OpenShift Dedicated 4

Architecture

An overview of the architecture for Dedicated 4

Last Updated: 2020-06-18

OpenShift Dedicated 4 Architecture

An overview of the architecture for Dedicated 4

Legal Notice

Abstract

This document provides an overview of the platform and application architecture in OpenShift Dedicated 4. The underlying platform architecture is drastically different from previous versions of OpenShift Dedicated.

Table of Contents

CHAPTER 1. OPENSHIFT DEDICATED ARCHITECTURE

1.1. INTRODUCTION TO OPENSHIFT DEDICATED

OpenShift Dedicated is a platform for developing and running containerized applications. It is designed to allow applications and the data centers that support them to expand from just a few machines and applications to thousands of machines that serve millions of clients.

With its foundation in Kubernetes, OpenShift Dedicated incorporates the same technology that serves as the engine for massive telecommunications, streaming video, gaming, banking, and other applications. Its implementation in open Red Hat technologies lets you extend your containerized applications beyond a single cloud to on-premise and multi-cloud environments.

1.1.1. About Kubernetes

Although container images and the containers that run from them are the primary building blocks for modern application development, to run them at scale requires a reliable and flexible distribution system. Kubernetes is the defacto standard for orchestrating containers.

Kubernetes is an open source container orchestration engine for automating deployment, scaling, and management of containerized applications. The general concept of Kubernetes is fairly simple:

- Start with one or more worker nodes to run the container workloads.
- Manage the deployment of those workloads from one or more master nodes.
- Wrap containers in a deployment unit called a Pod. Using Pods provides extra metadata with the container and offers the ability to group several containers in a single deployment entity.
- Create special kinds of assets. For example, services are represented by a set of Pods and a policy that defines how they are accessed. This policy allows containers to connect to the services that they need even if they do not have the specific IP addresses for the services. Replication controllers are another special asset that indicates how many Pod Replicas are required to run at a time. You can use this capability to automatically scale your application to adapt to its current demand.

In only a few years, Kubernetes has seen massive cloud and on-premise adoption. The open source development model allows many people to extend Kubernetes by implementing different technologies for components such as networking, storage, and authentication.

1.1.2. The benefits of containerized applications

Using containerized applications offers many advantages over using traditional deployment methods. Where applications were once expected to be installed on operating systems that included all their dependencies, containers let an application carry their dependencies with them. Creating containerized applications offers many benefits.

1.1.2.1. Operating system benefits

Containers use small, dedicated Linux operating systems without a kernel. Their file system, networking, cgroups, process tables, and namespaces are separate from the host Linux system, but the containers can integrate with the hosts seamlessly when necessary. Being based on Linux allows containers to use all the advantages that come with the open source development model of rapid innovation.

Because each container uses a dedicated operating system, you can deploy applications that require

conflicting software dependencies on the same host. Each container carries its own dependent software and manages its own interfaces, such as networking and file systems, so applications never need to compete for those assets.

1.1.2.2. Deployment and scaling benefits

If you employ rolling upgrades between major releases of your application, you can continuously improve your applications without downtime and still maintain compatibility with the current release.

You can also deploy and test a new version of an application alongside the existing version. Deploy the new application version in addition to the current version. If the container passes your tests, simply deploy more new containers and remove the old ones.

Since all the software dependencies for an application are resolved within the container itself, you can use a generic operating system on each host in your data center. You do not need to configure a specific operating system for each application host. When your data center needs more capacity, you can deploy another generic host system.

Similarly, scaling containerized applications is simple. OpenShift Dedicated offers a simple, standard way of scaling any containerized service. For example, if you build applications as a set of microservices rather than large, monolithic applications, you can scale the individual microservices individually to meet demand. This capability allows you to scale only the required services instead of the entire application, which can allow you to meet application demands while using minimal resources.

1.1.3. OpenShift Dedicated overview

OpenShift Dedicated provides enterprise-ready enhancements to Kubernetes, including the following enhancements:

- OpenShift Dedicated clusters are deployed on AWS environments and can be used as part of a hybrid approach for application management.
- Integrated Red Hat technology. Major components in OpenShift Dedicated come from Red Hat Enterprise Linux and related Red Hat technologies. OpenShift Dedicated benefits from the intense testing and certification initiatives for Red Hat's enterprise quality software.
- Open source development model. Development is completed in the open, and the source code is available from public software repositories. This open collaboration fosters rapid innovation and development.

Although Kubernetes excels at managing your applications, it does not specify or manage platform-level requirements or deployment processes. Powerful and flexible platform management tools and processes are important benefits that OpenShift Dedicated 4 offers. The following sections describe some unique features and benefits of OpenShift Dedicated.

1.1.3.1. Custom operating system

OpenShift Dedicated uses Red Hat Enterprise Linux CoreOS (RHCOS), a container-oriented operating system that combines some of the best features and functions of the CoreOS and Red Hat Atomic Host operating systems. RHCOS is specifically designed for running containerized applications from OpenShift Dedicated and works with new tools to provide fast installation, Operator-based management, and simplified upgrades.

RHCOS includes:

- Ignition, which OpenShift Dedicated uses as a firstboot system configuration for initially bringing up and configuring machines.
- CRI-O, a Kubernetes native container runtime implementation that integrates closely with the operating system to deliver an efficient and optimized Kubernetes experience. CRI-O provides facilities for running, stopping, and restarting containers. It fully replaces the Docker Container Engine , which was used in OpenShift Dedicated 3.
- Kubelet, the primary node agent for Kubernetes that is responsible for launching and monitoring containers.

In OpenShift Dedicated 4, you must use RHCOS for all control plane machines, but you can use Red Hat Enterprise Linux (RHEL) as the operating system for compute machines, which are also known as worker machines. If you choose to use RHEL workers, you must perform more system maintenance than if you use RHCOS for all of the cluster machines.

1.1.3.2. Simplified installation and update process

With OpenShift Dedicated 4, if you have an account with the right permissions, you can deploy a production cluster in supported clouds by running a single command and providing a few values. You can also customize your cloud installation or install your cluster in your data center if you use a supported platform.

For clusters that use RHCOS for all machines, updating, or upgrading, OpenShift Dedicated is a simple, highly-automated process. Because OpenShift Dedicated completely controls the systems and services that run on each machine, including the operating system itself, from a central control plane, upgrades are designed to become automatic events. If your cluster contains RHEL worker machines, the control plane benefits from the streamlined update process, but you must perform more tasks to upgrade the RHEL machines.

1.1.3.3. Other key features

Operators are both the fundamental unit of the OpenShift Dedicated 4 code base and a convenient way to deploy applications and software components for your applications to use. In OpenShift Dedicated, Operators serve as the platform foundation and remove the need for manual upgrades of operating systems and control plane applications. OpenShift Dedicated Operators such as the Cluster Version Operator and Machine Config Operator allow simplified, cluster-wide management of those critical components.

Operator Lifecycle Manager (OLM) and the OperatorHub provide facilities for storing and distributing Operators to people developing and deploying applications.

The Red Hat Quay Container Registry is a Quay.io container registry that serves most of the container images and Operators to OpenShift Dedicated clusters. Quay.io is a public registry version of Red Hat Quay that stores millions of images and tags.

Other enhancements to Kubernetes in OpenShift Dedicated include improvements in software defined networking (SDN), authentication, log aggregation, monitoring, and routing. OpenShift Dedicated also offers a comprehensive web console and the custom OpenShift CLI (**oc**) interface.

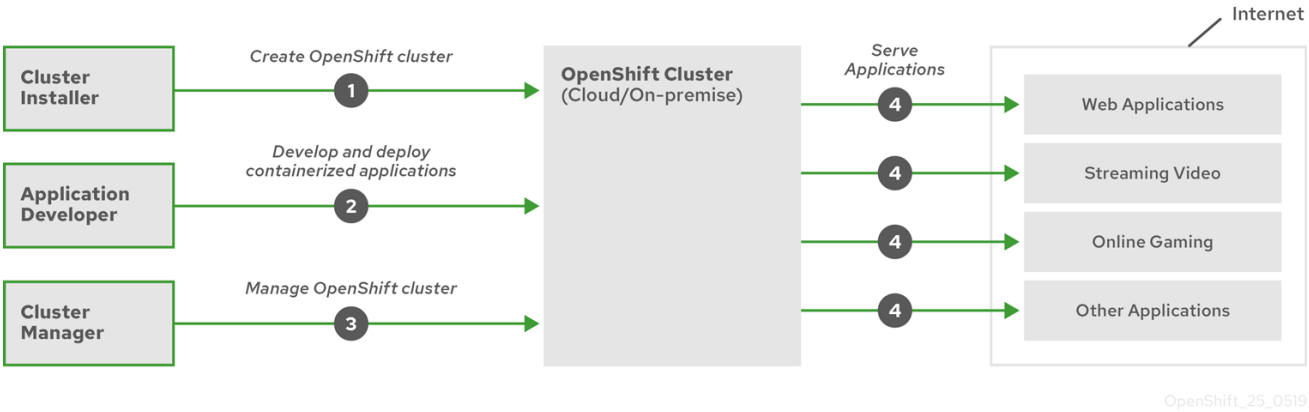
1.1.3.4. OpenShift Dedicated lifecycle

The following figure illustrates the basic OpenShift Dedicated lifecycle:

- Creating an OpenShift Dedicated cluster

- Managing the cluster
- Developing and deploying applications
- Scaling up applications

Figure 1.1. High level OpenShift Dedicated overview



1.1.4. Internet and Telemetry access for OpenShift Dedicated

In OpenShift Dedicated 4, you require access to the internet to install your cluster. The Telemetry service, which runs by default to provide metrics about cluster health and the success of updates, also requires internet access. If your cluster is connected to the internet, Telemetry runs automatically, and your cluster is registered to the Red Hat OpenShift Cluster Manager (OCM) .

Once you confirm that your Red Hat OpenShift Cluster Manager inventory is correct, either maintained automatically by Telemetry or manually using OCM, use subscription watch to track your OpenShift Dedicated subscriptions at the account or multi-cluster level.

You must have internet access to:

- Access the Red Hat OpenShift Cluster Manager page to download the installation program and perform subscription management. If the cluster has internet access and you do not disable Telemetry, that service automatically entitles your cluster.
- Access Quay.io to obtain the packages that are required to install your cluster.
- Obtain the packages that are required to perform cluster updates.

CHAPTER 2. THE OPENSHIFT DEDICATED CONTROL PLANE

2.1. UNDERSTANDING THE OPENSHIFT DEDICATED CONTROL PLANE

The control plane, which is composed of master machines, manages the OpenShift Dedicated cluster. The control plane machines manage workloads on the compute machines, which are also known as worker machines. The cluster itself manages all upgrades to the machines by the actions of the Cluster Version Operator, the Machine Config Operator, and set of individual Operators.

2.1.1. Machine roles in OpenShift Dedicated

OpenShift Dedicated assigns hosts different roles. These roles define the function of the machine within the cluster. The cluster contains definitions for the standard master and worker role types.

NOTE

The cluster also contains the definition for the bootstrap role. Because the bootstrap machine is used only during cluster installation, its function is explained in the cluster installation documentation.

2.1.1.1. Cluster workers

In a Kubernetes cluster, the worker nodes are where the actual workloads requested by Kubernetes users run and are managed. The worker nodes advertise their capacity and the scheduler, which is part of the master services, determines on which nodes to start containers and Pods. Important services run on each worker node, including CRI-O, which is the container engine, Kubelet, which is the service that accepts and fulfills requests for running and stopping container workloads, and a service proxy, which manages communication for pods across workers.

In OpenShift Dedicated, MachineSets control the worker machines. Machines with the worker role drive compute workloads that are governed by a specific machine pool that autoscales them. Because OpenShift Dedicated has the capacity to support multiple machine types, the worker machines are classed as *compute* machines. In this release, the terms "worker machine" and "compute machine" are used interchangeably because the only default type of compute machine is the worker machine. In future versions of OpenShift Dedicated, different types of compute machines, such as infrastructure machines, might be used by default.

2.1.1.2. Cluster masters

In a Kubernetes cluster, the master nodes run services that are required to control the Kubernetes cluster. In OpenShift Dedicated, the master machines are the control plane. They contain more than just the Kubernetes services for managing the OpenShift Dedicated cluster. Because all of the machines with the control plane role are master machines, the terms *master* and *control plane* are used interchangeably to describe them. Instead of being grouped into a MachineSet, master machines are defined by a series of standalone machine API resources. Extra controls apply to master machines to prevent you from deleting all master machines and breaking your cluster.

NOTE

Use three master nodes. Although you can theoretically use any number of master nodes, the number is constrained by etcd quorum due to master static Pods and etcd static Pods working on the same hosts.

Services that fall under the Kubernetes category on the master include the API server, etcd, controller manager server, and HAProxy services.

Table 2.1. Kubernetes services that run on the control plane

Component	Description
API Server	The Kubernetes API server validates and configures the data for Pods, Services, and replication controllers. It also provides a focal point for cluster's shared state.
etcd	etcd stores the persistent master state while other components watch etcd for changes to bring themselves into the specified state.
Controller Manager Server	The Controller Manager Server watches etcd for changes to objects such as replication, namespace, and serviceaccount controller objects, and then uses the API to enforce the specified state. Several such processes create a cluster with one active leader at a time.

Some of these services on the master machines run as systemd services, while others run as static Pods.

Systemd services are appropriate for services that you need to always come up on that particular system shortly after it starts. For master machines, those include sshd, which allows remote login. It also includes services such as:

- The CRI-O container engine (crio), which runs and manages the containers. OpenShift Dedicated 4 uses CRI-O instead of the Docker Container Engine.
- Kubelet (kubelet), which accepts requests for managing containers on the machine from master services.

CRI-O and Kubelet must run directly on the host as systemd services because they need to be running before you can run other containers.

2.1.2. Operators in OpenShift Dedicated

In OpenShift Dedicated, Operators are the preferred method of packaging, deploying, and managing services on the control plane. They also provide advantages to applications that users run. Operators integrate with Kubernetes APIs and CLI tools such as **kubectl** and **oc** commands. They provide the means of watching over an application, performing health checks, managing over-the-air updates, and ensuring that the applications remain in your specified state.

Because CRI-O and the Kubelet run on every node, almost every other cluster function can be managed on the control plane by using Operators. Operators are among the most important components of OpenShift Dedicated 4. Components that are added to the control plane by using Operators include critical networking and credential services.

The Operator that manages the other Operators in an OpenShift Dedicated cluster is the Cluster Version Operator.

OpenShift Dedicated 4 uses different classes of Operators to perform cluster operations and run services on the cluster for your applications to use.

2.1.2.1. Operators managed by OLM

The Cluster Operator Lifecycle Management (OLM) component manages Operators that are available for use in applications. It does not manage the Operators that comprise OpenShift Dedicated. OLM is a framework that manages Kubernetes-native applications as Operators. Instead of managing Kubernetes manifests, it manages Kubernetes Operators. OLM manages two classes of Operators, Red Hat Operators and certified Operators.

Some Red Hat Operators drive the cluster functions, like the scheduler and problem detectors. Others are provided for you to manage yourself and use in your applications, like etcd. OpenShift Dedicated also offers certified Operators, which the community built and maintains. These certified Operators provide an API layer to traditional applications so you can manage the application through Kubernetes constructs.

2.1.2.2. About the OpenShift Dedicated update service

The OpenShift Dedicated update service is the hosted service that provides over-the-air updates to both OpenShift Dedicated and Red Hat Enterprise Linux CoreOS (RHCOS). It provides a graph, or diagram that contain *vertices* and the *edges* that connect them, of component Operators. The edges in the graph show which versions you can safely update to, and the vertices are update payloads that specify the intended state of the managed cluster components.

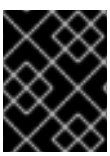
The Cluster Version Operator (CVO) in your cluster checks with the OpenShift Dedicated update service to see the valid updates and update paths based on current component versions and information in the graph. When you request an update, the OpenShift Dedicated CVO uses the release image for that update to upgrade your cluster. The release artifacts are hosted in Quay as container images.

To allow the OpenShift Dedicated update service to provide only compatible updates, a release verification pipeline exists to drive automation. Each release artifact is verified for compatibility with supported cloud platforms and system architectures as well as other component packages. After the pipeline confirms the suitability of a release, the OpenShift Dedicated update service notifies you that it is available.

IMPORTANT

Because the update service displays all valid updates, you must not force an update to a version that the update service does not display.

During continuous update mode, two controllers run. One continuously updates the payload manifests, applies them to the cluster, and outputs the status of the controlled rollout of the Operators, whether they are available, upgrading, or failed. The second controller polls the OpenShift Dedicated update service to determine if updates are available.



IMPORTANT

Reverting your cluster to a previous version, or a rollback, is not supported. Only upgrading to a newer version is supported.

During the upgrade process, the Machine Config Operator (MCO) applies the new configuration to your cluster machines. It cordons the number of nodes that is specified by the **maxUnavailable** field on the machine configuration pool and marks them as unavailable. By default, this value is set to **1**. It then

applies the new configuration and reboots the machine. If you use Red Hat Enterprise Linux (RHEL) machines as workers, the MCO does not update the kubelet on these machines because you must update the OpenShift API on them first. Because the specification for the new version is applied to the old kubelet, the RHEL machine cannot return to the **Ready** state. You cannot complete the update until the machines are available. However, the maximum number of nodes that are unavailable is set to ensure that normal cluster operations are likely to continue with that number of machines out of service.

2.1.2.3. Understanding the Machine Config Operator

OpenShift Dedicated 4 integrates both operating system and cluster management. Because the cluster manages its own updates, including updates to Red Hat Enterprise Linux CoreOS (RHCOS) on cluster nodes, OpenShift Dedicated provides an opinionated lifecycle management experience that simplifies the orchestration of node upgrades.

OpenShift Dedicated employs three DaemonSets and controllers to simplify node management. These DaemonSets orchestrate operating system updates and configuration changes to the hosts by using standard Kubernetes-style constructs. They include:

- The **machine-config-controller**, which coordinates machine upgrades from the control plane. It monitors all of the cluster nodes and orchestrates their configuration updates.
- The **machine-config-daemon** DaemonSet, which runs on each node in the cluster and updates a machine to configuration defined by MachineConfig as instructed by the MachineConfigController. When the node sees a change, it drains off its pods, applies the update, and reboots. These changes come in the form of Ignition configuration files that apply the specified machine configuration and control kubelet configuration. The update itself is delivered in a container. This process is key to the success of managing OpenShift Dedicated and RHCOS updates together.
- The **machine-config-server** DaemonSet, which provides the Ignition config files to master nodes as they join the cluster.

The machine configuration is a subset of the Ignition configuration. The **machine-config-daemon** reads the machine configuration to see if it needs to do an OSTree update or if it must apply a series of systemd kubelet file changes, configuration changes, or other changes to the operating system or OpenShift Dedicated configuration.

When you perform node management operations, you create or modify a KubeletConfig Custom Resource (CR).

CHAPTER 3. UNDERSTANDING OPENSHIFT DEDICATED DEVELOPMENT

To fully leverage the capability of containers when developing and running enterprise-quality applications, ensure your environment is supported by tools that allow containers to be:

- Created as discrete microservices that can be connected to other containerized, and non-containerized, services. For example, you might want to join your application with a database or attach a monitoring application to it.
- Resilient, so if a server crashes or needs to go down for maintenance or to be decommissioned, containers can start on another machine.
- Automated to pick up code changes automatically and then start and deploy new versions of themselves.
- Scaled up, or replicated, to have more instances serving clients as demand increases and then spun down to fewer instances as demand declines.
- Run in different ways, depending on the type of application. For example, one application might run once a month to produce a report and then exit. Another application might need to run constantly and be highly available to clients.
- Managed so you can watch the state of your application and react when something goes wrong.

Containers' widespread acceptance, and the resulting requirements for tools and methods to make them enterprise-ready, resulted in many options for them.

The rest of this section explains options for assets you can create when you build and deploy containerized Kubernetes applications in OpenShift Dedicated. It also describes which approaches you might use for different kinds of applications and development requirements.

3.1. ABOUT DEVELOPING CONTAINERIZED APPLICATIONS

You can approach application development with containers in many ways, and different approaches might be more appropriate for different situations. To illustrate some of this variety, the series of approaches that is presented starts with developing a single container and ultimately deploys that container as a mission-critical application for a large enterprise. These approaches show different tools, formats, and methods that you can employ with containerized application development. This topic describes:

- Building a simple container and storing it in a registry
- Creating a Kubernetes manifest and saving it to a Git repository
- Making an Operator to share your application with others

3.2. BUILDING A SIMPLE CONTAINER

You have an idea for an application and you want to containerize it.

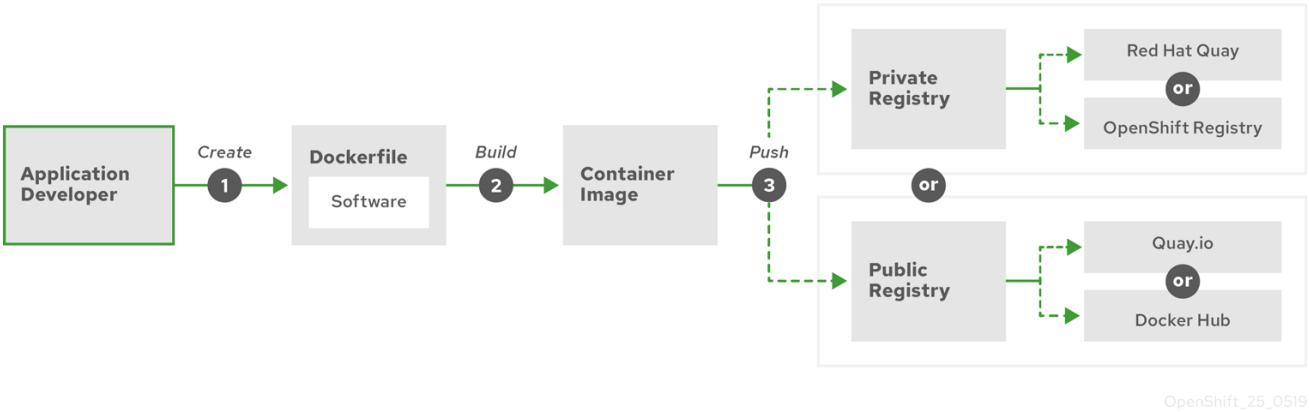
First you require a tool for building a container, like buildah or docker, and a file that describes what goes in your container, which is typically a Dockerfile.

Next, you require a location to push the resulting container image so you can pull it to run anywhere you want it to run. This location is a container registry.

Some examples of each of these components are installed by default on most Linux operating systems, except for the Dockerfile, which you provide yourself.

The following diagram displays the process of building and pushing an image:

Figure 3.1. Create a simple containerized application and push it to a registry



If you use a computer that runs Red Hat Enterprise Linux (RHEL) as the operating system, the process of creating a containerized application requires the following steps:

1. Install container build tools: RHEL contains a set of tools that includes podman, buildah, and skopeo that you use to build and manage containers.

2. Create a Dockerfile to combine base image and software: Information about building your container goes into a file that is named **Dockerfile**. In that file, you identify the base image you build from, the software packages you install, and the software you copy into the container. You also identify parameter values like network ports that you expose outside the container and volumes that you mount inside the container. Put your Dockerfile and the software you want to containerized in a directory on your RHEL system.

3. Run buildah or docker build: Run the **buildah build-using-dockerfile** or the **docker build** command to pull you chosen base image to the local system and creates a container image that is stored locally. You can also build container without a Dockerfile by using buildah.

4. Tag and push to a registry: Add a tag to your new container image that identifies the location of the registry in which you want to store and share your container. Then push that image to the registry by running the **podman push** or **docker push** command.

5. Pull and run the image: From any system that has a container client tool, such as podman or docker, run a command that identifies your new image. For example, run the **podman run <image_name>** or **docker run <image_name>** command. Here **<image_name>** is the name of your new container image, which resembles **quay.io/myrepo/myapp:latest**. The registry might require credentials to push and pull images.

3.2.1. Container build tool options

While the Docker Container Engine and **docker** command are popular tools to work with containers, with RHEL and many other Linux systems, you can instead choose a different set of container tools that includes podman, skopeo, and buildah. You can still use Docker Container Engine tools to create containers that will run in OpenShift Dedicated and any other container platform.

Building and managing containers with buildah, podman, and skopeo results in industry standard container images that include features tuned specifically for ultimately deploying those containers in OpenShift Dedicated or other Kubernetes environments. These tools are daemonless and can be run without root privileges, so there is less overhead in running them.

When you ultimately run your containers in OpenShift Dedicated, you use the CRI-O container engine. CRI-O runs on every worker and master machine in an OpenShift Dedicated cluster, but CRI-O is not yet supported as a standalone runtime outside of OpenShift Dedicated.

3.2.2. Base image options

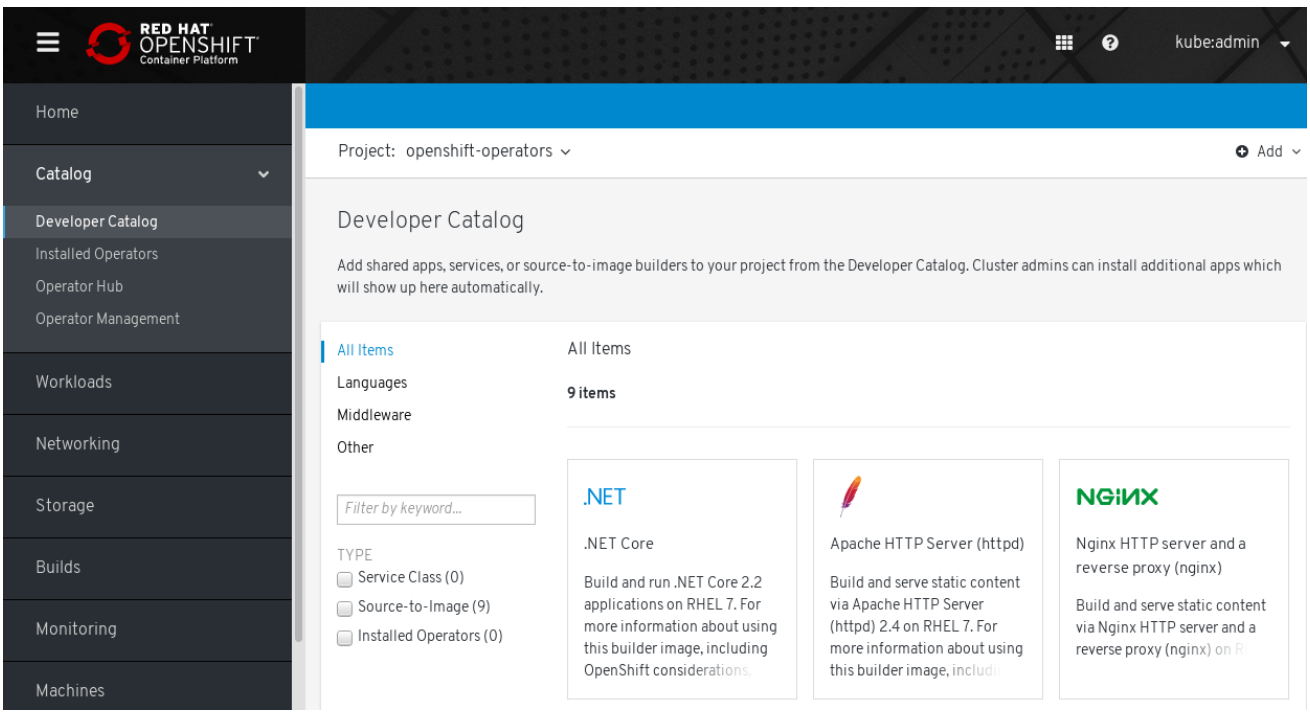
The base image you choose to build your application on contains a set of software that resembles a Linux system to your application. When you build your own image, your software is placed into that file system and sees that file system as though it were looking at its operating system. Choosing this base image has major impact on how secure, efficient and upgradeable your container is in the future.

Red Hat provides a new set of base images referred to as Red Hat Universal Base Images (UBI). These images are based on Red Hat Enterprise Linux and are similar to base images that Red Hat has offered in the past, with one major difference: they are freely redistributable without a Red Hat subscription. As a result, you can build your application on UBI images without having to worry about how they are shared or the need to create different images for different environments.

These UBI images have standard, init, and minimal versions. You can also use the Red Hat Software Collections images as a foundation for applications that rely on specific runtime environments such as Node.js, Perl, or Python. Special versions of some of these runtime base images referred to as Source-to-image (S2I) images. With S2I images, you can insert your code into a base image environment that is ready to run that code.

S2I images are available for you to use directly from the OpenShift Dedicated web UI by selecting **Catalog → Developer Catalog**, as shown in the following figure:

Figure 3.2. Choose S2I base images for apps that need specific runtimes



3.2.3. Registry options

Container Registries are where you store container images so you can share them with others and make them available to the platform where they ultimately run. You can select large, public container registries that offer free accounts or a premium version that offer more storage and special features. You can also install your own registry that can be exclusive to your organization or selectively shared with others.

To get Red Hat images and certified partner images, you can draw from the Red Hat Registry. The Red Hat Registry is represented by two locations: **registry.access.redhat.com**, which is unauthenticated and deprecated, and **registry.redhat.io**, which requires authentication. You can learn about the Red Hat and partner images in the Red Hat Registry from the Red Hat Container Catalog . Besides listing Red Hat container images, it also shows extensive information about the contents and quality of those images, including health scores that are based on applied security updates.

Large, public registries include Docker Hub and Quay.io. The Quay.io registry is owned and managed by Red Hat. Many of the components used in OpenShift Dedicated are stored in Quay.io, including container images and the Operators that are used to deploy OpenShift Dedicated itself. Quay.io also offers the means of storing other types of content, including Helm Charts.

If you want your own, private container registry, OpenShift Dedicated itself includes a private container registry that is installed with OpenShift Dedicated and runs on its cluster. Red Hat also offers a private version of the Quay.io registry called Red Hat Quay. Red Hat Quay includes geo replication, Git build triggers, Clair image scanning, and many other features.

All of the registries mentioned here can require credentials to download images from those registries. Some of those credentials are presented on a cluster-wide basis from OpenShift Dedicated, while other credentials can be assigned to individuals.

3.3. CREATING A KUBERNETES MANIFEST FOR OPENSHIFT DEDICATED

While the container image is the basic building block for a containerized application, more information is required to manage and deploy that application in a Kubernetes environment such as OpenShift Dedicated. The typical next steps after you create an image are to:

- Understand the different resources you work with in Kubernetes manifests
- Make some decisions about what kind of an application you are running
- Gather supporting components
- Create a manifest and store that manifest in a Git repository so you can store it in a source versioning system, audit it, track it, promote and deploy it to the next environment, roll it back to earlier versions, if necessary, and share it with others

3.3.1. About Kubernetes Pods and services

While the container image is the basic unit with docker, the basic units that Kubernetes works with are called Pods. Pods represent the next step in building out an application. A Pod can contain one or more than one container. The key is that the Pod is the single unit that you deploy, scale, and manage.

Scalability and namespaces are probably the main items to consider when determining what goes in a Pod. For ease of deployment, you might want to deploy a container in a Pod and include its own logging and monitoring container in the Pod. Later, when you run the Pod and need to scale up an additional instance, those other containers are scaled up with it. For namespaces, containers in a Pod share the same network interfaces, shared storage volumes, and resource limitations, such as memory and CPU,

which makes it easier to manage the contents of the Pod as a single unit. Containers in a Pod can also communicate with each other by using standard inter-process communications, such as System V semaphores or POSIX shared memory.

While individual Pods represent a scalable unit in Kubernetes, a service provides a means of grouping together a set of Pods to create a complete, stable application that can complete tasks such as load balancing. A service is also more permanent than a Pod because the service remains available from the same IP address until you delete it. When the service is in use, it is requested by name and the OpenShift Dedicated cluster resolves that name into the IP addresses and ports where you can reach the Pods that compose the service.

By their nature, containerized applications are separated from the operating systems where they run and, by extension, their users. Part of your Kubernetes manifest describes how to expose the application to internal and external networks by defining network policies that allow fine-grained control over communication with your containerized applications. To connect incoming requests for HTTP, HTTPS, and other services from outside your cluster to services inside your cluster, you can use an Ingress resource.

If your container requires on-disk storage instead of database storage, which might be provided through a service, you can add volumes to your manifests to make that storage available to your Pods. You can configure the manifests to create persistent volumes (PVs) or dynamically create volumes that are added to your Pod definitions.

After you define a group of Pods that compose your application, you can define those Pods in deployments and deploymentconfigs.

3.3.2. Application types

Next, consider how your application type influences how to run it.

Kubernetes defines different types of workloads that are appropriate for different kinds of applications. To determine the appropriate workload for your application, consider if the application is:

- Meant to run to completion and be done. An example is an application that starts up to produce a report and exits when the report is complete. The application might not run again then for a month. Suitable OpenShift Dedicated objects for these types of applications include Jobs and CronJob objects.

- Expected to run continuously. For long-running applications, you can write a Deployment or a DeploymentConfig.

- Required to be highly available. If your application requires high availability, then you want to size your deployment to have more than one instance. A Deployment or DeploymentConfig can incorporate a ReplicaSet for that type of application. With ReplicaSets, Pods run across multiple nodes to make sure the application is always available, even if a worker goes down.

- Need to run on every node. Some types of Kubernetes applications are intended to run in the cluster itself on every master or worker node. DNS and monitoring applications are examples of applications that need to run continuously on every node. You can run this type of application as a DaemonSet. You can also run a DaemonSet on a subset of nodes, based on node labels.

- Require life-cycle management. When you want to hand off your application so that others can use it, consider creating an Operator. Operators let you build in intelligence, so it can handle things like backups and upgrades automatically. Coupled with the Operator Lifecycle Manager (OLM), cluster managers can expose Operators to selected namespaces so that users in the cluster can run them.

- Have identity or numbering requirements. An application might have identity requirements or numbering requirements. For example, you might be required to run exactly three instances of the application and to name the instances **0**, **1**, and **2**. A StatefulSet is suitable for this application. StatefulSets are most useful for applications that require independent storage, such as databases and zookeeper clusters.

3.3.3. Available supporting components

The application you write might need supporting components, like a database or a logging component. To fulfill that need, you might be able to obtain the required component from the following Catalogs that are available in the OpenShift Dedicated web console:

- OperatorHub, which is available in each OpenShift Dedicated 4 cluster. The OperatorHub makes Operators available from Red Hat, certified Red Hat partners, and community members to the cluster operator. The cluster operator can make those Operators available in all or selected namespaces in the cluster, so developers can launch them and configure them with their applications.

- Service Catalog, which offers alternatives to Operators. While deploying Operators is the preferred method of getting packaged applications in OpenShift Dedicated, there are some reasons why you might want to use the Service Catalog to get supporting applications for your own application. You might want to use the Service Catalog if you are an existing OpenShift Dedicated 3 customer and are invested in Service Catalog applications or if you already have a Cloud Foundry environment from which you are interested in consuming brokers from other ecosystems.

- Templates, which are useful for a one-off type of application, where the lifecycle of a component is not important after it is installed. A template provides an easy way to get started developing a Kubernetes application with minimal overhead. A template can be a list of resource definitions, which could be deployments, services, routes, or other objects. If you want to change names or resources, you can set these values as parameters in the template. The Template Service Broker Operator is a service broker that you can use to instantiate your own templates. You can also install templates directly from the command line.

You can configure the supporting Operators, Service Catalog applications, and templates to the specific needs of your development team and then make them available in the namespaces in which your developers work. Many people add shared templates to the **openshift** namespace because it is accessible from all other namespaces.

3.3.4. Applying the manifest

Kubernetes manifests let you create a more complete picture of the components that make up your Kubernetes applications. You write these manifests as YAML files and deploy them by applying them to the cluster, for example, by running the **oc apply** command.

3.3.5. Next steps

At this point, consider ways to automate your container development process. Ideally, you have some sort of CI pipeline that builds the images and pushes them to a registry. In particular, a GitOps pipeline integrates your container development with the Git repositories that you use to store the software that is required to build your applications.

The workflow to this point might look like:

- Day 1: You write some YAML. You then run the **oc apply** command to apply that YAML to the cluster and test that it works.
- Day 2: You put your YAML container configuration file into your own Git repository. From there, people who want to install that app, or help you improve it, can pull down the YAML and apply it to their cluster to run the app.
- Day 3: Consider writing an Operator for your application.

3.4. DEVELOP FOR OPERATORS

Packaging and deploying your application as an Operator might be preferred if you make your application available for others to run. As noted earlier, Operators add a lifecycle component to your application that acknowledges that the job of running an application is not complete as soon as it is installed.

When you create an application as an Operator, you can build in your own knowledge of how to run and maintain the application. You can build in features for upgrading the application, backing it up, scaling it, or keeping track of its state. If you configure the application correctly, maintenance tasks, like updating the Operator, can happen automatically and invisibly to the Operator's users.

An example of a useful Operator is one that is set up to automatically back up data at particular times. Having an Operator manage an application's backup at set times can save a system administrator from remembering to do it.

Any application maintenance that has traditionally been completed manually, like backing up data or rotating certificates, can be completed automatically with an Operator.