Secure Internal Application Hosting Guide

Kubernetes Implementation Framework
for Internal-Only Deployments

Companion to Comprehensive Cloud-Native Architecture Implementation Guide

Zero-Trust Security Controls
Network Isolation • Identity Management
Workload Hardening • Audit & Compliance

Technical Reference Guide

On-Premises and Private Network Deployments

January 19, 2026

Contents

1 Executive Summary 2
- 1.1 Document Purpose 2
- 1.2 Key Insight: "Internal" Does Not Mean "Trusted" 2
- 1.3 Scope and Target Audience 2
- 1.4 Security Maturity Phases 3

2 Official Documentation Resources 3
- 2.1 Core Security Documentation 3
- 2.2 Service Mesh Security 4
- 2.3 Secrets Management 4
- 2.4 Policy and Compliance 5
- 2.5 Monitoring and Audit 5
- 2.6 Container Security 5
- 2.7 Backup and Disaster Recovery 5

3 Security Maturity Model 6
- 3.1 Phase 2: Internal Pilot Readiness (Weeks 5-8) 6
- 3.2 Phase 3: Secure Production Readiness (Weeks 9-12) 7

4 Core Security Controls Implementation 9
- 4.1 Identity & Access Control 9
  - 4.1.1 Namespace Isolation Strategy 9
  - 4.1.2 RBAC Configuration 10
  - 4.1.3 Authentication Integration 12
- 4.2 Network Isolation 13
  - 4.2.1 CNI Selection and Configuration 13
  - 4.2.2 Default-Deny NetworkPolicies (Phase 3) 13
  - 4.2.3 Ingress Controller with TLS 15
- 4.3 Workload Security Baseline 16
  - 4.3.1 Pod Security Standards 16
  - 4.3.2 Security Context Configuration 17
  - 4.3.3 Admission Controllers (Phase 3) 18
- 4.4 Secrets Management 19
  - 4.4.1 Phase 2: Kubernetes Secrets 19
  - 4.4.2 Phase 3: External Secrets with Vault 20
- 4.5 Transport Security 22
  - 4.5.1 Service Mesh Deployment (Phase 3) 22
- 4.6 Audit & Compliance 24
  - 4.6.1 Kubernetes Audit Logging 24
  - 4.6.2 Runtime Security Monitoring (Falco) 25
- 4.7 Backup & Disaster Recovery 26
  - 4.7.1 Velero Installation and Configuration 26
  - 4.7.2 Database Backup Strategy 27

5 Implementation Roadmap **29**
5.1 Phase 2: Internal Pilot Readiness (Weeks 5-8) . . . 29
5.1.1 Week 5: Namespace and RBAC Setup . . . 29
5.1.2 Week 6: Network Policies and Ingress . . . 29
5.1.3 Week 7: Workload Security and Secrets . . . 29
5.1.4 Week 8: Observability and Backup . . . 30
5.2 Phase 3: Secure Production Readiness (Weeks 9-12) . . . 31
5.2.1 Week 9: Service Mesh Deployment . . . 31
5.2.2 Week 10: Zero-Trust Network Policies . . . 31
5.2.3 Week 11: Workload Hardening and Policy Enforcement . . . 31
5.2.4 Week 12: External Secrets and Runtime Security . . . 32

6 Validation and Testing **33**
6.1 Security Control Validation . . . 33
6.1.1 RBAC Validation . . . 33
6.1.2 NetworkPolicy Validation . . . 34
6.1.3 Pod Security Standards Validation . . . 34
6.1.4 mTLS Validation . . . 34
6.2 Disaster Recovery Testing . . . 35
6.2.1 Namespace Restore Test . . . 35
6.2.2 Database Restore Test . . . 35
6.3 Compliance Audit Validation . . . 35
6.3.1 Generate Compliance Report . . . 35

7 Common Pitfalls and Solutions **36**
7.1 Identity & Access Pitfalls . . . 36
7.2 Network Isolation Pitfalls . . . 36
7.3 Workload Security Pitfalls . . . 37
7.4 Secrets Management Pitfalls . . . 38
7.5 Observability Pitfalls . . . 38

8 Production Readiness Checklist **38**
8.1 Identity & Access Control . . . 39
8.2 Network Isolation . . . 39
8.3 Workload Security . . . 39
8.4 Secrets Management . . . 40
8.5 Transport Security . . . 40
8.6 Audit & Compliance . . . 40
8.7 Container Supply Chain . . . 41
8.8 Backup & Disaster Recovery . . . 41
8.9 Operational Readiness . . . 41

9 Continuous Security Improvement **42**
9.1 Regular Security Reviews . . . 42
9.2 Metrics and KPIs . . . 42
9.3 Technology Evolution . . . 43

10 Conclusion **43**
10.1 Key Takeaways 43
10.2 Next Steps 43
10.3 Additional Resources 44

1 Executive Summary

This guide provides a comprehensive implementation framework for securely hosting internal applications on Kubernetes in on-premises or private network environments. It serves as a companion to the *Comprehensive Cloud-Native Architecture Implementation Guide*, focusing specifically on the security controls, hardening measures, and operational practices required to achieve production-ready internal application hosting.

1.1 Document Purpose

This implementation guide serves as:

1. **Security Maturity Roadmap**: Clear progression from pilot to production-ready internal hosting
2. **Zero-Trust Implementation Framework**: Practical controls for internal environments that reject "trusted network" assumptions
3. **Go/No-Go Decision Guide**: Concrete validation criteria for each security maturity phase
4. **Hardening Playbook**: Step-by-step implementation of Kubernetes security controls
5. **Compliance Accelerator**: Audit-ready configurations for internal hosting requirements

1.2 Key Insight: "Internal" Does Not Mean "Trusted"

Critical Security Requirement

Critical Security Principle:
Your main cloud-native guide explicitly warns against the "Trusting Internal Network" pitfall (Section 10.6). This guide operationalizes that warning: **internal hosting requires the same zero-trust controls as external-facing systems**.
Lateral movement after a breach is a primary attack vector. Network location ("internal") is not a security boundary.

1.3 Scope and Target Audience

In Scope:

- Internal-only applications on Kubernetes (on-premises or private cloud)
- Stateful applications requiring persistent storage and databases
- Zero-trust security implementation for internal networks
- Production-grade security controls for non-internet-facing workloads
- Compliance and audit requirements for internal hosting

Out of Scope:

- Internet-facing applications with public ingress

- Multi-cloud or hybrid cloud deployments (covered in main guide)
- Specific application configurations (GLPI, osTicket, etc.)
- Development or testing environments without production security requirements

Target Audience:

- Platform engineers deploying internal Kubernetes clusters
- DevOps teams hosting internal applications
- Security engineers implementing zero-trust controls
- IT teams migrating legacy internal applications to Kubernetes
- Compliance officers validating internal hosting security

1.4 Security Maturity Phases

This guide defines two security maturity phases aligned with the main implementation roadmap:

Phase	Timeframe	Readiness Level
Phase 2	Weeks 5-8	**Internal Pilot Readiness** Minimum viable security for limited internal users Basic RBAC, NetworkPolicies, secrets management
Phase 3	Weeks 9-12	**Secure Production Readiness** Zero-trust controls, mTLS, hardened workloads Full compliance and audit capabilities

Table 1: Security Maturity Phases

Critical Distinction: Phase 2 allows you to *start* internal hosting with basic controls. Phase 3 is where you *harden* to production-grade security that defends against lateral movement and privilege escalation.

2 Official Documentation Resources

The following official documentation should be used alongside this guide for implementation details. These resources provide authoritative references for security controls and hardening measures.

2.1 Core Security Documentation

Kubernetes Security:

- Main Security Concepts: https://kubernetes.io/docs/concepts/security/
- Pod Security Standards: https://kubernetes.io/docs/concepts/security/pod-security-standards/
- RBAC Authorization: https://kubernetes.io/docs/reference/access-authn-authz/rbac/
- Network Policies: https://kubernetes.io/docs/concepts/services-networking/network-policies/

- Secrets Management: https://kubernetes.io/docs/concepts/configuration/secret/
- Audit Logging: https://kubernetes.io/docs/tasks/debug/debug-cluster/audit/
- Encrypting Data at Rest: https://kubernetes.io/docs/tasks/administer-cluster/encrypt-data/
- Authentication: https://kubernetes.io/docs/reference/access-authn-authz/authentication/
- Multi-Tenancy: https://kubernetes.io/docs/concepts/security/multi-tenancy/

2.2 Service Mesh Security

Istio:

- Security Concepts: https://istio.io/latest/docs/concepts/security/
- Mutual TLS: https://istio.io/latest/docs/tasks/security/authentication/mtls-migration/
- Authorization Policies: https://istio.io/latest/docs/tasks/security/authorization/
- Certificate Management: https://istio.io/latest/docs/tasks/security/cert-management/

Linkerd:

- Automatic mTLS: https://linkerd.io/2/features/automatic-mtls/
- Policy Documentation: https://linkerd.io/2/features/server-policy/

2.3 Secrets Management

External Secrets Operator:

- Main Documentation: https://external-secrets.io/
- Getting Started: https://external-secrets.io/latest/introduction/getting-started/
- Vault Provider: https://external-secrets.io/latest/provider/hashicorp-vault/

HashiCorp Vault:

- Vault Documentation: https://www.vaultproject.io/docs
- Kubernetes Auth: https://www.vaultproject.io/docs/auth/kubernetes
- Kubernetes Secrets Engine: https://www.vaultproject.io/docs/secrets/kv

Sealed Secrets:

- GitHub Repository: https://github.com/bitnami-labs/sealed-secrets

2.4 Policy and Compliance

Open Policy Agent (OPA):

- OPA Documentation: https://www.openpolicyagent.org/docs/latest/
- Kubernetes Integration: https://www.openpolicyagent.org/docs/latest/kubernetes-introduction/

Gatekeeper:

- Gatekeeper Documentation: https://open-policy-agent.github.io/gatekeeper/
- Policy Library: https://github.com/open-policy-agent/gatekeeper-library

Kyverno:

- Kyverno Documentation: https://kyverno.io/docs/
- Policy Samples: https://kyverno.io/policies/

2.5 Monitoring and Audit

Falco (Runtime Security):

- Falco Documentation: https://falco.org/docs/
- Kubernetes Rules: https://github.com/falcosecurity/rules

Prometheus:

- Security Best Practices: https://prometheus.io/docs/operating/security/

2.6 Container Security

Image Scanning:

- Trivy: https://github.com/aquasecurity/trivy
- Grype: https://github.com/anchore/grype
- Clair: https://github.com/quay/clair

Image Signing:

- Cosign: https://docs.sigstore.dev/cosign/overview/
- Notary: https://github.com/notaryproject/notary

2.7 Backup and Disaster Recovery

Velero:

- Main Documentation: https://velero.io/docs/
- How Velero Works: https://velero.io/docs/main/how-velero-works/

3 Security Maturity Model

This section defines the two security maturity phases for internal application hosting, aligned with the main implementation roadmap.

3.1 Phase 2: Internal Pilot Readiness (Weeks 5-8)

Security Posture: Minimum viable security controls for limited internal pilot deployment.
Acceptable Risk Profile:

- Limited user base (¡50 users)
- Non-critical applications or pilot deployments
- Controlled network environment with monitoring
- Ability to rapidly patch or take offline if issues discovered

Core Controls Implemented:

1. **Basic Identity & Access Control**
   - Namespace isolation per application
   - Kubernetes RBAC with least-privilege service accounts
   - User authentication via OIDC or LDAP
2. **Network Segmentation**
   - CNI with NetworkPolicy support (Calico, Cilium, or equivalent)
   - Basic NetworkPolicies (not yet default-deny)
   - Ingress controller with TLS termination
3. **Workload Security Baseline**
   - Pod Security Standards enforced at `baseline` level
   - Basic security contexts (non-root where feasible)
   - Resource requests and limits defined
4. **Secrets Management**
   - Kubernetes Secrets (base64 encoded)
   - No secrets in Git repositories
   - Manual secret rotation procedures
5. **Basic Observability**
   - Kubernetes audit logging enabled
   - Basic metrics collection (Prometheus)
   - Centralized logging (Loki or equivalent)
6. **Persistent Storage**

- PersistentVolumes with appropriate access modes
- StorageClass with volume expansion enabled
- Manual backup procedures documented

Go/No-Go Checkpoint

Phase 2 Go/No-Go Criteria:
Before deploying pilot applications, verify:

- ☐ Namespace created with ResourceQuotas
- ☐ RBAC roles defined with least privilege
- ☐ NetworkPolicies allow only required ingress/egress
- ☐ Pod Security Standards enforced (minimum: baseline)
- ☐ All Secrets created (no hardcoded credentials)
- ☐ Audit logging enabled and tested
- ☐ Backup procedure documented and tested
- ☐ Incident response contacts defined

3.2 Phase 3: Secure Production Readiness (Weeks 9-12)

Security Posture: Zero-trust hardened deployment suitable for production internal applications.

Acceptable Risk Profile:

- Full production deployment (any number of users)
- Critical internal applications
- Compliance and audit requirements
- Protection against lateral movement and privilege escalation

Additional Controls Beyond Phase 2:

1. **Zero-Trust Network Security**
   - Service mesh deployed (Istio or Linkerd)
   - Automatic mTLS for all service-to-service communication
   - Default-deny NetworkPolicies with explicit allow rules
   - Authorization policies at Layer 7
2. **Hardened Workload Security**
   - Pod Security Standards enforced at `restricted` level
   - Comprehensive security contexts (read-only root filesystem, drop all capabilities)
   - AppArmor or SELinux profiles where supported

- Admission controllers enforcing security policies (OPA Gatekeeper or Kyverno)

3. **Advanced Secrets Management**
    - External Secrets Operator integrated with Vault
    - Automatic secret rotation
    - Encryption at rest for etcd
    - Secrets encryption in transit via service mesh
4. **Comprehensive Audit & Compliance**
    - Full Kubernetes audit logging with policy
    - Centralized log retention (minimum 90 days)
    - Runtime security monitoring (Falco)
    - Regular compliance scanning
5. **Container Supply Chain Security**
    - Image scanning in CI/CD pipeline
    - Vulnerability blocking policies
    - Image signing and verification
    - Software Bill of Materials (SBOM) generation
6. **Production Backup & Recovery**
    - Automated backup with Velero
    - Regular disaster recovery testing
    - Multi-zone deployment for availability
    - Documented RTO/RPO objectives

Go/No-Go Checkpoint

Phase 3 Go/No-Go Criteria:
Before declaring production-ready, verify:

- ☐ Service mesh deployed with mTLS enabled
- ☐ Default-deny NetworkPolicies in place
- ☐ Pod Security Standards enforced at `restricted`
- ☐ External Secrets Operator operational
- ☐ Vault integration tested and documented
- ☐ Admission controller policies enforced
- ☐ Runtime security monitoring active (Falco)
- ☐ Image scanning integrated in CI/CD
- ☐ Automated backup tested and verified
- ☐ Disaster recovery procedure tested
- ☐ Security incident response plan documented
- ☐ Compliance audit documentation complete

4 Core Security Controls Implementation

This section provides detailed implementation guidance for the seven core security control areas required for secure internal application hosting.

4.1 Identity & Access Control

4.1.1 Namespace Isolation Strategy

Principle: Each application or service group should have its own namespace to provide logical isolation and RBAC boundaries.

Implementation Pattern:

```
# Create namespace for application
kubectl create namespace app-production

# Apply ResourceQuota to prevent resource exhaustion
kubectl apply -f - <<EOF
apiVersion: v1
kind: ResourceQuota
metadata:
  name: app-production-quota
  namespace: app-production
spec:
  hard:
```

```
    requests.cpu: "10"
    requests.memory: 20Gi
    persistentvolumeclaims: "5"
    services.loadbalancers: "0"  # No external LBs for internal apps
EOF

# Apply LimitRange for default resource constraints
kubectl apply -f - <<EOF
apiVersion: v1
kind: LimitRange
metadata:
  name: app-production-limits
  namespace: app-production
spec:
  limits:
  - default:
      cpu: "500m"
      memory: "512Mi"
    defaultRequest:
      cpu: "100m"
      memory: "128Mi"
    type: Container
EOF
```

Reference:

- Namespaces: https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/
- ResourceQuotas: https://kubernetes.io/docs/concepts/policy/resource-quotas/
- LimitRanges: https://kubernetes.io/docs/concepts/policy/limit-range/

4.1.2 RBAC Configuration

Principle: Apply least-privilege access using Role-Based Access Control. Separate human user access from service account access.

Service Account Pattern:

```
# Create dedicated ServiceAccount for application
kubectl apply -f - <<EOF
apiVersion: v1
kind: ServiceAccount
metadata:
  name: app-service-account
  namespace: app-production
automountServiceAccountToken: false  # Explicit mounting only
EOF

# Create Role with minimal permissions
kubectl apply -f - <<EOF
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
```

```
  name: app-role
  namespace: app-production
rules:
- apiGroups: [""]
  resources: ["configmaps"]
  verbs: ["get", "list"]
- apiGroups: [""]
  resources: ["secrets"]
  verbs: ["get"]
  resourceNames: ["app-database-secret"]  # Specific secret only
EOF

# Bind Role to ServiceAccount
kubectl apply -f - <<EOF
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: app-rolebinding
  namespace: app-production
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: app-role
subjects:
- kind: ServiceAccount
  name: app-service-account
  namespace: app-production
EOF
```

Human User Access Pattern (Cluster Admin):

```
# ClusterRole for cluster administrators
kubectl apply -f - <<EOF
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: cluster-admin-role
rules:
- apiGroups: ["*"]
  resources: ["*"]
  verbs: ["*"]
EOF

# ClusterRoleBinding for admin group (via OIDC)
kubectl apply -f - <<EOF
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: cluster-admin-binding
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin-role
subjects:
- kind: Group
```

```
  name: "system:masters"  # OIDC group
  apiGroup: rbac.authorization.k8s.io
EOF
```

Human User Access Pattern (Developer):

```
# Role for developers (namespace-scoped)
kubectl apply -f - <<EOF
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: developer-role
  namespace: app-production
rules:
- apiGroups: ["", "apps", "batch"]
  resources: ["pods", "pods/log", "deployments", "jobs"]
  verbs: ["get", "list", "watch"]
- apiGroups: [""]
  resources: ["pods/exec"]
  verbs: ["create"]  # Allow kubectl exec for debugging
EOF

# RoleBinding for developer group
kubectl apply -f - <<EOF
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: developer-binding
  namespace: app-production
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: developer-role
subjects:
- kind: Group
  name: "developers"  # OIDC group
  apiGroup: rbac.authorization.k8s.io
EOF
```

Reference:

- RBAC Overview: https://kubernetes.io/docs/reference/access-authn-authz/rbac/
- Service Accounts: https://kubernetes.io/docs/concepts/security/service-accounts/
- OIDC Authentication: https://kubernetes.io/docs/reference/access-authn-authz/authentication/#openid-connect-tokens

4.1.3 Authentication Integration

OIDC Configuration (Recommended):

Configure the Kubernetes API server to use OIDC for user authentication:

```
# API server flags for OIDC (add to kube-apiserver manifest)
--oidc-issuer-url=https://your-identity-provider.com
```

```
--oidc-client-id=kubernetes
--oidc-username-claim=email
--oidc-groups-claim=groups
--oidc-ca-file=/etc/kubernetes/pki/oidc-ca.crt
```

Reference:

- Authenticating: https://kubernetes.io/docs/reference/access-authn-authz/authentication/
- OIDC Authenticator: https://kubernetes.io/docs/reference/access-authn-authz/authentication/#openid-connect-tokens

4.2 Network Isolation

4.2.1 CNI Selection and Configuration

Principle: Use a CNI that supports NetworkPolicy enforcement. Calico and Cilium are recommended for their security features.

Calico Installation (Recommended for Internal Networks):

```
# Install Calico CNI
kubectl apply -f https://docs.projectcalico.org/manifests/calico.yaml

# Verify Calico is running
kubectl get pods -n kube-system | grep calico

# Enable NetworkPolicy support
kubectl apply -f - <<EOF
apiVersion: crd.projectcalico.org/v1
kind: FelixConfiguration
metadata:
  name: default
spec:
  defaultEndpointToHostAction: ACCEPT
  failsafeInboundHostPorts:
  - protocol: tcp
    port: 22
  - protocol: tcp
    port: 6443  # API server
EOF
```

Reference:

- Calico Documentation: https://docs.tigera.io/calico/latest/about/
- Cilium Documentation: https://docs.cilium.io/
- Network Plugins: https://kubernetes.io/docs/concepts/extend-kubernetes/compute-storage-net/network-plugins/

4.2.2 Default-Deny NetworkPolicies (Phase 3)

Principle: Start with default-deny and explicitly allow only required traffic.

Critical Security Requirement

Implementation Warning:
Implementing default-deny NetworkPolicies requires careful planning. Test in a non-production namespace first. Document all required network flows before applying.

Default-Deny Policy:

```
# Deny all ingress and egress by default
kubectl apply -f - <<EOF
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny-all
  namespace: app-production
spec:
  podSelector: {}
  policyTypes:
  - Ingress
  - Egress
EOF
```

Allow Ingress from Ingress Controller:

```
kubectl apply -f - <<EOF
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-from-ingress
  namespace: app-production
spec:
  podSelector:
    matchLabels:
      app: web-app
  policyTypes:
  - Ingress
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          name: ingress-nginx
    ports:
    - protocol: TCP
      port: 8080
EOF
```

Allow Egress to Database:

```
kubectl apply -f - <<EOF
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-to-database
  namespace: app-production
spec:
```

```
  podSelector:
    matchLabels:
      app: web-app
  policyTypes:
  - Egress
  egress:
  - to:
    - podSelector:
        matchLabels:
          app: database
    ports:
    - protocol: TCP
      port: 5432
  # Allow DNS resolution
  - to:
    - namespaceSelector:
        matchLabels:
          name: kube-system
    - podSelector:
        matchLabels:
          k8s-app: kube-dns
    ports:
    - protocol: UDP
      port: 53
EOF
```

Reference:

- NetworkPolicy: https://kubernetes.io/docs/concepts/services-networking/network-policies/
- Calico NetworkPolicy: https://docs.tigera.io/calico/latest/network-policy/

4.2.3 Ingress Controller with TLS

NGINX Ingress Controller Installation:

```
# Install NGINX Ingress Controller
kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-
    nginx/controller-v1.8.1/deploy/static/provider/baremetal/deploy.yaml

# Create TLS Secret (use cert-manager for automation)
kubectl create secret tls app-tls-cert \
  --cert=path/to/tls.crt \
  --key=path/to/tls.key \
  -n app-production
```

Ingress Resource with TLS:

```
kubectl apply -f - <<EOF
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: app-ingress
  namespace: app-production
  annotations:
```

```
    nginx.ingress.kubernetes.io/ssl-redirect: "true"
    nginx.ingress.kubernetes.io/force-ssl-redirect: "true"
spec:
  ingressClassName: nginx
  tls:
  - hosts:
    - app.internal.example.com
    secretName: app-tls-cert
  rules:
  - host: app.internal.example.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: app-service
            port:
              number: 80
EOF
```

Reference:

- NGINX Ingress: https://kubernetes.github.io/ingress-nginx/
- Ingress TLS: https://kubernetes.io/docs/concepts/services-networking/ingress/#tls
- cert-manager: https://cert-manager.io/docs/

4.3 Workload Security Baseline

4.3.1 Pod Security Standards

Principle: Enforce Pod Security Standards to restrict pod privileges.

Phase 2: Baseline Policy

```
# Apply baseline policy to namespace
kubectl label namespace app-production \
  pod-security.kubernetes.io/enforce=baseline \
  pod-security.kubernetes.io/audit=restricted \
  pod-security.kubernetes.io/warn=restricted
```

Phase 3: Restricted Policy

```
# Enforce restricted policy
kubectl label namespace app-production \
  pod-security.kubernetes.io/enforce=restricted \
  pod-security.kubernetes.io/audit=restricted \
  pod-security.kubernetes.io/warn=restricted --overwrite
```

Reference:

- Pod Security Standards: https://kubernetes.io/docs/concepts/security/pod-security-standards/
- Pod Security Admission: https://kubernetes.io/docs/concepts/security/pod-security-admission/

4.3.2 Security Context Configuration

Restricted SecurityContext Pattern:

```
kubectl apply -f - <<EOF
apiVersion: apps/v1
kind: Deployment
metadata:
  name: secure-app
  namespace: app-production
spec:
  replicas: 3
  selector:
    matchLabels:
      app: secure-app
  template:
    metadata:
      labels:
        app: secure-app
    spec:
      serviceAccountName: app-service-account
      securityContext:
        runAsNonRoot: true
        runAsUser: 10001
        fsGroup: 10001
        seccompProfile:
          type: RuntimeDefault
      containers:
      - name: app
        image: your-registry/app:v1.0.0
        securityContext:
          allowPrivilegeEscalation: false
          capabilities:
            drop:
            - ALL
          readOnlyRootFilesystem: true
          runAsNonRoot: true
          runAsUser: 10001
        resources:
          requests:
            memory: "256Mi"
            cpu: "100m"
          limits:
            memory: "512Mi"
            cpu: "500m"
        volumeMounts:
        - name: tmp
          mountPath: /tmp
        - name: cache
          mountPath: /app/cache
      volumes:
      - name: tmp
        emptyDir: {}
      - name: cache
        emptyDir: {}
```

```
EOF
```

Reference:

- Configure Security Context: https://kubernetes.io/docs/tasks/configure-pod-container/security-context/
- Seccomp: https://kubernetes.io/docs/tutorials/security/seccomp/

4.3.3 Admission Controllers (Phase 3)

OPA Gatekeeper Installation:

```
# Install Gatekeeper
kubectl apply -f https://raw.githubusercontent.com/open-policy-agent/
    gatekeeper/master/deploy/gatekeeper.yaml

# Create ConstraintTemplate for required labels
kubectl apply -f - <<EOF
apiVersion: templates.gatekeeper.sh/v1
kind: ConstraintTemplate
metadata:
  name: k8srequiredlabels
spec:
  crd:
    spec:
      names:
        kind: K8sRequiredLabels
      validation:
        openAPIV3Schema:
          type: object
          properties:
            labels:
              type: array
              items:
                type: string
  targets:
    - target: admission.k8s.gatekeeper.sh
      rego: |
        package k8srequiredlabels

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

# Create Constraint requiring app and version labels
kubectl apply -f - <<EOF
apiVersion: constraints.gatekeeper.sh/v1beta1
kind: K8sRequiredLabels
metadata:
```

```
  name: require-app-version-labels
spec:
  match:
    kinds:
    - apiGroups: ["apps"]
      kinds: ["Deployment"]
  parameters:
    labels: ["app", "version"]
EOF
```

Reference:

- Gatekeeper: https://open-policy-agent.github.io/gatekeeper/
- OPA: https://www.openpolicyagent.org/docs/latest/kubernetes-introduction/
- Kyverno: https://kyverno.io/docs/

4.4 Secrets Management

4.4.1 Phase 2: Kubernetes Secrets

Basic Secrets Creation (Non-Production):

```
# Create secret from literal values
kubectl create secret generic app-database-secret \
  --from-literal=username=appuser \
  --from-literal=password='SecureP@ssw0rd!' \
  -n app-production

# Create secret from file
kubectl create secret generic app-tls-cert \
  --from-file=tls.crt=path/to/tls.crt \
  --from-file=tls.key=path/to/tls.key \
  -n app-production
```

Critical Security Requirement

Critical Security Warning:
Kubernetes Secrets are only base64-encoded by default, NOT encrypted. For production deployments (Phase 3), you MUST use encryption at rest for etcd AND external secrets management (Vault).

Enable Encryption at Rest (Required for Phase 3):

```
# Create EncryptionConfiguration
cat > /etc/kubernetes/encryption-config.yaml <<EOF
apiVersion: apiserver.config.k8s.io/v1
kind: EncryptionConfiguration
resources:
  - resources:
    - secrets
    providers:
    - aescbc:
```

```
        keys:
        - name: key1
          secret: $(head -c 32 /dev/urandom | base64)
    - identity: {}
EOF

# Add to kube-apiserver manifest
--encryption-provider-config=/etc/kubernetes/encryption-config.yaml
```

Reference:

- Secrets: https://kubernetes.io/docs/concepts/configuration/secret/
- Encrypting Data at Rest: https://kubernetes.io/docs/tasks/administer-cluster/encrypt-data/

4.4.2 Phase 3: External Secrets with Vault

HashiCorp Vault Installation:

```
# Install Vault using Helm
helm repo add hashicorp https://helm.releases.hashicorp.com
helm install vault hashicorp/vault \
  --namespace vault \
  --create-namespace \
  --set server.dev.enabled=false \
  --set server.ha.enabled=true \
  --set server.ha.replicas=3

# Initialize Vault (save unseal keys and root token securely!)
kubectl exec -n vault vault-0 -- vault operator init

# Unseal Vault (repeat for all replicas)
kubectl exec -n vault vault-0 -- vault operator unseal <key1>
kubectl exec -n vault vault-0 -- vault operator unseal <key2>
kubectl exec -n vault vault-0 -- vault operator unseal <key3>
```

Configure Vault for Kubernetes:

```
# Enable Kubernetes auth
kubectl exec -n vault vault-0 -- vault auth enable kubernetes

# Configure Kubernetes auth
kubectl exec -n vault vault-0 -- vault write auth/kubernetes/config \
  kubernetes_host="https://$KUBERNETES_SERVICE_HOST:
      $KUBERNETES_SERVICE_PORT"

# Create policy for application
kubectl exec -n vault vault-0 -- vault policy write app-policy - <<EOF
path "secret/data/app-production/*" {
  capabilities = ["read"]
}
EOF

# Create role binding ServiceAccount to policy
kubectl exec -n vault vault-0 -- vault write auth/kubernetes/role/app-role
    \
```

```
  bound_service_account_names=app-service-account \
  bound_service_account_namespaces=app-production \
  policies=app-policy \
  ttl=24h
```

External Secrets Operator Installation:

```
# Install External Secrets Operator
helm repo add external-secrets https://charts.external-secrets.io
helm install external-secrets \
  external-secrets/external-secrets \
  -n external-secrets-system \
  --create-namespace

# Create SecretStore for Vault
kubectl apply -f - <<EOF
apiVersion: external-secrets.io/v1beta1
kind: SecretStore
metadata:
  name: vault-backend
  namespace: app-production
spec:
  provider:
    vault:
      server: "http://vault.vault:8200"
      path: "secret"
      version: "v2"
      auth:
        kubernetes:
          mountPath: "kubernetes"
          role: "app-role"
          serviceAccountRef:
            name: "app-service-account"
EOF

# Create ExternalSecret
kubectl apply -f - <<EOF
apiVersion: external-secrets.io/v1beta1
kind: ExternalSecret
metadata:
  name: app-database-external
  namespace: app-production
spec:
  refreshInterval: 1h
  secretStoreRef:
    name: vault-backend
    kind: SecretStore
  target:
    name: app-database-secret
    creationPolicy: Owner
  data:
  - secretKey: username
    remoteRef:
      key: app-production/database
      property: username
```

```
  - secretKey: password
    remoteRef:
      key: app-production/database
      property: password
EOF
```

Store Secret in Vault:

```
# Write secret to Vault
kubectl exec -n vault vault-0 -- vault kv put \
  secret/app-production/database \
  username=appuser \
  password='SecureP@ssw0rd!'
```

Reference:

- Vault Documentation: https://www.vaultproject.io/docs
- Vault on Kubernetes: https://developer.hashicorp.com/vault/docs/platform/k8s
- External Secrets: https://external-secrets.io/latest/
- Vault Provider: https://external-secrets.io/latest/provider/hashicorp-vault/

4.5 Transport Security

4.5.1 Service Mesh Deployment (Phase 3)

Istio Installation:

```
# Download Istio
curl -L https://istio.io/downloadIstio | sh -
cd istio-*
export PATH=$PWD/bin:$PATH

# Install Istio with default profile
istioctl install --set profile=default -y

# Enable sidecar injection for namespace
kubectl label namespace app-production istio-injection=enabled

# Verify installation
istioctl verify-install
```

Enable Strict mTLS:

```
# Require mTLS for all services in namespace
kubectl apply -f - <<EOF
apiVersion: security.istio.io/v1beta1
kind: PeerAuthentication
metadata:
  name: default
  namespace: app-production
spec:
  mtls:
    mode: STRICT
EOF
```

Authorization Policies:

```
# Deny all traffic by default
kubectl apply -f - <<EOF
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: deny-all
  namespace: app-production
spec:
  {}
EOF

# Allow ingress-gateway to web-app
kubectl apply -f - <<EOF
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: allow-ingress-to-web
  namespace: app-production
spec:
  selector:
    matchLabels:
      app: web-app
  action: ALLOW
  rules:
  - from:
    - source:
        principals: ["cluster.local/ns/istio-system/sa/istio-
           ingressgateway-service-account"]
EOF

# Allow web-app to database
kubectl apply -f - <<EOF
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: allow-web-to-db
  namespace: app-production
spec:
  selector:
    matchLabels:
      app: database
  action: ALLOW
  rules:
  - from:
    - source:
        principals: ["cluster.local/ns/app-production/sa/app-service-
           account"]
    to:
    - operation:
        ports: ["5432"]
EOF
```

Reference:

- Istio Installation: https://istio.io/latest/docs/setup/getting-started/
- Istio Security: https://istio.io/latest/docs/concepts/security/
- PeerAuthentication: https://istio.io/latest/docs/reference/config/security/peer_authentication/
- AuthorizationPolicy: https://istio.io/latest/docs/reference/config/security/authorization-poli

Linkerd Alternative (Lightweight):

```
# Install Linkerd CLI
curl --proto '=https' --tlsv1.2 -sSfL https://run.linkerd.io/install | sh

# Install Linkerd control plane
linkerd install --crds | kubectl apply -f -
linkerd install | kubectl apply -f -

# Verify installation
linkerd check

# Enable sidecar injection
kubectl annotate namespace app-production linkerd.io/inject=enabled

# Linkerd automatically enables mTLS between meshed pods
```

Reference:

- Linkerd Getting Started: https://linkerd.io/2/getting-started/
- Automatic mTLS: https://linkerd.io/2/features/automatic-mtls/

4.6 Audit & Compliance

4.6.1 Kubernetes Audit Logging

Audit Policy Configuration:

```
# Create audit policy file
cat > /etc/kubernetes/audit-policy.yaml <<EOF
apiVersion: audit.k8s.io/v1
kind: Policy
rules:
# Log all requests at RequestResponse level
- level: RequestResponse
  verbs: ["create", "update", "patch", "delete"]
  resources:
  - group: ""
    resources: ["secrets", "configmaps"]
  - group: "rbac.authorization.k8s.io"
    resources: ["roles", "rolebindings", "clusterroles", "
       clusterrolebindings"]

# Log pod exec/attach at Metadata level
- level: Metadata
  verbs: ["create"]
```

```
  resources:
  - group: ""
    resources: ["pods/exec", "pods/attach", "pods/portforward"]

# Log authentication decisions
- level: Metadata
  omitStages:
  - RequestReceived
  resources:
  - group: "authentication.k8s.io"
    resources: ["tokenreviews"]
  - group: "authorization.k8s.io"
    resources: ["subjectaccessreviews"]

# Log all other requests at Metadata level
- level: Metadata
  omitStages:
  - RequestReceived
EOF

# Add to kube-apiserver manifest
--audit-policy-file=/etc/kubernetes/audit-policy.yaml
--audit-log-path=/var/log/kubernetes/audit.log
--audit-log-maxage=30
--audit-log-maxbackup=10
--audit-log-maxsize=100
```

Forward Audit Logs to Centralized System:

```
# Use audit webhook for real-time forwarding
--audit-webhook-config-file=/etc/kubernetes/audit-webhook.yaml
--audit-webhook-batch-max-wait=5s
```

Reference:

- Audit Logging: https://kubernetes.io/docs/tasks/debug/debug-cluster/audit/
- Audit Policy: https://kubernetes.io/docs/reference/config-api/apiserver-audit.v1/

4.6.2 Runtime Security Monitoring (Falco)

Falco Installation:

```
# Install Falco using Helm
helm repo add falcosecurity https://falcosecurity.github.io/charts
helm install falco falcosecurity/falco \
  --namespace falco \
  --create-namespace \
  --set falco.grpc.enabled=true \
  --set falco.grpcOutput.enabled=true

# View Falco alerts
kubectl logs -n falco -l app.kubernetes.io/name=falco -f
```

Custom Falco Rules for Internal Apps:

```
kubectl apply -f - <<EOF
apiVersion: v1
kind: ConfigMap
metadata:
  name: falco-custom-rules
  namespace: falco
data:
  custom-rules.yaml: |
    - rule: Unexpected Network Connection
      desc: Detect unexpected outbound connections
      condition: >
        outbound and not fd.sip in (allowed_ips)
        and container.ns = "app-production"
      output: >
        Unexpected network connection
        (connection=%fd.name user=%user.name container=%container.name)
      priority: WARNING

    - rule: Terminal Shell in Container
      desc: Detect shell spawned in container
      condition: >
        spawned_process and container
        and shell_procs and proc.tty != 0
        and container.ns = "app-production"
      output: >
        Shell spawned in container
        (user=%user.name container=%container.name shell=%proc.name)
      priority: WARNING
EOF
```

Reference:

- Falco Documentation: https://falco.org/docs/
- Falco Rules: https://github.com/falcosecurity/rules

4.7 Backup & Disaster Recovery

4.7.1 Velero Installation and Configuration

Install Velero:

```
# Download Velero CLI
wget https://github.com/vmware-tanzu/velero/releases/download/v1.12.0/
   velero-v1.12.0-linux-amd64.tar.gz
tar -xvf velero-v1.12.0-linux-amd64.tar.gz
sudo mv velero-v1.12.0-linux-amd64/velero /usr/local/bin/

# Install Velero server (on-premises with MinIO)
velero install \
  --provider aws \
  --plugins velero/velero-plugin-for-aws:v1.8.0 \
  --bucket velero-backups \
  --secret-file ./credentials-velero \
```

```
  --use-volume-snapshots=false \
  --backup-location-config region=minio,s3ForcePathStyle="true",s3Url=http
     ://minio.minio:9000
```

Create Backup Schedule:

```
# Daily backup of app-production namespace
velero schedule create daily-app-backup \
  --schedule="0 2 * * *" \
  --include-namespaces app-production \
  --ttl 720h0m0s  # 30 days retention

# Manual backup
velero backup create app-production-manual \
  --include-namespaces app-production \
  --wait
```

Test Disaster Recovery:

```
# List backups
velero backup get

# Restore from backup
velero restore create --from-backup daily-app-backup-20240119020000

# Monitor restore
velero restore describe <restore-name>
```

Reference:

- Velero Documentation: https://velero.io/docs/
- Velero on-premises: https://velero.io/docs/main/on-premises/

4.7.2 Database Backup Strategy

PostgreSQL Backup with CronJob:

```
kubectl apply -f - <<EOF
apiVersion: v1
kind: ConfigMap
metadata:
  name: postgres-backup-script
  namespace: app-production
data:
  backup.sh: |
    #!/bin/bash
    set -e
    TIMESTAMP=$(date +%Y%m%d_%H%M%S)
    BACKUP_FILE="/backups/db_backup_\${TIMESTAMP}.sql.gz"

    pg_dump -h database-service -U \$POSTGRES_USER \$POSTGRES_DB | gzip >
       \$BACKUP_FILE

    # Keep only last 7 days
    find /backups -name "db_backup_*.sql.gz" -mtime +7 -delete
```

```
---
apiVersion: batch/v1
kind: CronJob
metadata:
  name: postgres-backup
  namespace: app-production
spec:
  schedule: "0 1 * * *"  # Daily at 1 AM
  jobTemplate:
    spec:
      template:
        spec:
          containers:
          - name: backup
            image: postgres:15
            command: ["/bin/bash", "/scripts/backup.sh"]
            env:
            - name: POSTGRES_USER
              valueFrom:
                secretKeyRef:
                  name: app-database-secret
                  key: username
            - name: POSTGRES_DB
              value: appdb
            - name: PGPASSWORD
              valueFrom:
                secretKeyRef:
                  name: app-database-secret
                  key: password
            volumeMounts:
            - name: backup-script
              mountPath: /scripts
            - name: backup-storage
              mountPath: /backups
          restartPolicy: OnFailure
          volumes:
          - name: backup-script
            configMap:
              name: postgres-backup-script
              defaultMode: 0755
          - name: backup-storage
            persistentVolumeClaim:
              claimName: database-backup-pvc
EOF
```

Reference:

- CronJob: https://kubernetes.io/docs/concepts/workloads/controllers/cron-jobs/
- pg_dump: https://www.postgresql.org/docs/current/app-pgdump.html

5 Implementation Roadmap

This section provides a phased implementation timeline aligned with the main cloud-native architecture guide.

5.1 Phase 2: Internal Pilot Readiness (Weeks 5-8)

5.1.1 Week 5: Namespace and RBAC Setup

Deliverables:

- Create namespaces with ResourceQuotas and LimitRanges
- Configure RBAC roles and bindings for human users
- Create ServiceAccounts with least-privilege roles
- Document RBAC design decisions

Validation:

- Verify users can only access permitted namespaces
- Test ServiceAccount permissions are minimal
- Audit RBAC configuration for overly broad permissions

5.1.2 Week 6: Network Policies and Ingress

Deliverables:

- Deploy CNI with NetworkPolicy support (Calico or Cilium)
- Create application-specific NetworkPolicies
- Deploy NGINX Ingress Controller
- Configure TLS certificates for internal domains

Validation:

- Verify NetworkPolicies block unauthorized traffic
- Test TLS termination at ingress
- Validate internal DNS resolution

5.1.3 Week 7: Workload Security and Secrets

Deliverables:

- Apply Pod Security Standards (`baseline`) to namespaces
- Configure security contexts for all deployments
- Create Kubernetes Secrets for applications

- Enable etcd encryption at rest

Validation:

- Test pod deployment fails without compliant security context
- Verify secrets are encrypted in etcd
- Audit container images for root user usage

5.1.4 Week 8: Observability and Backup

Deliverables:

- Enable Kubernetes audit logging
- Deploy Prometheus and Grafana
- Configure centralized logging (Loki)
- Install Velero for cluster backup
- Create database backup CronJob
- Document backup and restore procedures

Validation:

- Review audit logs for create/update/delete events
- Test backup creation and restoration
- Verify metrics collection from all pods

Go/No-Go Checkpoint

Phase 2 Completion Criteria:
Before proceeding to production deployment:

- ☐ All namespaces have ResourceQuotas and LimitRanges
- ☐ RBAC configured with least privilege
- ☐ NetworkPolicies limit traffic to required flows
- ☐ TLS enabled for all ingress traffic
- ☐ Pod Security Standards enforced at `baseline`
- ☐ All secrets stored in Kubernetes Secrets (not Git)
- ☐ etcd encryption at rest enabled
- ☐ Audit logging enabled and tested
- ☐ Backup tested and documented
- ☐ Incident response contacts defined

5.2 Phase 3: Secure Production Readiness (Weeks 9-12)

5.2.1 Week 9: Service Mesh Deployment

Deliverables:

- Deploy Istio or Linkerd service mesh
- Enable sidecar injection for application namespaces
- Configure strict mTLS for all service-to-service traffic
- Verify mTLS with traffic analysis

Validation:

- Verify all pods have sidecar proxies injected
- Capture network traffic showing TLS encryption
- Test service-to-service communication requires valid mTLS certificates

5.2.2 Week 10: Zero-Trust Network Policies

Deliverables:

- Document all required network flows
- Implement default-deny NetworkPolicies
- Create explicit allow rules for required traffic
- Deploy Layer 7 authorization policies (service mesh)

Validation:

- Test unauthorized traffic is blocked
- Verify application functionality with restrictive policies
- Audit for NetworkPolicy gaps

5.2.3 Week 11: Workload Hardening and Policy Enforcement

Deliverables:

- Upgrade Pod Security Standards to `restricted`
- Deploy admission controller (OPA Gatekeeper or Kyverno)
- Create and enforce security policies
- Configure read-only root filesystems where possible

Validation:

- Test policy violations are rejected
- Verify all containers run as non-root
- Audit for containers with unnecessary privileges

5.2.4 Week 12: External Secrets and Runtime Security

Deliverables:

- Deploy HashiCorp Vault
- Install External Secrets Operator
- Migrate secrets from Kubernetes to Vault
- Deploy Falco for runtime security monitoring
- Configure automated backup with Velero
- Test disaster recovery procedures

Validation:

- Verify secrets are dynamically fetched from Vault
- Test secret rotation
- Review Falco alerts for suspicious activity
- Perform full disaster recovery test

Go/No-Go Checkpoint

Phase 3 Production Readiness Criteria:
Before declaring production-ready:

- ☐ Service mesh deployed with strict mTLS
- ☐ Default-deny NetworkPolicies enforced
- ☐ Layer 7 authorization policies configured
- ☐ Pod Security Standards enforced at `restricted`
- ☐ Admission controller blocking policy violations
- ☐ All containers run as non-root with minimal capabilities
- ☐ Read-only root filesystems where applicable
- ☐ External Secrets Operator integrated with Vault
- ☐ Secret rotation automated and tested
- ☐ Falco monitoring active with alert routing
- ☐ Image scanning integrated in CI/CD
- ☐ Automated backup tested and verified
- ☐ Disaster recovery runbook documented and tested
- ☐ Security incident response plan documented
- ☐ Compliance audit evidence collected

6 Validation and Testing

6.1 Security Control Validation

6.1.1 RBAC Validation

Test Least Privilege:

```
# Test ServiceAccount cannot list secrets in other namespaces
kubectl auth can-i list secrets \
  --as=system:serviceaccount:app-production:app-service-account \
  --namespace=default
# Expected: no

# Test user can only access permitted namespace
kubectl auth can-i get pods \
  --as=developer@example.com \
  --namespace=app-production
# Expected: yes
```

```
kubectl auth can-i get pods \
  --as=developer@example.com \
  --namespace=kube-system
# Expected: no
```

6.1.2 NetworkPolicy Validation

Test Default-Deny:

```
# Deploy test pod
kubectl run -n app-production test-pod \
  --image=nicolaka/netshoot \
  --rm -it -- /bin/bash

# Inside test pod, try unauthorized connection
curl http://database-service.other-namespace:5432
# Expected: timeout or connection refused

# Try authorized connection
curl http://database-service.app-production:5432
# Expected: connection successful (based on NetworkPolicy)
```

6.1.3 Pod Security Standards Validation

Test Restricted Policy:

```
# Try to deploy privileged pod (should fail)
kubectl apply -f - <<EOF
apiVersion: v1
kind: Pod
metadata:
  name: privileged-test
  namespace: app-production
spec:
  containers:
  - name: test
    image: nginx
    securityContext:
      privileged: true
EOF
# Expected: Error - violates Pod Security Standards
```

6.1.4 mTLS Validation

Verify Mutual TLS:

```
# Istio: Check mTLS status
istioctl x describe pod <pod-name> -n app-production

# Capture traffic to verify encryption
kubectl exec -n app-production <pod-name> -c istio-proxy -- \
  tcpdump -i eth0 -w /tmp/capture.pcap
```

```
# Linkerd: Check mTLS status
linkerd viz -n app-production stat deploy --from deploy/<source-deploy>
# Look for "secured" connections
```

6.2 Disaster Recovery Testing

6.2.1 Namespace Restore Test

```
# 1. Create backup
velero backup create test-backup \
  --include-namespaces app-production \
  --wait

# 2. Delete namespace
kubectl delete namespace app-production

# 3. Restore from backup
velero restore create --from-backup test-backup

# 4. Verify restoration
kubectl get all -n app-production
kubectl get pvc -n app-production
```

6.2.2 Database Restore Test

```
# 1. List available backups
kubectl exec -n app-production postgres-backup-<pod> -- ls /backups

# 2. Restore database
kubectl exec -n app-production database-0 -- \
  bash -c "gunzip < /backups/db_backup_20240119_010000.sql.gz | psql -U 
     appuser appdb"

# 3. Verify data integrity
kubectl exec -n app-production database-0 -- \
  psql -U appuser -d appdb -c "SELECT COUNT(*) FROM <table>;"
```

6.3 Compliance Audit Validation

6.3.1 Generate Compliance Report

Audit Checklist:

```
# Check Pod Security Standards enforcement
kubectl get ns -L pod-security.kubernetes.io/enforce

# List RBAC bindings
kubectl get rolebindings,clusterrolebindings --all-namespaces

# Verify NetworkPolicies exist
kubectl get networkpolicies --all-namespaces
```

```
# Check audit logging is enabled
kubectl get pods -n kube-system | grep kube-apiserver
kubectl exec -n kube-system kube-apiserver-<node> -- \
  cat /etc/kubernetes/manifests/kube-apiserver.yaml | grep audit

# List all secrets (verify none are in Git)
kubectl get secrets --all-namespaces

# Verify backup schedule
velero schedule get
```

7 Common Pitfalls and Solutions

7.1 Identity & Access Pitfalls

Pitfall	**Problem**	**Solution**
Overly Broad RBAC	Users or ServiceAccounts with cluster-admin or excessive permissions	Apply least privilege. Create specific Roles per application. Audit bindings regularly. Reference: https://kubernetes.io/docs/reference/access-authn-authz/rbac/
Default ServiceAccount Usage	Using default ServiceAccount grants unnecessary permissions	Create dedicated ServiceAccounts per application. Set `automountServiceAccountToken: false` unless required.
No User Authentication	Relying on certificate-based auth without identity provider	Integrate OIDC or LDAP for user authentication. Map OIDC groups to Kubernetes RBAC. Reference: https://kubernetes.io/docs/reference/access-authn-authz/authentication/
Shared ServiceAccounts	Multiple applications sharing ServiceAccount	Create separate ServiceAccount per application. Bind specific permissions to each.

7.2 Network Isolation Pitfalls

Pitfall	Problem	Solution
No NetworkPolicies	All pods can communicate freely	Implement default-deny NetworkPolicies. Explicitly allow required traffic. Reference: https://kubernetes.io/docs/concepts/services-networking/network-policies/
CNI Without Policy Support	NetworkPolicies defined but not enforced	Use Calico, Cilium, or other CNI with NetworkPolicy support. Verify enforcement.
Allowing All Egress	Pods can reach external networks unrestricted	Define egress rules limiting external access. Block internet access for internal-only apps.
Ingress Without TLS	Unencrypted traffic to applications	Require TLS at ingress. Use cert-manager for automated certificate management. Reference: https://cert-manager.io/docs/

7.3 Workload Security Pitfalls

Pitfall	Problem	Solution
Running as Root	Containers run as root user	Configure `runAsNonRoot: true` and `runAsUser: ¡uid¿`. Enforce with Pod Security Standards. Reference: https://kubernetes.io/docs/concepts/security/pod-security-standards/
Privileged Containers	Containers with `privileged: true`	Remove privileged flag. Drop all capabilities with `capabilities.drop: [ALL]`.
Writable Root Filesystem	Containers can modify filesystem	Set `readOnlyRootFilesystem: true`. Mount `emptyDir` for temp storage.
No Resource Limits	Containers can consume unlimited resources	Set `requests` and `limits` for CPU and memory. Enforce with LimitRanges. Reference: https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/
Missing Probes	No health checks for containers	Configure liveness and readiness probes. Reference: https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-startup-probes/

7.4 Secrets Management Pitfalls

Pitfall	Problem	Solution
Secrets in Git	Hardcoded credentials in repositories	Use Sealed Secrets or External Secrets Operator. Never commit secrets. Reference: https://external-secrets.io/
Unencrypted etcd	Secrets stored as base64 in etcd	Enable encryption at rest for etcd. Reference: https://kubernetes.io/docs/tasks/administer-cluster/encrypt-data/
No Secret Rotation	Static secrets never rotated	Implement automated rotation with Vault. Define rotation schedule. Reference: https://www.vaultproject.io/docs
Secrets in Environment Variables	Secrets visible in pod spec and logs	Use volume mounts for secrets. Avoid environment variables for sensitive data.

7.5 Observability Pitfalls

Pitfall	Problem	Solution
No Audit Logging	No forensic trail after incident	Enable Kubernetes audit logging. Retain logs for 90+ days. Reference: https://kubernetes.io/docs/tasks/debug/debug-cluster/audit/
Insufficient Log Retention	Logs deleted before investigation	Centralize logs with retention policy. Use Loki or ELK stack. Reference: https://grafana.com/docs/loki/
No Runtime Monitoring	Malicious activity undetected	Deploy Falco for runtime security monitoring. Alert on suspicious behavior. Reference: https://falco.org/docs/
Missing SLOs	No definition of service health	Define SLIs and SLOs for critical services. Track error budgets. Reference: https://sre.google/sre-book/service-level-objectives/

8 Production Readiness Checklist

This comprehensive checklist validates all security controls before production deployment.

8.1 Identity & Access Control

- ☐ Namespaces created with ResourceQuotas and LimitRanges
- ☐ Dedicated ServiceAccount per application with least-privilege RBAC
- ☐ `automountServiceAccountToken: false` where not required
- ☐ OIDC or LDAP integration for user authentication
- ☐ RBAC roles follow least-privilege principle
- ☐ No cluster-admin bindings except for administrators
- ☐ Regular RBAC audit conducted

8.2 Network Isolation

- ☐ CNI with NetworkPolicy support deployed (Calico/Cilium)
- ☐ Default-deny NetworkPolicies in place
- ☐ Explicit allow rules for all required traffic
- ☐ NetworkPolicies tested and verified
- ☐ Ingress controller with TLS termination
- ☐ Internal DNS resolution tested
- ☐ No unnecessary external network access

8.3 Workload Security

- ☐ Pod Security Standards enforced at `restricted` level
- ☐ All containers run as non-root (`runAsNonRoot: true`)
- ☐ Read-only root filesystem where applicable
- ☐ All capabilities dropped (`capabilities.drop: [ALL]`)
- ☐ Resource requests and limits defined
- ☐ Liveness and readiness probes configured
- ☐ Admission controller enforcing policies (OPA/Kyverno)
- ☐ No privileged containers

8.4 Secrets Management

- ☐ No secrets in Git repositories
- ☐ etcd encryption at rest enabled
- ☐ External Secrets Operator deployed (Phase 3)
- ☐ HashiCorp Vault integrated (Phase 3)
- ☐ Secret rotation automated and tested (Phase 3)
- ☐ Secrets mounted as volumes (not environment variables)
- ☐ Access to Vault restricted via RBAC

8.5 Transport Security

- ☐ Service mesh deployed (Istio/Linkerd) (Phase 3)
- ☐ Strict mTLS enabled for all service-to-service traffic (Phase 3)
- ☐ Layer 7 authorization policies configured (Phase 3)
- ☐ TLS certificates valid and trusted
- ☐ Certificate rotation automated (cert-manager)
- ☐ mTLS verified with traffic analysis (Phase 3)

8.6 Audit & Compliance

- ☐ Kubernetes audit logging enabled
- ☐ Audit logs forwarded to centralized system
- ☐ Log retention policy (90+ days)
- ☐ Falco runtime security monitoring active (Phase 3)
- ☐ Prometheus metrics collection configured
- ☐ Grafana dashboards for security metrics
- ☐ SLOs defined and monitored
- ☐ Compliance evidence documented

8.7 Container Supply Chain

☐ Image scanning in CI/CD pipeline (Phase 3)

☐ Vulnerability blocking policy (Critical/High) (Phase 3)

☐ Image signing implemented (Cosign) (Phase 3)

☐ Image verification in admission controller (Phase 3)

☐ SBOM generation automated (Phase 3)

☐ Base images updated regularly

☐ Private registry with access controls

8.8 Backup & Disaster Recovery

☐ Velero installed and configured

☐ Automated backup schedule created

☐ Database backup CronJob configured

☐ Backup restoration tested successfully

☐ RTO/RPO objectives defined

☐ Disaster recovery runbook documented

☐ Multi-zone deployment for availability

☐ PodDisruptionBudgets configured

8.9 Operational Readiness

☐ Security incident response plan documented

☐ On-call rotation defined

☐ Escalation procedures documented

☐ Runbooks for common incidents

☐ Regular security reviews scheduled

☐ Chaos engineering tests performed (optional)

☐ User training completed

☐ Change management process established

9 Continuous Security Improvement

9.1 Regular Security Reviews

Monthly Reviews:

- Audit RBAC bindings for excessive permissions
- Review NetworkPolicies for coverage gaps
- Check for unencrypted secrets
- Analyze Falco alerts for trends
- Review container images for vulnerabilities

Quarterly Reviews:

- Update Pod Security Standards to latest version
- Review and update admission controller policies
- Test disaster recovery procedures
- Conduct security training for teams
- External security assessment or penetration test

9.2 Metrics and KPIs

Security Metrics to Track:

- Time to patch critical vulnerabilities
- Number of policy violations per month
- Mean time to detect (MTTD) security incidents
- Mean time to respond (MTTR) security incidents
- Percentage of workloads with restricted security context
- Secret rotation frequency
- Backup success rate
- Disaster recovery test success rate

9.3 Technology Evolution

Stay current with security best practices:

- Monitor Kubernetes security advisories: https://kubernetes.io/docs/reference/issues-security/security/
- Follow CNCF security projects: https://www.cncf.io/projects/
- Track CVEs for container images
- Participate in Kubernetes security SIG: https://github.com/kubernetes/community/tree/master/sig-security
- Review CIS Kubernetes Benchmark updates: https://www.cisecurity.org/benchmark/kubernetes

10 Conclusion

10.1 Key Takeaways

1. **Internal $\neq$ Trusted**: Internal networks require the same zero-trust controls as external-facing systems. Lateral movement is a primary attack vector.
2. **Phased Approach**: Start with Phase 2 (pilot readiness) for limited deployments, then harden to Phase 3 (production readiness) with zero-trust controls before full production rollout.
3. **Defense in Depth**: Security is achieved through layered controls: identity, network, workload, secrets, transport, and audit.
4. **Automation is Critical**: Manual security processes don't scale. Automate secrets rotation, backup, vulnerability scanning, and policy enforcement.
5. **Continuous Validation**: Security is not a one-time implementation. Regular testing, auditing, and improvement are essential.

10.2 Next Steps

After completing this implementation guide:

1. Deploy pilot applications in Phase 2 environment
2. Collect feedback and refine security controls
3. Harden to Phase 3 before production rollout
4. Establish operational processes (incident response, backup testing)
5. Plan for continuous security improvement

10.3 Additional Resources

Security Standards:

- CIS Kubernetes Benchmark: https://www.cisecurity.org/benchmark/kubernetes
- NSA/CISA Kubernetes Hardening Guide: https://www.nsa.gov/Press-Room/News-Highlights/Article/Article/2716980/
- OWASP Kubernetes Security Cheat Sheet: https://cheatsheetseries.owasp.org/cheatsheets/Kubernetes_Security_Cheat_Sheet.html

Community Resources:

- Kubernetes Security SIG: https://github.com/kubernetes/community/tree/master/sig-security
- Cloud Native Security Whitepaper: https://www.cncf.io/blog/2022/06/07/introduction-to-the-cloud
- CNCF Security TAG: https://tag-security.cncf.io/

Tools and Utilities:

- kube-bench (CIS compliance): https://github.com/aquasecurity/kube-bench
- kube-hunter (penetration testing): https://github.com/aquasecurity/kube-hunter
- kubescape (security posture): https://github.com/kubescape/kubescape
- kubectl-who-can (RBAC analysis): https://github.com/aquasecurity/kubectl-who-can

---

End of Secure Internal Application Hosting Guide

This guide is a companion to the *Comprehensive Cloud-Native Architecture Implementation Guide*. Together, they provide a complete framework for building and securing internal applications on Kubernetes.
