Chapter 6

DEPLOYING CONTAINERIZED APPLICATIONS ON OPENSHIFT

Describing Kubernetes and OpenShift Architecture

Kubernetes

- Kubernetes is an orchestration service that simplifies the deployment, management, and scaling of containerized applications.
- One of the main advantages of using Kubernetes is that it uses several nodes to ensure the flexibility and scalability of its managed applications.
- Kubernetes forms a **cluster** of **node servers** that run containers and are centrally managed by a set of **control plane servers**.
- A server can act as both a **control plane node** and a **compute node**, but those roles are usually separated for increased stability.

Kubernetes Terminology

Node	A server that hosts applications in a Kubernetes cluster.
Control Plane	Provides basic cluster services such as APIs or controllers.
Compute Node	This node executes workloads for the cluster. Application pods are scheduled onto compute nodes.
Resource	Resources are any kind of component definition managed by Kubernetes. Resources contain the configuration of the managed component (for example, the role assigned to a node), and the current state of the component (for example, if the node is available).
Controller	A controller is a Kubernetes process that watches resources and makes changes attempting to move the current state towards the desired state.
Label	A key-value pair that can be assigned to any Kubernetes resource. Selectors use labels to filter eligible resources for scheduling and other operations.
Namespace	A scope for Kubernetes resources and processes, so that resources with the same name can be used in different boundaries.

OpenShift

- Red Hat OpenShift Container Platform is a set of modular components and services built on top of Red Hat CoreOS and Kubernetes.
- RHOCP adds PaaS capabilities such as remote management, increased security, monitoring and auditing, application lifecycle management, and self-service interfaces for developers.
- An OpenShift cluster is a Kubernetes cluster that can be managed the same way, but using the management tools provided by OpenShift, such as the **command-line interface** or the **web console.**
- This allows for more productive workflows and makes common tasks much easier.

OpenShift Terminology

Infra Node	A node server containing infrastructure services like monitoring, logging, or external routing.
Console	A web UI provided by the RHOCP cluster that allows developers and administrators to interact with cluster resources.
Project	OpenShift extension of Kubernetes' namespaces. Allows the definition of user access control (UAC) to resources.

OpenShift Component Stack

DevOps Tools and User Experience
Web Console, CLI, REST API, SCM integration

Containerized Services
Auth, Networking, Image Registry

Runtimes and xPaaS
Java, Ruby, Node.js and more

Kubernetes
Container orchestration and management

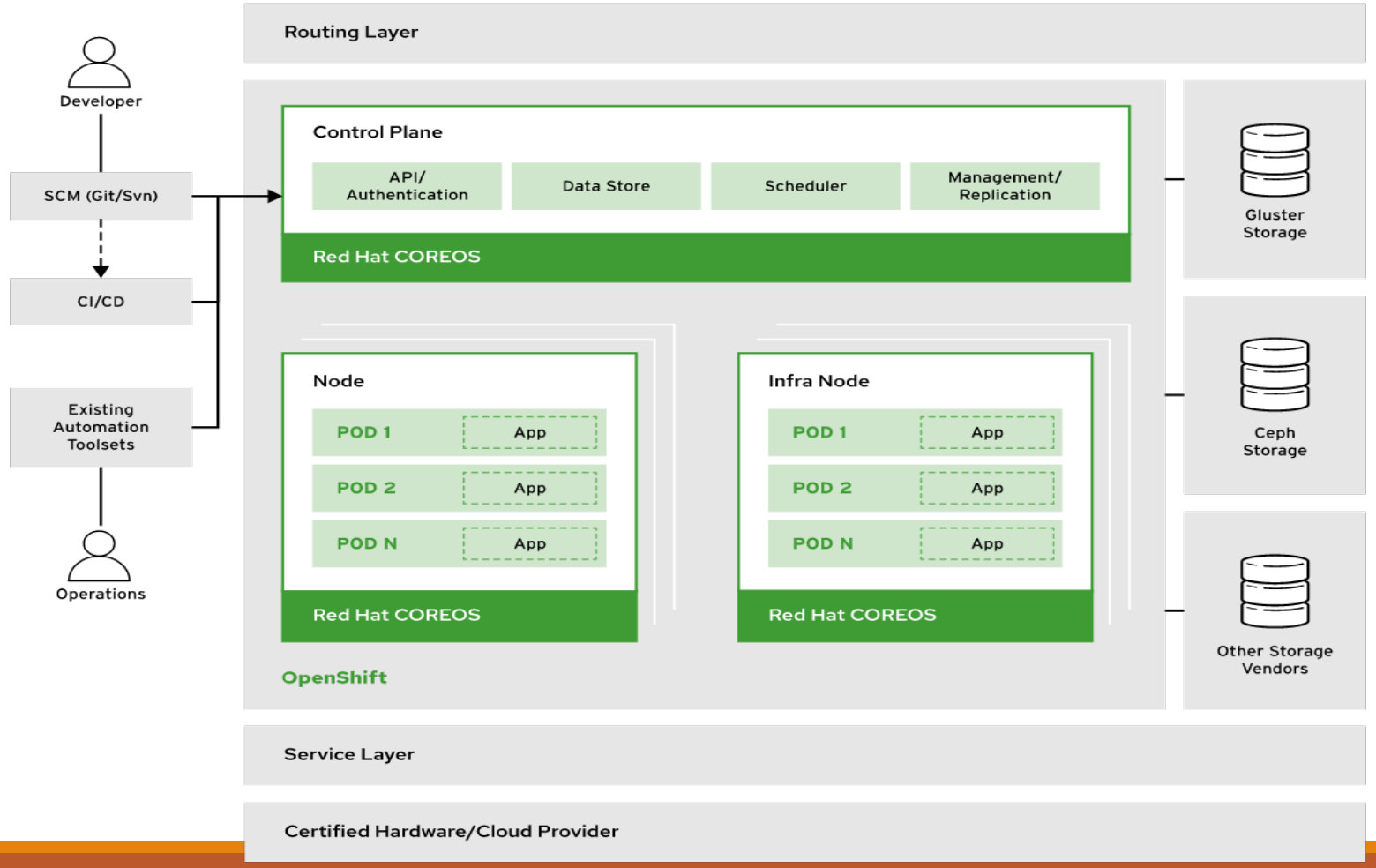
Etcd
Cluster state and configs

CRDs
Kubernetes Operators

CRI-O
Container runtime

Red Hat CoreOS
Container optimized OS

OpenShift and Kubernetes Architecture



Describing Kubernetes Resource Types

- **Pods (pod):** Represent a collection of containers that share resources, such as IP addresses and persistent storage volumes. It is the basic unit of work for Kubernetes.
- **Services (svc):** Define a single IP/port combination that provides access to a pool of pods. By default, services connect clients to pods in a round-robin fashion.
- **Replication Controllers (rc):** A Kubernetes resource that defines how pods are replicated (horizontally scaled) into different nodes. Replication controllers are a basic Kubernetes service to provide high availability for pods and containers.
- **Persistent Volumes (pv):** Define storage areas to be used by Kubernetes pods.
- **Persistent Volume Claims (pvc):** Represent a request for storage by a pod. PVCs links a PV to a pod so its containers can make use of it, usually by mounting the storage into the container's file system.

OpenShift Resource Types

- **Deployment and Deployment config (dc)**

  OpenShift 4.5 introduced the Deployment resource concept to replace the DeploymentConfig as the default configuration for pods. Both are the **representation of a set of containers included in a pod**, and the **deployment strategies** to be used. It contains the configuration to be applied to all containers of each pod **replica**, such as the base image, tags, storage definitions and the commands to be executed when the containers start.

- **Build config (bc)**

  Defines a process to be executed in the OpenShift project. Used by **the OpenShift Source-to-Image** (S2I) feature to build a container image from application source code stored in a Git repository. A bc works together with a dc to provide a basic but extensible continuous integration and continuous delivery workflows.

- **Routes**

  Represent a DNS host name recognized by the OpenShift router as an ingress point for applications and microservices.

Networking

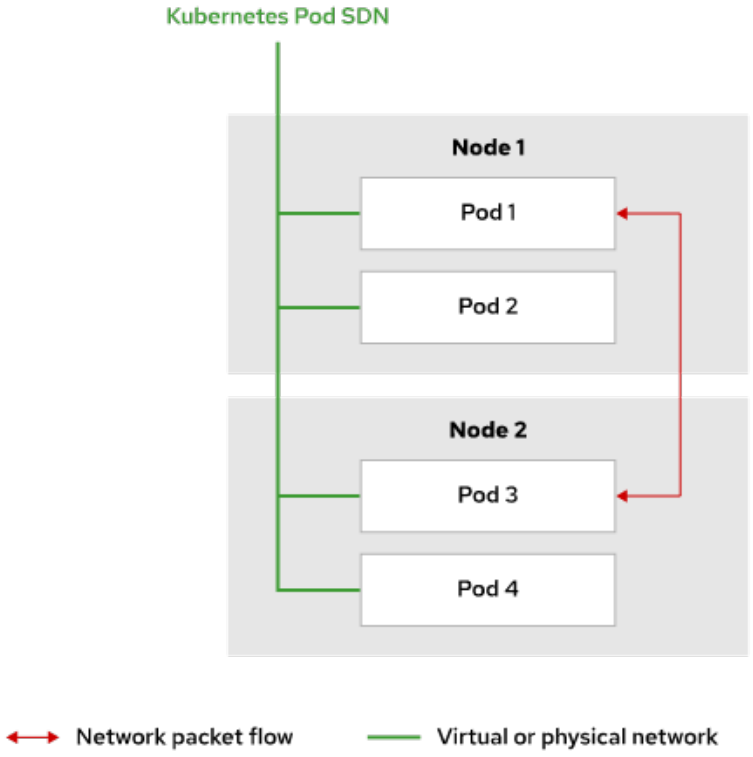
- Each container deployed in a Kubernetes cluster has an **IP address** assigned from an internal network that is accessible only from the node running the container. Because of the container's ephemeral nature, IP addresses are constantly **assigned and released**.
- External access to containers is more complicated. Kubernetes services can be set as a **NodePort service type**. With this type of service, Kubernetes allocates a network port from a predefined range in each of the cluster nodes and proxies it into your service. Unfortunately, this approach does not scale well.
- **OpenShift** makes external access to containers both scalable and simpler by **defining route resources.** A route defines external-facing DNS names and ports for a service. A router (ingress controller) forwards HTTP and TLS requests to the service addresses inside the Kubernetes SDN. The only requirement is that the desired DNS names are mapped to the IP addresses of the RHOCP router nodes.

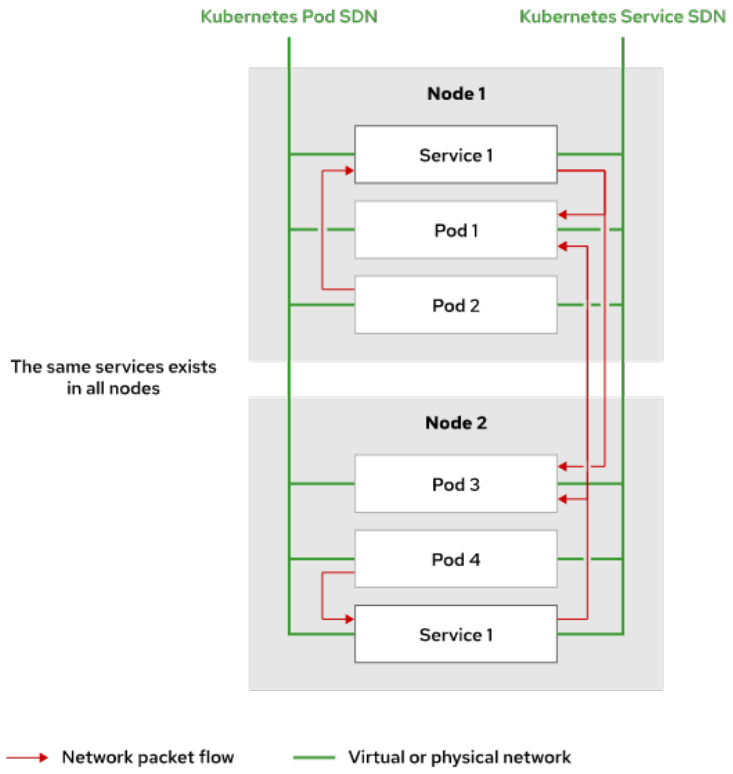
Creating Kubernetes Resources

Describing Pod Resource Definition Syntax

```
apiVersion: v1
kind: Pod                                  (1)
metadata:
  name: wildfly                            (2)
  labels:
    name: wildfly                          (3)
spec:
  containers:
    - resources:
        limits:
          cpu: 0.5
      image: do276/todojee                 (4)
      name: wildfly
      ports:
        - containerPort: 8080              (5)
          name: wildfly
      env:                                 (6)
        - name: MYSQL_ENV_MYSQL_DATABASE
          value: items
        - name: MYSQL_ENV_MYSQL_USER
          value: user1
        - name: MYSQL_ENV_MYSQL_PASSWORD
          value: mypa55
```

Describing Service Resource Definition Syntax





Minimal service definition in JSON syntax

```
{
    "kind": "Service", (1)
    "apiVersion": "v1",
    "metadata": {
        "name": "quotedb" (2)
    },
    "spec": {
        "ports": [ (3)
            {
                "port": 3306,
                "targetPort": 3306
            }
        ],
        "selector": {
            "name": "mysqldb" (4)
        }
    }
}
```

Discovering Services - Internally

- An application typically finds a service IP address and port by using environment variables. For each service inside an OpenShift project, the following environment variables are automatically defined and **injected into containers for all pods inside the same project**:

SVC_NAME_SERVICE_HOST is the service **IP address**.

SVC_NAME_SERVICE_PORT is the service **TCP port**.

Two ways how an application can access the service from **outside** an OpenShift cluster

- **NodePort type:** This is an older Kubernetes-based approach, where the service is exposed to external clients by binding to **available ports on the compute node host**, which then proxies connections to the service IP address. Use the oc edit svc command to edit service attributes and specify NodePort as the value for type, and provide a port value for the nodePort attribute. OpenShift then proxies connections to the service via the public IP address of the compute node host and the port value set in NodePort.

- **OpenShift Routes:** This is the preferred approach in OpenShift to expose services using a **unique URL**. Use the **oc expose** command to expose a service for external access or expose a service from the OpenShift web console.

(RHOCP) Command-line Tool

```
[user@host ~]$ oc <command>
```

```
[user@host ~]$ oc login <cluster-url>
```

Creating Applications

The following command creates an application based on an image from a private Docker image registry:

```
[user@host ~]$ oc new-app --image=myregistry.com/mycompany/myapp --name=myapp
```

The following command creates an application based on source code stored in a Git repository:

```
[user@host ~]$ oc new-app https://github.com/openshift/ruby-hello-world \
> --name=ruby-hello
```

Managing Persistent Storage

You can create persistent storage and attach it to your application. In this way you can make sure your data is not lost when deleting your pods.

To list the PersistentVolume objects in a cluster, use **the oc get pv** command:

```
[admin@host ~]$ oc get pv
NAME     CAPACITY  ACCESS MODES  RECLAIM POLICY   STATUS      CLAIM   ...
pv0001   1Mi       RWO           Retain           Available           ...
pv0002   10Mi      RWX           Recycle          Available           ...
...output omitted...
```

To add more PersistentVolume objects to a cluster, use the oc create command:

```
[admin@host ~]$ oc create -f pv1001.yaml
```

Managing OpenShift Resources at the Command Line

The oc get RESOURCE_TYPE command displays a summary of all resources of the specified type. The following illustrates example output of the **oc get pods** command.

```
NAME              READY   STATUS      RESTARTS   AGE
nginx-1-5r583     1/1     Running     0          1h
myapp-1-l44m7     1/1     Running     0          1h
```

- **oc get Resource_Type**
- **oc get all**
- **oc describe Resource_Type/Resource_Name**
- **oc edit**
- **oc create**
- **oc delete**
- **oc exec**

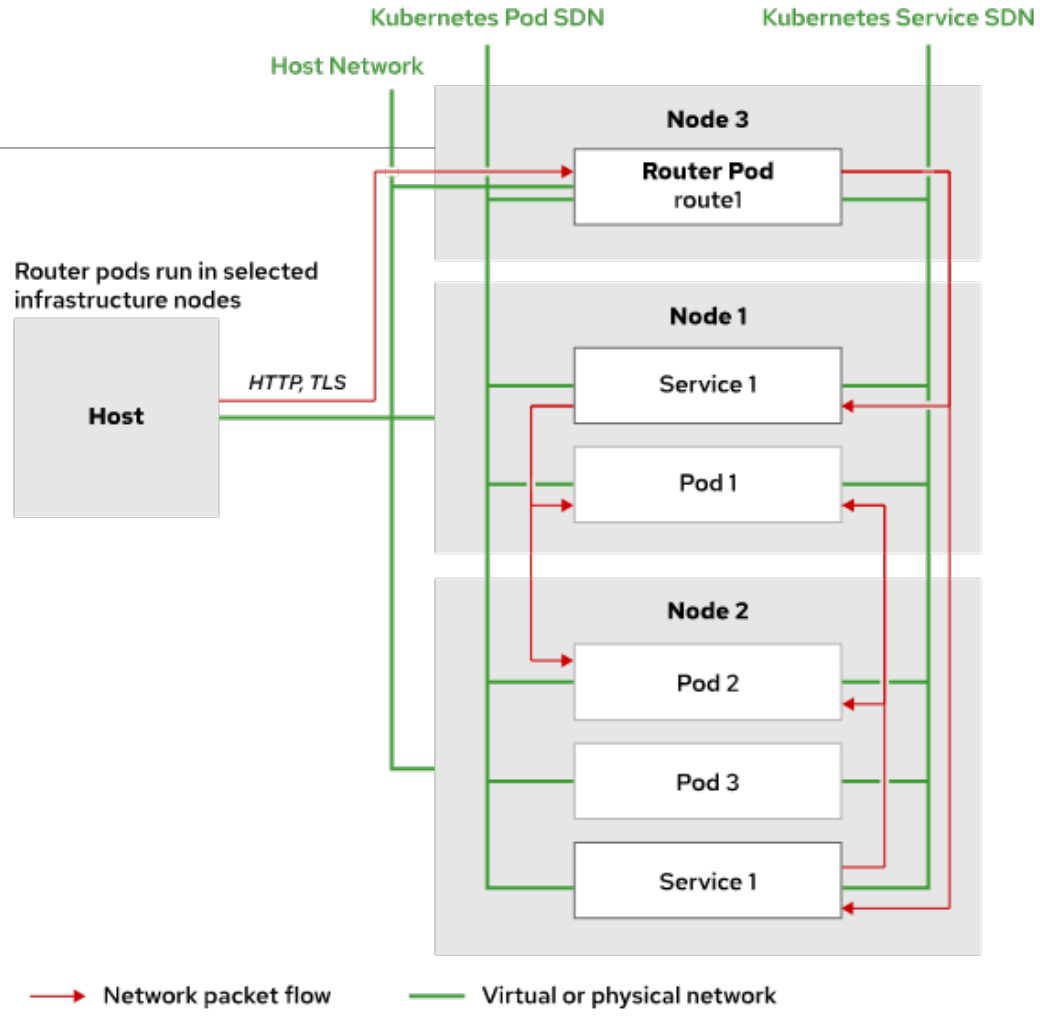
Guided Exercise: Deploying a Database Server on OpenShift

https://rha.ole.redhat.com/rha/app/courses/do180-4.10/52f11f0e-a277-4441-9d11-e3d56d7defca/pages/ch06s04

Creating Routes

Working with Routes

Services allow for network access between pods inside an OpenShift instance, and routes allow for network access to pods from users and applications outside the OpenShift instance.



Routes Definition in JSON

```
{
    "apiVersion": "v1",
    "kind": "Route",
    "metadata": {
        "name": "quoteapp"
    },
    "spec": {
        "host": "quoteapp.apps.example.com",
        "to": {
            "kind": "Service",
            "name": "quoteapp"
        }
    }
}
```

Creating routes

By the **oc create** command just any other resource

OR

Exposing a service as a route

oc expose service quotedb --name quote

Guided Exercise: Exposing a Service as a Route

https://rha.ole.redhat.com/rha/app/courses/do180-4.10/52f11f0e-a277-4441-9d11-e3d56d7defca/pages/ch06s06

Creating Applications with Source-to-Image

The Source-to-Image (S2I) Process

Source-to-Image (S2I) is a tool that makes it easy to build container images from application source code.

This tool takes an application's source code from a Git repository, injects the source code into a base container based on the language and framework desired, and produces a new container image that runs the assembled application.

Image Streams

- OpenShift deploys new versions of user applications into pods quickly. To create a new application, in addition to the application source code, a base image (the S2I builder image) is required.
- If either of these two components gets updated, OpenShift creates a new container image.
- Pods created using the older container image are replaced by pods using the new image.
- The **image stream resource** is a configuration that names specific container images associated with image stream tags, an alias for these container images. OpenShift builds applications against an image stream.

Building an Application with S2I and the CLI

```
[user@host ~]$ oc new-app php~http://my.git.server.com/my-app \ (1)(2)
>  --name=myapp (3)
```

(1) The image stream used in the process appears to the left of the tilde (~).

(2) The URL after the tilde indicates the location of the source code's Git repository.

(3) Sets the application name.

Some Options of S2I

creating an application using the Git repository in the current directory:

```
[user@host ~]$ oc new-app .
```

creating an application using a remote Git repository with a specific branch reference:

```
[user@host ~]$ oc new-app https://github.com/openshift/ruby-hello-world.git#beta4
```

Creating resources for the image

Create a JSON resource definition file by using the -o json parameter and output redirection:

```
[user@host ~]$ oc -o json new-app php~http://services.lab.example.com/app \
> --name=myapp > s2i.json
```

This JSON definition file creates a list of resources, like image stream, build config (bc), deployment object and service.

Guided Exercise: Creating a Containerized Application with Source-to-Image

https://rha.ole.redhat.com/rha/app/courses/do180-4.10/52f11f0e-a277-4441-9d11-e3d56d7defca/pages/ch06s08

Creating Applications with the OpenShift Web Console

Creating Applications with the OpenShift Web Console

- The default URL is of the format:

  https://console-openshift-console.${RHT_OCP4_WILDCARD_DOMAIN}/

- Create new Projects and applications
- Managing Build configuration
- Creating Routes

Guided Exercise: Creating an Application with the Web Console

https://rha.ole.redhat.com/rha/app/courses/do180-4.10/52f11f0e-a277-4441-9d11-e3d56d7defca/pages/ch06s10

Lab: Deploying Containerized Applications on OpenShift

https://rha.ole.redhat.com/rha/app/courses/do180-4.10/52f11f0e-a277-4441-9d11-e3d56d7defca/pages/ch06s11