



Red Hat

Red Hat OpenShift Administration I: Operating a Production Cluster

DO180



Red Hat

Welcome!

Let's meet each other!



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



OpenShift Learning Path

D0480 OpenShift Admin IV
RHACM Quay.io RHACS

D0380 OpenShift Admin III
Advanced Security Backup/Restore Logging GitOps

DO280 OpenShift Admin II
Security Operators Update

DO288 OpenShift Development
Build Develop Automate
Deployments

DO180 Openshift Container Administration I
Admin Deploy Manage Deployments

DO188 Openshift Container Development I
Podman Container Images Compose

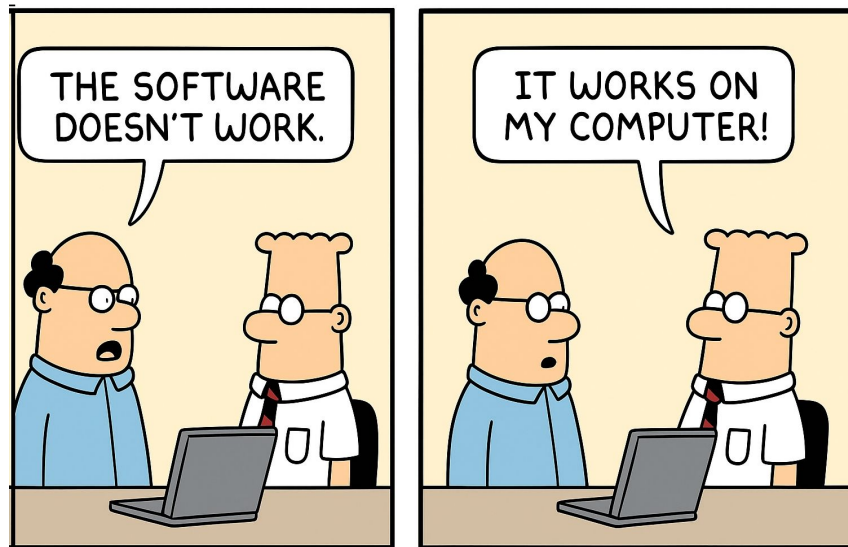




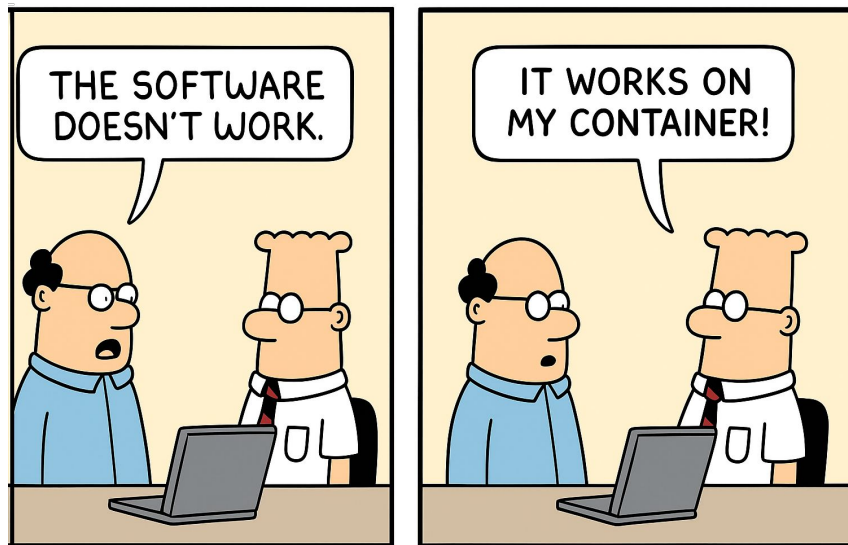
Containers Overview

Capitolo 1

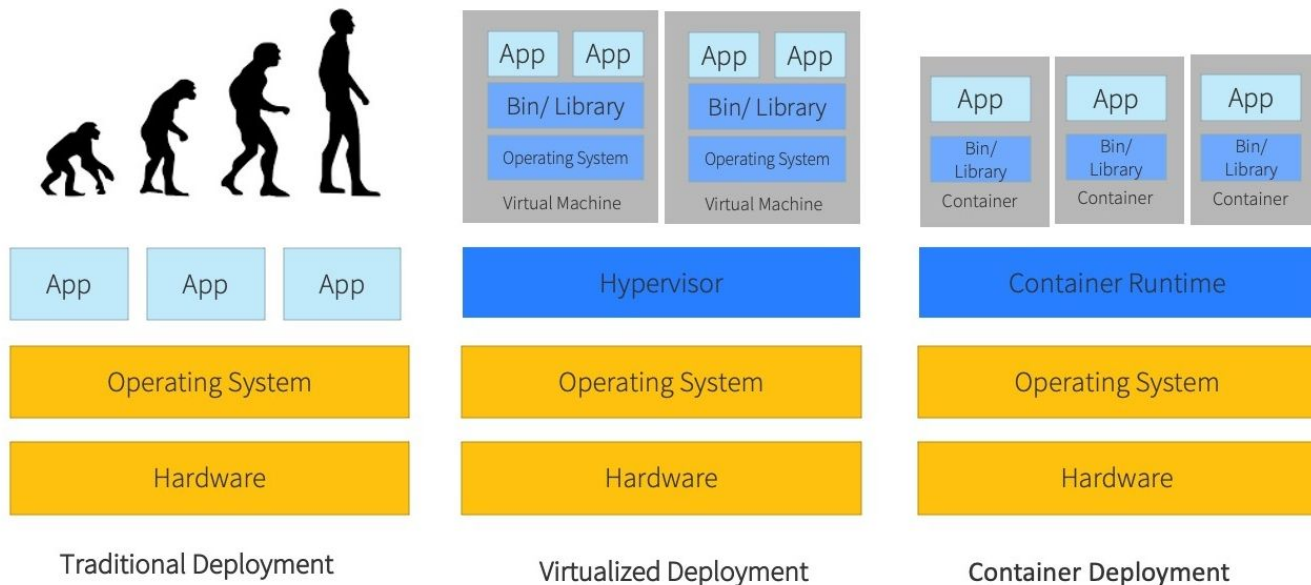
Before DevOps.....



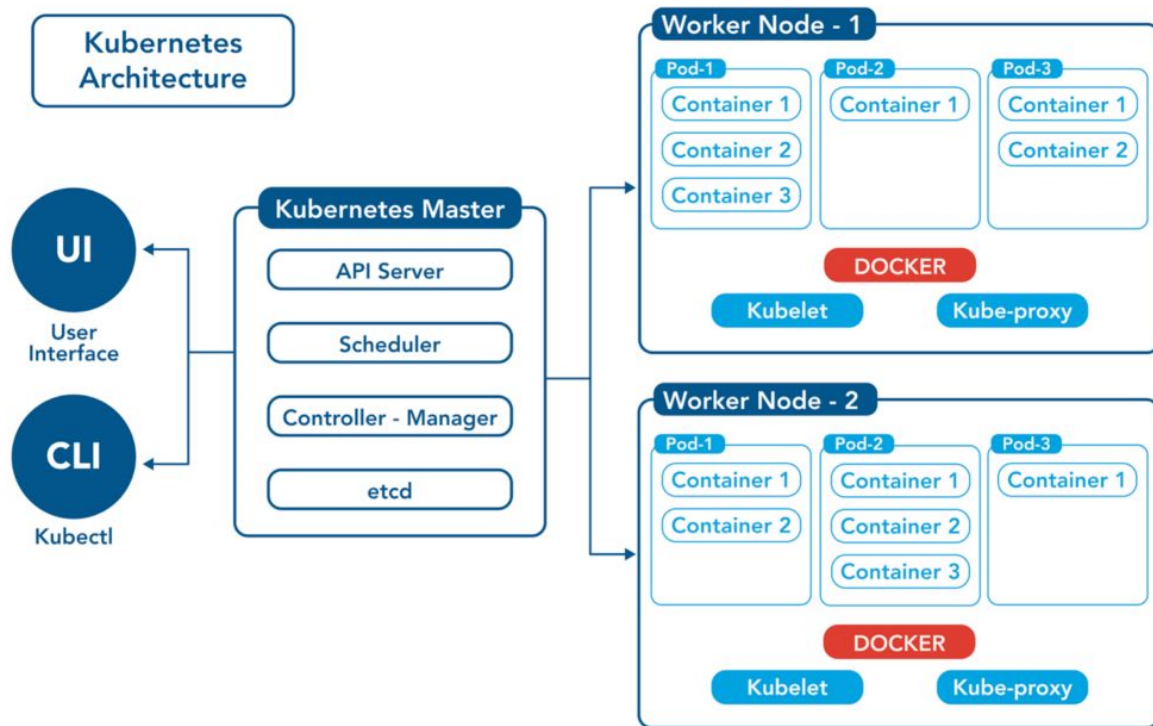
After DevOps.....



Evolution of Application Deployments



Kubernetes Architecture

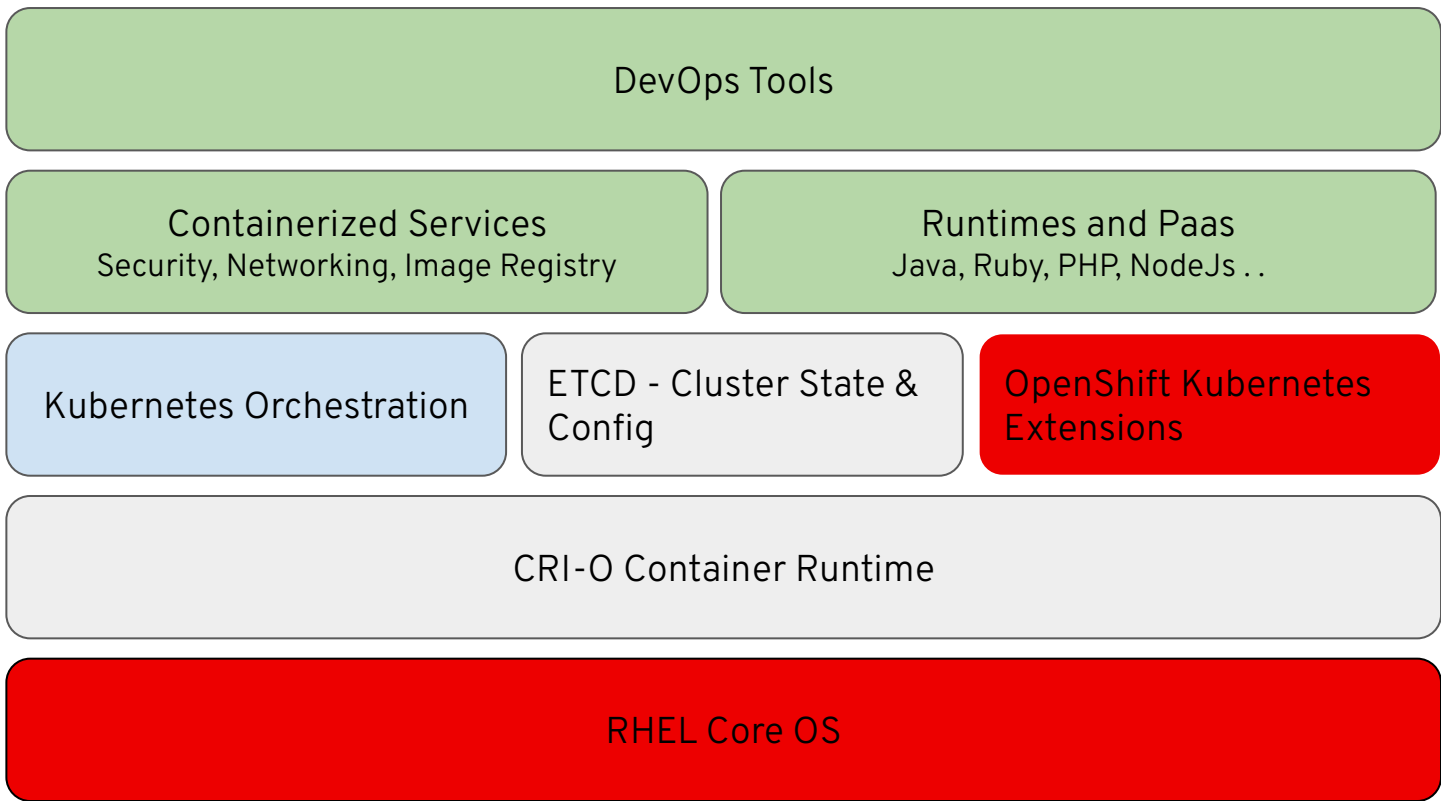




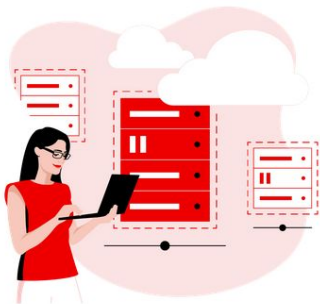
Red Hat OpenShift Components and Editions

Capitolo 1

OpenShift Architecture



OpenShift Offering



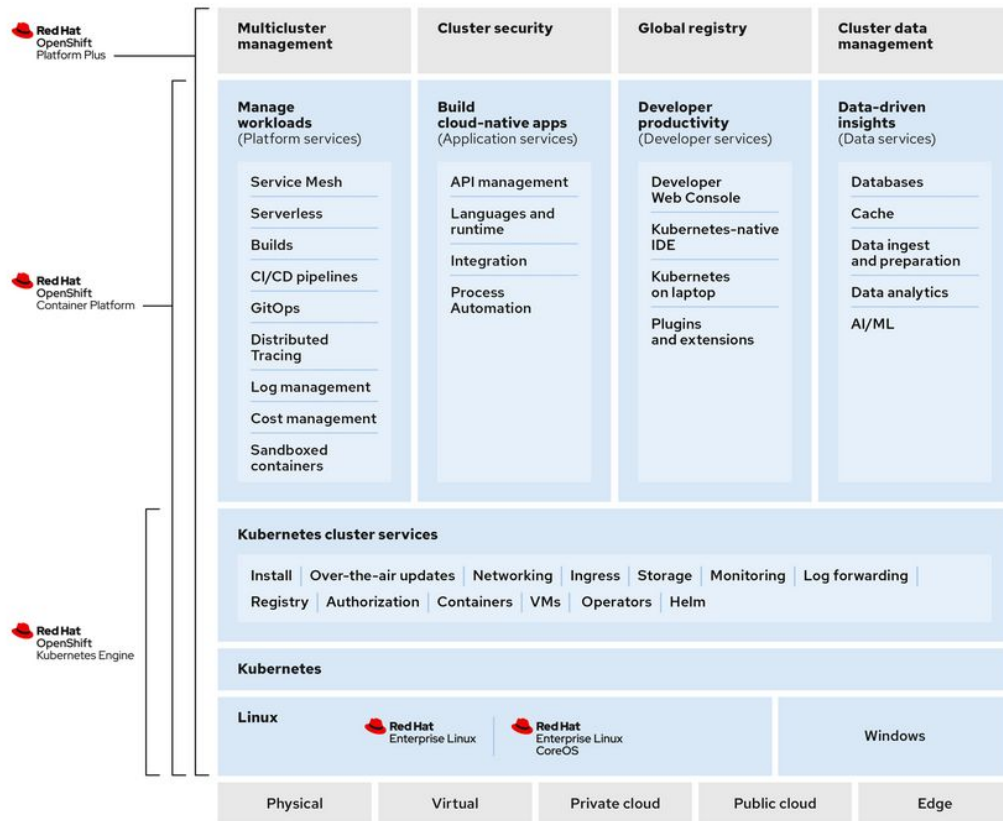
Self-Managed Editions

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

Managed Editions



Comparing OpenShift versions





OpenShift Command Line Interfaces

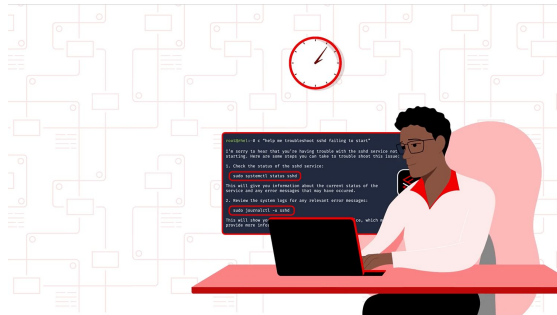
Capitolo 2

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 apply -f <file>.yaml`



kubectl vs oc

kubectl:

- Kubernetes provides the **kubectl** CLI for managing resources, while OpenShift extends it with **oc**, adding extra capabilities for developers and administrators.

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

oc

- **oc** includes all **kubectl** commands, plus OpenShift-specific enhancements.
- **oc** simplifies authentication, project management, and application deployment.
- If working with OpenShift, prefer **oc** for better integration and productivity.

```
oc new-app --image=httpd:1.0
```


oc exclusive commands



Authentication & User Management

- `oc login` – Authenticate to OpenShift with credentials or a token.
- `oc whoami` – Show the current user and token.



Project & Namespace Management

- `oc new-project <name>` – Create a new OpenShift project.
- `oc project <name>` – Switch between projects.



Application Deployment & Builds

- `oc new-app <source>` – Create an app from source, images, or templates.
- `oc start-build <buildconfig>` – Manually trigger a build.

oc exclusive commands

Networking & Routing

- `oc get routes` – List OpenShift routes (external URLs for services).

Image Management

- `oc import-image <image-stream>` – Import external images into an ImageStream.
- `oc tag <source> <destination>` – Manage ImageStream tags, similar to Docker tagging.

Administration

- `oc adm <subcommand>` – Cluster admin commands (e.g., `oc adm policy`, `oc adm prune`).

Kubernetes Resource

```
apiVersion: v1
kind: Pod
metadata:
  name: my-sample-pod
  labels:
    app: my-app
    env: dev
spec:
  containers:
    - name: my-container
      image: nginx:latest
      ports:
        - containerPort: 80
      resources:
        requests:
          memory: "64Mi"
          cpu: "250m"
```

- apiVersion: Identifier of the object schema version
- Kind: Schema identifier.
- Labels: Key-value pairs for categorization and selection
- Annotations: Non-identifying metadata (e.g., version info, links).
- Spec: Identifies the requested configuration for the resource



OpenShift Operators

Capitolo 2

OpenShift Operators

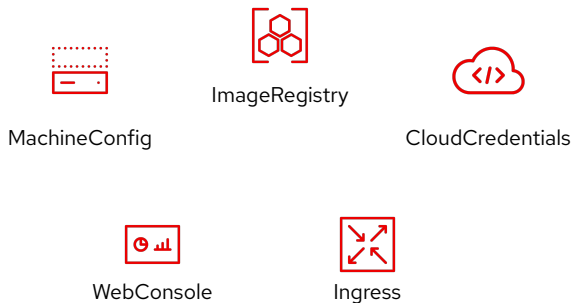
OpenShift Operators are Kubernetes-native applications that automate the deployment, management, and lifecycle of complex software on OpenShift. They extend Kubernetes capabilities by encapsulating operational knowledge into code.

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 OperatorHub, supporting both Red Hat-certified and community operators.

Cluster Operators

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



```
oc get cluster operators
```

Workload Operators / Add-Ons

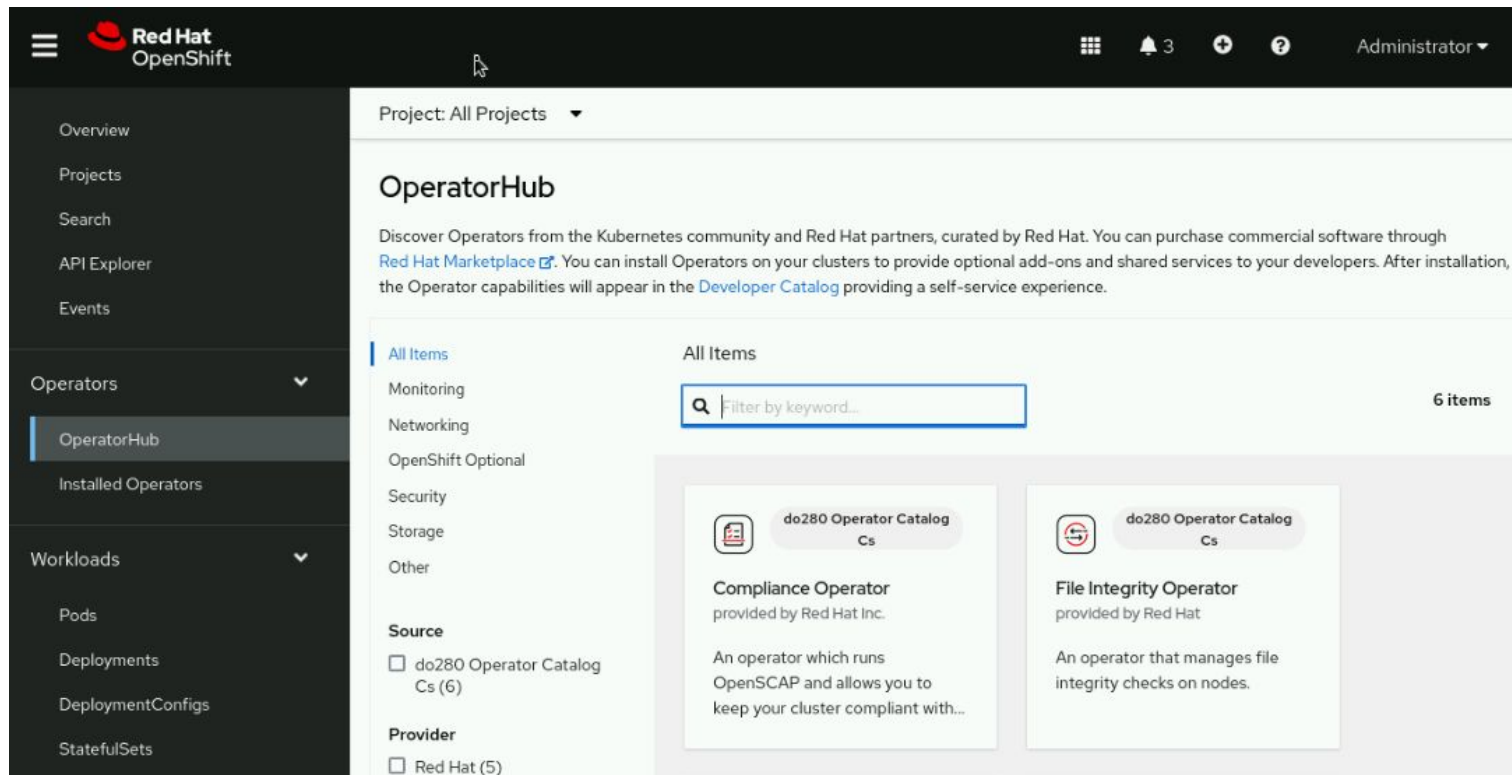
- Extend OpenShift with additional applications and services.
- Installed from OperatorHub on demand.
- Managed by **OLM**



```
oc get operators
```

Installing Operators through the Hub

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



The screenshot displays the OpenShift Operator Hub interface. The top navigation bar includes the Red Hat OpenShift logo, a hamburger menu, and user information (Administrator). The left sidebar contains navigation links: Overview, Projects, Search, API Explorer, Events, Operators (selected), Workloads, Pods, Deployments, DeploymentConfigs, and StatefulSets. The main content area is titled "OperatorHub" and includes a description of the hub's purpose. Below the description, there is a filter section with "All Items" selected and a search bar. A list of operators is displayed, including the "do280 Operator Catalog" and the "Compliance Operator".

Project: All Projects ▼

OperatorHub

Discover Operators from the Kubernetes community and Red Hat partners, curated by Red Hat. You can purchase commercial software through [Red Hat Marketplace](#). You can install Operators on your clusters to provide optional add-ons and shared services to your developers. After installation, the Operator capabilities will appear in the [Developer Catalog](#) providing a self-service experience.

All Items

Monitoring

Networking

OpenShift Optional

Security

Storage

Other

Source


☐ do280 Operator Catalog Cs (6)

Provider

☐ Red Hat (5)


Filter by keyword...

6 items

 do280 Operator Catalog Cs

Compliance Operator
provided by Red Hat Inc.

An operator which runs OpenSCAP and allows you to keep your cluster compliant with...

 do280 Operator Catalog Cs

File Integrity Operator
provided by Red Hat

An operator that manages file integrity checks on nodes.

Subscription Modes

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

◆ Automatic Mode

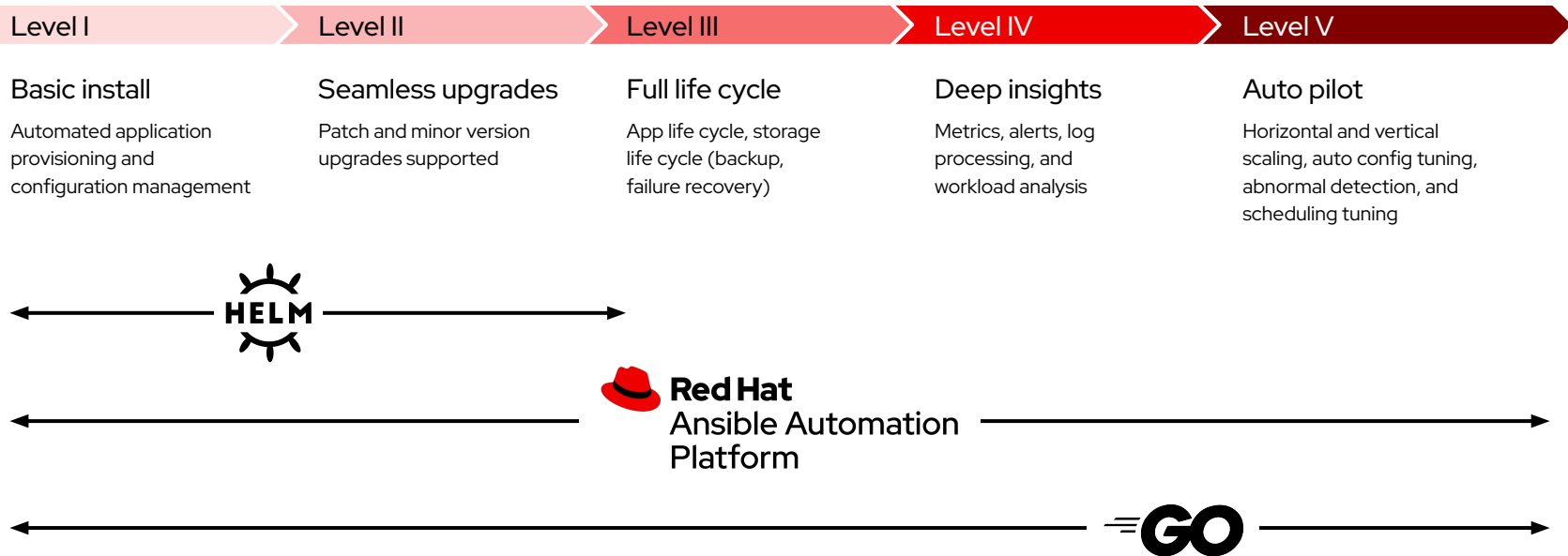
- The Operator updates automatically whenever a new version is available.
- Ensures the latest features, bug fixes, and security patches are applied.
- Best for develop/test environments that need continuous updates.

◆ 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



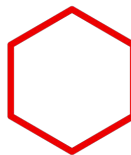
lab start cli-review

An abstract graphic on the left side of the slide, rendered in various shades of red. It features a vertical stack of server racks at the bottom, a cloud with a keyhole icon, a database cylinder, and several curved arrows pointing upwards and outwards, suggesting a flow or process. There are also some 'x' and 'o' symbols scattered within the graphic.

Run Applications as Containers and Pods

Chapter 3

A container is the smallest compute unit



CONTAINER

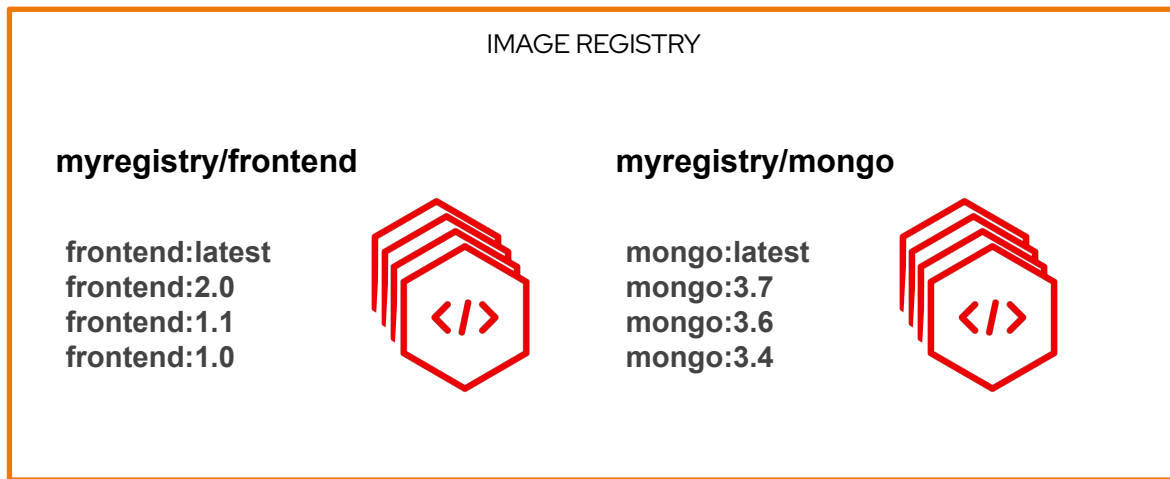
Containers are created from container images



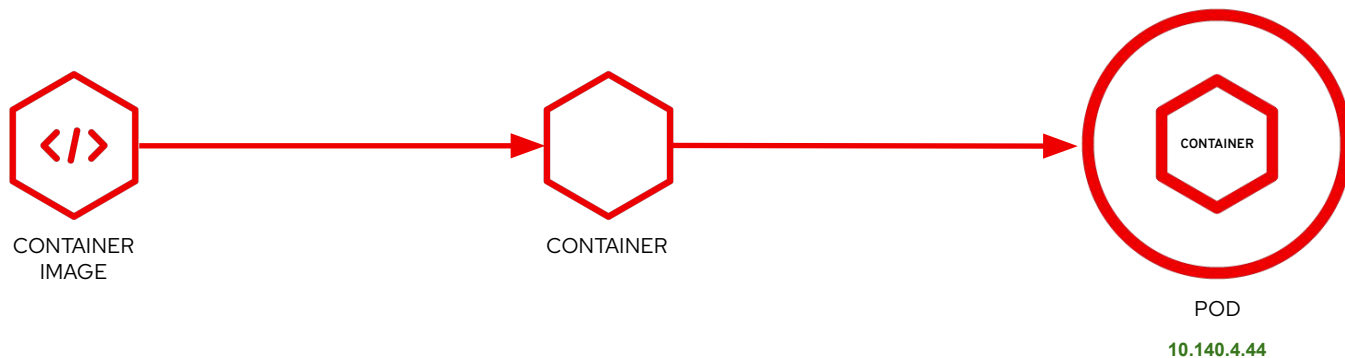
Container images are stored in an image registry



An image repository contains all versions of an image in the image registry



In OpenShift everything runs in pods

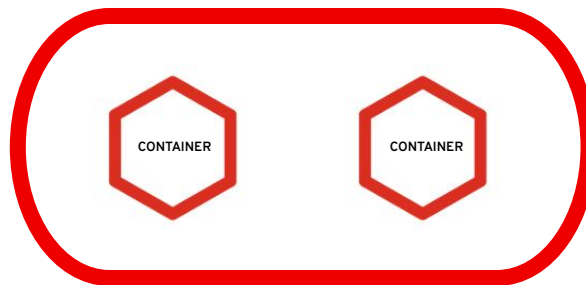


Containers are wrapped in pods which are units of deployment and management



POD

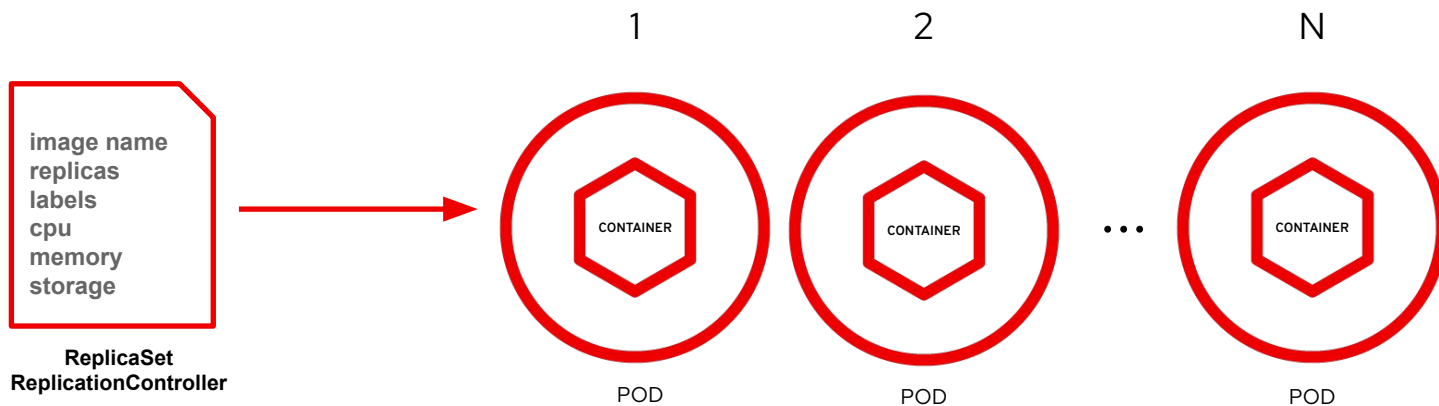
10.140.4.44



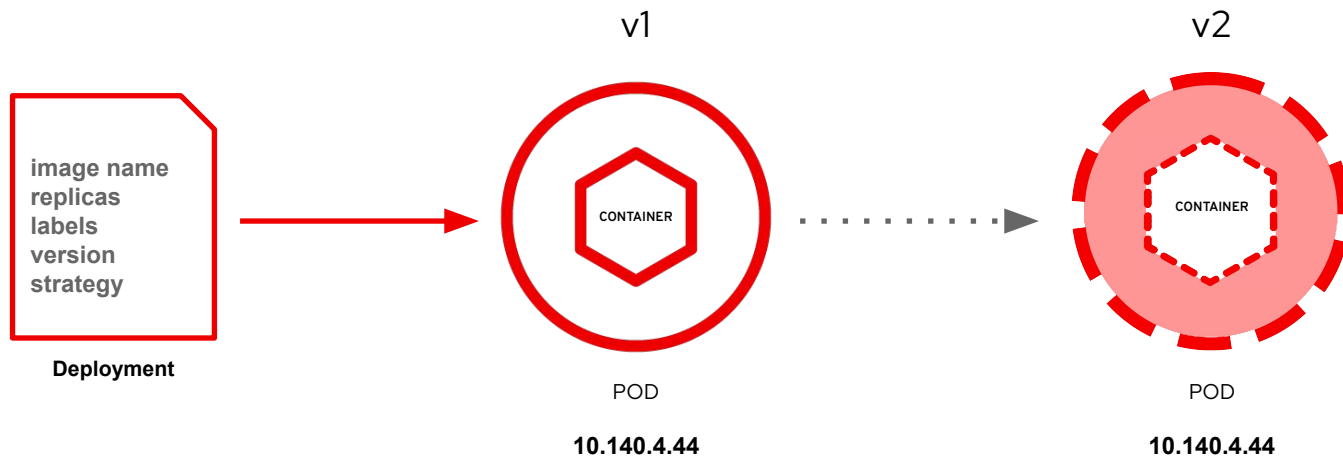
POD

10.15.6.55

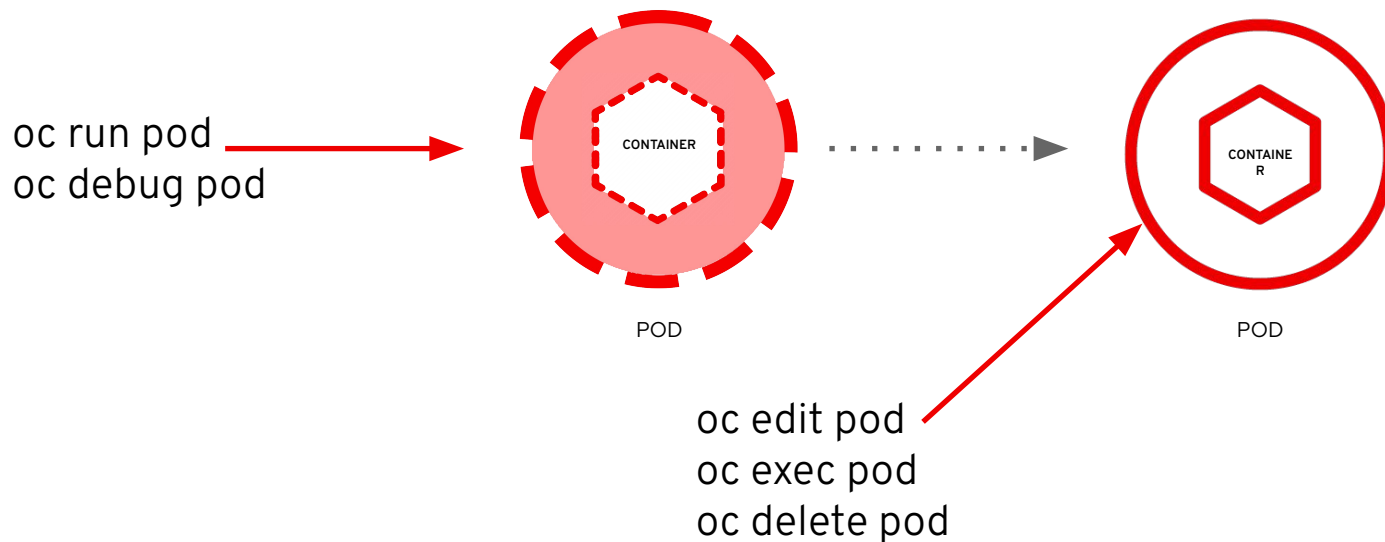
ReplicationControllers & ReplicaSets ensure a specified number of pods are running at any given time



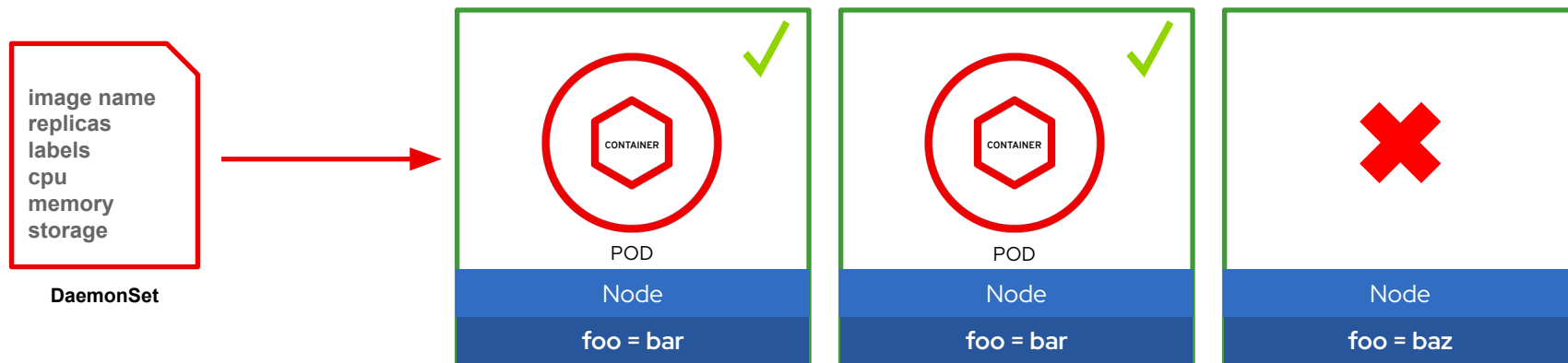
Deployments define how to roll out new versions of Pods



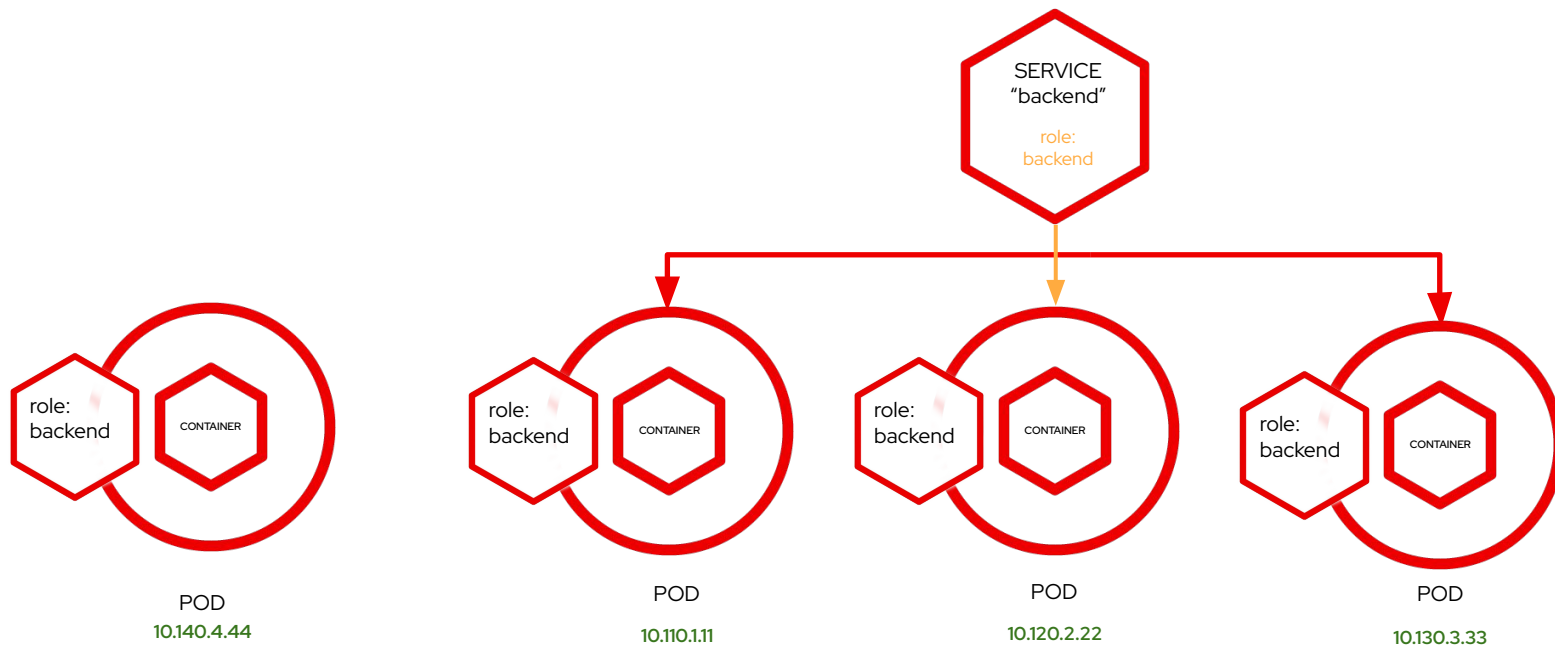
Managing Pods



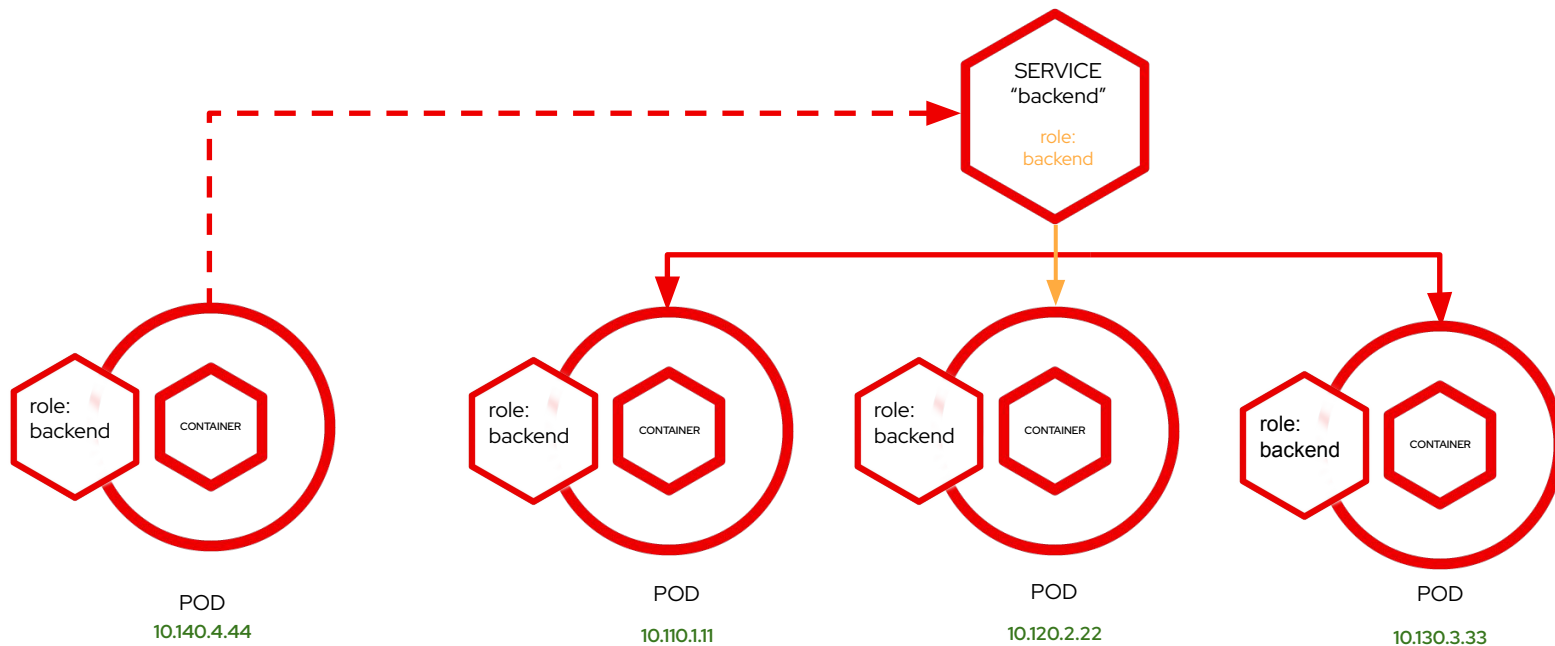
A daemonset ensures that all (or some)
nodes run a copy of a pod



Services provide internal load-balancing and service discovery across pods



Apps can talk to each other via services





Deploy Applications on Kubernetes

Chapter 4



Kubernetes Workloads

Kubernetes provides different workload resources to manage how applications run within the cluster. Each resource is designed for a specific use case, from stateless applications to scheduled jobs.

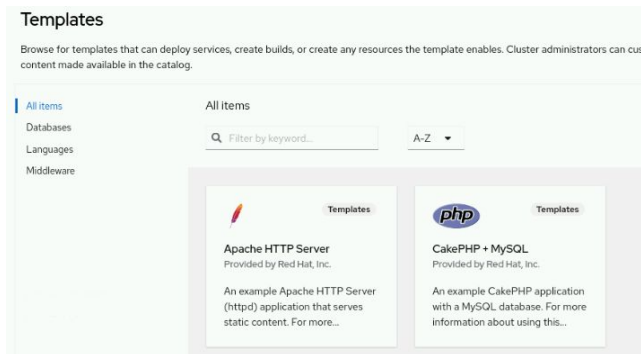
- Deployment ○ Manages stateless applications with rolling updates.
○ Ideal for web applications and APIs.
- StatefulSet ○ Manages stateful applications, ensuring stable identities and persistent storage.
○ Commonly used for databases like PostgreSQL and MongoDB.
- Job/CronJob ○ Runs a task once or at fixed times or intervals.
○ Useful for batch processing, data migrations, or ETL jobs.

OpenShift Templates

In real world projects, several packaging options are available to deploy OpenShift applications. Some of them include:

OpenShift Templates: Reusable JSON/YAML templates that define OpenShift objects with parameters.

- ✓ Built into OpenShift
- ✓ Simple to use and share
- ✓ Can defining multiple related resources
- ✓ Use Parameters to customize the workloads



OpenShift Templates: Commands

Check Templates available in the cluster:

```
oc get templates -n openshift
```

Create a new Template:

```
oc create -f <template-file.yaml> -n <namespace>
```

Process Template Parameters:

```
oc process <template-name> -p PARAM=value
```

Create an application from a Template:

```
oc new-app --template=<template-name> -p PARAM=value
```

Jobs

- Jobs run a Pod to completion (batch task).
- Useful for one-off tasks like database migrations, batch processing, or scripts.
- Once the Job completes successfully, Pods are not restarted.

```
apiVersion: batch/v1
kind: Job
metadata:
  name: simple-job
spec:
  template:
    spec:
      containers:
      - name: simple-task
        image: busybox
        command: ["sh", "-c", "echo Hello OpenShift; sleep 5"]
      restartPolicy: OnFailure
```

CronJobs

- CronJobs run Jobs on a scheduled basis (like cron in Linux).
- Schedule uses cron syntax (* * * * *).
- Each execution creates a Job that runs to completion.
- Useful for periodic tasks, backups, cleanups, or monitoring scripts.

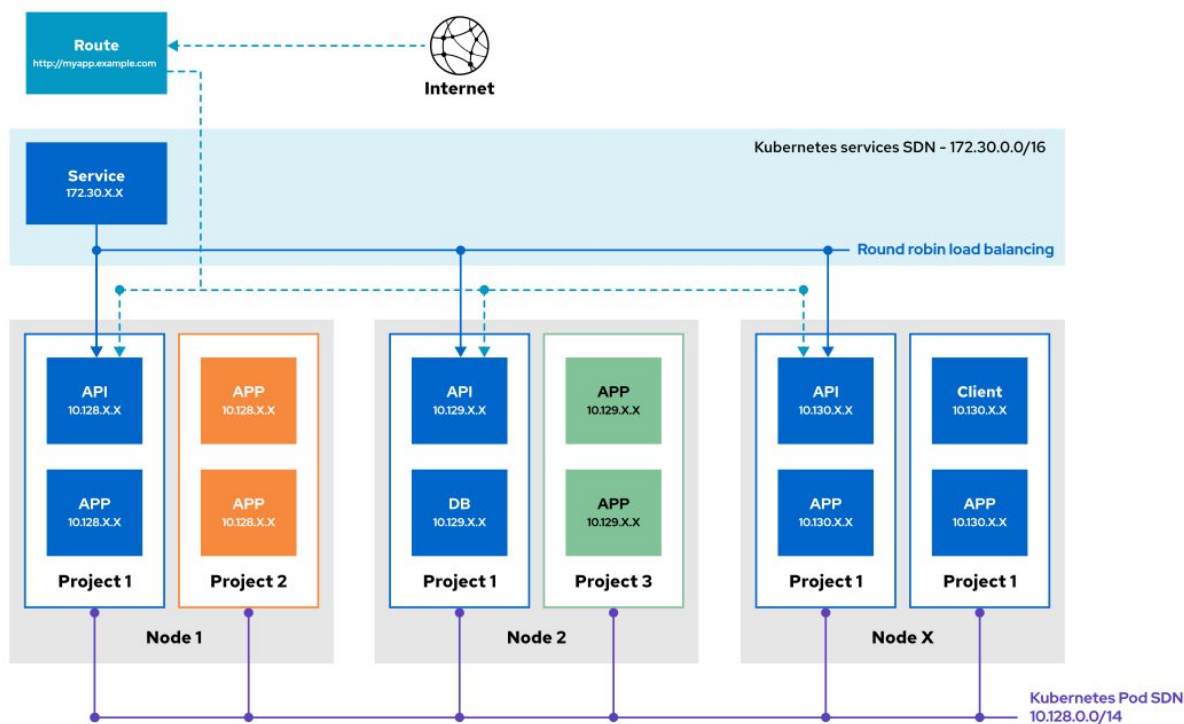
```
apiVersion: batch/v1
kind: CronJob
metadata:
  name: example-cronjob
spec:
  schedule: "0 */6 * * *"      # every 6 hours
  jobTemplate:
    spec:
      template:
        spec:
          containers:
            - name: my-cron-job
              image: busybox
              command: ["sh", "-c", "echo Running CronJob; sleep 10"]
          restartPolicy: OnFailure
```



OpenShift Network

Chapter 4

OpenShift Network: high level picture



OpenShift Network: low level picture

```
$ cat install-config.yaml
[...]  
networking:  
  clusterNetwork:  
    - cidr: 10.128.0.0/14  
      hostPrefix: 23  
  networkType:  
    OpenShiftSDN  
  serviceNetwork:  
    - 172.30.0.0/16  
[...]
```

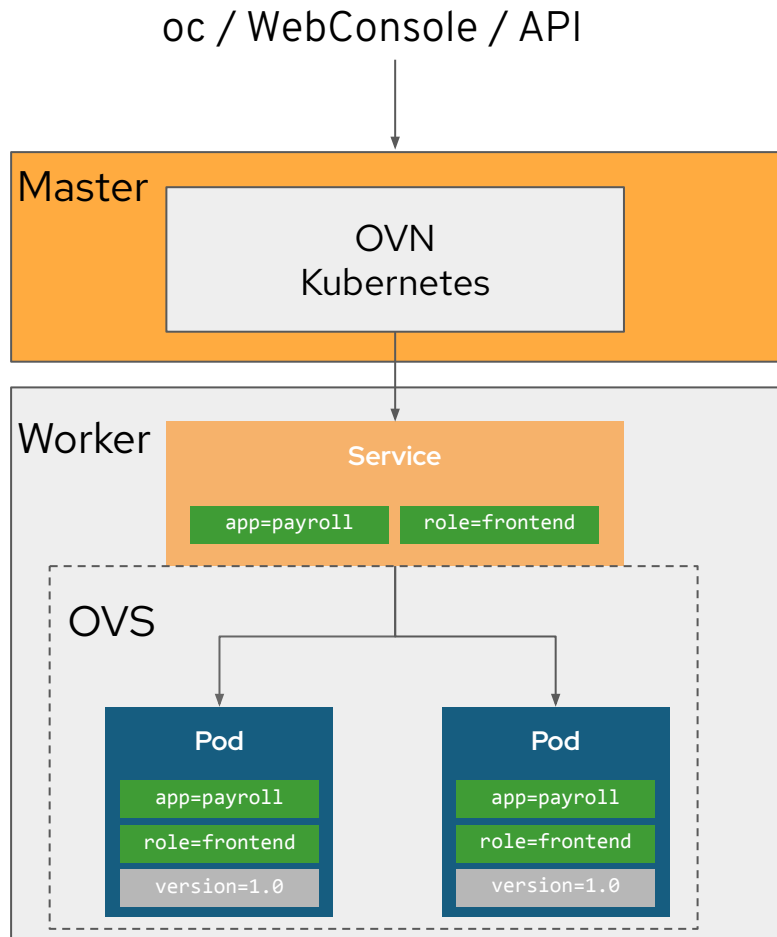
clusterNetwork.cidr: The cluster-wide network where **Pod IP** addresses are allocated

- 10.128.0.0/14
 - Host min: 10.128.0.0
 - Host max: 10.131.255.255
 - Addresses in network: 262144 (max pods per cluster)

serviceNetwork cidr: The IP address pool to use for **service IP** addresses.

- 172.30.0.0/16
 - Host min: 172.30.0.0
 - Host max: 172.30.255.255
 - Addresses in network: 65536 (max number of services)

OpenShift Network Plugins



OVN Kubernetes

- It is the CNI (Container Network Interface) plug-in used by OpenShift when OVN is selected as the networking solution.
- Translates Kubernetes network requests (such as pod creation, services, and network policies) into rules which are then applied at the OVS level.

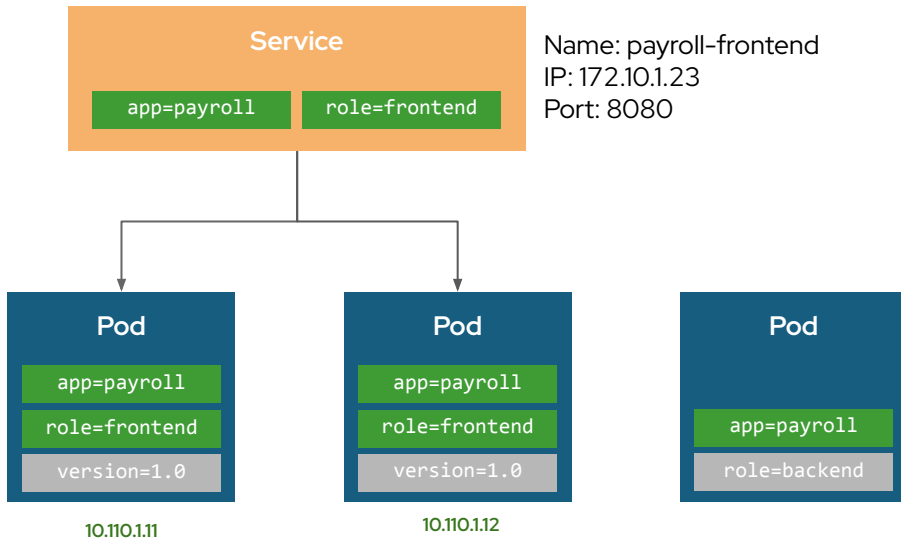
OVS (Open VSwitch)

- It is a virtual switch that runs on each node of the cluster.
- Acts as the network packet forwarding engine, applying the rules configured by OVN.

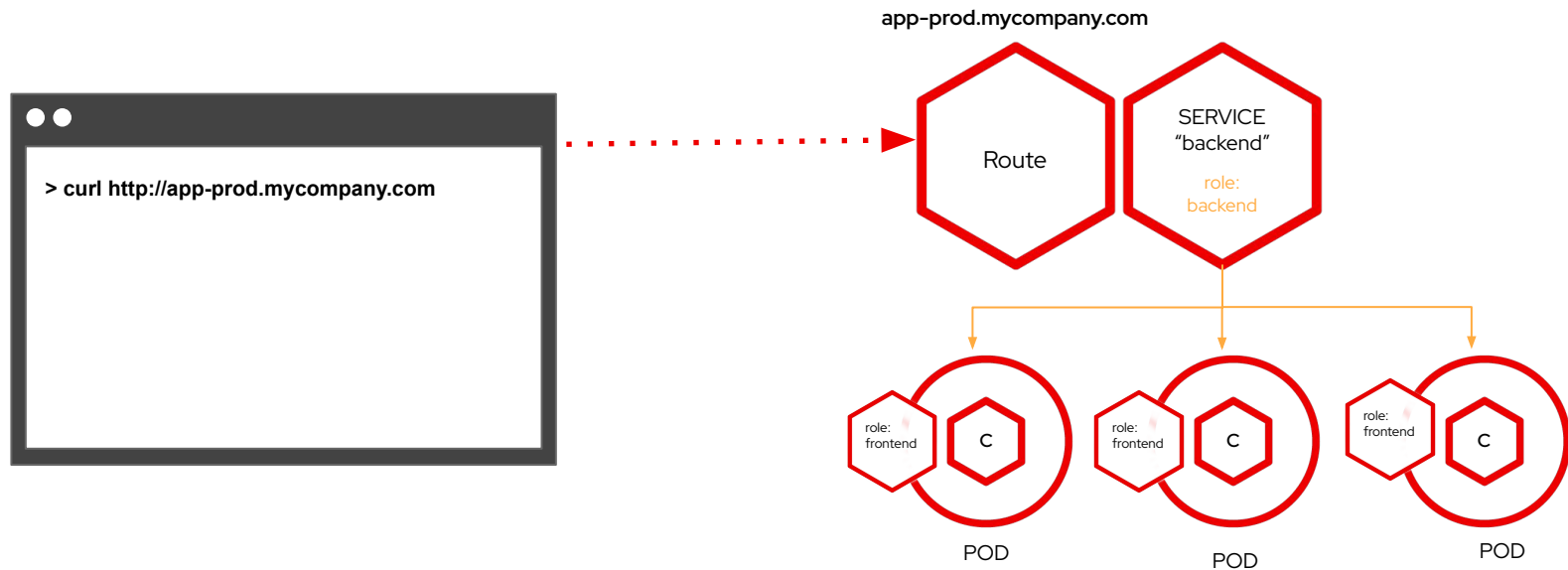
Kubernetes Services

A Service is an abstraction defining a logical set of Pods and a policy by which to access them. A Kubernetes service is an IP address from the overlay network load-balancing the pods matching the defined selector

```
apiVersion: v1
kind: Service
metadata:
  name: payroll-frontend
spec:
  selector:
    app: payroll
    role: frontend
  clusterIP: 172.10.1.23
  ports:
    - port: 8080
      protocol: TCP
      targetPort: 8080
```



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



External Traffic to a Service using NodePort

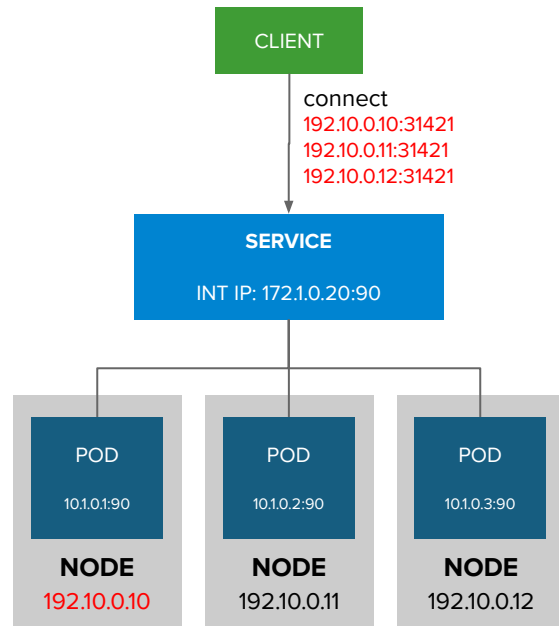
- NodePort binds a service to a unique port on all the nodes
- Ports in 30K-60K range which usually differs from the service
- Firewall rules must allow traffic to all nodes on the specific port

✓ Advantages

- Works without external load balancers.

✗ Disadvantages

- Requires knowing node IPs and firewall configuration



External Traffic to a Service using External IP

Assigns a fixed external IP address to a service.

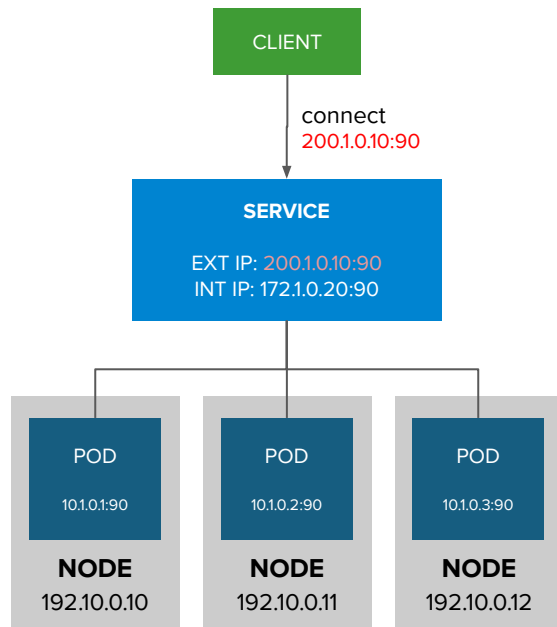
Requires manual network configuration to route traffic to the correct node.

✓ Advantages

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

✗ Disadvantages

- OpenShift does not manage external IPs automatically.



Load Balancer Services

- 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.

```
$ oc get svc -n openshift-ingress
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)
service/router-default	LoadBalancer	172.30.22.68	xx-yy.ap-northeast-1.elb.amazonaws.com	80:31155/TCP,443:32009/TCP

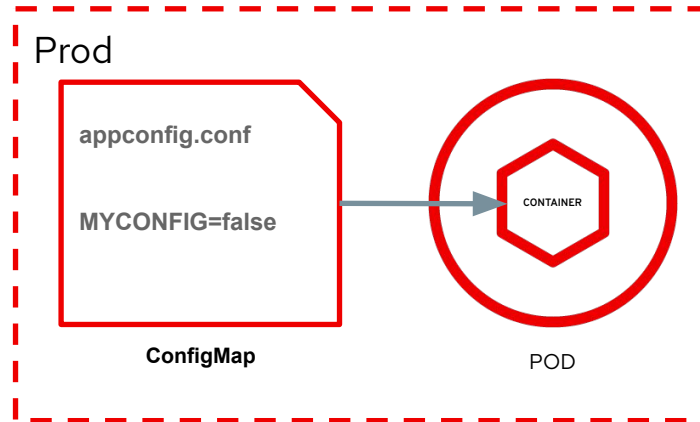
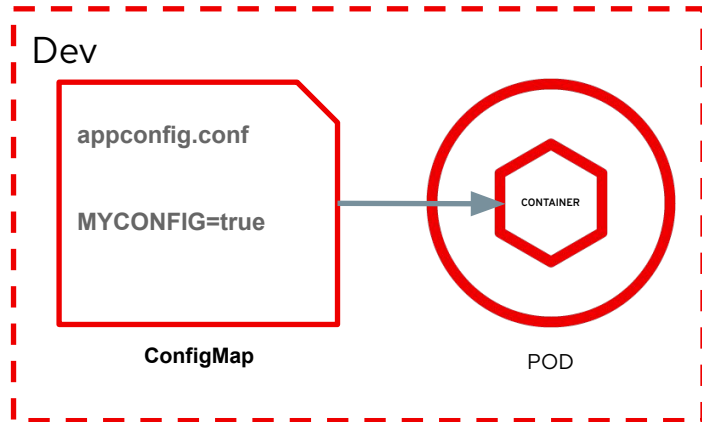
- MetalLB is a load balancer component that provides a load balancing service for clusters that do not run on a cloud provider

An abstract graphic on the left side of the slide, rendered in various shades of red. It features a vertical stack of server racks at the bottom, a cloud with a keyhole icon, a large upward-pointing arrow, and several curved arrows indicating data flow or movement. There are also some 'X' and 'O' symbols scattered throughout the design.

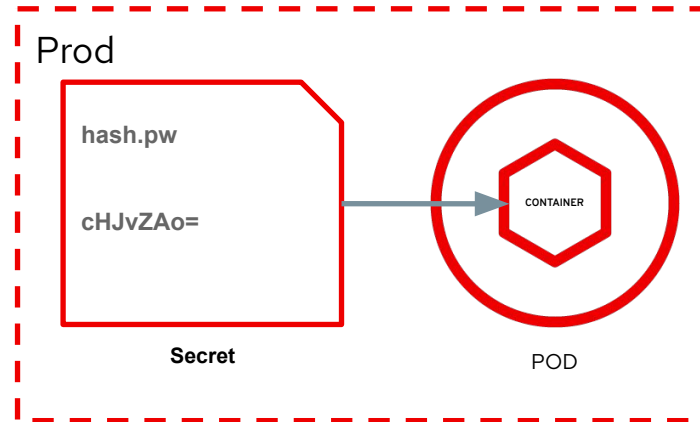
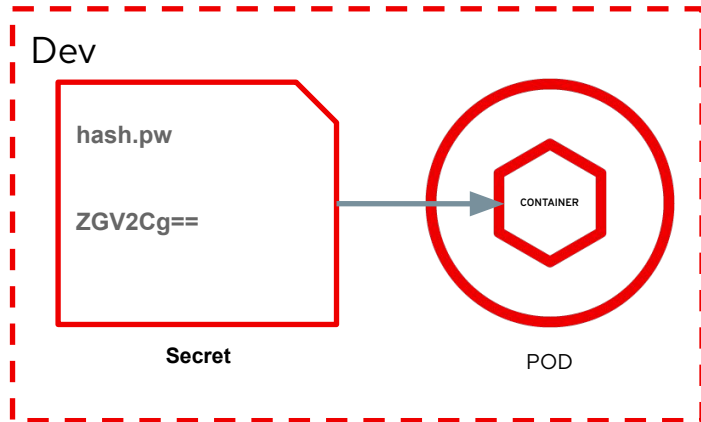
Manage Storage

Chapter 5

Configmaps allow you to decouple configuration artifacts from image content



Secrets provide a mechanism to hold sensitive information such as passwords



Persistent Storage

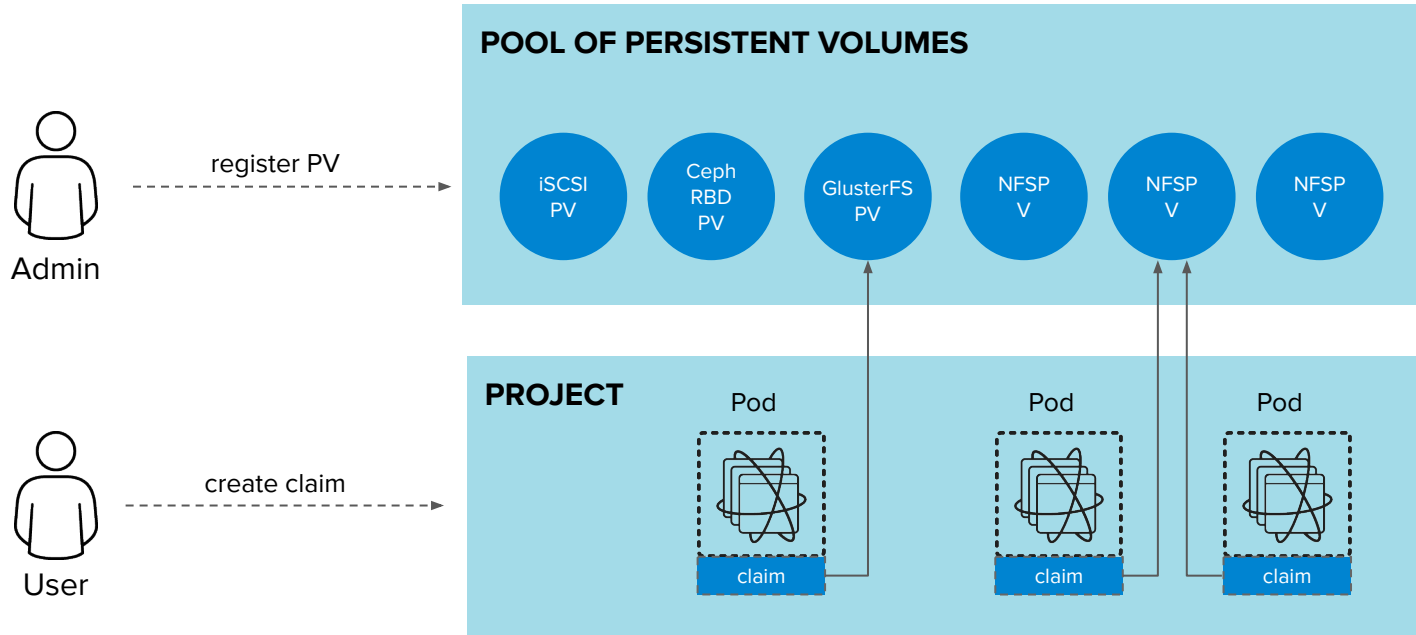
- Persistent Volume (PV) is tied to a piece of network storage
- Provisioned by an administrator (static or dynamically)
- Allows admins to describe storage and users to request storage
- Assigned to pods based on the requested size, access mode, labels and type

NFS	OpenStack Cinder	iSCSI	Azure Disk	AWS EBS	FlexVolume
GlusterFS	Ceph RBD	Fiber Channel	Azure File	GCE Persistent Disk	VMWare vSphere VMDK
		NetApp Trident*	Container Storage Interface (CSI)**		

* Shipped and supported by NetApp via TSANet

** Tech Preview

Persistent Storage



Persistent Volumes (PV)

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv0001
  annotations:
    volume.beta.kubernetes.io/mount-options: rw,nfsvers=4,noexec
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  nfs:
    path: /tmp
    server: 172.17.0.2
  persistentVolumeReclaimPolicy: Retain
  claimRef:
    name: claim1
    namespace: default
```

Specify Mount options (Explained in the next slides)

Set the specific the capacity of the PersistentVolume

Set the access Mode (Explained in the next slides)

Indicates how the resource should be handled once it is released

Persistent Volume Claims (PVC)

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: myclaim
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 8Gi
  storageClassName: gold
status:
  ...
```

Set the access Mode

Specify the amount of storage

Specify the Storage Class (Explained in the next slides)

PVC and PV binding

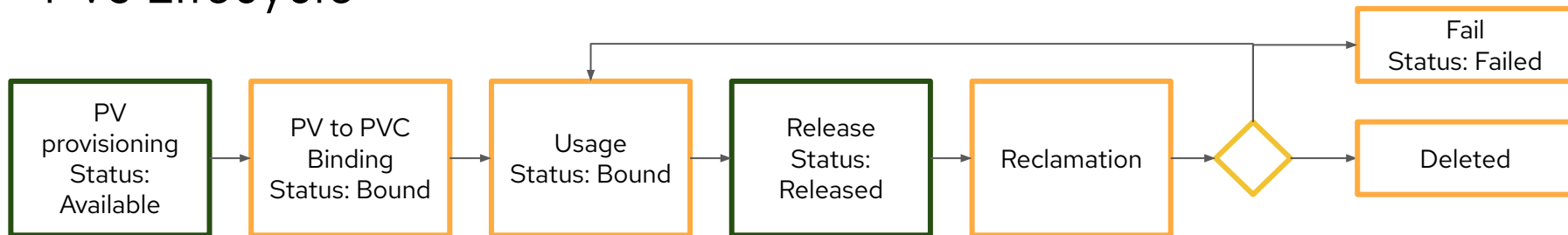
PVs and PVCs are bound according to the compatibility of:

- ▶ accessMode
- ▶ Capacity
- ▶ storageClass (optional)

Claims will remain unbound indefinitely if a matching volume does not exist

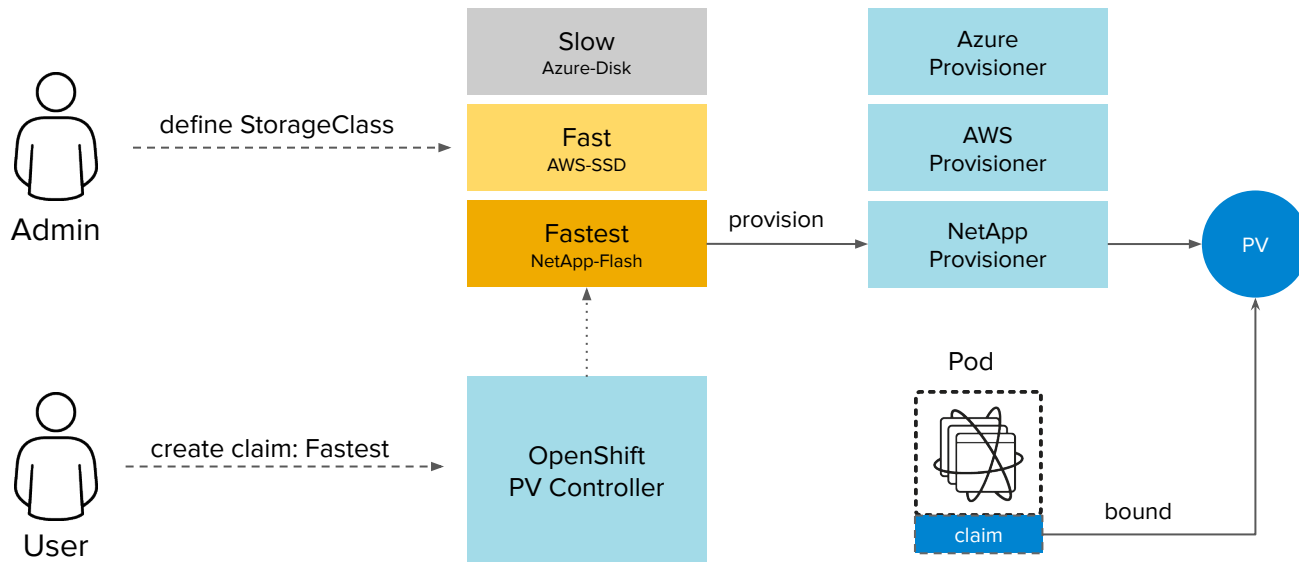
Dynamic Provisioning: A PV is dynamically provisioned for a new PVC. This PV is bounded to the PVC

PVs Lifecycle



Phase	Description
Available	A free resource not yet bound to a claim.
Bound	The volume is bound to a claim.
Released	The claim was deleted, but the resource is not yet reclaimed by the cluster.
Failed	The volume has failed its automatic reclamation.

DYNAMIC VOLUME PROVISIONING



Storage Class

The StorageClass resource object describes and classifies storage that can be requested

StorageClass objects are a globally scoped object

The cluster administrator can configure dynamic provisioners to service one or more storage classes.

The storageClass classifies storage by different criteria:

- Technology (AWS, Vsphere..)
- Service Quality (Example: Disktype for vSphere: zeroedthick or thin)

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: gp2
  annotations:
    storageclass.kubernetes.io/is-default-class: 'true'
  ...
provisioner: kubernetes.io/aws-ebs
parameters:
  type: gp2
```

Annotations for the storage class. Include kubernetes.io/description to add a description.

Storage Provisioner (only dynamic)

Parameters specific to the storage provisioner

Stateful Sets



StatefulSet

What Are StatefulSets?

Definition: StatefulSets are a Kubernetes resource used to manage stateful applications with stable network identities, persistent storage, and ordered deployments.

Key Features:

- Stable Pod Names: Pods are created in a predictable order (e.g., db-0, db-1).
- Persistent Storage: Each pod retains its own volume across restarts.
- Ordered Scaling & Rolling Updates: Ensures controlled pod creation and deletion.

Stateful Sets use cases

StatefulSets: Managing Distributed Databases

- Use Case: StatefulSets are useful for databases that require primary-secondary replication or sharding.
- Persistence: Each pod in a StatefulSet has its own persistent storage and a stable network identity.
 - Primary-secondary replication (PostgreSQL, MySQL, MongoDB replica set).
 - Sharded databases (MongoDB Sharded Cluster, Cassandra, Elasticsearch, CockroachDB).
 - Benefit: Ensures data consistency, failover capabilities, and workload distribution.

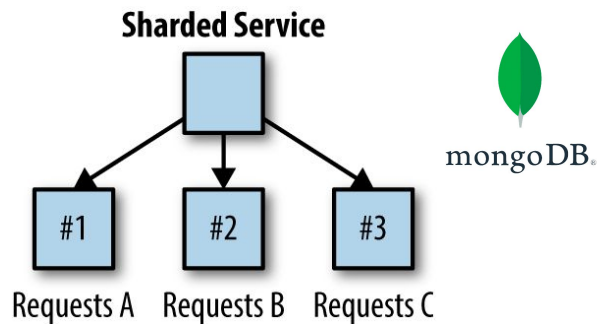
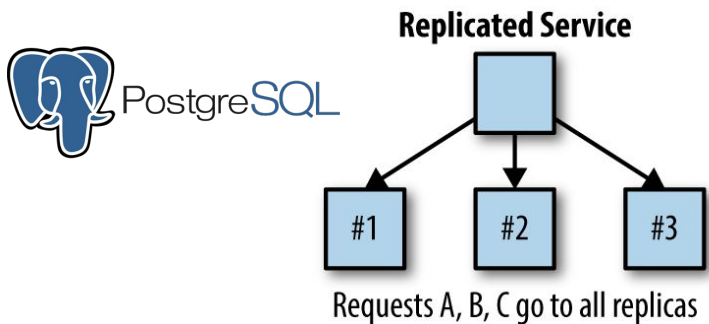
Stateful Sets and Databases

Primary-Secondary Replication:

- Primary pod handles writes.
- Secondary pods replicate data and serve read queries.

Sharded Databases:

- Each pod manages a subset of data.
- No shared storage between shards.



An abstract graphic on the left side of the slide, rendered in various shades of red. It features a vertical stack of server racks at the bottom, a cloud with a keyhole icon, a large upward-pointing arrow, and several curved arrows indicating flow or movement. There are also some 'X' and 'O' symbols scattered throughout the design.

High Availability

Chapter 6

Kubernetes High Availability

Kubernetes uses the following HA techniques to improve application reliability:

- **Restarting pods:** By configuring a restart policy on a pod, the cluster restarts misbehaving instances of an application.
- **Probing:** By using health probes, the cluster knows when applications cannot respond to requests, and can automatically act to mitigate the issue.
- **Horizontal scaling:** When the application load changes, the cluster can scale the number of replicas to match the load.

Restart Policy

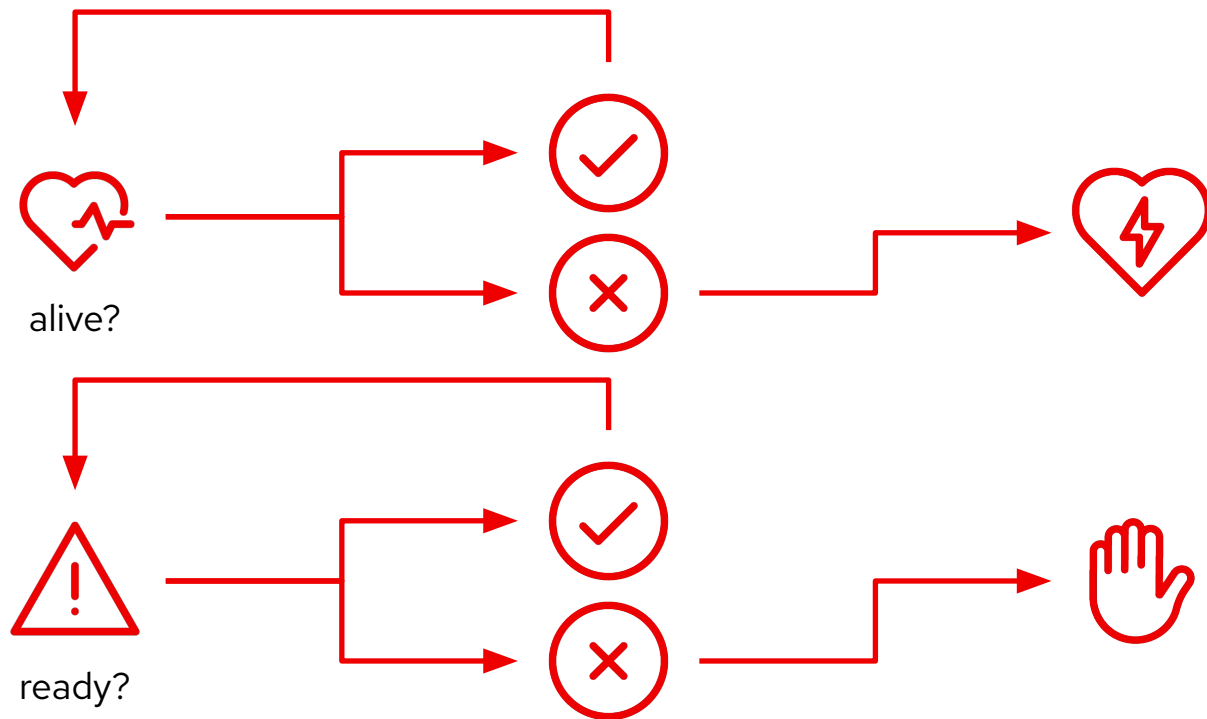
```
apiVersion: v1
kind: Pod
metadata:
  name: example-pod
  namespace: my-namespace
spec:
  restartPolicy: OnFailure
  containers:
    - name: my-container
      image: httpd:latest
      ports:
        - containerPort: 8080
```

The Pod restarts only if the container exits with a failure (non-zero exit code).

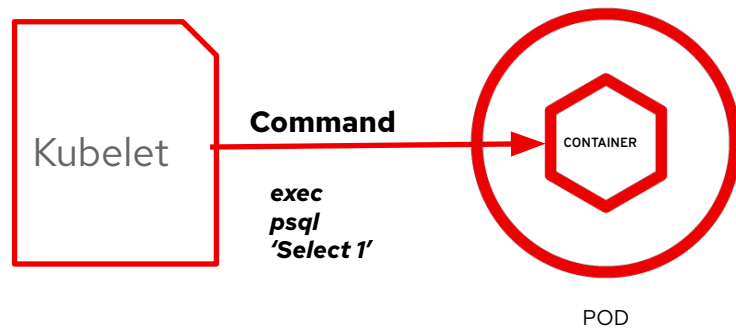
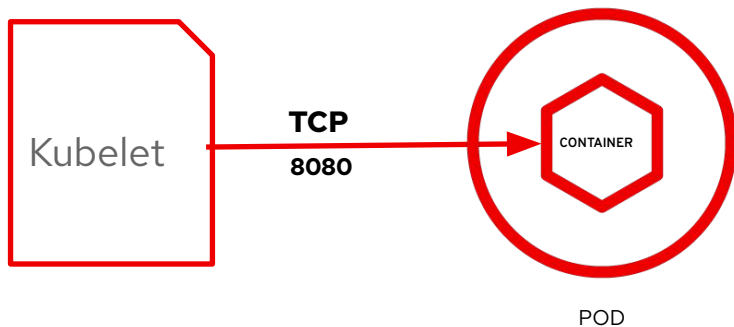
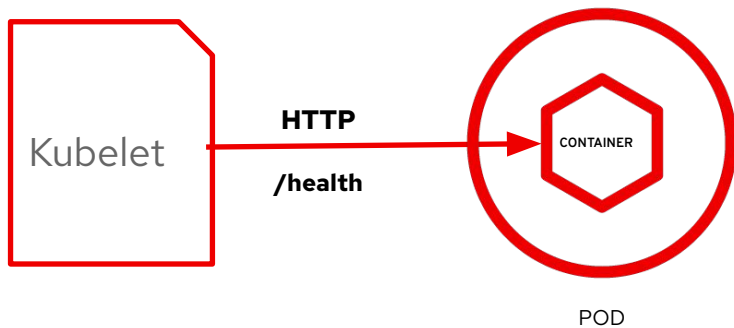
Explanation of restartPolicy Values

- Always (default) → The Pod restarts automatically if the container stops (even if it exits successfully).
- OnFailure → The Pod restarts only if the container exits with a failure (non-zero exit code).
- Never → The Pod never restarts, even if it fails.

Liveness and Readiness



Type of Probes



An abstract graphic on the left side of the slide, rendered in various shades of red. It features a vertical stack of server racks at the bottom, a cloud with a keyhole icon, a large upward-pointing arrow, and several curved arrows indicating flow or movement. There are also some 'X' and 'O' symbols scattered throughout the design.

Limit Compute Capacity for Applications

Chapter 6

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)
```

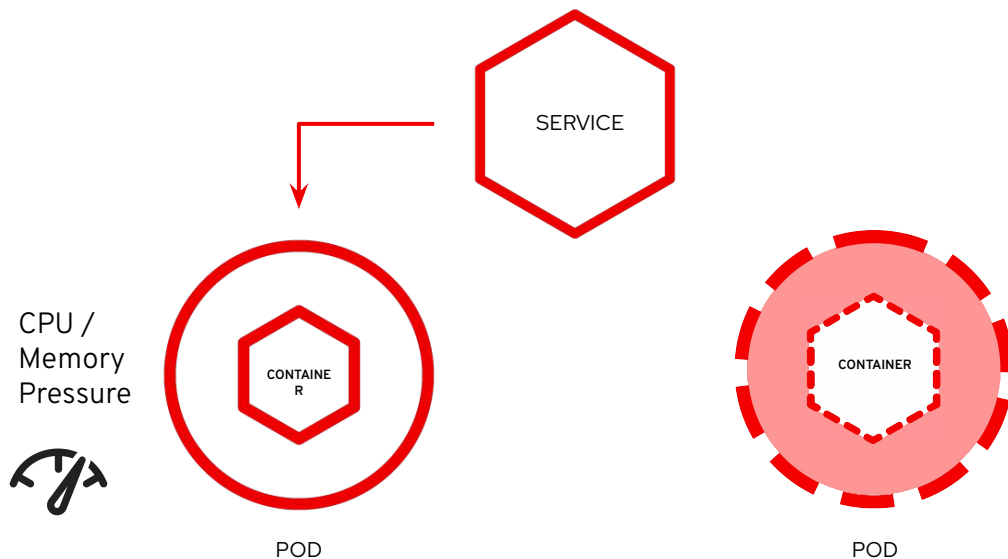
Object Count Quotas

You can define the maximum number of resources you can allow for a single namespace

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: example
spec:
  hard:
    count/pods: "1"
```

Horizontal Autoscaler

The Horizontal Pod Autoscaler (HPA) automatically scales the number of pods in a deployment, replica set, or stateful set based on CPU, memory, or custom metrics.



How the HPA works

- **Monitors Resource Usage:** HPA checks metrics (e.g., CPU, memory) through the Kubernetes Metrics Server.
- **Adjusts Pod Count:** If resource usage exceeds the defined threshold, HPA increases the number of pods; if usage drops, it reduces them.
- **Ensures Efficient Scaling:** Helps maintain application performance while optimizing resource usage.

```
oc autoscale deployment/hello --min 1 --max 10 --cpu-percent 80
```



ImageStreams

Chapter 7

How an ImageStream works

ImageStreams reference images from registries (internal or external).

They store metadata about images, not the images themselves.

Image updates trigger Builds, Deployments, or CI/CD workflows.

An ImageStream contains these references:

- ImageStream (logical reference)
- ImageStreamTag (specific image version)
- ImageStreamImage (immutable reference to an image SHA)

An example

- This defines an ImageStream named my-app.
- It can track images from a registry (like Quay.io or Docker Hub).
- Can be used in BuildConfigs and DeploymentConfigs.

```
apiVersion: image.openshift.io/v1
kind: ImageStream
metadata:
  name: my-app
spec:
  lookupPolicy:
    local: true
```

ImageStream Use Cases

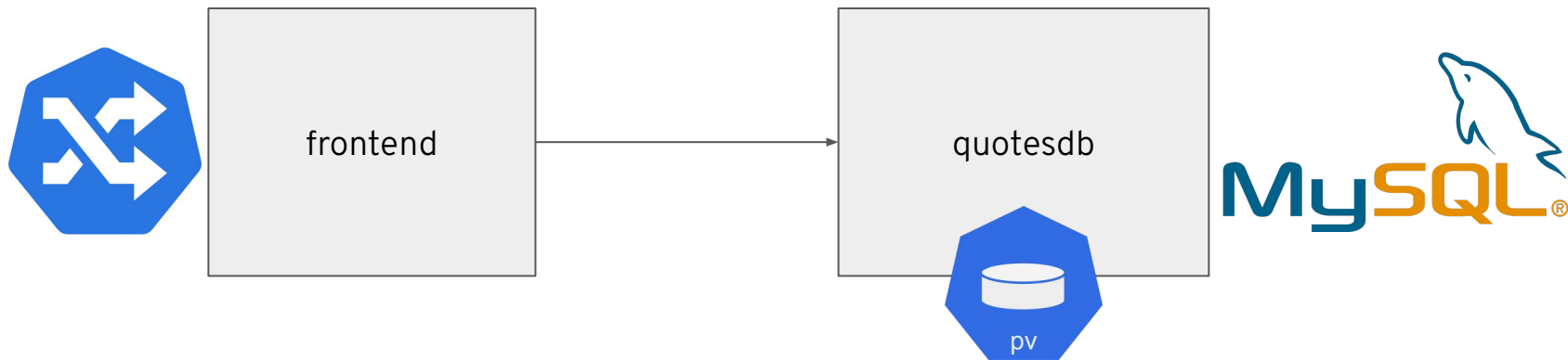
- Automated Deployments: Update apps when a new image is available.
- CI/CD Integration: Trigger builds when a new base image is pushed.
- Rollback Support: Keep multiple image versions for quick recovery.
- Security Compliance: Ensure only approved images are deployed.



Final Review

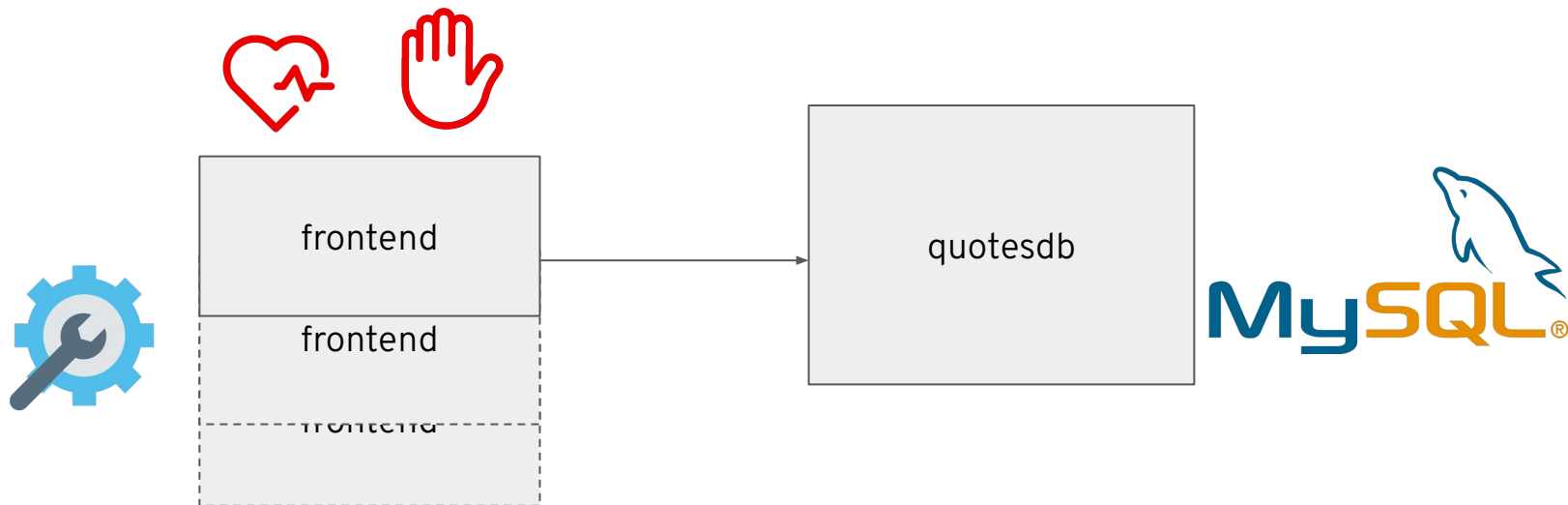
Lab compreview-deploy

Estimated
time:
45 minutes



Lab compreview-scale

Estimated
time:
45 minutes



Pull Policies

Policy	Tag	Behaviour
IfNotPresent	Specific tag (1.1)	Uses the local image if it exists on the node; otherwise, pulls the image from the registry.
IfNotPresent	Floating Tag (latest)	Uses the local image if it exists on the node, even if there is a newer image with the same tag in the registry.
Always	Any Tag	Always checks the registry for an updated image. If the local image's SHA ID matches the registry's SHA ID, it uses the local image. Otherwise, it pulls the image.
Always	No Tag	Defaults to latest, always checks the registry, and pulls the image if it's not up-to-date.
Never		Does not pull the image from the registry. The image must already be preloaded on the node, or the deployment will fail.

Questions ?

