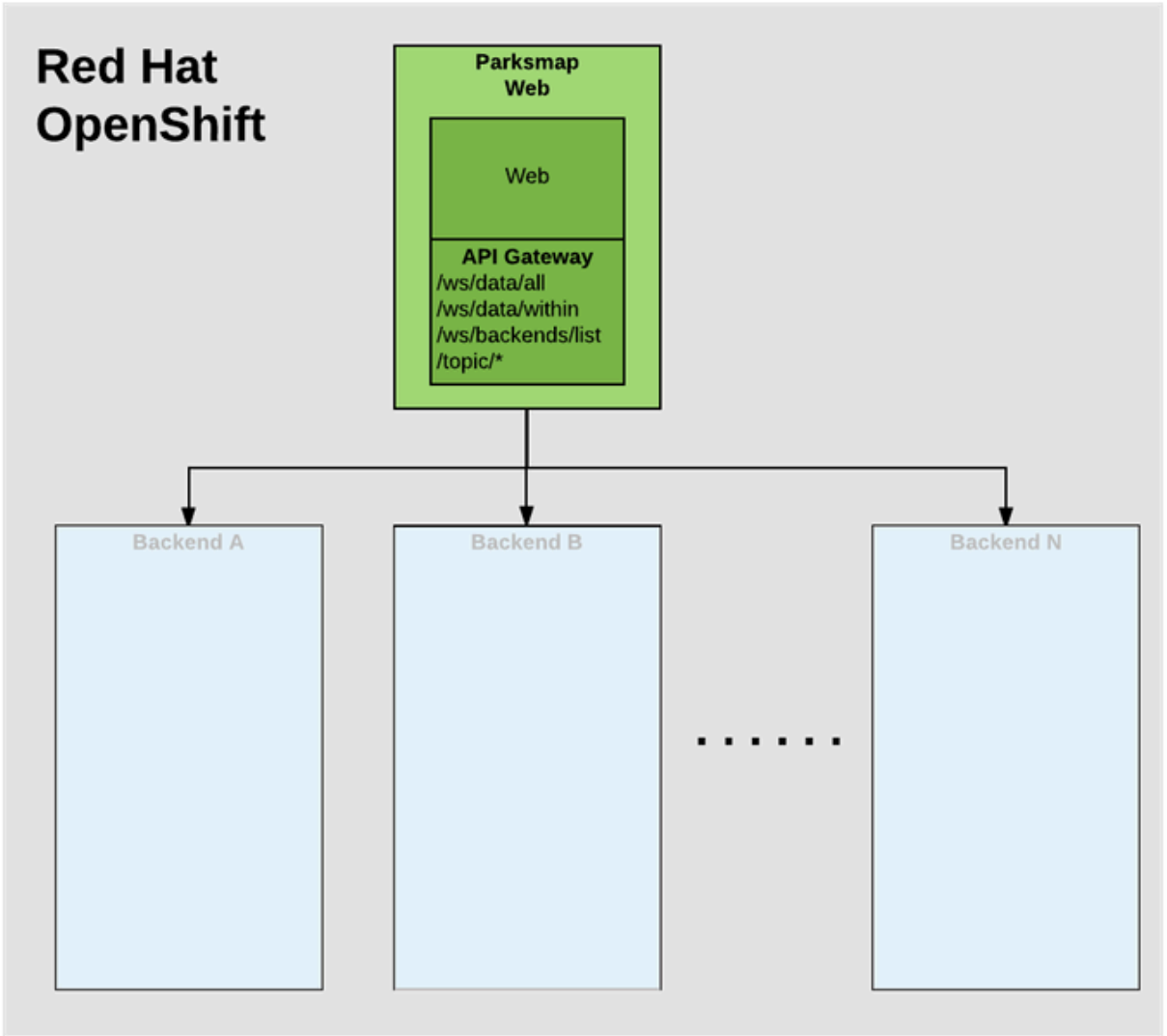
Deploying Your First Container Image

In this lab, we're going to deploy the web component of the ParksMap application which is also called `parksmap` and uses OpenShift's service discovery mechanism to discover the backend services deployed and shows their data on the map.



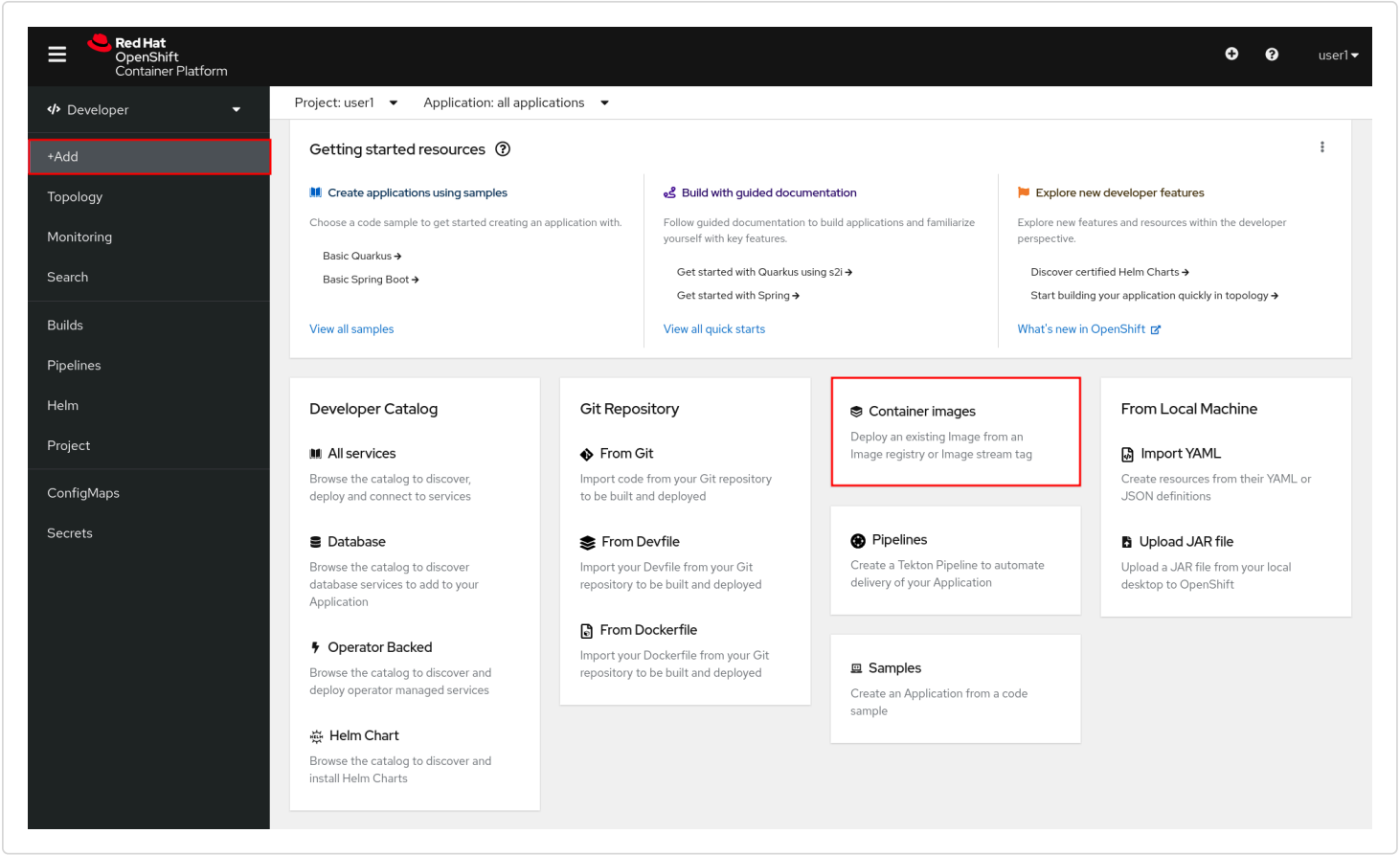
Exercise: Deploying your First Image

Let's start by doing the simplest thing possible - get a plain old Docker-formatted image to run on OpenShift. This is incredibly simple to do. With OpenShift it can be done directly from the web console.

Return to the Web Console.

If you're no longer on the Developer perspective, return there now.

From the left menu, click **+Add**. You will see a screen where you have multiple options to deploy application to OpenShift. Click **Container Image** to open a dialog that will allow you to specify the information for the image you want to deploy.

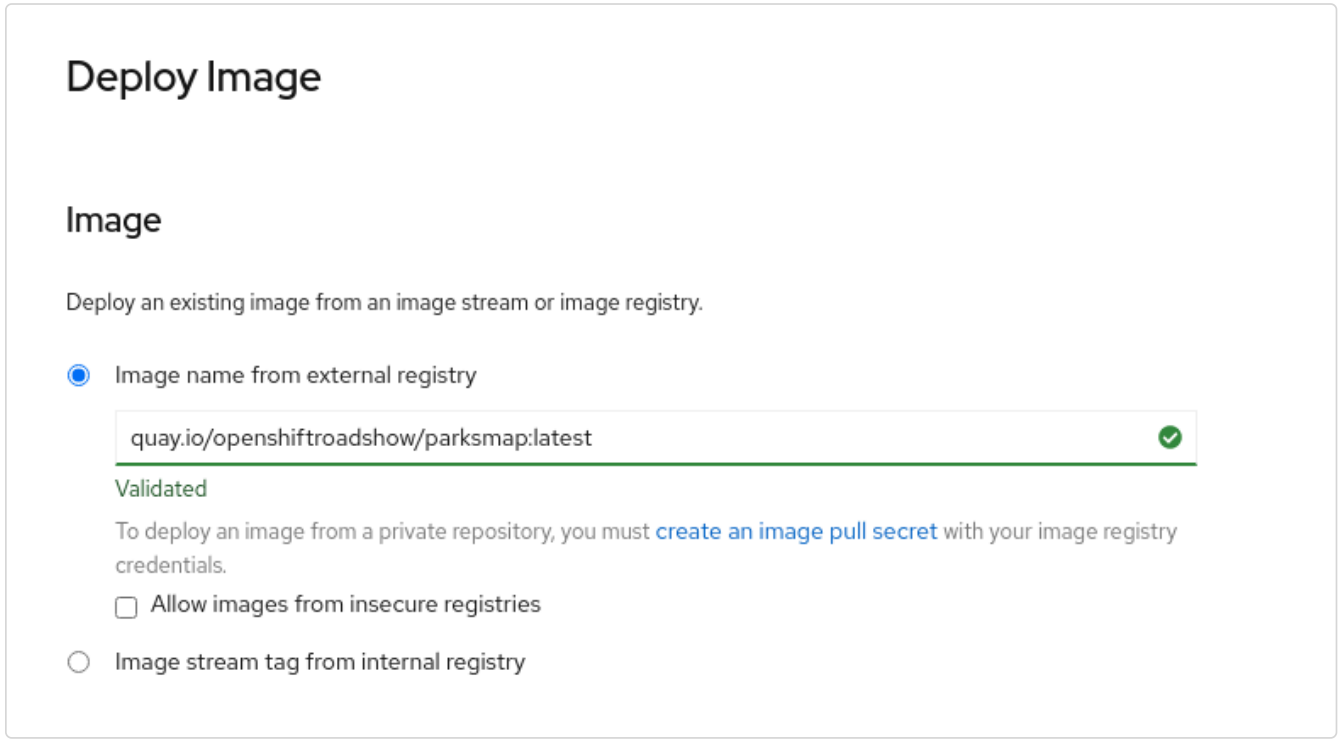


In the **Image Name** field, copy/paste the following into the box:

```
quay.io/openshiftroadshow/parksmap:latest
```

OpenShift will then go out to the container registry specified and interrogate the image.

Your screen will end up looking something like this:



In **Runtime Icon** you can select the icon to use in OpenShift Topology View for the app. You can leave the default OpenShift icon, or select one from the list.

> The purpose of this exercise is to deploy a microservice from an agnostic existing container image (Frontend, this was made with Spring Boot). The specific programming language path you have chosen is described and implemented in the next microservice chapter (Backend).

Make sure to have the correct values in:

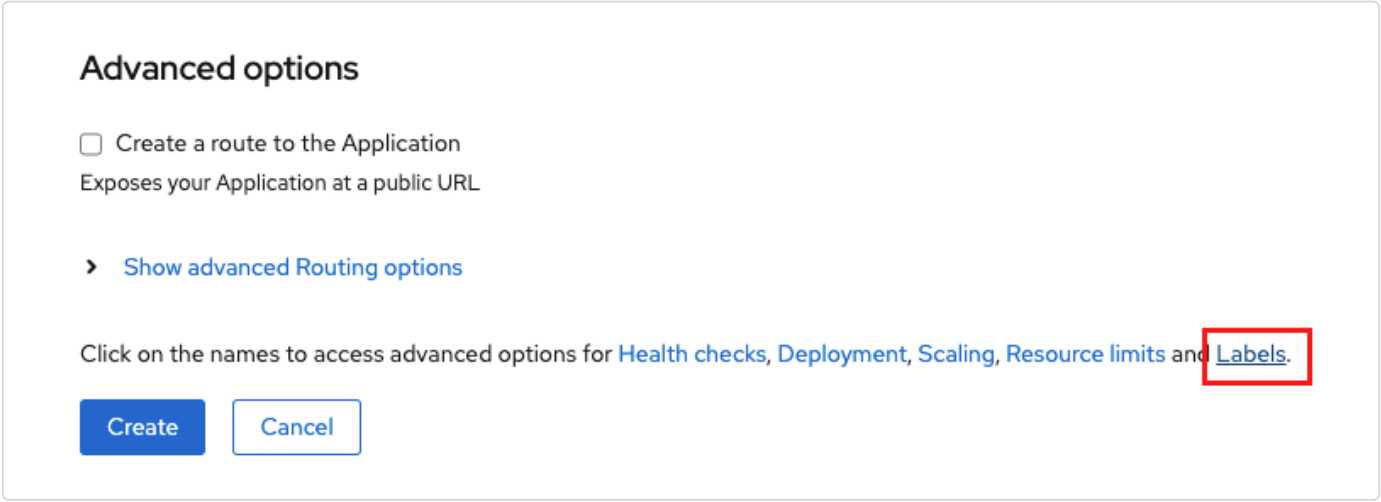
- **Application Name** : workshop

- **Name** : parksmap

Ensure **Deployment** is selected from **Resource** section.

Un-check the checkbox next to **Create a route to the application**. For learning purposes, we will create a **Route** for the application later in the workshop.

At the bottom of the page, click **Labels** in the Advanced Options section and add some labels to better identify this deployment later. Labels will help us identify and filter components in the web console and in the command line.



We will add 3 labels. After you enter the name=value pair for each label, press **tab** or de-focus with mouse before typing the next. First the name to be given to the application.

```
app=workshop
```

Next the name of this deployment.

```
component=parksmap
```

And finally, the role this component plays in the overall application.

```
role=frontend
```

Deploy Image

Image

Deploy an existing image from an image stream or image registry.

- (•) Image name from external registry

  quay.io/openshiftroadshow/parksmap:1.3.0

  Validated

  To deploy an image from a private repository, you must create an image pull secret with your image registry credentials.

  - [ ] Allow images from insecure registries

- ( ) Image stream tag from internal registry

Runtime Icon

spring-boot

The icon represents your image in Topology view. A label will also be added to the resource defining the icon.

General

Application Name

workshop

A unique name given to the application grouping to label your resources.

Name *

parksmap

A unique name given to the component that will be used to name associated resources.

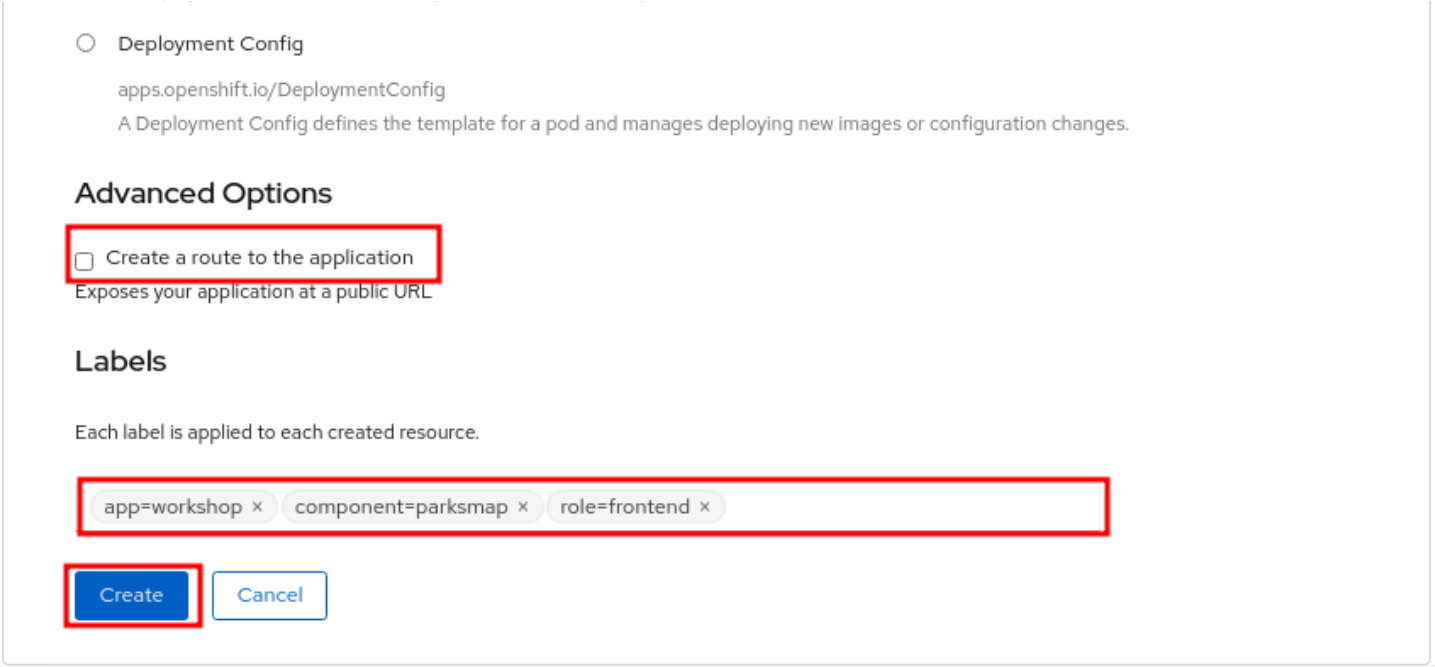
Resources

Select the resource type to generate

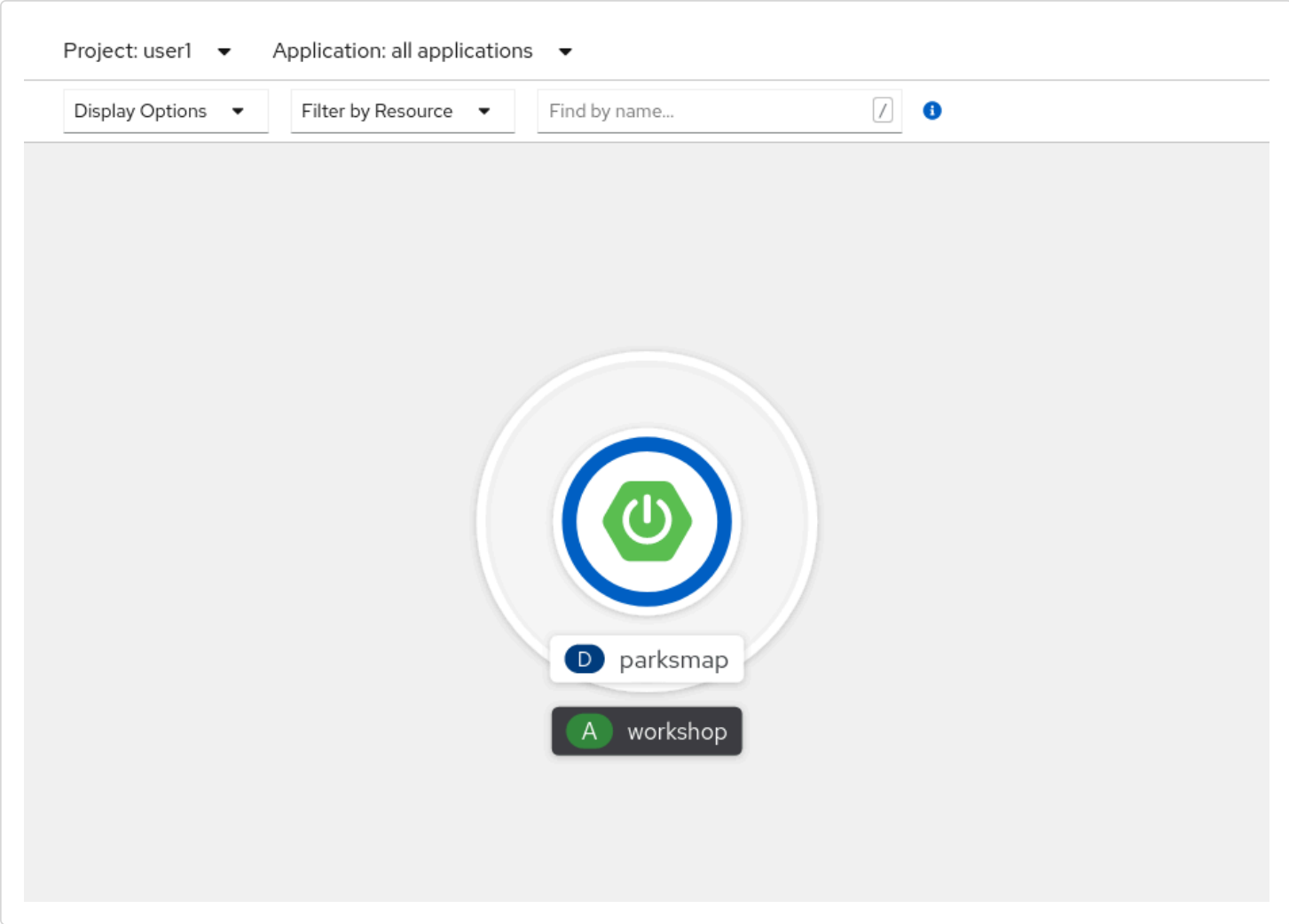
- (•) Deployment

  apps/Deployment

  A Deployment enables declarative updates for Pods and ReplicaSets.



Next, click the blue **Create** button. You will be directed to the **Topology** page, where you should see the visualization for the `parksmap` deployment config in the `workshop` application.



These few steps are the only ones you need to run to get a container image deployed on OpenShift. This should work with any container image that follows best practices, such as defining an EXPOSE port, not needing to run specifically as the **root user** or other user name, and a single non-exiting CMD to execute on start.

> Providing appropriate labels is desired when deploying complex applications for organization purposes. OpenShift uses a label **app** to define and group components together in the Overview page. OpenShift will create this label with some default if the user doesn't provide it explicitly.

Background: Containers and Pods

Before we start digging in, we need to understand how containers and **Pods** are related. We will not be covering the background on these technologies in this lab but if you have questions please inform the instructor. Instead, we will dive right in and start using them.

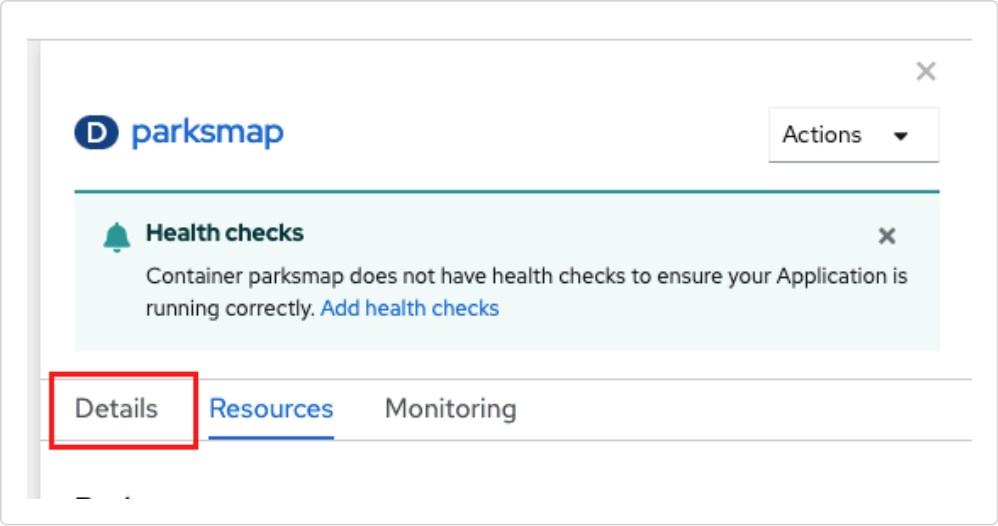
In OpenShift, the smallest deployable unit is a **Pod**. A **Pod** is a group of one or more OCI containers deployed together and guaranteed to be on the same host. From the official OpenShift documentation:

> *Each **Pod** has its own IP address, therefore owning its entire port space, and containers within pods can share storage. **Pods** can be "tagged" with one or more labels, which are then used to select and manage groups of **pods** in a single operation.*

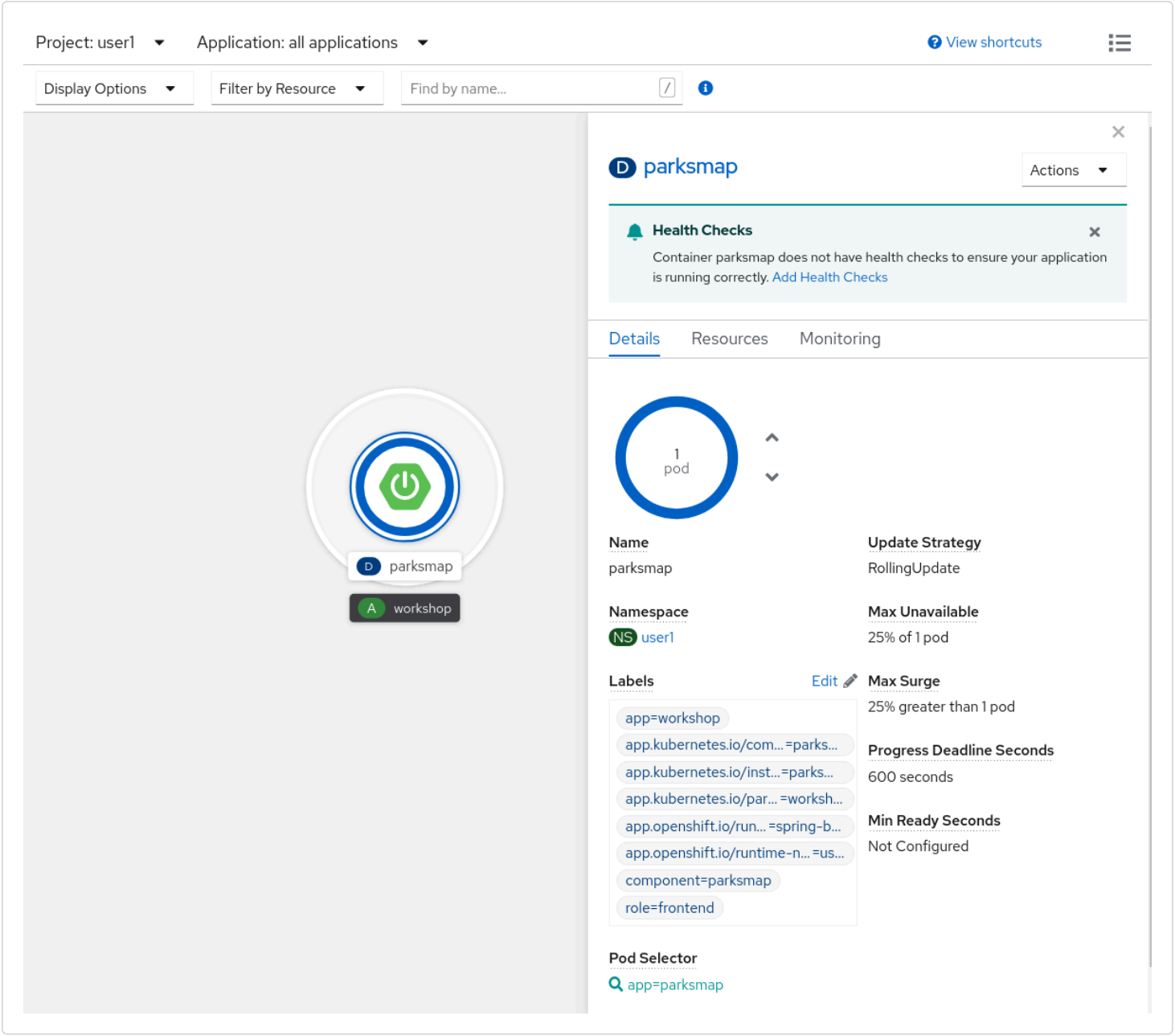
Pods can contain multiple OCI containers. The general idea is for a **Pod** to contain a "main process" and any auxiliary services you want to run along with that process. Examples of containers you might put in a **Pod** are, an Apache HTTPD server, a log analyzer, and a file service to help manage uploaded files.

Exercise: Examining the Pod

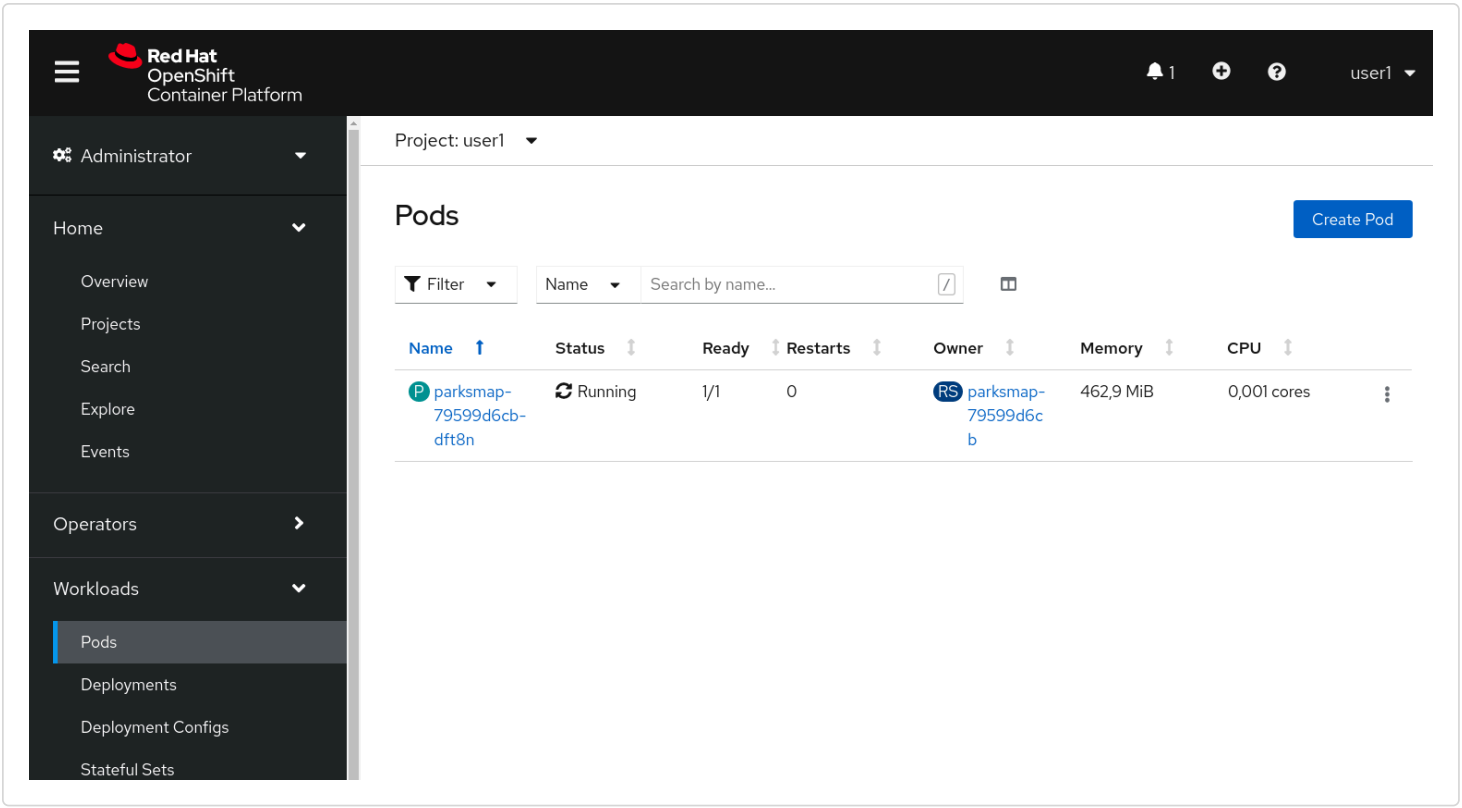
If you click on the `parksmap` entry in the Topology view, you will see some information about that deployment config. The **Resources** tab may be displayed by default. If so, click on the **Details** tab.



On that panel, you will see that there is a single **Pod** that was created by your actions.



You can also get a list of all the **Pods** created within your **Project**, by navigating to **Workloads** → **Pods** in the Administrator perspective of the web console.



This **Pod** contains a single container, which happens to be the `parksmap` application - a simple Spring Boot/Java application.

You can also examine **Pods** from the command line:

```
oc get pods
```

You should see output that looks similar to:

```
NAME                 READY   STATUS      RESTARTS   AGE
parksmap-65c4f8b676-k5gkk    1/1     Running     0          20s
```

The above output lists all of the **Pods** in the current **Project**, including the **Pod** name, state, restarts, and uptime. Once you have a **Pod**'s name, you can get more information about the **Pod** using the `oc get` command. To make the output readable, I suggest changing the output type to **YAML** using the following syntax:

> Make sure you use the correct **Pod** name from your output.

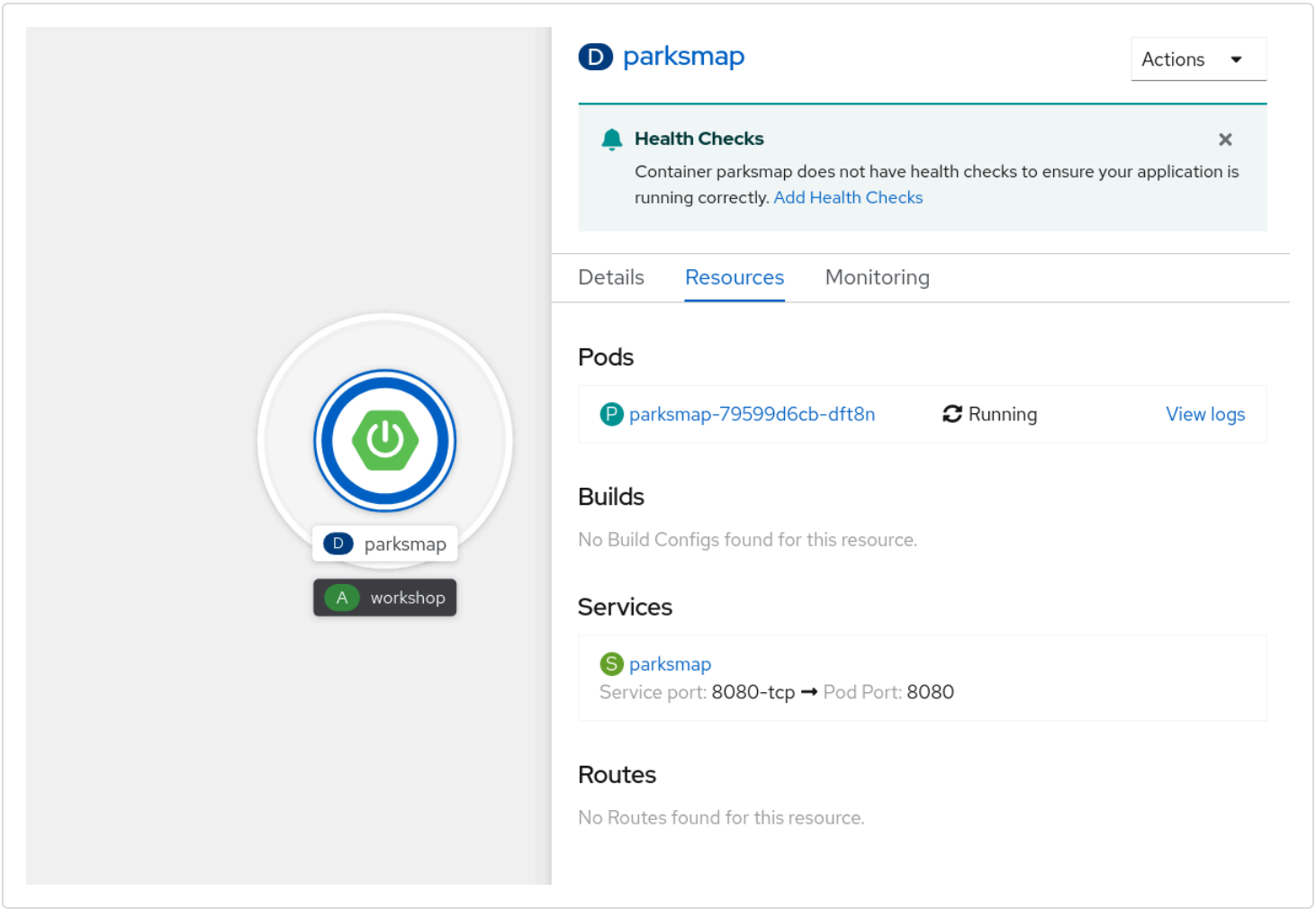
```
oc get pod parksmap-65c4f8b676-k5gkk -o yaml
```

You should see something like the following output (which has been truncated due to space considerations of this workshop manual):

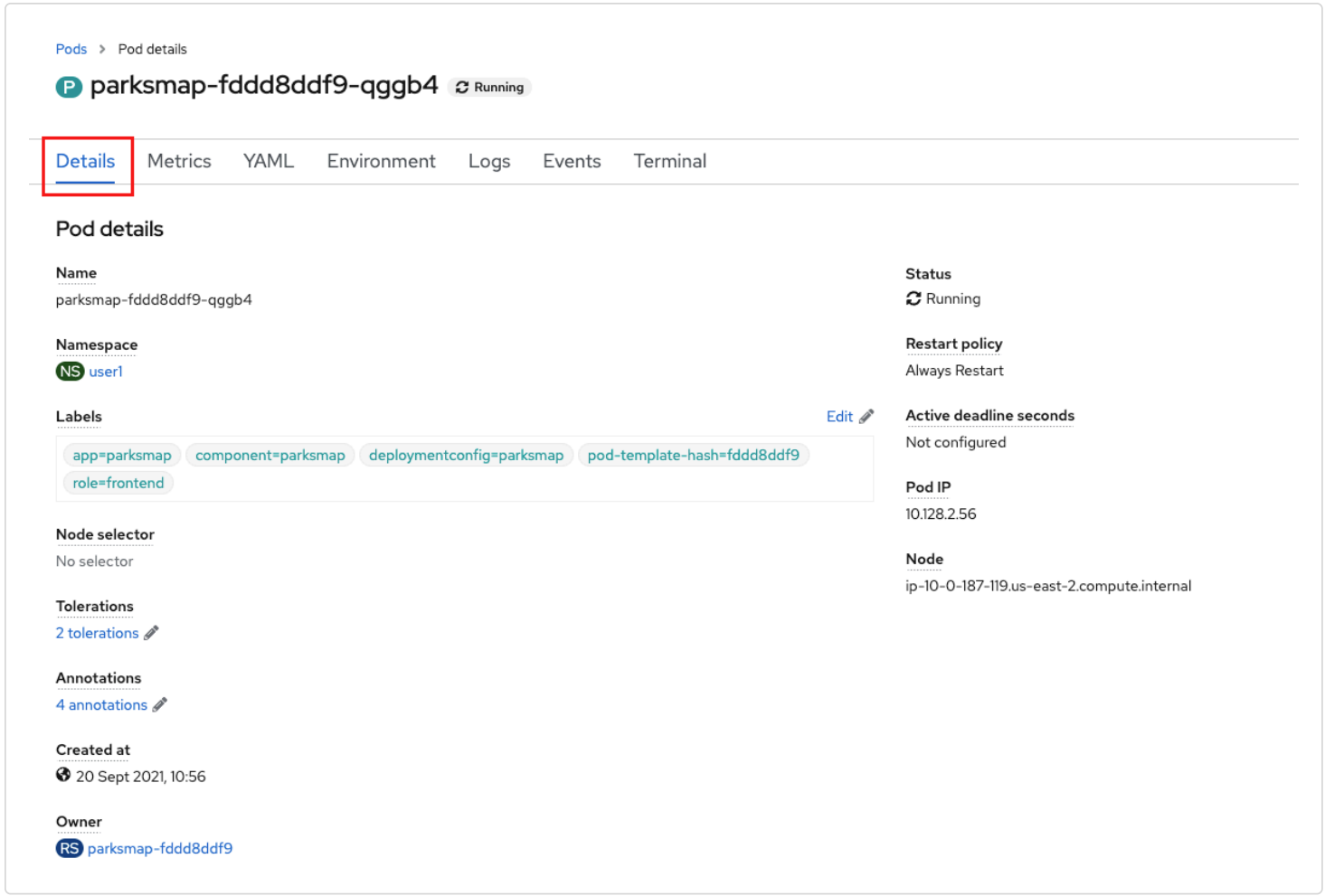
```
apiVersion: v1
kind: Pod
metadata:
  annotations:
    k8s.v1.cni.cncf.io/network-status: |-
      [{
          "name": "",
          "interface": "eth0",
          "ips": [
              "10.131.0.93"
          ],
          "default": true,
          "dns": {}
      }]
    k8s.v1.cni.cncf.io/networks-status: |-
      [{
          "name": "",
          "interface": "eth0",
          "ips": [
              "10.131.0.93"
          ],
```

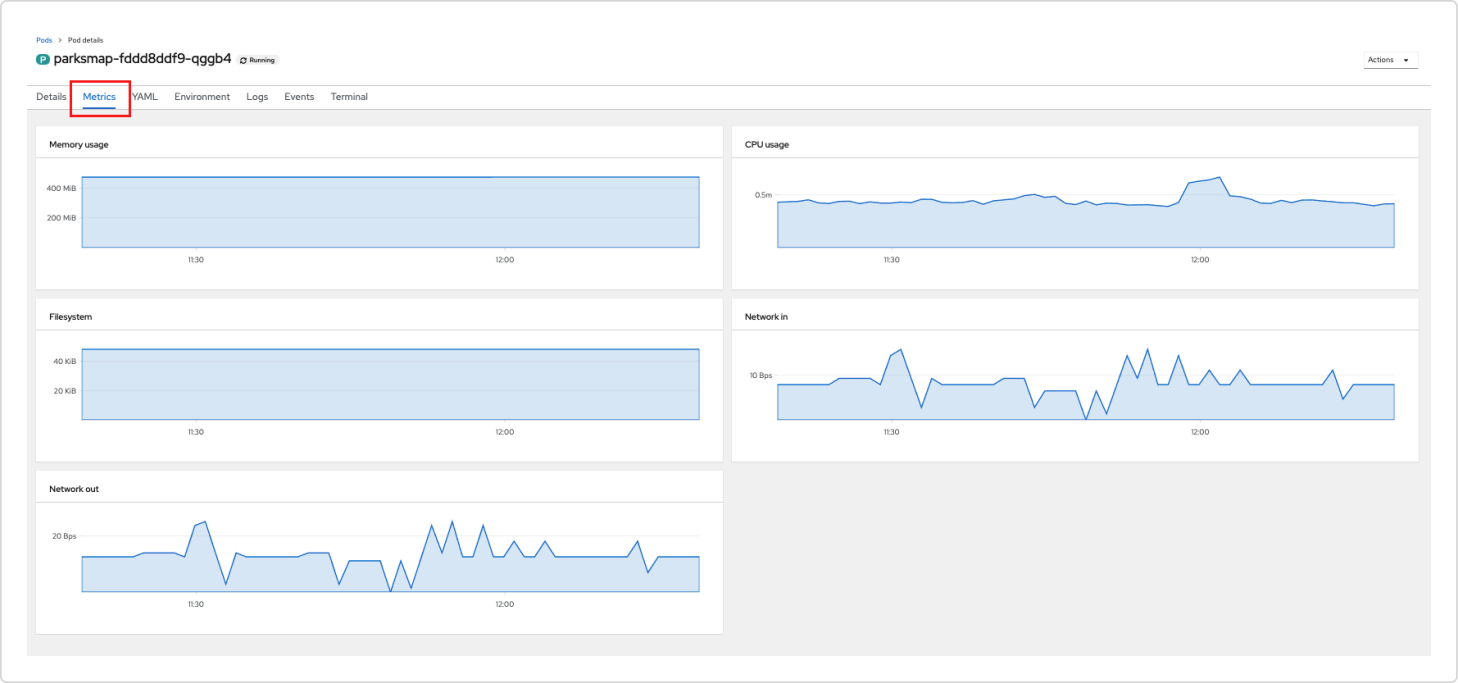
```
          "default": true,
          "dns": {}
      }]
    openshift.io/generated-by: OpenShiftWebConsole
    openshift.io/scc: restricted
  creationTimestamp: "2021-01-05T17:00:32Z"
  generateName: parksmap-65c4f8b676-
  labels:
    app: parksmap
    component: parksmap
    deployment: parksmap
    pod-template-hash: 65c4f8b676
    role: frontend
...............
```

The web interface also shows a lot of the same information on the **Pod** details page. If you click on the name of the **Pod**, you will find the details page. You can also get there by clicking on the `parksmap` deployment config on the **Topology** page, selecting **Resources**, and then clicking the **Pod** name.



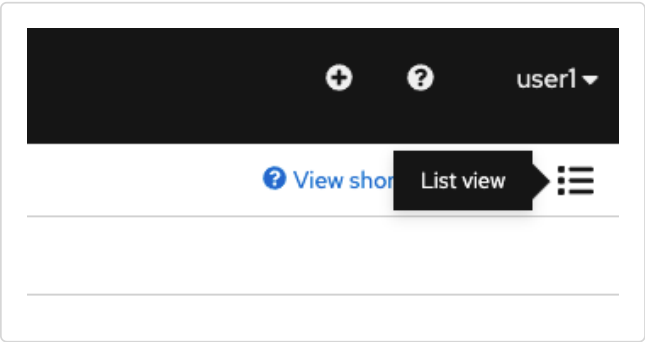
From here you can see configuration, metrics, environment variables, logs, events and get a Terminal shell on the running pod.

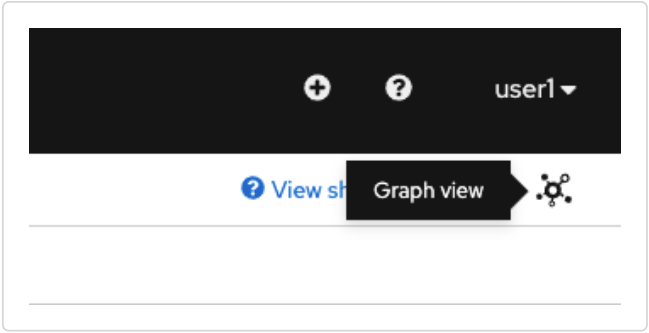

Pods > Pod details
parksmap-fddd8ddf9-qggb4
Running
Details
Metrics
YAML
Environment
Logs
Events
Terminal
Pod details
Name
parksmap-fddd8ddf9-qggb4
Namespace
NS user1
Labels
Edit
app=parksmap
component=parksmap
deploymentconfig=parksmap
pod-template-hash=fddd8ddf9
role=frontend
Node selector
No selector
Tolerations
2 tolerations
Annotations
4 annotations
Created at
20 Sept 2021, 10:56
Owner
RS parksmap-fddd8ddf9
Status
Running
Restart policy
Always Restart
Active deadline seconds
Not configured
Pod IP
10.128.2.56
Node
ip-10-0-187-119.us-east-2.compute.internal



Getting the `parksmap` image running may take a little while to complete. Each OpenShift node that is asked to run the image has to pull (download) it, if the node does not already have it cached locally. You can check on the status of the image download and deployment in the **Pod** details page, or from the command line with the `oc get pods` command that you used before.

The default view in the **Developer** console is **Graph View**. You can switch between **Graph** and **List** views by using the toggle in the top right of the console.





Background: Customizing the Image Lifecycle Behavior

Whenever OpenShift asks the node's CRI (Container Runtime Interface) runtime (Docker daemon or CRI-O) to run an image, the runtime will check to make sure it has the right "version" of the image to run. If it doesn't, it will pull it from the specified registry.

There are a number of ways to customize this behavior. They are documented in specifying an image as well as image pull policy.

Background: Services

Services provide a convenient abstraction layer inside OpenShift to find a group of similar **Pods**. They also act as an internal proxy/load balancer between those **Pods** and anything else that needs to access them from inside the OpenShift environment. For example, if you needed more `parksmap` instances to handle the load, you could spin up more **Pods**. OpenShift automatically maps them as endpoints to the **Service**, and the incoming requests would not notice anything different except that the **Service** was now doing a better job handling the requests.

When you asked OpenShift to run the image, it automatically created a **Service** for you. Remember that services are an internal construct. They are not available to the "outside world", or anything that is outside the OpenShift environment. That's okay, as you will learn later.

The way that a **Service** maps to a set of **Pods** is via a system of **Labels** and **Selectors**. **Services** are assigned a fixed IP address and many ports and protocols can be mapped.

There is a lot more information about Services, including the YAML format to make one by hand, in the official documentation.

Now that we understand the basics of what a **Service** is, let's take a look at the **Service** that was created for the image that we just deployed. In order to view the **Services** defined in your **Project**, enter in the following command:

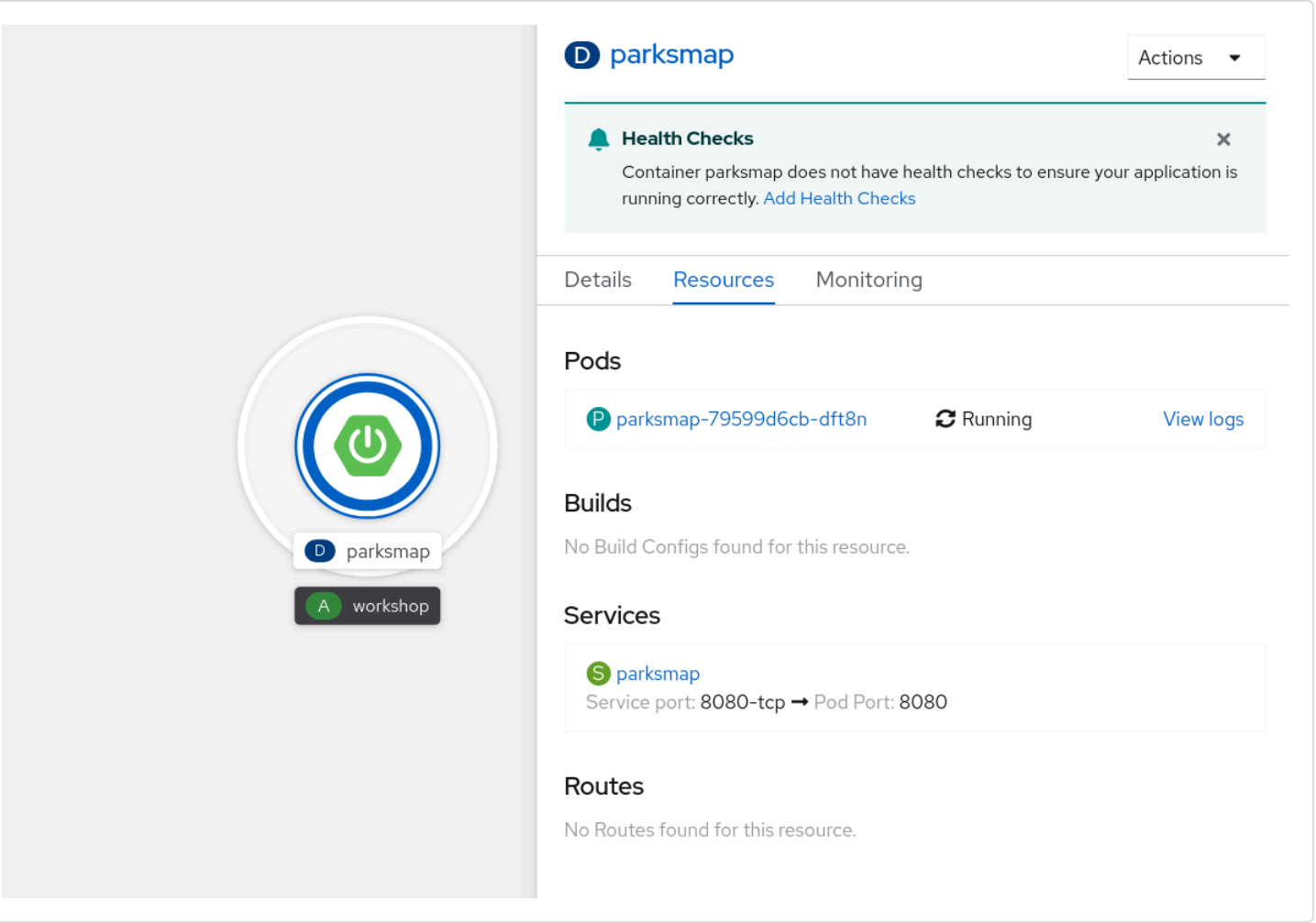
```
oc get services
```

You should see output similar to the following:

```
NAME       TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
parksmap   ClusterIP   172.30.22.209   <none>        8080/TCP   3h
```

In the above output, we can see that we have a **Service** named `parksmap` with an IP/Port combination of 172.30.22.209/8080TCP. Your IP address may be different, as each **Service** receives a unique IP address upon creation. **Service** IPs are fixed and never change for the life of the **Service**.

In the Developer perspective from the **Topology** view, service information is available by clicking the `parksmap` deployment config, then **Resources**, and then you should see the `parksmap` entry in the **Services** section.



You can also get more detailed information about a **Service** by using the following command to display the data in YAML:

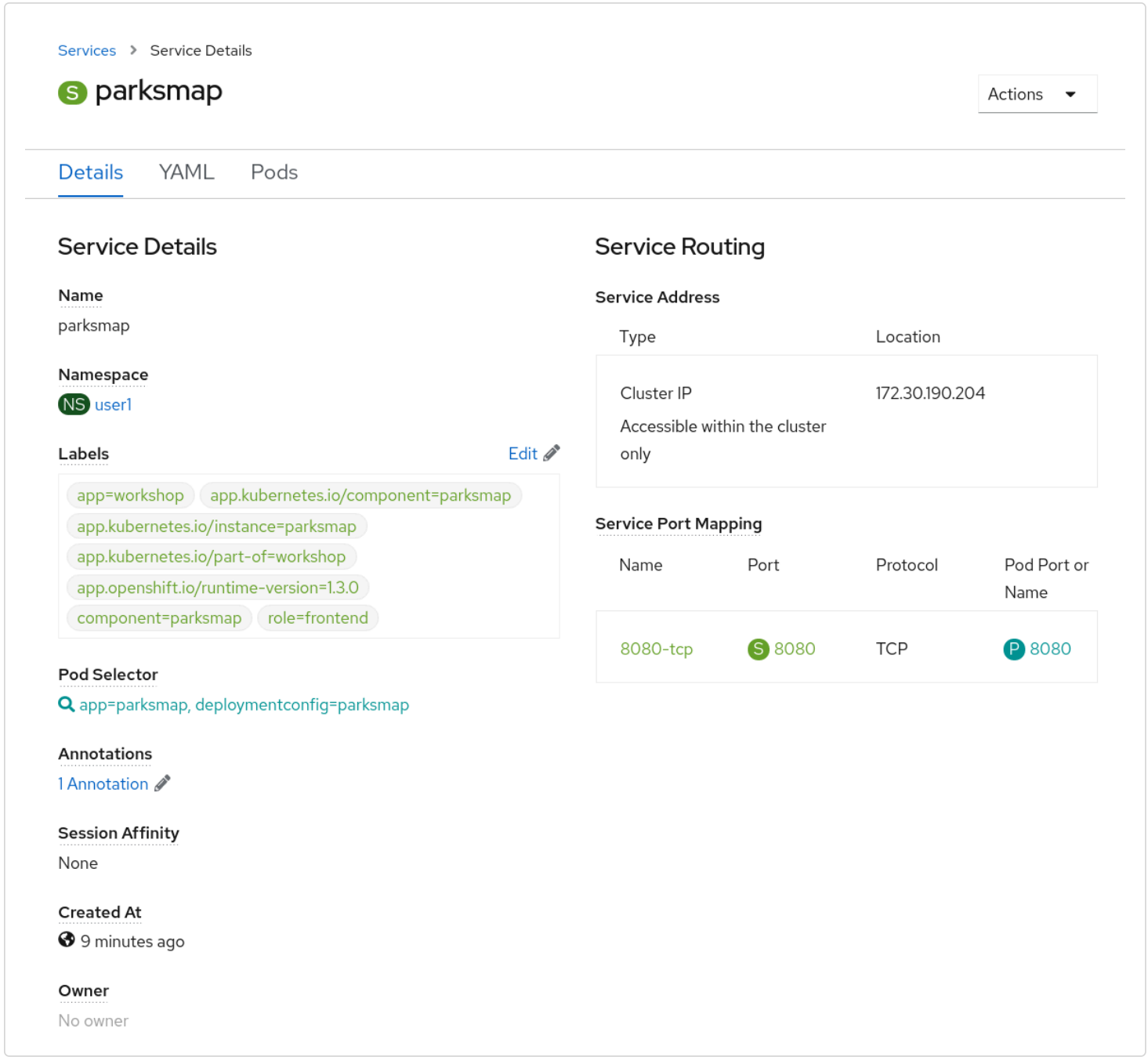
```
oc get service parksmap -o yaml
```

You should see output similar to the following:

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    openshift.io/generated-by: OpenShiftWebConsole
  creationTimestamp: "2020-09-30T14:10:12Z"
  labels:
    app: workshop
    app.kubernetes.io/component: parksmap
    app.kubernetes.io/instance: parksmap
    app.kubernetes.io/part-of: workshop
    component: parksmap
    role: frontend
  name: parksmap
  namespace: user1
  resourceVersion: "1062269"
  selfLink: /api/v1/namespaces/user1/services/parksmap
  uid: e1ff69c8-cb2f-11e9-82a1-0267eec7e1a0
spec:
  clusterIP: 172.30.22.209
  ports:
  - name: 8080-tcp
    port: 8080
    protocol: TCP
    targetPort: 8080
  selector:
    app: parksmap
    deploymentconfig: parksmap
  sessionAffinity: None
  type: ClusterIP
status:
  loadBalancer: {}
```

Take note of the `selector` stanza. Remember it.

Alternatively, you can use the web console to view information about the **Service** by clicking on it from the previous screen.


Services > Service Details
parksmap
Actions
Details
YAML
Pods
Service Details
Name
parksmap
Namespace
NS user1
Labels
Edit
app=workshop
app.kubernetes.io/component=parksmap
app.kubernetes.io/instance=parksmap
app.kubernetes.io/part-of=workshop
app.openshift.io/runtime-version=1.3.0
component=parksmap
role=frontend
Pod Selector
app=parksmap, deploymentconfig=parksmap
Annotations
1 Annotation
Session Affinity
None
Created At
9 minutes ago
Owner
No owner
Service Routing
Service Address
Type
Location
Cluster IP
172.30.190.204
Accessible within the cluster only
Service Port Mapping
Name
Port
Protocol
Pod Port or Name
8080-tcp
S 8080
TCP
P 8080

It is also of interest to view the YAML of the **Pod** to understand how OpenShift wires components together. For example, run the following command to get the name of your `parksmap` **Pod**:

```
oc get pods
```

You should see output similar to the following:

```
NAME                        READY   STATUS    RESTARTS   AGE
parksmap-65c4f8b676-k5gkk   1/1     Running   0          5m12s
```

Now you can view the detailed data for your **Pod** with the following command:

```
oc get pod parksmap-65c4f8b676-k5gkk -o yaml
```

Under the `metadata` section you should see the following:

```
  labels:
    app: parksmap
    deployment: parksmap
```

- The **Service** has `selector` stanza that refers to `deployment=parksmap`.
- The **Pod** has multiple **Labels**:
  - `app=parksmap`
  - `deployment=parksmap`

Labels are just key/value pairs. Any **Pod** in this **Project** that has a **Label** that matches the **Selector** will be associated with the **Service**. To see this in action, issue the following command:

```
oc describe service parksmap
```

You should see something like the following output:

```
Name:              parksmap
Namespace:         user1
Labels:            app=workshop
                   app.kubernetes.io/component=parksmap
                   app.kubernetes.io/instance=parksmap
                   app.kubernetes.io/part-of=workshop
                   component=parksmap
                   role=frontend
Annotations:       openshift.io/generated-by: OpenShiftWebConsole
Selector:          app=parksmap,deploymentconfig=parksmap
Type:              ClusterIP
IP:                172.30.22.209
Port:              8080-tcp  8080/TCP
TargetPort:        8080/TCP
Endpoints:         10.128.2.90:8080
Session Affinity:  None
Events:            <none>
```

You may be wondering why only one endpoint is listed. That is because there is only one **Pod** currently running. In the next lab, we will learn how to scale an application, at which point you will be able to see multiple endpoints associated with the **Service**.

Continue