# Red Hat OpenShift Administration II: Configuring a Production Cluster

DO280

# Let’s meet each other!

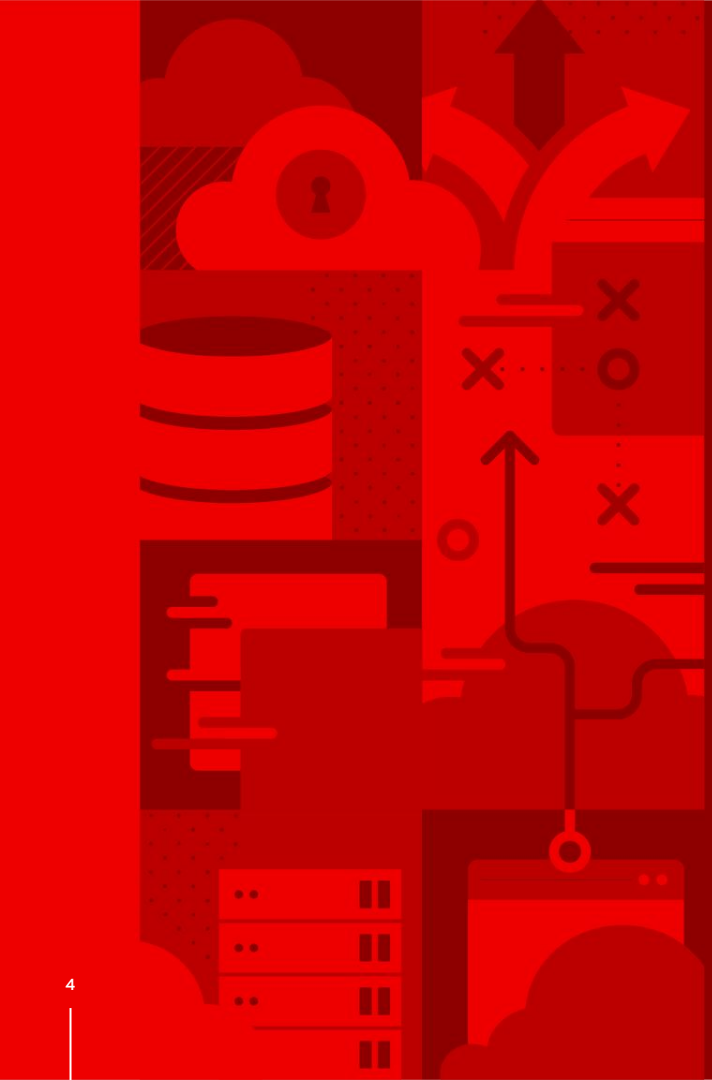

Francesco Marchioni - fmarchio@redhat.com
Red Hat Certified Architect (RHCA)

# Course Scheduling


| Mon | Tue | Wed | Thu | Fri |
|---|---|---|---|---|
| Chapters 1 - 2 | Chapters 3 - 4 | Chapters 5 - 6 | Chapters 7 - 8 | Chapters 9 |
| Labs review | Labs review | Labs review | Labs review | Final review |



Red Hat

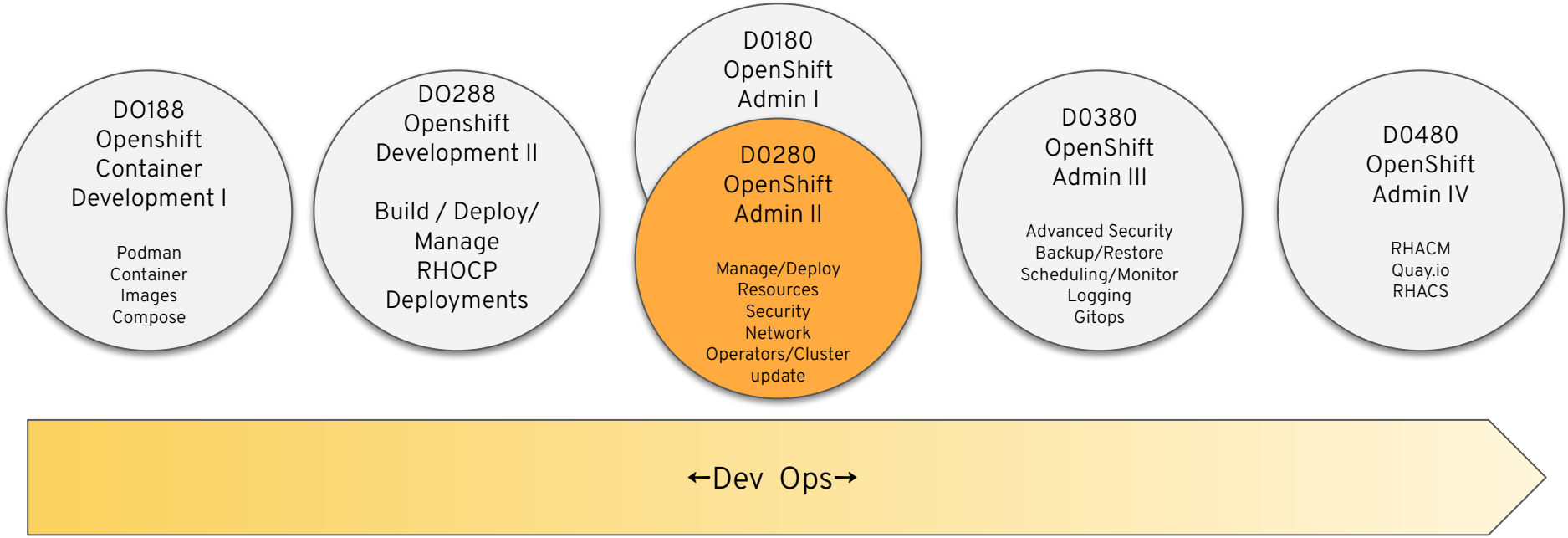


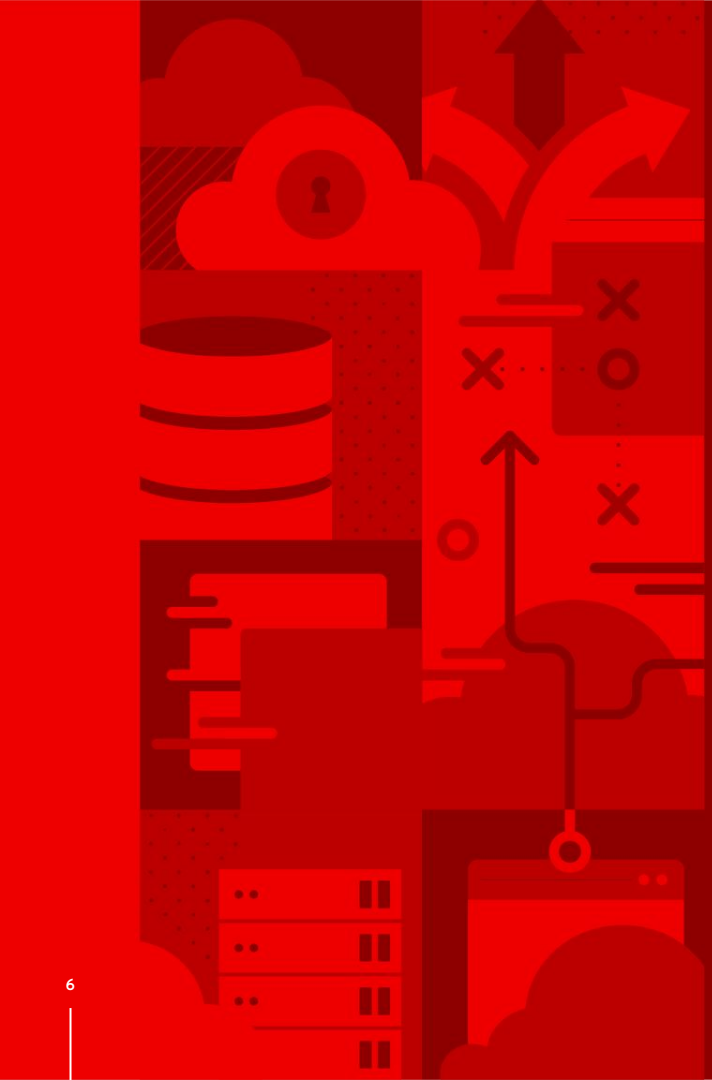
# OpenShift Learning Path



Red Hat

# OpenShift Training



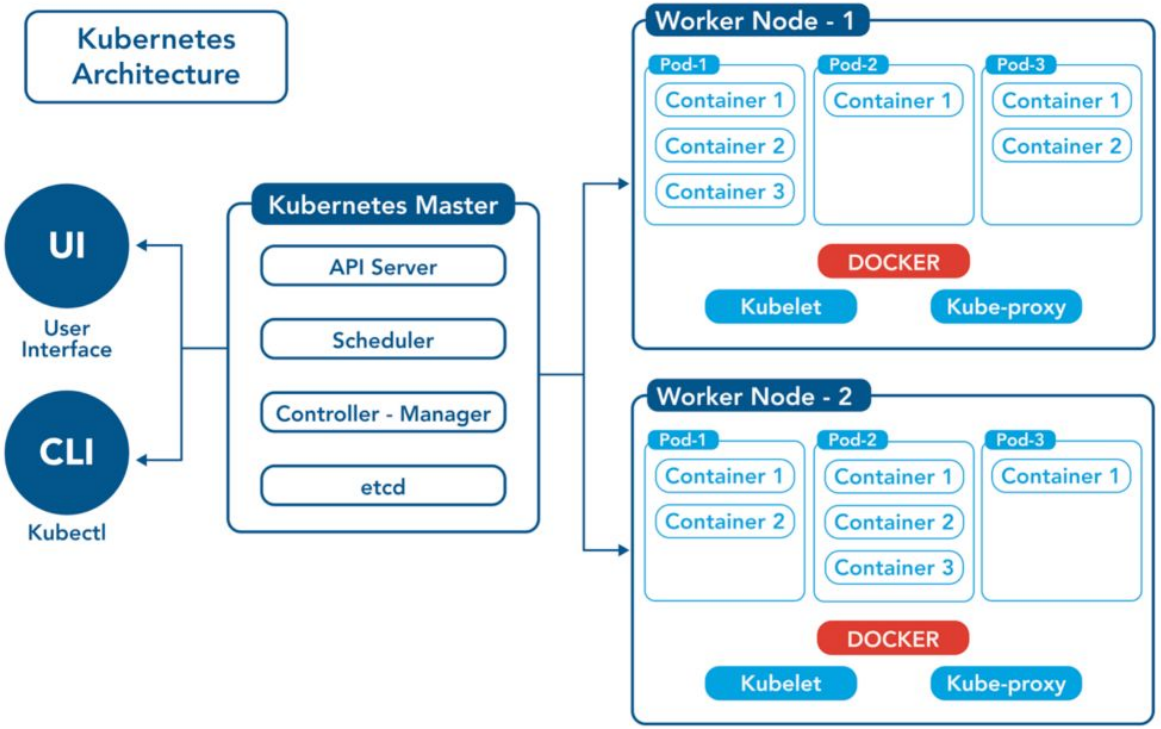


# Introducing Kubernetes & OpenShift

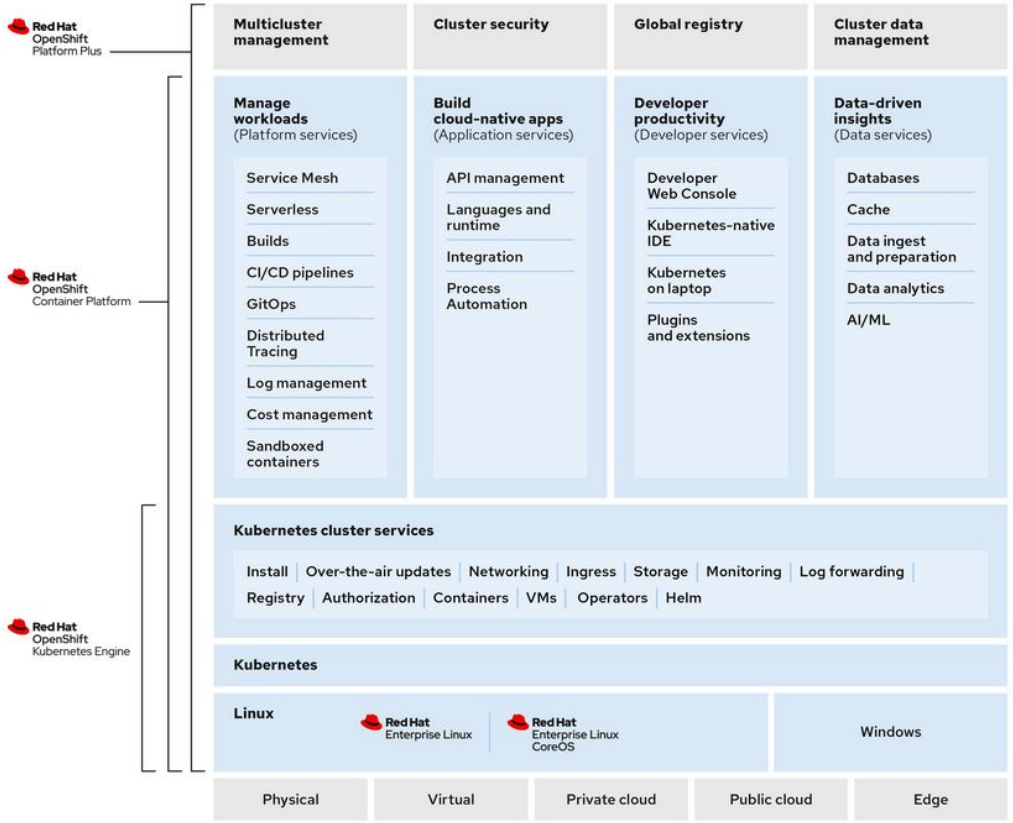


Red Hat

# Kubernetes Architecture



# Comparing OpenShift versions




Red Hat

# Openshift Offering

## Self-Managed Editions

- Red Hat OpenShift Container Platform
- Red Hat OpenShift Kubernetes Engine
- Red Hat OpenShift Virtualization Engine

## Managed Editions

Red Hat | aws

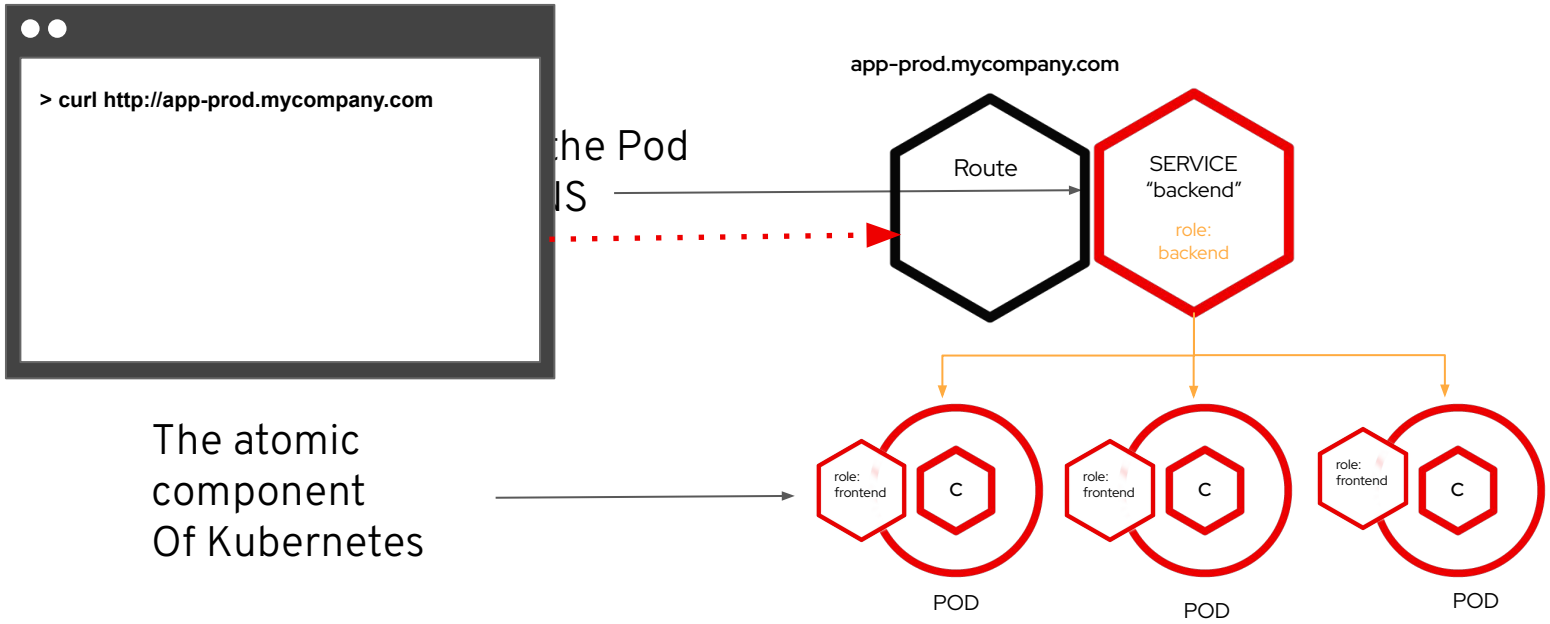
Red Hat | Microsoft Azure

Red Hat OpenShift Dedicated

Available on Google Cloud

Red Hat | IBM.

# Recap from previous episodes.....

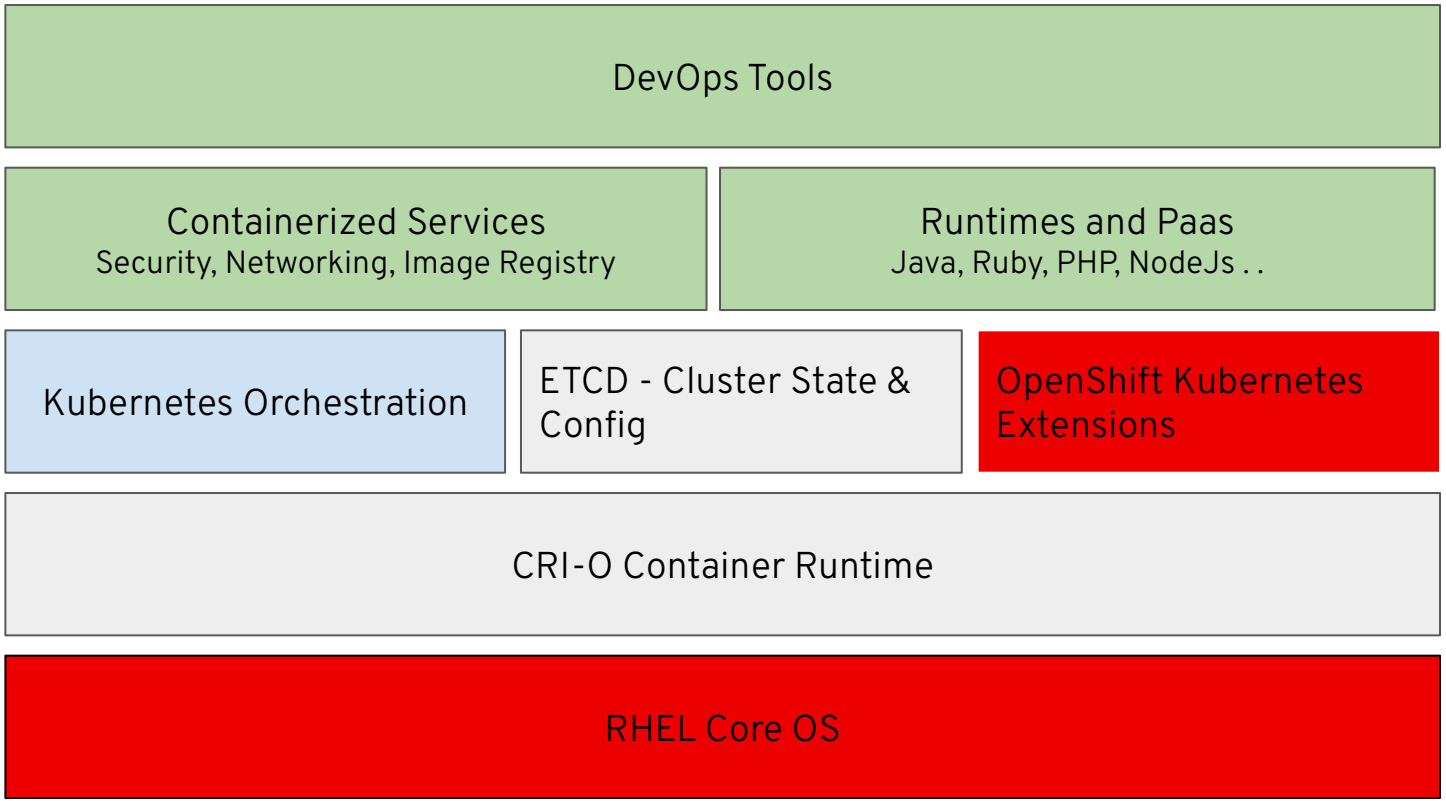


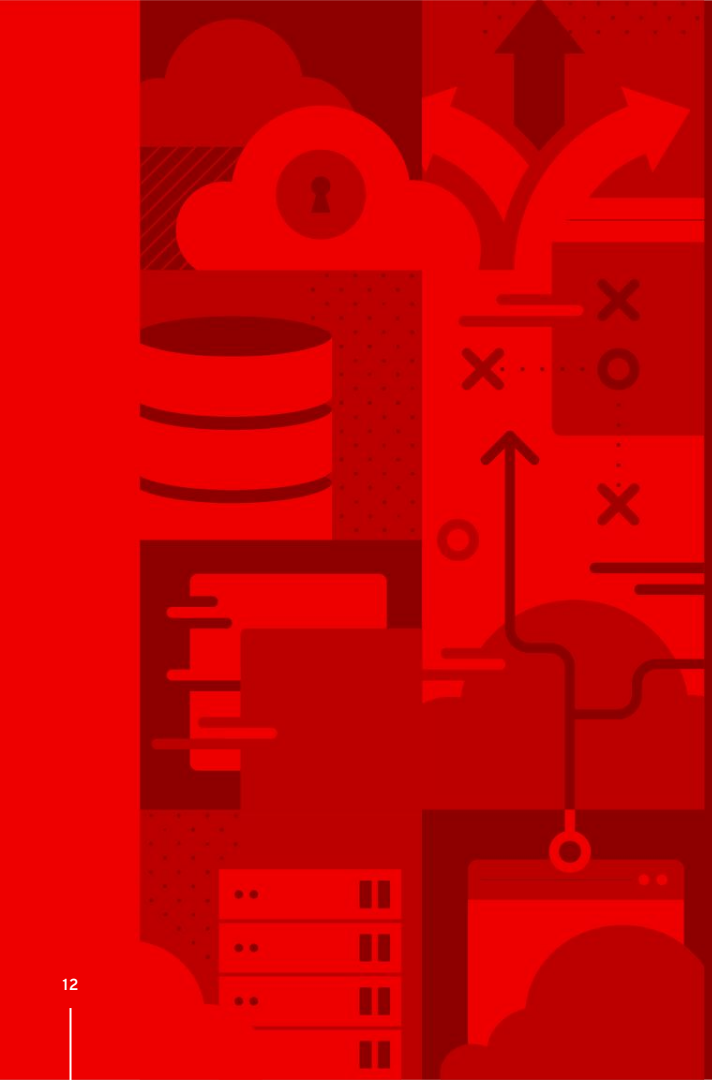

The atomic component Of Kubernetes



Red Hat

# OpenShift Architecture





# Resource Management With OpenShift



Red Hat

# Resource Management with OpenShift

## Imperative Resource Management

- Uses direct CLI commands to create, modify, and delete resources
- Immediate execution with oc commands

```
oc create deployment myapp  -image=nginx
```

## Declarative Resource Management

- Uses YAML or JSON manifests to define the desired state
- Applied using oc create -f <file>.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: mypod
spec:
  containers:
    - name: mycontainer
      image: busybox
```

# Kubernetes Manifest Creation

There are multiple ways to create Manifests:

- Create from scratch ( or use existing templates)
- Create from imperative commands:

```
kubectl create deployment hello -o yaml --image nginx:v1.0  --save-config
--dry-run=client > deployment.yaml
```

- Use tools (Kustomize/Helm) to generate or transform files into valid Manifest files

# Applying Manifest Changes

- oc apply creates or updates OpenShift/Kubernetes resources from a manifest
- It uses a declarative approach – you declare the desired state, and OpenShift applies the changes.

```
oc apply -f <filename.yaml>
```

🛠️ How It Works:

- If the resource does not exist, it gets created.
- If it already exists, it gets updated to match the YAML file.
- Only the fields defined in the file are managed.

# What is Kustomize?

- An open-source configuration management tool for Kubernetes
- Provides a template-free solution for customizing Kubernetes resource configuration

Do you need more Kubernetes tools?

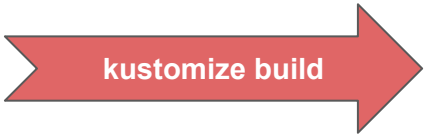
Check here


# Base Overlay

base.yaml

```yaml
apiVersion: tenant/v1
kind: Fake
metadata:
  name: demo
spec:
  team:
    name: it-gos
  overlay: base
```

```
├── base
│   ├── base.yaml
│   └── kustomization.yaml
└── preprod
    ├── kustomization.yaml
    └── patch.yaml
```




```yaml
apiVersion: tenant/v1
kind: Fake
metadata:
  name: demo
spec:
  overlay: base
  team:
    name: it-gos
```

kustomization.yaml

```yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization

resources:
- base.yaml
```

# Preprod Overlay

patch.yaml

```yaml
apiVersion: tenant/v1
kind: Fake
metadata:
  name: demo
spec:
  overlay: preprod
```

```
├── base
│   ├── base.yaml
│   └── kustomization.yaml
└── preprod
    ├── kustomization.yaml
    └── patch.yaml
```

kustomization.yaml

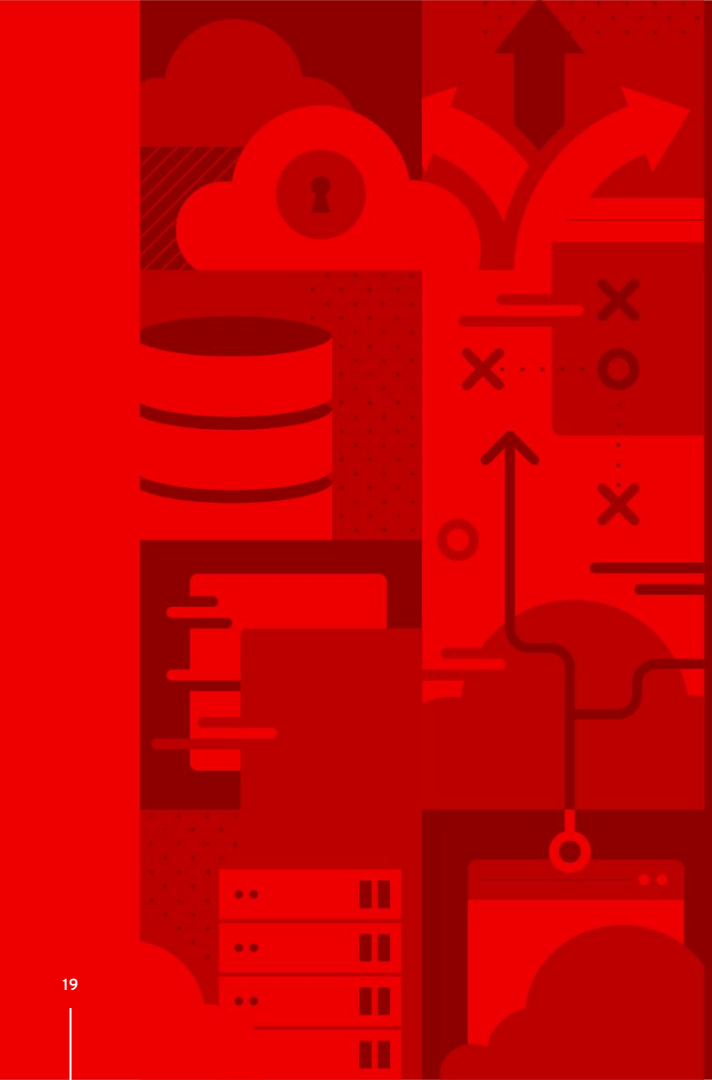
```yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization

resources:
- ../base

patchesStrategicMerge:
- patch.yaml
```

**kustomize build**

```yaml
apiVersion: tenant/v1
kind: Fake
metadata:
  name: demo
spec:
  overlay: preprod
  team:
    name: it-gos
```



# Helm Charts

# Helm Components

## CLI

The *helm* binary provides a mechanism for interacting with the helm ecosystem

## Templates

Provides dynamic capabilities for Kubernetes resources that are to be instantiated

## Charts

Packages representing Kubernetes deployable resources

## Values

Configuration variables that are injected into templated resources

## Revisions

Configurations of a chart at a particular point in time


# Package manager for Kubernetes

## Project Overview

- https://helm.sh/
- https://github.com/helm/helm

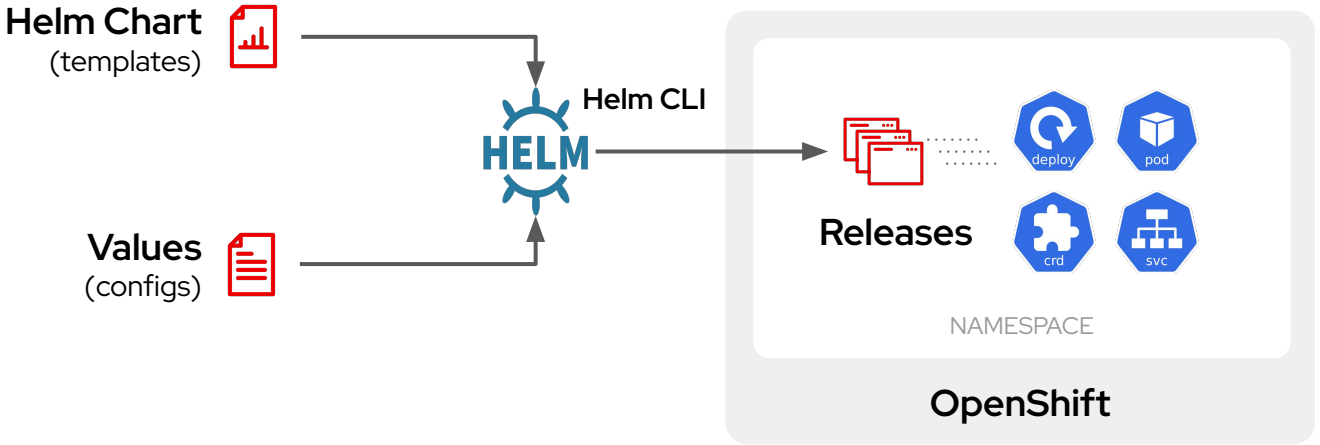

## Top level CNCF Project

- 2016 - Joined CNCF
- 2020 - Graduated status

## Active development community

- 13,000+ contributors
- 1,700+ contributing companies
- 9,500+ code commits

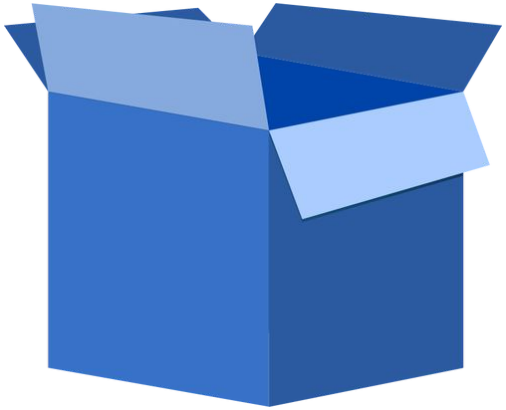
# Helm Components



## Helm Chart


A collection of files that describe a set of related Kubernetes resources.

## Helm Release



A chart instance once installed / upgraded / rollback into a namespace

# Creating and Deploying a Helm Chart

- Creating a  new Chart

```
$ helm create mychart
```

- Install Chart to Kubernetes cluster

```
$ helm install mychart
```

```
mychart/
  Chart.yaml          # Information about the chart
  LICENSE             # OPTIONAL: Chart license
  README.md           # OPTIONAL: README file
  values.yaml         # The default configuration values
  values.schema.json  # OPTIONAL: A JSON Schema for values
  charts/             # Dependency charts
  crds/               # Custom Resource Definitions
  templates/          # Directory of templates
  templates/NOTES.txt # OPTIONAL: Usage notes
```

Helm Chart Directory Structure

# Chart.yaml

Helm metadata file

```
apiVersion: v2
name: mychart
version: 0.1.0
description: A Helm chart for Kubernetes
type: application

dependencies:
  - name: jenkins
    version: 2.5.0

appVersion: "1.16.0"
```

# Managing Helm Charts

- Listing Charts

```
$ helm list
```

- Upgrading a release

```
$ helm upgrade <release_name> <chart>
--set version=1.1
```

- Rolling back an upgrade

```
$ helm rollback <release> <revision>
```

- Uninstalling a Chart

```
$ helm uninstall <release>
```

# Setting Helm Charts Values

Values for a chart can be overridden by values contained in files or explicitly set

## Using values.yaml

```
$ helm install -f <values_file>  ./mychart
```

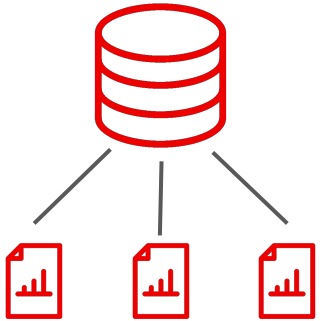
## Command Line

```
$ helm install --set foo=bar  ./mychart
```

Multiple values can be specified

# Locating Helm Charts Values in Repositories

Share and source charts from Repositories to accelerate productivity



## Repository management

The **helm repo** command can be used to manage repositories

## Installing charts from repositories

The helm repo subcommand

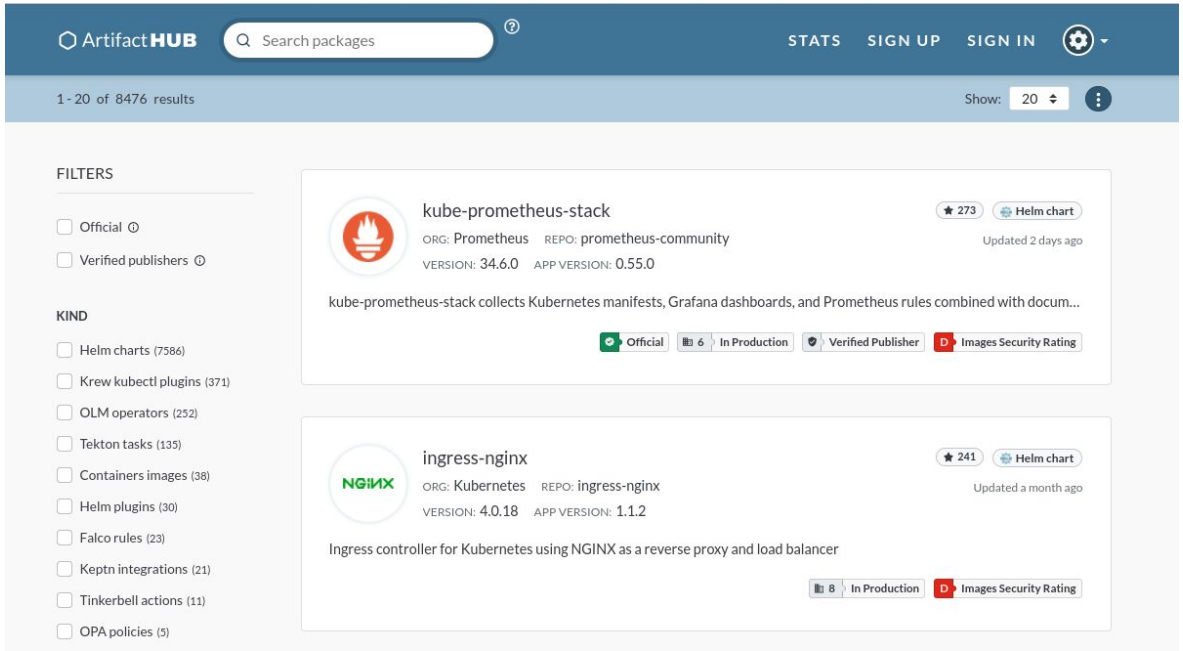
```
$ helm install redhat-cop/jenkins --generate-name
```

## Searching for charts

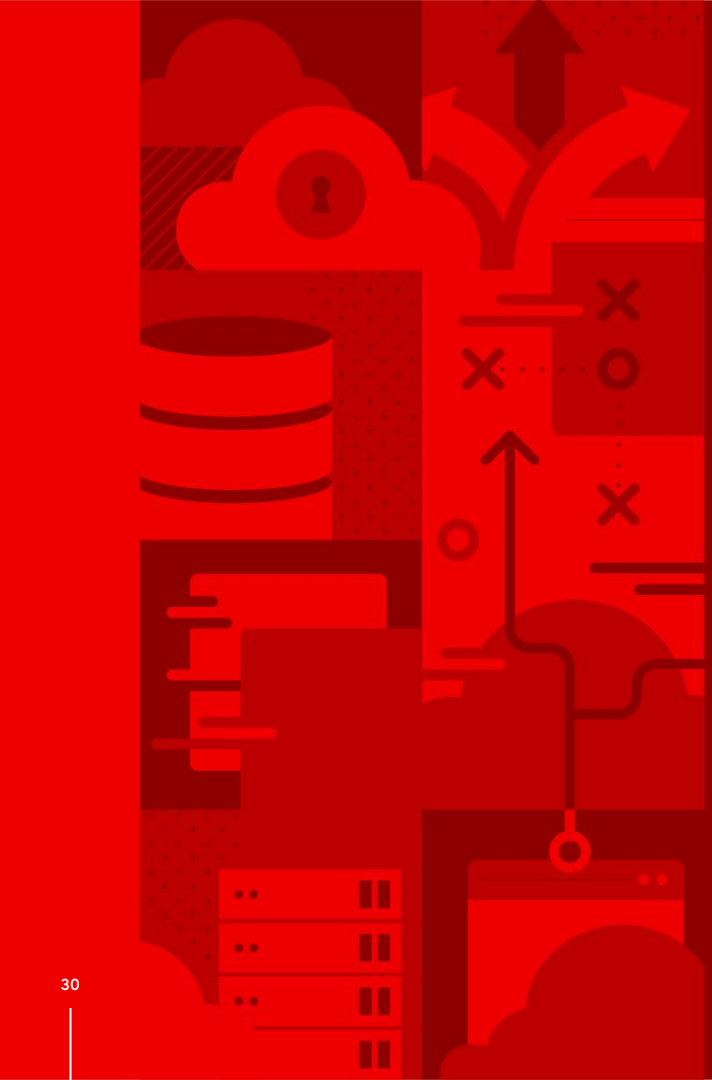
Charts located within repositories can be searched by keywords

```
$ helm search repo nginx
```

# Artifact Hub - Finding Helm Charts




[Artifact Hub: Discover & launch great Kubernetes-ready apps]



# Openshift Authentication & Authorization

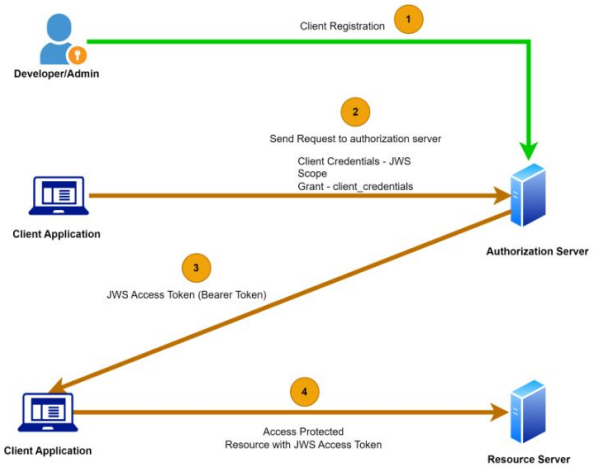


Red Hat

# OpenShift Authentication

The OpenShift API has two methods for authenticating requests:

- OAuth access tokens
- X.509 client certificates

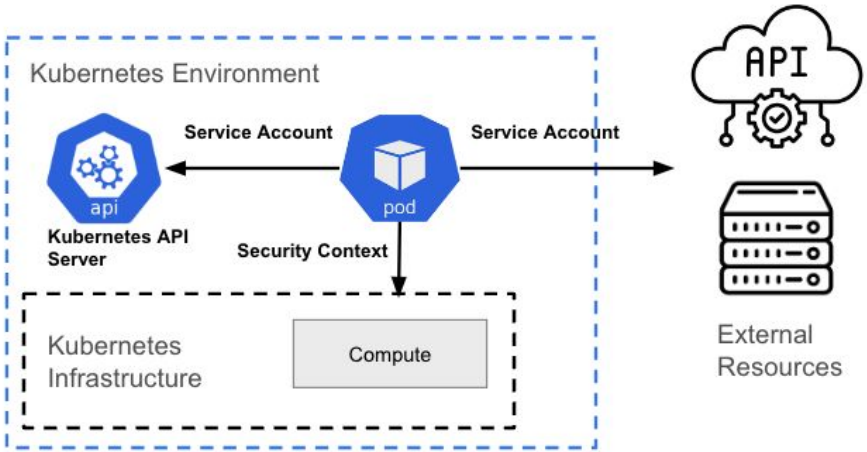


# OpenShift Users & Groups

- 👤 **User**: Represents an actor interacting with the API; permissions are granted directly or via group membership.
- 🆔 **Identity**: Records details of successful user authentications, including the source identity provider.
- 🤖 **Service Account**: Enables applications to access the API without using regular user credentials, maintaining credential integrity.
- 👥 **Group**: A collection of users for streamlined permission assignment through authorization policies.
- 🛡️ **Role**: Specifies allowed API operations on resources and is assigned to users, groups, or service accounts to manage access.

# Focus on the Service Account

A Service account is designed for applications or automated processes to access the API securely without using personal credentials.



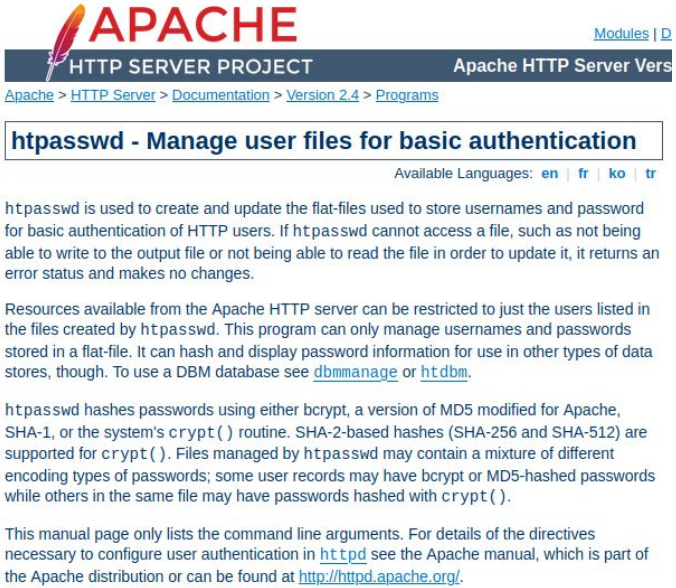
# OpenShift Identity Providers

- **HTPasswd**: Validates usernames and passwords against a secret that stores credentials that are generated by using the htpasswd command.
- **OpenID Connect**: Integrates with an OpenID Connect identity provider by using an Authorization Code Flow.
- **Keystone**: Enables shared authentication with an OpenStack Keystone v3 server.
- **LDAP**: Configures the LDAP identity provider to validate usernames and passwords against an LDAPv3 server, by using simple bind authentication.
- **GitHub**: Configures a GitHub identity provider to validate usernames and passwords against GitHub or the GitHub Enterprise OAuth authentication server.



# Httpd Provider Example

```
apiVersion: config.openshift.io/v1
kind: OAuth
metadata:
  name: cluster
spec:
  identityProviders:
  - name: my_htpasswd_provider
    mappingMethod: claim
    type: HTPasswd
    htpasswd:
      fileData:
        name: htpasswd-secret
```



APACHE HTTP SERVER PROJECT

Modules | D

Apache > HTTP Server > Documentation > Version 2.4 > Programs

**htpasswd - Manage user files for basic authentication**

Available Languages: en | fr | ko | tr

htpasswd is used to create and update the flat-files used to store usernames and password for basic authentication of HTTP users. If htpasswd cannot access a file, such as not being able to write to the output file or not being able to read the file in order to update it, it returns an error status and makes no changes.

Resources available from the Apache HTTP server can be restricted to just the users listed in the files created by htpasswd. This program can only manage usernames and passwords stored in a flat-file. It can hash and display password information for use in other types of data stores, though. To use a DBM database see dbmmanage or htdbm.

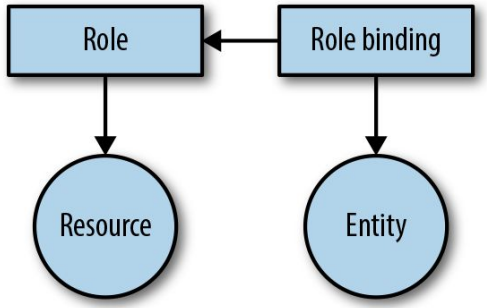
htpasswd hashes passwords using either bcrypt, a version of MD5 modified for Apache, SHA-1, or the system's crypt() routine. SHA-2-based hashes (SHA-256 and SHA-512) are supported for crypt(). Files managed by htpasswd may contain a mixture of different encoding types of passwords; some user records may have bcrypt or MD5-hashed passwords while others in the same file may have passwords hashed with crypt().

This manual page only lists the command line arguments. For details of the directives necessary to configure user authentication in httpd see the Apache manual, which is part of the Apache distribution or can be found at http://httpd.apache.org/.

# OpenShift RBAC

**Role-based access control (RBAC)** is a technique for managing access to resources in a computer system.

- **Rule**: Allowed actions for objects or groups of objects.
- **Role**: Sets of rules. Users and groups can be associated with multiple roles.
- **Binding**: Assignment of users or groups to a role.



# OpenShift Roles

**Cluster Roles** (Global Scope):

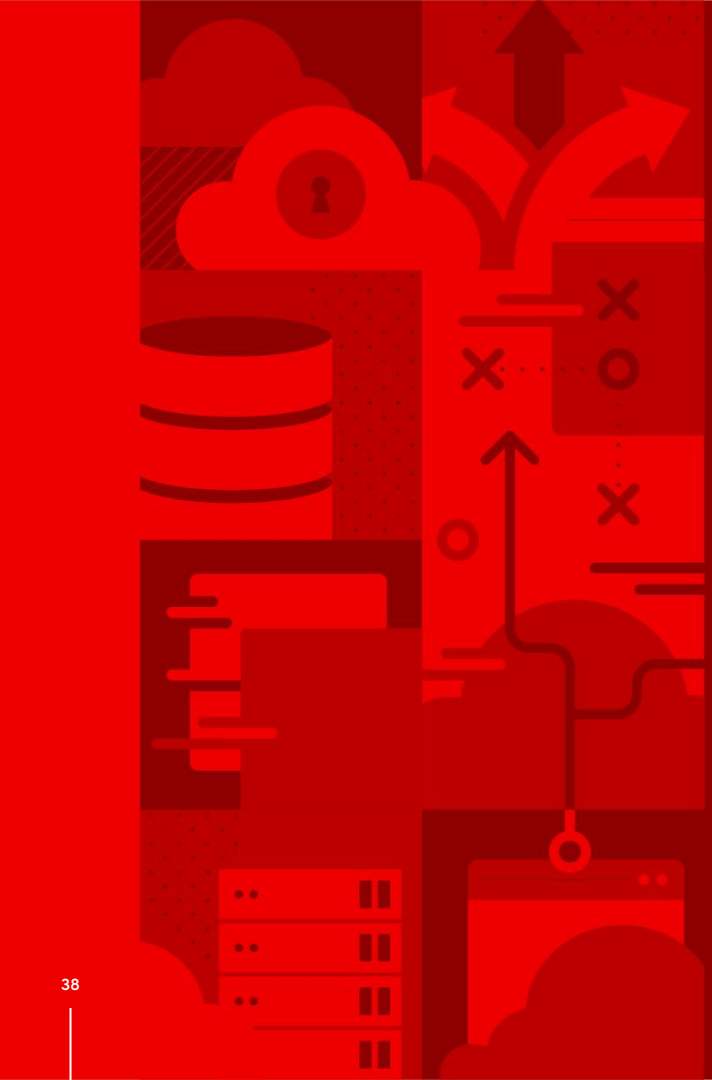
```
oc adm policy –help
```

- cluster-admin – Full control over the entire cluster.
- cluster-reader – Read-only access to all cluster resources.
- admin – Full control within a project (namespace), except modifying quota and membership.
- edit – Can modify most resources but cannot manage RBAC or quotas.
- view – Read-only access to most resources, except secrets and RBAC.

**Namespace** (Project) Roles

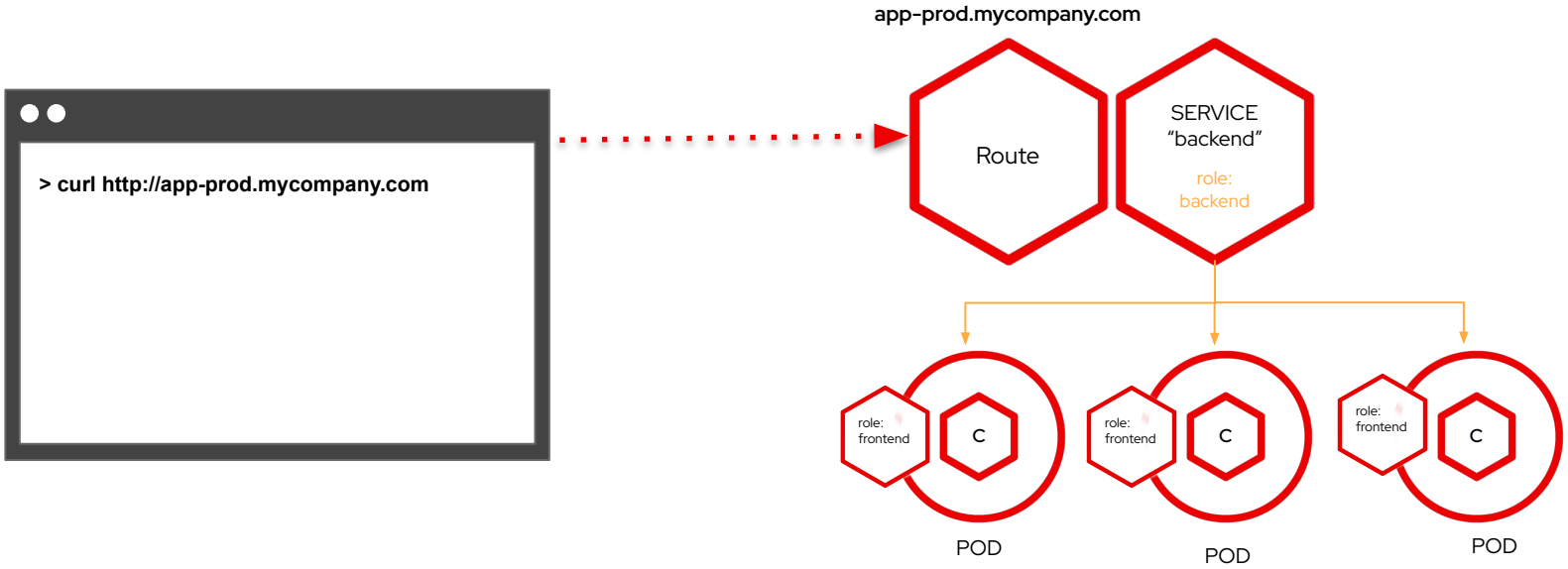
```
oc policy –help
```

- self-provisioner – Allows users to create new projects.
- basic-user – Can view basic cluster info and self-related details.
- registry-editor – Can manage images in the internal registry.
- registry-viewer – Can only view images in the internal registry.
- system:image-puller – Can pull images from the internal registry.



# Openshift Routes

# Routes make services accessible to clients outside the environment via real-world urls






Red Hat

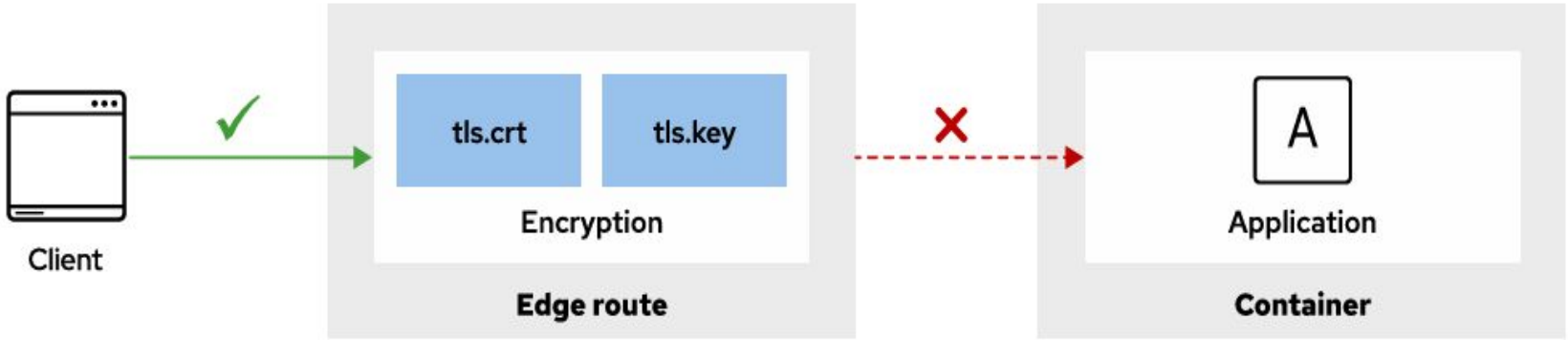
# Routes vs Ingress

| Feature | Ingress | Route |
|---|---|---|
| Standard Kubernetes object | ✓ | |
| External access to services | ✓ | ✓ |
| Persistent (sticky) sessions | ✓ | ✓ |
| Load-balancing strategies (e.g. round robin) | ✓ | ✓ |
| Rate-limit and throttling | ✓ | ✓ |
| IP whitelisting | ✓ | ✓ |
| TLS edge termination | ✓ | ✓ |
| TLS re-encryption | ✓ | ✓ |
| TLS passthrough | ✓ | ✓ |
| Multiple weighted backends (split traffic) | | ✓ |
| Generated pattern-based hostnames | | ✓ |
| Wildcard domains | | ✓ |

# OpenShift Edge Routing

An **Edge Route** in OpenShift is a type of route that terminates TLS at the router, meaning:

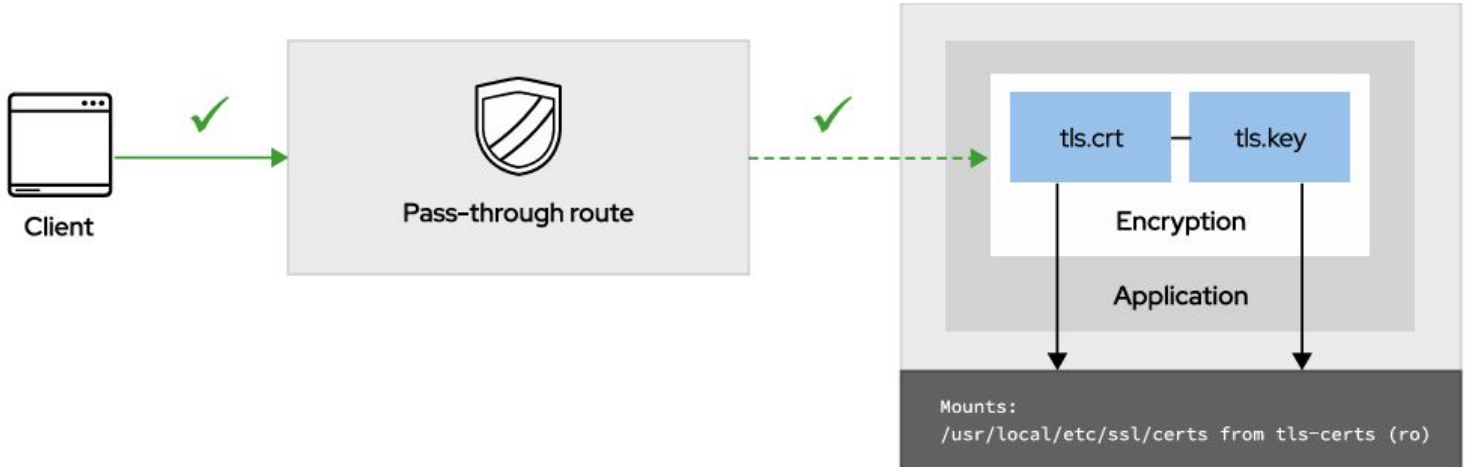
- The OpenShift router handles SSL/TLS decryption.
- Traffic is then forwarded unencrypted (HTTP) to the backend service.
- The backend does not need to handle TLS, simplifying app configuration.



# OpenShift Passthrough Routing

Encrypted traffic is sent straight to the destination pod without TLS termination from the router.

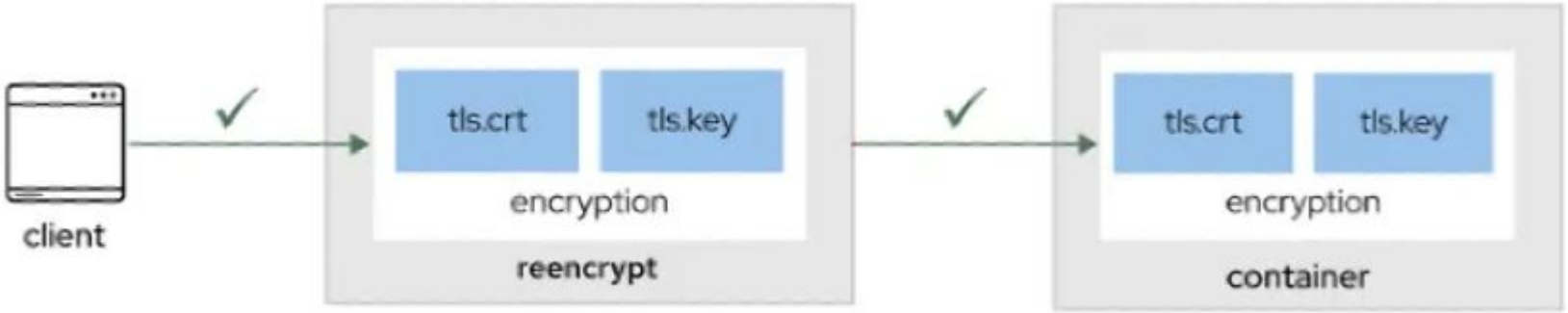
- In this mode, the application is responsible for serving certificates for the traffic.
- Supports mutual Authentication

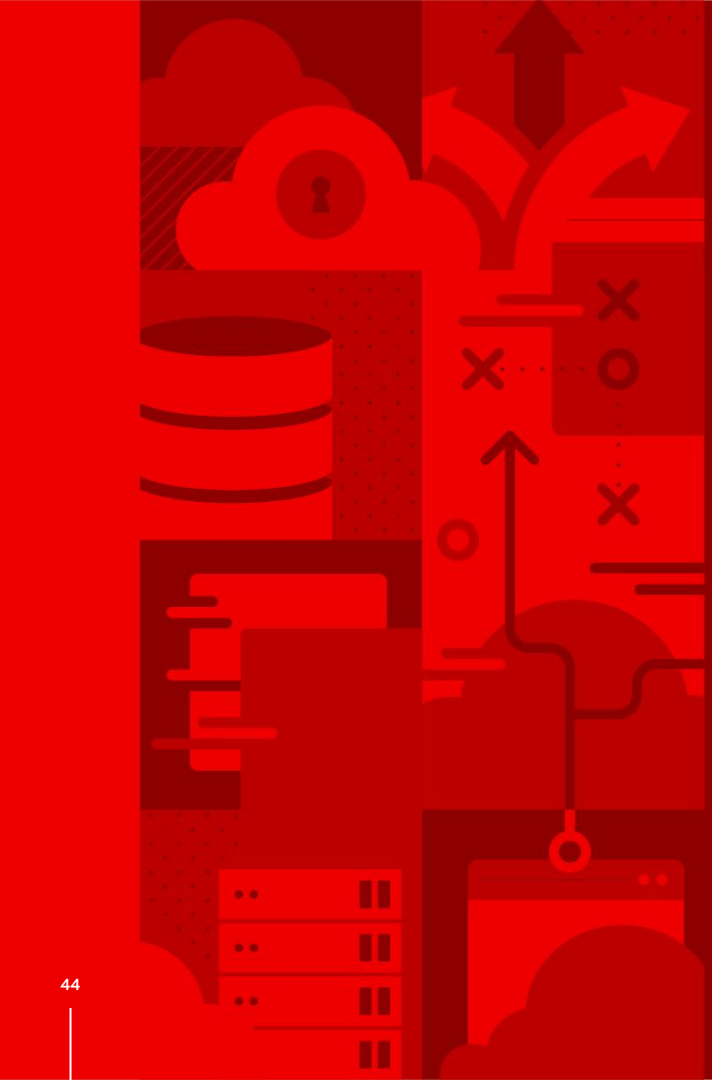


# OpenShift Re-encryption Routing

- First, re-encrypt routes terminate the encryption between the client and the router. This encryption uses a certificate with a FQDN that is trusted by the client

- Then, the router re-encrypts the connection when accessing a cluster service. This internal communication requires a certificate for the target service with an OpenShift FQDN





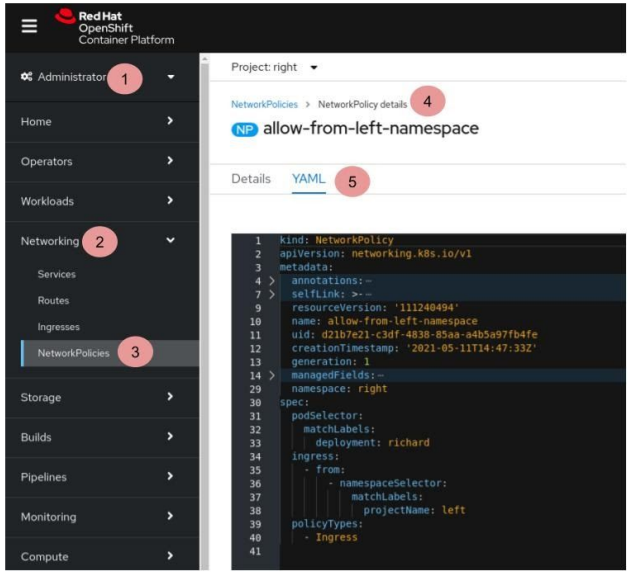
# Network Policies



Red Hat

# Network Policies

- Based on labeling or annotations
- Empty label selector match all
- Rules for allowing:
  - **Ingress** -> who can connect to this POD
  - **Egress** -> where can this POD connect to

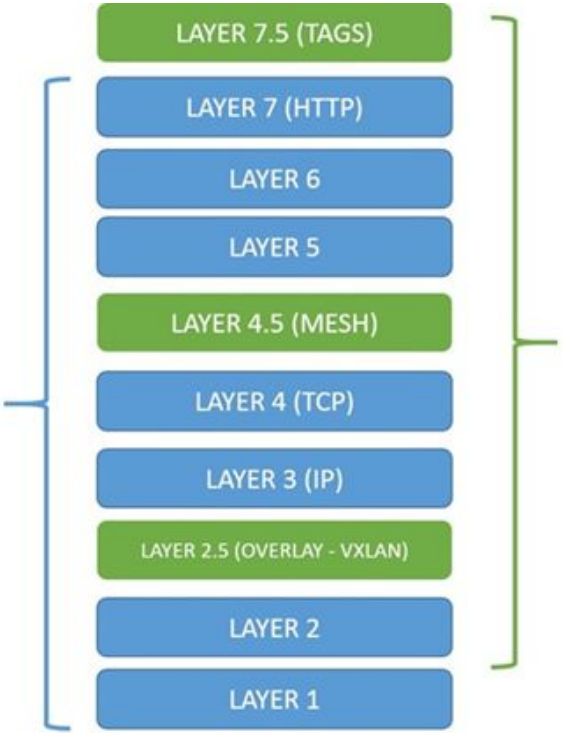


# Default Network Policy

- OpenShift SDN supports using NetworkPolicy in its default network isolation mode.
- By default, **all Pods in a project are accessible from other Pods** and network endpoints.
- To isolate one or more Pods in a project, you can create NetworkPolicy objects in that project to indicate the allowed incoming connections.
- A network policy is a specification of how groups of pods (using labels) are allowed to communicate with each other and other network endpoints.

# Network Policies in the OSI Stack

Traditional OSI Stack

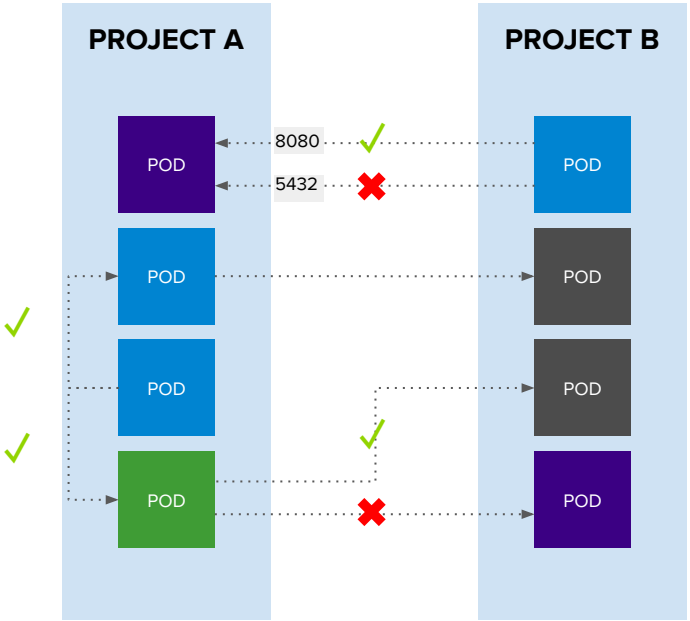



## Service Mesh Stack

Allows to execute domain specific routing and network policies for services across the container environment

# NetworkPolicy

Traffic in/out from the project




- Allow all traffic inside the cluster
- Allow **Ingress** traffic from Blue Pods and port 8080

- Allow all traffic inside the cluster
- Allow **Egress** traffic to Blue Pods with PodSelector=green

# With and without a deny-all policy

**Without a deny-all policy**:

- Pods that are *not* selected by any NetworkPolicy can communicate freely (based on the default behavior of the network plugin, which usually allows everything).
- Pods *selected* by a NetworkPolicy, however, can only communicate if the traffic matches the defined rules.

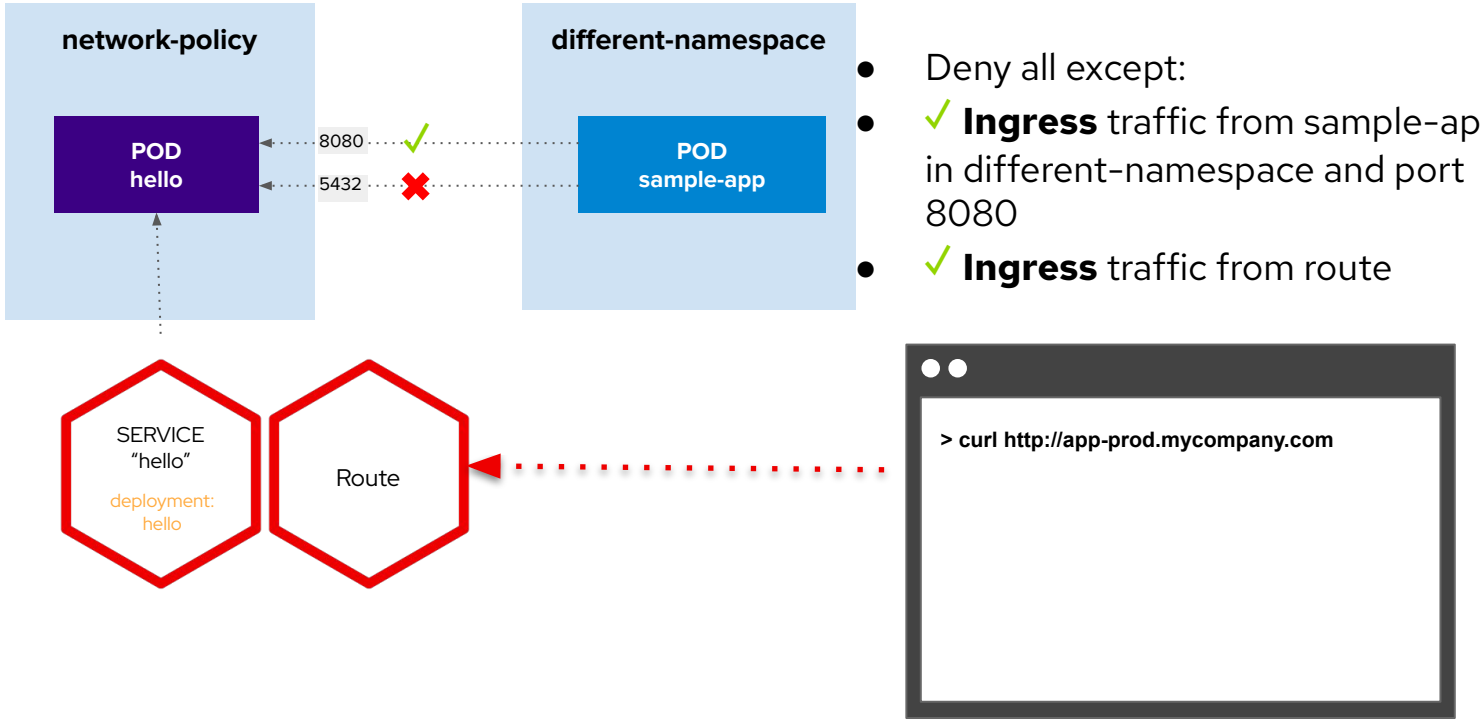
**With a deny-all policy**:

- All traffic (ingress, egress, or both) is blocked for the target pods, unless explicitly allowed by additional specific rules.

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

# Network Policy in one word 🧙 !

# NetworkPolicy - lab network-policy




- Deny all except:
- ✓ **Ingress** traffic from sample-app in different-namespace and port 8080
- ✓ **Ingress** traffic from route



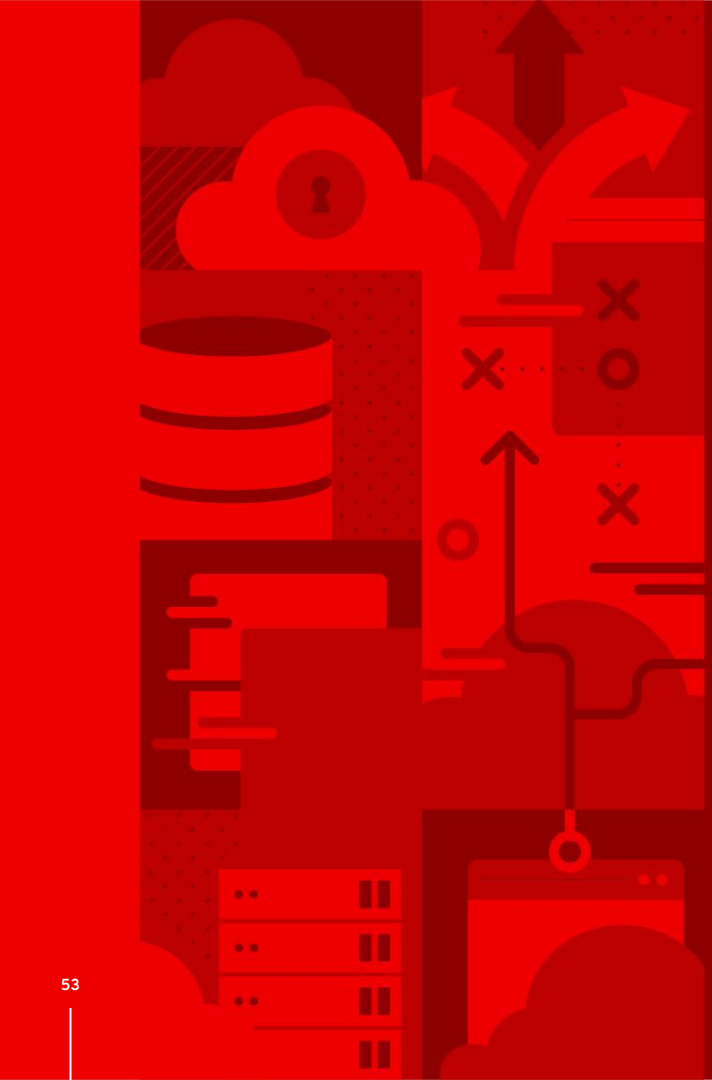
Red Hat

# Network Policy Demo

Demo session



Red Hat



# Protect Internal Traffic



Red Hat

# Zero-trust environments


- Assumes all interactions start as untrusted
- Users can access only explicitly allowed resources
- Encryption is mandatory for all communications
- Client applications must verify server authenticity

# Zero-trust environments

## OpenShift & Zero-Trust

- Encrypts network traffic between nodes and control plane
- Prevents external access to internal traffic
- Stronger security than Kubernetes (which lacks default encryption)
- Application-level verification & encryption not enforced by default

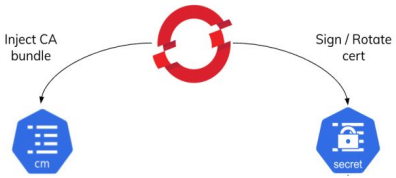
## Certificate Authority (CA) Role

- Trusted CA must sign certificates for encryption
- Applications can verify authenticity using CA-signed certs

# Strategies to encrypt internal traffic

## Server Certificates (service-ca controller)

- Automatically generates signed certificates for internal services
- Stores certificates in secrets, which applications can mount as volumes
- Clients must trust the service-ca controller's CA for authentication




## CertManager Operator

- Automates certificate issuance and renewal using external or internal CAs
- Supports Let's Encrypt, HashiCorp Vault, and custom CAs
- Works with Kubernetes Ingress, Routes, and service-to-service TLS


Red Hat

# Server Certificates in detail

## Service-CA Controller

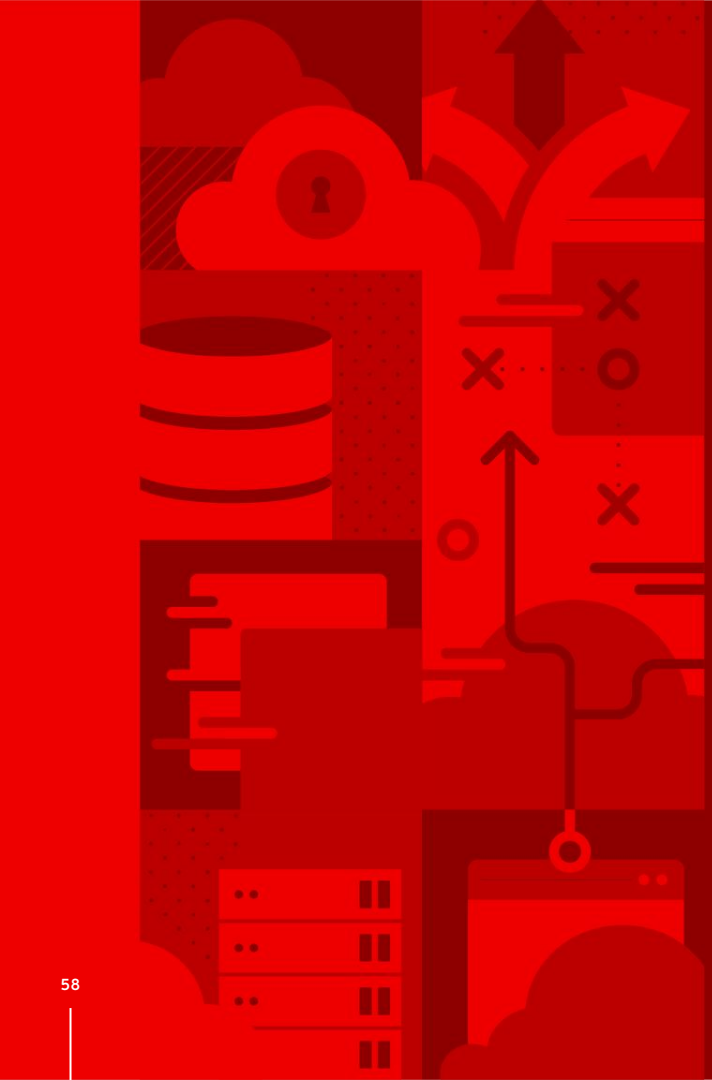
- Generates and signs service certificates for internal traffic
- Creates a secret containing the signed certificate and key


## Using the Certificate

- Deployments can mount the secret as a volume
- Allows services to use the signed certificate for secure communication

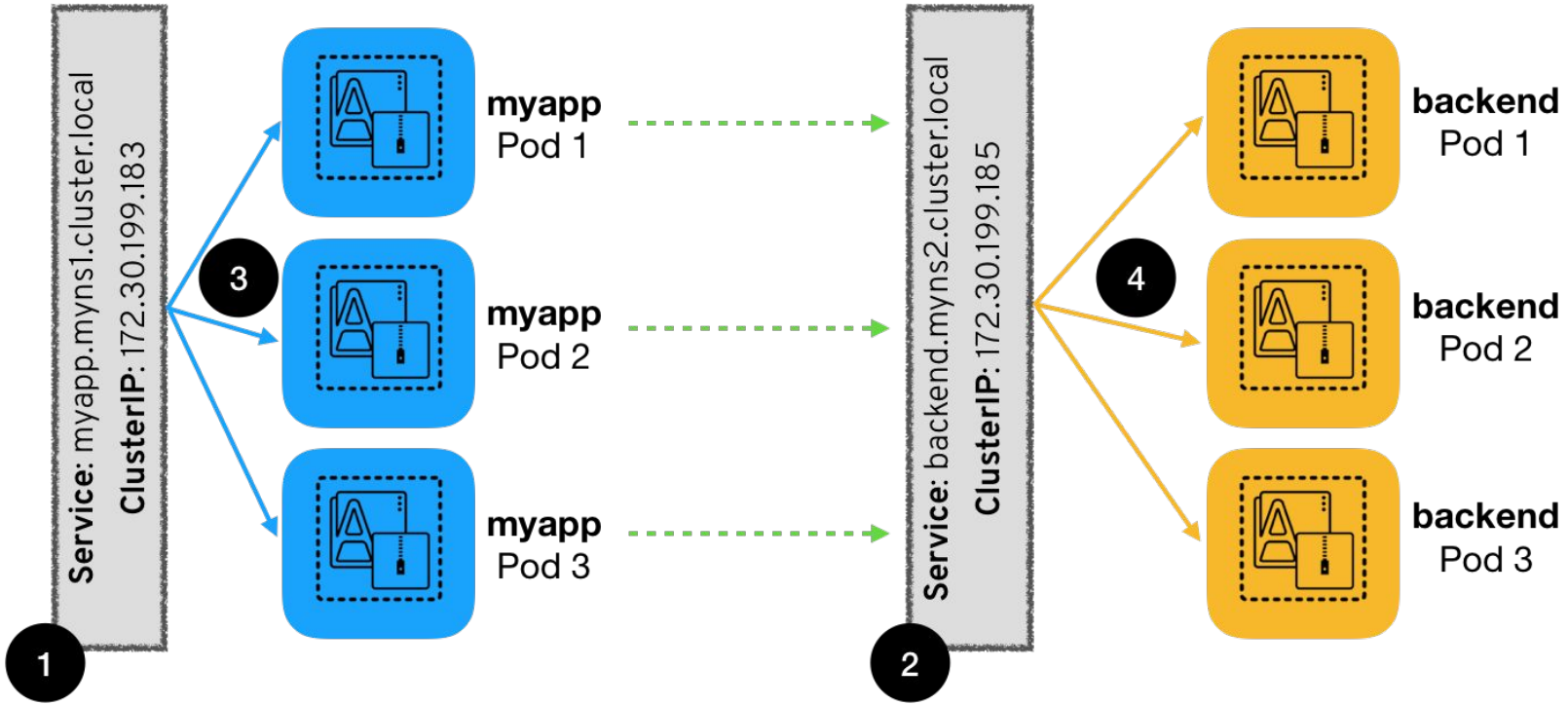
## Client Trust

- Client applications must trust the service-ca controller's CA
- Ensures secure and verified communication between services



# Exposing non-HTTP applications

# Service using ClusterIP



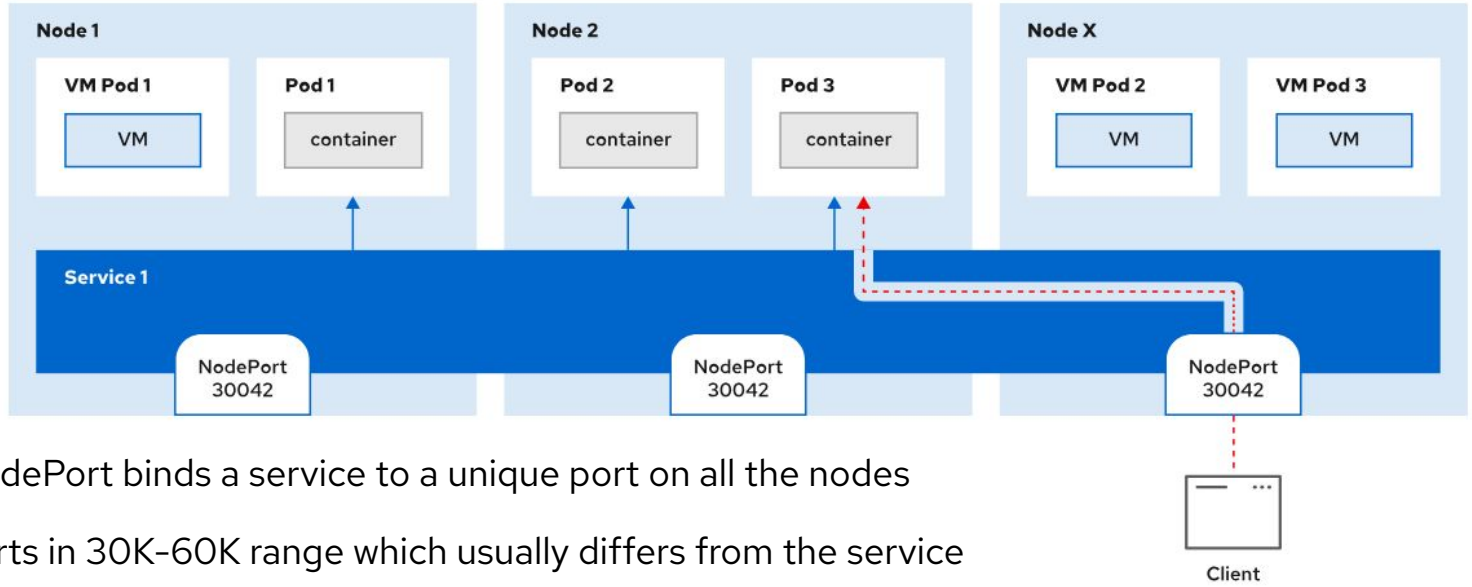

Services of the ClusterIP type provide service access within the cluster.

# Common use cases for NON-HTTP Services

- Database PostgreSQL/MySQL
- Broker MQTT, Kafka, ActiveMQ
- Custom TCP/UDP Services



# External Traffic to a Service using NodePort

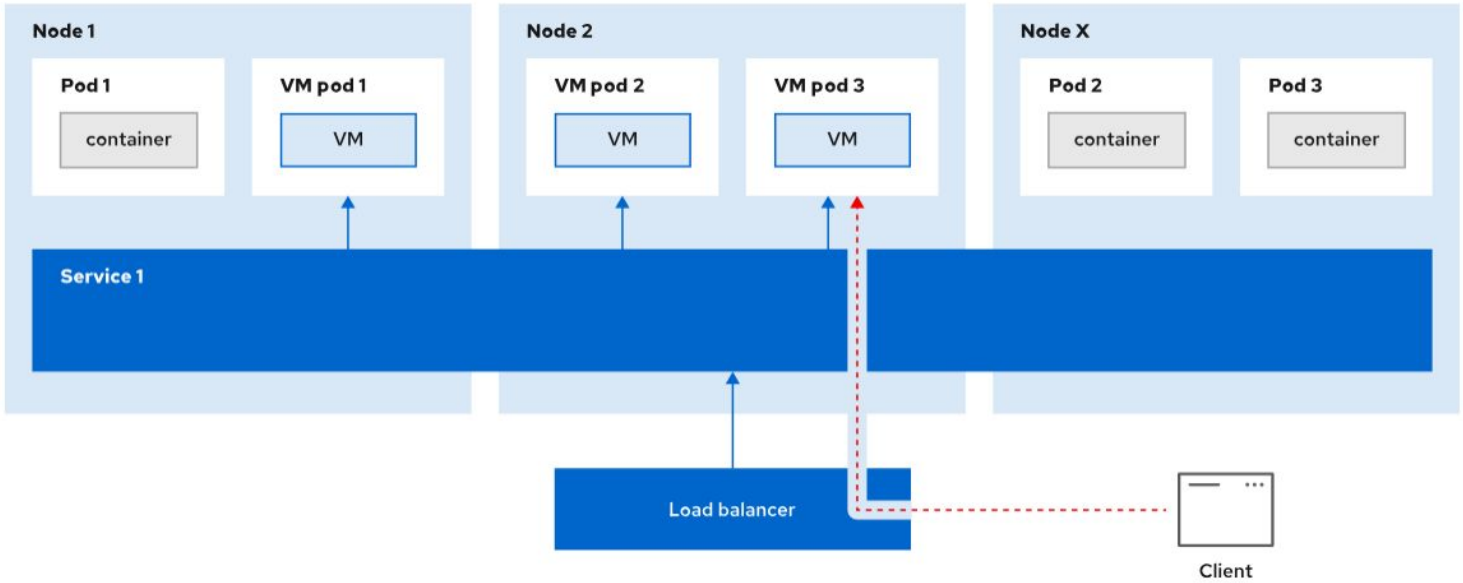



- NodePort binds a service to a unique port on all the nodes
- Ports in 30K-60K range which usually differs from the service
- Firewall rules must allow traffic to all nodes on the specific port

✅ Advantages: Works without external load balancers.

❌ Disadvantages: Requires knowing node IPs and firewall configuration

# External Traffic to a Service using LoadBalancer




- Load balancer services require the use of network features that are not available in all environments.
- Cloud providers typically provide their own load balancer services.
- **MetalLB** is a load balancer component that provides a load balancing service for clusters that do not run on a cloud provider

# External Traffic to a Service using External IP

Assigns a fixed external IP address to a service.

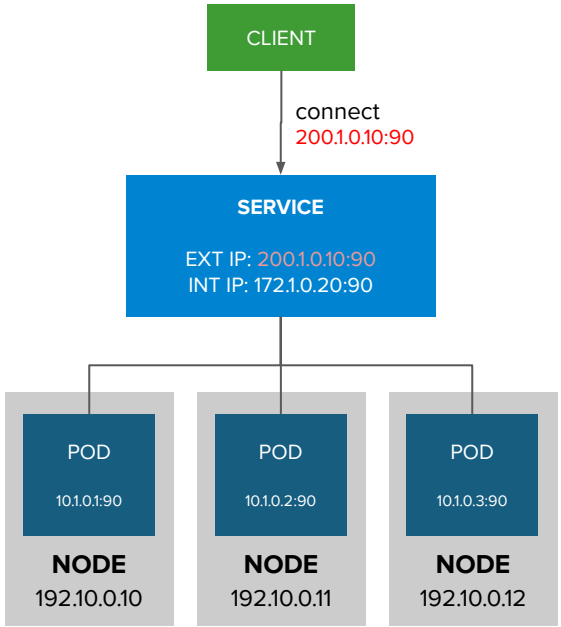
Requires manual network configuration to route traffic to the correct node.

✅ Advantages

- Useful for integrating with on-premise or legacy networks using external LBs.

❌ Disadvantages

- OpenShift does not manage external IPs automatically.



# MetalLB Operator

- Can be used to load balance traffic to OpenShift on premises or on cloud
- MetalLB has two modes to announce reachability information for load balancer IP addresses:
  - Layer 2
  - BGP
- Two components:
  - Controller - One per cluster
  - Speaker - Per Node (DaemonSet)
- Uses ARP/NDP to announce the Node: this can create a bottleneck and limit performance

```
apiVersion: v1
kind: Service
metadata:
  name: mylb
spec:
  ports:
  - name: http
    port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: myapp
  type: LoadBalancer
```

# MetalLB BGP Support

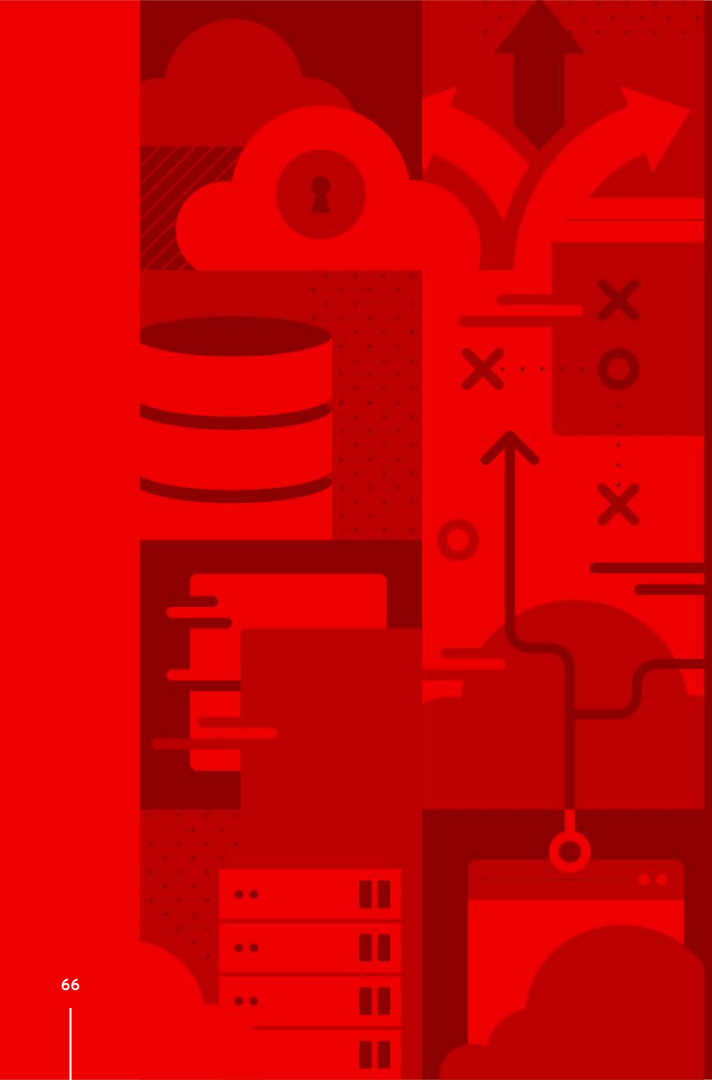
**What is BGP (Border Gateway Protocol)?**
BGP is a routing protocol used to exchange network routes between autonomous systems (AS). It is commonly used in large-scale networks, including the internet.

- MetalLB announces the external IP of a service to the network using BGP.
- Routers recognize the advertised IP and direct traffic to the correct node.
- This allows Kubernetes services to have real public/external IPs without NAT.

```yaml
apiVersion: metallb.io/v1beta1
kind: AddressPool
metadata:
  name: addresspool-sample1
  namespace: metallb-system
spec:
  protocol: bgp
  addresses:
    - 172.18.0.100-172.18.0.255
```

```yaml
apiVersion: metallb.io/v1beta1
kind: BGPPeer
metadata:
  name: peer-sample1
  namespace: metallb-system
spec:
  peerAddress: 10.0.0.1
  peerASN: 64501
  myASN: 64500
  peerPort: 179
  holdTime: "180s"
  keepaliveTime: "180s"
  password: "test"
```

Red Hat



# Advanced Networking

# Advanced Networking

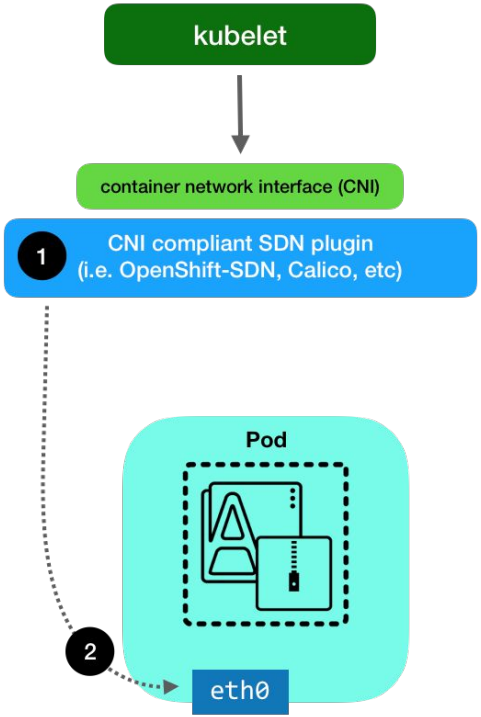
## 🔌 Multus-CNI: Network Interface Management in Pods

- **What is it?** A CNI (Container Network Interface) plugin that allows multiple network interfaces to be assigned to a single Pod.
- **Purpose**: Enable multi-homing in Pods, meaning a Pod can be connected to multiple networks simultaneously.
- **Where does it operate?** At the Pod level, assigning multiple interfaces and configuring the respective networks.

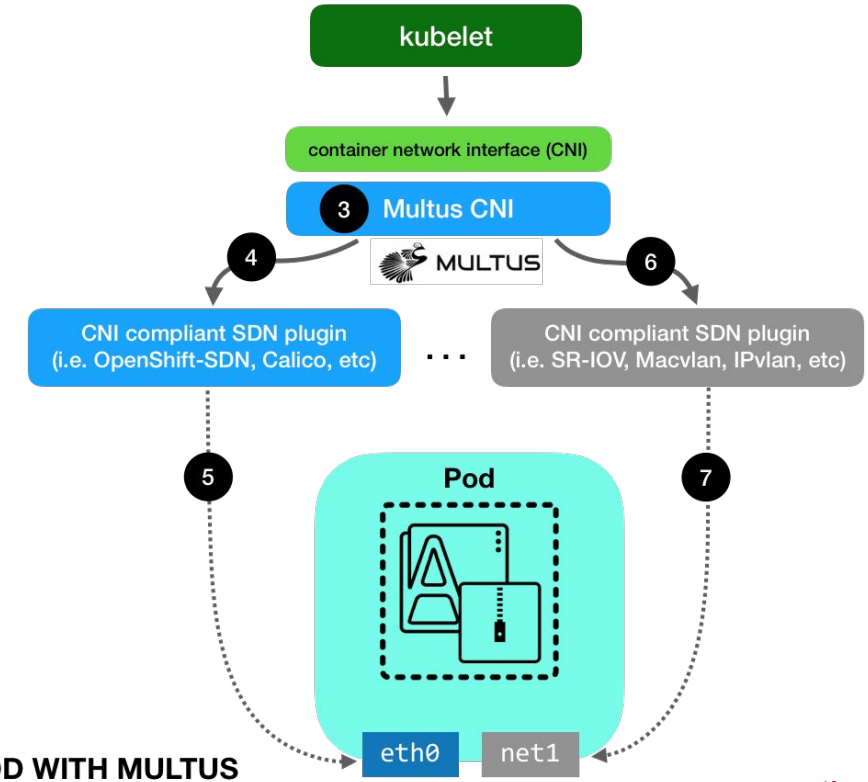
## ⚙️ NMState Operator: Node-Level Network Configuration

- **What is it?** An operator that allows for the declarative management of cluster node network configurations.
- **Purpose**: To configure the network of physical or virtual nodes in the OpenShift infrastructure.
- **Where does it operate?** At the node level, configuring NIC, VLAN, bonding, bridge, and other network settings of the host operating system.

# Multus




POD WITHOUT MULTUS




POD WITH MULTUS

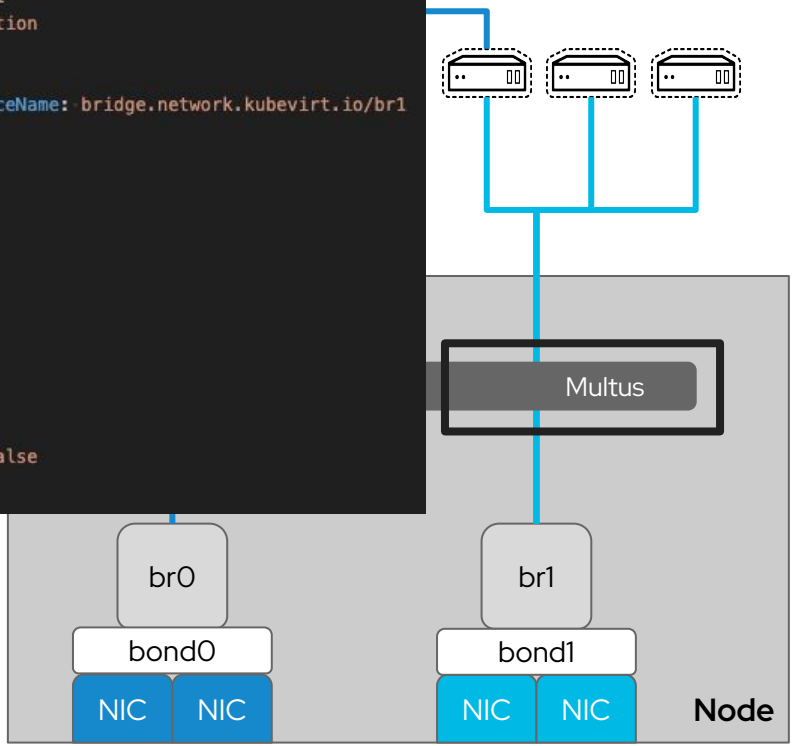


Red Hat

# Network Attachment Definition

- A Network Attachment Definition allows to  to define additional network interfaces for pods or virtual machines
- Common types:
  - Host device
  - Bridge
  - IPVLAN
  - MACVLAN

```yaml
apiVersion: k8s.cni.cncf.io/v1
kind: NetworkAttachmentDefinition
metadata:
  annotations:
    k8s.v1.cni.cncf.io/resourceName: bridge.network.kubevirt.io/br1
  name: vlan-93
  namespace: default
spec:
  config: >-
    {
      "name":"vlan-93"
      "type":"cnv-bridge",
      "cniVersion":"0.3.1",
      "bridge":"br1",
      "vlan":93,"
      macspoofchk":true,
      "ipam":{},
      "preserveDefaultVlan":false
    }
```






Red Hat

# Adding the NAD to Workloads

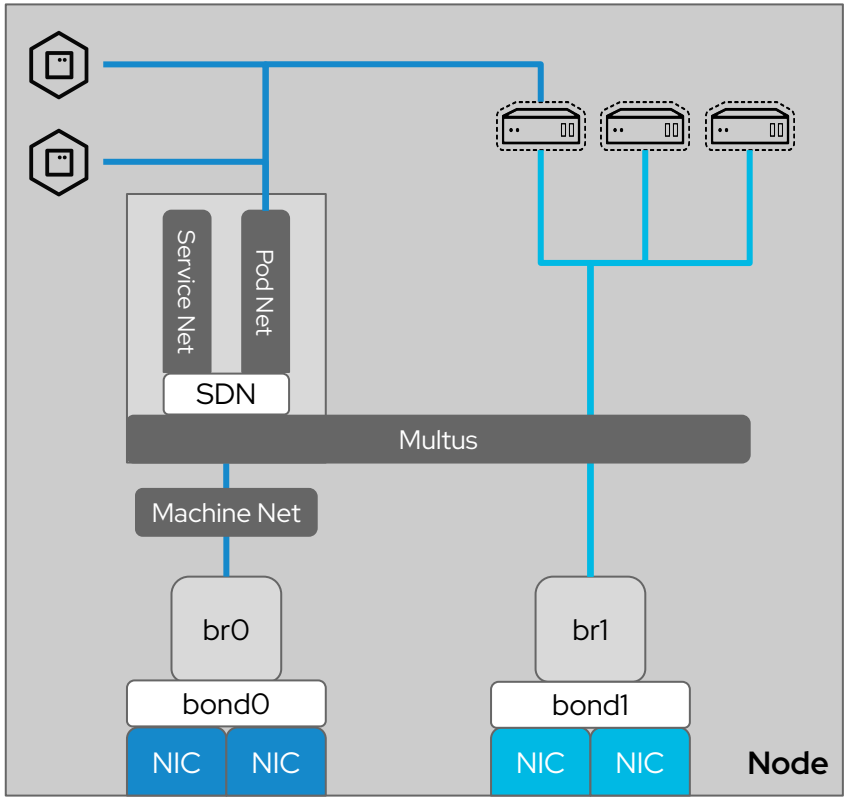
- When the cluster has additional networks, you can add the k8s.v1.cni.cncf.io/networks annotation to the pod's template to use one of the additional networks.

```
apiVersion: apps/v1
kind: Deployment
metadata:
name: example
namespace: example
spec:
  selector:
    matchLabels:
     app: example
     name: example
  template:
    metadata:
      annotations:
        k8s.v1.cni.cncf.io/networks: example
```

# Advanced Scenarios: NMState Operator

You can use the NMState Operator to manage more complex use cases:

- Dynamic Updates: NMState can dynamically update network configurations without requiring a restart of the network interfaces or the pods
- VLAN Bonding: Allows state-driven configuration using VLAN Bonding to improve network redundancy and performance.
- https://github.com/nmstate/kubernetes-nmstate

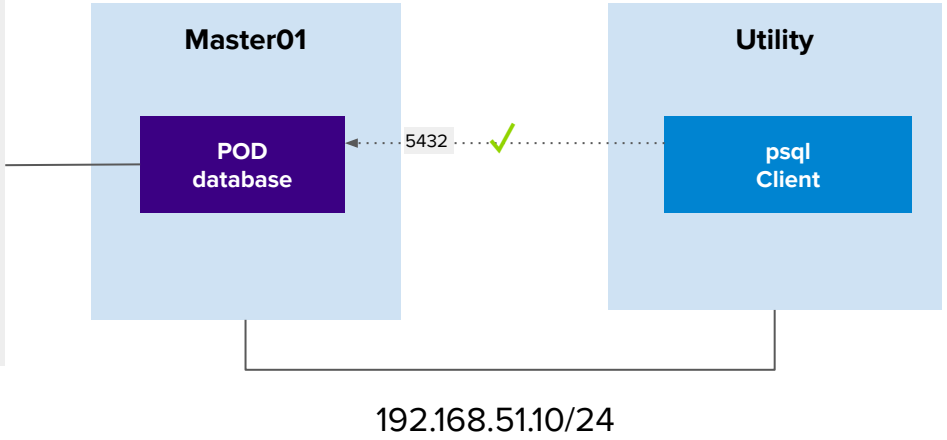




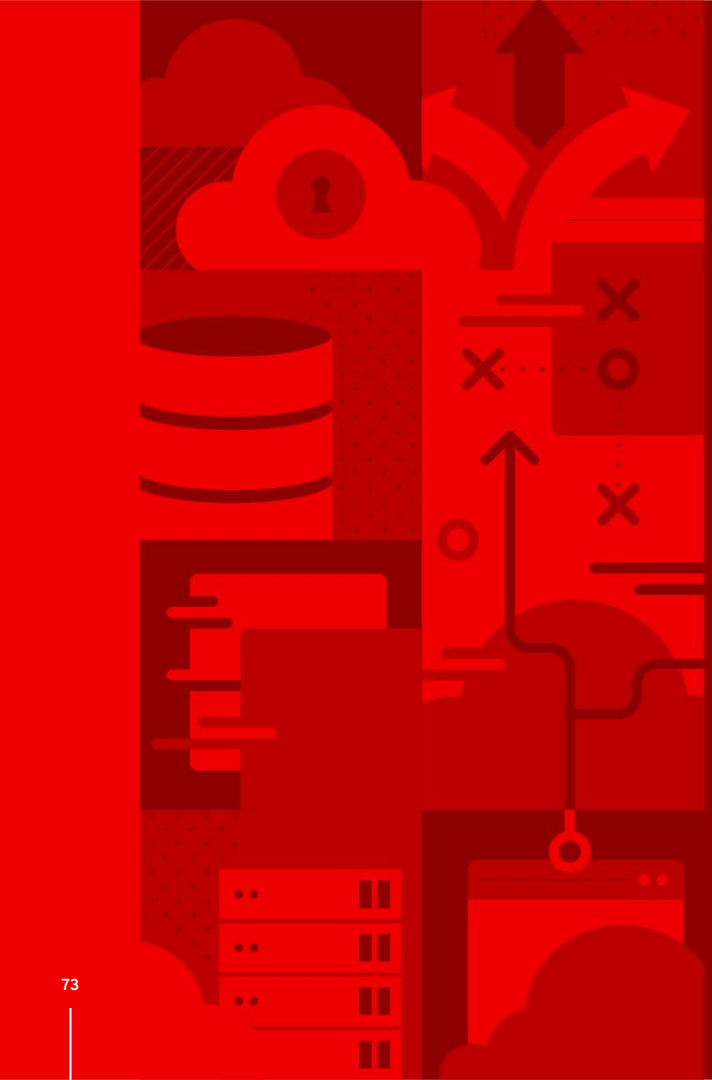

Red Hat

# Multus - lab  non-http-multus

```
apiVersion: apps/v1
kind: Deployment
metadata:
name: database
namespace: database
spec:
  selector:
    matchLabels:
     app: database
     name: database
  template:
    metadata:
      annotations:
        k8s.v1.cni.cncf.io/networks: custom
```




Make a PostgreSQL database accessible outside the cluster on an isolated network by using an existing node network interface



# Resource Quotas and Limits

# Limiting Workloads

📋 **Requests** → Affect Scheduling

Define the minimum guaranteed CPU & memory for a pod.

- The scheduler ensures the node has enough resources to meet requests.
- Pods stay in Pending if requests cannot be satisfied.

🚧 **Limits** → Affect Runtime & Killing Behavior

Define the maximum CPU & memory a pod can use.

- Exceeding CPU limit → Pod is throttled (not killed).
- Exceeding memory limit → Pod is OOMKilled (Out of Memory Killed).

# Comparing Requests with Limits

```
apiVersion: v1
kind: Pod
metadata:
  name: example-pod
spec:
  containers:
     - name: app
     image: my-app:latest
     resources:
     requests:
          cpu: "250m"     # Minimum guaranteed CPU (0.25 cores)
          memory: "256Mi" # Minimum guaranteed memory
     limits:
          cpu: "500m"     # Max CPU (can be throttled)
          memory: "512Mi" # Max memory (exceeding kills pod)
```

# Resource Quotas

A **ResourceQuota** is a policy applied at the **namespace level** to limit aggregate resource usage (like CPU, memory, and objects) across all resources in that namespace.

- 📌 Purpose: Prevent overuse of cluster resources by enforcing team-level limits.
- ✋ Enforces usages of Resource and Limits
- 🔗 Works with LimitRange for default and max/min per container

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: example-quota
spec:
  hard:
    pods: "5"
    requests.cpu: "2"
    requests.memory: 4Gi
```

# Per-Project Resource Constraints: Limit Ranges

## 🧭 What is a **LimitRange**?

- Defines default resource Requests & Limits for pods/containers in a specific project (namespace).
- Ensures fair resource usage within a project.
- Prevents pods from over-consuming CPU & memory.

## 🛠️ How It Works:

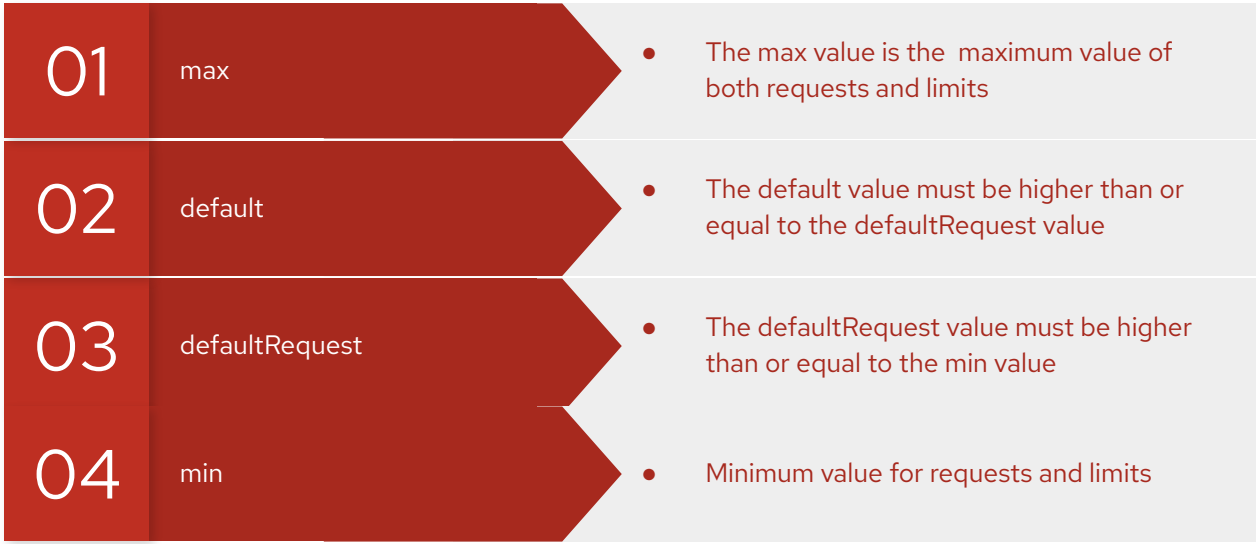
- If a pod/container does not specify requests/limits, the default values from LimitRange apply.
- Enforces max/min constraints per container.

# Limit Types

- **Default limit:** Use the default key to specify default **limits** for workloads.
- **Default request:** Use the defaultRequest key to specify default **requests** for workloads.
- **Maximum**: Use the max key to specify the maximum value of both requests and limits.
- **Minimum**: Use the min key to specify the minimum value of both requests and limits.
- **Limit-to-request ratio**: Ratio between Limit-to-request

# Limit Types

- Define your limits and requests according to the following order:



# 🎯 Checkpoint on Limits/Limits ranges

- To make sure you have understood, answer this question:
- Given the following LimitRange (right) will you be able to schedule the Pod (left) with limits?

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: example-pod
spec:
  containers:
      - name: app
      image: my-app:latest
      resources:
      requests:
        cpu: "250m"
        memory: "256Mi"
      limits:
        cpu: "1"
        memory: "512Mi"
```

❌ greater than LimitRange.max

```yaml
apiVersion: v1
kind: LimitRange
metadata:
  name: limit-range
spec:
  limits:
    - type: Container
      max:
        cpu: "500m"
        memory: "512Mi"
```

❌ This Pod will fail scheduling as cpu: 1 > LimitRange.max.cpu: 500m.

Red Hat

# OpenShift Bootstrap Template

- A tool/script to bootstrap standardized OpenShift projects.
- Provides a repeatable template that integrates best practices.
- Helps ensure consistency in project configurations across your organization.

Why Use It?

- Enables pre-configuration of resources such as build configs, image streams, and CI/CD pipelines.
- Rapid project setup with minimal manual intervention.

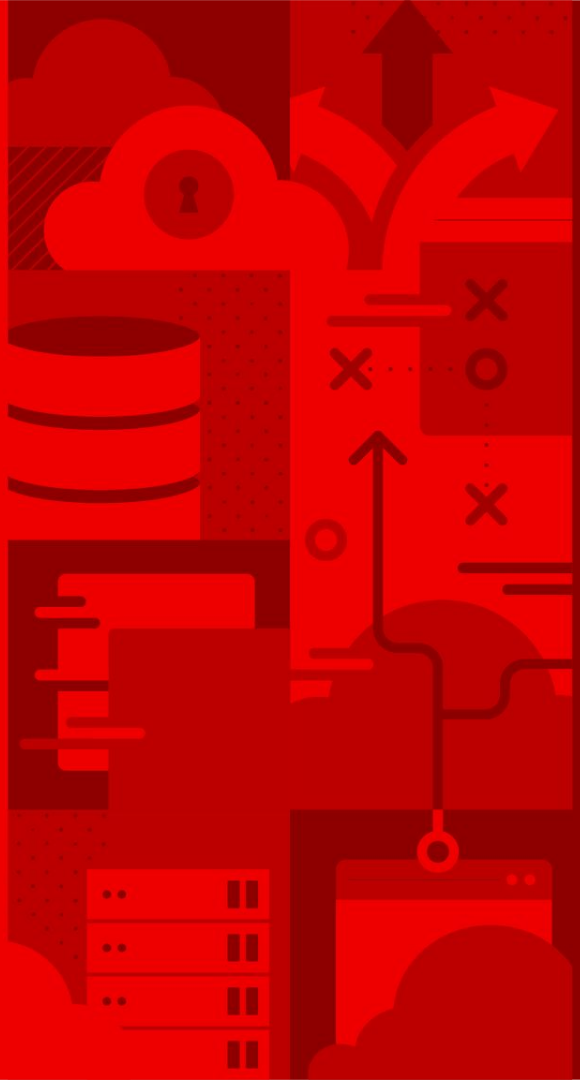
# OpenShift Bootstrap Template: Steps

- **Export** a Project Template

```
oc adm create-bootstrap-project-template -o yaml >template.yaml
```

- **Edit** the Template
- **Create** the Template in the openshift-config namespace
- **Insert** the Template in the Project resource definition

```
apiVersion: config.openshift.io/v1
kind: Project
metadata:
...output omitted...
name: cluster
...output omitted...
spec:
  projectRequestTemplate:
  name: project-request
```



# OpenShift Operators

# OpenShift Operators

OpenShift Operators are Kubernetes-native applications that automate the deployment, management, and lifecycle of complex software on OpenShift. They extend Kubernetes capabilities by encapsulating operational knowledge into code.

📌 Key Points:

- Automate installation, updates, and maintenance of applications.
- Use Kubernetes Custom Resource Definitions (CRDs) to manage applications declaratively.
- Reduce manual intervention by handling scaling, self-healing, and monitoring.
- Improve reliability and consistency across OpenShift clusters.
- Available via Operator Hub, supporting both Red Hat-certified and community operators.

# Operators as a First-Class Citizen

*Operator Developer creates:*
Off-Cluster Metadata (Bundle)

Operator Deployment
RBAC
API Dependencies
Update Path
Metadata

+

Custom Resource Types

=

YourOperator v1.1.2

*Cluster user sees:*
On-Cluster Resources (CSV + CRD)

`ClusterServiceVersion` → OPERATOR LIFECYCLE MANAGER

+

`CustomResourceDefinitions`

*Manages creation of:*

`Deployment`
`Role`
`ClusterRole`
`RoleBinding`
`ClusterRoleBinding`
`ServiceAccount`
`CustomResourceDefinition`

## Cluster Operators

- Manage the OpenShift control plane and cluster infrastructure
- Cluster functionality exposed via Operator APIs
- 40+ Operators, managed by Cluster Version Operator

MachineConfig

ImageRegistry

CloudCredentials

WebConsole

Ingress

## Workload Operators / Add-Ons

- Extend OpenShift with additional applications and services.
- Installed from OperatorHub on demand.
- Managed by **OLM**

Red Hat Quay

OpenShift GitOps

OpenShift Pipelines

CrunchyDB Postgres

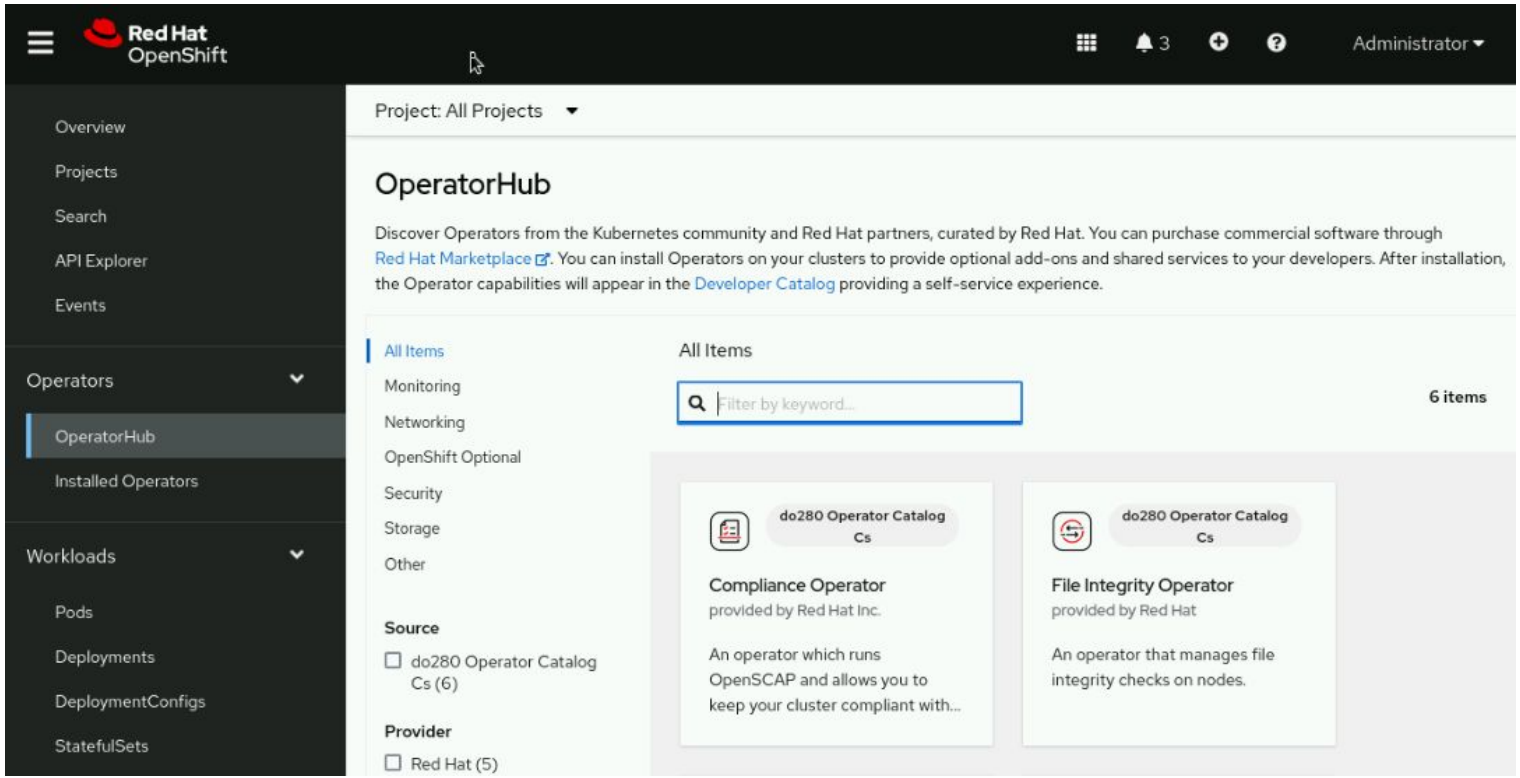
GitLabs Runner



Red Hat

# Installing Operators through the Hub

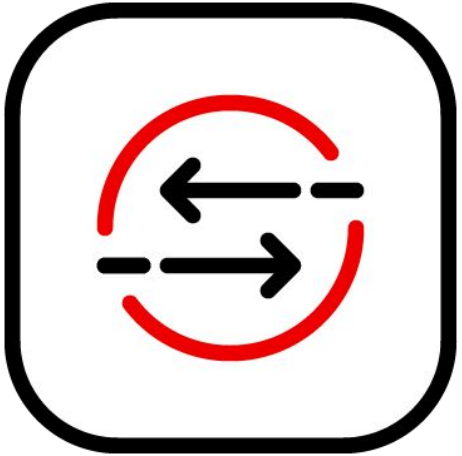
You can install or update Operators through the OpenShift Operator Hub:



# File Integrity Operator

**What is File Integrity Operator?**

- The File Integrity Operator continuously monitors file integrity on OpenShift cluster nodes using AIDE (Advanced Intrusion Detection Environment).
- It deploys a DaemonSet that runs privileged AIDE containers on each node, logging file modifications from the initial scan.
- ✅ Supports RHEL CoreOS nodes only.
- ✅ Works in disconnected environments.
- ✅ Detects unexpected file changes, indicating potential intrusions.



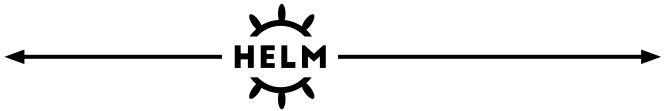
# Subscription Modes

When installing an Operator via the Operator Lifecycle Manager (OLM), you can choose between two subscription modes:

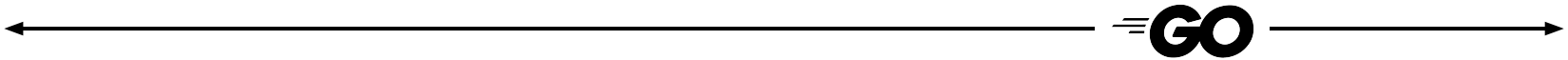
- **Automatic Mode**
  - The Operator updates automatically whenever a new version is available.
  - Ensures the latest features, bug fixes, and security patches are applied.
  - Best for develop/test environments that need continuous updates.
- **Manual Mode**
  - Updates require manual approval before being applied.
  - Provides more control over updates, allowing testing before deployment.
  - Recommended for production environments where stability is a priority.

# Operator Maturity

| Level I | Level II | Level III | Level IV | Level V |
|---|---|---|---|---|
| **Basic install** | **Seamless upgrades** | **Full life cycle** | **Deep insights** | **Auto pilot** |
| Automated application provisioning and configuration management | Patch and minor version upgrades supported | App life cycle, storage life cycle (backup, failure recovery) | Metrics, alerts, log processing, and workload analysis | Horizontal and vertical scaling, auto config tuning, abnormal detection, and scheduling tuning |





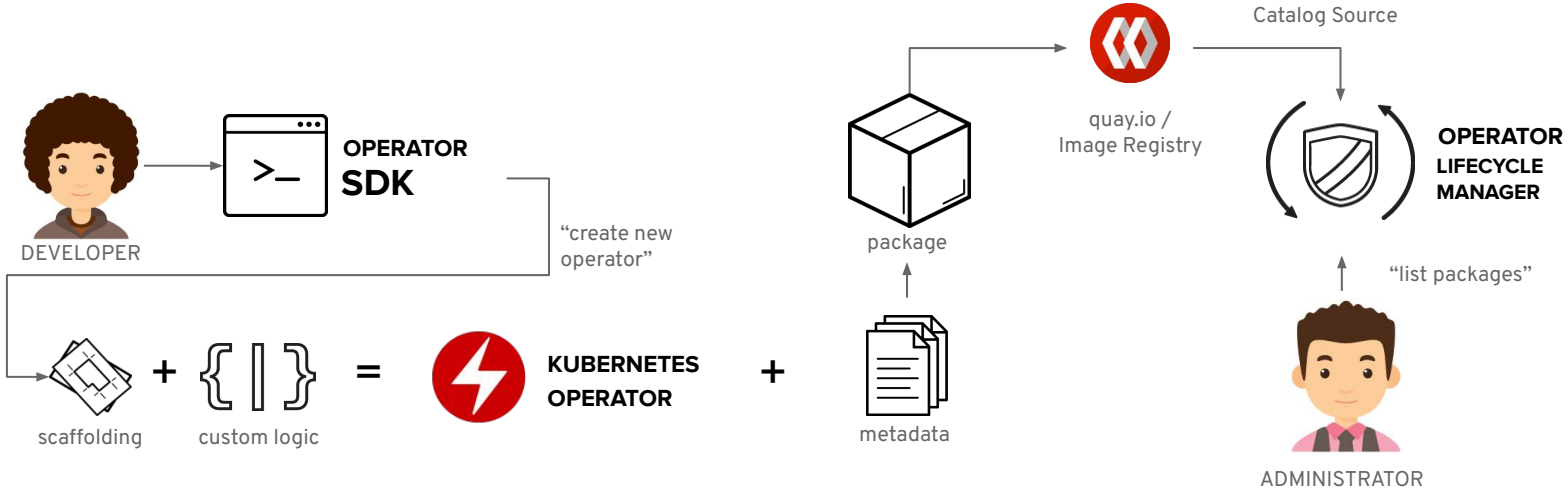



- **Operator SDK** - Allows developers to build, package and test an Operator based on your expertise without requiring all the knowledge of Kubernetes API complexities

- **Operator Lifecycle Manager** - Helps you to deploy, and update, and generally manage the lifecycle of all of the Operators (and their associated services) running across your clusters

- **OperatorHub.io** - Publishing platform for Kubernetes Operators, allows for easy discovery and install of Operators using a graphical user interface

# Operator Framework in Action




Package Discovery:

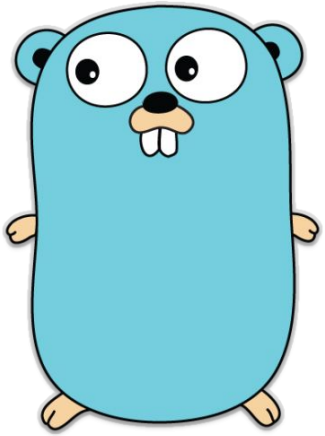
```
$ oc get packagemanifests
```



Red Hat

# Tools to build an Operator

Tools for building Operators: Operator SDK, Java, Go Programming Language, Bash and more!

# Security Context Contraints (scc)

How to configure Security Context Constraint (SCC) that permits elevated permissions

# OpenShift Enforced Security

By default, OpenShift enforces strict security policies to prevent containers from running as the root user. This helps:
✅ Improve cluster security.
✅ Reduce the risk of container breakout attacks.
✅ Enforce least-privilege access by default.

**How to Enable Privileged Operations?**
To allow root user access or other privileged actions, you must assign a Security Context Constraint (SCC) that permits elevated permissions

- Without SCC modifications, OpenShift enforces strict security policies to run containers as a non-root user by default.

# Security Context Constraints

**What are SCCs?**

Security Context Constraints (SCCs) in OpenShift define security policies for controlling how pods interact with system resources. They help enforce security best practices by restricting permissions and capabilities.

Why are SCCs Important?

✅ Prevent privilege escalation.
✅ Control access to sensitive resources.
✅ Define pod security policies at a granular level.
✅ Enhance cluster security and compliance.

- SCCs apply only in OpenShift (Kubernetes uses Pod Security Standards instead).

# Red Hat OpenShift Security Context Constraints

## Use them to manage these controls

- Allow administrators to control permissions for pods
- Restricted-v2 SCC
- By default, no containers can run as root
- Admin can grant access to privileged SCC
- Custom SCCs can be created
- Restricted-v2 SCC is more restrictive than Restricted SCC.

```
$ oc describe scc restricted-v2
Name:                                   restricted-v2
Priority:                               <none>
Access:
  Users:                                <none>
  Groups:                               <none>
Settings:
  Allow Privileged:                     false
  Allow Privilege Escalation:           false
  Default Add Capabilities:             <none>
  Required Drop Capabilities:           ALL
  Allowed Capabilities:                 NET_BIND_SERVICE
  Allowed Seccomp Profiles:             runtime/default
  Allowed Volume Types:                 configMap,csi,downwardAPI,
emptyDir,ephemeral,persistentVolumeClaim,projected,secret
  Allowed Flexvolumes:                  <all>
  Allowed Unsafe Sysctls:               <none>
  Forbidden Sysctls:                    <none>
  Allow Host Network:                   false
  Allow Host Ports:                     false
  Allow Host PID:                       false
  Allow Host IPC:                       false
  Read Only Root Filesystem:            false
  Run As User Strategy: MustRunAsRange
    UID:                                <none>
    UID Range Min:                      <none>
    UID Range Max:                      <none>
  SELinux Context Strategy: MustRunAs
    User:                               <none>
    Role:                               <none>
    Type:                               <none>
    Level:                              <none>
  FSGroup Strategy: MustRunAs
    Ranges:                             <none>
  Supplemental Groups Strategy: RunAsAny
    Ranges:                             <none>
```

1. Lists which users and service accounts the SCC is applied to.
2. Lists which groups the SCC is applied to. For example: system:authenticated



Red Hat

# Configuring SCCs

- Common Security Controls in SCCs:
  - RunAsUser: Defines the user ID (UID) for running containers.
  - SELinuxOptions: Enforces SELinux labels for pods.
  - FSGroup & SupplementalGroups: Controls filesystem permissions.
  - AllowPrivileged: Determines whether a pod can run in privileged mode.
  - AllowedCapabilities & RequiredDropCapabilities: Restricts the use of Linux capabilities.


- Built-in SCCs:
  - restricted (default, most secure)
  - privileged (allows full access, least secure)
  - anyuid, hostaccess, hostmount-anyuid

An example of Custom SecurityContextConstraint:

```
apiVersion: security.openshift.io/v1
kind: SecurityContextConstraints
metadata:
  name: custom-scc
allowPrivilegedContainer: false
runAsUser:
  type: MustRunAs
  uid: 1000
fsGroup:
  type: MustRunAs
  ranges:
     - min: 1000
       max: 2000
supplementalGroups:
  type: RunAsAny
users:
  - system:serviceaccount:myproject:default
```

This custom SecurityContextConstraint can be used to:

- Enforce fixed user/group IDs
- Blocks privileged containers

# Managing SCC

- Viewing Available SCCs

```
oc get scc
```

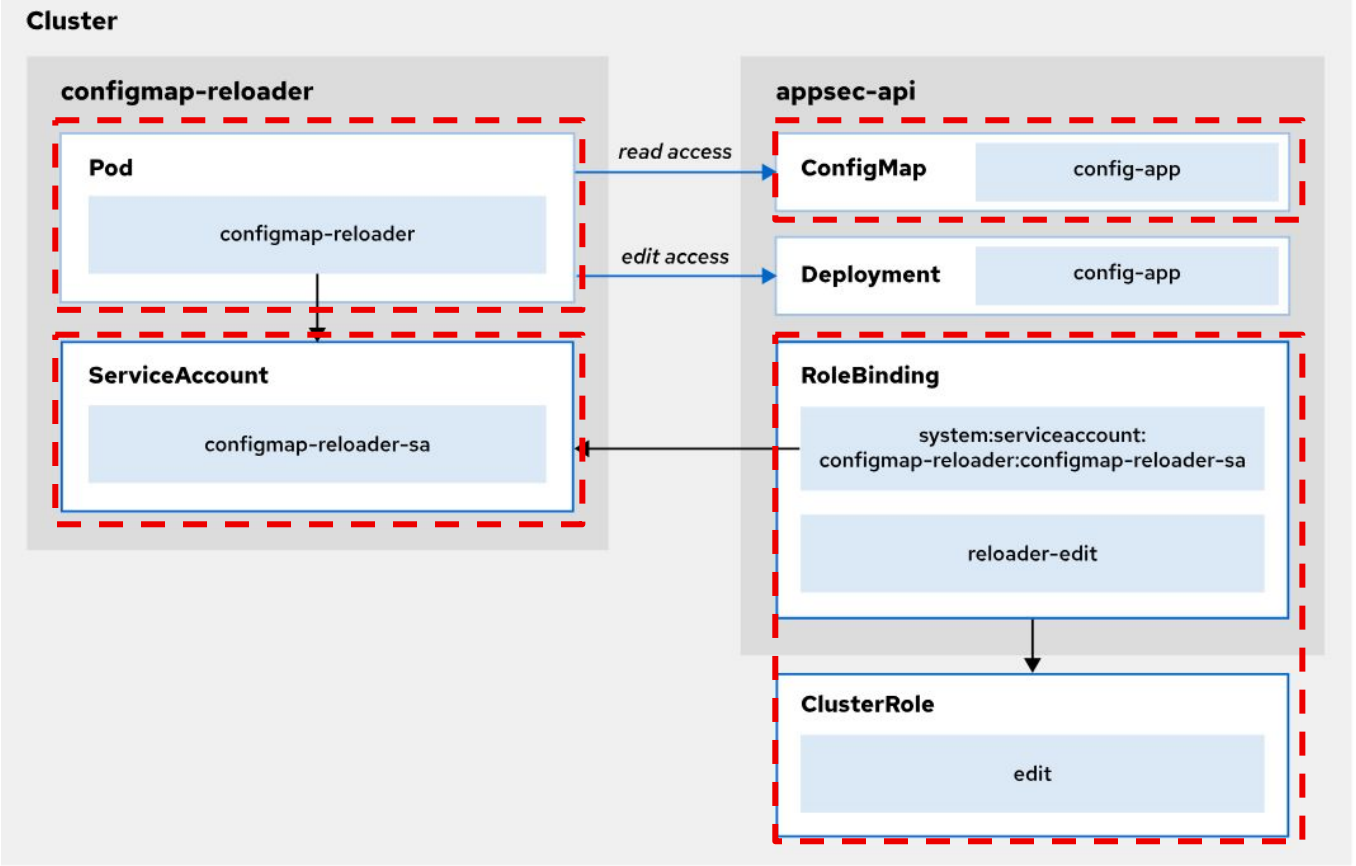
- Assigning an SCC to a Service Account

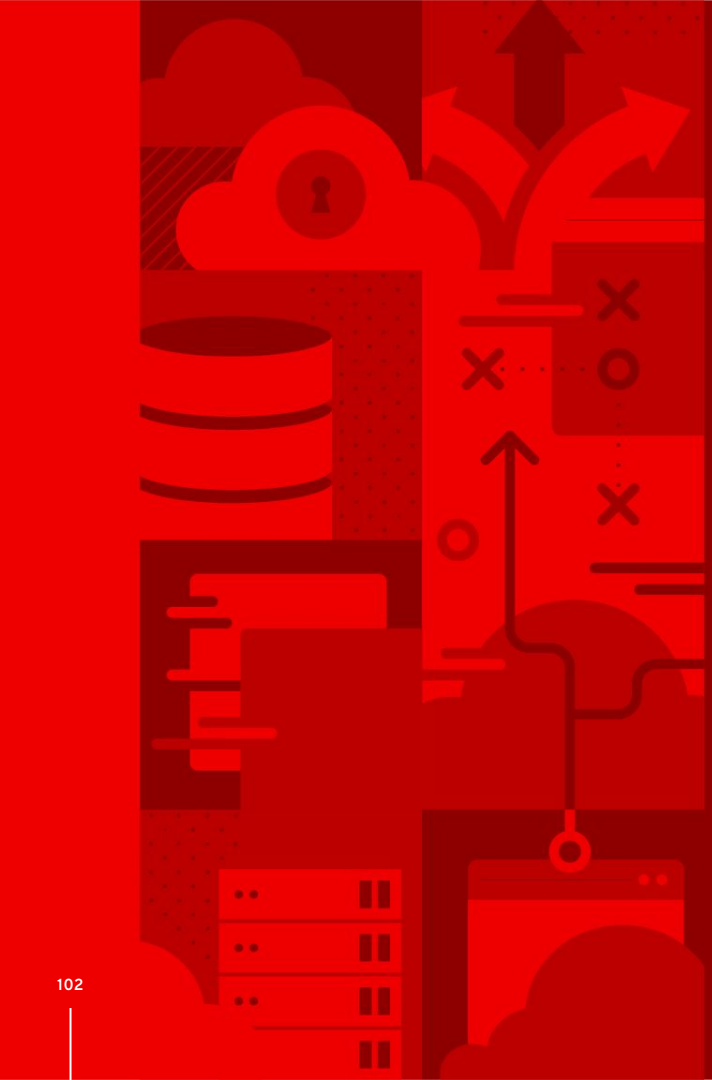
```
oc adm policy add-scc-to-user restricted -z myserviceaccount -n mynamespace
```

- Modify an SCC using YAML:

```
kind: SecurityContextConstraints
metadata:
  name: custom-scc
  allowPrivilegedContainer: false
  runAsUser:
         type: MustRunAsNonRoot
```

# Lab appsec-api



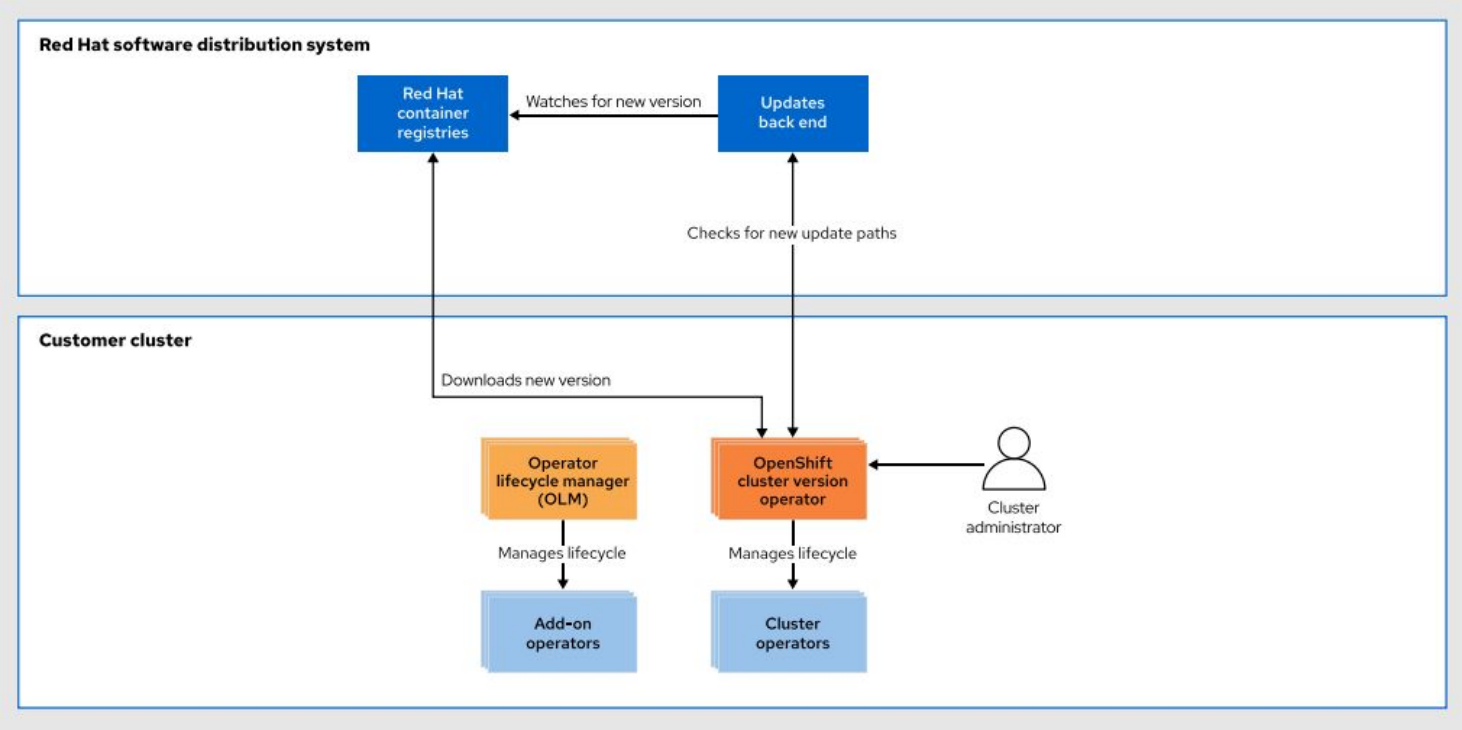


# OpenShift Update



Red Hat

# OpenShift Update



# Why OpenShift Updates Matters

- Keep your clusters aligned with Red Hat support lifecycle (EUS vs non-EUS)
- Critical for compliance (security CVEs, audit readiness)
- Updates improve SRE observability and cluster telemetry fidelity
- Unlock new Kubernetes APIs and features in managed workloads
- Allow compatibility with new versions of Operators and ISVs

# OpenShift Update Checklist

- Have a recent etcd backup in case your update fails
- Have a recent workload backup (Velero/OADP)
- Add-on Operators must be compatible with the new RHOCP version
- Check removed/deprecated API
- Pause Machine Health Checks
- Ensure that all machine config pools (MCPs) are running and not paused

# Questions ?



Red Hat

# Mystery Lab!




Red Hat

# Lab Review OCP 4.14

- Installazione Helm Charts WildFly
- Deploy applicazione Microprofile Config utilizzando Helm Charts
- Customizzazione Configuration con ConfigMap
- Scaling & LimitsRange della applicazione
- Health checks

# Lab Review OCP 4.14

- Crea progetto wildfly-demo

- Installazione Helm Charts WildFly dal repository https://docs.wildfly.org/wildfly-charts/

- Installare questa applicazione da questo Chart [1]: https://github.com/wildfly/quickstart/blob/main/microprofile-config/charts/helm.yaml

  [1] Usare raw.githubusercontent.com

- Verificare che l'applicazione sia disponibile su: https://microprofile-config-app-wildfly-demo.apps.ocp4.example.com/config/value

# Lab Review OCP 4.14

- Crea una ConfigMap con questa chiave/valore:
  CONFIG_PROP="Hello D280!"

- Configura il deployment dell'applicazione in modo da usare questa ConfigMap

- Verificare che l'applicazione restituisca "Hello D280!" invocando https://microprofile-config-app-wildfly-demo.apps.ocp4.example.com/config/value

# Lab Review OCP 4.14

Crea un Limit range per i Pods che abbia:
default:  memory: 512Mi defaultRequest:  memory: 256Mi

Verificare quanti Pods è possibile scalare al massimo con questo LimitRange

Comandi Utili:
oc get event --sort-by=metadata.creationTimestamp

oc adm top nodes

oc scale deployment microprofile-config-app  --replicas X

# Lab Review OCP 4.14

- Installa questo Chart Helm [1] che fornisce uno stato di Health check sulla porta 9990

- Il Servizio di default usa la Porta 8080 - modificare il Servizio e rimappare la porta in modo da poter interrogare il Servizio sul path /health e porta 9990

[1] https://raw.githubusercontent.com/wildfly/quickstart/refs/heads/main/microprofile-health/charts/helm.yaml

# Final Review Labs

Prepare for the final review lab!



Red Hat

# Final Review Labs

## Cluster Self-service Setup

- Create a project template that sets quotas, ranges, and network policies.
- Restrict access to the self-provisioners cluster role.
- Create groups and assign users to groups.
- Use role-based access control (RBAC) to grant permissions to groups.

Estimated time:
60 minutes

# Final Review Labs

## Secure Applications

- Create a limit range.
- Use role-based access control to grant permissions to service accounts and groups.
- Encrypt the traffic end-to-end with TLS by using a signed certificate.
- Restrict cluster internal traffic to pods by using network policies.
- Grant application access to Kubernetes APIs.
- Configure a cluster maintenance application to run periodically.

Estimated time:
40 minutes

# Final Review Labs

## Deploy Packaged Applications

- Deploy an application from a chart.
- Deploy an application from a Kustomize layer.
- Configure an application to connect to the MySQL database

Estimated time:
20 minutes