Visual representation of cluster health

Webhook Mode: Can run as admission controller

Security Monitoring and Auditing

Kubernetes Audit Logging

Audit Events:

Log all API server requests

Audit Levels:

Metadata, Request, RequestResponse, None

Audit Policies:

Define what to log and at what level

Storage Backends:

Log files, webhooks, dynamic backends

Key Metrics to Monitor

01 **Failed Authentication Attempts:** Potential security breaches

02 **RBAC Violations:** Unauthorized access attempts

03 **Pod Security Policy Violations:** Security policy breaches

04 **Network Policy Violations:** Unauthorized network traffic

05 **Resource Usage:** Potential DoS attacks

Alerting Strategies

01

Real-time Alerts: Critical security events

02

Trend Analysis: Behavioral pattern analysis

03

Compliance Reporting: Regular security posture reports

Security Scanning and Compliance

Compliance Frameworks

CIS Kubernetes Benchmark: Industry standard security configurations

NSA/CISA Hardening Guide: Government security recommendations

PCI DSS: Payment card industry requirements

SOC 2: Service organization control requirements

Scanning Tools

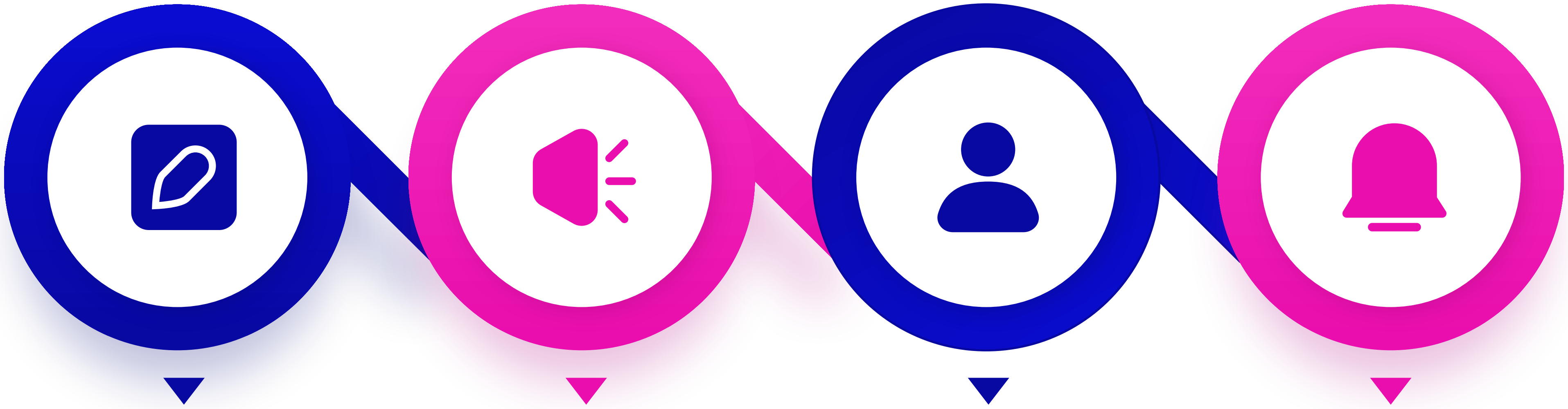
kube-bench: CIS Kubernetes Benchmark scanner

kube-hunter: Penetration testing tool

Kubesec: Security risk analysis for Kubernetes manifests

Starboard: Kubernetes security toolkit

Continuous Compliance



Automated Scanning: Regular security assessments

Policy Drift Detection: Monitor configuration changes

Remediation Workflows: Automated fixing of security issues

Compliance Dashboard: Visual security posture tracking

Incident Response and Recovery

Security Incident Response Plan

01

Detection: Identify security incidents quickly

02

Containment: Isolate affected components

03

Investigation: Determine scope and impact

04

Remediation: Fix vulnerabilities and restore services

05

Recovery: Return to normal operations

06

Lessons Learned: Improve security posture

Kubernetes-Specific Response

- **Pod Isolation:** Network policies and resource quotas
- **Node Quarantine:** Cordon and drain compromised nodes
- **RBAC Review:** Audit and revoke compromised permissions
- **Secret Rotation:** Rotate potentially compromised secrets

Forensics and Evidence

01

Audit Logs: Preserve API server audit trails

02

Container Images: Preserve images for analysis

03

Network Flows: Capture network traffic logs

04

System State: Snapshot cluster state during incident

07

Security Best Practices Summary

Cluster Hardening Checklist

- Enable RBAC and implement least privilege
- Configure Pod Security Standards
- Implement Network Policies (default deny)
- Enable audit logging
- Encrypt etcd data at rest
- Use TLS for all communications
- Regular security scanning and patching
- Implement admission controllers

Operational Security

Regular Updates: Keep Kubernetes and components updated

Access Reviews: Regularly audit and clean up permissions

Secret Management: Use external secret management systems

Backup and Recovery: Test backup and recovery procedures

Security Training: Keep team updated on security practices

Advanced Security Topics

Service Mesh Security

mTLS: Mutual TLS for service- to- service communication

Traffic Policy: Fine-grained traffic control

Security Policies: Application- level security rules

Observability: Detailed security metrics and tracing

Supply Chain Security

01

Image Signing: Verify image authenticity with tools like Sigstore

02

SBOM: Software Bill of Materials for dependency tracking

03

Build Provenance: Verify build integrity and source

04

Admission Controllers: Enforce signed image policies

Zero Trust Architecture

Identity-Based Security: All entities must be authenticated

Micro-segmentation: Network isolation at granular level

Continuous Verification: Regular re-authentication and authorization

Minimal Access: Just-in-time and just-enough access

Troubleshooting Security Issues

Common RBAC Issues

01

Permission Denied: Check role bindings and cluster role bindings

02

Service Account Issues: Verify service account token mounting

03

Cross-Namespace: AccessReview namespace-specific permissions

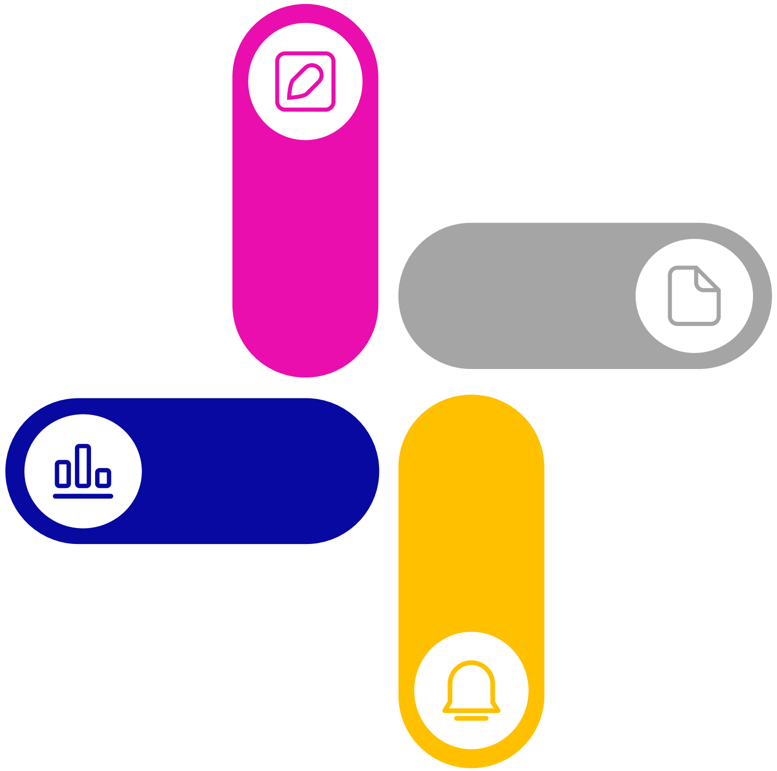
Network Policy Troubleshooting

01

CNI Compatibility: Ensure CNI supports network policies

02

Policy Conflicts: Check for overlapping or conflicting policies



03

DNS Resolution: Verify DNS traffic is allowed

04

Default Deny: Ensure proper allow rules are in place

Debugging Tools and Commands

```
# Check RBAC permissions
kubectl auth can-i create pods --as=user@example.com

# Describe network policies
kubectl describe networkpolicy

# Check admission controller logs
kubectl logs -n kube-system kube-apiserver-node1
```

Future of Kubernetes Security

Emerging Trends

01.

Runtime Security: Advanced behavioral analysis

02.

AI/ML Integration: Machine learning for threat detection

03.

Zero Trust Networking: Identity-based network security

04.

Confidential Computing: Hardware- based isolation

05.

Supply Chain Security: End- to- end software supply chain protection

Evolving Standards

01

SLSA: Supply- chain Levels for Software Artifacts

02

SPIRE/SPIFFE: Secure identity framework

03

Sigstore: Code signing and verification

04

in-toto: Supply chain integrity framework

Community Initiatives

CNCF Security SIG: Cloud Native security best practices

Kubernetes Security Working Group: Core security improvements

OpenSSF: Open Source Security Foundation initiatives

Session Recap and Q&A

Key Takeaways

01

Defense in Depth: Implement security at every layer

02

Principle of Least Privilege: Grant minimum necessary permissions

03

Continuous Monitoring: Monitor and audit all activities

04

Regular Updates: Keep all components updated and patched

05

Education: Keep team informed about latest security practices

Thanks

Karthikeyan Vaiyapuri