OpenShift Container Platform 4.12

Getting started

Getting started in OpenShift Container Platform

Last Updated: 2024-05-23

OpenShift Container Platform 4.12 Getting started

Getting started in OpenShift Container Platform

Legal Notice

Abstract

This document provides information to help you get started in OpenShift Container Platform. This includes definitions for common terms found in Kubernetes and OpenShift Container Platform. This also contains a walkthrough of the OpenShift Container Platform web console, as well as creating and building applications by using the command-line interface.

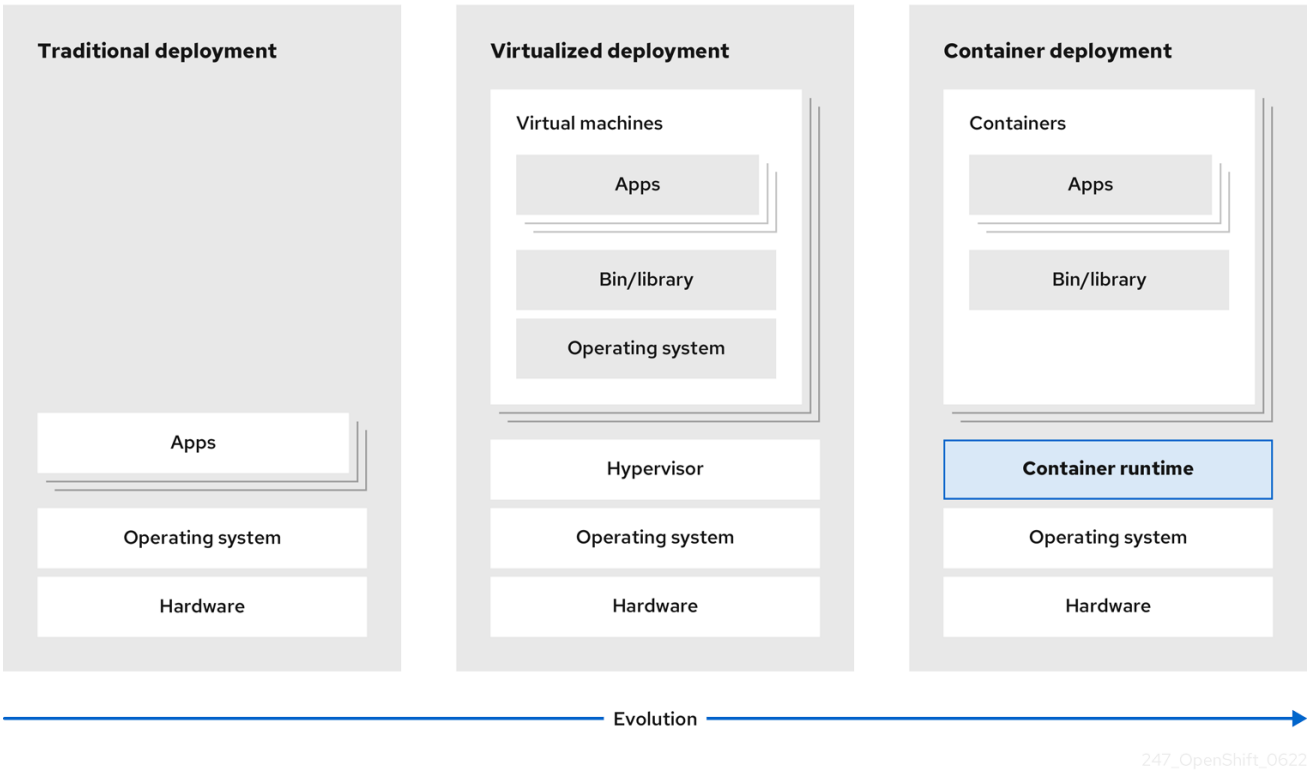
Table of Contents

CHAPTER 1. KUBERNETES OVERVIEW

Kubernetes is an open source container orchestration tool developed by Google. You can run and manage container-based workloads by using Kubernetes. The most common Kubernetes use case is to deploy an array of interconnected microservices, building an application in a cloud native way. You can create Kubernetes clusters that can span hosts across on-premise, public, private, or hybrid clouds.

Traditionally, applications were deployed on top of a single operating system. With virtualization, you can split the physical host into several virtual hosts. Working on virtual instances on shared resources is not optimal for efficiency and scalability. Because a virtual machine (VM) consumes as many resources as a physical machine, providing resources to a VM such as CPU, RAM, and storage can be expensive. Also, you might see your application degrading in performance due to virtual instance usage on shared resources.

Figure 1.1. Evolution of container technologies for classical deployments



To solve this problem, you can use containerization technologies that segregate applications in a containerized environment. Similar to a VM, a container has its own filesystem, vCPU, memory, process space, dependencies, and more. Containers are decoupled from the underlying infrastructure, and are portable across clouds and OS distributions. Containers are inherently much lighter than a fully-featured OS, and are lightweight isolated processes that run on the operating system kernel. VMs are slower to boot, and are an abstraction of physical hardware. VMs run on a single machine with the help of a hypervisor.

You can perform the following actions by using Kubernetes:

- Sharing resources
- Orchestrating containers across multiple hosts
- Installing new hardware configurations
- Running health checks and self-healing applications

- Scaling containerized applications

1.1. KUBERNETES COMPONENTS

Table 1.1. Kubernetes components

Component	Purpose
kube-proxy	Runs on every node in the cluster and maintains the network traffic between the Kubernetes resources.
kube-controller-manager	Governs the state of the cluster.
kube-scheduler	Allocates pods to nodes.
etcd	Stores cluster data.
kube-apiserver	Validates and configures data for the API objects.
kubelet	Runs on nodes and reads the container manifests. Ensures that the defined containers have started and are running.
kubectl	Allows you to define how you want to run workloads. Use the **kubectl** command to interact with the **kube-apiserver**.
Node	Node is a physical machine or a VM in a Kubernetes cluster. The control plane manages every node and schedules pods across the nodes in the Kubernetes cluster.
container runtime	container runtime runs containers on a host operating system. You must install a container runtime on each node so that pods can run on the node.
Persistent storage	Stores the data even after the device is shut down. Kubernetes uses persistent volumes to store the application data.
container-registry	Stores and accesses the container images.
Pod	The pod is the smallest logical unit in Kubernetes. A pod contains one or more containers to run in a worker node.

1.2. KUBERNETES RESOURCES

A custom resource is an extension of the Kubernetes API. You can customize Kubernetes clusters by using custom resources. Operators are software extensions which manage applications and their components with the help of custom resources. Kubernetes uses a declarative model when you want a fixed desired result while dealing with cluster resources. By using Operators, Kubernetes defines its states in a declarative way. You can modify the Kubernetes cluster resources by using imperative

commands. An Operator acts as a control loop which continuously compares the desired state of resources with the actual state of resources and puts actions in place to bring reality in line with the desired state.

Figure 1.2. Kubernetes cluster overview

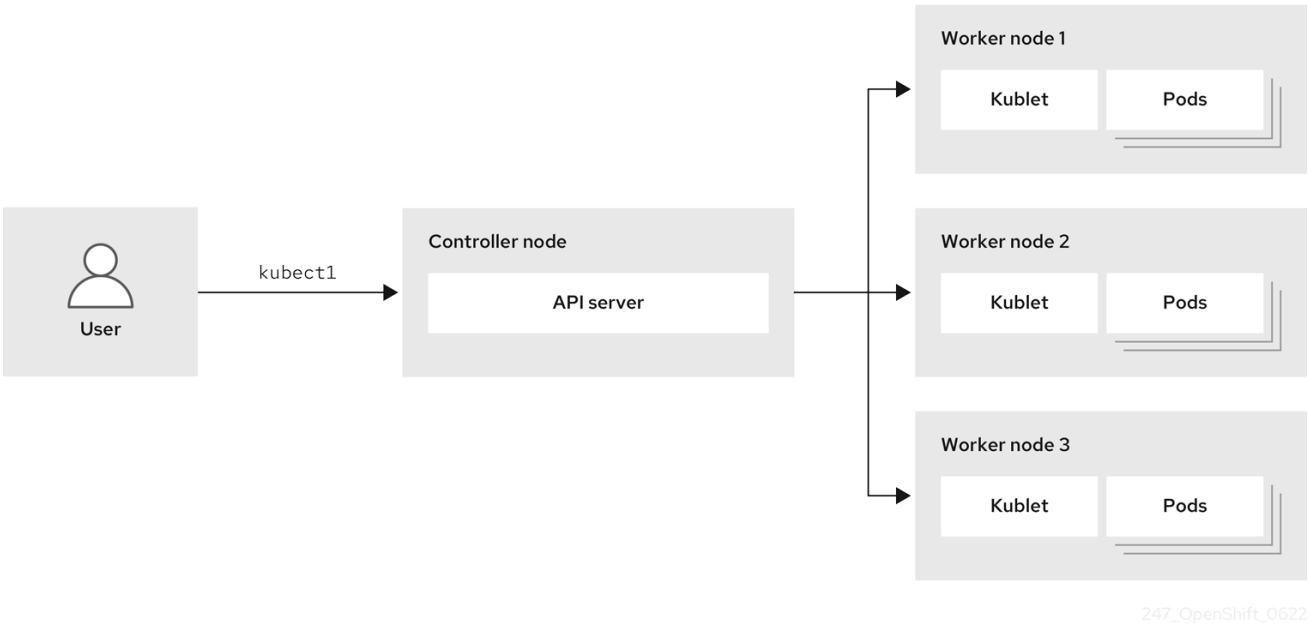
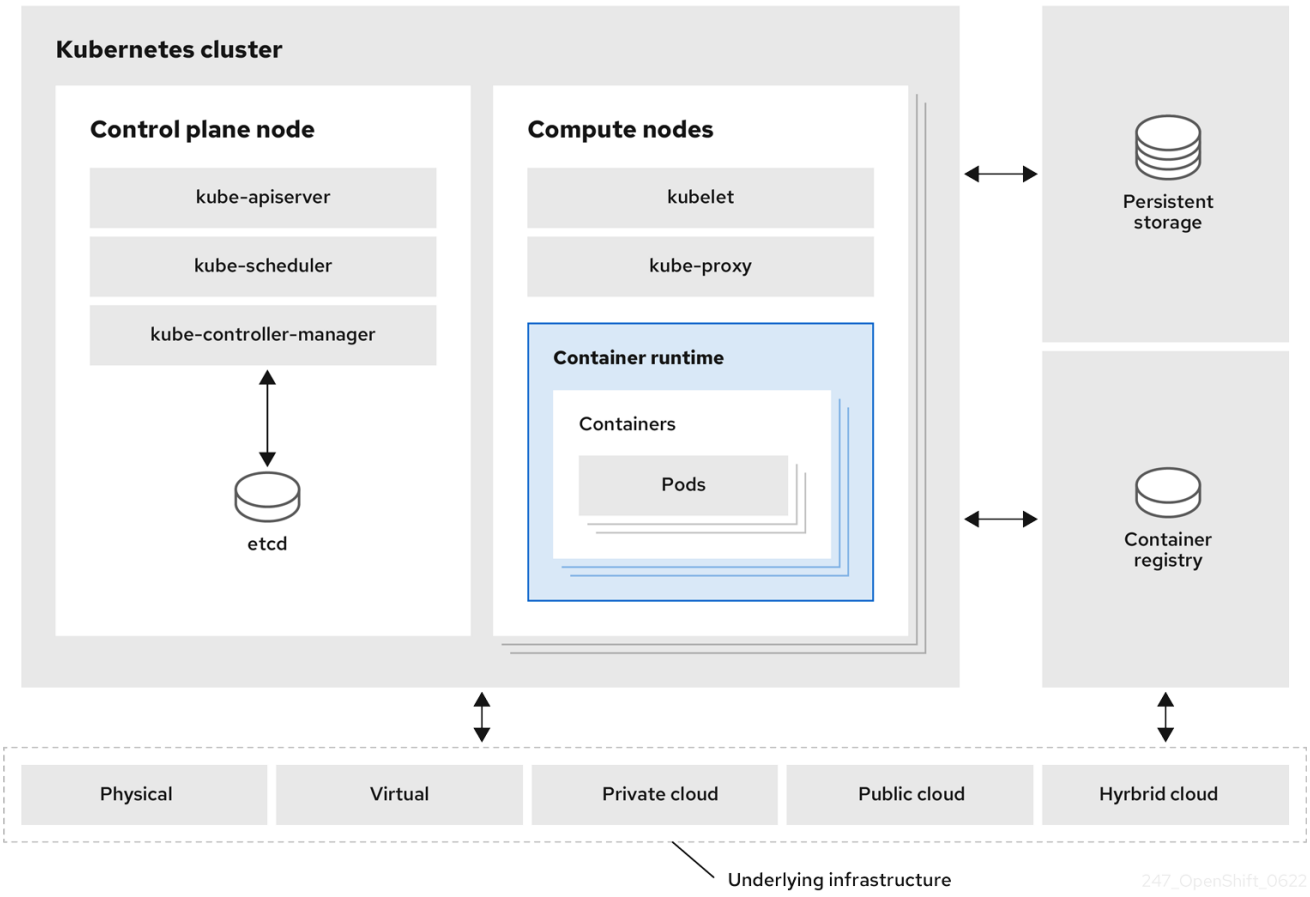


Table 1.2. Kubernetes Resources

Resource	Purpose
Service	Kubernetes uses services to expose a running application on a set of pods.
ReplicaSets	Kubernetes uses the **ReplicaSets** to maintain the constant pod number.
Deployment	A resource object that maintains the life cycle of an application.

Kubernetes is a core component of an OpenShift Container Platform. You can use OpenShift Container Platform for developing and running containerized applications. With its foundation in Kubernetes, the OpenShift Container Platform incorporates the same technology that serves as the engine for massive telecommunications, streaming video, gaming, banking, and other applications. You can extend your containerized applications beyond a single cloud to on-premise and multi-cloud environments by using the OpenShift Container Platform.

Figure 1.3. Architecture of Kubernetes



A cluster is a single computational unit consisting of multiple nodes in a cloud environment. A Kubernetes cluster includes a control plane and worker nodes. You can run Kubernetes containers across various machines and environments. The control plane node controls and maintains the state of a cluster. You can run the Kubernetes application by using worker nodes. You can use the Kubernetes namespace to differentiate cluster resources in a cluster. Namespace scoping is applicable for resource objects, such as deployment, service, and pods. You cannot use namespace for cluster-wide resource objects such as storage class, nodes, and persistent volumes.

1.3. KUBERNETES CONCEPTUAL GUIDELINES

Before getting started with the OpenShift Container Platform, consider these conceptual guidelines of Kubernetes:

- Start with one or more worker nodes to run the container workloads.
- Manage the deployment of those workloads from one or more control plane nodes.
- Wrap containers in a deployment unit called a pod. By 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.

The API to OpenShift Container Platform cluster is 100% Kubernetes. Nothing changes between a container running on any other Kubernetes and running on OpenShift Container Platform. No changes to the application. OpenShift Container Platform brings added-value features to provide enterprise-ready enhancements to Kubernetes. OpenShift Container Platform CLI tool (**oc**) is compatible with **kubectl**. While the Kubernetes API is 100% accessible within OpenShift Container Platform, the **kubectl** command-line lacks many features that could make it more user-friendly. OpenShift Container Platform offers a set of features and command-line tool like **oc**. 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 Container Platform offers. You must add authentication, networking, security, monitoring, and logs management to your containerization platform.

CHAPTER 2. OPENSHIFT CONTAINER PLATFORM OVERVIEW

OpenShift Container Platform is a cloud-based Kubernetes container platform. The foundation of OpenShift Container Platform is based on Kubernetes and therefore shares the same technology. 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.

OpenShift Container Platform enables you to do the following:

- Provide developers and IT organizations with cloud application platforms that can be used for deploying applications on secure and scalable resources.
- Require minimal configuration and management overhead.
- Bring the Kubernetes platform to customer data centers and cloud.
- Meet security, privacy, compliance, and governance requirements.

With its foundation in Kubernetes, OpenShift Container Platform 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.

2.1. GLOSSARY OF COMMON TERMS FOR OPENSHIFT CONTAINER PLATFORM

This glossary defines common Kubernetes and OpenShift Container Platform terms.

Kubernetes

Kubernetes is an open source container orchestration engine for automating deployment, scaling, and management of containerized applications.

Containers

Containers are application instances and components that run in OCI-compliant containers on the worker nodes. A container is the runtime of an Open Container Initiative (OCI)-compliant image. An image is a binary application. A worker node can run many containers. A node capacity is related to memory and CPU capabilities of the underlying resources whether they are cloud, hardware, or virtualized.

Pod

A pod is one or more containers deployed together on one host. It consists of a colocated group of containers with shared resources such as volumes and IP addresses. A pod is also the smallest compute unit defined, deployed, and managed.
In OpenShift Container Platform, pods replace individual application containers as the smallest deployable unit.

Pods are the orchestrated unit in OpenShift Container Platform. OpenShift Container Platform schedules and runs all containers in a pod on the same node. Complex applications are made up of many pods, each with their own containers. They interact externally and also with another inside the OpenShift Container Platform environment.

Replica set and replication controller

The Kubernetes replica set and the OpenShift Container Platform replication controller are both available. The job of this component is to ensure the specified number of pod replicas are running at all times. If pods exit or are deleted, the replica set or replication controller starts more. If more pods

are running than needed, the replica set deletes as many as necessary to match the specified number of replicas.

Deployment and DeploymentConfig

OpenShift Container Platform implements both Kubernetes **Deployment** objects and OpenShift Container Platform **DeploymentConfigs** objects. Users may select either.

Deployment objects control how an application is rolled out as pods. They identify the name of the container image to be taken from the registry and deployed as a pod on a node. They set the number of replicas of the pod to deploy, creating a replica set to manage the process. The labels indicated instruct the scheduler onto which nodes to deploy the pod. The set of labels is included in the pod definition that the replica set instantiates.

Deployment objects are able to update the pods deployed onto the worker nodes based on the version of the **Deployment** objects and the various rollout strategies for managing acceptable application availability. OpenShift Container Platform **DeploymentConfig** objects add the additional features of change triggers, which are able to automatically create new versions of the **Deployment** objects as new versions of the container image are available, or other changes.

Service

A service defines a logical set of pods and access policies. It provides permanent internal IP addresses and hostnames for other applications to use as pods are created and destroyed.

Service layers connect application components together. For example, a front-end web service connects to a database instance by communicating with its service. Services allow for simple internal load balancing across application components. OpenShift Container Platform automatically injects service information into running containers for ease of discovery.

Route

A route is a way to expose a service by giving it an externally reachable hostname, such as www.example.com. Each route consists of a route name, a service selector, and optionally a security configuration. A router can consume a defined route and the endpoints identified by its service to provide a name that lets external clients reach your applications. While it is easy to deploy a complete multi-tier application, traffic from anywhere outside the OpenShift Container Platform environment cannot reach the application without the routing layer.

Build

A build is the process of transforming input parameters into a resulting object. Most often, the process is used to transform input parameters or source code into a runnable image. A **BuildConfig** object is the definition of the entire build process. OpenShift Container Platform leverages Kubernetes by creating containers from build images and pushing them to the integrated registry.

Project

OpenShift Container Platform uses projects to allow groups of users or developers to work together, serving as the unit of isolation and collaboration. It defines the scope of resources, allows project administrators and collaborators to manage resources, and restricts and tracks the user's resources with quotas and limits.

A project is a Kubernetes namespace with additional annotations. It is the central vehicle for managing access to resources for regular users. A project lets a community of users organize and manage their content in isolation from other communities. Users must receive access to projects from administrators. But cluster administrators can allow developers to create their own projects, in which case users automatically have access to their own projects.

Each project has its own set of objects, policies, constraints, and service accounts.

Projects are also known as namespaces.

Operators

An Operator is a Kubernetes-native application. The goal of an Operator is to put operational knowledge into software. Previously this knowledge only resided in the minds of administrators, various combinations or shell scripts or automation software such as Ansible. It was outside your Kubernetes cluster and hard to integrate. With Operators, all of this changes.
Operators are purpose-built for your applications. They implement and automate common Day 1 activities such as installation and configuration as well as Day 2 activities such as scaling up and down, reconfiguration, updates, backups, fail overs, and restores in a piece of software running inside your Kubernetes cluster by integrating natively with Kubernetes concepts and APIs. This is called a Kubernetes-native application.

With Operators, applications must not be treated as a collection of primitives, such as pods, deployments, services, or config maps. Instead, Operators should be treated as a single object that exposes the options that make sense for the application.

2.2. UNDERSTANDING OPENSHIFT CONTAINER PLATFORM

OpenShift Container Platform is a Kubernetes environment for managing the lifecycle of container-based applications and their dependencies on various computing platforms, such as bare metal, virtualized, on-premise, and in cloud. OpenShift Container Platform deploys, configures and manages containers. OpenShift Container Platform offers usability, stability, and customization of its components.

OpenShift Container Platform utilises a number of computing resources, known as nodes. A node has a lightweight, secure operating system based on Red Hat Enterprise Linux (RHEL), known as Red Hat Enterprise Linux CoreOS (RHCOS).

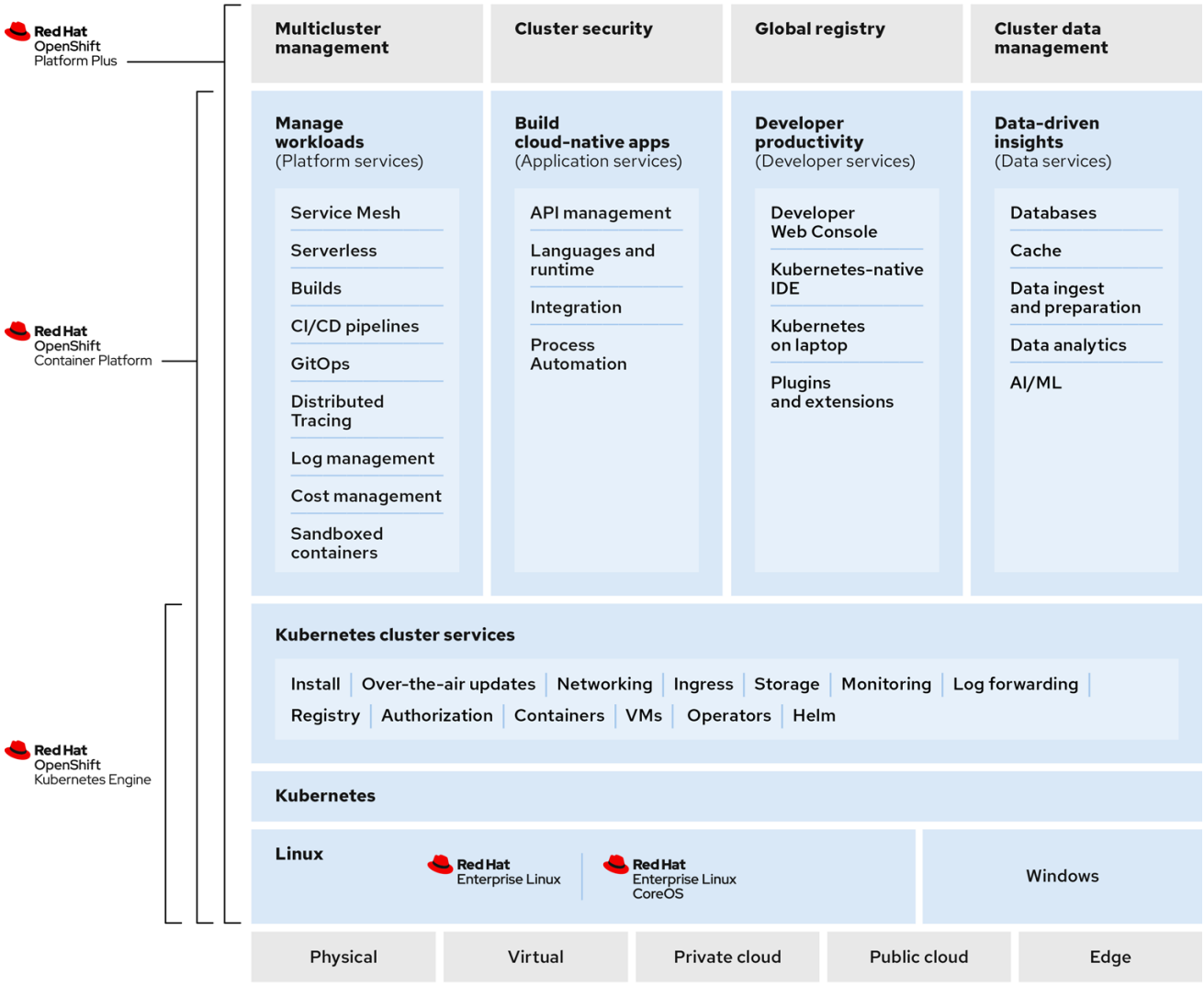
After a node is booted and configured, it obtains a container runtime, such as CRI-O or Docker, for managing and running the images of container workloads scheduled to it. The Kubernetes agent, or kubelet schedules container workloads on the node. The kubelet is responsible for registering the node with the cluster and receiving the details of container workloads.

OpenShift Container Platform configures and manages the networking, load balancing and routing of the cluster. OpenShift Container Platform adds cluster services for monitoring the cluster health and performance, logging, and for managing upgrades.

The container image registry and OperatorHub provide Red Hat certified products and community built softwares for providing various application services within the cluster. These applications and services manage the applications deployed in the cluster, databases, frontends and user interfaces, application runtimes and business automation, and developer services for development and testing of container applications.

You can manage applications within the cluster either manually by configuring deployments of containers running from pre-built images or through resources known as Operators. You can build custom images from pre-build images and source code, and store these custom images locally in an internal, private or public registry.

The Multicluster Management layer can manage multiple clusters including their deployment, configuration, compliance and distribution of workloads in a single console.



2.3. INSTALLING OPENSHIFT CONTAINER PLATFORM

The OpenShift Container Platform installation program offers you flexibility. You can use the installation program to deploy a cluster on infrastructure that the installation program provisions and the cluster maintains or deploy a cluster on infrastructure that you prepare and maintain.

For more information about the installation process, the supported platforms, and choosing a method of installing and preparing your cluster, see the following:

- OpenShift Container Platform installation overview
- Installation process
- Supported platforms for OpenShift Container Platform clusters
- Selecting a cluster installation type

2.3.1. OpenShift Local overview

OpenShift Local supports rapid application development to get started building OpenShift Container Platform clusters. OpenShift Local is designed to run on a local computer to simplify setup and testing, and to emulate the cloud development environment locally with all of the tools needed to develop

container-based applications.

Regardless of the programming language you use, OpenShift Local hosts your application and brings a minimal, preconfigured Red Hat OpenShift Container Platform cluster to your local PC without the need for a server-based infrastructure.

On a hosted environment, OpenShift Local can create microservices, convert them into images, and run them in Kubernetes-hosted containers directly on your laptop or desktop running Linux, macOS, or Windows 10 or later.

For more information about OpenShift Local, see Red Hat OpenShift Local Overview .

2.4. NEXT STEPS

2.4.1. For developers

Develop and deploy containerized applications with OpenShift Container Platform. OpenShift Container Platform is a platform for developing and deploying containerized applications. OpenShift Container Platform documentation helps you:

- **Understand OpenShift Container Platform development**: Learn the different types of containerized applications, from simple containers to advanced Kubernetes deployments and Operators.
- **Work with projects**: Create projects from the OpenShift Container Platform web console or OpenShift CLI (**oc**) to organize and share the software you develop.
- **Work with applications**:

Use the **Developer** perspective in the OpenShift Container Platform web console to create and deploy applications.

Use the **Topology** view to see your applications, monitor status, connect and group components, and modify your code base.

- **Use the developer CLI tool (odo)**: The **odo** CLI tool lets developers create single or multi-component applications and automates deployment, build, and service route configurations. It abstracts complex Kubernetes and OpenShift Container Platform concepts, allowing you to focus on developing your applications.
- **Create CI/CD Pipelines**: Pipelines are serverless, cloud-native, continuous integration, and continuous deployment systems that run in isolated containers. They use standard Tekton custom resources to automate deployments and are designed for decentralized teams working on microservices-based architecture.
- **Deploy Helm charts**: Helm 3 is a package manager that helps developers define, install, and update application packages on Kubernetes. A Helm chart is a packaging format that describes an application that can be deployed using the Helm CLI.
- **Understand image builds**: Choose from different build strategies (Docker, S2I, custom, and pipeline) that can include different kinds of source materials (Git repositories, local binary inputs, and external artifacts). Then, follow examples of build types from basic builds to advanced builds.
- **Create container images**: A container image is the most basic building block in OpenShift Container Platform (and Kubernetes) applications. Defining image streams lets you gather

multiple versions of an image in one place as you continue its development. S2I containers let you insert your source code into a base container that is set up to run code of a particular type, such as Ruby, Node.js, or Python.

- **Create deployments**: Use **Deployment** and **DeploymentConfig** objects to exert fine-grained management over applications. Manage deployments using the **Workloads** page or OpenShift CLI (**oc**). Learn rolling, recreate, and custom deployment strategies.

- **Create templates**: Use existing templates or create your own templates that describe how an application is built or deployed. A template can combine images with descriptions, parameters, replicas, exposed ports and other content that defines how an application can be run or built.

- **Understand Operators**: Operators are the preferred method for creating on-cluster applications for OpenShift Container Platform 4.12. Learn about the Operator Framework and how to deploy applications using installed Operators into your projects.

- **Develop Operators**: Operators are the preferred method for creating on-cluster applications for OpenShift Container Platform 4.12. Learn the workflow for building, testing, and deploying Operators. Then, create your own Operators based on Ansible or Helm, or configure built-in Prometheus monitoring using the Operator SDK.

- **REST API reference**: Learn about OpenShift Container Platform application programming interface endpoints.

2.4.2. For administrators

- **Understand OpenShift Container Platform management**: Learn about components of the OpenShift Container Platform 4.12 control plane. See how OpenShift Container Platform control plane and worker nodes are managed and updated through the Machine API and Operators.

- **Manage users and groups**: Add users and groups with different levels of permissions to use or modify clusters.

- **Manage authentication**: Learn how user, group, and API authentication works in OpenShift Container Platform. OpenShift Container Platform supports multiple identity providers.

- **Manage networking**: The cluster network in OpenShift Container Platform is managed by the Cluster Network Operator (CNO). The CNO uses iptables rules in kube-proxy to direct traffic between nodes and pods running on those nodes. The Multus Container Network Interface adds the capability to attach multiple network interfaces to a pod. Using network policy features, you can isolate your pods or permit selected traffic.

- **Manage storage**: OpenShift Container Platform allows cluster administrators to configure persistent storage.

- **Manage Operators**: Lists of Red Hat, ISV, and community Operators can be reviewed by cluster administrators and installed on their clusters. After you install them, you can run, upgrade, back up, or otherwise manage the Operator on your cluster.

- **Use custom resource definitions (CRDs) to modify the cluster**: Cluster features implemented with Operators can be modified with CRDs. Learn to create a CRD and manage resources from CRDs.

- **Set resource quotas**: Choose from CPU, memory, and other system resources to set quotas.

- **Prune and reclaim resources**: Reclaim space by pruning unneeded Operators, groups, deployments, builds, images, registries, and cron jobs.
- **Scale and tune clusters**: Set cluster limits, tune nodes, scale cluster monitoring, and optimize networking, storage, and routes for your environment.
- **Understanding the OpenShift Update Service**: Learn about installing and managing a local OpenShift Update Service for recommending OpenShift Container Platform updates in disconnected environments.
- **Monitor clusters**: Learn to configure the monitoring stack. After configuring monitoring, use the web console to access monitoring dashboards. In addition to infrastructure metrics, you can also scrape and view metrics for your own services.
- **Remote health monitoring**: OpenShift Container Platform collects anonymized aggregated information about your cluster. Using Telemetry and the Insights Operator, this data is received by Red Hat and used to improve OpenShift Container Platform. You can view the data collected by remote health monitoring.

CHAPTER 3. CREATING AND BUILDING AN APPLICATION USING THE WEB CONSOLE

3.1. BEFORE YOU BEGIN

- Review Accessing the web console.
- You must be able to access a running instance of OpenShift Container Platform. If you do not have access, contact your cluster administrator.

3.2. LOGGING IN TO THE WEB CONSOLE

You can log in to the OpenShift Container Platform web console to access and manage your cluster.

Prerequisites

- You must have access to an OpenShift Container Platform cluster.

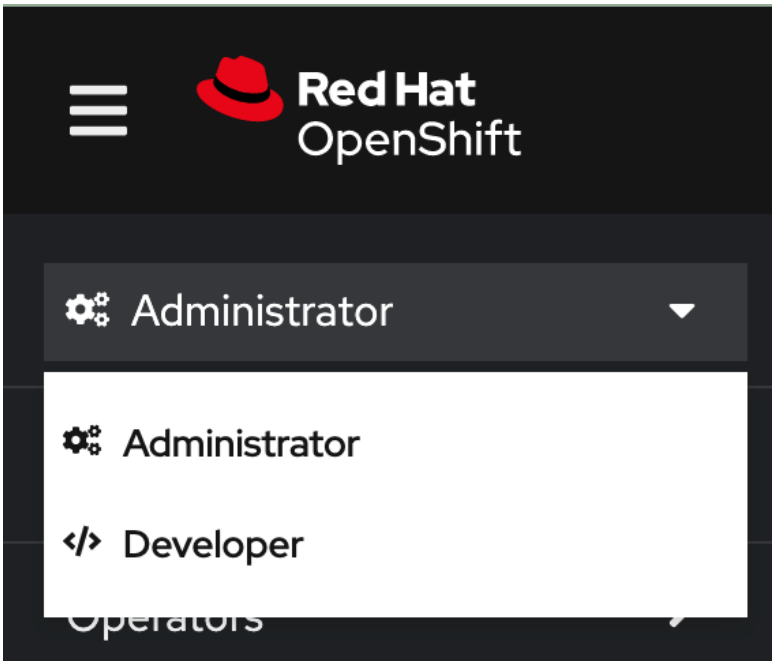
Procedure

- Log in to the OpenShift Container Platform web console using your login credentials.

You are redirected to the **Projects** page. For non-administrative users, the default view is the **Developer** perspective. For cluster administrators, the default view is the **Administrator** perspective. If you do not have **cluster-admin** privileges, you will not see the **Administrator** perspective in your web console.

The web console provides two perspectives: the **Administrator** perspective and **Developer** perspective. The **Developer** perspective provides workflows specific to the developer use cases.

Figure 3.1. Perspective switcher



Use the perspective switcher to switch to the **Developer** perspective. The **Topology** view with options to create an application is displayed.

3.3. CREATING A NEW PROJECT

A project enables a community of users to organize and manage their content in isolation. Projects are OpenShift Container Platform extensions to Kubernetes namespaces. Projects have additional features that enable user self-provisioning.

Users must receive access to projects from administrators. Cluster administrators can allow developers to create their own projects. In most cases, users automatically have access to their own projects.

Each project has its own set of objects, policies, constraints, and service accounts.

Prerequisites

- You are logged in to the OpenShift Container Platform web console.
- You are in the **Developer** perspective.
- You have the appropriate roles and permissions in a project to create applications and other workloads in OpenShift Container Platform.

Procedure

1. In the **+Add** view, select **Project → Create Project**.
2. In the **Name** field, enter **user-getting-started**.
3. Optional: In the **Display name** field, enter **Getting Started with OpenShift**.

NOTE

Display name and **Description** fields are optional.

4. Click **Create**.

You have created your first project on OpenShift Container Platform.

Additional resources

- Default cluster roles
- Viewing a project using the web console
- Providing access permissions to your project using the Developer perspective
- Deleting a project using the web console

3.4. GRANTING VIEW PERMISSIONS

OpenShift Container Platform automatically creates a few special service accounts in every project. The default service account takes responsibility for running the pods. OpenShift Container Platform uses and injects this service account into every pod that launches.

The following procedure creates a **RoleBinding** object for the default **ServiceAccount** object. The service account communicates with the OpenShift Container Platform API to learn about pods, services, and resources within the project.

Prerequisites

- You are logged in to the OpenShift Container Platform web console.
- You have a deployed image.
- You are in the **Administrator** perspective.

Procedure

1. Navigate to **User Management** and then click **RoleBindings**.
2. Click **Create binding**.
3. Select **Namespace role binding (RoleBinding)**.
4. In the **Name** field, enter **sa-user-account**.
5. In the **Namespace** field, search for and select **user-getting-started**.
6. In the **Role name** field, search for **view** and select **view**.
7. In the **Subject** field, select **ServiceAccount**.
8. In the **Subject namespace** field, search for and select **user-getting-started**.
9. In the **Subject name** field, enter **default**.
10. Click **Create**.

Additional resources

- Understanding authentication
- RBAC overview

3.5. DEPLOYING YOUR FIRST IMAGE

The simplest way to deploy an application in OpenShift Container Platform is to run an existing container image. The following procedure deploys a front end component of an application called **national-parks-app**. The web application displays an interactive map. The map displays the location of major national parks across the world.

Prerequisites

- You are logged in to the OpenShift Container Platform web console.
- You are in the **Developer** perspective.
- You have the appropriate roles and permissions in a project to create applications and other workloads in OpenShift Container Platform.

Procedure

1. From the **+Add** view in the **Developer** perspective, click **Container images** to open a dialog.

2. In the **Image Name** field, enter the following: **quay.io/openshiftroadshow/parksmap:latest**
3. Ensure that you have the current values for the following:
   a. Application: **national-parks-app**
   b. Name: **parksmap**
4. Select **Deployment** as the **Resource**.
5. Select **Create route to the application**
6. In the **Advanced Options** section, click **Labels** and add labels to better identify this deployment later. Labels help identify and filter components in the web console and in the command line. Add the following labels:
   - **app=national-parks-app**
   - **component=parksmap**
   - **role=frontend**
7. Click **Create**.

You are redirected to the **Topology** page where you can see the **parksmap** deployment in the **national-parks-app** application.

Additional resources

- Creating applications using the Developer perspective
- Viewing a project using the web console
- Viewing the topology of your application
- Deleting a project using the web console

3.5.1. Examining the pod

OpenShift Container Platform leverages the Kubernetes concept of a pod, which is one or more containers deployed together on one host, and the smallest compute unit that can be defined, deployed, and managed. Pods are the rough equivalent of a machine instance, physical or virtual, to a container.

The **Overview** panel enables you to access many features of the **parksmap** deployment. The **Details** and **Resources** tabs enable you to scale application pods, check build status, services, and routes.

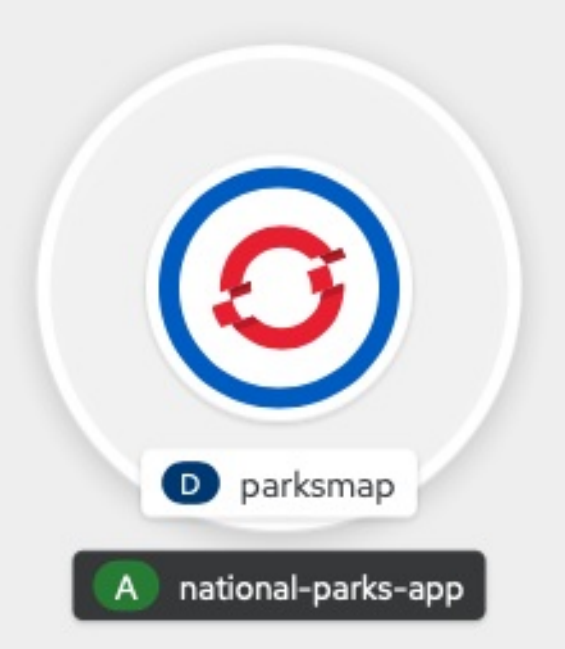
Prerequisites

- You are logged in to the OpenShift Container Platform web console.
- You are in the **Developer** perspective.
- You have a deployed image.

Procedure

- Click **D parksmap** in the **Topology** view to open the **Overview** panel.

Figure 3.2. Parksmap deployment



The **Overview** panel includes tabs for **Details**, **Resources**, and **Observe**. The **Details** tab might be displayed by default.

Table 3.1. Overview panel tab definitions

Tab	Defintion
Details	Enables you to scale your application and view pod configuration such as labels, annotations, and the status of the application.
Resources	Displays the resources that are associated with the deployment.
	Pods are the basic units of OpenShift Container Platform applications. You can see how many pods are being used, what their status is, and you can view the logs.
	Services that are created for your pod and assigned ports are listed under the **Services** heading.
	Routes enable external access to the pods and a URL is used to access them.
Observe	View various **Events** and **Metrics** information as it relates to your pod.

Additional resources

- Interacting with applications and components

- Scaling application pods and checking builds and routes
- Labels and annotations used for the Topology view

3.5.2. Scaling the application

In Kubernetes, a **Deployment** object defines how an application deploys. In most cases, users use **Pod**, **Service**, **ReplicaSets**, and **Deployment** resources together. In most cases, OpenShift Container Platform creates the resources for you.

When you deploy the **national-parks-app** image, a deployment resource is created. In this example, only one **Pod** is deployed.

The following procedure scales the **national-parks-image** to use two instances.

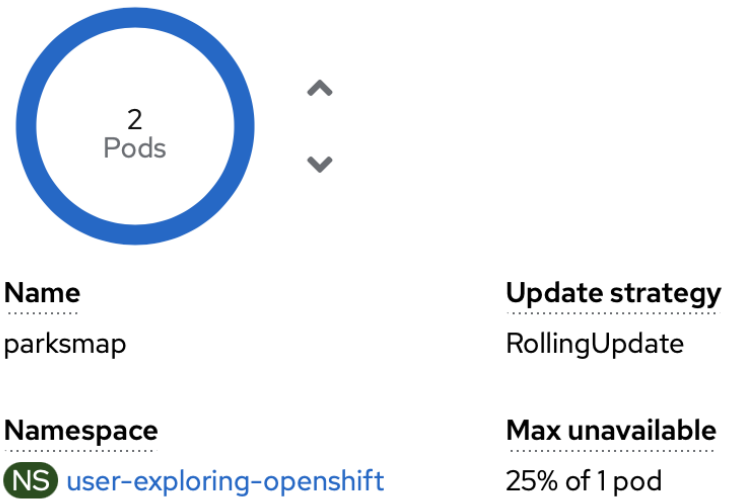
Prerequisites

- You are logged in to the OpenShift Container Platform web console.
- You are in the **Developer** perspective.
- You have a deployed image.

Procedure

1. In the **Topology** view, click the **national-parks-app** application.
2. Click the **Details** tab.
3. Use the up arrow to scale the pod to two instances.

   Figure 3.3. Scaling application



NOTE

Application scaling can happen quickly because OpenShift Container Platform is launching a new instance of an existing image.

4. Use the down arrow to scale the pod down to one instance.

Additional resources

- Recommended practices for scaling the cluster
- Understanding horizontal pod autoscalers
- About the Vertical Pod Autoscaler Operator

3.6. DEPLOYING A PYTHON APPLICATION

The following procedure deploys a back-end service for the **parksmap** application. The Python application performs 2D geo-spatial queries against a MongoDB database to locate and return map coordinates of all national parks in the world.

The deployed back-end service that is **nationalparks**.

Prerequisites

- You are logged in to the OpenShift Container Platform web console.
- You are in the **Developer** perspective.
- You have a deployed image.

Procedure

1. From the **+Add** view in the **Developer** perspective, click **Import from Git** to open a dialog.
2. Enter the following URL in the Git Repo URL field: **https://github.com/openshift-roadshow/nationalparks-py.git**
   A builder image is automatically detected.

NOTE

If the detected builder image is Dockerfile, select **Edit Import Strategy**. Select **Builder Image** and then click **Python**.

3. Scroll to the **General** section.
4. Ensure that you have the current values for the following:
   a. Application: **national-parks-app**
   b. Name: **nationalparks**
5. Select **Deployment** as the **Resource**.
6. Select **Create route to the application**
7. In the **Advanced Options** section, click **Labels** and add labels to better identify this deployment later. Labels help identify and filter components in the web console and in the command line. Add the following labels:

a. **app=national-parks-app**

b. **component=nationalparks**

c. **role=backend**

d. **type=parksmap-backend**

8. Click **Create**.

9. From the **Topology** view, select the **nationalparks** application.

NOTE

Click the **Resources** tab. In the **Builds** section, you can see your build running.

Additional resources

- Adding services to your application
- Importing a codebase from Git to create an application
- Viewing the topology of your application
- Providing access permissions to your project using the Developer perspective
- Deleting a project using the web console

3.7. CONNECTING TO A DATABASE

Deploy and connect a MongoDB database where the **national-parks-app** application stores location information. Once you mark the **national-parks-app** application as a backend for the map visualization tool, **parksmap** deployment uses the OpenShift Container Platform discover mechanism to display the map automatically.

Prerequisites

- You are logged in to the OpenShift Container Platform web console.
- You are in the **Developer** perspective.
- You have a deployed image.

Procedure

1. From the **+Add** view in the **Developer** perspective, click **Container images** to open a dialog.
2. In the **Image Name** field, enter **quay.io/centos7/mongodb-36-centos7**.
3. In the **Runtime icon** field, search for **mongodb**.
4. Scroll down to the **General** section.
5. Ensure that you have the current values for the following:

a. Application: **national-parks-app**

b. Name: **mongodb-nationalparks**

6. Select **Deployment** as the **Resource**.
7. Unselect the checkbox next to **Create route to the application**
8. In the **Advanced Options** section, click **Deployment** to add environment variables to add the following environment variables:

   Table 3.2. Environment variable names and values

Name	Value
MONGODB_USER	**mongodb**
MONGODB_PASSWORD	**mongodb**
MONGODB_DATABASE	**mongodb**
MONGODB_ADMIN_PASSWORD	**mongodb**

9. Click **Create**.

Additional resources

- Adding services to your application
- Viewing a project using the web console
- Viewing the topology of your application
- Providing access permissions to your project using the Developer perspective
- Deleting a project using the web console

3.7.1. Creating a secret

The **Secret** object provides a mechanism to hold sensitive information such as passwords, OpenShift Container Platform client configuration files, private source repository credentials, and so on. Secrets decouple sensitive content from the pods. You can mount secrets into containers using a volume plugin or the system can use secrets to perform actions on behalf of a pod. The following procedure adds the secret **nationalparks-mongodb-parameters** and mounts it to the **nationalparks** workload.

Prerequisites

- You are logged in to the OpenShift Container Platform web console.
- You are in the **Developer** perspective.
- You have a deployed image.

Procedure

1. From the **Developer** perspective, navigate to **Secrets** on the left hand navigation and click **Secrets**.
2. Click **Create → Key/value secret**
   a. In the **Secret name** field, enter **nationalparks-mongodb-parameters**.
   b. Enter the following values for **Key** and **Value**:

   Table 3.3. Secret keys and values

Key	Value
MONGODB_USER	**mongodb**
DATABASE_SERVICE_NAME	**mongodb-nationalparks**
MONGODB_PASSWORD	**mongodb**
MONGODB_DATABASE	**mongodb**
MONGODB_ADMIN_PASSWORD	**mongodb**

   c. Click **Create**.
3. Click **Add Secret to workload**
   a. From the drop down menu, select **nationalparks** as the workload to add.
   b. Click **Save**.

This change in configuration triggers a new rollout of the **nationalparks** deployment with the environment variables properly injected.

Additional resources

- Understanding secrets

3.7.2. Loading data and displaying the national parks map

You deployed the **parksmap** and **nationalparks** applications and then deployed the **mongodb-nationalparks** database. However, no data has been loaded *into* the database. Before loading the data, add the proper labels to the **mongodb-nationalparks** and **nationalparks** deployment.

Prerequisites

- You are logged in to the OpenShift Container Platform web console.
- You are in the **Developer** perspective.
- You have a deployed image.

Procedure

1. From the **Topology** view, navigate to **nationalparks** deployment and click **Resources** and retrieve your route information.
2. Copy and paste the URL into your web browser and add the following at the end of the URL:

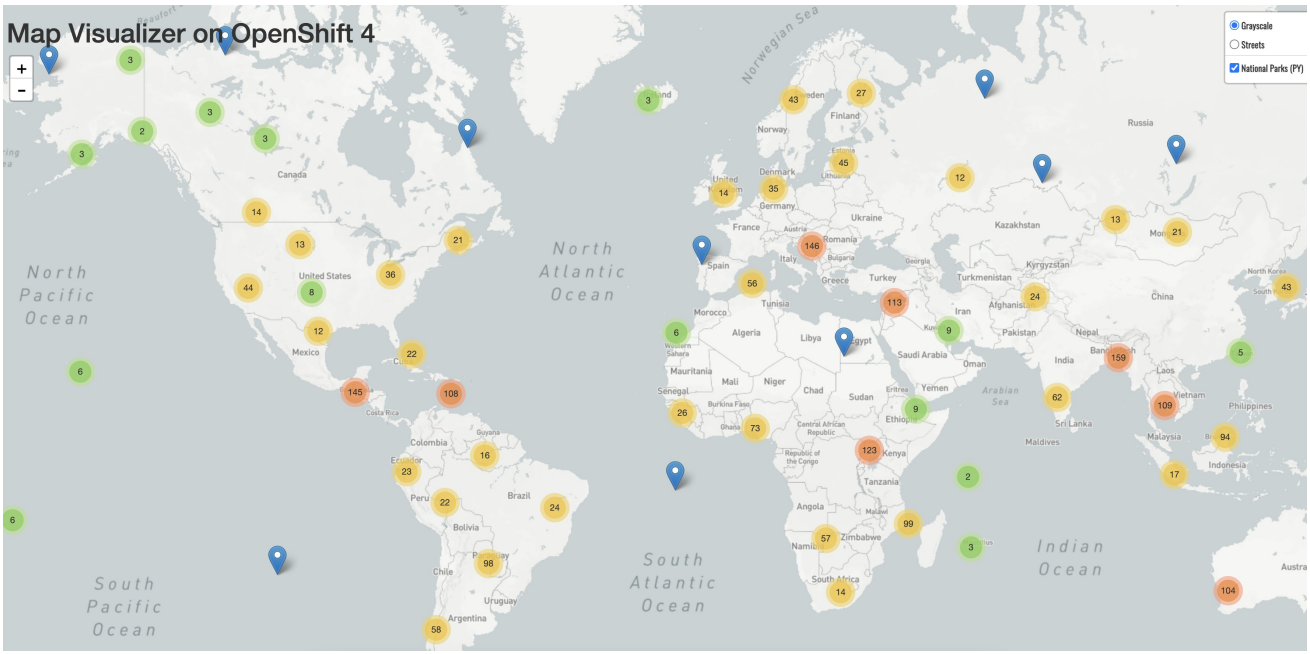
```
/ws/data/load
```

Example output

```
Items inserted in database: 2893
```

3. From the **Topology** view, navigate to **parksmap** deployment and click **Resources** and retrieve your route information.
4. Copy and paste the URL into your web browser to view your national parks across the world map.

Figure 3.4. National parks across the world



Additional resources

- Providing access permissions to your project using the Developer perspective
- Labels and annotations used for the Topology view

CHAPTER 4. CREATING AND BUILDING AN APPLICATION USING THE CLI

4.1. BEFORE YOU BEGIN

- Review About the OpenShift CLI.
- You must be able to access a running instance of OpenShift Container Platform. If you do not have access, contact your cluster administrator.
- You must have the OpenShift CLI (**oc**) downloaded and installed.

4.2. LOGGING IN TO THE CLI

You can log in to the OpenShift CLI (**oc**) to access and manage your cluster.

Prerequisites

- You must have access to an OpenShift Container Platform cluster.
- You must have installed the OpenShift CLI (**oc**).

Procedure

- Log into OpenShift Container Platform from the CLI using your username and password or with an OAuth token:
  - With username and password:

```
$ oc login -u=<username> -p=<password> --server=<your-openshift-server> --insecure-skip-tls-verify
```

  - With an OAuth token:

```
$ oc login <https://api.your-openshift-server.com> --token=<tokenID>
```

You can now create a project or issue other commands for managing your cluster.

Additional resources

- oc login
- oc logout

4.3. CREATING A NEW PROJECT

A project enables a community of users to organize and manage their content in isolation. Projects are OpenShift Container Platform extensions to Kubernetes namespaces. Projects have additional features that enable user self-provisioning.

Users must receive access to projects from administrators. Cluster administrators can allow developers to create their own projects. In most cases, users automatically have access to their own projects.

Each project has its own set of objects, policies, constraints, and service accounts.

Prerequisites

- You must have access to an OpenShift Container Platform cluster.
- You must have installed the OpenShift CLI (**oc**).

Procedure

- To create a new project, enter the following command:

```
$ oc new-project user-getting-started --display-name="Getting Started with OpenShift"
```

Example output

```
Now using project "user-getting-started" on server "https://openshift.example.com:6443".
```

Additional resources

- oc new-project

4.4. GRANTING VIEW PERMISSIONS

OpenShift Container Platform automatically creates a few special service accounts in every project. The default service account takes responsibility for running the pods. OpenShift Container Platform uses and injects this service account into every pod that launches.

The following procedure creates a **RoleBinding** object for the default **ServiceAccount** object. The service account communicates with the OpenShift Container Platform API to learn about pods, services, and resources within the project.

Prerequisites

- You must have access to an OpenShift Container Platform cluster.
- You must have installed the OpenShift CLI (**oc**).
- You have a deployed image.
- You must have **cluster-admin** or **project-admin** privileges.

Procedure

- To add the view role to the default service account in the **user-getting-started project**, enter the following command:

```
$ oc adm policy add-role-to-user view -z default -n user-getting-started
```

Additional resources

- Understanding authentication

- RBAC overview
- oc policy add-role-to-user

4.5. DEPLOYING YOUR FIRST IMAGE

The simplest way to deploy an application in OpenShift Container Platform is to run an existing container image. The following procedure deploys a front-end component of an application called **national-parks-app**. The web application displays an interactive map. The map displays the location of major national parks across the world.

Prerequisites

- You must have access to an OpenShift Container Platform cluster.
- Install the OpenShift CLI (**oc**).

Procedure

- To deploy an application, enter the following command:

```
$ oc new-app quay.io/openshiftroadshow/parksmap:latest --name=parksmap -l
'app=national-parks-app,component=parksmap,role=frontend,app.kubernetes.io/part-
of=national-parks-app'
```

Example output

```
--> Found container image 0c2f55f (12 months old) from quay.io for
"quay.io/openshiftroadshow/parksmap:latest"

    * An image stream tag will be created as "parksmap:latest" that will track this image

--> Creating resources with label app=national-parks-app,app.kubernetes.io/part-of=national-
parks-app,component=parksmap,role=frontend ...
    imagestream.image.openshift.io "parksmap" created
    deployment.apps "parksmap" created
    service "parksmap" created
--> Success
```

Additional resources

- oc new-app

4.5.1. Creating a route

External clients can access applications running on OpenShift Container Platform through the routing layer and the data object behind that is a *route*. The default OpenShift Container Platform router (HAProxy) uses the HTTP header of the incoming request to determine where to proxy the connection.

Optionally, you can define security, such as TLS, for the route.

Prerequisites

- You must have access to an OpenShift Container Platform cluster.
- You must have installed the OpenShift CLI (**oc**).
- You have a deployed image.
- You must have **cluster-admin** or **project-admin** privileges.

Procedure

1. To retrieve the created application service, enter the following command:

```
$ oc get service
```

Example output

```
NAME      TYPE       CLUSTER-IP         EXTERNAL-IP    PORT(S)   AGE
parksmap  ClusterIP  <your-cluster-IP>  <123.456.789>  8080/TCP  8m29s
```

2. To create a route, enter the following command:

```
$ oc create route edge parksmap --service=parksmap
```

Example output

```
route.route.openshift.io/parksmap created
```

3. To retrieve the created application route, enter the following command:

```
$ oc get route
```

Example output

```
NAME       HOST/PORT                                          PATH   SERVICES   PORT
TERMINATION   WILDCARD
parksmap   parksmap-user-getting-started.apps.cluster.example.com          parksmap
8080-tcp   edge          None
```

Additional resources

- oc create route edge
- oc get

4.5.2. Examining the pod

OpenShift Container Platform leverages the Kubernetes concept of a pod, which is one or more containers deployed together on one host, and the smallest compute unit that can be defined, deployed, and managed. Pods are the rough equivalent of a machine instance, physical or virtual, to a container.

You can view the pods in your cluster and to determine the health of those pods and the cluster as a whole.

Prerequisites

- You must have access to an OpenShift Container Platform cluster.
- You must have installed the OpenShift CLI (**oc**).
- You have a deployed image.

Procedure

1. To list all pods with node names, enter the following command:

```
$ oc get pods
```

Example output

```
NAME                       READY   STATUS    RESTARTS   AGE
parksmap-5f9579955-6sng8   1/1     Running   0          77s
```

2. To list all pod details, enter the following command:

```
$ oc describe pods
```

Example output

```
Name:         parksmap-848bd4954b-5pvcc
Namespace:    user-getting-started
Priority:     0
Node:         ci-ln-fr1rt92-72292-4fzf9-worker-a-g9g7c/10.0.128.4
Start Time:   Sun, 13 Feb 2022 14:14:14 -0500
Labels:       app=national-parks-app
              app.kubernetes.io/part-of=national-parks-app
              component=parksmap
              deployment=parksmap
              pod-template-hash=848bd4954b
              role=frontend
Annotations:  k8s.v1.cni.cncf.io/network-status:
                [{
                    "name": "openshift-sdn",
                    "interface": "eth0",
                    "ips": [
                        "10.131.0.14"
                    ],
                    "default": true,
                    "dns": {}
                }]
              k8s.v1.cni.cncf.io/networks-status:
                [{
                    "name": "openshift-sdn",
                    "interface": "eth0",
                    "ips": [
```

```
                    "10.131.0.14"
                ],
                "default": true,
                "dns": {}
            }]
          openshift.io/generated-by: OpenShiftNewApp
          openshift.io/scc: restricted
Status:       Running
IP:           10.131.0.14
IPs:
  IP:           10.131.0.14
Controlled By:  ReplicaSet/parksmap-848bd4954b
Containers:
  parksmap:
    Container ID:   cri-
o://4b2625d4f61861e33cc95ad6d455915ea8ff6b75e17650538cc33c1e3e26aeb8
    Image:
quay.io/openshiftroadshow/parksmap@sha256:89d1e324846cb431df9039e1a7fd0ed2ba0c51a
afbae73f2abd70a83d5fa173b
    Image ID:
quay.io/openshiftroadshow/parksmap@sha256:89d1e324846cb431df9039e1a7fd0ed2ba0c51a
afbae73f2abd70a83d5fa173b
    Port:           8080/TCP
    Host Port:      0/TCP
    State:          Running
      Started:      Sun, 13 Feb 2022 14:14:25 -0500
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-6f844 (ro)
Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True
Volumes:
  kube-api-access-6f844:
    Type:                    Projected (a volume that contains injected data from multiple sources)
    TokenExpirationSeconds:  3607
    ConfigMapName:           kube-root-ca.crt
    ConfigMapOptional:       <nil>
    DownwardAPI:             true
    ConfigMapName:           openshift-service-ca.crt
    ConfigMapOptional:       <nil>
QoS Class:                   BestEffort
Node-Selectors:              <none>
Tolerations:                 node.kubernetes.io/not-ready:NoExecute op=Exists for 300s
                             node.kubernetes.io/unreachable:NoExecute op=Exists for 300s
Events:
  Type    Reason          Age   From               Message
  ----    ------          ----  ----               -------
  Normal  Scheduled       46s   default-scheduler  Successfully assigned user-getting-
started/parksmap-848bd4954b-5pvcc to ci-ln-fr1rt92-72292-4fzf9-worker-a-g9g7c
  Normal  AddedInterface  44s   multus             Add eth0 [10.131.0.14/23] from openshift-sdn
```

```
  Normal  Pulling         44s   kubelet            Pulling image
"quay.io/openshiftroadshow/parksmap@sha256:89d1e324846cb431df9039e1a7fd0ed2ba0c51
aafbae73f2abd70a83d5fa173b"
  Normal  Pulled          35s   kubelet            Successfully pulled image
"quay.io/openshiftroadshow/parksmap@sha256:89d1e324846cb431df9039e1a7fd0ed2ba0c51
aafbae73f2abd70a83d5fa173b" in 9.49243308s
  Normal  Created         35s   kubelet            Created container parksmap
  Normal  Started         35s   kubelet            Started container parksmap
```

Additional resources

- oc describe
- oc get
- oc label
- Viewing pods
- Viewing pod logs

4.5.3. Scaling the application

In Kubernetes, a **Deployment** object defines how an application deploys. In most cases, users use **Pod**, **Service**, **ReplicaSets**, and **Deployment** resources together. In most cases, OpenShift Container Platform creates the resources for you.

When you deploy the **national-parks-app** image, a deployment resource is created. In this example, only one **Pod** is deployed.

The following procedure scales the **national-parks-image** to use two instances.

Prerequisites

- You must have access to an OpenShift Container Platform cluster.
- You must have installed the OpenShift CLI (**oc**).
- You have a deployed image.

Procedure

- To scale your application from one pod instance to two pod instances, enter the following command:

```
$ oc scale --current-replicas=1 --replicas=2 deployment/parksmap
```

Example output

```
deployment.apps/parksmap scaled
```

Verification

1. To ensure that your application scaled properly, enter the following command:

```
$ oc get pods
```

Example output

```
NAME                  READY   STATUS    RESTARTS   AGE
parksmap-5f9579955-6sng8   1/1     Running   0          7m39s
parksmap-5f9579955-8tgft   1/1     Running   0          24s
```

2. To scale your application back down to one pod instance, enter the following command:

```
$ oc scale --current-replicas=2 --replicas=1 deployment/parksmap
```

Additional resources

- oc scale

4.6. DEPLOYING A PYTHON APPLICATION

The following procedure deploys a back-end service for the **parksmap** application. The Python application performs 2D geo-spatial queries against a MongoDB database to locate and return map coordinates of all national parks in the world.

The deployed back-end service is **nationalparks**.

Prerequisites

- You must have access to an OpenShift Container Platform cluster.
- You must have installed the OpenShift CLI (**oc**).
- You have a deployed image.

Procedure

1. To create a new Python application, enter the following command:

```
$ oc new-app python~https://github.com/openshift-roadshow/nationalparks-py.git --name
nationalparks -l 'app=national-parks-
app,component=nationalparks,role=backend,app.kubernetes.io/part-of=national-parks-
app,app.kubernetes.io/name=python' --allow-missing-images=true
```

Example output

```
--> Found image 0406f6c (13 days old) in image stream "openshift/python" under tag "3.9-
ubi8" for "python"

    Python 3.9
    ----------
    Python 3.9 available as container is a base platform for building and running various
Python 3.9 applications and frameworks. Python is an easy to learn, powerful programming
language. It has efficient high-level data structures and a simple but effective approach to
object-oriented programming. Python's elegant syntax and dynamic typing, together with its
interpreted nature, make it an ideal language for scripting and rapid application development
```

```
    in many areas on most platforms.

      Tags: builder, python, python39, python-39, rh-python39

      * A source build using source code from https://github.com/openshift-
roadshow/nationalparks-py.git will be created
        * The resulting image will be pushed to image stream tag "nationalparks:latest"
        * Use 'oc start-build' to trigger a new build

--> Creating resources with label app=national-parks-
app,app.kubernetes.io/name=python,app.kubernetes.io/part-of=national-parks-
app,component=nationalparks,role=backend ...
    imagestream.image.openshift.io "nationalparks" created
    buildconfig.build.openshift.io "nationalparks" created
    deployment.apps "nationalparks" created
    service "nationalparks" created
--> Success
```

2. To create a route to expose your application, **nationalparks**, enter the following command:

```
$ oc create route edge nationalparks --service=nationalparks
```

Example output

```
route.route.openshift.io/parksmap created
```

3. To retrieve the created application route, enter the following command:

```
$ oc get route
```

Example output

```
NAME            HOST/PORT                                                PATH   SERVICES
PORT       TERMINATION   WILDCARD
nationalparks    nationalparks-user-getting-started.apps.cluster.example.com
nationalparks   8080-tcp    edge          None
parksmap         parksmap-user-getting-started.apps.cluster.example.com
parksmap        8080-tcp    edge          None
```

Additional resources

- oc new-app

4.7. CONNECTING TO A DATABASE

Deploy and connect a MongoDB database where the **national-parks-app** application stores location information. Once you mark the **national-parks-app** application as a backend for the map visualization tool, **parksmap** deployment uses the OpenShift Container Platform discover mechanism to display the map automatically.

Prerequisites

- You must have access to an OpenShift Container Platform cluster.

- You must have installed the OpenShift CLI (**oc**).
- You have a deployed image.

Procedure

- To connect to a database, enter the following command:

```
$ oc new-app quay.io/centos7/mongodb-36-centos7 --name mongodb-nationalparks -e
MONGODB_USER=mongodb -e MONGODB_PASSWORD=mongodb -e
MONGODB_DATABASE=mongodb -e MONGODB_ADMIN_PASSWORD=mongodb -l
'app.kubernetes.io/part-of=national-parks-app,app.kubernetes.io/name=mongodb'
```

Example output

```
--> Found container image dc18f52 (8 months old) from quay.io for
"quay.io/centos7/mongodb-36-centos7"

    MongoDB 3.6
    -----------
    MongoDB (from humongous) is a free and open-source cross-platform document-oriented
database program. Classified as a NoSQL database program, MongoDB uses JSON-like
documents with schemas. This container image contains programs to run mongod server.

    Tags: database, mongodb, rh-mongodb36

    * An image stream tag will be created as "mongodb-nationalparks:latest" that will track this
image

--> Creating resources with label app.kubernetes.io/name=mongodb,app.kubernetes.io/part-
of=national-parks-app ...
    imagestream.image.openshift.io "mongodb-nationalparks" created
    deployment.apps "mongodb-nationalparks" created
    service "mongodb-nationalparks" created
--> Success
```

Additional resources

- oc new-project

4.7.1. Creating a secret

The **Secret** object provides a mechanism to hold sensitive information such as passwords, OpenShift Container Platform client configuration files, private source repository credentials, and so on. Secrets decouple sensitive content from the pods. You can mount secrets into containers using a volume plugin or the system can use secrets to perform actions on behalf of a pod. The following procedure adds the secret **nationalparks-mongodb-parameters** and mounts it to the **nationalparks** workload.

Prerequisites

- You must have access to an OpenShift Container Platform cluster.
- You must have installed the OpenShift CLI (**oc**).

- You have a deployed image.

Procedure

1. To create a secret, enter the following command:

```
$ oc create secret generic nationalparks-mongodb-parameters --from-
literal=DATABASE_SERVICE_NAME=mongodb-nationalparks --from-
literal=MONGODB_USER=mongodb --from-literal=MONGODB_PASSWORD=mongodb --
from-literal=MONGODB_DATABASE=mongodb --from-
literal=MONGODB_ADMIN_PASSWORD=mongodb
```

Example output

```
secret/nationalparks-mongodb-parameters created
```

2. To update the environment variable to attach the mongodb secret to the **nationalpartks** workload, enter the following command:

```
$ oc set env --from=secret/nationalparks-mongodb-parameters deploy/nationalparks
```

Example output

```
deployment.apps/nationalparks updated
```

3. To show the status of the **nationalparks** deployment, enter the following command:

```
$ oc rollout status deployment nationalparks
```

Example output

```
deployment "nationalparks" successfully rolled out
```

4. To show the status of the **mongodb-nationalparks** deployment, enter the following command:

```
$ oc rollout status deployment mongodb-nationalparks
```

Example output

```
deployment "nationalparks" successfully rolled out
deployment "mongodb-nationalparks" successfully rolled out
```

Additional resources

- oc create secret generic
- oc set env
- oc rollout status

4.7.2. Loading data and displaying the national parks map

You deployed the **parksmap** and **nationalparks** applications and then deployed the **mongodb-nationalparks** database. However, no data has been loaded *into* the database.

Prerequisites

- You must have access to an OpenShift Container Platform cluster.
- You must have installed the OpenShift CLI (**oc**).
- You have a deployed image.

Procedure

1. To load national parks data, enter the following command:

```
$ oc exec $(oc get pods -l component=nationalparks | tail -n 1 | awk '{print $1;}') -- curl -s http://localhost:8080/ws/data/load
```

Example output

```
"Items inserted in database: 2893"
```

2. To verify that your data is loaded properly, enter the following command:

```
$ oc exec $(oc get pods -l component=nationalparks | tail -n 1 | awk '{print $1;}') -- curl -s http://localhost:8080/ws/data/all
```

Example output (trimmed)

```
, {"id": "Great Zimbabwe", "latitude": "-20.2674635", "longitude": "30.9337986", "name": "Great Zimbabwe"}]
```

3. To add labels to the route, enter the following command:

```
$ oc label route nationalparks type=parksmap-backend
```

Example output

```
route.route.openshift.io/nationalparks labeled
```

4. To retrieve your routes to view your map, enter the following command:

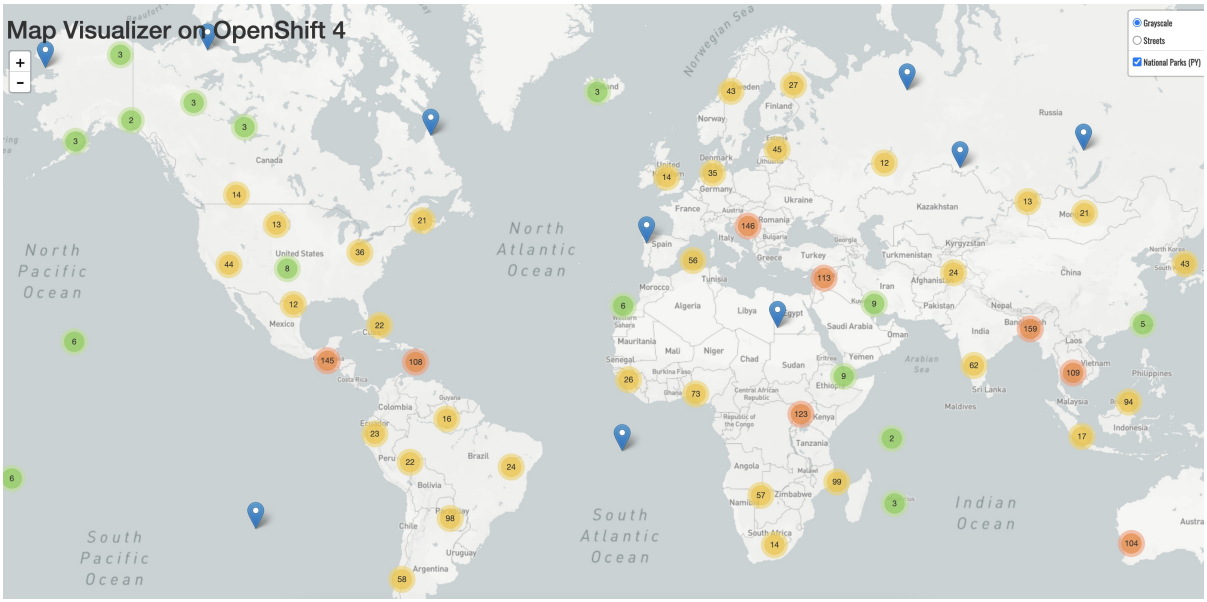
```
$ oc get routes
```

Example output

```
NAME          HOST/PORT                                        PATH   SERVICES      PORT       TERMINATION   WILDCARD
nationalparks   nationalparks-user-getting-started.apps.cluster.example.com
nationalparks   8080-tcp   edge          None
parksmap        parksmap-user-getting-started.apps.cluster.example.com                parksmap
8080-tcp   edge          None
```

5. Copy and paste the **HOST/PORT** path you retrieved above into your web browser. Your browser should display a map of the national parks across the world.

Figure 4.1. National parks across the world



Additional resources

- oc exec
- oc label
- oc get