OpenShift Container Platform 4.19

Tutorials

Getting started in OpenShift Container Platform

Last Updated: 2026-03-27

OpenShift Container Platform 4.19 Tutorials

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. TUTORIALS OVERVIEW

You can follow an end-to-end example of deploying an application on OpenShift Container Platform either by using the OpenShift CLI (**oc**) or the web console.

Choose one of the following tutorials:

- Tutorial: Deploying an application by using the CLI
- Tutorial: Deploying an application by using the web console

1.1. ADDITIONAL LEARNING RESOURCES

To discover additional tutorials and hands-on learning resources for OpenShift Container Platform, see Additional hands-on learning.

CHAPTER 2. TUTORIAL: DEPLOYING AN APPLICATION BY USING THE WEB CONSOLE

To learn how to stand up an application on OpenShift Container Platform by using the web console, follow the provided tutorial. In this tutorial, you will deploy the services that are required for an application that displays a map of national parks across the world.

To complete this tutorial, you will perform the following steps:

1. Create a project for the application.
   This step allows your application to be isolated from other cluster user's workloads.

2. Grant view permissions.
   This step grants **view** permissions to interact with the OpenShift API to help discover services and other resources running within the project.

3. Deploy the front-end application.
   This step deploys the **parksmap** front-end application, exposes it externally, and scales it up to two instances.

4. Deploy the back-end application.
   This step deploys the **nationalparks** back-end application and exposes it externally.

5. Deploy the database application.
   This step deploys the **mongodb-nationalparks** MongoDB database, loads data into the database, and sets up the necessary credentials to access the database.

After you complete these steps, you can view the national parks application in a web browser.

2.1. PREREQUISITES

Before you start this tutorial, ensure that you have the following required prerequisites:

- You have access to a test OpenShift Container Platform cluster.
  If your organization does not have a cluster to test on, you can request access to the Developer Sandbox to get a trial of OpenShift Container Platform.

- You have the appropriate permissions, such as the **cluster-admin** cluster role, to create a project and applications within it.
  If you do not have the required permissions, contact your cluster administrator. You need the **self-provisioner** role to create a project and the **admin** role on the project to modify resources in that project.

  If you are using Developer Sandbox, a project is created for you with the required permissions.

- You have logged in to the OpenShift Container Platform web console.

2.2. CREATING A PROJECT

Create a new project to contain all required resources and application components for the tutorial.

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

service accounts.

Cluster administrators can allow developers to create their own projects. In most cases, you automatically have access to your own projects. Administrators can grant access to other projects as needed.

This procedure creates a new project called **user-getting-started**. You will use this project throughout the rest of this tutorial.

IMPORTANT

If you are using Developer Sandbox to complete this tutorial, skip this procedure. A project has already been created for you.

Prerequisites

- You have logged in to the OpenShift Container Platform web console.

Procedure

1. Navigate to **Home → Projects**.
2. Click **Create Project**.
3. In the **Name** field, enter **user-getting-started**.
4. Click **Create**.

Additional resources

- Viewing a project by using the web console

2.3. GRANTING VIEW PERMISSIONS

Configure the necessary permissions for the application to access the required cluster resources.

OpenShift Container Platform automatically creates several 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.

By default, the **default** service account has limited permissions to interact with the OpenShift API.

As a requirement of the application, you must assign the **view** role to the **default** service account to allow it to communicate with the OpenShift API to learn about pods, services, and resources within the project.

Prerequisites

- You have **cluster-admin** or project-level **admin** privileges.

Procedure

1. Navigate to **User Management → RoleBindings**.

2. Click **Create binding**.
3. In the **Name** field, enter **sa-user-account**.
4. In the **Namespace** field, search for and select **user-getting-started**.

IMPORTANT

If you are using a different project, select the name of your project.

5. In the **Role name** field, search for and select **view**.
6. Under **Subject**, select **ServiceAccount**.
7. In the **Subject namespace** field, search for and select **user-getting-started**.

IMPORTANT

If you are using a different project, select the name of your project.

8. In the **Subject name** field, enter **default**.
9. Click **Create**.

Additional resources

- RBAC overview

2.4. DEPLOYING THE FRONT-END APPLICATION

Deploy the front-end application that provides the external-facing web component for the tutorial.

The simplest way to deploy an application in OpenShift Container Platform is to run a provided container image.

The following procedure deploys **parksmap**, which is the front-end component of the **national-parks-app** application. The web application displays an interactive map of the locations of national parks across the world.

Procedure

1. From the **Quick create** ( ) menu in the upper right corner, click **Container images**.
2. Select **Image name from external registry** and enter **quay.io/openshiftroadshow/parksmap:latest**.
3. Scroll to the **General** section.
4. In the **Application name** field, enter **national-parks-app**.
5. In the **Name** field, ensure that the value is **parksmap**.
6. Scroll to the **Deploy** section.

7. In the **Resource type** field, ensure that **Deployment** is selected.

8. In the **Advanced options** section, ensure that **Create a route** is selected.
   By default, services running on OpenShift Container Platform are not accessible externally. You must select this option to create a route so that external clients can access your service.

9. Click the **Labels** hyperlink.
   The application code requires certain labels to be set.

10. Add the following labels to the text area and press Enter after each key/value pair:

    - **app=national-parks-app**
    - **component=parksmap**
    - **role=frontend**

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

Additional resources

- Viewing the topology of your application

2.4.1. Viewing pod details

Retrieve detailed pod information to confirm the running status and resource configuration of the applications in this tutorial.

OpenShift Container Platform uses 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 and check the status of builds, services, and routes.

Prerequisites

- You have deployed the **parksmap** front-end application.

Procedure

1. Navigate to **Workloads → Topology**.

2. Click the **parksmap** deployment in the **national-parks-app** application.

Figure 2.1. Parksmap deployment

This opens an overview panel with the following tabs:

- **Details**: View details about your deployment, edit certain settings, and scale your deployment.
- **Resources**: View details for the pods, services, and routes associated with your deployment.
- **Observe**: View metrics and events for your deployment.

3. To view the logs for a pod, select the **Resources** tab and click **View logs** next to the **parksmap** pod.

Additional resources

- Interacting with applications and components
- Scaling application pods and checking builds and routes
- Labels and annotations used for the Topology view

2.4.2. Scaling up the application

Scale the application deployment up or down to meet workload demands.

In Kubernetes, a **Deployment** object defines how an application deploys. In most cases when you deploy an application, OpenShift Container Platform creates the **Pod**, **Service**, **ReplicaSet**, and **Deployment** resources for you.

When you deploy the **parksmap** image, a deployment resource is created. In this example, only one pod is deployed. You might want to scale up your application to keep up with user demand or to ensure that your application is always running even if one pod is down.

The following procedure scales the **parksmap** deployment to use two instances.

Prerequisites

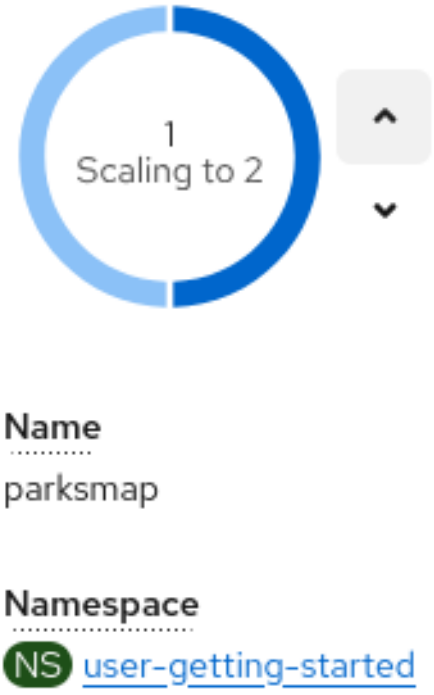
- You have deployed the **parksmap** front-end application.

Procedure

1. Navigate to **Workloads → Topology** and click the **parksmap** deployment.

2. Select the **Details** tab.
3. Use the up arrow to scale the pod to two instances.

Figure 2.2. Scaling application



TIP

You can use the down arrow to scale your deployment back down to one pod instance.

Additional resources

- Recommended practices for scaling the cluster

2.5. DEPLOYING THE BACK-END APPLICATION

Deploy the back-end application that provides the service that queries the database to return the national park data required for your application.

The following procedure deploys **nationalparks**, which is the back-end component for the **national-parks-app** 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.

Prerequisites

- You have deployed the **parksmap** front-end application.

Procedure

1. From the **Quick create** ( ) menu in the upper right corner, click **Import from Git**.
2. In the **Git Repo URL** field, enter **https://github.com/openshift-roadshow/nationalparks-py.git**.

A builder image is automatically detected, but the import strategy defaults to Dockerfile instead of Python.

3. Change the import strategy:
   a. Click **Edit Import Strategy**.
   b. Select **Builder Image**.
   c. Select **Python**.
4. Scroll to the **General** section.
5. In the **Application** field, ensure that the value is **national-parks-app**.
6. In the **Name** field, enter **nationalparks**.
7. Scroll to the **Deploy** section.
8. In the **Resource type** field, ensure that **Deployment** is selected.
9. In the **Advanced options** section, ensure that **Create a route** is selected.
   By default, services running on OpenShift Container Platform are not accessible externally. You must select this option to create a route so that external clients can access your service.
10. Click the **Labels** hyperlink.
    The application code requires certain labels to be set.
11. Add the following labels to the text area and press Enter after each key/value pair:
    - **app=national-parks-app**
    - **component=nationalparks**
    - **role=backend**
    - **type=parksmap-backend**
12. Click **Create**.
    You are redirected to the **Topology** page where you can see the **nationalparks** deployment in the **national-parks-app** application.

Verification

1. Navigate to **Workloads → Topology**.
2. Click the **nationalparks** deployment in the **national-parks-app** application.
3. Click the **Resources** tab.
   Wait for the build to complete successfully.

Additional resources

- Adding services to your application
- Importing a codebase from Git to create an application

2.6. DEPLOYING THE DATABASE APPLICATION

Deploy a MongoDB database application to contain the information that your application requires. For this tutorial, you will deploy a database application called **mongodb-nationalparks** that holds the national park location information.

Prerequisites

- You have deployed the **parksmap** front-end application.
- You have deployed the **nationalparks** back-end application.

Procedure

1. From the **Quick create** () menu in the upper right corner, click **Container images**.
2. Select **Image name from external registry** and enter **registry.redhat.io/rhmap47/mongodb**.
3. In the **Runtime icon** field, search for and select **mongodb**.
4. Scroll to the **General** section.
5. In the **Application name** field, enter **national-parks-app**.
6. In the **Name** field, enter **mongodb-nationalparks**.
7. Scroll to the **Deploy** section.
8. In the **Resource type** field, ensure that **Deployment** is selected.
9. Click **Show advanced Deployment option**.
10. Under **Environment variables (runtime only)**, add the following names and values:

    Table 2.1. Environment variable names and values

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

    TIP

    Click **Add value** to add each additional environment variable.

11. In the **Advanced options** section, clear **Create a route**.
    The database application does not need to be accessed externally, so a route is not required.

12. Click **Create**.
    You are redirected to the **Topology** page where you can see the **mongodb-nationalparks** deployment in the **national-parks-app** application.

2.6.1. Providing access to the database by creating a secret

Create a **Secret** resource to securely provide the back-end application with the sensitive database connection credentials.

The **nationalparks** application needs information, such as the database name, username, and passwords, to access the MongoDB database. However, because this information is sensitive, you should not store it directly in the pod.

You can use a *secret* to store sensitive information, and share that secret with workloads.

Secret objects provide a mechanism to hold sensitive information such as passwords, OpenShift Container Platform client configuration files, and private source repository credentials. Secrets decouple sensitive content from the pods. You can mount secrets into containers by using a volume plugin or by passing the secret in as an environment variable. The system can then use secrets to provide the pod with the sensitive information.

The following procedure creates the **nationalparks-mongodb-parameters** secret and mounts it to the **nationalparks** workload.

Prerequisites

- You have deployed the **nationalparks** back-end application.
- You have deployed the **mongodb-nationalparks** database application.

Procedure

1. Navigate to **Workloads → Secrets**.
2. Click **Create → Key/value secret**
3. In the **Secret name** field, enter **nationalparks-mongodb-parameters**.
4. Enter the following values for **Key** and **Value**:

   Table 2.2. Secret keys and values

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

Key	Value
MONGODB_ADMIN_PASSWORD	**mongodb**

TIP

Click **Add key/value** to add each additional key/value pair.

5. Click **Create**.
6. Click **Add Secret to workload**
7. From the **Add this secret to workload**list, select **nationalparks**.
8. Click **Save**.
   This change in configuration triggers a new rollout of the **nationalparks** deployment with the environment variables properly injected.

Additional resources

- Understanding secrets

2.6.2. Loading data into the database

After you have deployed the **mongodb-nationalparks** database application, load the national park location information into the database.

Prerequisites

- You have deployed the **nationalparks** back-end application.
- You have deployed the **mongodb-nationalparks** database application.

Procedure

1. Navigate to **Workloads → Topology**.
2. Click the **nationalparks** deployment and select the **Resources** tab.
3. Copy the **Location** URL from your route.
4. Paste the URL into your web browser and add the following at the end of the URL:

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

   For example:

   ```
   https://nationalparks-user-getting-started.apps.cluster.example.com/ws/data/load
   ```

   Example output

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

2.7. VIEWING THE APPLICATION IN A WEB BROWSER

After you have deployed the necessary applications and loaded data into the database, you are now ready view your application through a browser. You can access the application by opening the URL for the front-end application.

Prerequisites

- You have deployed the **parksmap** front-end application.
- You have deployed the **nationalparks** back-end application.
- You have deployed the **mongodb-nationalparks** database application.
- You have loaded the data into the **mongodb-nationalparks** database.

Procedure

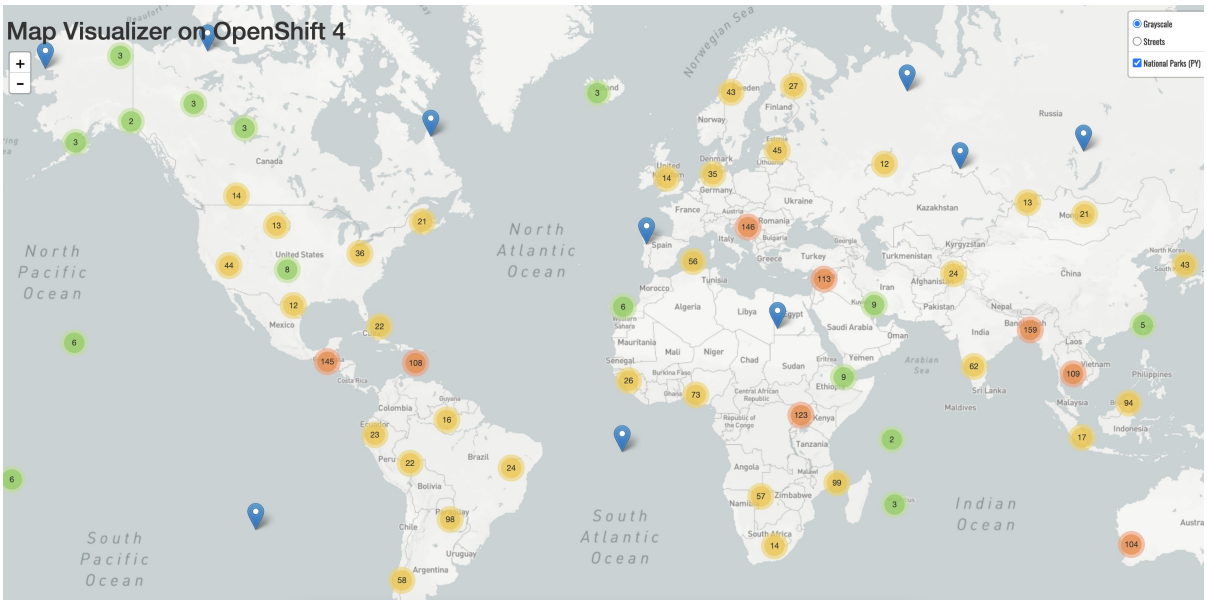
1. Navigate to **Workloads → Topology**.
2. Click the **Open URL** link from the **parksmap** deployment.

   Figure 2.3. National parks across the world

3. Verify that your web browser displays a map of the national parks across the world.

Figure 2.4. National parks across the world



If you allow the application to access your location, the map will center on your location.

CHAPTER 3. TUTORIAL: DEPLOYING AN APPLICATION BY USING THE CLI

To learn how to stand up an application on OpenShift Container Platform by using the OpenShift CLI (**oc**), follow the provided tutorial. In this tutorial, you will deploy the services that are required for an application that displays a map of national parks across the world.

To complete this tutorial, you will perform the following steps:

1. Create a project for the application.
   This step allows your application to be isolated from other cluster user's workloads.

2. Grant view permissions.
   This step grants **view** permissions to interact with the OpenShift API to help discover services and other resources running within the project.

3. Deploy the front-end application.
   This step deploys the **parksmap** front-end application, exposes it externally, and scales it up to two instances.

4. Deploy the back-end application.
   This step deploys the **nationalparks** back-end application and exposes it externally.

5. Deploy the database application.
   This step deploys the **mongodb-nationalparks** MongoDB database, loads data into the database, and sets up the necessary credentials to access the database.

After you complete these steps, you can view the national parks application in a web browser.

3.1. PREREQUISITES

Before you start this tutorial, ensure that you have the following required prerequisites:

- You have installed the OpenShift CLI (**oc**).

- You have access to a test OpenShift Container Platform cluster.
  If your organization does not have a cluster to test on, you can request access to the Developer Sandbox to get a trial of OpenShift Container Platform.

- You have the appropriate permissions, such as the **cluster-admin** cluster role, to create a project and applications within it.
  If you do not have the required permissions, contact your cluster administrator. You need the **self-provisioner** role to create a project and the **admin** role on the project to modify resources in that project.

  If you are using Developer Sandbox, a project is created for you with the required permissions.

- You have logged in to your cluster by using the OpenShift CLI (**oc**).

3.2. CREATING A PROJECT

Create a new project to contain all required resources and application components for the tutorial.

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

Cluster administrators can allow developers to create their own projects. In most cases, you automatically have access to your own projects. Administrators can grant access to other projects as needed.

This procedure creates a new project called **user-getting-started**. You will use this project throughout the rest of this tutorial.

IMPORTANT

If you are using Developer Sandbox to complete this tutorial, skip this procedure. A project has already been created for you.

Prerequisites

- You have logged in to the OpenShift CLI (**oc**).

Procedure

- Create a project by running the following command:

```
$ oc new-project user-getting-started
```

Example output

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

Additional resources

- oc new-project

3.3. GRANTING VIEW PERMISSIONS

Configure the necessary permissions for the application to access the required cluster resources.

OpenShift Container Platform automatically creates several 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.

By default, the **default** service account has limited permissions to interact with the OpenShift API.

As a requirement of the application, you must assign the **view** role to the **default** service account to allow it to communicate with the OpenShift API to learn about pods, services, and resources within the project.

Prerequisites

- You have access to an OpenShift Container Platform cluster.

- You have installed the OpenShift CLI (**oc**).
- You have **cluster-admin** or project-level **admin** privileges.

Procedure

- Add the **view** role to the **default** service account in the **user-getting-started** project by running the following command:

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

IMPORTANT

If you are using a different project, replace **user-getting-started** with the name of your project.

Additional resources

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

3.4. DEPLOYING THE FRONT-END APPLICATION

Deploy the front-end application that provides the external-facing web component for the tutorial.

The simplest way to deploy an application in OpenShift Container Platform is to run a provided container image.

The following procedure deploys **parksmap**, which is the front-end component of the **national-parks-app** application. The web application displays an interactive map of the locations of national parks across the world.

Prerequisites

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

Procedure

- Deploy the **parksmap** application by running 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 (4 years 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
    Application is not exposed. You can expose services to the outside world by executing one or more of the commands below:
     'oc expose service/parksmap'
    Run 'oc status' to view your app.
```

Additional resources

- oc new-app

3.4.1. Exposing the front-end service

By default, services running on OpenShift Container Platform are not accessible externally. To expose your service so that external clients can access it, you can create a *route*.

A **Route** object is a OpenShift Container Platform networking resource similar to a Kubernetes **Ingress** object. 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 have deployed the **parksmap** front-end application.
- You have **cluster-admin** or project-level **admin** privileges.

Procedure

- Create a route to expose the **parksmap** front-end application by running the following command:

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

Verification

- Verify that the application route was successfully created by running the following command:

```
$ oc get route parksmap
```

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

3.4.2. Viewing pod details

Retrieve detailed pod information to confirm the running status and resource configuration of the applications in this tutorial.

OpenShift Container Platform uses 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 have deployed the **parksmap** front-end application.

Procedure

- List all pods in the current project by running the following command:

```
$ oc get pods
```

Example output

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

- Show details for a pod by running the following command:

```
$ oc describe pod parksmap-5f9579955-6sng8
```

Example output

```
Name:             parksmap-5f9579955-6sng8
Namespace:        user-getting-started
Priority:         0
Service Account:  default
Node:             ci-ln-fr1rt92-72292-4fzf9-worker-a-g9g7c/10.0.128.4
Start Time:       Wed, 26 Mar 2025 14:03:19 -0400
Labels:           app=national-parks-app
                  app.kubernetes.io/part-of=national-parks-app
                  component=parksmap
                  deployment=parksmap
                  pod-template-hash=848bd4954b
                  role=frontend
...
```

- View logs for a pod by running the following command:

```
$ oc logs parksmap-5f9579955-6sng8
```

Example output

```
...
2025-03-26 18:03:24.774  INFO 1 --- [          main] o.s.m.s.b.SimpleBrokerMessageHandler
: Started.
2025-03-26 18:03:24.798  INFO 1 --- [          main]
s.b.c.e.t.TomcatEmbeddedServletContainer : Tomcat started on port(s): 8080 (http)
2025-03-26 18:03:24.801  INFO 1 --- [          main] c.o.evg.roadshow.ParksMapApplication
: Started ParksMapApplication in 4.053 seconds (JVM running for 4.46)
```

Additional resources

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

3.4.3. Scaling up the deployment

Scale the application deployment up or down to meet workload demands.

In Kubernetes, a **Deployment** object defines how an application deploys. In most cases when you deploy an application, OpenShift Container Platform creates the **Pod**, **Service**, **ReplicaSet**, and **Deployment** resources for you.

When you deploy the **parksmap** image, a deployment resource is created. In this example, only one pod is deployed. You might want to scale up your application to keep up with user demand or to ensure that your application is always running even if one pod is down.

The following procedure scales the **parksmap** deployment to use two instances.

Prerequisites

- You have deployed the **parksmap** front-end application.

Procedure

- Scale your deployment from one pod instance to two pod instances by running the following command:

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

Example output

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

Verification

- Verify that your deployment scaled up properly by running 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
```

Verify that two **parksmap** pods are listed.

TIP

To scale your deployment back down to one pod instance, pass in **1** to the **--replicas** option:

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

Additional resources

- oc scale

3.5. DEPLOYING THE BACK-END APPLICATION

Deploy the back-end application that provides the service that queries the database to return the national park data required for your application.

The following procedure deploys **nationalparks**, which is the back-end component for the **national-parks-app** 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.

Prerequisites

- You have deployed the **parksmap** front-end application.

Procedure

- Create the **nationalparks** back-end application by running 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 9531750 (2 weeks old) in image stream "openshift/python" under tag "3.11-ubi8" for "python"

    Python 3.11
    -----------
...
```

```
--> 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
    Build scheduled, use 'oc logs -f buildconfig/nationalparks' to track its progress.
    Application is not exposed. You can expose services to the outside world by executing one
or more of the commands below:
     'oc expose service/nationalparks'
    Run 'oc status' to view your app.
```

3.5.1. Exposing the back-end service

To expose the back-end service so that it is accessible externally, create a route.

Prerequisites

- You have deployed the **nationalparks** back-end application.
- You have **cluster-admin** or project-level **admin** privileges.

Procedure

1. Create a route to expose the **nationalparks** back-end application by running the following command:

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

2. Label the **nationalparks** route by running the following command:

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

   The application code expects the **nationalparks** route to be labeled with **type=parksmap-backend**.

Additional resources

- oc label

3.6. DEPLOYING THE DATABASE APPLICATION

Deploy a MongoDB database application to contain the information that your application requires. For this tutorial, you will deploy a database application called **mongodb-nationalparks** that holds the national park location information.

Prerequisites

- You have deployed the **parksmap** front-end application.

- You have deployed the **nationalparks** back-end application.

Procedure

- Deploy the **mongodb-nationalparks** database application by running the following command:

```
$ oc new-app registry.redhat.io/rhmap47/mongodb --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 7a61087 (12 days old) from quay.io for
"quay.io/mongodb/mongodb-enterprise-server"

    * 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
    Application is not exposed. You can expose services to the outside world by executing one
or more of the commands below:
     'oc expose service/mongodb-nationalparks'
    Run 'oc status' to view your app.
```

3.6.1. Providing access to the database by creating a secret

Create a **Secret** resource to securely provide the back-end application with the sensitive database connection credentials.

The **nationalparks** application needs information, such as the database name, username, and passwords, to access the MongoDB database. However, because this information is sensitive, you should not store it directly in the pod.

You can use a *secret* to store sensitive information, and share that secret with workloads.

Secret objects provide a mechanism to hold sensitive information such as passwords, OpenShift Container Platform client configuration files, and private source repository credentials. Secrets decouple sensitive content from the pods. You can mount secrets into containers by using a volume plugin or by passing the secret in as an environment variable. The system can then use secrets to provide the pod with the sensitive information.

The following procedure creates the **nationalparks-mongodb-parameters** secret and mounts it to the **nationalparks** workload.

Prerequisites

- You have deployed the **nationalparks** back-end application.

- You have deployed the **mongodb-nationalparks** database application.

Procedure

1. Create the secret with the required database access information by running 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
```

2. Import the environment from the secret to the **nationalparks** workload by running the following command:

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

3. Wait for the **nationalparks** deployment to roll out a new revision with this environment information. Check the status of the **nationalparks** deployment by running the following command:

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

Example output

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

Additional resources

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

3.6.2. Loading data into the database

After you have deployed the **mongodb-nationalparks** database application, load the national park location information into the database.

Prerequisites

- You have deployed the **nationalparks** back-end application.
- You have deployed the **mongodb-nationalparks** database application.

Procedure

- Load the national parks data by running 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"
```

Verification

- Verify that the map data was loaded properly by running 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"}]
```

Additional resources

- oc exec

3.7. VIEWING THE APPLICATION IN A WEB BROWSER

After you have deployed the necessary applications and loaded data into the database, you are now ready view your application through a browser. You can get the URL for the application by retrieving the route information for the front-end application.

Prerequisites

- You have deployed the **parksmap** front-end application.
- You have deployed the **nationalparks** back-end application.
- You have deployed the **mongodb-nationalparks** database application.
- You have loaded the data into the **mongodb-nationalparks** database.

Procedure

1. Get your route information to retrieve your map application URL by running the following command:

```
$ oc get route parksmap
```

Example output

```
NAME       HOST/PORT                                          PATH   SERVICES    PORT
TERMINATION   WILDCARD
parksmap   parksmap-user-getting-started.apps.cluster.example.com          parksmap
```

```
8080-tcp    edge          None
```

2. From the above output, copy the value in the **HOST/PORT** column.

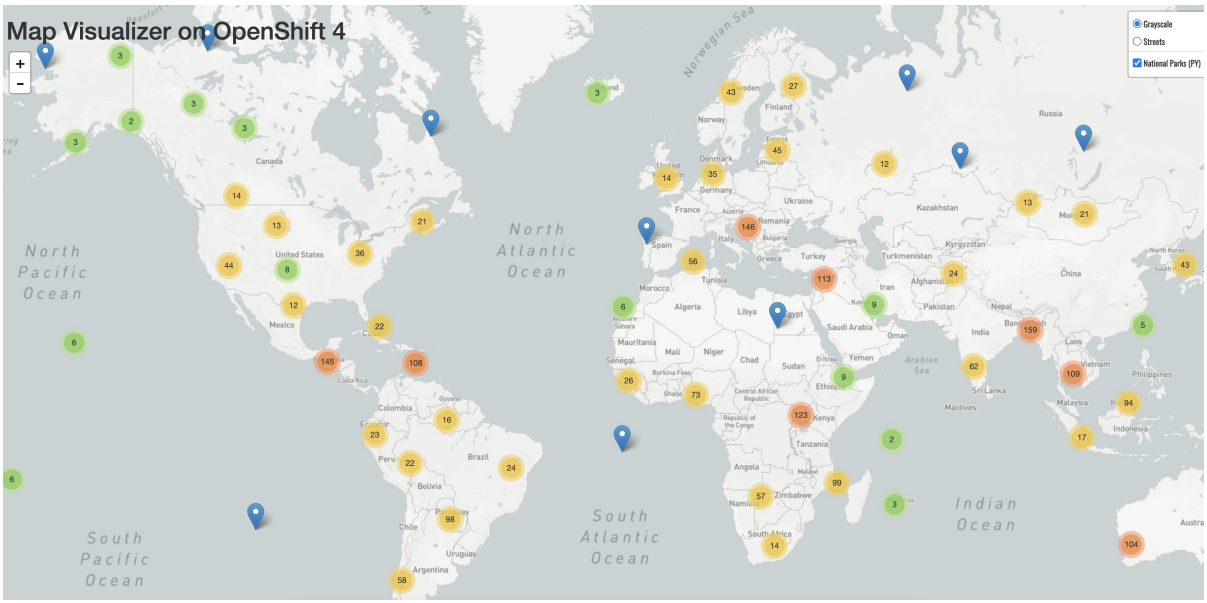
3. Add **https://** in front of the copied value to get the application URL. This is necessary because the route is a secured route.

   Example application URL

   ```
   https://parksmap-user-getting-started.apps.cluster.example.com
   ```

4. Paste this application URL into your web browser. Your browser should display a map of the national parks across the world.

   Figure 3.1. National parks across the world



   If you allow the application to access your location, the map will center on your location.

CHAPTER 4. ADDITIONAL HANDS-ON LEARNING

Explore additional learning resources provided by Red Hat for administrators and developers to gain hands-on experience with OpenShift Container Platform.

4.1. RED HAT DEVELOPER LEARNING PATHS

The Red Hat Developer program provides several learning paths for developers to get started working with OpenShift Container Platform.

The following table lists several recommended learning paths for OpenShift Container Platform:

Table 4.1. Red Hat Developer learning paths

Learning path	Description
Foundations of OpenShift	This learning path covers basic Red Hat OpenShift concepts and how to create and deploy applications through various methods.
Using OpenShift	This learning path covers managing cluster access, database operations, and resource management.
Developing applications on OpenShift	This learning path covers deploying applications from source code and images, and developing with Node.js.
How to deploy full-stack JavaScript applications in OpenShift	This learning path covers how to deploy a full-stack JavaScript application in an OpenShift Container Platform cluster.
Store persistent data in Red Hat OpenShift using PVCs	This learning path covers how to create and use persistent volume claims (PVCs) for persistent storage in OpenShift Container Platform.

For the full list of available Red Hat Developer learning paths for OpenShift Container Platform, see OpenShift and Kubernetes learning.

4.2. RED HAT TRAINING COURSES

Red Hat Training offers a variety of courses to help you learn Red Hat OpenShift and related technologies. Free and paid courses are available online and in-person.

The following tables list several recommended training courses for OpenShift Container Platform, both for developers and administrators:

Table 4.2. Red Hat Training courses for developers

Course	Description

Course	Description
DO101: Introduction to OpenShift Applications	This course helps developers learn to deploy, scale, and troubleshoot applications in OpenShift Container Platform.
DO188: Red Hat OpenShift Development I: Introduction to Containers with Podman	This course helps developers learn to build, run, and manage containers with Podman and OpenShift Container Platform.
DO288: Red Hat OpenShift Developer II: Building and Deploying Cloud-native Applications	This course helps developers learn to design, build, and deploy containerized software applications on an OpenShift Container Platform cluster.

Table 4.3. Red Hat Training courses for administrators

Course	Description
DO180: Red Hat OpenShift Administration I: Operating a Production Cluster	This course helps cluster administrators learn to manage OpenShift Container Platform clusters and collaborate with developers to support application workloads.
DO280: Red Hat OpenShift Administration II: Configuring a Production Cluster	This course helps cluster administrators learn to configure security features, manage Operators, and perform cluster updates.
DO322: Red Hat OpenShift Installation Lab	This course helps cluster administrators learn to install OpenShift Container Platform clusters in various environments.

For the full list of available courses, see Red Hat Training and Certification. You can also take the skills assessment to get recommendations for where to start learning.

4.3. RED HAT CHEAT SHEETS

Red Hat publishes several cheat sheets that provide quick references of common OpenShift CLI (**oc**) commands for working with OpenShift Container Platform.

The following table lists several recommended cheat sheets for OpenShift Container Platform:

Table 4.4. Red Hat cheat sheets

Cheat sheet	Description
Red Hat OpenShift Cheat Sheet	This cheat sheet provides many OpenShift CLI (**oc**) commands for managing an application's lifecycle.

Cheat sheet	Description
OpenShift command line essentials cheat sheet	This cheat sheet provides a quick look at several essential OpenShift CLI (**oc**) commands, such as creating applications, debugging, and editing deployments.

For the full list of available cheat sheets, see Red Hat Developer cheat sheets.