RH OVE Use Cases Documentation

Red Hat OpenShift Virtualization Ecosystem Team

2025-08-04

Table of Contents

# Use Cases

## Overview

### RH OVE Use Cases Overview

#### Introduction

This section provides comprehensive use cases for the Red Hat OpenShift Virtualization Engine (RH OVE) multi-cluster ecosystem, demonstrating real-world scenarios and implementation patterns.

#### 📊 [Complete Use Cases Summary Table](use-cases-table.md)

For a comprehensive overview of all use cases with complexity ratings, implementation timelines, and prerequisites, see our [Use Cases Summary Table](use-cases-table.md).

#### Use Case Categories

##### 1. VM Lifecycle Management

* **VM Import and Migration**: Importing existing VMs from various sources
* **VM Template Management**: Creating and managing VM templates
* **VM Scaling and Performance**: Dynamic resource allocation and optimization
* **VM Backup and Recovery**: Data protection strategies for virtual machines

##### 2. Hybrid Application Deployment

* **VM + Container Integration**: Running legacy VMs alongside containerized services
* **Multi-Tier Applications**: Web, application, and database tiers across different deployment models
* **Service Mesh Integration**: Connecting VMs and containers through service mesh
* **Data Sharing**: Persistent storage sharing between VMs and containers

##### 3. Platform-as-a-Service (PaaS) Integration

* **Database Services**: Running databases as VMs with containerized management
* **Middleware Platforms**: Message queues, application servers, and integration platforms
* **Development Environments**: Self-service development platforms
* **CI/CD Pipeline Integration**: Automated deployment across VM and container workloads

##### 4. Enterprise Integration Scenarios

* **Legacy Application Modernization**: Gradual migration strategies
* **Disaster Recovery**: Cross-cluster failover and recovery procedures
* **Multi-Cloud Deployment**: Hybrid cloud scenarios with on-premises integration
* **Compliance and Security**: Meeting enterprise security and regulatory requirements

#### Architecture Patterns

##### Hub and Spoke Pattern

* Management cluster orchestrates multiple application clusters
* Centralized policy and governance with distributed execution
* GitOps-driven configuration management

##### Network Integration

* Cilium CNI with Multus multi-network support
* VLAN integration for legacy systems
* Service mesh connectivity between VMs and containers

##### Identity and Access Management

* OIDC-based authentication with Keycloak
* RBAC integration across VM and container environments
* Multi-factor authentication for administrative access

#### Use Case Implementation Guide

##### Prerequisites

* OpenShift 4.12+ clusters with KubeVirt enabled
* Sufficient compute, memory, and storage resources
* Network connectivity between clusters
* Identity provider integration (Keycloak/LDAP/AD)

##### Common Patterns

1. **Resource Provisioning**: Using CDI for VM disk management
2. **Network Configuration**: Multi-network setup with Multus
3. **Storage Management**: Persistent volume strategies for VMs
4. **Monitoring Integration**: Unified monitoring for VMs and containers
5. **Backup Strategies**: Integrated backup solutions for hybrid workloads

#### Getting Started

1. Review the specific use case documentation
2. Understand the architectural requirements
3. Prepare the environment according to prerequisites
4. Follow the step-by-step implementation guide
5. Validate the deployment and test functionality
6. Monitor and maintain the solution

Each use case provides: - **Business Context**: Why this pattern is needed - **Technical Requirements**: Infrastructure and software requirements - **Implementation Steps**: Detailed configuration procedures - **Validation Procedures**: Testing and verification steps - **Troubleshooting Guide**: Common issues and solutions - **Best Practices**: Recommendations for production deployment

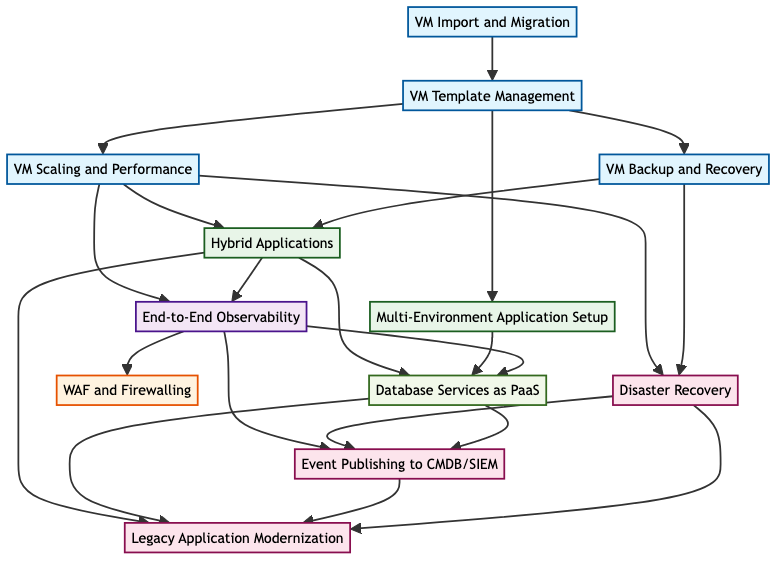
## Use Cases Table

#### Use Cases Dependencies Graph

This diagram shows the logical dependencies and recommended implementation flow between various use cases within the RH OVE ecosystem. The graph illustrates prerequisite relationships - arrows point from foundational capabilities to dependent advanced features.

##### Color Legend

* 🔵 **VM Lifecycle** (Light Blue): Core virtualization capabilities
* 🟣 **Observability** (Purple): Monitoring and insights
* 🟢 **Application Deployment** (Green): Application management and deployment
* 🟠 **Security** (Orange): Security and protection services
* 🔴 **Enterprise Integration** (Pink): Enterprise systems integration
* 🟡 **PaaS Services** (Light Green): Platform-as-a-Service offerings



### Use Cases Summary Table

#### Overview

This table provides a comprehensive overview of all use cases documented for the Red Hat OpenShift Virtualization Engine (RH OVE) ecosystem.

#### Use Cases Matrix

| Use Case | Category | Complexity | Key Technologies | Business Value | Implementation Time | Prerequisites |
| --- | --- | --- | --- | --- | --- | --- |
| [VM Import and Migration](vm-importation.md) | VM Lifecycle | Medium | KubeVirt, CDI, MTV | Legacy system modernization | 2-4 weeks | OpenShift 4.12+, Source VM access |
| [VM Template Management](vm-template-management.md) | VM Lifecycle | Low | KubeVirt, CDI | Standardized deployments | 1-2 weeks | OpenShift 4.12+, Template storage |
| [VM Scaling and Performance](vm-scaling-performance.md) | VM Lifecycle | High | HPA, VPA, KubeVirt | Resource optimization | 3-6 weeks | Metrics server, Monitoring stack |
| [VM Backup and Recovery](vm-backup-recovery.md) | VM Lifecycle | Medium | Rubrik CDM, Polaris | Data protection | 2-4 weeks | Rubrik infrastructure |
| [Hybrid Applications](hybrid-applications.md) | Application Deployment | High | Cilium, Service Mesh | Modernization flexibility | 4-8 weeks | Multi-network setup |
| [Database Services as PaaS](database-services-paas.md) | PaaS Integration | High | DB Operators, Helm | Self-service databases | 6-10 weeks | Persistent storage, Operators |
| [Legacy Application Modernization](legacy-modernization.md) | Enterprise Integration | Very High | MTA, Service Mesh, Tekton | Digital transformation | 12-24 weeks | Application analysis |
| [Disaster Recovery](disaster-recovery.md) | Enterprise Integration | High | RHACM, Storage replication | Business continuity | 8-12 weeks | Multi-site infrastructure |
| [End-to-End Observability](end-to-end-observability.md) | Observability | Medium | Prometheus, Jaeger, Dynatrace | Operational insights | 3-6 weeks | Monitoring infrastructure |
| [WAF and Firewalling](waf-firewalling.md) | Security | Medium | Cilium L4-L7, F5 BigIP | Application security | 2-4 weeks | F5 BigIP appliance |
| [Event Publishing to CMDB/SIEM](publishing-events-to-cmdb-siem.md) | Integration | High | Event Bus, Adapters | Enterprise integration | 4-8 weeks | CMDB/SIEM connectivity |
| [Multi-Environment Application Setup](setup-multi-env-application.md) | Application Deployment | Medium | RHACM, ArgoCD, Namespaces | Development workflow | 2-4 weeks | Multi-cluster setup |

#### Complexity Levels

| Level | Description | Skills Required | Timeline |
| --- | --- | --- | --- |
| **Low** | Basic configuration with standard components | Platform administrator | 1-2 weeks |
| **Medium** | Integration of multiple components with custom configuration | Senior platform engineer | 2-6 weeks |
| **High** | Complex multi-component solutions requiring custom development | Solution architect + team | 4-12 weeks |
| **Very High** | Enterprise-wide transformation requiring extensive planning | Enterprise architect + multiple teams | 12+ weeks |

#### Category Breakdown

##### VM Lifecycle Management

* **Purpose**: Managing virtual machine operations and lifecycle
* **Use Cases**: 4 use cases covering import, templates, scaling, and backup
* **Key Benefits**: Infrastructure consolidation, operational efficiency

##### Application Deployment

* **Purpose**: Deploying and managing hybrid application architectures
* **Use Cases**: 1 comprehensive use case for hybrid applications
* **Key Benefits**: Application modernization, deployment flexibility

##### PaaS Integration

* **Purpose**: Providing platform services for development teams
* **Use Cases**: 1 comprehensive database services platform
* **Key Benefits**: Developer productivity, service standardization

##### Enterprise Integration

* **Purpose**: Integrating with existing enterprise systems and processes
* **Use Cases**: 2 use cases covering modernization and disaster recovery
* **Key Benefits**: Risk mitigation, business continuity

##### Observability

* **Purpose**: Monitoring and understanding system behavior
* **Use Cases**: 1 comprehensive observability solution
* **Key Benefits**: Operational visibility, proactive issue resolution

##### Security

* **Purpose**: Protecting applications and infrastructure
* **Use Cases**: 1 WAF and firewalling solution
* **Key Benefits**: Security compliance, threat protection

##### Integration

* **Purpose**: Connecting with external enterprise systems
* **Use Cases**: 1 event publishing integration
* **Key Benefits**: Enterprise integration, compliance reporting

#### Implementation Priority Matrix

##### Phase 1: Foundation (Weeks 1-8)

1. VM Template Management
2. VM Import and Migration
3. End-to-End Observability

##### Phase 2: Core Services (Weeks 9-20)

1. VM Scaling and Performance
2. VM Backup and Recovery
3. WAF and Firewalling

##### Phase 3: Advanced Integration (Weeks 21-36)

1. Hybrid Applications
2. Database Services as PaaS
3. Event Publishing to CMDB/SIEM

##### Phase 4: Enterprise Transformation (Weeks 37+)

1. Legacy Application Modernization
2. Disaster Recovery

#### Prerequisites Summary

##### Common Prerequisites

* OpenShift 4.12+ with KubeVirt enabled
* Sufficient compute, memory, and storage resources
* Network connectivity between components
* Identity provider integration

##### Specialized Prerequisites

* **Rubrik Infrastructure**: For VM backup and recovery
* **F5 BigIP**: For advanced WAF capabilities
* **External Systems**: CMDB, SIEM, legacy systems for integration use cases
* **Multi-site Setup**: For disaster recovery scenarios

#### Success Metrics

| Use Case Category | Key Performance Indicators |
| --- | --- |
| VM Lifecycle | VM provisioning time, resource utilization, backup success rate |
| Application Deployment | Deployment frequency, rollback rate, application performance |
| PaaS Integration | Service provisioning time, developer satisfaction, service availability |
| Enterprise Integration | Integration success rate, compliance score, incident response time |
| Observability | Mean time to detection (MTTD), alert accuracy, dashboard usage |
| Security | Security incident reduction, compliance pass rate, threat detection rate |

#### Getting Started

1. **Assessment**: Review your current infrastructure and identify priority use cases
2. **Planning**: Create implementation roadmap based on complexity and business value
3. **Prerequisites**: Ensure all required infrastructure and tools are available
4. **Pilot**: Start with low-complexity use cases to build expertise
5. **Scale**: Gradually implement more complex use cases as team capabilities grow

For detailed implementation guidance, refer to the individual use case documentation linked in the table above.

## Vm Importation

### Use Case: VM Importation and Migration

#### Business Context

Organizations often need to migrate existing virtual machines from various virtualization platforms (VMware vSphere, Red Hat Virtualization, Hyper-V, or KVM) to OpenShift Virtualization. This use case demonstrates how to import VMs while maintaining data integrity, minimizing downtime, and ensuring seamless operation in the new environment.

#### Technical Requirements

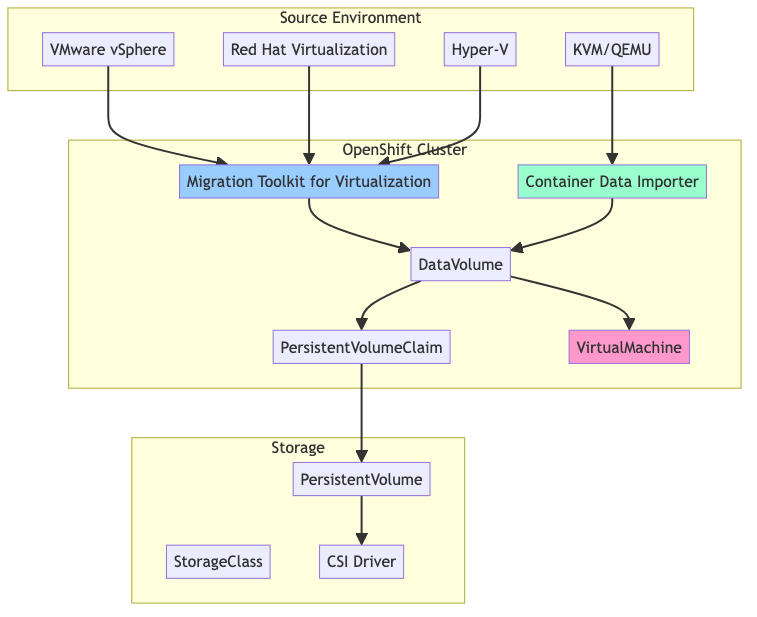
##### Infrastructure Requirements

* OpenShift 4.12+ cluster with KubeVirt enabled
* Container Data Importer (CDI) operator installed
* Sufficient storage capacity for VM disk images
* Network connectivity to source virtualization platform
* Migration Toolkit for Virtualization (MTV) operator (optional)

##### Resource Requirements

* **CPU**: 2+ cores per VM being imported
* **Memory**: 4GB+ RAM per VM being imported
* **Storage**: 2x the source VM disk size (for staging and final storage)
* **Network**: High-bandwidth connection for large VM transfers

#### Implementation Architecture



#### Implementation Steps

##### Step 1: Prepare the Source Environment

###### For VMware vSphere

### Create a service account for MTV  
### In vSphere Client:  
### 1. Create a dedicated user account  
### 2. Assign minimal required permissions:  
### - Datastore > Browse datastore  
### - Virtual machine > Snapshot management  
### - Virtual machine > Provisioning

###### For KVM/QEMU Direct Import

### Prepare VM disk images  
qemu-img convert -f qcow2 -O qcow2 /path/to/source.qcow2 /path/to/optimized.qcow2  
  
### Compress for faster transfer  
gzip /path/to/optimized.qcow2

##### Step 2: Install Migration Tools

###### Install Migration Toolkit for Virtualization

apiVersion: operators.coreos.com/v1alpha1  
kind: Subscription  
metadata:  
 name: mtv-operator  
 namespace: openshift-mtv  
spec:  
 channel: release-v2.4  
 installPlanApproval: Automatic  
 name: mtv-operator  
 source: redhat-operators  
 sourceNamespace: openshift-marketplace  
---  
apiVersion: v1  
kind: Namespace  
metadata:  
 name: openshift-mtv

###### Install Container Data Importer

apiVersion: operators.coreos.com/v1alpha1  
kind: Subscription  
metadata:  
 name: containerized-data-importer  
 namespace: openshift-cnv  
spec:  
 channel: stable  
 installPlanApproval: Automatic  
 name: containerized-data-importer  
 source: redhat-operators  
 sourceNamespace: openshift-marketplace

##### Step 3: Configure Source Provider (MTV)

apiVersion: forklift.konveyor.io/v1beta1  
kind: Provider  
metadata:  
 name: vmware-source  
 namespace: openshift-mtv  
spec:  
 type: vsphere  
 url: https://vcenter.example.com/sdk  
 secret:  
 name: vmware-credentials  
 namespace: openshift-mtv  
---  
apiVersion: v1  
kind: Secret  
metadata:  
 name: vmware-credentials  
 namespace: openshift-mtv  
type: Opaque  
stringData:  
 user: migration-user@vsphere.local  
 password: "your-secure-password"  
 thumbprint: "AA:BB:CC:DD:EE:FF:00:11:22:33:44:55:66:77:88:99:AA:BB:CC:DD"

##### Step 4: Create Migration Plan

apiVersion: forklift.konveyor.io/v1beta1  
kind: Plan  
metadata:  
 name: webapp-migration-plan  
 namespace: openshift-mtv  
spec:  
 provider:  
 source:  
 name: vmware-source  
 namespace: openshift-mtv  
 destination:  
 name: host  
 namespace: openshift-mtv  
 targetNamespace: app-web-prod  
 vms:  
 - id: vm-12345  
 name: web-server-01  
 - id: vm-12346  
 name: app-server-01  
 map:  
 network:  
 - source:  
 name: "VM Network"  
 destination:  
 name: default  
 type: pod  
 - source:  
 name: "Storage Network"  
 destination:  
 name: storage-net  
 type: multus  
 storage:  
 - source:  
 name: "datastore1"  
 destination:  
 storageClass: ocs-storagecluster-ceph-rbd  
 - source:  
 name: "datastore2"  
 destination:  
 storageClass: ocs-storagecluster-cephfs

##### Step 5: Direct Import Using CDI (Alternative Method)

###### Import from HTTP/HTTPS Source

apiVersion: cdi.kubevirt.io/v1beta1  
kind: DataVolume  
metadata:  
 name: imported-vm-disk  
 namespace: app-web-prod  
spec:  
 source:  
 http:  
 url: "https://storage.example.com/vm-images/web-server.qcow2"  
 secretRef: "image-pull-secret"  
 storage:  
 accessModes:  
 - ReadWriteOnce  
 resources:  
 requests:  
 storage: 50Gi  
 storageClassName: ocs-storagecluster-ceph-rbd  
---  
apiVersion: v1  
kind: Secret  
metadata:  
 name: image-pull-secret  
 namespace: app-web-prod  
type: kubernetes.io/basic-auth  
stringData:  
 username: "storage-user"  
 password: "storage-password"

###### Import from Registry

apiVersion: cdi.kubevirt.io/v1beta1  
kind: DataVolume  
metadata:  
 name: registry-imported-vm  
 namespace: app-web-prod  
spec:  
 source:  
 registry:  
 url: "docker://quay.io/example/vm-images:web-server-v1.0"  
 pullMethod: pod  
 secretRef: "registry-credentials"  
 storage:  
 accessModes:  
 - ReadWriteOnce  
 resources:  
 requests:  
 storage: 30Gi  
 storageClassName: ocs-storagecluster-ceph-rbd

##### Step 6: Create Virtual Machine Definition

apiVersion: kubevirt.io/v1  
kind: VirtualMachine  
metadata:  
 name: imported-web-server  
 namespace: app-web-prod  
 labels:  
 app: web-server  
 tier: frontend  
 environment: production  
 annotations:  
 description: "Imported web server from VMware vSphere"  
spec:  
 running: false  
 template:  
 metadata:  
 labels:  
 kubevirt.io/vm: imported-web-server  
 spec:  
 domain:  
 cpu:  
 cores: 4  
 sockets: 1  
 threads: 1  
 memory:  
 guest: 8Gi  
 devices:  
 disks:  
 - name: rootdisk  
 disk:  
 bus: virtio  
 - name: cloudinitdisk  
 disk:  
 bus: virtio  
 interfaces:  
 - name: default  
 masquerade: {}  
 - name: storage-network  
 bridge: {}  
 machine:  
 type: pc-q35-rhel8.6.0  
 resources:  
 requests:  
 memory: 8Gi  
 cpu: 4  
 networks:  
 - name: default  
 pod: {}  
 - name: storage-network  
 multus:  
 networkName: storage-net  
 volumes:  
 - name: rootdisk  
 dataVolume:  
 name: imported-vm-disk  
 - name: cloudinitdisk  
 cloudInitNoCloud:  
 userData: |  
 #cloud-config  
 user: admin  
 password: changeme  
 chpasswd: { expire: False }  
 ssh\_pwauth: True  
 package\_upgrade: true  
 packages:  
 - qemu-guest-agent  
 runcmd:  
 - systemctl enable qemu-guest-agent  
 - systemctl start qemu-guest-agent

##### Step 7: Execute Migration

###### Start Migration Plan

### Apply the migration plan  
oc apply -f migration-plan.yaml  
  
### Monitor migration progress  
oc get migration -n openshift-mtv -w  
  
### Check migration status  
oc describe migration webapp-migration -n openshift-mtv

###### Monitor Import Progress (CDI)

### Check DataVolume status  
oc get dv imported-vm-disk -n app-web-prod  
  
### Monitor import progress  
oc describe dv imported-vm-disk -n app-web-prod  
  
### Check PVC status  
oc get pvc imported-vm-disk -n app-web-prod

##### Step 8: Post-Migration Validation

###### Start and Validate VM

### Start the imported VM  
oc patch vm imported-web-server -n app-web-prod --type merge -p '{"spec":{"running":true}}'  
  
### Check VM status  
oc get vm imported-web-server -n app-web-prod  
  
### Check VMI status  
oc get vmi imported-web-server -n app-web-prod  
  
### Access VM console  
virtctl console imported-web-server -n app-web-prod

###### Network Connectivity Test

### Test pod network connectivity  
oc exec -it deployment/test-client -- ping $(oc get vmi imported-web-server -n app-web-prod -o jsonpath='{.status.interfaces[0].ipAddress}')  
  
### Test multus network connectivity  
oc exec -it deployment/storage-client -- ping $(oc get vmi imported-web-server -n app-web-prod -o jsonpath='{.status.interfaces[1].ipAddress}')

#### Troubleshooting Guide

##### Common Issues and Solutions

###### 1. Import Fails with “Insufficient Storage”

### Check available storage  
oc get pv | grep Available  
  
### Check storage class  
oc describe storageclass ocs-storagecluster-ceph-rbd  
  
### Solution: Request smaller disk or add more storage capacity

###### 2. Network Import Timeout

### Increase timeout in DataVolume  
apiVersion: cdi.kubevirt.io/v1beta1  
kind: DataVolume  
metadata:  
 name: imported-vm-disk  
 annotations:  
 cdi.kubevirt.io/storage.pod.restarts: "10"  
 cdi.kubevirt.io/storage.pod.retries: "4"  
spec:  
 # ... rest of spec

###### 3. VM Boot Issues After Import

### Check VM events  
oc describe vm imported-web-server -n app-web-prod  
  
### Common fixes:  
### 1. Update machine type  
### 2. Install virtio drivers  
### 3. Adjust memory/CPU allocation

###### 4. Performance Issues

### Optimize disk performance  
spec:  
 domain:  
 devices:  
 disks:  
 - name: rootdisk  
 disk:  
 bus: virtio  
 cache: writethrough # Add cache policy  
 io: native # Add I/O policy

#### Best Practices

##### Pre-Migration

1. **Inventory Assessment**: Document all VMs, their dependencies, and resource requirements
2. **Network Planning**: Map source networks to destination networks
3. **Storage Planning**: Choose appropriate storage classes for different workload types
4. **Security Review**: Ensure imported VMs meet security standards

##### During Migration

1. **Batch Processing**: Migrate VMs in small batches to manage resources
2. **Monitoring**: Continuously monitor migration progress and resource utilization
3. **Validation**: Test each migrated VM before proceeding to the next batch
4. **Documentation**: Keep detailed logs of the migration process

##### Post-Migration

1. **Performance Tuning**: Optimize VM configurations for the new environment
2. **Security Hardening**: Apply security policies and configurations
3. **Backup Configuration**: Set up backup strategies for migrated VMs
4. **Monitoring Setup**: Configure monitoring and alerting for new VMs

##### Production Considerations

1. **Resource Quotas**: Implement appropriate resource quotas per namespace
2. **Network Policies**: Apply network segmentation policies
3. **Storage Policies**: Use appropriate storage classes for different tiers
4. **Disaster Recovery**: Plan for cross-cluster failover scenarios

#### Integration with RH OVE Ecosystem

##### GitOps Integration

### ArgoCD Application for VM management  
apiVersion: argoproj.io/v1alpha1  
kind: Application  
metadata:  
 name: imported-vms  
 namespace: argocd  
spec:  
 project: production  
 source:  
 repoURL: https://git.company.com/rh-ove/vm-configs  
 targetRevision: HEAD  
 path: imported-vms  
 destination:  
 server: https://kubernetes.default.svc  
 namespace: app-web-prod  
 syncPolicy:  
 automated:  
 prune: false  
 selfHeal: true

##### Monitoring Integration

### ServiceMonitor for VM metrics  
apiVersion: monitoring.coreos.com/v1  
kind: ServiceMonitor  
metadata:  
 name: vm-metrics  
 namespace: app-web-prod  
spec:  
 selector:  
 matchLabels:  
 app: kubevirt-vm  
 endpoints:  
 - port: metrics  
 interval: 30s  
 path: /metrics

This comprehensive guide provides everything needed to successfully import and migrate VMs into the RH OVE multi-cluster ecosystem while maintaining enterprise-grade security, monitoring, and operational standards.

## Vm Template Management

### Use Case: VM Template Management

#### Business Context

Efficient VM template management is crucial for maintaining consistency, reducing deployment times, and ensuring that VMs adhere to organizational standards. By leveraging templates, organizations can streamline the creation of VMs with predefined configurations that meet specific operational, security, and compliance requirements.

#### Technical Requirements

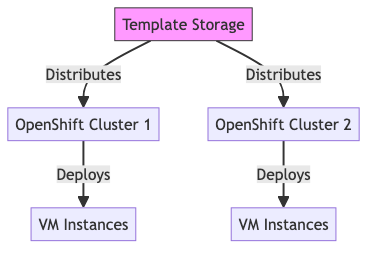
##### Infrastructure Requirements

* OpenShift 4.12+ with KubeVirt enabled
* Storage solutions for template images (e.g., OpenShift Data Foundation)
* Network connectivity between clusters for template distribution

##### Resource Requirements

* **Storage**: Adequate space for storing multiple VM templates
* **Network**: Fast connectivity for template distribution and deployment

#### Architecture Overview



#### Implementation Steps

##### Step 1: Create VM Templates

###### Define Template Specification

apiVersion: kubevirt.io/v1alpha3  
kind: VirtualMachineTemplate  
metadata:  
 name: webserver-template  
spec:  
 domain:  
 cpu:  
 cores: 2  
 devices:  
 disks:  
 - name: rootdisk  
 disk:  
 bus: virtio  
 memory:  
 guest: 4Gi  
 volumes:  
 - name: rootdisk  
 containerDisk:  
 image: kubevirt/cirros-registry-disk-demo

##### Step 2: Store Templates

###### Upload Template to Storage

* Use OpenShift Data Foundation or another persistent storage solution to store VM templates.

##### Step 3: Distribute Templates

###### Sync Templates Across Clusters

* Ensure templates are replicated to all relevant clusters using synchronized storage.

##### Step 4: Deploy VMs Using Templates

###### Create VMs from Templates

apiVersion: kubevirt.io/v1alpha3  
kind: VirtualMachine  
metadata:  
 name: new-webserver  
spec:  
 running: false  
 template:  
 metadata:  
 labels:  
 vm.kubevirt.io/template: webserver-template  
 spec:  
 domain:  
 cpu:  
 cores: 2  
 devices:  
 disks:  
 - disk:  
 bus: virtio  
 name: rootdisk  
 memory:  
 guest: 4Gi

#### Troubleshooting Guide

##### Common Issues and Solutions

###### Template Sync Issues

* Verify storage connectivity and check log files for errors.

###### Deployment Failures

* Ensure templates are properly defined and contain all necessary configurations.

#### Best Practices

* **Template Versioning**: Keep versions of templates to track changes and rollback if necessary.
* **Regular Audits**: Periodically review and update templates to ensure they meet current security and compliance standards.
* **Backup Templates**: Ensure templates are backed up to prevent loss in case of storage failures.

#### Integration with RH OVE Ecosystem

##### Monitoring Integration

* Set up monitoring for template storage systems to detect and resolve issues promptly.

This guide provides the necessary steps and best practices for efficient VM template management within an RH OVE environment, facilitating faster and more consistent VM deployments.

## Vm Scaling Performance

### Use Case: VM Scaling and Performance Optimization

#### Business Context

Dynamic scaling and performance optimization are essential for maintaining optimal VM performance while managing resource costs. This use case demonstrates how to implement auto-scaling, resource optimization, and performance monitoring for virtual machines in the RH OVE ecosystem.

#### Technical Requirements

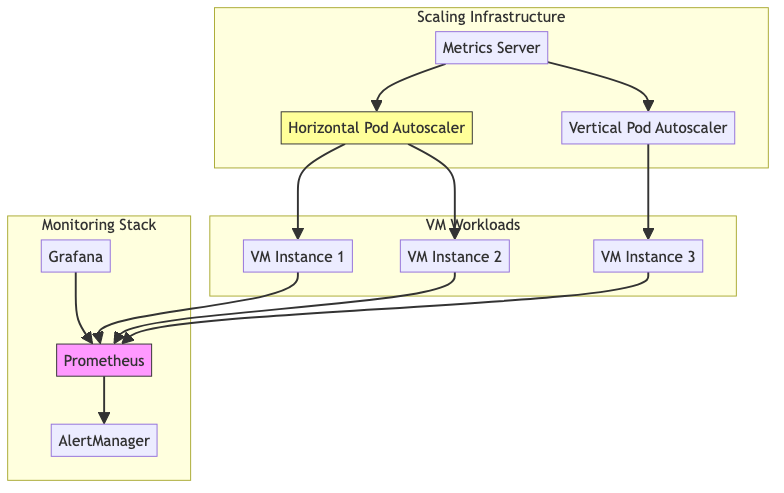
##### Infrastructure Requirements

* OpenShift 4.12+ with KubeVirt enabled
* Horizontal Pod Autoscaler (HPA) support
* Metrics server and custom metrics API
* Persistent storage with performance monitoring capabilities
* CPU and memory monitoring tools (Prometheus/Grafana)

##### Resource Requirements

* **CPU**: Variable based on workload demands
* **Memory**: Dynamic allocation based on usage patterns
* **Storage**: High-performance storage with IOPS monitoring
* **Network**: Low-latency network for performance-sensitive applications

#### Architecture Overview



#### Implementation Steps

##### Step 1: Enable VM Performance Monitoring

###### Deploy VM with Performance Monitoring

apiVersion: kubevirt.io/v1  
kind: VirtualMachine  
metadata:  
 name: performance-vm  
 namespace: vm-workloads  
spec:  
 running: true  
 template:  
 metadata:  
 labels:  
 app: performance-vm  
 monitoring: "enabled"  
 spec:  
 domain:  
 cpu:  
 cores: 2  
 model: host-passthrough  
 devices:  
 disks:  
 - disk:  
 bus: virtio  
 name: rootdisk  
 interfaces:  
 - name: default  
 bridge: {}  
 memory:  
 guest: 4Gi  
 resources:  
 requests:  
 memory: 4Gi  
 cpu: 2  
 limits:  
 memory: 8Gi  
 cpu: 4  
 networks:  
 - name: default  
 pod: {}  
 volumes:  
 - dataVolume:  
 name: performance-vm-dv  
 name: rootdisk  
---  
apiVersion: cdi.kubevirt.io/v1beta1  
kind: DataVolume  
metadata:  
 name: performance-vm-dv  
 namespace: vm-workloads  
spec:  
 source:  
 http:  
 url: "https://vm-images.example.com/performance-vm.img"  
 pvc:  
 accessModes:  
 - ReadWriteOnce  
 resources:  
 requests:  
 storage: 50Gi  
 storageClassName: fast-ssd

##### Step 2: Configure Horizontal Pod Autoscaler for VMs

###### HPA Configuration

apiVersion: autoscaling/v2  
kind: HorizontalPodAutoscaler  
metadata:  
 name: vm-hpa  
 namespace: vm-workloads  
spec:  
 scaleTargetRef:  
 apiVersion: kubevirt.io/v1  
 kind: VirtualMachine  
 name: performance-vm  
 minReplicas: 1  
 maxReplicas: 5  
 metrics:  
 - type: Resource  
 resource:  
 name: cpu  
 target:  
 type: Utilization  
 averageUtilization: 70  
 - type: Resource  
 resource:  
 name: memory  
 target:  
 type: Utilization  
 averageUtilization: 80  
 behavior:  
 scaleUp:  
 stabilizationWindowSeconds: 60  
 policies:  
 - type: Percent  
 value: 50  
 periodSeconds: 60  
 scaleDown:  
 stabilizationWindowSeconds: 300  
 policies:  
 - type: Percent  
 value: 25  
 periodSeconds: 60

##### Step 3: Implement Vertical Pod Autoscaler

###### VPA Configuration

apiVersion: autoscaling.k8s.io/v1  
kind: VerticalPodAutoscaler  
metadata:  
 name: vm-vpa  
 namespace: vm-workloads  
spec:  
 targetRef:  
 apiVersion: kubevirt.io/v1  
 kind: VirtualMachine  
 name: performance-vm  
 updatePolicy:  
 updateMode: "Auto"  
 resourcePolicy:  
 containerPolicies:  
 - containerName: compute  
 minAllowed:  
 cpu: 100m  
 memory: 1Gi  
 maxAllowed:  
 cpu: 8  
 memory: 16Gi  
 controlledResources: ["cpu", "memory"]

##### Step 4: Performance Monitoring and Alerting

###### ServiceMonitor for VM Metrics

apiVersion: monitoring.coreos.com/v1  
kind: ServiceMonitor  
metadata:  
 name: vm-performance-monitor  
 namespace: vm-workloads  
 labels:  
 app: vm-monitor  
spec:  
 selector:  
 matchLabels:  
 monitoring: "enabled"  
 endpoints:  
 - port: metrics  
 interval: 30s  
 path: /metrics  
 honorLabels: true

###### Performance Alerting Rules

apiVersion: monitoring.coreos.com/v1  
kind: PrometheusRule  
metadata:  
 name: vm-performance-alerts  
 namespace: vm-workloads  
spec:  
 groups:  
 - name: vm.performance  
 rules:  
 - alert: VMHighCPUUsage  
 expr: kubevirt\_vm\_cpu\_usage > 0.8  
 for: 5m  
 labels:  
 severity: warning  
 annotations:  
 summary: "VM {{ $labels.name }} has high CPU usage"  
 description: "VM {{ $labels.name }} CPU usage is above 80% for more than 5 minutes."  
   
 - alert: VMHighMemoryUsage  
 expr: kubevirt\_vm\_memory\_usage > 0.9  
 for: 5m  
 labels:  
 severity: critical  
 annotations:  
 summary: "VM {{ $labels.name }} has high memory usage"  
 description: "VM {{ $labels.name }} memory usage is above 90% for more than 5 minutes."  
   
 - alert: VMDiskIOHigh  
 expr: kubevirt\_vm\_disk\_iops > 1000  
 for: 10m  
 labels:  
 severity: warning  
 annotations:  
 summary: "VM {{ $labels.name }} has high disk I/O"  
 description: "VM {{ $labels.name }} disk IOPS is above 1000 for more than 10 minutes."

##### Step 5: Performance Optimization Strategies

###### CPU Pinning for Performance-Critical VMs

apiVersion: kubevirt.io/v1  
kind: VirtualMachine  
metadata:  
 name: high-performance-vm  
 namespace: vm-workloads  
spec:  
 running: true  
 template:  
 metadata:  
 annotations:  
 cpu-load-balancing.crio.io: "disable"  
 cpu-quota.crio.io: "disable"  
 irq-load-balancing.crio.io: "disable"  
 spec:  
 domain:  
 cpu:  
 cores: 4  
 sockets: 1  
 threads: 1  
 dedicatedCpuPlacement: true  
 isolateEmulatorThread: true  
 model: host-passthrough  
 devices:  
 disks:  
 - disk:  
 bus: virtio  
 name: rootdisk  
 memory:  
 guest: 8Gi  
 hugepages:  
 pageSize: 1Gi  
 resources:  
 requests:  
 memory: 8Gi  
 cpu: 4  
 limits:  
 memory: 8Gi  
 cpu: 4  
 nodeSelector:  
 node-role.kubernetes.io/worker: ""  
 performance-node: "true"  
 volumes:  
 - dataVolume:  
 name: high-performance-vm-dv  
 name: rootdisk

##### Step 6: Storage Performance Optimization

###### High-Performance Storage Configuration

apiVersion: storage.k8s.io/v1  
kind: StorageClass  
metadata:  
 name: high-performance-ssd  
provisioner: kubernetes.io/csi-driver  
parameters:  
 type: gp3  
 iops: "3000"  
 throughput: "125"  
 fsType: ext4  
reclaimPolicy: Delete  
allowVolumeExpansion: true  
volumeBindingMode: WaitForFirstConsumer

#### Troubleshooting Guide

##### Common Issues and Solutions

###### Scaling Not Triggering

* **Issue**: HPA/VPA not scaling VMs as expected
* **Solution**:
  + Check metrics server functionality: kubectl top nodes
  + Verify resource requests are set properly
  + Check HPA status: kubectl describe hpa vm-hpa

###### Performance Degradation

* **Issue**: VM performance is not meeting expectations
* **Solution**:
  + Review CPU pinning configuration
  + Check for resource contention on nodes
  + Verify storage performance metrics
  + Analyze network latency and throughput

###### Memory Issues

* **Issue**: Out of memory errors or high memory pressure
* **Solution**:
  + Increase memory limits in VM specification
  + Enable hugepages for better memory performance
  + Check for memory leaks in applications

#### Best Practices

##### Resource Management

* **Right-sizing**: Start with conservative resource allocations and scale based on monitoring data
* **Resource Limits**: Always set both requests and limits to prevent resource starvation
* **Node Selection**: Use node selectors and taints to ensure VMs are scheduled on appropriate nodes

##### Performance Tuning

* **CPU Optimization**: Use CPU pinning for performance-critical workloads
* **Memory Optimization**: Configure hugepages for memory-intensive applications
* **Storage Optimization**: Use high-performance storage classes for I/O intensive workloads
* **Network Optimization**: Configure SR-IOV for network-intensive applications

##### Monitoring and Alerting

* **Proactive Monitoring**: Set up comprehensive monitoring for all performance metrics
* **Alert Thresholds**: Configure appropriate alert thresholds to prevent performance issues
* **Capacity Planning**: Use historical data for capacity planning and resource allocation

#### Integration with RH OVE Ecosystem

##### GitOps Integration

apiVersion: argoproj.io/v1alpha1  
kind: Application  
metadata:  
 name: vm-performance  
 namespace: argocd  
spec:  
 project: default  
 source:  
 repoURL: https://git.example.com/vm-performance-config  
 targetRevision: HEAD  
 path: performance  
 destination:  
 server: https://kubernetes.default.svc  
 namespace: vm-workloads  
 syncPolicy:  
 automated:  
 prune: false  
 selfHeal: true  
 syncOptions:  
 - CreateNamespace=true

##### Multi-Cluster Performance Management

* **Centralized Monitoring**: Aggregate performance metrics across multiple clusters
* **Cross-Cluster Scaling**: Implement scaling policies that consider cluster resource availability
* **Performance Benchmarking**: Establish performance baselines across different cluster configurations

This comprehensive guide provides the tools and strategies needed to implement effective VM scaling and performance optimization within the RH OVE ecosystem, ensuring optimal resource utilization and application performance.

## Vm Backup Recovery

### Use Case: VM Backup and Recovery

#### Business Context

Ensuring data integrity and availability requires effective VM backup and recovery strategies. This use case explores how to implement comprehensive backup solutions and recovery processes to protect critical VM workloads, ensuring minimal downtime and data loss.

#### Technical Requirements

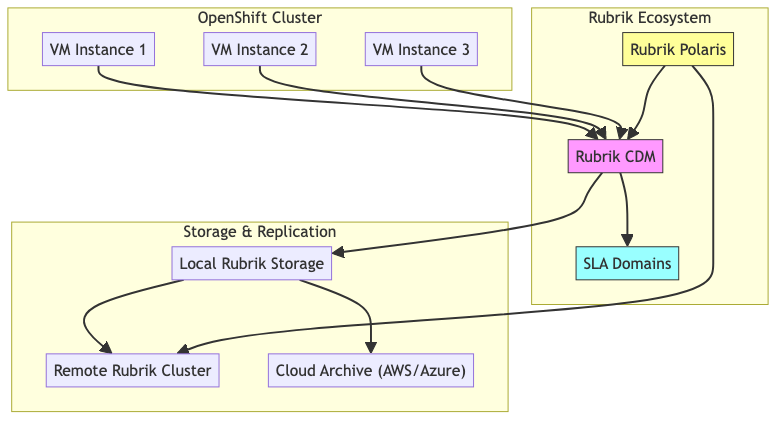
##### Infrastructure Requirements

* OpenShift 4.12+ with KubeVirt enabled
* Rubrik Cloud Data Management for comprehensive backup and recovery
* Network connectivity for Rubrik management and data transfer

##### Resource Requirements

* **Storage**: Adequate space for backup snapshots and archival
* **Network**: Sufficient bandwidth for backup data transfer

#### Architecture Overview



#### Implementation Steps

##### Step 1: Setup Backup Infrastructure

###### Deploy Rubrik Kubernetes Connector

apiVersion: apps/v1  
kind: Deployment  
metadata:  
 name: rubrik-connector  
 namespace: rubrik-system  
spec:  
 replicas: 1  
 selector:  
 matchLabels:  
 app: rubrik-connector  
 template:  
 metadata:  
 labels:  
 app: rubrik-connector  
 spec:  
 containers:  
 - name: connector  
 image: rubrik/kubernetes-connector:latest  
 env:  
 - name: RUBRIK\_CDM\_URL  
 valueFrom:  
 secretKeyRef:  
 name: rubrik-credentials  
 key: cdm-url  
 - name: RUBRIK\_USERNAME  
 valueFrom:  
 secretKeyRef:  
 name: rubrik-credentials  
 key: username  
 - name: RUBRIK\_PASSWORD  
 valueFrom:  
 secretKeyRef:  
 name: rubrik-credentials  
 key: password  
 ports:  
 - containerPort: 8080  
 volumeMounts:  
 - name: kubeconfig  
 mountPath: /etc/kubeconfig  
 readOnly: true  
 volumes:  
 - name: kubeconfig  
 secret:  
 secretName: rubrik-kubeconfig  
---  
apiVersion: v1  
kind: Secret  
metadata:  
 name: rubrik-credentials  
 namespace: rubrik-system  
type: Opaque  
data:  
 cdm-url: <base64-encoded-rubrik-cdm-url>  
 username: <base64-encoded-username>  
 password: <base64-encoded-password>

###### Configure Rubrik SLA Domains

### Rubrik SLA Domain Configuration  
apiVersion: rubrik.com/v1  
kind: SLADomain  
metadata:  
 name: vm-backup-sla  
 namespace: vm-workloads  
spec:  
 name: "VM Production Backup"  
 frequencies:  
 - timeUnit: "Daily"  
 frequency: 1  
 retention: 30 # days  
 - timeUnit: "Weekly"  
 frequency: 1  
 retention: 12 # weeks  
 - timeUnit: "Monthly"  
 frequency: 1  
 retention: 12 # months  
 archivalSpecs:  
 - locationId: "cloud-archive-location"  
 archivalThreshold: 7 # days  
 replicationSpecs:  
 - locationId: "remote-rubrik-cluster"  
 retentionLimit: 90 # days

##### Step 2: Configure VM Backup Policies

###### Assign VMs to SLA Domains

apiVersion: rubrik.com/v1  
kind: VMBackupPolicy  
metadata:  
 name: vm-backup-policy  
 namespace: vm-workloads  
spec:  
 selector:  
 matchLabels:  
 backup: "enabled"  
 slaRef:  
 name: vm-backup-sla  
 backupOptions:  
 consistencyType: "application-consistent"  
 excludedDisks: []  
 includeIndexing: true  
---  
### Apply backup policy to specific VMs  
apiVersion: kubevirt.io/v1  
kind: VirtualMachine  
metadata:  
 name: critical-app-vm  
 namespace: vm-workloads  
 labels:  
 backup: "enabled"  
 tier: "production"  
spec:  
 running: true  
 template:  
 metadata:  
 labels:  
 backup: "enabled"  
 spec:  
 domain:  
 devices:  
 disks:  
 - disk:  
 bus: virtio  
 name: rootdisk  
 memory:  
 guest: 8Gi  
 cpu:  
 cores: 4  
 volumes:  
 - dataVolume:  
 name: critical-app-vm-dv  
 name: rootdisk

###### On-Demand Backup Configuration

### Trigger immediate backup using Rubrik CLI  
rubrik backup vm --vm-name "critical-app-vm" --namespace "vm-workloads" --sla "VM Production Backup"  
  
### Schedule backup using Rubrik API  
curl -X POST "https://rubrik-cdm.example.com/api/internal/vmware/vm/{vm-id}/snapshot" \  
 -H "Authorization: Bearer ${RUBRIK\_TOKEN}" \  
 -H "Content-Type: application/json" \  
 -d '{  
 "slaId": "vm-backup-sla-id",  
 "archivalPolicy": {  
 "locationId": "cloud-archive-location",  
 "retentionDays": 2555  
 }  
 }'

##### Step 3: Configure Backup Storage and Replication

###### Cloud Archive Configuration

apiVersion: rubrik.com/v1  
kind: CloudArchive  
metadata:  
 name: aws-s3-archive  
 namespace: rubrik-system  
spec:  
 provider: "AWS"  
 bucketName: "rubrik-vm-backups"  
 region: "us-west-2"  
 encryptionType: "AWS\_KMS"  
 credentials:  
 secretRef:  
 name: aws-credentials  
 namespace: rubrik-system  
 storageClass: "GLACIER"  
 retentionLock: true  
 retentionLockDurationDays: 2555 # 7 years  
---  
apiVersion: v1  
kind: Secret  
metadata:  
 name: aws-credentials  
 namespace: rubrik-system  
type: Opaque  
data:  
 access-key-id: <base64-encoded-access-key>  
 secret-access-key: <base64-encoded-secret-key>

###### Replication Target Configuration

apiVersion: rubrik.com/v1  
kind: ReplicationTarget  
metadata:  
 name: dr-site-rubrik  
 namespace: rubrik-system  
spec:  
 targetClusterAddress: "rubrik-dr.example.com"  
 credentials:  
 secretRef:  
 name: dr-rubrik-credentials  
 replicationBandwidth: "100 Mbps"  
 encryptionInTransit: true  
 retentionOnTarget: 90 # days

##### Step 4: Implement Recovery Procedures

###### Instant Recovery Configuration

apiVersion: rubrik.com/v1  
kind: VMRecovery  
metadata:  
 name: vm-instant-recovery  
 namespace: vm-workloads  
spec:  
 sourceVM:  
 name: "critical-app-vm"  
 namespace: "vm-workloads"  
 recoveryPoint:  
 snapshotId: "snapshot-id-from-rubrik"  
 # or use timestamp  
 # timestamp: "2024-01-15T10:30:00Z"  
 recoveryOptions:  
 type: "instant" # instant, full, or export  
 targetNamespace: "vm-recovery"  
 targetVMName: "critical-app-vm-recovered"  
 powerOn: true  
 preserveMAC: false  
 networkMapping:  
 - sourceNetwork: "production-network"  
 targetNetwork: "recovery-network"  
---  
### Alternative: Full VM Recovery  
apiVersion: rubrik.com/v1  
kind: VMRecovery  
metadata:  
 name: vm-full-recovery  
 namespace: vm-workloads  
spec:  
 sourceVM:  
 name: "critical-app-vm"  
 namespace: "vm-workloads"  
 recoveryPoint:  
 snapshotId: "latest"  
 recoveryOptions:  
 type: "full"  
 targetNamespace: "vm-workloads"  
 targetVMName: "critical-app-vm"  
 powerOn: false  
 overwriteExisting: true

###### Granular File Recovery

### Mount VM snapshot for file-level recovery  
rubrik vm mount --vm-name "critical-app-vm" \  
 --snapshot-id "snapshot-id" \  
 --mount-path "/mnt/recovery" \  
 --read-only  
  
### Extract specific files  
cp /mnt/recovery/important-file.txt /recovery/destination/  
  
### Unmount when recovery is complete  
rubrik vm unmount --mount-id "mount-id"

###### Cross-Platform Recovery

### Export VM to different platform (e.g., VMware to KubeVirt)  
apiVersion: rubrik.com/v1  
kind: VMExport  
metadata:  
 name: vm-cross-platform-recovery  
 namespace: vm-workloads  
spec:  
 sourceVM:  
 name: "legacy-vmware-vm"  
 platform: "vmware"  
 exportOptions:  
 targetPlatform: "kubevirt"  
 targetNamespace: "migrated-workloads"  
 format: "ova"  
 targetStorage:  
 storageClass: "fast-ssd"  
 size: "100Gi"  
 conversionOptions:  
 optimizeForKubernetes: true  
 removeVMwareTools: true  
 installKubernetesAgents: true

#### Troubleshooting Guide

##### Common Issues and Solutions

###### Rubrik Connector Issues

* **Issue**: Rubrik connector cannot communicate with CDM
* **Solution**:
  + Verify network connectivity between OpenShift cluster and Rubrik CDM
  + Check Rubrik credentials in the secret
  + Ensure proper RBAC permissions for the connector service account

###### Backup Failures

* **Issue**: VM backups failing or incomplete
* **Solution**:
  + Check VM quiesce settings and ensure VM tools are installed
  + Verify SLA domain configuration and policies
  + Monitor Rubrik cluster storage capacity
  + Review backup logs in Rubrik Polaris dashboard

###### Recovery Performance Issues

* **Issue**: Slow recovery or instant recovery not working
* **Solution**:
  + Verify network bandwidth between Rubrik and OpenShift cluster
  + Check storage performance on target environment
  + Use Rubrik’s storage optimization features
  + Consider using local recovery points for faster access

###### Replication and Archive Issues

* **Issue**: Replication to remote site or cloud archive failing
* **Solution**:
  + Verify bandwidth allocation and network policies
  + Check cloud credentials and permissions
  + Review replication target configuration
  + Monitor Rubrik cluster bandwidth utilization

#### Best Practices

* **Regular Backups**: Schedule regular backups and snapshots for critical VMs.
* **Test Recovery**: Regularly test recovery procedures using Rubrik capabilities to ensure data integrity.
* **Security and Compliance**: Leverage Rubrik’s built-in encryption and compliance features.

#### Integration with RH OVE Ecosystem

##### Monitoring and Alerting Integration

### Rubrik Metrics ServiceMonitor  
apiVersion: monitoring.coreos.com/v1  
kind: ServiceMonitor  
metadata:  
 name: rubrik-metrics  
 namespace: rubrik-system  
spec:  
 selector:  
 matchLabels:  
 app: rubrik-connector  
 endpoints:  
 - port: metrics  
 interval: 30s  
 path: /metrics  
---  
### Rubrik Backup Alerts  
apiVersion: monitoring.coreos.com/v1  
kind: PrometheusRule  
metadata:  
 name: rubrik-backup-alerts  
 namespace: rubrik-system  
spec:  
 groups:  
 - name: rubrik.backup  
 rules:  
 - alert: RubrikBackupFailure  
 expr: rubrik\_backup\_failures\_total > 0  
 for: 5m  
 labels:  
 severity: critical  
 annotations:  
 summary: "Rubrik backup failure detected"  
 description: "VM backup has failed for {{ $labels.vm\_name }}"  
   
 - alert: RubrikClusterCapacity  
 expr: rubrik\_cluster\_used\_capacity\_percentage > 85  
 for: 10m  
 labels:  
 severity: warning  
 annotations:  
 summary: "Rubrik cluster capacity high"  
 description: "Rubrik cluster is {{ $value }}% full"  
   
 - alert: RubrikReplicationDelay  
 expr: rubrik\_replication\_lag\_minutes > 60  
 for: 15m  
 labels:  
 severity: warning  
 annotations:  
 summary: "Rubrik replication lag detected"  
 description: "Replication is {{ $value }} minutes behind"

##### GitOps Integration

apiVersion: argoproj.io/v1alpha1  
kind: Application  
metadata:  
 name: rubrik-backup-policies  
 namespace: argocd  
spec:  
 project: platform-services  
 source:  
 repoURL: https://git.example.com/rubrik-backup-config  
 targetRevision: HEAD  
 path: policies/production  
 destination:  
 server: https://kubernetes.default.svc  
 namespace: vm-workloads  
 syncPolicy:  
 automated:  
 prune: false  
 selfHeal: true  
 syncOptions:  
 - CreateNamespace=true

##### Multi-Cluster Backup Management

* **Centralized Management**: Use Rubrik Polaris to manage backup policies across multiple OpenShift clusters
* **Cross-Cluster Recovery**: Implement recovery procedures that can restore VMs to different clusters
* **Global SLA Management**: Define consistent backup policies across the entire RH OVE ecosystem

This guide provides the necessary steps and best practices for implementing VM backup and recovery processes in the RH OVE environment, ensuring data resilience and availability for enterprise workloads.

## Hybrid Applications

### Use Case: Hybrid Application Deployment (VMs, Containers, and PaaS)

#### Business Context

Organizations increasingly adopt hybrid application models involving VMs, containers, and Platform-as-a-Service (PaaS) solutions to streamline operations and leverage the cloud. This use case demonstrates how to integrate these diverse technologies within the RH OVE multi-cluster ecosystem, maximizing flexibility and efficiency.

#### Technical Requirements

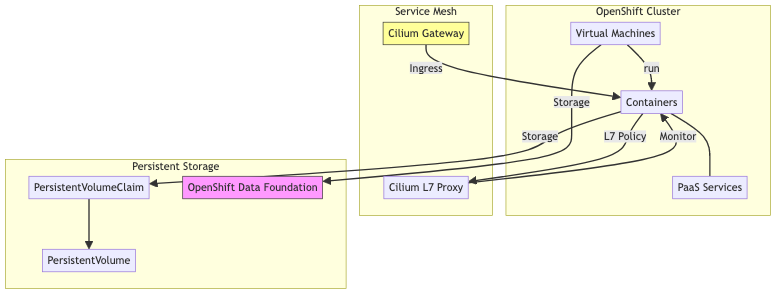
##### Infrastructure Requirements

* OpenShift 4.12+ with OpenShift Virtualization enabled
* KubeVirt for VM support
* Cilium CNI with service mesh capabilities
* Persistent storage solution (e.g., OpenShift Data Foundation)
* Multus for multi-network support

##### Resource Requirements

* **CPU**: 2+ cores per microservice or VM
* **Memory**: 4GB+ RAM per microservice or VM
* **Storage**: 20GB+ per microservice, scalable as needed
* **Network**: High throughput, low latency network configuration

#### Architecture Overview



#### Implementation Steps

##### Step 1: VM Integration

###### Provision Virtual Machines

apiVersion: kubevirt.io/v1  
kind: VirtualMachine  
metadata:  
 name: app-vm  
 namespace: hybrid-app  
spec:  
 running: true  
 template:  
 spec:  
 domain:  
 devices:  
 disks:  
 - disk:  
 bus: virtio  
 name: rootdisk  
 networks:  
 - networkName: default  
 pod: {}  
 volumes:  
 - dataVolume:  
 name: app-vm-dv  
 name: rootdisk  
---  
apiVersion: cdi.kubevirt.io/v1beta1  
kind: DataVolume  
metadata:  
 name: app-vm-dv  
 namespace: hybrid-app  
spec:  
 source:  
 http:  
 url: "https://vm-images.example.com/app-vm.img"  
 pvc:  
 accessModes:  
 - ReadWriteOnce  
 resources:  
 requests:  
 storage: 50Gi

##### Step 2: Container Integration

###### Deploy Containerized Services

apiVersion: apps/v1  
kind: Deployment  
metadata:  
 name: app-container  
 namespace: hybrid-app  
spec:  
 replicas: 3  
 selector:  
 matchLabels:  
 app: app-container  
 template:  
 metadata:  
 labels:  
 app: app-container  
 spec:  
 containers:  
 - name: app  
 image: quay.io/example/app-image:latest  
 ports:  
 - containerPort: 8080  
 volumes:  
 - name: app-storage  
 persistentVolumeClaim:  
 claimName: app-pvc  
---  
apiVersion: v1  
kind: PersistentVolumeClaim  
metadata:  
 name: app-pvc  
 namespace: hybrid-app  
spec:  
 accessModes:  
 - ReadWriteMany  
 resources:  
 requests:  
 storage: 100Gi

##### Step 3: Service Mesh Enablement

###### Configure Cilium Gateway API and L7 Policies

apiVersion: gateway.networking.k8s.io/v1beta1  
kind: Gateway  
metadata:  
 name: app-gateway  
 namespace: hybrid-app  
spec:  
 gatewayClassName: cilium  
 listeners:  
 - name: http  
 port: 80  
 protocol: HTTP  
 hostname: "app.example.com"  
---  
apiVersion: gateway.networking.k8s.io/v1beta1  
kind: HTTPRoute  
metadata:  
 name: app-routing  
 namespace: hybrid-app  
spec:  
 parentRefs:  
 - name: app-gateway  
 hostnames:  
 - "app.example.com"  
 rules:  
 - matches:  
 - path:  
 type: PathPrefix  
 value: "/"  
 backendRefs:  
 - name: app-container  
 port: 8080  
---  
apiVersion: "cilium.io/v2"  
kind: CiliumNetworkPolicy  
metadata:  
 name: app-l7-policy  
 namespace: hybrid-app  
spec:  
 endpointSelector:  
 matchLabels:  
 app: app-container  
 ingress:  
 - fromEndpoints:  
 - matchLabels:  
 "k8s:io.cilium.k8s.policy.cluster": "default"  
 toPorts:  
 - ports:  
 - port: "8080"  
 protocol: TCP  
 rules:  
 http:  
 - method: "GET"  
 - method: "POST"  
 - method: "PUT"  
 - method: "DELETE"  
  
##### Step 4: PaaS Integration  
  
###### Deploy Middleware and Database Services  
```yaml  
apiVersion: apps/v1  
kind: Deployment  
metadata:  
 name: middleware  
 namespace: hybrid-app  
spec:  
 replicas: 2  
 selector:  
 matchLabels:  
 app: middleware  
 template:  
 metadata:  
 labels:  
 app: middleware  
 spec:  
 containers:  
 - name: middleware  
 image: quay.io/example/middleware-image:latest  
 env:  
 - name: DB\_HOST  
 value: db.example.com  
 - name: DB\_USER  
 valueFrom:  
 secretKeyRef:  
 name: db-secrets  
 key: username  
---  
apiVersion: v1  
kind: Service  
metadata:  
 name: middleware-service  
 namespace: hybrid-app  
spec:  
 selector:  
 app: middleware  
 ports:  
 - protocol: TCP  
 port: 8080  
 targetPort: 8080  
 type: ClusterIP

##### Step 5: Continuous Integration/Continuous Deployment (CI/CD)

###### Integrate with ArgoCD

apiVersion: argoproj.io/v1alpha1  
kind: Application  
metadata:  
 name: hybrid-app  
 namespace: argocd  
spec:  
 project: default  
 source:  
 repoURL: https://git.example.com/hybrid-app-config  
 targetRevision: HEAD  
 path: hybrid  
 destination:  
 server: https://kubernetes.default.svc  
 namespace: hybrid-app  
 syncPolicy:  
 automated:  
 prune: false  
 selfHeal: true

#### Troubleshooting Guide

##### Common Issues and Solutions

###### VM Performance Degradation

* Check resource allocation and adjust CPU/RAM as needed.
* Ensure the underlying storage is not a bottleneck.

###### Service Mesh Connectivity Issues

* Validate Cilium Gateway API configurations and ensure L7 policies are applied.
* Check Cilium network policies and ensure proper endpoint selection.

###### Storage Access Issues

* Verify PVC binding and ensure correct storage class is applied.
* Check application logs for connectivity and permission errors.

#### Best Practices

* **Resource Management**: Dynamic scaling policies for both VMs and containers.
* **Network Policies**: Zero-trust approach with Cilium network policies for clear segmentation.
* **Security Hardening**: Apply security best practices for VM and container images with Cilium L7 policies.
* **Monitoring and Logging**: Unified logging solutions using Fluentd or Logstash with direct integration to Prometheus and Cilium Hubble for network observability.

#### Integration with RH OVE Ecosystem

##### Monitoring Integration

### Monitoring and alerting configuration  
apiVersion: monitoring.coreos.com/v1  
kind: ServiceMonitor  
metadata:  
 name: hybrid-app-monitor  
 namespace: hybrid-app  
spec:  
 selector:  
 matchLabels:  
 app: app-container  
 endpoints:  
 - port: metrics  
 interval: 30s  
 path: /metrics

This comprehensive guide provides everything needed to deploy and manage hybrid applications within the RH OVE multi-cluster ecosystem while adhering to enterprise-grade security, monitoring, and operational standards.

## Setup Multi Env Application

### Use Case: Setup a New Application with 3 Environments (Dev, Lat, Prod)

#### Business Context

Setting up distinct environments for development, testing (LAT - User Acceptance Testing level), and production is crucial for ensuring software quality and stability. This use case demonstrates deploying applications across multiple clusters and namespaces, with development running on a separate cluster for enhanced isolation, while LAT and production environments share a cluster but use different namespaces.

**Key Focus: Label-Based Application Management** This use case emphasizes the strategic use of Kubernetes labels for effective application lifecycle management, environment identification, resource organization, and automated operations across multi-cluster deployments.

#### Label Management Strategy

##### Core Labeling Standards

This implementation follows a comprehensive labeling strategy for consistent resource management:

###### Standard Label Schema

### Application identification labels  
app.kubernetes.io/name: <application-name> # Application name  
app.kubernetes.io/version: <version> # Application version  
app.kubernetes.io/component: <component> # Component type (frontend, backend, database)  
app.kubernetes.io/part-of: <system> # Higher-level application/system  
app.kubernetes.io/managed-by: <tool> # Management tool (argocd, helm, etc.)  
  
### Environment and deployment labels  
environment: <env> # Environment (dev, lat, prod)   
tier: <tier> # Tier (production, non-production)  
cluster: <cluster-name> # Target cluster identifier  
region: <region> # Geographic region  
  
### Operational labels  
owner: <team> # Owning team  
cost-center: <code> # Cost allocation  
monitoring: <enabled/disabled> # Monitoring status  
backup: <policy> # Backup policy

###### Label Selection and Filtering Examples

### Select all resources for a specific application  
kubectl get all -l app.kubernetes.io/name=sample-app  
  
### Select resources by environment  
kubectl get pods -l environment=prod  
  
### Select resources by cluster and environment  
kubectl get deployments -l cluster=prod-cluster,environment=lat  
  
### Select all production tier resources  
kubectl get services -l tier=production  
  
### Select resources managed by ArgoCD  
kubectl get all -l app.kubernetes.io/managed-by=argocd

#### Technical Requirements

##### Infrastructure Requirements

* **Development Cluster**: Dedicated OpenShift 4.12+ cluster for development workloads
* **Production Cluster**: OpenShift 4.12+ cluster hosting LAT and Production environments
* Red Hat Advanced Cluster Management (RHACM) for multi-cluster management
* Network connectivity between clusters for GitOps and management operations
* Shared container registry accessible by both clusters

##### Resource Requirements

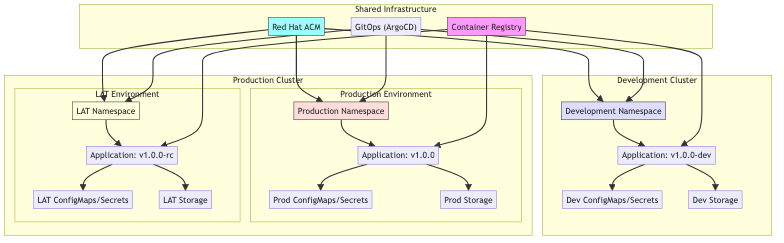
###### Development Cluster

* **CPU**: 4-8 cores for development workloads
* **Memory**: 16-32GB RAM for development environments
* **Storage**: Fast local storage for rapid development cycles
* **Network**: Internal network access, limited external exposure

###### Production Cluster

* **CPU**: 16-32 cores for LAT and Production workloads
* **Memory**: 64-128GB RAM for production-grade environments
* **Storage**: Enterprise-grade persistent storage with backup capabilities
* **Network**: Full production network access with security controls

#### Architecture Overview



#### Implementation Steps

##### Step 1: Setup Multi-Cluster Management with RHACM

###### Deploy RHACM Hub Cluster

apiVersion: operator.open-cluster-management.io/v1  
kind: MultiClusterHub  
metadata:  
 name: multiclusterhub  
 namespace: open-cluster-management  
spec:  
 availabilityConfig: High  
 enableClusterBackup: true  
 imagePullSecret: multiclusterhub-operator-pull-secret

###### Register Development Cluster as Managed Cluster

apiVersion: cluster.open-cluster-management.io/v1  
kind: ManagedCluster  
metadata:  
 name: dev-cluster  
 labels:  
 # Cluster identification labels  
 cloud: auto-detect  
 vendor: OpenShift  
 environment: development  
 cluster-role: development  
 # Management labels  
 managed-by: rhacm  
 region: us-east-1  
 tier: non-production  
 # Operational labels  
 monitoring: enabled  
 cost-center: dev-ops  
 owner: platform-team  
spec:  
 hubAcceptsClient: true  
---  
apiVersion: agent.open-cluster-management.io/v1  
kind: KlusterletAddonConfig  
metadata:  
 name: dev-cluster  
 namespace: dev-cluster  
spec:  
 clusterName: dev-cluster  
 clusterNamespace: dev-cluster  
 clusterLabels:  
 cloud: auto-detect  
 vendor: OpenShift  
 environment: development  
 cluster-role: development  
 tier: non-production  
 region: us-east-1  
 applicationManager:  
 enabled: true  
 policyController:  
 enabled: true  
 searchCollector:  
 enabled: true  
 certPolicyController:  
 enabled: true  
 iamPolicyController:  
 enabled: true

##### Step 2: Create Namespaces Across Clusters

###### Development Cluster - Namespace Configuration

### Apply to Development Cluster  
apiVersion: v1  
kind: Namespace  
metadata:  
 name: development  
 labels:  
 # Standard Kubernetes labels  
 name: development  
 app.kubernetes.io/name: sample-app  
 app.kubernetes.io/part-of: sample-system  
 app.kubernetes.io/managed-by: argocd  
   
 # Environment and deployment labels  
 environment: dev  
 tier: non-production  
 cluster: dev-cluster  
 region: us-east-1  
   
 # Operational labels  
 owner: dev-team  
 cost-center: engineering  
 monitoring: enabled  
 backup: daily  
   
 # Feature flags for this environment  
 feature.experimental: "true"  
 feature.debug: "true"  
 annotations:  
 openshift.io/description: "Development environment for sample application"  
 openshift.io/display-name: "Sample App - Development"  
 cluster.open-cluster-management.io/managedCluster: "dev-cluster"  
 # Label management annotations  
 label-policy.io/required: "environment,tier,owner"  
 label-policy.io/validation: "strict"

###### Production Cluster - LAT and Production Namespaces

### Apply to Production Cluster  
apiVersion: v1  
kind: Namespace  
metadata:  
 name: lat  
 labels:  
 # Standard Kubernetes labels  
 name: lat  
 app.kubernetes.io/name: sample-app  
 app.kubernetes.io/part-of: sample-system  
 app.kubernetes.io/managed-by: argocd  
   
 # Environment and deployment labels  
 environment: lat  
 tier: non-production  
 cluster: prod-cluster  
 region: us-east-1  
   
 # Operational labels  
 owner: qa-team  
 cost-center: quality-assurance  
 monitoring: enabled  
 backup: weekly  
   
 # Feature flags for LAT environment  
 feature.experimental: "false"  
 feature.debug: "false"  
 feature.performance-testing: "true"  
 annotations:  
 openshift.io/description: "LAT/UAT environment for sample application"  
 openshift.io/display-name: "Sample App - LAT"  
 cluster.open-cluster-management.io/managedCluster: "prod-cluster"  
 label-policy.io/required: "environment,tier,owner"  
 label-policy.io/validation: "strict"  
---  
apiVersion: v1  
kind: Namespace  
metadata:  
 name: production  
 labels:  
 # Standard Kubernetes labels  
 name: production  
 app.kubernetes.io/name: sample-app  
 app.kubernetes.io/part-of: sample-system  
 app.kubernetes.io/managed-by: argocd  
   
 # Environment and deployment labels  
 environment: prod  
 tier: production  
 cluster: prod-cluster  
 region: us-east-1  
   
 # Operational labels  
 owner: platform-team  
 cost-center: production-ops  
 monitoring: enhanced  
 backup: continuous  
   
 # Production-specific feature flags  
 feature.experimental: "false"  
 feature.debug: "false"  
 feature.high-availability: "true"  
 feature.auto-scaling: "true"  
 annotations:  
 openshift.io/description: "Production environment for sample application"  
 openshift.io/display-name: "Sample App - Production"  
 cluster.open-cluster-management.io/managedCluster: "prod-cluster"  
 label-policy.io/required: "environment,tier,owner,backup"  
 label-policy.io/validation: "strict"

##### Step 3: Configure Cross-Cluster GitOps with ArgoCD

###### ArgoCD ApplicationSet for Multi-Cluster Deployment with Label-Based Selection

apiVersion: argoproj.io/v1alpha1  
kind: ApplicationSet  
metadata:  
 name: sample-app-multi-env  
 namespace: argocd  
 labels:  
 # ApplicationSet identification  
 app.kubernetes.io/name: sample-app  
 app.kubernetes.io/component: applicationset  
 app.kubernetes.io/managed-by: argocd  
 app.kubernetes.io/part-of: sample-system  
 # Management labels  
 owner: platform-team  
 deployment-strategy: multi-cluster  
spec:  
 generators:  
 # Development cluster selector  
 - clusters:  
 selector:  
 matchLabels:  
 environment: development  
 tier: non-production  
 values:  
 environment: development  
 namespace: development  
 imageTag: "1.0.0-dev"  
 replicas: "2"  
 resources: "small"  
 cluster\_type: "development"  
 # LAT environment selector (on production cluster)  
 - clusters:  
 selector:  
 matchLabels:  
 cluster-role: production  
 tier: production  
 values:  
 environment: lat  
 namespace: lat  
 imageTag: "1.0.0-rc"  
 replicas: "2"  
 resources: "medium"  
 cluster\_type: "production"  
 # Production environment selector  
 - clusters:  
 selector:  
 matchLabels:  
 cluster-role: production  
 tier: production  
 values:  
 environment: production  
 namespace: production  
 imageTag: "1.0.0"  
 replicas: "5"  
 resources: "large"  
 cluster\_type: "production"  
 template:  
 metadata:  
 name: 'sample-app-{{values.environment}}'  
 labels:  
 # Application labels for ArgoCD application  
 app.kubernetes.io/name: sample-app  
 app.kubernetes.io/component: application  
 app.kubernetes.io/managed-by: argocd  
 environment: '{{values.environment}}'  
 cluster-type: '{{values.cluster\_type}}'  
 spec:  
 project: default  
 source:  
 repoURL: https://git.example.com/sample-app-config  
 targetRevision: HEAD  
 path: overlays/{{values.environment}}  
 helm:  
 parameters:  
 - name: image.tag  
 value: '{{values.imageTag}}'  
 - name: replicaCount  
 value: '{{values.replicas}}'  
 - name: resources.profile  
 value: '{{values.resources}}'  
 - name: environment  
 value: '{{values.environment}}'  
 - name: cluster\_type  
 value: '{{values.cluster\_type}}'  
 destination:  
 server: '{{server}}'  
 namespace: '{{values.namespace}}'  
 syncPolicy:  
 automated:  
 prune: true  
 selfHeal: true  
 syncOptions:  
 - CreateNamespace=true  
 - RespectIgnoreDifferences=true

##### Step 4: Configure RBAC and Service Accounts

###### Development Cluster Service Account with Enhanced Labels

### Apply to Development Cluster  
apiVersion: v1  
kind: ServiceAccount  
metadata:  
 name: sample-app-sa  
 namespace: development  
 labels:  
 # Standard Kubernetes labels  
 app.kubernetes.io/name: sample-app  
 app.kubernetes.io/component: serviceaccount  
 app.kubernetes.io/managed-by: kubectl  
 app.kubernetes.io/part-of: sample-system  
   
 # Environment and deployment labels  
 app: sample-app  
 environment: dev  
 cluster: dev-cluster  
 tier: non-production  
   
 # RBAC and security labels  
 security.io/rbac: enabled  
 security.io/scope: namespace  
   
 # Operational labels  
 owner: dev-team  
 managed-by: platform-team  
---  
apiVersion: rbac.authorization.k8s.io/v1  
kind: Role  
metadata:  
 name: sample-app-role  
 namespace: development  
rules:  
- apiGroups: [""]  
 resources: ["configmaps", "secrets", "pods", "services"]  
 verbs: ["get", "list", "watch"]  
- apiGroups: ["apps"]  
 resources: ["deployments", "replicasets"]  
 verbs: ["get", "list", "watch"]  
---  
apiVersion: rbac.authorization.k8s.io/v1  
kind: RoleBinding  
metadata:  
 name: sample-app-rolebinding  
 namespace: development  
subjects:  
- kind: ServiceAccount  
 name: sample-app-sa  
 namespace: development  
roleRef:  
 kind: Role  
 name: sample-app-role  
 apiGroup: rbac.authorization.k8s.io

###### Production Cluster Service Accounts with Enhanced Labels

### Apply to Production Cluster - LAT Environment  
apiVersion: v1  
kind: ServiceAccount  
metadata:  
 name: sample-app-sa  
 namespace: lat  
 labels:  
 # Standard Kubernetes labels  
 app.kubernetes.io/name: sample-app  
 app.kubernetes.io/component: serviceaccount  
 app.kubernetes.io/managed-by: kubectl  
 app.kubernetes.io/part-of: sample-system  
   
 # Environment and deployment labels  
 app: sample-app  
 environment: lat  
 cluster: prod-cluster  
 tier: non-production  
   
 # RBAC and security labels  
 security.io/rbac: enabled  
 security.io/scope: namespace  
 security.io/audit: enabled  
   
 # Operational labels  
 owner: qa-team  
 managed-by: platform-team  
---  
### Apply to Production Cluster - Production Environment  
apiVersion: v1  
kind: ServiceAccount  
metadata:  
 name: sample-app-sa  
 namespace: production  
 labels:  
 # Standard Kubernetes labels  
 app.kubernetes.io/name: sample-app  
 app.kubernetes.io/component: serviceaccount  
 app.kubernetes.io/managed-by: kubectl  
 app.kubernetes.io/part-of: sample-system  
   
 # Environment and deployment labels  
 app: sample-app  
 environment: prod  
 cluster: prod-cluster  
 tier: production  
   
 # RBAC and security labels  
 security.io/rbac: enabled  
 security.io/scope: namespace  
 security.io/audit: enhanced  
 security.io/compliance: required  
   
 # Operational labels  
 owner: platform-team  
 managed-by: platform-team  
 criticality: high

##### Step 5: Configure Environment Specific Resources

###### Development Environment Configuration (Apply to Development Cluster)

apiVersion: v1  
kind: ConfigMap  
metadata:  
 name: app-config  
 namespace: development  
 labels:  
 app: sample-app  
 environment: dev  
 cluster: dev-cluster  
 app.kubernetes.io/name: sample-app  
 app.kubernetes.io/component: config  
data:  
 APP\_ENV: "development"  
 LOG\_LEVEL: "debug"  
 DATABASE\_HOST: "dev-postgres.development.svc.cluster.local"  
 DATABASE\_PORT: "5432"  
 DATABASE\_NAME: "sampleapp\_dev"  
 REDIS\_HOST: "dev-redis.development.svc.cluster.local"  
 REDIS\_PORT: "6379"  
 API\_BASE\_URL: "https://api-dev.example.com"  
 FEATURE\_FLAGS: "new-ui:true,beta-features:true"  
 MAX\_CONNECTIONS: "10"  
 CACHE\_TTL: "300"  
 CLUSTER\_NAME: "dev-cluster"  
---  
apiVersion: v1  
kind: Secret  
metadata:  
 name: app-secrets  
 namespace: development  
 labels:  
 app: sample-app  
 environment: dev  
 app.kubernetes.io/name: sample-app  
 app.kubernetes.io/component: secrets  
type: Opaque  
stringData:  
 DATABASE\_PASSWORD: "dev\_password\_123"  
 REDIS\_PASSWORD: "dev\_redis\_password"  
 JWT\_SECRET: "dev\_jwt\_secret\_key"  
 API\_KEY: "dev\_api\_key\_12345"  
 ENCRYPTION\_KEY: "dev\_encryption\_key"  
---  
### LAT Environment Configuration (Apply to Production Cluster)  
apiVersion: v1  
kind: ConfigMap  
metadata:  
 name: app-config  
 namespace: lat  
 labels:  
 app: sample-app  
 environment: lat  
 cluster: prod-cluster  
 app.kubernetes.io/name: sample-app  
 app.kubernetes.io/component: config  
data:  
 APP\_ENV: "lat"  
 LOG\_LEVEL: "info"  
 DATABASE\_HOST: "lat-postgres.lat.svc.cluster.local"  
 DATABASE\_PORT: "5432"  
 DATABASE\_NAME: "sampleapp\_lat"  
 REDIS\_HOST: "lat-redis.lat.svc.cluster.local"  
 REDIS\_PORT: "6379"  
 API\_BASE\_URL: "https://api-lat.example.com"  
 FEATURE\_FLAGS: "new-ui:true,beta-features:false"  
 MAX\_CONNECTIONS: "20"  
 CACHE\_TTL: "600"  
 CLUSTER\_NAME: "prod-cluster"  
---  
apiVersion: v1  
kind: Secret  
metadata:  
 name: app-secrets  
 namespace: lat  
 labels:  
 app: sample-app  
 environment: lat  
 app.kubernetes.io/name: sample-app  
 app.kubernetes.io/component: secrets  
type: Opaque  
stringData:  
 DATABASE\_PASSWORD: "lat\_password\_456"  
 REDIS\_PASSWORD: "lat\_redis\_password"  
 JWT\_SECRET: "lat\_jwt\_secret\_key"  
 API\_KEY: "lat\_api\_key\_67890"  
 ENCRYPTION\_KEY: "lat\_encryption\_key"  
---  
### Production Environment Configuration (Apply to Production Cluster)  
apiVersion: v1  
kind: ConfigMap  
metadata:  
 name: app-config  
 namespace: production  
 labels:  
 app: sample-app  
 environment: prod  
 cluster: prod-cluster  
 app.kubernetes.io/name: sample-app  
 app.kubernetes.io/component: config  
data:  
 APP\_ENV: "production"  
 LOG\_LEVEL: "warn"  
 DATABASE\_HOST: "prod-postgres.production.svc.cluster.local"  
 DATABASE\_PORT: "5432"  
 DATABASE\_NAME: "sampleapp\_prod"  
 REDIS\_HOST: "prod-redis.production.svc.cluster.local"  
 REDIS\_PORT: "6379"  
 API\_BASE\_URL: "https://api.example.com"  
 FEATURE\_FLAGS: "new-ui:false,beta-features:false"  
 MAX\_CONNECTIONS: "50"  
 CACHE\_TTL: "3600"  
 CLUSTER\_NAME: "prod-cluster"  
---  
apiVersion: v1  
kind: Secret  
metadata:  
 name: app-secrets  
 namespace: production  
 labels:  
 app: sample-app  
 environment: prod  
 app.kubernetes.io/name: sample-app  
 app.kubernetes.io/component: secrets  
type: Opaque  
stringData:  
 DATABASE\_PASSWORD: "prod\_secure\_password\_789"  
 REDIS\_PASSWORD: "prod\_redis\_secure\_password"  
 JWT\_SECRET: "prod\_jwt\_secret\_key\_secure"  
 API\_KEY: "prod\_api\_key\_secure\_abcdef"  
 ENCRYPTION\_KEY: "prod\_encryption\_key\_secure"

##### Step 6: Deploy Application Across Clusters and Environments

###### Development Environment Deployment (Apply to Development Cluster)

apiVersion: apps/v1  
kind: Deployment  
metadata:  
 name: sample-app  
 namespace: development  
 labels:  
 app: sample-app  
 environment: dev  
 cluster: dev-cluster  
 app.kubernetes.io/name: sample-app  
 app.kubernetes.io/version: "1.0.0"  
 app.kubernetes.io/component: application  
spec:  
 replicas: 2  
 strategy:  
 type: RollingUpdate  
 rollingUpdate:  
 maxUnavailable: 1  
 maxSurge: 1  
 selector:  
 matchLabels:  
 app: sample-app  
 environment: dev  
 template:  
 metadata:  
 labels:  
 app: sample-app  
 environment: dev  
 app.kubernetes.io/name: sample-app  
 app.kubernetes.io/version: "1.0.0"  
 spec:  
 serviceAccountName: sample-app-sa  
 containers:  
 - name: sample-app  
 image: quay.io/example/sample-app:1.0.0-dev  
 imagePullPolicy: Always  
 ports:  
 - containerPort: 8080  
 name: http  
 protocol: TCP  
 envFrom:  
 - configMapRef:  
 name: app-config  
 - secretRef:  
 name: app-secrets  
 resources:  
 requests:  
 memory: "256Mi"  
 cpu: "100m"  
 limits:  
 memory: "512Mi"  
 cpu: "200m"  
 livenessProbe:  
 httpGet:  
 path: /health/live  
 port: 8080  
 initialDelaySeconds: 30  
 periodSeconds: 10  
 timeoutSeconds: 5  
 failureThreshold: 3  
 readinessProbe:  
 httpGet:  
 path: /health/ready  
 port: 8080  
 initialDelaySeconds: 5  
 periodSeconds: 5  
 timeoutSeconds: 3  
 failureThreshold: 3  
 volumeMounts:  
 - name: app-data  
 mountPath: /data  
 - name: temp-storage  
 mountPath: /tmp  
 volumes:  
 - name: app-data  
 persistentVolumeClaim:  
 claimName: sample-app-data  
 - name: temp-storage  
 emptyDir: {}  
---  
apiVersion: v1  
kind: Service  
metadata:  
 name: sample-app-service  
 namespace: development  
 labels:  
 app: sample-app  
 environment: dev  
spec:  
 selector:  
 app: sample-app  
 environment: dev  
 ports:  
 - name: http  
 port: 80  
 targetPort: 8080  
 protocol: TCP  
 type: ClusterIP  
---  
### LAT Environment Deployment  
apiVersion: apps/v1  
kind: Deployment  
metadata:  
 name: sample-app  
 namespace: lat  
 labels:  
 app: sample-app  
 environment: lat  
 app.kubernetes.io/name: sample-app  
 app.kubernetes.io/version: "1.0.0"  
 app.kubernetes.io/component: application  
spec:  
 replicas: 2  
 strategy:  
 type: RollingUpdate  
 rollingUpdate:  
 maxUnavailable: 0  
 maxSurge: 1  
 selector:  
 matchLabels:  
 app: sample-app  
 environment: lat  
 template:  
 metadata:  
 labels:  
 app: sample-app  
 environment: lat  
 app.kubernetes.io/name: sample-app  
 app.kubernetes.io/version: "1.0.0"  
 spec:  
 serviceAccountName: sample-app-sa  
 containers:  
 - name: sample-app  
 image: quay.io/example/sample-app:1.0.0-rc  
 imagePullPolicy: IfNotPresent  
 ports:  
 - containerPort: 8080  
 name: http  
 protocol: TCP  
 envFrom:  
 - configMapRef:  
 name: app-config  
 - secretRef:  
 name: app-secrets  
 resources:  
 requests:  
 memory: "512Mi"  
 cpu: "250m"  
 limits:  
 memory: "1Gi"  
 cpu: "500m"  
 livenessProbe:  
 httpGet:  
 path: /health/live  
 port: 8080  
 initialDelaySeconds: 30  
 periodSeconds: 10  
 timeoutSeconds: 5  
 failureThreshold: 3  
 readinessProbe:  
 httpGet:  
 path: /health/ready  
 port: 8080  
 initialDelaySeconds: 5  
 periodSeconds: 5  
 timeoutSeconds: 3  
 failureThreshold: 3  
 volumeMounts:  
 - name: app-data  
 mountPath: /data  
 - name: temp-storage  
 mountPath: /tmp  
 volumes:  
 - name: app-data  
 persistentVolumeClaim:  
 claimName: sample-app-data  
 - name: temp-storage  
 emptyDir: {}  
---  
### Production Environment Deployment  
apiVersion: apps/v1  
kind: Deployment  
metadata:  
 name: sample-app  
 namespace: production  
 labels:  
 app: sample-app  
 environment: prod  
 app.kubernetes.io/name: sample-app  
 app.kubernetes.io/version: "1.0.0"  
 app.kubernetes.io/component: application  
spec:  
 replicas: 5  
 strategy:  
 type: RollingUpdate  
 rollingUpdate:  
 maxUnavailable: 1  
 maxSurge: 2  
 selector:  
 matchLabels:  
 app: sample-app  
 environment: prod  
 template:  
 metadata:  
 labels:  
 app: sample-app  
 environment: prod  
 app.kubernetes.io/name: sample-app  
 app.kubernetes.io/version: "1.0.0"  
 spec:  
 serviceAccountName: sample-app-sa  
 containers:  
 - name: sample-app  
 image: quay.io/example/sample-app:1.0.0  
 imagePullPolicy: IfNotPresent  
 ports:  
 - containerPort: 8080  
 name: http  
 protocol: TCP  
 envFrom:  
 - configMapRef:  
 name: app-config  
 - secretRef:  
 name: app-secrets  
 resources:  
 requests:  
 memory: "1Gi"  
 cpu: "500m"  
 limits:  
 memory: "2Gi"  
 cpu: "1"  
 livenessProbe:  
 httpGet:  
 path: /health/live  
 port: 8080  
 initialDelaySeconds: 60  
 periodSeconds: 10  
 timeoutSeconds: 5  
 failureThreshold: 3  
 readinessProbe:  
 httpGet:  
 path: /health/ready  
 port: 8080  
 initialDelaySeconds: 10  
 periodSeconds: 5  
 timeoutSeconds: 3  
 failureThreshold: 3  
 volumeMounts:  
 - name: app-data  
 mountPath: /data  
 - name: temp-storage  
 mountPath: /tmp  
 volumes:  
 - name: app-data  
 persistentVolumeClaim:  
 claimName: sample-app-data  
 - name: temp-storage  
 emptyDir: {}  
 affinity:  
 podAntiAffinity:  
 preferredDuringSchedulingIgnoredDuringExecution:  
 - weight: 100  
 podAffinityTerm:  
 labelSelector:  
 matchExpressions:  
 - key: app  
 operator: In  
 values:  
 - sample-app  
 topologyKey: kubernetes.io/hostname

##### Step 5: Configure Persistent Storage

###### PersistentVolumeClaims for Each Environment

apiVersion: v1  
kind: PersistentVolumeClaim  
metadata:  
 name: sample-app-data  
 namespace: development  
 labels:  
 app: sample-app  
 environment: dev  
spec:  
 accessModes:  
 - ReadWriteOnce  
 resources:  
 requests:  
 storage: 5Gi  
 storageClassName: ocs-storagecluster-ceph-rbd  
---  
apiVersion: v1  
kind: PersistentVolumeClaim  
metadata:  
 name: sample-app-data  
 namespace: lat  
 labels:  
 app: sample-app  
 environment: lat  
spec:  
 accessModes:  
 - ReadWriteOnce  
 resources:  
 requests:  
 storage: 10Gi  
 storageClassName: ocs-storagecluster-ceph-rbd  
---  
apiVersion: v1  
kind: PersistentVolumeClaim  
metadata:  
 name: sample-app-data  
 namespace: production  
 labels:  
 app: sample-app  
 environment: prod  
spec:  
 accessModes:  
 - ReadWriteOnce  
 resources:  
 requests:  
 storage: 50Gi  
 storageClassName: ocs-storagecluster-ceph-rbd

##### Step 6: Configure Network and Resource Policies

###### Network Policies

apiVersion: networking.k8s.io/v1  
kind: NetworkPolicy  
metadata:  
 name: allow-specific  
 namespace: development  
spec:  
 podSelector:  
 matchLabels:  
 app: sample-app  
 policyTypes:  
 - Ingress  
 - Egress  
 ingress:  
 - from:  
 - ipBlock:  
 cidr: 10.0.0.0/8  
 egress:  
 - to:  
 - ipBlock:  
 cidr: 10.0.0.0/8  
---  
### Repeat for LAT and Prod with respective rules

###### Resource Quotas

apiVersion: v1  
kind: ResourceQuota  
metadata:  
 name: compute-resources  
 namespace: development  
spec:  
 hard:  
 requests.cpu: "2"  
 requests.memory: 4Gi  
 limits.cpu: "4"  
 limits.memory: 8Gi  
---  
### Repeat for LAT and Prod with tailored allocations

##### Validation and Testing

###### Validate Deployments

kubectl get deployments -n development  
kubectl get pods -n development  
kubectl get services -n development

* Verify that all resources are correctly deployed and running.

###### Test Application Functionality

* Ensure each environment functions as expected from development to LAT and into production.

##### Label-Based Management Operations

###### Advanced Label Queries and Operations

### Application lifecycle management using labels  
  
### Find all resources belonging to sample-app across all environments  
kubectl get all --all-namespaces -l app.kubernetes.io/name=sample-app  
  
### Get all production tier resources  
kubectl get all --all-namespaces -l tier=production  
  
### List all resources managed by ArgoCD  
kubectl get all --all-namespaces -l app.kubernetes.io/managed-by=argocd  
  
### Find resources by owner team  
kubectl get all --all-namespaces -l owner=platform-team  
  
### Get all development environment resources  
kubectl get all --all-namespaces -l environment=dev  
  
### Find all resources with backup enabled  
kubectl get all --all-namespaces -l backup!=none  
  
### Complex queries combining multiple labels  
kubectl get deployments --all-namespaces -l 'app.kubernetes.io/name=sample-app,tier=production'  
  
### Scale all deployments in development environment  
kubectl scale deployments -l environment=dev --replicas=1 --all-namespaces  
  
### Delete all test resources (be careful!)  
kubectl delete all -l environment=test --all-namespaces

###### Label-Based Resource Policies

### OPA Gatekeeper policy to enforce required labels  
apiVersion: templates.gatekeeper.sh/v1beta1  
kind: ConstraintTemplate  
metadata:  
 name: k8srequiredlabels  
spec:  
 crd:  
 spec:  
 names:  
 kind: K8sRequiredLabels  
 validation:  
 type: object  
 properties:  
 labels:  
 type: array  
 items:  
 type: string  
 targets:  
 - target: admission.k8s.gatekeeper.sh  
 rego: |  
 package k8srequiredlabels  
   
 violation[{"msg": msg}] {  
 required := input.parameters.labels  
 provided := input.review.object.metadata.labels  
 missing := required[\_]  
 not provided[missing]  
 msg := sprintf("Missing required label: %v", [missing])  
 }  
---  
apiVersion: constraints.gatekeeper.sh/v1beta1  
kind: K8sRequiredLabels  
metadata:  
 name: must-have-environment  
spec:  
 match:  
 kinds:  
 - apiGroups: ["apps"]  
 kinds: ["Deployment"]  
 - apiGroups: [""]  
 kinds: ["Service"]  
 parameters:  
 labels: ["environment", "owner", "app.kubernetes.io/name"]

##### Best Practices for Label Management

###### Label Strategy Best Practices

* **Consistent Naming**: Use standardized label keys across all environments
* **Hierarchical Organization**: Implement app.kubernetes.io standard labels
* **Environment Identification**: Always include environment and tier labels
* **Ownership Tracking**: Include owner and cost-center labels for accountability
* **Automation Friendly**: Design labels for automated selection and management
* **Policy Enforcement**: Use admission controllers to enforce required labels

###### Configuration Management

* **Consistent Configuration Management**: Use ConfigMaps and Secrets effectively across environments.
* **Isolation**: Ensure namespaces are isolated to prevent cross-environment contamination.
* **Resource Management**: Tailor quotas and limits to reflect environment purposes.
* **Automation**: Implement CI/CD pipelines for streamlined deployments.
* **Monitoring**: Use Prometheus and Grafana for monitoring and alerts.

###### Label-Based Monitoring and Alerting

### Prometheus monitoring rules using labels  
apiVersion: monitoring.coreos.com/v1  
kind: PrometheusRule  
metadata:  
 name: sample-app-alerts  
 labels:  
 app.kubernetes.io/name: sample-app  
 monitoring: prometheus  
spec:  
 groups:  
 - name: sample-app  
 rules:  
 - alert: ApplicationDown  
 expr: up{app\_kubernetes\_io\_name="sample-app"} == 0  
 for: 5m  
 labels:  
 severity: critical  
 environment: "{{ $labels.environment }}"  
 owner: "{{ $labels.owner }}"  
 annotations:  
 summary: "Sample app is down in {{ $labels.environment }}"  
 description: "Application {{ $labels.app\_kubernetes\_io\_name }} in {{ $labels.environment }} environment has been down for more than 5 minutes."

#### Integration with RH OVE Ecosystem

* **GitOps Integration**: Use ArgoCD for deploying and managing environment configurations.
* **Policy Automation**: Enforce standards and policies through OPA Gatekeeper.
* **Multi-Cluster Management**: Utilize RHACM for managing environments across multiple clusters.

This guide provides a clear pathway for setting up a robust, multi-environment application deployment workflow within RH OVE using Kubernetes namespaces, ensuring seamless transitions from development through to production.

## Database Services Paas

### Use Case: Database Services as Platform-as-a-Service (PaaS)

#### Business Context

Organizations require flexible, scalable database solutions that can support various application needs while minimizing operational overhead. This use case demonstrates how to implement database services as Platform-as-a-Service (PaaS) offerings within the RH OVE ecosystem, providing self-service database provisioning, automated management, and seamless integration with application workloads.

#### Technical Requirements

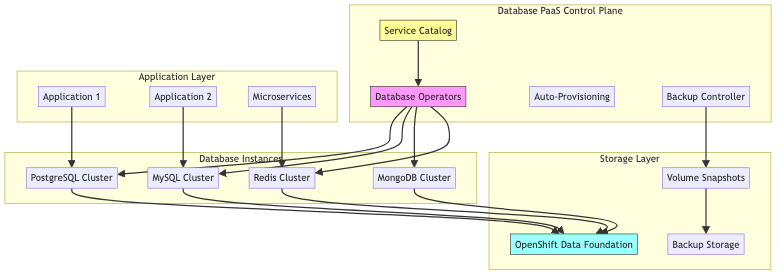
##### Infrastructure Requirements

* OpenShift 4.12+ with KubeVirt enabled
* OpenShift Data Foundation for persistent storage
* Database operators (PostgreSQL, MySQL, MongoDB, Redis operators)
* Service mesh for secure communication (OpenShift Service Mesh)
* Monitoring and observability stack (Prometheus, Grafana)
* Backup and disaster recovery solutions (OADP/Velero/Rubrik)

##### Resource Requirements

* **CPU**: Variable based on database workload (2-16 cores per instance)
* **Memory**: 4GB-64GB RAM per database instance
* **Storage**: High-performance persistent storage with snapshot capabilities
* **Network**: Low-latency networking for database replication and client connections

#### Architecture Overview



#### Implementation Steps

##### Step 1: Install Database Operators

###### PostgreSQL Operator Deployment

apiVersion: operators.coreos.com/v1alpha1  
kind: Subscription  
metadata:  
 name: postgresql-operator  
 namespace: database-operators  
spec:  
 channel: stable  
 name: postgresql-operator  
 source: certified-operators  
 sourceNamespace: openshift-marketplace  
---  
apiVersion: v1  
kind: Namespace  
metadata:  
 name: database-operators  
 labels:  
 openshift.io/cluster-monitoring: "true"

###### MySQL Operator Deployment

apiVersion: operators.coreos.com/v1alpha1  
kind: Subscription  
metadata:  
 name: mysql-operator  
 namespace: database-operators  
spec:  
 channel: stable  
 name: mysql-operator  
 source: certified-operators  
 sourceNamespace: openshift-marketplace

##### Step 2: Create Database Service Templates

###### PostgreSQL Service Template

apiVersion: postgresql.cnpg.io/v1  
kind: Cluster  
metadata:  
 name: postgres-template  
 namespace: database-services  
spec:  
 instances: 3  
   
 postgresql:  
 parameters:  
 max\_connections: "200"  
 shared\_buffers: "256MB"  
 effective\_cache\_size: "1GB"  
 work\_mem: "4MB"  
 maintenance\_work\_mem: "64MB"  
   
 bootstrap:  
 initdb:  
 database: app\_db  
 owner: app\_user  
 secret:  
 name: postgres-credentials  
   
 storage:  
 size: 100Gi  
 storageClass: fast-ssd  
   
 resources:  
 requests:  
 memory: "2Gi"  
 cpu: "1"  
 limits:  
 memory: "4Gi"  
 cpu: "2"  
   
 monitoring:  
 enabled: true  
 customQueriesConfigMap:  
 - name: postgres-monitoring  
 key: custom-queries.yaml  
   
 backup:  
 barmanObjectStore:  
 destinationPath: s3://db-backups/postgres  
 s3Credentials:  
 accessKeyId:  
 name: backup-credentials  
 key: ACCESS\_KEY\_ID  
 secretAccessKey:  
 name: backup-credentials  
 key: SECRET\_ACCESS\_KEY  
 wal:  
 retention: "7d"  
 data:  
 retention: "30d"

###### MySQL Service Template

apiVersion: mysql.oracle.com/v2  
kind: InnoDBCluster  
metadata:  
 name: mysql-template  
 namespace: database-services  
spec:  
 secretName: mysql-credentials  
 tlsUseSelfSigned: true  
 instances: 3  
 router:  
 instances: 2  
   
 datadirVolumeClaimTemplate:  
 accessModes:  
 - ReadWriteOnce  
 resources:  
 requests:  
 storage: 100Gi  
 storageClassName: fast-ssd  
   
 mycnf: |  
 [mysqld]  
 max\_connections = 500  
 innodb\_buffer\_pool\_size = 2G  
 innodb\_log\_file\_size = 256M  
 binlog\_expire\_logs\_seconds = 604800  
   
 resources:  
 requests:  
 memory: "2Gi"  
 cpu: "1"  
 limits:  
 memory: "4Gi"  
 cpu: "2"  
   
 backupProfiles:  
 - name: daily-backup  
 dumpInstance:  
 dumpOptions:  
 users: true  
 excludeSchemas: ["information\_schema", "performance\_schema"]  
 schedule: "0 2 \* \* \*"  
 backupRetentionDays: 30

##### Step 3: Implement Self-Service Provisioning

###### Service Catalog Integration

apiVersion: servicecatalog.k8s.io/v1beta1  
kind: ServiceClass  
metadata:  
 name: postgresql-service  
spec:  
 clusterServiceBrokerName: database-service-broker  
 externalName: postgresql  
 description: "Managed PostgreSQL Database Service"  
 externalMetadata:  
 displayName: "PostgreSQL Database"  
 imageUrl: "https://example.com/postgresql-icon.png"  
 longDescription: "High-availability PostgreSQL database with automated backups and monitoring"  
 providerDisplayName: "Database Team"  
 supportUrl: "https://example.com/support"  
 plans:  
 - name: small  
 externalID: postgres-small  
 description: "Small PostgreSQL instance (2 CPU, 4GB RAM, 100GB storage)"  
 free: false  
 externalMetadata:  
 displayName: "Small"  
 bullets:  
 - "2 CPU cores"  
 - "4GB RAM"  
 - "100GB storage"  
 - "Daily backups"  
 - name: medium  
 externalID: postgres-medium  
 description: "Medium PostgreSQL instance (4 CPU, 8GB RAM, 500GB storage)"  
 free: false  
 externalMetadata:  
 displayName: "Medium"  
 bullets:  
 - "4 CPU cores"  
 - "8GB RAM"  
 - "500GB storage"  
 - "Daily backups"  
 - "Read replicas"

###### Database Provisioning Operator

apiVersion: apps/v1  
kind: Deployment  
metadata:  
 name: database-provisioning-operator  
 namespace: database-operators  
spec:  
 replicas: 1  
 selector:  
 matchLabels:  
 app: database-provisioning-operator  
 template:  
 metadata:  
 labels:  
 app: database-provisioning-operator  
 spec:  
 containers:  
 - name: operator  
 image: quay.io/example/database-provisioning-operator:latest  
 env:  
 - name: WATCH\_NAMESPACE  
 value: ""  
 - name: POD\_NAME  
 valueFrom:  
 fieldRef:  
 fieldPath: metadata.name  
 - name: OPERATOR\_NAME  
 value: "database-provisioning-operator"  
 ports:  
 - containerPort: 8080  
 name: metrics  
 - containerPort: 8081  
 name: health  
 livenessProbe:  
 httpGet:  
 path: /healthz  
 port: 8081  
 initialDelaySeconds: 15  
 periodSeconds: 20  
 readinessProbe:  
 httpGet:  
 path: /readyz  
 port: 8081  
 initialDelaySeconds: 5  
 periodSeconds: 10  
 resources:  
 limits:  
 cpu: 200m  
 memory: 256Mi  
 requests:  
 cpu: 100m  
 memory: 128Mi

##### Step 4: Configure Automated Backup and Recovery

###### Backup Schedule Configuration

apiVersion: batch/v1  
kind: CronJob  
metadata:  
 name: database-backup-scheduler  
 namespace: database-services  
spec:  
 schedule: "0 2 \* \* \*" # Daily at 2 AM  
 jobTemplate:  
 spec:  
 template:  
 spec:  
 containers:  
 - name: backup-job  
 image: quay.io/example/database-backup:latest  
 env:  
 - name: BACKUP\_TYPE  
 value: "full"  
 - name: RETENTION\_DAYS  
 value: "30"  
 - name: S3\_BUCKET  
 value: "database-backups"  
 - name: AWS\_ACCESS\_KEY\_ID  
 valueFrom:  
 secretKeyRef:  
 name: backup-credentials  
 key: access-key-id  
 - name: AWS\_SECRET\_ACCESS\_KEY  
 valueFrom:  
 secretKeyRef:  
 name: backup-credentials  
 key: secret-access-key  
 command:  
 - /bin/bash  
 - -c  
 - |  
 #!/bin/bash  
 set -e  
   
 # Discover all database instances  
 for db in $(kubectl get postgresql,mysql,mongodb -o name --all-namespaces); do  
 echo "Backing up $db"  
 backup-database.sh "$db"  
 done  
 restartPolicy: OnFailure

##### Step 5: Implement Monitoring and Alerting

###### Database Monitoring Configuration

apiVersion: monitoring.coreos.com/v1  
kind: ServiceMonitor  
metadata:  
 name: database-services-monitor  
 namespace: database-services  
 labels:  
 monitoring: database-services  
spec:  
 selector:  
 matchLabels:  
 monitoring: enabled  
 endpoints:  
 - port: metrics  
 interval: 30s  
 path: /metrics  
 honorLabels: true  
---  
apiVersion: monitoring.coreos.com/v1  
kind: PrometheusRule  
metadata:  
 name: database-services-alerts  
 namespace: database-services  
spec:  
 groups:  
 - name: database.alerts  
 rules:  
 - alert: DatabaseInstanceDown  
 expr: up{job="database-services"} == 0  
 for: 5m  
 labels:  
 severity: critical  
 annotations:  
 summary: "Database instance {{ $labels.instance }} is down"  
 description: "Database instance {{ $labels.instance }} has been down for more than 5 minutes."  
   
 - alert: DatabaseHighConnections  
 expr: database\_connections\_active / database\_connections\_max > 0.8  
 for: 10m  
 labels:  
 severity: warning  
 annotations:  
 summary: "Database {{ $labels.instance }} has high connection usage"  
 description: "Database {{ $labels.instance }} is using {{ $value | humanizePercentage }} of available connections."  
   
 - alert: DatabaseSlowQueries  
 expr: rate(database\_slow\_queries\_total[5m]) > 10  
 for: 5m  
 labels:  
 severity: warning  
 annotations:  
 summary: "Database {{ $labels.instance }} has high slow query rate"  
 description: "Database {{ $labels.instance }} has {{ $value }} slow queries per second."  
   
 - alert: DatabaseBackupFailed  
 expr: database\_backup\_last\_success\_timestamp < (time() - 86400)  
 for: 1m  
 labels:  
 severity: critical  
 annotations:  
 summary: "Database backup failed for {{ $labels.instance }}"  
 description: "Database backup for {{ $labels.instance }} has not succeeded in the last 24 hours."

##### Step 6: Implement Database Security

###### Network Policies for Database Isolation

apiVersion: networking.k8s.io/v1  
kind: NetworkPolicy  
metadata:  
 name: database-isolation  
 namespace: database-services  
spec:  
 podSelector:  
 matchLabels:  
 tier: database  
 policyTypes:  
 - Ingress  
 - Egress  
 ingress:  
 - from:  
 - namespaceSelector:  
 matchLabels:  
 name: application-services  
 - podSelector:  
 matchLabels:  
 tier: application  
 ports:  
 - protocol: TCP  
 port: 5432 # PostgreSQL  
 - protocol: TCP  
 port: 3306 # MySQL  
 - protocol: TCP  
 port: 27017 # MongoDB  
 - protocol: TCP  
 port: 6379 # Redis  
 egress:  
 - to: []  
 ports:  
 - protocol: TCP  
 port: 53 # DNS  
 - protocol: UDP  
 port: 53 # DNS  
 - to:  
 - namespaceSelector:  
 matchLabels:  
 name: backup-services  
 ports:  
 - protocol: TCP  
 port: 443 # HTTPS for backup storage

###### Database Secret Management

apiVersion: external-secrets.io/v1beta1  
kind: SecretStore  
metadata:  
 name: database-secret-store  
 namespace: database-services  
spec:  
 provider:  
 vault:  
 server: "https://vault.example.com"  
 path: "secret"  
 version: "v2"  
 auth:  
 kubernetes:  
 mountPath: "kubernetes"  
 role: "database-secrets"  
---  
apiVersion: external-secrets.io/v1beta1  
kind: ExternalSecret  
metadata:  
 name: postgres-credentials  
 namespace: database-services  
spec:  
 refreshInterval: 1h  
 secretStoreRef:  
 name: database-secret-store  
 kind: SecretStore  
 target:  
 name: postgres-credentials  
 creationPolicy: Owner  
 template:  
 type: Opaque  
 data:  
 username: "{{ .username }}"  
 password: "{{ .password }}"  
 database: "{{ .database }}"  
 data:  
 - secretKey: username  
 remoteRef:  
 key: database/postgres  
 property: username  
 - secretKey: password  
 remoteRef:  
 key: database/postgres  
 property: password  
 - secretKey: database  
 remoteRef:  
 key: database/postgres  
 property: database

#### Application Integration Examples

##### Spring Boot Application Configuration

apiVersion: apps/v1  
kind: Deployment  
metadata:  
 name: spring-app  
 namespace: application-services  
spec:  
 replicas: 3  
 selector:  
 matchLabels:  
 app: spring-app  
 template:  
 metadata:  
 labels:  
 app: spring-app  
 tier: application  
 spec:  
 containers:  
 - name: app  
 image: quay.io/example/spring-app:latest  
 env:  
 - name: SPRING\_DATASOURCE\_URL  
 value: "jdbc:postgresql://postgres-service.database-services.svc.cluster.local:5432/app\_db"  
 - name: SPRING\_DATASOURCE\_USERNAME  
 valueFrom:  
 secretKeyRef:  
 name: app-db-credentials  
 key: username  
 - name: SPRING\_DATASOURCE\_PASSWORD  
 valueFrom:  
 secretKeyRef:  
 name: app-db-credentials  
 key: password  
 - name: SPRING\_JPA\_HIBERNATE\_DDL\_AUTO  
 value: "validate"  
 ports:  
 - containerPort: 8080  
 livenessProbe:  
 httpGet:  
 path: /actuator/health  
 port: 8080  
 initialDelaySeconds: 60  
 periodSeconds: 30  
 readinessProbe:  
 httpGet:  
 path: /actuator/health/readiness  
 port: 8080  
 initialDelaySeconds: 30  
 periodSeconds: 10

#### Troubleshooting Guide

##### Common Issues and Solutions

###### Database Connection Issues

* **Issue**: Applications cannot connect to database services
* **Solution**:
  + Verify network policies allow traffic between application and database namespaces
  + Check service discovery and DNS resolution
  + Validate database credentials and authentication

###### Performance Issues

* **Issue**: Slow database queries and high latency
* **Solution**:
  + Review database configuration parameters
  + Analyze slow query logs and optimize indexes
  + Scale database resources (CPU, memory, storage IOPS)
  + Implement connection pooling

###### Backup and Recovery Failures

* **Issue**: Database backups failing or restoration issues
* **Solution**:
  + Verify backup storage credentials and permissions
  + Check backup retention policies and storage quotas
  + Test backup restoration procedures regularly
  + Monitor backup job logs for errors

###### Resource Constraints

* **Issue**: Database instances running out of resources
* **Solution**:
  + Implement resource monitoring and alerting
  + Configure horizontal and vertical scaling policies
  + Optimize database configuration for workload patterns
  + Consider database sharding for large datasets

#### Best Practices

##### Database Design and Configuration

* **Resource Planning**: Size database instances based on expected workload patterns
* **Configuration Tuning**: Optimize database parameters for specific use cases
* **Connection Management**: Implement connection pooling and limit concurrent connections
* **Index Strategy**: Create appropriate indexes for query performance

##### Security and Compliance

* **Encryption**: Enable encryption at rest and in transit for all database communications
* **Access Control**: Implement least-privilege access using RBAC and network policies
* **Audit Logging**: Enable database audit logging for compliance requirements
* **Secret Rotation**: Implement automated secret rotation for database credentials

##### Operations and Maintenance

* **Monitoring**: Implement comprehensive monitoring for performance, availability, and capacity
* **Backup Strategy**: Implement automated backups with tested restoration procedures
* **Disaster Recovery**: Plan and test disaster recovery procedures regularly
* **Maintenance Windows**: Schedule regular maintenance windows for updates and optimizations

#### Integration with RH OVE Ecosystem

##### GitOps Integration

apiVersion: argoproj.io/v1alpha1  
kind: Application  
metadata:  
 name: database-services  
 namespace: argocd  
spec:  
 project: platform-services  
 source:  
 repoURL: https://git.example.com/database-services-config  
 targetRevision: HEAD  
 path: overlays/production  
 destination:  
 server: https://kubernetes.default.svc  
 namespace: database-services  
 syncPolicy:  
 automated:  
 prune: false  
 selfHeal: true  
 syncOptions:  
 - CreateNamespace=true

##### Multi-Cluster Database Federation

* **Cross-Cluster Replication**: Implement database replication across multiple clusters
* **Global Load Balancing**: Use global load balancers for database read replicas
* **Disaster Recovery**: Maintain database replicas in different geographical regions

This comprehensive guide provides everything needed to implement Database Services as PaaS within the RH OVE ecosystem, enabling self-service database provisioning, automated management, and seamless integration with application workloads while maintaining enterprise-grade security, performance, and reliability.

## Legacy Modernization

### Use Case: Legacy Application Modernization

#### Business Context

Legacy application modernization is critical for organizations looking to leverage modern cloud-native technologies while preserving existing business logic and data. This use case demonstrates a phased approach to modernizing legacy applications using the RH OVE ecosystem, enabling gradual transformation with minimal business disruption.

#### Technical Requirements

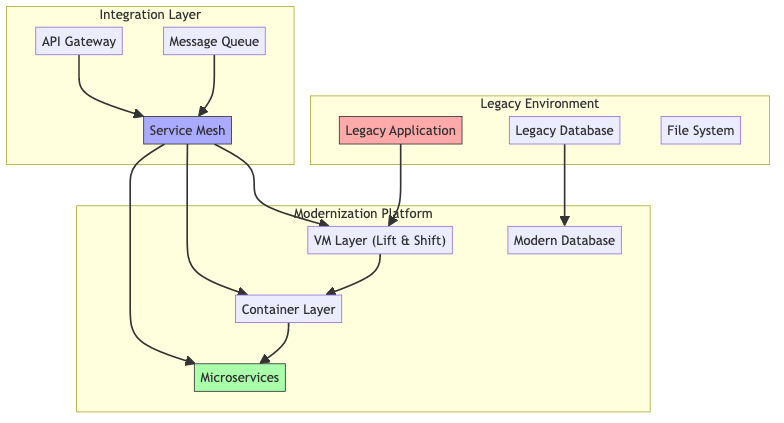
##### Infrastructure Requirements

* OpenShift 4.12+ with KubeVirt enabled
* Application migration tools (Migration Toolkit for Applications - MTA)
* Service mesh for hybrid connectivity (OpenShift Service Mesh/Istio)
* CI/CD pipelines (OpenShift Pipelines/Tekton)
* Container registry (Quay.io or OpenShift integrated registry)

##### Resource Requirements

* **Compute**: Sufficient resources for both legacy and modernized components during transition
* **Storage**: Persistent storage for data migration and synchronization
* **Network**: High-bandwidth connectivity for data replication and service communication

#### Architecture Overview



#### Implementation Steps

##### Phase 1: Assessment and Planning

###### Application Discovery

### MTA Configuration for Application Analysis  
apiVersion: tackle.konveyor.io/v1alpha1  
kind: Application  
metadata:  
 name: legacy-app-analysis  
 namespace: konveyor-tackle  
spec:  
 name: "Legacy ERP System"  
 description: "Monolithic ERP application requiring modernization"  
 repository:  
 kind: git  
 url: "https://git.example.com/legacy-erp"  
 binary: "erp-application.war"

###### Migration Assessment

### Run application analysis using MTA CLI  
konveyor-cli analyze \  
 --input /path/to/legacy-app \  
 --output /path/to/analysis-results \  
 --target cloud-readiness \  
 --target containers

##### Phase 2: Lift and Shift (VM Migration)

###### VM-based Legacy Application Deployment

apiVersion: kubevirt.io/v1  
kind: VirtualMachine  
metadata:  
 name: legacy-erp-vm  
 namespace: modernization  
spec:  
 running: true  
 template:  
 metadata:  
 labels:  
 app: legacy-erp  
 tier: application  
 phase: lift-shift  
 spec:  
 domain:  
 cpu:  
 cores: 4  
 devices:  
 disks:  
 - disk:  
 bus: virtio  
 name: rootdisk  
 - disk:  
 bus: virtio  
 name: datadisk  
 interfaces:  
 - name: default  
 bridge: {}  
 memory:  
 guest: 8Gi  
 resources:  
 requests:  
 memory: 8Gi  
 cpu: 4  
 networks:  
 - name: default  
 pod: {}  
 volumes:  
 - dataVolume:  
 name: legacy-erp-root  
 name: rootdisk  
 - dataVolume:  
 name: legacy-erp-data  
 name: datadisk  
---  
apiVersion: cdi.kubevirt.io/v1beta1  
kind: DataVolume  
metadata:  
 name: legacy-erp-root  
 namespace: modernization  
spec:  
 source:  
 http:  
 url: "https://vm-images.example.com/legacy-erp-root.img"  
 pvc:  
 accessModes:  
 - ReadWriteOnce  
 resources:  
 requests:  
 storage: 100Gi  
---  
apiVersion: cdi.kubevirt.io/v1beta1  
kind: DataVolume  
metadata:  
 name: legacy-erp-data  
 namespace: modernization  
spec:  
 source:  
 http:  
 url: "https://vm-images.example.com/legacy-erp-data.img"  
 pvc:  
 accessModes:  
 - ReadWriteOnce  
 resources:  
 requests:  
 storage: 500Gi

##### Phase 3: Containerization

###### Legacy Application Container Deployment

apiVersion: apps/v1  
kind: Deployment  
metadata:  
 name: legacy-erp-container  
 namespace: modernization  
spec:  
 replicas: 2  
 selector:  
 matchLabels:  
 app: legacy-erp  
 tier: application  
 phase: containerized  
 template:  
 metadata:  
 labels:  
 app: legacy-erp  
 tier: application  
 phase: containerized  
 spec:  
 containers:  
 - name: erp-app  
 image: quay.io/example/legacy-erp:containerized  
 ports:  
 - containerPort: 8080  
 name: http  
 - containerPort: 8443  
 name: https  
 env:  
 - name: DB\_HOST  
 value: "legacy-database-service"  
 - name: DB\_PORT  
 value: "5432"  
 - name: DB\_NAME  
 valueFrom:  
 secretKeyRef:  
 name: db-credentials  
 key: database  
 - name: DB\_USER  
 valueFrom:  
 secretKeyRef:  
 name: db-credentials  
 key: username  
 - name: DB\_PASSWORD  
 valueFrom:  
 secretKeyRef:  
 name: db-credentials  
 key: password  
 volumeMounts:  
 - name: app-data  
 mountPath: /opt/erp/data  
 - name: app-config  
 mountPath: /opt/erp/config  
 volumes:  
 - name: app-data  
 persistentVolumeClaim:  
 claimName: erp-data-pvc  
 - name: app-config  
 configMap:  
 name: erp-config

##### Phase 4: Service Decomposition

###### Extract User Management Service

apiVersion: apps/v1  
kind: Deployment  
metadata:  
 name: user-management-service  
 namespace: modernization  
spec:  
 replicas: 3  
 selector:  
 matchLabels:  
 app: user-management  
 tier: microservice  
 template:  
 metadata:  
 labels:  
 app: user-management  
 tier: microservice  
 version: v1  
 spec:  
 containers:  
 - name: user-management  
 image: quay.io/example/user-management:v1.0.0  
 ports:  
 - containerPort: 8080  
 name: http  
 env:  
 - name: DATABASE\_URL  
 valueFrom:  
 secretKeyRef:  
 name: user-db-credentials  
 key: url  
 - name: REDIS\_URL  
 valueFrom:  
 secretKeyRef:  
 name: redis-credentials  
 key: url  
 livenessProbe:  
 httpGet:  
 path: /health  
 port: 8080  
 initialDelaySeconds: 30  
 periodSeconds: 10  
 readinessProbe:  
 httpGet:  
 path: /ready  
 port: 8080  
 initialDelaySeconds: 5  
 periodSeconds: 5  
---  
apiVersion: v1  
kind: Service  
metadata:  
 name: user-management-service  
 namespace: modernization  
spec:  
 selector:  
 app: user-management  
 ports:  
 - name: http  
 port: 80  
 targetPort: 8080  
 type: ClusterIP

##### Phase 5: Service Mesh Integration

###### Istio Service Mesh Configuration

apiVersion: networking.istio.io/v1beta1  
kind: VirtualService  
metadata:  
 name: legacy-erp-routing  
 namespace: modernization  
spec:  
 hosts:  
 - erp.example.com  
 http:  
 - match:  
 - uri:  
 prefix: /api/users  
 route:  
 - destination:  
 host: user-management-service  
 port:  
 number: 80  
 weight: 100  
 - match:  
 - uri:  
 prefix: /  
 route:  
 - destination:  
 host: legacy-erp-container  
 port:  
 number: 8080  
 weight: 90  
 - destination:  
 host: legacy-erp-vm  
 port:  
 number: 8080  
 weight: 10  
---  
apiVersion: networking.istio.io/v1beta1  
kind: DestinationRule  
metadata:  
 name: legacy-erp-destination  
 namespace: modernization  
spec:  
 host: legacy-erp-container  
 trafficPolicy:  
 circuitBreaker:  
 consecutive5xxErrors: 3  
 interval: 30s  
 baseEjectionTime: 30s

##### Phase 6: Data Migration Strategy

###### Database Migration Pipeline

apiVersion: tekton.dev/v1beta1  
kind: Pipeline  
metadata:  
 name: data-migration-pipeline  
 namespace: modernization  
spec:  
 params:  
 - name: source-db-url  
 description: Source database connection URL  
 - name: target-db-url  
 description: Target database connection URL  
 - name: migration-batch-size  
 default: "1000"  
 description: Number of records to migrate per batch  
   
 tasks:  
 - name: validate-source  
 taskRef:  
 name: database-validation  
 params:  
 - name: db-url  
 value: $(params.source-db-url)  
   
 - name: create-target-schema  
 taskRef:  
 name: schema-creation  
 params:  
 - name: db-url  
 value: $(params.target-db-url)  
 runAfter:  
 - validate-source  
   
 - name: migrate-data  
 taskRef:  
 name: data-migration  
 params:  
 - name: source-db-url  
 value: $(params.source-db-url)  
 - name: target-db-url  
 value: $(params.target-db-url)  
 - name: batch-size  
 value: $(params.migration-batch-size)  
 runAfter:  
 - create-target-schema  
   
 - name: validate-migration  
 taskRef:  
 name: migration-validation  
 params:  
 - name: source-db-url  
 value: $(params.source-db-url)  
 - name: target-db-url  
 value: $(params.target-db-url)  
 runAfter:  
 - migrate-data

##### Phase 7: Progressive Traffic Migration

###### Canary Deployment Strategy

apiVersion: argoproj.io/v1alpha1  
kind: Rollout  
metadata:  
 name: legacy-erp-rollout  
 namespace: modernization  
spec:  
 replicas: 5  
 strategy:  
 canary:  
 steps:  
 - setWeight: 10  
 - pause: {duration: 300s}  
 - setWeight: 25  
 - pause: {duration: 300s}  
 - setWeight: 50  
 - pause: {duration: 300s}  
 - setWeight: 75  
 - pause: {duration: 300s}  
 canaryService: legacy-erp-canary  
 stableService: legacy-erp-stable  
 trafficRouting:  
 istio:  
 virtualService:  
 name: legacy-erp-routing  
 routes:  
 - primary  
 selector:  
 matchLabels:  
 app: legacy-erp  
 template:  
 metadata:  
 labels:  
 app: legacy-erp  
 spec:  
 containers:  
 - name: erp-app  
 image: quay.io/example/legacy-erp:modernized  
 ports:  
 - containerPort: 8080

#### Monitoring and Observability

##### Application Performance Monitoring

apiVersion: monitoring.coreos.com/v1  
kind: ServiceMonitor  
metadata:  
 name: modernization-monitoring  
 namespace: modernization  
spec:  
 selector:  
 matchLabels:  
 monitoring: enabled  
 endpoints:  
 - port: metrics  
 interval: 30s  
 path: /metrics  
---  
apiVersion: monitoring.coreos.com/v1  
kind: PrometheusRule  
metadata:  
 name: modernization-alerts  
 namespace: modernization  
spec:  
 groups:  
 - name: modernization.alerts  
 rules:  
 - alert: LegacyAppHighErrorRate  
 expr: rate(http\_requests\_total{status=~"5.."}[5m]) > 0.1  
 for: 5m  
 labels:  
 severity: warning  
 annotations:  
 summary: "High error rate in legacy application"  
   
 - alert: MigrationDataInconsistency  
 expr: migration\_data\_consistency\_check != 1  
 for: 1m  
 labels:  
 severity: critical  
 annotations:  
 summary: "Data inconsistency detected during migration"

#### Troubleshooting Guide

##### Common Issues and Solutions

###### Performance Degradation During Migration

* **Issue**: Application performance degrades during data migration
* **Solution**:
  + Implement read replicas for database queries
  + Use incremental migration strategies
  + Schedule heavy migration tasks during off-peak hours

###### Service Communication Failures

* **Issue**: Communication failures between legacy and modern components
* **Solution**:
  + Verify service mesh configuration
  + Check network policies and firewall rules
  + Implement circuit breakers and retry mechanisms

###### Data Consistency Issues

* **Issue**: Data inconsistencies between old and new systems
* **Solution**:
  + Implement two-phase commit protocols
  + Use event sourcing for critical operations
  + Regular data validation and reconciliation

#### Best Practices

##### Migration Strategy

* **Incremental Approach**: Migrate functionality incrementally to reduce risk
* **Feature Toggles**: Use feature flags to control rollout of new functionality
* **Rollback Plans**: Always have rollback procedures for each migration phase

##### Testing and Validation

* **Automated Testing**: Implement comprehensive test suites for both legacy and modern components
* **Performance Testing**: Conduct load testing throughout the migration process
* **Data Validation**: Implement automated data consistency checks

##### Security Considerations

* **Zero-Trust Architecture**: Implement zero-trust principles across all components
* **Secrets Management**: Use proper secrets management for database credentials and API keys
* **Network Segmentation**: Implement proper network segmentation between components

#### Integration with RH OVE Ecosystem

##### GitOps Integration

apiVersion: argoproj.io/v1alpha1  
kind: Application  
metadata:  
 name: legacy-modernization  
 namespace: argocd  
spec:  
 project: modernization  
 source:  
 repoURL: https://git.example.com/legacy-modernization-config  
 targetRevision: HEAD  
 path: environments/production  
 destination:  
 server: https://kubernetes.default.svc  
 namespace: modernization  
 syncPolicy:  
 automated:  
 prune: false  
 selfHeal: true  
 syncOptions:  
 - CreateNamespace=true

##### Multi-Cluster Deployment

* **Staging Environment**: Use separate clusters for testing and validation
* **Production Rollout**: Implement blue-green deployment across clusters
* **Disaster Recovery**: Maintain legacy systems as backup during transition

This comprehensive guide provides a structured approach to legacy application modernization within the RH OVE ecosystem, enabling organizations to gradually transform their applications while maintaining business continuity and reducing risk.

## Disaster Recovery

### Use Case: Disaster Recovery

#### Business Context

Disaster recovery is a crucial aspect of business continuity, ensuring that workloads can be swiftly restored following catastrophic events. This use case outlines strategies and tools for implementing effective disaster recovery plans within the RH OVE ecosystem.

#### Technical Requirements

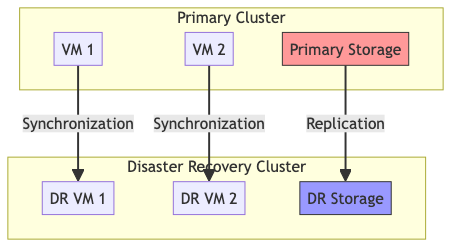
##### Infrastructure Requirements

* OpenShift 4.12+ clusters with multi-cluster management enabled
* Cross-cluster networking with VPN or direct connectivity
* Data replication and backup solutions
* Disaster recovery orchestration tools (Red Hat Advanced Cluster Management - RHACM)

##### Resource Requirements

* **Compute**: Sufficient capacity on recovery clusters
* **Storage**: Redundant storage solutions with replication
* **Network**: Reliable, high-speed connections between primary and secondary sites

#### Architecture Overview



#### Implementation Steps

##### Step 1: Plan and Prepare

###### Define Disaster Recovery Objectives

* Identify RTO (Recovery Time Objective) and RPO (Recovery Point Objective)

###### Inventory Assessment

* Document existing resources and dependencies

##### Step 2: Configure Data Replication

###### Persistent Storage Replication

* Configure synchronous or asynchronous replication between primary and DR sites.

apiVersion: apps/v1  
kind: StatefulSet  
metadata:  
 name: dr-replication-demo  
 namespace: storage-replication  
spec:  
 selector:  
 matchLabels:  
 app: replication  
 serviceName: "replication"  
 replicas: 2  
 template:  
 metadata:  
 labels:  
 app: replication  
 spec:  
 containers:  
 - name: replication-agent  
 image: replication-agent:latest  
 args:  
 - --source-pvc  
 - source-storage-pvc  
 - --target-pvc  
 - target-storage-pvc

##### Step 3: Implement Cross-Cluster Networking

###### VPN Configuration for Cluster Connectivity

* Set up VPN tunnels or configure direct connectivity between cluster sites.

##### Step 4: Deploy DR Orchestration Tools

###### RHACM Configuration

* Deploy Red Hat Advanced Cluster Management for cluster failover management.

apiVersion: cluster.open-cluster-management.io/v1  
kind: ManagedCluster  
metadata:  
 name: disaster-recovery-cluster  
spec:  
 hubAcceptsClient: true  
 managedClusterClientConfigs:  
 - url: https://api.dr-cluster.example.com:6443

##### Step 5: Automate Failover and Recovery

###### Failover Scripts and Automation

* Develop scripts to automate the failover process based on RHACM policies.

### !/bin/bash  
### Failover script for disaster recovery activation  
  
### Scale down primary workloads  
kubectl scale deployment --all --replicas=0 -n primary-workloads  
  
### Scale up DR workloads  
kubectl scale deployment --all --replicas=1 -n disaster-recovery-workloads  
  
### Update DNS settings  
update-dns --zone=example.com --record=\*.example.com --new-ip=dr-cluster-ip

##### Step 6: Testing and Validation

###### Disaster Recovery Drills

* Conduct regular DR drills to test and validate recovery procedures.

### Trigger disaster recovery drill  
run-drill --cluster=disaster-recovery-cluster --scenario=full-cluster-failure

#### Troubleshooting Guide

##### Common Issues and Solutions

###### Replication Lag

* **Issue**: Data replication falls behind
* **Solution**:
  + Increase network bandwidth
  + Optimize replication frequencies
  + Monitor replication service for bottlenecks

###### Failover Errors

* **Issue**: Failover task errors or delays
* **Solution**:
  + Verify failover scripts and automation procedures
  + Test DNS updates and propagation
  + Check cluster configuration consistency

###### Network Connectivity Issues

* **Issue**: VPN or network interruptions
* **Solution**:
  + Test alternate routes and consider multi-path routing
  + Verify firewall and security group configurations
  + Implement continuous network monitoring

#### Best Practices

##### Strategy and Planning

* **Comprehensive Planning**: Develop detailed DR plans aligned with business priorities
* **Periodic Reviews**: Regularly review DR strategies and update based on changes in infrastructure
* **Stakeholder Engagement**: Involve all relevant stakeholders in DR planning and testing

##### Technology and Tools

* **Automation**: Leverage automation for failover processes to minimize human error
* **Monitoring and Alerts**: Implement monitoring and alerting for quick detection of failures
* **Compliance and Auditing**: Ensure DR plans meet compliance and regulatory requirements

#### Integration with RH OVE Ecosystem

##### Multi-Cluster Management

* Use RHACM for managing multiple clusters, facilitating disaster recovery coordination

##### Environmental Parity

* Ensure consistency in configurations between primary and secondary environments

This guide provides the steps and best practices necessary to establish robust disaster recovery systems within the RH OVE ecosystem, ensuring business continuity and data availability even in the event of a major failure or disaster.

## End To End Observability

### Use Case: End-to-End Application Observability in RH OVE

#### Business Context

In the Red Hat OpenShift Virtualization Engine (RH OVE) ecosystem, comprehensive observability is essential for monitoring both containerized applications and virtual machines, understanding performance bottlenecks, troubleshooting issues, and ensuring optimal resource utilization across hybrid workloads. This use case demonstrates two complementary approaches: native OpenShift observability tools and integration with Dynatrace for enterprise-grade observability.

#### What Developers Need to Expose

For effective end-to-end observability, developers must instrument their applications to expose:

##### Required Metrics

* **Business Metrics**: Transaction counts, success rates, revenue metrics
* **Application Metrics**: Response times, error rates, throughput
* **Resource Metrics**: CPU, memory, disk I/O, network usage
* **Custom Metrics**: Domain-specific KPIs and performance indicators

##### Required Traces

* **Request Traces**: End-to-end request flow across microservices
* **Database Traces**: SQL queries and database connection metrics
* **External Service Traces**: API calls to third-party services
* **Async Operations**: Message queue operations, background jobs

##### Required Logs

* **Structured Logs**: JSON formatted with consistent fields
* **Error Logs**: Exception details with stack traces
* **Audit Logs**: Security and compliance events
* **Performance Logs**: Slow queries, long-running operations

##### Health Endpoints

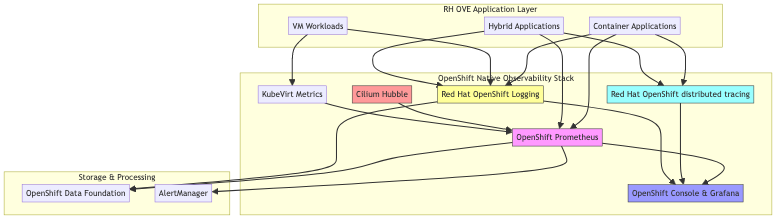
* **Liveness Probes**: /health/live - Application is running
* **Readiness Probes**: /health/ready - Application ready to serve traffic
* **Metrics Endpoint**: /metrics - Prometheus-formatted metrics
* **Info Endpoint**: /info - Application version and build information

#### 1. Native OpenShift Observability

##### Infrastructure Requirements

* OpenShift 4.12+ with built-in monitoring stack
* OpenShift Data Foundation for persistent storage
* Red Hat OpenShift Logging (based on Loki)
* Red Hat OpenShift distributed tracing (Jaeger)
* Cilium Hubble for network observability
* KubeVirt monitoring for VM workloads

##### Architecture Overview



##### Implementation Steps

##### Step 1: Enable OpenShift Built-in Monitoring

###### Configure User Workload Monitoring

apiVersion: v1  
kind: ConfigMap  
metadata:  
 name: cluster-monitoring-config  
 namespace: openshift-monitoring  
data:  
 config.yaml: |  
 enableUserWorkloadMonitoring: true  
 prometheusK8s:  
 retention: 30d  
 volumeClaimTemplate:  
 spec:  
 storageClassName: ocs-storagecluster-ceph-rbd  
 resources:  
 requests:  
 storage: 100Gi

##### Step 2: Application Instrumentation for Container Applications

###### Comprehensive Metrics Configuration (Go Example)

package main  
  
import (  
 "context"  
 "net/http"  
 "time"  
   
 "github.com/prometheus/client\_golang/prometheus"  
 "github.com/prometheus/client\_golang/prometheus/promauto"  
 "github.com/prometheus/client\_golang/prometheus/promhttp"  
)  
  
var (  
 // Business metrics  
 httpRequestsTotal = promauto.NewCounterVec(  
 prometheus.CounterOpts{  
 Name: "myapp\_http\_requests\_total",  
 Help: "Total number of HTTP requests by status code and method",  
 },  
 []string{"method", "status\_code", "endpoint"},  
 )  
   
 // Performance metrics  
 httpRequestDuration = promauto.NewHistogramVec(  
 prometheus.HistogramOpts{  
 Name: "myapp\_http\_request\_duration\_seconds",  
 Help: "HTTP request duration in seconds",  
 Buckets: prometheus.DefBuckets,  
 },  
 []string{"method", "endpoint"},  
 )  
   
 // Resource metrics  
 activeConnections = promauto.NewGauge(  
 prometheus.GaugeOpts{  
 Name: "myapp\_active\_connections",  
 Help: "Number of active connections",  
 },  
 )  
   
 // Custom business metrics  
 ordersProcessed = promauto.NewCounterVec(  
 prometheus.CounterOpts{  
 Name: "myapp\_orders\_processed\_total",  
 Help: "Total number of orders processed",  
 },  
 []string{"status"},  
 )  
)  
  
func instrumentHandler(next http.HandlerFunc, endpoint string) http.HandlerFunc {  
 return http.HandlerFunc(func(w http.ResponseWriter, r \*http.Request) {  
 start := time.Now()  
   
 // Process request  
 next.ServeHTTP(w, r)  
   
 // Record metrics  
 duration := time.Since(start).Seconds()  
 httpRequestDuration.WithLabelValues(r.Method, endpoint).Observe(duration)  
 httpRequestsTotal.WithLabelValues(r.Method, "200", endpoint).Inc()  
 })  
}  
  
func main() {  
 // Health endpoints  
 http.HandleFunc("/health/live", func(w http.ResponseWriter, r \*http.Request) {  
 w.WriteHeader(http.StatusOK)  
 w.Write([]byte("alive"))  
 })  
   
 http.HandleFunc("/health/ready", func(w http.ResponseWriter, r \*http.Request) {  
 // Check dependencies (DB, external services)  
 w.WriteHeader(http.StatusOK)  
 w.Write([]byte("ready"))  
 })  
   
 // Metrics endpoint  
 http.Handle("/metrics", promhttp.Handler())  
   
 // Business endpoints with instrumentation  
 http.HandleFunc("/api/orders", instrumentHandler(ordersHandler, "/api/orders"))  
   
 http.ListenAndServe(":8080", nil)  
}

###### Distributed Tracing Configuration (Node.js Example)

const { NodeSDK } = require('@opentelemetry/auto-instrumentations-node');  
const { JaegerExporter } = require('@opentelemetry/exporter-jaeger');  
const { Resource } = require('@opentelemetry/resources');  
const { SemanticResourceAttributes } = require('@opentelemetry/semantic-conventions');  
  
// Configure Jaeger exporter for OpenShift distributed tracing  
const jaegerExporter = new JaegerExporter({  
 endpoint: 'http://jaeger-collector.openshift-distributed-tracing-system.svc.cluster.local:14268/api/traces',  
});  
  
// Initialize OpenTelemetry SDK  
const sdk = new NodeSDK({  
 resource: new Resource({  
 [SemanticResourceAttributes.SERVICE\_NAME]: 'myapp-service',  
 [SemanticResourceAttributes.SERVICE\_VERSION]: process.env.APP\_VERSION || '1.0.0',  
 [SemanticResourceAttributes.DEPLOYMENT\_ENVIRONMENT]: process.env.NODE\_ENV || 'development',  
 }),  
 traceExporter: jaegerExporter,  
});  
  
sdk.start();  
  
const express = require('express');  
const { trace, context } = require('@opentelemetry/api');  
const app = express();  
  
// Custom tracing for business operations  
app.get('/api/orders/:id', async (req, res) => {  
 const tracer = trace.getTracer('myapp');  
   
 await tracer.startActiveSpan('process\_order', async (span) => {  
 try {  
 // Add custom attributes  
 span.setAttributes({  
 'order.id': req.params.id,  
 'user.id': req.headers['user-id'],  
 'operation.type': 'order\_processing'  
 });  
   
 // Simulate database call with tracing  
 await tracer.startActiveSpan('database\_query', async (dbSpan) => {  
 // Database operation  
 dbSpan.setAttributes({  
 'db.operation': 'SELECT',  
 'db.table': 'orders'  
 });  
 dbSpan.end();  
 });  
   
 // Simulate external API call  
 await tracer.startActiveSpan('external\_api\_call', async (apiSpan) => {  
 apiSpan.setAttributes({  
 'http.method': 'POST',  
 'http.url': 'https://payment-service/process'  
 });  
 apiSpan.end();  
 });  
   
 res.json({ orderId: req.params.id, status: 'processed' });  
 } catch (error) {  
 span.recordException(error);  
 span.setStatus({ code: trace.SpanStatusCode.ERROR, message: error.message });  
 res.status(500).json({ error: 'Processing failed' });  
 } finally {  
 span.end();  
 }  
 });  
});

###### Structured Logging Configuration

### Python example with structured logging for OpenShift Logging  
import logging  
import json  
import sys  
from datetime import datetime  
  
class StructuredLogger:  
 def \_\_init\_\_(self, service\_name):  
 self.service\_name = service\_name  
 self.logger = logging.getLogger(service\_name)  
 self.logger.setLevel(logging.INFO)  
   
 # Configure JSON formatter for OpenShift Logging  
 handler = logging.StreamHandler(sys.stdout)  
 handler.setFormatter(self.JsonFormatter())  
 self.logger.addHandler(handler)  
   
 class JsonFormatter(logging.Formatter):  
 def format(self, record):  
 log\_entry = {  
 'timestamp': datetime.utcnow().isoformat() + 'Z',  
 'level': record.levelname,  
 'service': record.name,  
 'message': record.getMessage(),  
 }  
   
 # Add custom fields if present  
 if hasattr(record, 'user\_id'):  
 log\_entry['user\_id'] = record.user\_id  
 if hasattr(record, 'trace\_id'):  
 log\_entry['trace\_id'] = record.trace\_id  
 if hasattr(record, 'span\_id'):  
 log\_entry['span\_id'] = record.span\_id  
   
 return json.dumps(log\_entry)  
   
 def info(self, message, \*\*kwargs):  
 extra = {k: v for k, v in kwargs.items()}  
 self.logger.info(message, extra=extra)  
   
 def error(self, message, \*\*kwargs):  
 extra = {k: v for k, v in kwargs.items()}  
 self.logger.error(message, extra=extra)  
  
### Usage in application  
logger = StructuredLogger('myapp-service')  
  
def process\_order(order\_id, user\_id):  
 logger.info(  
 "Processing order",  
 user\_id=user\_id,  
 order\_id=order\_id,  
 operation='order\_processing'  
 )  
   
 try:  
 # Business logic  
 result = do\_business\_logic()  
 logger.info(  
 "Order processed successfully",  
 user\_id=user\_id,  
 order\_id=order\_id,  
 result=result  
 )  
 except Exception as e:  
 logger.error(  
 "Order processing failed",  
 user\_id=user\_id,  
 order\_id=order\_id,  
 error=str(e),  
 stack\_trace=traceback.format\_exc()  
 )  
 raise

##### Step 3: Configure OpenShift Native Observability Components

###### Enable Red Hat OpenShift Logging

apiVersion: logging.coreos.com/v1  
kind: ClusterLogging  
metadata:  
 name: instance  
 namespace: openshift-logging  
spec:  
 managementState: Managed  
 logStore:  
 type: lokistack  
 lokistack:  
 name: logging-loki  
 collection:  
 type: vector  
 vector:  
 resources:  
 limits:  
 memory: 1Gi  
 requests:  
 memory: 512Mi  
 visualization:  
 type: ocp-console  
---  
apiVersion: loki.grafana.com/v1  
kind: LokiStack  
metadata:  
 name: logging-loki  
 namespace: openshift-logging  
spec:  
 size: 1x.small  
 storage:  
 schemas:  
 - version: v12  
 effectiveDate: '2022-06-01'  
 secret:  
 name: logging-loki-s3  
 type: s3  
 storageClassName: ocs-storagecluster-ceph-rbd  
 tenants:  
 mode: openshift-logging

###### Deploy Red Hat OpenShift distributed tracing

apiVersion: jaegertracing.io/v1  
kind: Jaeger  
metadata:  
 name: jaeger-production  
 namespace: openshift-distributed-tracing-system  
spec:  
 strategy: production  
 storage:  
 type: elasticsearch  
 elasticsearch:  
 nodeCount: 3  
 storage:  
 storageClassName: ocs-storagecluster-ceph-rbd  
 size: 100Gi  
 resources:  
 requests:  
 memory: 4Gi  
 cpu: 1  
 limits:  
 memory: 4Gi  
 cpu: 1

##### Step 4: Configure Application Monitoring

###### ServiceMonitor for Application Metrics

apiVersion: monitoring.coreos.com/v1  
kind: ServiceMonitor  
metadata:  
 name: myapp-metrics  
 namespace: myapp-namespace  
 labels:  
 app: myapp  
spec:  
 selector:  
 matchLabels:  
 app: myapp  
 endpoints:  
 - port: metrics  
 interval: 30s  
 path: /metrics  
 honorLabels: true  
---  
apiVersion: v1  
kind: Service  
metadata:  
 name: myapp-metrics  
 namespace: myapp-namespace  
 labels:  
 app: myapp  
spec:  
 ports:  
 - name: metrics  
 port: 8080  
 targetPort: 8080  
 selector:  
 app: myapp

###### PrometheusRule for Custom Alerts

apiVersion: monitoring.coreos.com/v1  
kind: PrometheusRule  
metadata:  
 name: myapp-alerts  
 namespace: myapp-namespace  
spec:  
 groups:  
 - name: myapp.rules  
 rules:  
 - alert: MyAppHighErrorRate  
 expr: |  
 (  
 sum(rate(myapp\_http\_requests\_total{status\_code=~"5.."}[5m]))  
 /  
 sum(rate(myapp\_http\_requests\_total[5m]))  
 ) > 0.05  
 for: 5m  
 labels:  
 severity: warning  
 annotations:  
 summary: "High error rate detected in MyApp"  
 description: "Error rate is {{ $value | humanizePercentage }} for the last 5 minutes"  
   
 - alert: MyAppHighLatency  
 expr: |  
 histogram\_quantile(0.95,   
 sum(rate(myapp\_http\_request\_duration\_seconds\_bucket[5m])) by (le)  
 ) > 1.0  
 for: 5m  
 labels:  
 severity: warning  
 annotations:  
 summary: "High latency detected in MyApp"  
 description: "95th percentile latency is {{ $value }}s"  
   
 - alert: MyAppPodCrashLooping  
 expr: rate(kube\_pod\_container\_status\_restarts\_total{namespace="myapp-namespace"}[15m]) > 0  
 for: 5m  
 labels:  
 severity: critical  
 annotations:  
 summary: "MyApp pod is crash looping"  
 description: "Pod {{ $labels.pod }} is restarting frequently"

##### Step 5: VM Workload Monitoring

###### KubeVirt VM Monitoring

apiVersion: monitoring.coreos.com/v1  
kind: ServiceMonitor  
metadata:  
 name: kubevirt-vm-metrics  
 namespace: kubevirt-system  
spec:  
 selector:  
 matchLabels:  
 prometheus.kubevirt.io: "true"  
 endpoints:  
 - port: metrics  
 interval: 30s  
 honorLabels: true  
---  
### VM-specific PrometheusRule  
apiVersion: monitoring.coreos.com/v1  
kind: PrometheusRule  
metadata:  
 name: vm-alerts  
 namespace: vm-workloads  
spec:  
 groups:  
 - name: vm.rules  
 rules:  
 - alert: VMHighCPUUsage  
 expr: kubevirt\_vmi\_vcpu\_seconds\_total > 0.8  
 for: 10m  
 labels:  
 severity: warning  
 annotations:  
 summary: "VM {{ $labels.name }} has high CPU usage"  
   
 - alert: VMHighMemoryUsage  
 expr: |  
 (  
 kubevirt\_vmi\_memory\_resident\_bytes  
 /  
 kubevirt\_vmi\_memory\_maximum\_bytes  
 ) > 0.9  
 for: 5m  
 labels:  
 severity: critical  
 annotations:  
 summary: "VM {{ $labels.name }} has high memory usage"

##### Step 6: Network Observability with Cilium Hubble

###### Enable Cilium Hubble

apiVersion: cilium.io/v2alpha1  
kind: CiliumConfig  
metadata:  
 name: cilium-config  
 namespace: cilium-system  
spec:  
 hubble:  
 enabled: true  
 metrics:  
 enabled:  
 - dns:query;ignoreAAAA  
 - drop  
 - tcp  
 - flow  
 - icmp  
 - http  
 relay:  
 enabled: true  
 ui:  
 enabled: true

##### Best Practices for Native Observability

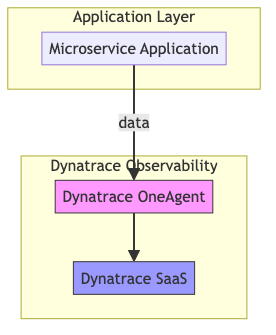
* **Consistent Labeling**: Use standardized labels across all metrics (service, version, environment)
* **Cardinality Management**: Avoid high-cardinality labels that can overwhelm Prometheus
* **Sampling Strategy**: Implement trace sampling for high-traffic applications (1-10% sample rate)
* **Log Levels**: Use appropriate log levels and structured logging with consistent fields
* **Resource Limits**: Set appropriate resource limits for observability components
* **Retention Policies**: Configure appropriate retention for metrics (30d) and logs (7d for debug, 30d for info/error)
* **Alert Fatigue**: Create meaningful alerts with proper thresholds and runbooks

##### 2. Observability with Dynatrace

#### Infrastructure Requirements

* Dynatrace OneAgent deployed on OpenShift nodes
* Dynatrace SaaS or Managed account
* Network access to Dynatrace monitoring endpoints

#### Architecture Overview



#### Implementation Steps

##### Step 1: Deploy Dynatrace OneAgent

* Use Dynatrace Operator for OpenShift to deploy OneAgent.

apiVersion: dynatrace.com/v1alpha1  
kind: Dynakube  
metadata:  
 name: dynakube  
 namespace: dynatrace  
spec:  
 oneAgent:  
 classicFullStack: true  
 apiUrl: "https://<environment-id>.live.dynatrace.com/api"  
 tokens: "api-token"

##### Step 2: Application Configuration

* No changes required for application code, as OneAgent will automatically instrument all services.

##### Step 3: Monitor and Analyze

* Use Dynatrace dashboards for comprehensive observability and performance analysis.
* Implement AI-driven alerts for proactive issue detection.

##### Best Practices

* **Ensure Network Connectivity**: Verify network connectivity to Dynatrace endpoints.
* **Optimize Resource Allocation**: Ensure sufficient resources for OneAgent processing.
* **Leverage Dynatrace AI**: Utilize Dynatrace’s AI capabilities for automated root cause analysis.

This comprehensive guide provides both native and third-party observability solutions, enabling holistic insights into application performance and behavior within the RH OVE ecosystem.

## Waf Firewalling

### Use Case: Web Application Firewall (WAF) Firewalling and Integration with F5 BigIP

#### Business Context

As enterprises adopt cloud-native architectures, securing web applications becomes crucial for protecting sensitive data and maintaining business continuity. This use case demonstrates how to implement a robust Web Application Firewall (WAF) using native L4-L7 capabilities and integrate it with F5 BigIP for enhanced security management within the RH OVE ecosystem.

#### Technical Requirements

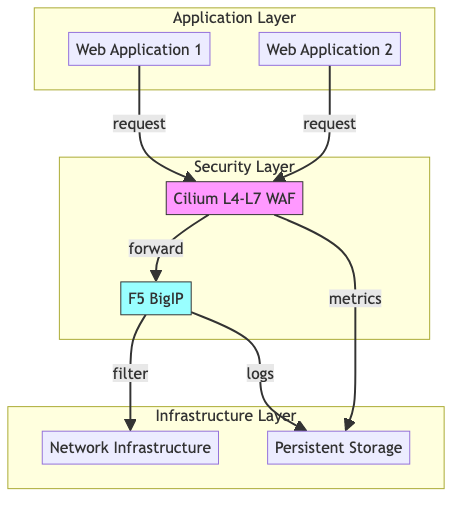
##### Infrastructure Requirements

* OpenShift 4.12+ cluster with Multus CNI enabled
* Cilium CNI for L4-L7 Policy Enforcement
* F5 BigIP for advanced traffic management and security policies
* Persistent storage solutions for logs and reports

##### Resource Requirements

* **CPU**: Sufficient compute resources to support firewall processing
* **Memory**: Adequate memory allocation for traffic inspection and logs
* **Storage**: High-performance storage for log retention and reporting
* **Network**: Scalable network infrastructure for seamless traffic flow

#### Architecture Overview



#### Implementation Steps

##### Step 1: Deploy Cilium L4-L7 Firewall

###### Cilium Configuration

apiVersion: cilium.io/v2  
kind: CiliumNetworkPolicy  
metadata:  
 name: web-app-waf-policy  
 namespace: infrastructure  
spec:  
 endpointSelector:  
 matchLabels:  
 app.kubernetes.io/name: web  
 ingress:  
 - rules:  
 http:  
 - method: "POST"  
 path: "/api"  
 - method: "GET"  
 path: "/"  
 egress:  
 - toEndpoints:  
 - matchLabels:  
 infrastructure: f5-bigip

##### Step 2: Integrate with F5 BigIP

###### F5 BigIP Virtual Server Configuration

* Configure F5 virtual server to handle traffic directed from Cilium WAF.
* Implement F5 policies for SSL termination, traffic redirection, and detailed logging.

### Example F5 BigIP CLI configuration  
create ltm virtual vs-web-app {  
 destination 192.168.1.100:80  
 ip-protocol tcp  
 profiles add { http { context clientside } }  
 pool my-web-app-pool  
 rules { waf-inspection }   
}  
  
### Associate WAF policies  
create ltm policy waf-inspection {  
 rules add {  
 10 { conditions { tcp } actions { forward pool-member my-web-app-pool  
 } }  
 }  
}

##### Step 3: Advanced Traffic Monitoring and Logging

###### Persistent Storage Configuration

* Configure persistent volumes for log storage using Cilium and F5 BigIP integrations.

apiVersion: v1  
kind: PersistentVolumeClaim  
metadata:  
 name: storage-logs  
 namespace: infrastructure  
spec:  
 accessModes:  
 - ReadWriteOnce  
 resources:  
 requests:  
 storage: 500Gi  
 storageClassName: high-performance

##### Step 4: Deploy Monitoring and Analytics Tools

###### Monitoring with Grafana and Prometheus

* Use Grafana dashboards to visualize traffic patterns and security metrics.
* Implement Prometheus alerting for suspicious activity detection.

### Grafana Dashboard Configuration  
apiVersion: integreatly.org/v1alpha1  
kind: GrafanaDashboard  
metadata:  
 name: waf-dashboard  
 namespace: monitoring  
spec:  
 json: |  
 {  
 "title": "Web Application Firewall Overview",  
 "panels": [  
 {  
 "type": "graph",  
 "title": "HTTP Requests",  
 "targets": [  
 { "expr": "sum(rate(http\_requests\_total[5m]))", "interval": ",5m" }  
 ]  
 }  
 ]  
 }

##### Troubleshooting and Maintenance

###### Common Issues and Solutions

* **Policy Misconfiguration**: Verify Cilium and F5 policy configurations for errors.
* **Performance Degradation**: Ensure adequate resources for Cilium and F5 processing.
* **Logging Failures**: Check storage availability and permissions.

##### Best Practices

* **Regular Audits**: Conduct regular audits of firewall rules and configurations.
* **Security Hardening**: Apply security patches to F5 BigIP and Cilium regularly.
* **Performance Monitoring**: Continuously monitor firewall performance and resource usage.

#### Integration with RH OVE Ecosystem

* **Seamless Traffic Flow**: Ensure smooth integration of traffic between application components and security layers.
* **Consistent Policy Management**: Use GitOps practices to manage and version firewall rules.
* **Network Observability**: Leverage Cilium Hubble for enhanced network observability within the OpenShift clusters.

This comprehensive guide provides the steps and best practices required to deploy and manage an effective Web Application Firewall solution in combination with F5 BigIP, ensuring robust protection for your web applications while seamlessly integrating within the RH OVE multi-cluster ecosystem.

## Publishing Events To Cmdb Siem

### Use Case: Publishing Kubernetes Events to CMDB and SIEM Solutions

#### Business Context

Enterprise IT environments require robust integration between Kubernetes events and existing IT service management and security incident management platforms. Integrating Kubernetes events with CMDB (such as ServiceNow or GLPI) and SIEM solutions enhances visibility, compliance, and proactive issue resolution. This use case focuses on using an event-bus to publish infrastructure changes, security issues, and admission control issues from Kubernetes to these platforms.

#### Technical Requirements

##### Infrastructure Requirements

* OpenShift 4.12+ cluster with event capture capabilities
* Event-bus solution such as NATS, Kafka, or RabbitMQ
* Connectivity to CMDB (e.g., ServiceNow, GLPI) and SIEM solutions
* Persistent storage for event retention and replay

##### Resource Requirements

* **CPU**: Adequate resources for handling and processing events
* **Memory**: Sufficient memory for event processing and transformation
* **Storage**: Persistent storage for event logs and audit trails
* **Network**: Reliable connectivity to external platforms

#### Event Categories and Sources

##### Infrastructure Change Events

* **Node Events**: Node ready/not ready, resource pressure, kubelet issues
* **Pod Events**: Pod scheduling, image pulling, container creation/termination
* **Storage Events**: Volume mounting/unmounting, PVC binding issues
* **Network Events**: Service endpoint changes, ingress controller updates
* **Resource Events**: Deployment scaling, ConfigMap/Secret updates

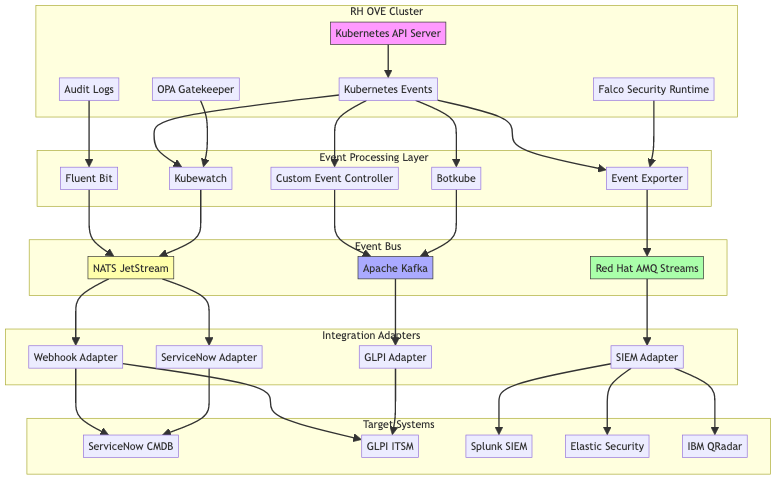
##### Security Events

* **RBAC Violations**: Unauthorized access attempts, permission denials
* **Pod Security Policy Violations**: SecurityContext violations, privileged access attempts
* **Image Security Events**: Image scanning failures, vulnerable image deployments
* **Network Policy Violations**: Blocked network connections, policy enforcement
* **Certificate Events**: TLS certificate expiration, rotation failures

##### Admission Control Events

* **ValidatingAdmissionWebhook**: Policy violations, validation failures
* **MutatingAdmissionWebhook**: Resource mutations, injection failures
* **Resource Quota Violations**: Quota exceeded, resource limit breaches
* **LimitRange Violations**: Container resource limit violations

#### Architecture Overview



#### Implementation Steps

##### Step 1: Deploy Event Bus Solution

###### Deploy NATS Streaming

apiVersion: apps/v1  
kind: Deployment  
metadata:  
 name: nats-streaming  
 namespace: event-bus  
spec:  
 replicas: 3  
 selector:  
 matchLabels:  
 app: nats-streaming  
 template:  
 metadata:  
 labels:  
 app: nats-streaming  
 spec:  
 containers:  
 - name: nats  
 image: nats-streaming:latest  
 ports:  
 - containerPort: 4222  
 - containerPort: 8222  
 resources:  
 limits:  
 memory: 512Mi  
 cpu: 0.5  
 requests:  
 memory: 256Mi  
 cpu: 0.25

##### Step 2: Configure Event Processing Solutions

###### Option 1: Deploy Kubewatch for Event Monitoring

Kubewatch is a Kubernetes watcher that publishes notifications to various channels.

apiVersion: apps/v1  
kind: Deployment  
metadata:  
 name: kubewatch  
 namespace: event-processing  
spec:  
 replicas: 1  
 selector:  
 matchLabels:  
 app: kubewatch  
 template:  
 metadata:  
 labels:  
 app: kubewatch  
 spec:  
 serviceAccountName: kubewatch  
 containers:  
 - name: kubewatch  
 image: bitnami/kubewatch:latest  
 env:  
 - name: KW\_CONFIG  
 value: /config/kubewatch-config.yaml  
 volumeMounts:  
 - name: config  
 mountPath: /config  
 resources:  
 limits:  
 memory: 256Mi  
 cpu: 100m  
 requests:  
 memory: 128Mi  
 cpu: 50m  
 volumes:  
 - name: config  
 configMap:  
 name: kubewatch-config  
---  
apiVersion: v1  
kind: ConfigMap  
metadata:  
 name: kubewatch-config  
 namespace: event-processing  
data:  
 kubewatch-config.yaml: |  
 namespace: ""  
 handler:  
 webhook:  
 url: "http://event-router.event-processing.svc.cluster.local:8080/webhook"  
 resource:  
 deployment: true  
 replicationcontroller: false  
 replicaset: false  
 daemonset: true  
 services: true  
 pod: true  
 job: true  
 node: true  
 clusterrole: false  
 serviceaccount: false  
 persistentvolume: true  
 namespace: true  
 secret: false  
 configmap: false  
 ingress: true  
 event: true  
---  
apiVersion: v1  
kind: ServiceAccount  
metadata:  
 name: kubewatch  
 namespace: event-processing  
---  
apiVersion: rbac.authorization.k8s.io/v1  
kind: ClusterRole  
metadata:  
 name: kubewatch  
rules:  
- apiGroups: [""]  
 resources: ["pods", "nodes", "namespaces", "events", "services", "persistentvolumes"]  
 verbs: ["list", "watch", "get"]  
- apiGroups: ["apps"]  
 resources: ["deployments", "daemonsets", "replicasets"]  
 verbs: ["list", "watch", "get"]  
- apiGroups: ["extensions"]  
 resources: ["ingresses"]  
 verbs: ["list", "watch", "get"]  
- apiGroups: ["batch"]  
 resources: ["jobs"]  
 verbs: ["list", "watch", "get"]  
---  
apiVersion: rbac.authorization.k8s.io/v1  
kind: ClusterRoleBinding  
metadata:  
 name: kubewatch  
roleRef:  
 apiGroup: rbac.authorization.k8s.io  
 kind: ClusterRole  
 name: kubewatch  
subjects:  
- kind: ServiceAccount  
 name: kubewatch  
 namespace: event-processing

###### Option 2: Deploy Botkube for Advanced Event Processing

Botkube provides intelligent event filtering and routing capabilities.

apiVersion: v1  
kind: ConfigMap  
metadata:  
 name: botkube-config  
 namespace: event-processing  
data:  
 config.yaml: |  
 settings:  
 clustername: "rh-ove-cluster"  
 kubectl:  
 enabled: false  
 configwatcher: true  
 upgradenotifier: true  
   
 communications:  
 webhook:  
 enabled: true  
 url: "http://event-router.event-processing.svc.cluster.local:8080/botkube"  
   
 resources:  
 - name: "v1/pods"  
 namespaces:  
 include:  
 - "default"  
 - "kube-system"  
 - "openshift-\*"  
 events:  
 - create  
 - delete  
 - error  
 updateSetting:  
 includeDiff: true  
 fields:  
 - "spec.containers[\*].image"  
 - "status.phase"  
   
 - name: "v1/services"  
 namespaces:  
 include:  
 - "default"  
 events:  
 - create  
 - delete  
 - error  
   
 - name: "v1/nodes"  
 events:  
 - create  
 - delete  
 - error  
   
 - name: "v1/events"  
 events:  
 - error  
 - warning  
 filters:  
 objectAnnotations:  
 pattern: ".\*"  
 reason:  
 - "FailedScheduling"  
 - "Unhealthy"  
 - "FailedMount"  
 - "NetworkNotReady"  
 - "NodeNotReady"  
 - "Rebooted"  
 - "SystemOOM"  
---  
apiVersion: apps/v1  
kind: Deployment  
metadata:  
 name: botkube  
 namespace: event-processing  
spec:  
 replicas: 1  
 selector:  
 matchLabels:  
 app: botkube  
 template:  
 metadata:  
 labels:  
 app: botkube  
 spec:  
 serviceAccountName: botkube  
 containers:  
 - name: botkube  
 image: infracloudio/botkube:latest  
 env:  
 - name: CONFIG\_PATH  
 value: "/tmp/config/"  
 - name: LOG\_LEVEL  
 value: "info"  
 volumeMounts:  
 - name: config  
 mountPath: "/tmp/config"  
 resources:  
 limits:  
 memory: 256Mi  
 cpu: 100m  
 requests:  
 memory: 128Mi  
 cpu: 50m  
 volumes:  
 - name: config  
 configMap:  
 name: botkube-config  
---  
apiVersion: v1  
kind: ServiceAccount  
metadata:  
 name: botkube  
 namespace: event-processing  
---  
apiVersion: rbac.authorization.k8s.io/v1  
kind: ClusterRole  
metadata:  
 name: botkube  
rules:  
- apiGroups: ["\*"]  
 resources: ["\*"]  
 verbs: ["get", "watch", "list"]  
---  
apiVersion: rbac.authorization.k8s.io/v1  
kind: ClusterRoleBinding  
metadata:  
 name: botkube  
roleRef:  
 apiGroup: rbac.authorization.k8s.io  
 kind: ClusterRole  
 name: botkube  
subjects:  
- kind: ServiceAccount  
 name: botkube  
 namespace: event-processing

###### Option 3: Deploy Kubernetes Event Exporter

For more granular control over event processing and routing.

apiVersion: apps/v1  
kind: Deployment  
metadata:  
 name: event-exporter  
 namespace: event-processing  
spec:  
 replicas: 1  
 selector:  
 matchLabels:  
 app: event-exporter  
 template:  
 metadata:  
 labels:  
 app: event-exporter  
 spec:  
 serviceAccountName: event-exporter  
 containers:  
 - name: event-exporter  
 image: giantswarm/event-exporter:latest  
 args:  
 - --config-file=/config/config.yaml  
 - --logtostderr  
 - --v=2  
 volumeMounts:  
 - name: config  
 mountPath: /config  
 resources:  
 limits:  
 memory: 256Mi  
 cpu: 100m  
 requests:  
 memory: 128Mi  
 cpu: 50m  
 volumes:  
 - name: config  
 configMap:  
 name: event-exporter-config  
---  
apiVersion: v1  
kind: ConfigMap  
metadata:  
 name: event-exporter-config  
 namespace: event-processing  
data:  
 config.yaml: |  
 logLevel: 2  
 logFormat: json  
 receivers:  
 - name: "webhook-receiver"  
 webhook:  
 endpoint: "http://event-router.event-processing.svc.cluster.local:8080/events"  
 headers:  
 X-Source: "kubernetes-events"  
 Content-Type: "application/json"  
 - name: "kafka-receiver"  
 kafka:  
 brokers:  
 - "kafka.event-bus.svc.cluster.local:9092"  
 topic: "kubernetes-events"  
   
 route:  
 routes:  
 # Infrastructure change events  
 - match:  
 - receiver: "webhook-receiver"  
 name: "infrastructure-changes"  
 - receiver: "kafka-receiver"  
 name: "infrastructure-changes"  
 groupBy: ["namespace", "reason"]  
 groupWait: 10s  
 groupInterval: 30s  
 repeatInterval: 1h  
 routes:  
 - match:  
 - field: "reason"  
 op: "in"  
 values: ["Scheduled", "Pulled", "Created", "Started"]  
 name: "pod-lifecycle"  
 - match:  
 - field: "reason"  
 op: "in"  
 values: ["NodeReady", "NodeNotReady", "Rebooted"]  
 name: "node-events"  
 - match:  
 - field: "reason"  
 op: "in"  
 values: ["SuccessfulMount", "FailedMount"]  
 name: "storage-events"  
   
 # Security events  
 - match:  
 - receiver: "webhook-receiver"  
 name: "security-events"  
 groupBy: ["namespace", "reason", "involvedObject.kind"]  
 routes:  
 - match:  
 - field: "reason"  
 op: "in"  
 values: ["Forbidden", "Unauthorized", "PolicyViolation"]  
 name: "security-violations"  
 - match:  
 - field: "message"  
 op: "re"  
 values: [".\*security.\*", ".\*violation.\*", ".\*denied.\*"]  
 name: "security-related"  
   
 # Admission control events  
 - match:  
 - receiver: "kafka-receiver"  
 name: "admission-control"  
 routes:  
 - match:  
 - field: "reason"  
 op: "in"  
 values: ["FailedCreate", "AdmissionWebhookDenied"]  
 name: "admission-denied"  
---  
apiVersion: v1  
kind: ServiceAccount  
metadata:  
 name: event-exporter  
 namespace: event-processing  
---  
apiVersion: rbac.authorization.k8s.io/v1  
kind: ClusterRole  
metadata:  
 name: event-exporter  
rules:  
- apiGroups: [""]  
 resources: ["events"]  
 verbs: ["get", "watch", "list"]  
---  
apiVersion: rbac.authorization.k8s.io/v1  
kind: ClusterRoleBinding  
metadata:  
 name: event-exporter  
roleRef:  
 apiGroup: rbac.authorization.k8s.io  
 kind: ClusterRole  
 name: event-exporter  
subjects:  
- kind: ServiceAccount  
 name: event-exporter  
 namespace: event-processing

###### Option 4: Custom Event Controller with Event Router

For maximum flexibility, deploy a custom event router that can transform and route events.

apiVersion: apps/v1  
kind: Deployment  
metadata:  
 name: event-router  
 namespace: event-processing  
spec:  
 replicas: 2  
 selector:  
 matchLabels:  
 app: event-router  
 template:  
 metadata:  
 labels:  
 app: event-router  
 spec:  
 containers:  
 - name: event-router  
 image: quay.io/example/event-router:latest  
 ports:  
 - containerPort: 8080  
 env:  
 - name: NATS\_URL  
 value: "nats://nats.event-bus.svc.cluster.local:4222"  
 - name: KAFKA\_BROKERS  
 value: "kafka.event-bus.svc.cluster.local:9092"  
 - name: LOG\_LEVEL  
 value: "info"  
 resources:  
 limits:  
 memory: 512Mi  
 cpu: 200m  
 requests:  
 memory: 256Mi  
 cpu: 100m  
 livenessProbe:  
 httpGet:  
 path: /health  
 port: 8080  
 initialDelaySeconds: 30  
 periodSeconds: 10  
 readinessProbe:  
 httpGet:  
 path: /ready  
 port: 8080  
 initialDelaySeconds: 5  
 periodSeconds: 5  
---  
apiVersion: v1  
kind: Service  
metadata:  
 name: event-router  
 namespace: event-processing  
spec:  
 selector:  
 app: event-router  
 ports:  
 - name: http  
 port: 8080  
 targetPort: 8080  
 type: ClusterIP

##### Step 3: Deploy Event Bus Solutions

###### Deploy Red Hat AMQ Streams (Kafka)

apiVersion: kafka.strimzi.io/v1beta2  
kind: Kafka  
metadata:  
 name: event-cluster  
 namespace: event-bus  
spec:  
 kafka:  
 version: 3.4.0  
 replicas: 3  
 listeners:  
 - name: plain  
 port: 9092  
 type: internal  
 tls: false  
 - name: tls  
 port: 9093  
 type: internal  
 tls: true  
 config:  
 offsets.topic.replication.factor: 3  
 transaction.state.log.replication.factor: 3  
 transaction.state.log.min.isr: 2  
 default.replication.factor: 3  
 min.insync.replicas: 2  
 inter.broker.protocol.version: "3.4"  
 log.retention.hours: 168 # 7 days  
 log.segment.bytes: 1073741824 # 1GB  
 storage:  
 type: persistent-claim  
 size: 100Gi  
 class: ocs-storagecluster-ceph-rbd  
 resources:  
 requests:  
 memory: 2Gi  
 cpu: 500m  
 limits:  
 memory: 4Gi  
 cpu: 1  
 zookeeper:  
 replicas: 3  
 storage:  
 type: persistent-claim  
 size: 10Gi  
 class: ocs-storagecluster-ceph-rbd  
 resources:  
 requests:  
 memory: 1Gi  
 cpu: 500m  
 limits:  
 memory: 2Gi  
 cpu: 1  
 entityOperator:  
 topicOperator: {}  
 userOperator: {}  
---  
apiVersion: kafka.strimzi.io/v1beta2  
kind: KafkaTopic  
metadata:  
 name: kubernetes-events  
 namespace: event-bus  
 labels:  
 strimzi.io/cluster: event-cluster  
spec:  
 partitions: 12  
 replicas: 3  
 config:  
 retention.ms: 604800000 # 7 days  
 segment.ms: 3600000 # 1 hour  
---  
apiVersion: kafka.strimzi.io/v1beta2  
kind: KafkaTopic  
metadata:  
 name: security-events  
 namespace: event-bus  
 labels:  
 strimzi.io/cluster: event-cluster  
spec:  
 partitions: 6  
 replicas: 3  
 config:  
 retention.ms: 2592000000 # 30 days  
 segment.ms: 3600000 # 1 hour  
---  
apiVersion: kafka.strimzi.io/v1beta2  
kind: KafkaTopic  
metadata:  
 name: infrastructure-changes  
 namespace: event-bus  
 labels:  
 strimzi.io/cluster: event-cluster  
spec:  
 partitions: 6  
 replicas: 3  
 config:  
 retention.ms: 1209600000 # 14 days  
 segment.ms: 3600000 # 1 hour

###### Deploy NATS JetStream

apiVersion: apps/v1  
kind: StatefulSet  
metadata:  
 name: nats  
 namespace: event-bus  
spec:  
 serviceName: nats  
 replicas: 3  
 selector:  
 matchLabels:  
 app: nats  
 template:  
 metadata:  
 labels:  
 app: nats  
 spec:  
 containers:  
 - name: nats  
 image: nats:alpine  
 ports:  
 - containerPort: 4222  
 name: client  
 - containerPort: 7422  
 name: leafnodes  
 - containerPort: 6222  
 name: cluster  
 - containerPort: 8222  
 name: monitor  
 - containerPort: 7777  
 name: metrics  
 args:  
 - --config  
 - /etc/nats-config/nats.conf  
 volumeMounts:  
 - name: config-volume  
 mountPath: /etc/nats-config  
 - name: data  
 mountPath: /data  
 resources:  
 requests:  
 memory: 512Mi  
 cpu: 200m  
 limits:  
 memory: 1Gi  
 cpu: 500m  
 livenessProbe:  
 httpGet:  
 path: /healthz  
 port: 8222  
 initialDelaySeconds: 10  
 timeoutSeconds: 5  
 readinessProbe:  
 httpGet:  
 path: /healthz  
 port: 8222  
 initialDelaySeconds: 10  
 timeoutSeconds: 5  
 volumes:  
 - name: config-volume  
 configMap:  
 name: nats-config  
 volumeClaimTemplates:  
 - metadata:  
 name: data  
 spec:  
 accessModes:  
 - ReadWriteOnce  
 resources:  
 requests:  
 storage: 10Gi  
 storageClassName: ocs-storagecluster-ceph-rbd  
---  
apiVersion: v1  
kind: ConfigMap  
metadata:  
 name: nats-config  
 namespace: event-bus  
data:  
 nats.conf: |  
 port: 4222  
 http\_port: 8222  
   
 cluster {  
 name: nats-cluster  
 port: 6222  
 routes = [  
 nats://nats-0.nats.event-bus.svc.cluster.local:6222  
 nats://nats-1.nats.event-bus.svc.cluster.local:6222  
 nats://nats-2.nats.event-bus.svc.cluster.local:6222  
 ]  
 }  
   
 jetstream {  
 store\_dir: "/data"  
 max\_memory\_store: 256MB  
 max\_file\_store: 2GB  
 }  
   
 accounts {  
 $SYS { users = [ { user: "admin", pass: "password" } ] }  
 events {  
 jetstream: enabled  
 users = [  
 { user: "event-publisher", pass: "publisher-secret" }  
 { user: "event-consumer", pass: "consumer-secret" }  
 ]  
 }  
 }  
---  
apiVersion: v1  
kind: Service  
metadata:  
 name: nats  
 namespace: event-bus  
spec:  
 selector:  
 app: nats  
 clusterIP: None  
 ports:  
 - name: client  
 port: 4222  
 - name: cluster  
 port: 6222  
 - name: monitor  
 port: 8222

##### Step 4: Configure Integration Adapters

###### ServiceNow Integration Adapter

apiVersion: apps/v1  
kind: Deployment  
metadata:  
 name: servicenow-adapter  
 namespace: integration  
spec:  
 replicas: 2  
 selector:  
 matchLabels:  
 app: servicenow-adapter  
 template:  
 metadata:  
 labels:  
 app: servicenow-adapter  
 spec:  
 containers:  
 - name: adapter  
 image: quay.io/example/servicenow-adapter:latest  
 env:  
 - name: SERVICENOW\_INSTANCE  
 value: "https://dev12345.service-now.com"  
 - name: SERVICENOW\_USERNAME  
 valueFrom:  
 secretKeyRef:  
 name: servicenow-credentials  
 key: username  
 - name: SERVICENOW\_PASSWORD  
 valueFrom:  
 secretKeyRef:  
 name: servicenow-credentials  
 key: password  
 - name: KAFKA\_BROKERS  
 value: "event-cluster-kafka-bootstrap.event-bus.svc.cluster.local:9092"  
 - name: KAFKA\_TOPICS  
 value: "kubernetes-events,infrastructure-changes,security-events"  
 - name: SERVICENOW\_TABLE\_MAPPING  
 value: |  
 {  
 "kubernetes-events": "incident",  
 "infrastructure-changes": "change\_request",  
 "security-events": "sn\_si\_incident"  
 }  
 resources:  
 requests:  
 memory: 256Mi  
 cpu: 100m  
 limits:  
 memory: 512Mi  
 cpu: 200m  
 livenessProbe:  
 httpGet:  
 path: /health  
 port: 8080  
 initialDelaySeconds: 30  
 periodSeconds: 10  
 readinessProbe:  
 httpGet:  
 path: /ready  
 port: 8080  
 initialDelaySeconds: 5  
 periodSeconds: 5  
---  
apiVersion: v1  
kind: Secret  
metadata:  
 name: servicenow-credentials  
 namespace: integration  
type: Opaque  
data:  
 username: <base64-encoded-username>  
 password: <base64-encoded-password>  
---  
apiVersion: v1  
kind: ConfigMap  
metadata:  
 name: servicenow-field-mapping  
 namespace: integration  
data:  
 mapping.json: |  
 {  
 "incident": {  
 "short\_description": "{{ .reason }}: {{ .message | truncate 100 }}",  
 "description": "Kubernetes Event Details:\n\nNamespace: {{ .namespace }}\nObject: {{ .involvedObject.kind }}/{{ .involvedObject.name }}\nReason: {{ .reason }}\nMessage: {{ .message }}\nFirst Seen: {{ .firstTimestamp }}\nLast Seen: {{ .lastTimestamp }}\nCount: {{ .count }}",  
 "category": "Software",  
 "subcategory": "Kubernetes",  
 "urgency": "{{ if eq .type \"Warning\" }}2{{ else }}3{{ end }}",  
 "impact": "{{ if contains .reason \"Failed\" }}2{{ else }}3{{ end }}",  
 "assignment\_group": "Platform Engineering",  
 "caller\_id": "kubernetes-system",  
 "u\_kubernetes\_cluster": "{{ .clusterName }}",  
 "u\_kubernetes\_namespace": "{{ .namespace }}",  
 "u\_kubernetes\_object": "{{ .involvedObject.kind }}/{{ .involvedObject.name }}"  
 },  
 "change\_request": {  
 "short\_description": "Infrastructure Change: {{ .reason }}",  
 "description": "{{ .message }}",  
 "category": "Standard",  
 "type": "Normal",  
 "risk": "Low",  
 "impact": "3",  
 "priority": "4",  
 "assignment\_group": "Platform Engineering"  
 }  
 }

###### Publish Events to SIEM

* Configure integration using SIEM platform’s ingress mechanisms.
* Example for Splunk:

apiVersion: batch/v1beta1  
kind: CronJob  
metadata:  
 name: publish-events-splunk  
 namespace: integration  
spec:  
 schedule: "\*/5 \* \* \* \*"  
 jobTemplate:  
 spec:  
 template:  
 spec:  
 containers:  
 - name: splunk-publisher  
 image: splunk-integration:latest  
 env:  
 - name: SPLUNK\_HEC\_URL  
 value: "https://splunk-hec.local/services/collector"  
 - name: SPLUNK\_TOKEN  
 valueFrom:  
 secretKeyRef:  
 name: splunk-credentials  
 key: hec\_token  
 command: ["publish", "--source=nats", "--target=splunk"]

##### Step 4: Persistent Storage for Event Retention

###### Storage Configuration

* Use OpenShift Data Foundation for persistent storage.

apiVersion: v1  
kind: PersistentVolumeClaim  
metadata:  
 name: events-storage  
 namespace: event-bus  
spec:  
 accessModes:  
 - ReadWriteOnce  
 resources:  
 requests:  
 storage: 100Gi  
 storageClassName: ocs-storagecluster-ceph-rbd

##### Troubleshooting

###### Common Issues and Solutions

* **Connectivity Issues**: Verify network policies and DNS resolution to external CMDB and SIEM platforms.
* **Event Capture Failures**: Ensure event-controller configurations are correct and have appropriate RBAC permissions.
* **Performance Bottlenecks**: Scale event-bus components and monitor resource utilization.

##### Best Practices

* **Secure Integration**: Use secure channels (TLS) for communication with external platforms.
* **Scalable Event Processing**: Implement horizontal scaling for event-bus solutions.
* **Audit and Compliance**: Maintain detailed logs and audit trails for event processing.

#### Integration with RH OVE Ecosystem

* **Automation and Policy Enforcement**: Use event-driven automation for rapid remediation and policy enforcement.
* **Proactive Monitoring**: Enhance monitoring and observability for real-time insights into infrastructure and application changes.

This use case guides implementing a scalable and flexible solution to publish vital Kubernetes events to enterprise IT management and security platforms, enhancing operational visibility and control.