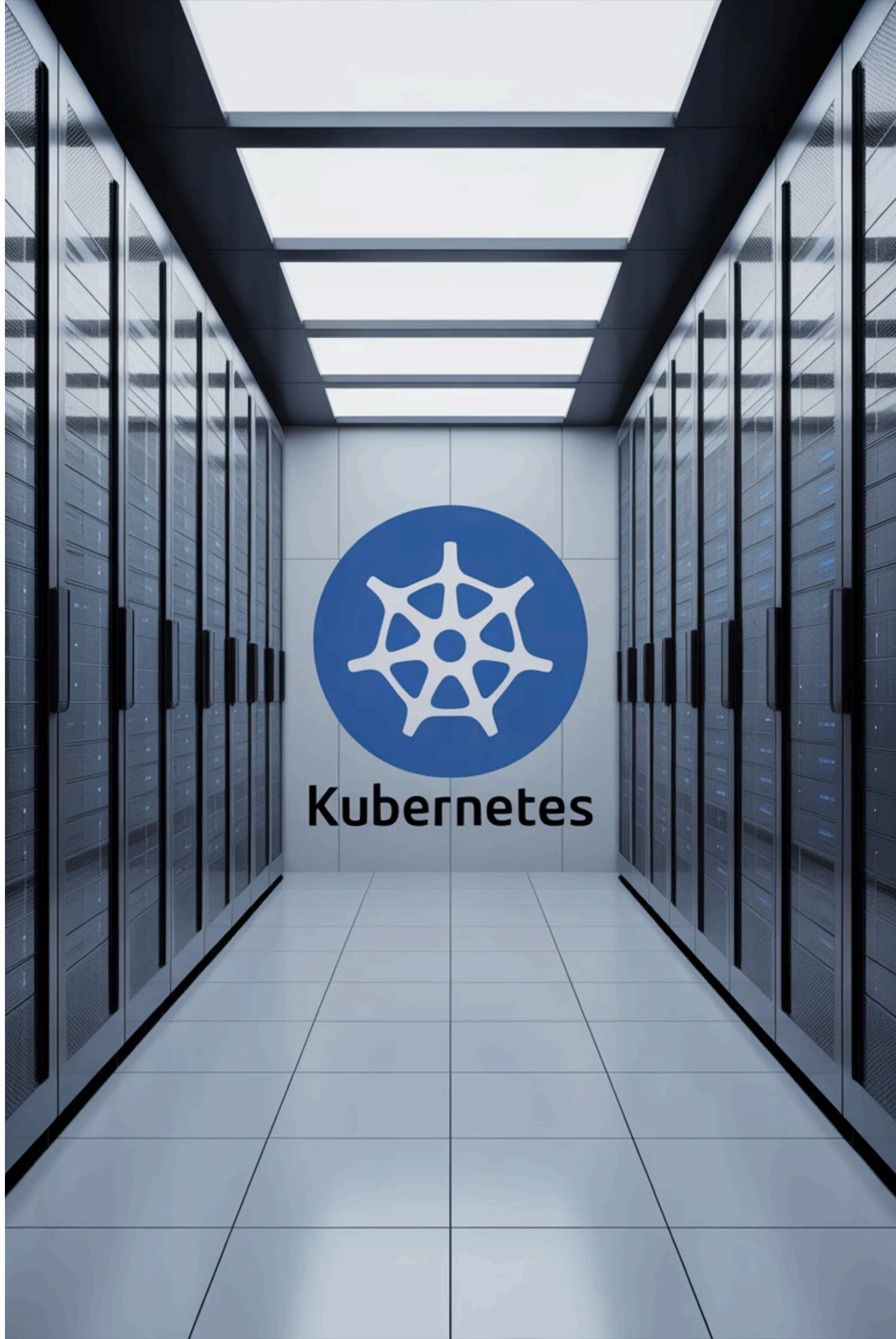
Kubernetes Security Best Practices

A practical guide for platform engineers at TechVault Inc.



Meet TechVault Inc.

TechVault Inc. is a mid-sized fintech company that recently migrated its payment processing platform to Kubernetes. With 50 microservices handling sensitive financial data, their platform team quickly learned that default Kubernetes configurations weren't enough to meet compliance requirements.

Their DevOps team, led by Sarah Chen, discovered multiple security gaps during their first audit. What follows are the hard-won lessons they learned while securing their production clusters—lessons that can help you avoid similar pitfalls.

The Wake-Up Call

The Incident

During a routine penetration test, an external security firm gained unauthorized access to TechVault's development cluster through misconfigured RBAC policies. While no production data was compromised, the incident revealed critical security gaps.

The security team identified seven major vulnerability categories that needed immediate attention across their Kubernetes infrastructure.

The Seven Pillars of Kubernetes Security

CIS Benchmarks

Component configuration validation

Network Policies

Cluster-level access control

TLS Ingress

Encrypted traffic management

Metadata Protection

Node security hardening

Binary Verification

Supply chain security

RBAC Controls

Principle of least privilege

Service Accounts

Identity management

Chapter 1: Configuration Compliance

CIS Benchmarks

Establishing a security baseline for Kubernetes components

Understanding CIS Benchmarks

The Center for Internet Security (CIS) publishes comprehensive security configuration benchmarks for Kubernetes. These benchmarks provide detailed guidance on hardening etcd, kubelet, kube-dns, and kube-apiserver components.

At TechVault, the platform team discovered their etcd database was running with weak encryption settings and their API server allowed anonymous requests—both critical CIS benchmark failures that could expose sensitive configuration data.

01

Download CIS Kubernetes Benchmark

Obtain the latest version from cisecurity.org

02

Run automated scanning tools

Use kube-bench for automated compliance checks

03

Review failed checks

Prioritize by severity and business impact

04

Implement remediations

Apply fixes systematically across clusters

05

Establish continuous monitoring

Automate compliance checking in CI/CD

TechVault's CIS Implementation

The Challenge

TechVault's initial kube-bench scan revealed 47 failed checks across three production clusters. The most critical issues involved API server authentication, etcd encryption at rest, and kubelet authorization modes.

Sarah's team needed a systematic approach to address these findings without disrupting production services.

The Solution

```
# Sample kube-bench remediation
apiVersion: v1
kind: Pod
metadata:
 name: kube-apiserver
spec:
 containers:
 - command:
 - kube-apiserver
 - --anonymous-auth=false
 - --enable-admission-plugins=NodeRestriction
 - --audit-log-path=/var/log/apiserver/audit.log
 - --encryption-provider-config=/etc/kubernetes/enc/config.yaml
```

Key Components to Secure

1

etcd

Enable encryption at rest, restrict peer communication to TLS, disable unused endpoints, and implement client certificate authentication for all connections.

2

kube-apiserver

Disable anonymous authentication, enable audit logging, enforce strong admission controllers like PodSecurityPolicy, and require TLS for all API communications.

3

kubelet

Enable certificate rotation, disable read-only port, enforce authorization mode to Webhook, and restrict node access to only necessary API resources.

4

kube-dns/CoreDNS

Limit query logging exposure, apply network policies to restrict DNS traffic, update to patched versions regularly, and monitor for DNS tunneling attacks.

Results After CIS Hardening

47

Initial Failures

Critical security gaps identified

3

Remaining Issues

Low-priority findings deferred

94%

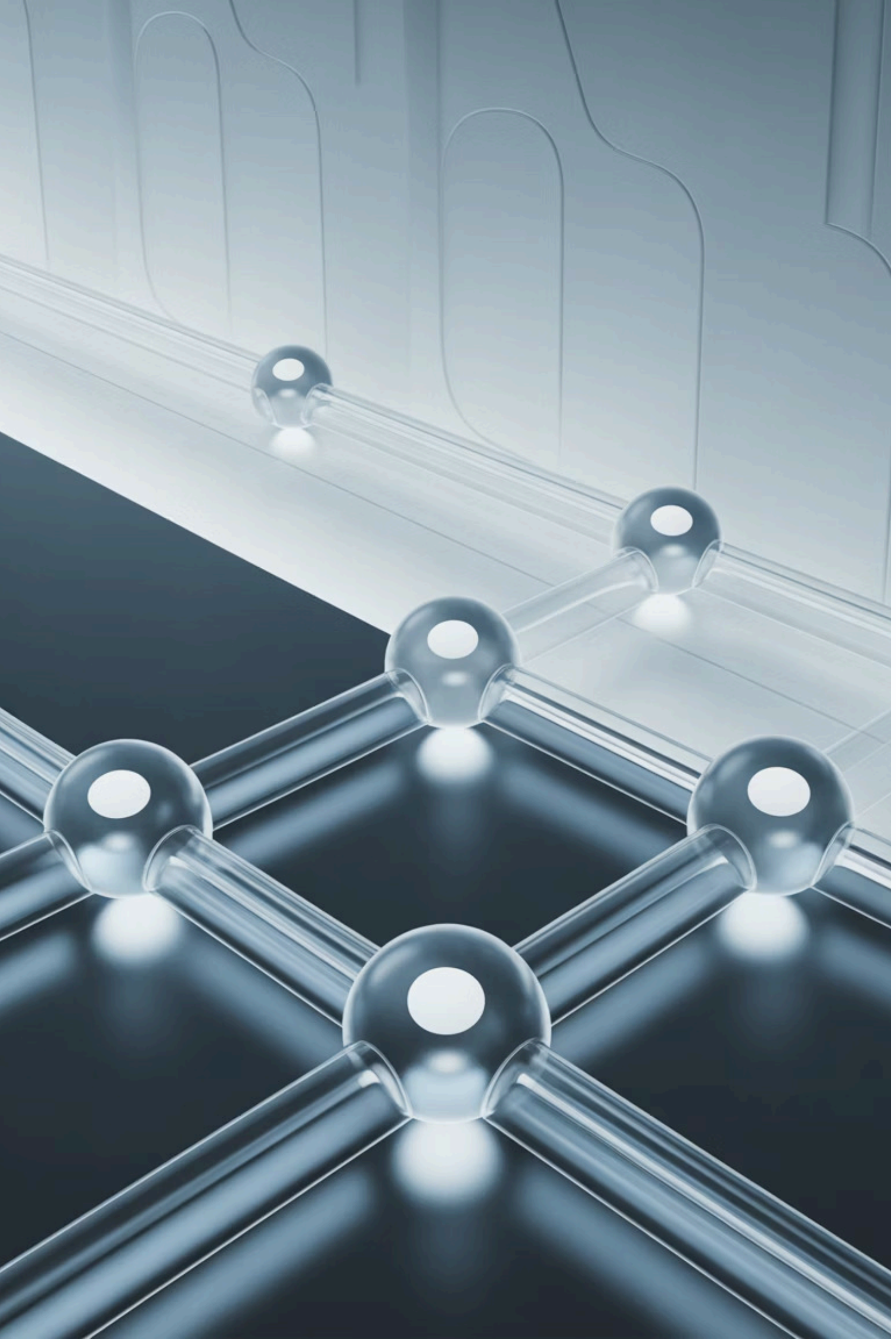
Compliance Rate

After systematic remediation

2wks

Implementation Time

From scan to production rollout

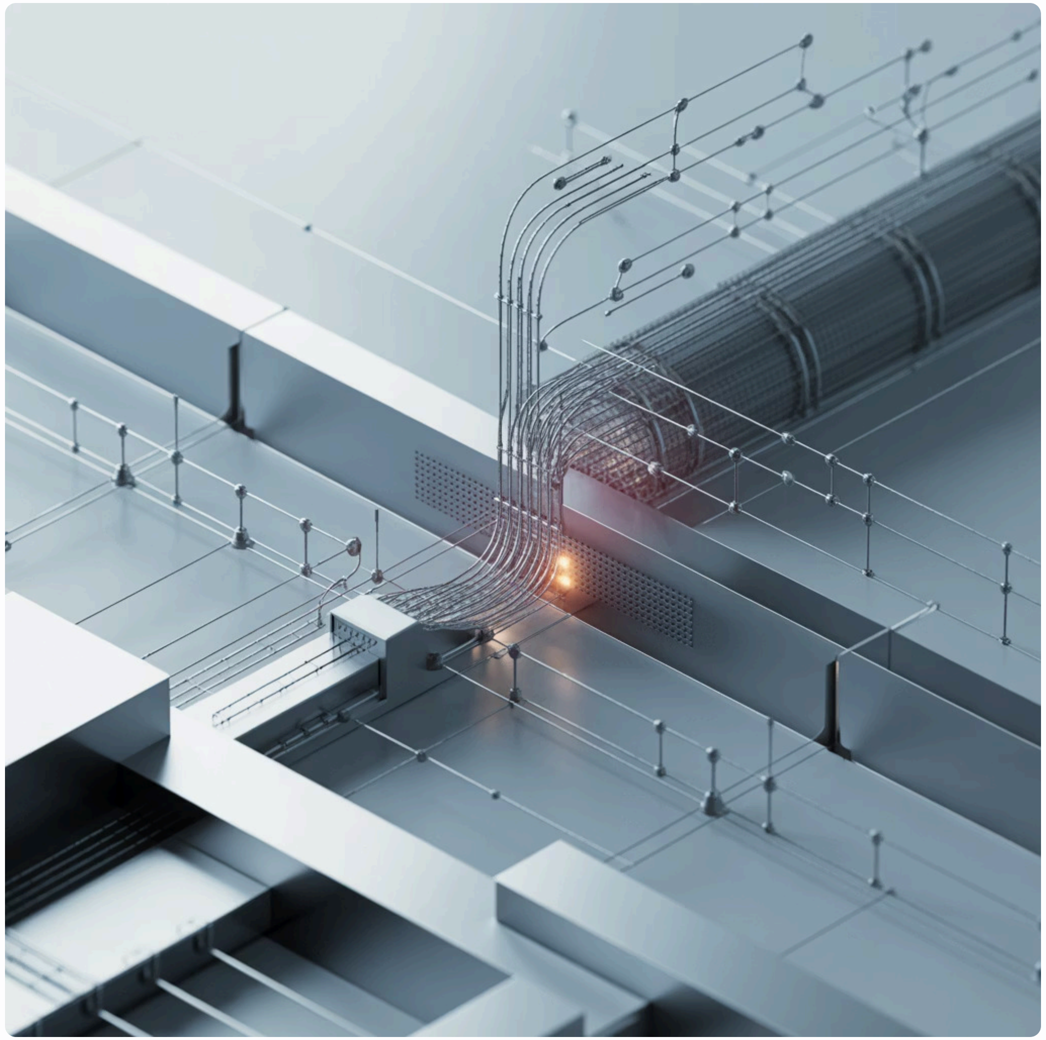


Chapter 2: Network Segmentation

Network Policies

Implementing zero-trust networking in Kubernetes

The Default Network Problem



All Pods Can Talk to Each Other

By default, Kubernetes allows unrestricted pod-to-pod communication across namespaces. This means a compromised pod in the development namespace could potentially access production databases.

At TechVault, their payment processing pods were initially accessible from their marketing analytics pods—a serious compliance violation that could have resulted in PCI-DSS audit failure.

Network Policy Fundamentals

Network policies act as distributed firewalls within your Kubernetes cluster. They define rules for pod-to-pod communication using label selectors, enabling microsegmentation at the application layer.

Default Deny

Start with deny-all policies

Explicit Allow

Whitelist required connections

Namespace Isolation

Segment by environment

Monitor & Audit

Track policy violations

TechVault's Network Policy Strategy

Sarah's team implemented a three-tier network segmentation model:

Tier 1: Public-Facing

- Ingress controllers
- Web application pods
- Can receive external traffic
- Limited egress to Tier 2 only

Tier 2: Application Logic

- Business logic microservices
- API gateways
- No direct external access
- Controlled access to Tier 3

Tier 3: Data Layer

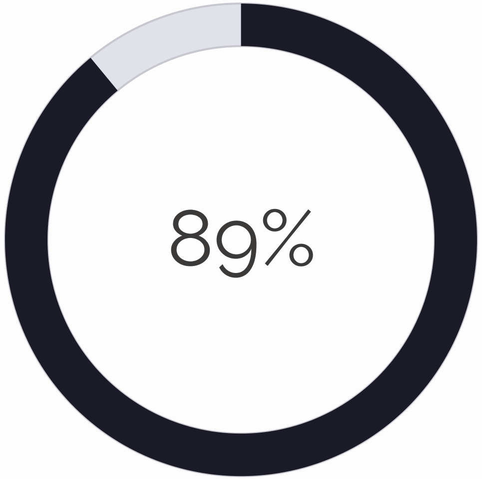
- Database pods
- Message queues
- Accepts connections from Tier 2 only
- No internet egress allowed

Example: Payment Service Network Policy

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: payment-service-policy
  namespace: production
spec:
  podSelector:
    matchLabels:
      app: payment-processor
  policyTypes:
  - Ingress
  - Egress
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          env: production
    - podSelector:
        matchLabels:
          tier: application
    ports:
    - protocol: TCP
      port: 8080
  egress:
  - to:
    - podSelector:
        matchLabels:
          app: postgres
    ports:
    - protocol: TCP
      port: 5432
```

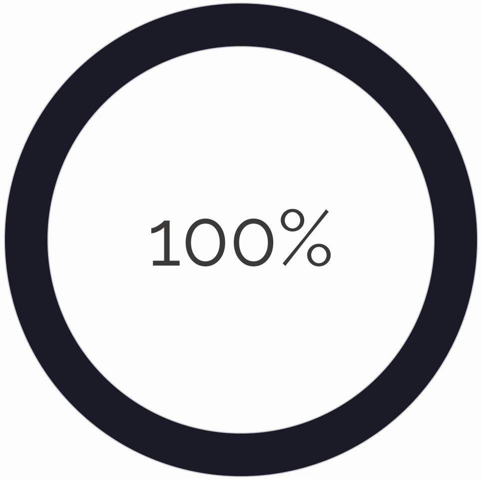
This policy ensures payment processors only accept connections from production application-tier pods and can only reach the PostgreSQL database.

Network Policy Impact at TechVault



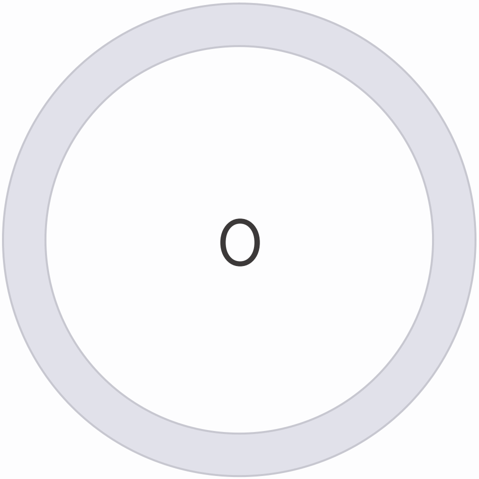
Attack Surface Reduction

Unauthorized pod communication eliminated



Namespace Isolation

Production fully isolated from dev/staging



Cross-Namespace Breaches

Since implementation six months ago

Chapter 3: Encrypted Traffic

TLS Ingress

Protecting data in transit with proper certificate management

Why TLS Ingress Matters

The Risk

Without TLS termination at the ingress layer, sensitive data flows unencrypted between external clients and your services. Man-in-the-middle attacks can intercept credentials, session tokens, and business data.

TechVault initially deployed ingress controllers without TLS, exposing customer payment information during transmission—a critical security finding discovered during their penetration test.

The Solution

Implement TLS termination at ingress using certificate management tools like cert-manager. This ensures all external traffic is encrypted before entering your cluster, with automatic certificate renewal to prevent expiration incidents.

Setting Up Secure Ingress

Install cert-manager

Deploy cert-manager to automate certificate lifecycle management using Let's Encrypt or your internal CA.

Configure ClusterIssuer

Define how certificates should be issued and which certificate authority to use for your domains.

Create TLS Secrets

Let cert-manager automatically provision and store TLS certificates as Kubernetes secrets.

Configure Ingress Resources

Reference TLS secrets in ingress definitions and enforce HTTPS redirection for all routes.

Monitor Certificate Expiry

Set up alerts for certificate renewal failures and expiration warnings to prevent outages.

TechVault's Ingress Configuration

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: payment-api-ingress
  annotations:
    cert-manager.io/cluster-issuer: "letsencrypt-prod"
    nginx.ingress.kubernetes.io/ssl-redirect: "true"
    nginx.ingress.kubernetes.io/force-ssl-redirect: "true"
spec:
  tls:
  - hosts:
    - api.techvault.com
    secretName: payment-api-tls
  rules:
  - host: api.techvault.com
    http:
      paths:
      - path: /payments
        pathType: Prefix
        backend:
          service:
            name: payment-service
            port:
              number: 443
```

This configuration automatically provisions TLS certificates and enforces HTTPS for all payment API traffic.

TLS Best Practices

Use Strong Cipher Suites

Configure ingress controllers to reject weak ciphers and enforce TLS 1.2 or higher. Disable deprecated protocols like TLS 1.0 and 1.1.

Implement Certificate Pinning

For sensitive services, use HTTP Public Key Pinning (HPKP) or similar mechanisms to prevent certificate substitution attacks.

Enable HSTS Headers

Force browsers to use HTTPS by enabling HTTP Strict Transport Security with appropriate max-age values and includeSubDomains.

Monitor Certificate Health

Set up Prometheus metrics for certificate expiration and renewal status. Alert 30 days before expiration with escalating urgency.

Chapter 4: Metadata Security

Node Protection

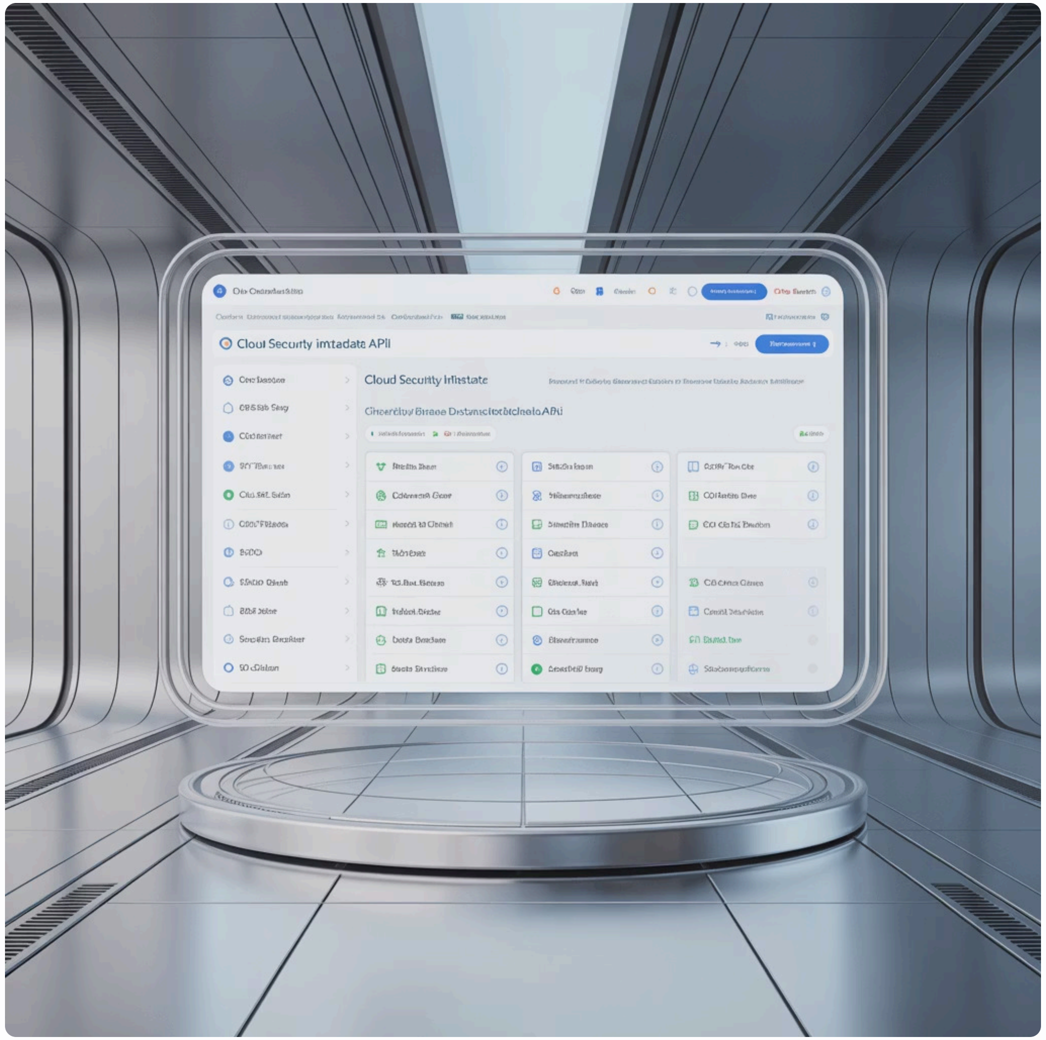
Preventing metadata exploitation and endpoint exposure

The Metadata API Threat

Cloud Provider Metadata APIs

Cloud platforms like AWS, Azure, and GCP expose metadata endpoints that provide instance credentials, IAM roles, and configuration data. If pods can access these endpoints, compromised containers can escalate privileges or access cloud resources.

At TechVault running on AWS, a misconfigured pod accessed the instance metadata service at 169.254.169.254, obtaining temporary credentials with excessive S3 permissions.



Protecting Node Metadata

Block Metadata Endpoints

Use network policies to prevent pods from accessing 169.254.169.254 and other metadata IP ranges. Deploy this as a cluster-wide default policy.

Use IMDSv2 on AWS

Require session tokens for metadata access by enforcing IMDSv2, which prevents simple HTTP GET requests from succeeding.

Implement Workload Identity

Use Workload Identity (GKE) or IAM Roles for Service Accounts (EKS) to provide scoped cloud credentials instead of node-level access.

Securing Kubernetes Endpoints

Beyond cloud metadata, Kubernetes itself exposes sensitive endpoints that require protection:

API Server Endpoint

Restrict access to the API server to authorized IP ranges only. Never expose kube-apiserver directly to the internet without strong authentication.

Kubelet Read-Only Port

Disable the kubelet read-only port (10255) which exposes pod and node information without authentication. Set --read-only-port=0 in kubelet configuration.

Dashboard Access

If using Kubernetes Dashboard, place it behind authentication proxy, restrict access via RBAC, and never grant cluster-admin permissions.

TechVault's Metadata Protection

```yaml
# Network policy blocking metadata access
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: block-metadata-access
  namespace: default
spec:
  podSelector: {}
  policyTypes:
  - Egress
  egress:
  - to:
    - ipBlock:
        cidr: 0.0.0.0/0
        except:
        - 169.254.169.254/32
        - 169.254.0.0/16
```

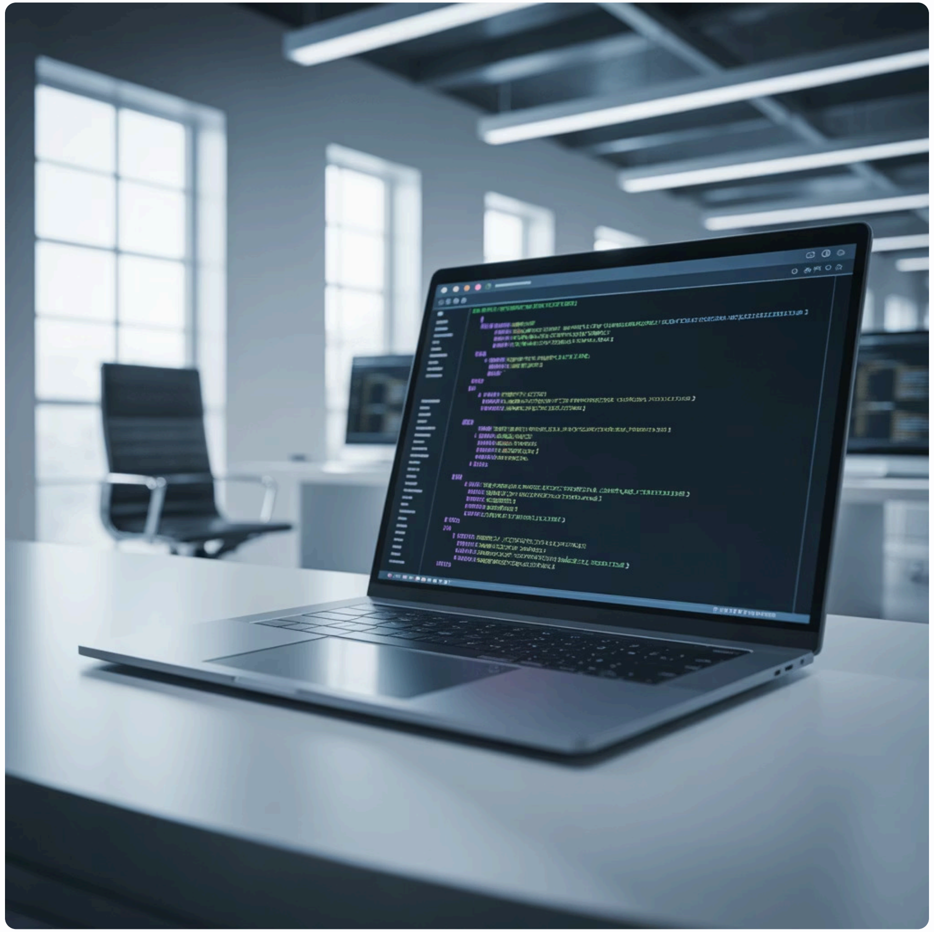
This global policy prevents all pods from accessing cloud metadata endpoints while allowing normal internet egress.

Chapter 5: Supply Chain Security

Binary Verification

Ensuring platform integrity through cryptographic validation

The Software Supply Chain Risk



Trust but Verify

Kubernetes binaries, container images, and Helm charts could be tampered with during distribution. Without verification, you might deploy compromised components into production.

TechVault learned this lesson when a developer accidentally deployed an unsigned third-party image that contained cryptomining malware, consuming significant CPU resources before detection.

Binary Verification Strategy

1 Verify Kubernetes Binaries

Download checksums and GPG signatures for kubectl, kubelet, and other components. Verify signatures match official Kubernetes signing keys before deployment.

2 Implement Image Signing

Use tools like Cosign or Notary to sign container images during CI/CD. Configure admission controllers to reject unsigned images at deployment time.

3 Scan for Vulnerabilities

Integrate container image scanning with Trivy, Clair, or Anchore in your pipeline. Block deployment of images with critical CVEs.

4 Use Private Registries

Mirror approved images to internal registries. Prevent pods from pulling directly from public registries like Docker Hub without review.

Image Policy Enforcement

TechVault implemented admission webhooks to enforce image policies cluster-wide:

1. **Only allow images from approved registries**

   Reject deployments pulling from docker.io or other public registries unless explicitly whitelisted by security team.

2. **Require image signatures**

   Using Sigstore policy controller, verify Cosign signatures exist and match approved signing keys before admission.

3. **Enforce scan status**

   Query vulnerability database and block images with critical or high severity CVEs from deployment to production namespaces.

Chapter 6: Access Control

RBAC & Service Accounts

Implementing least privilege through identity management