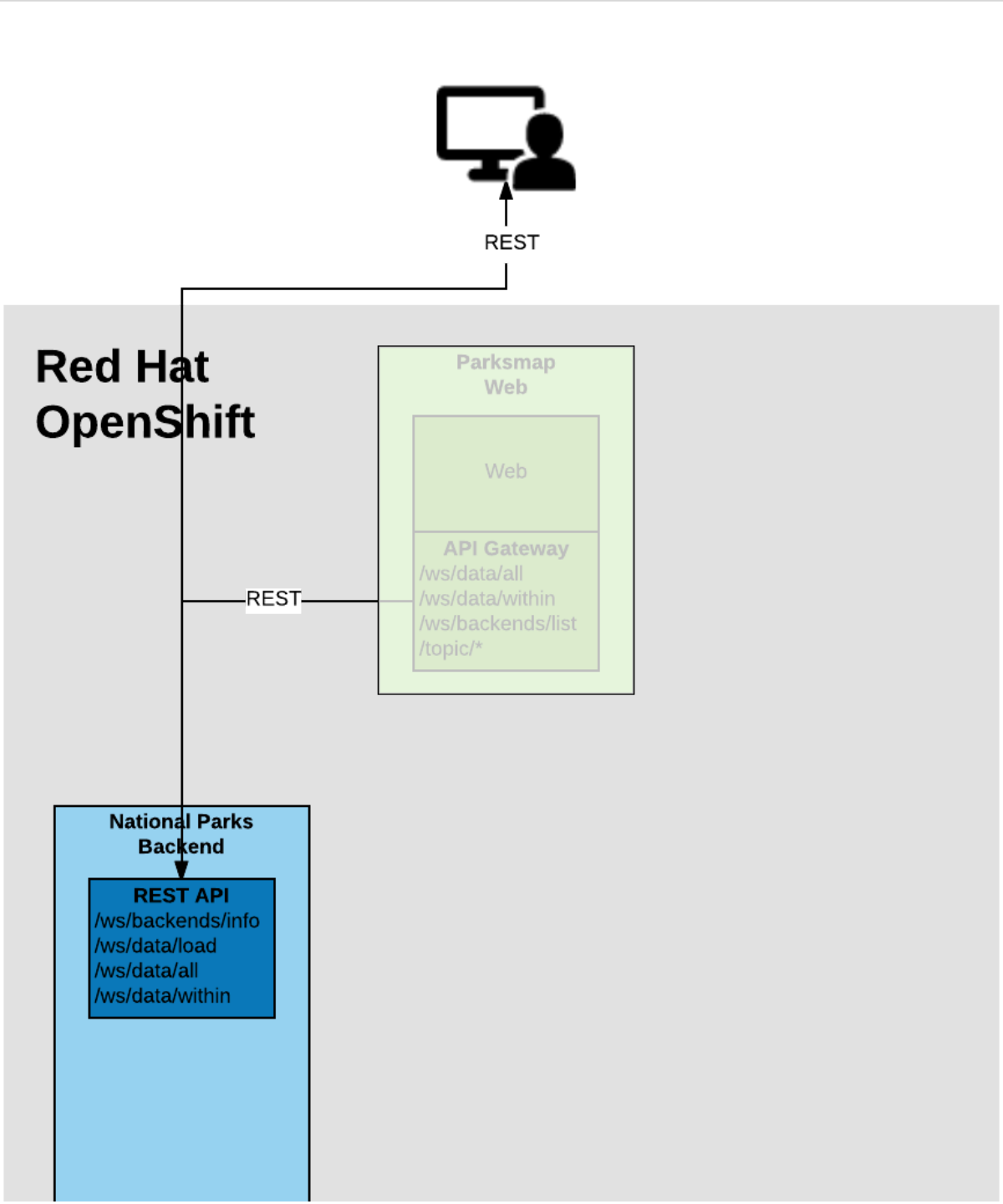
Deploying Python Code

In this lab, we're going to deploy a backend service, developed in Python that will expose 2 main REST endpoints to the visualizer application (`parksmap` web component that was deployed in the previous labs). The application will query for national parks information (including its coordinates) that is stored in a MongoDB database. This application will also provide an external access point, so that the API provided can be directly used by the end user.


REST
Red Hat
OpenShift
Parksmap
Web
Web
API Gateway
/ws/data/all
/ws/data/within
/ws/backends/list
/topic/*
REST
National Parks
Backend
REST API
/ws/backends/info
/ws/data/load
/ws/data/all
/ws/data/within

Background: Source-to-Image (S2I)

In a previous lab, we learned how to deploy a pre-existing container image. Now we will expand on that by learning how OpenShift builds images using source code from an existing repository. This is accomplished using the Source-to-Image project.

Source-to-Image (S2I) is a open source project sponsored by Red Hat that has the following goal:

```
Source-to-image (S2I) is a tool for building reproducible images. S2I
produces ready-to-run images by injecting source code into a container image and
assembling a new container image which incorporates the builder image and built
source. The result is then ready to use with docker run. S2I supports
incremental builds which re-use previously downloaded dependencies, previously
built artifacts, etc.
```

OpenShift is S2I-enabled and can use S2I as one of its build mechanisms (in addition to building container images from Dockerfiles, and "custom" builds).

OpenShift runs the S2I process inside a special **Pod**, called a Build Pod, and thus builds are subject to quotas, limits, resource scheduling, and other aspects of OpenShift.

A full discussion of S2I is beyond the scope of this class, but you can find more information about it either in the OpenShift S2I documentation or on GitHub. The only key concept you need to remember about S2I is that it's magic.

Exercise: Creating a Python application

The backend service that we will be deploying as part of this exercise is called `nationalparks`. This is a python application that performs 2D geo-spatial queries against a MongoDB database to locate and return map coordinates of all National Parks in the world. That was just a fancy way of saying that we are going to deploy a webservice that returns a JSON list of places.

Add to Project

Because the `nationalparks` component is a backend to serve data that our existing frontend (parksmap) will consume, we are going to build it inside the existing project that we have been working with. To illustrate how you can interact with OpenShift via the CLI or the Web Console, we will deploy the nationalparks component using the web console.

Using application code on embedded Git server

OpenShift can work with any accessible Git repository. This could be GitHub, GitLab, or any other server that speaks Git. You can even register webhooks in your Git server to initiate OpenShift builds triggered by any update to the application code!

The repository that we are going to use is already cloned in the internal Gogs repository and located at the following URL:

Gogs Repository

Your Gogs credentials are:

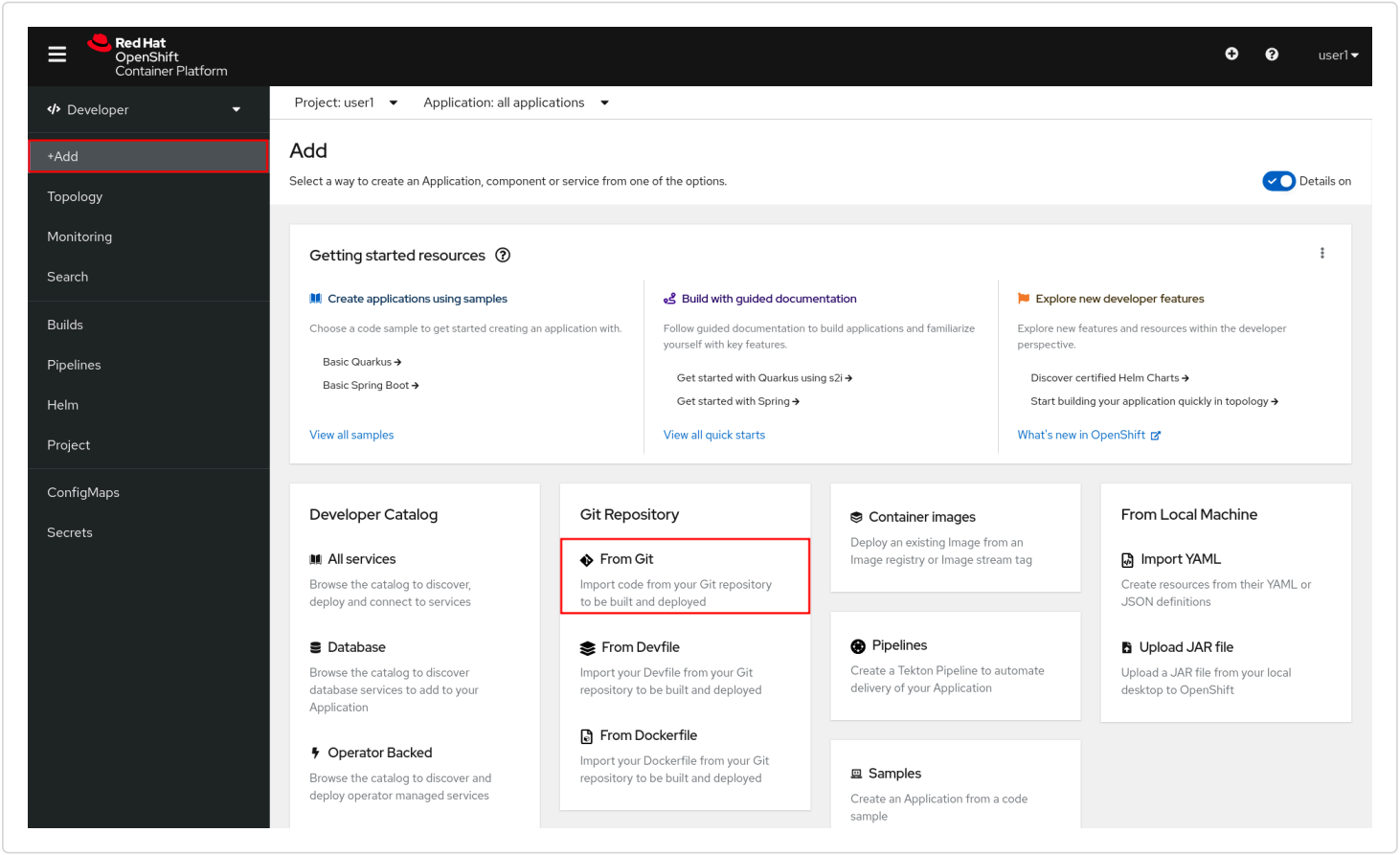
```
username: user4
password: gogs
```

Later in the lab, we want you to make a code change and then rebuild your application. This is a fairly simple Python application.

Build the Code on OpenShift

Similar to how we used **+Add** before with an existing image, we can do the same for specifying a source code repository. Since for this lab you have your own git repository, let's use it with a simple Python S2I image.

In the Developer Perspective, click **+Add** in the left navigation and then choose "From Git"



The **Import from Git** workflow will guide you through the process of deploying your app based on a few selections.

Enter the following for Git Repo URL:

```
http://gogs-labs.apps.rosa-7s42b.rfax.p1.openshiftapps.com/user4/nationalparks-py.git
```

In **Git Type** select **Other**.

> if you copied the Gogs URL correctly (check spaces etc) and you still get a warning like 'URL is valid but cannot be reached' please ignore it at this time, since the UI may fail to ping the repo sometimes.

Select **Python** as your Builder Image.

Import from Git

Git

Git Repo URL *

http://gogs-lab.apps.cluster-jazz-0733.jazz-0733.example.opentlc.com/user1/nationalparks-py.git

Git Type *

Other

Builder

Builder Image

Perl | PHP | Nginx | Httpd | .NET Core | Go | Ruby

Python | Java | Node.js

Builder Image Version *

IST 3.8-ubi7

Python 3.8 (UBI 7)

BUILDER PYTHON

Build and run Python 3.8 applications on UBI 7. For more information about using this builder image, including OpenShift considerations, see https://github.com/sclorg/s2i-python-container/blob/master/3.8/README.md.

Sample repository: https://github.com/sclorg/django-ex.git

> All of these builder images shown are made available via **Templates** and **ImageStreams**, which will be discussed in a later lab.

Scroll down to the **General** section. Select:

- **Application Name** : workshop
- **Name** : nationalparks

In **Resources** section, select **Deployment**.

Inside **Pipeline** section, check **Add pipeline** box. This will create a Tekton Pipeline for us that we will use after in the Pipeline modules.

> Click "Show pipeline visualization" to preview the Pipeline inside Pipeline UI that we are going to use later on.

Expand the Labels section and add 3 labels:

The name of the Application group:

```
app=workshop
```

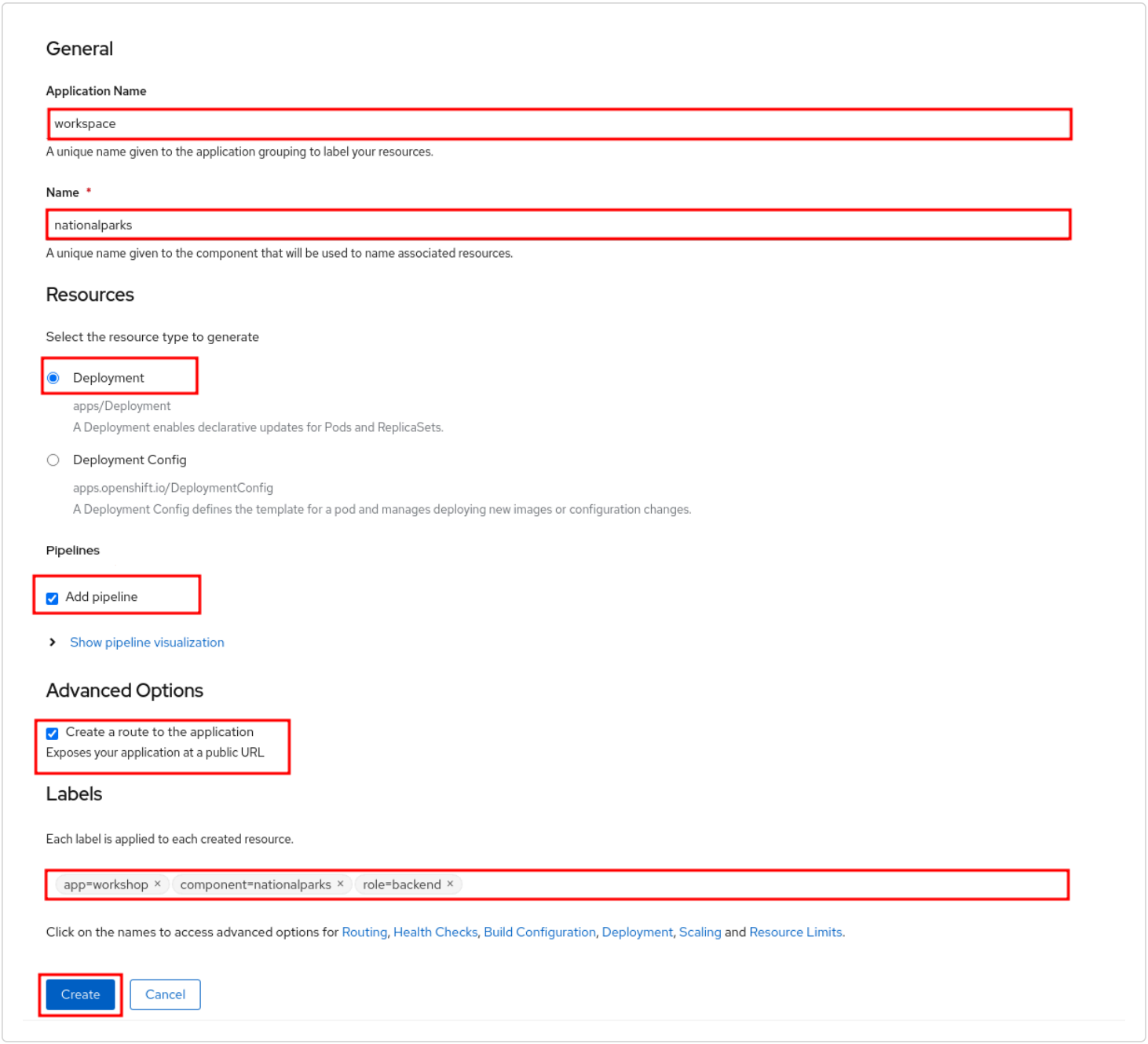
Next the name of this deployment.

```
component=nationalparks
```

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

```
role=backend
```

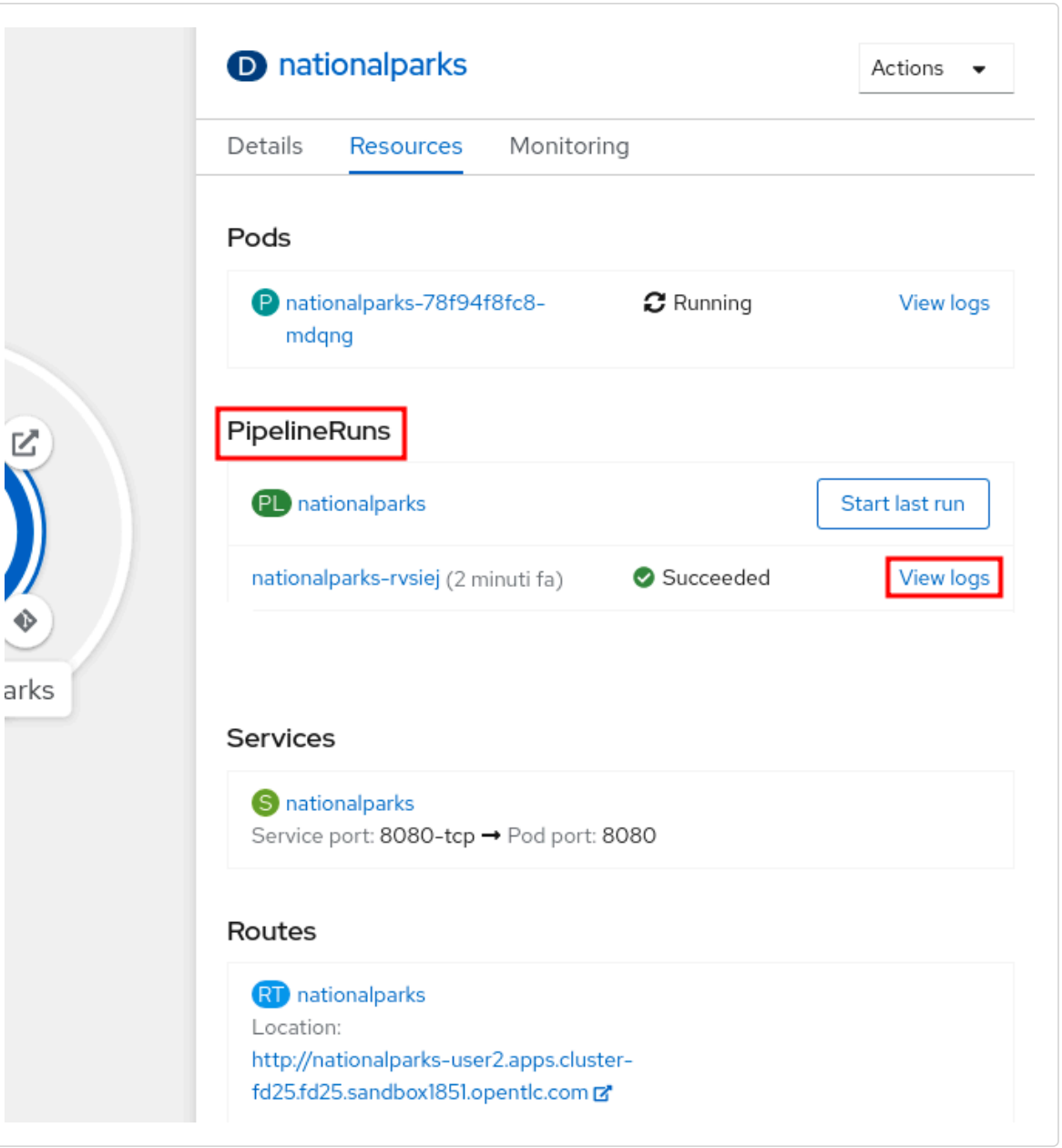
Now click the **Create** button.



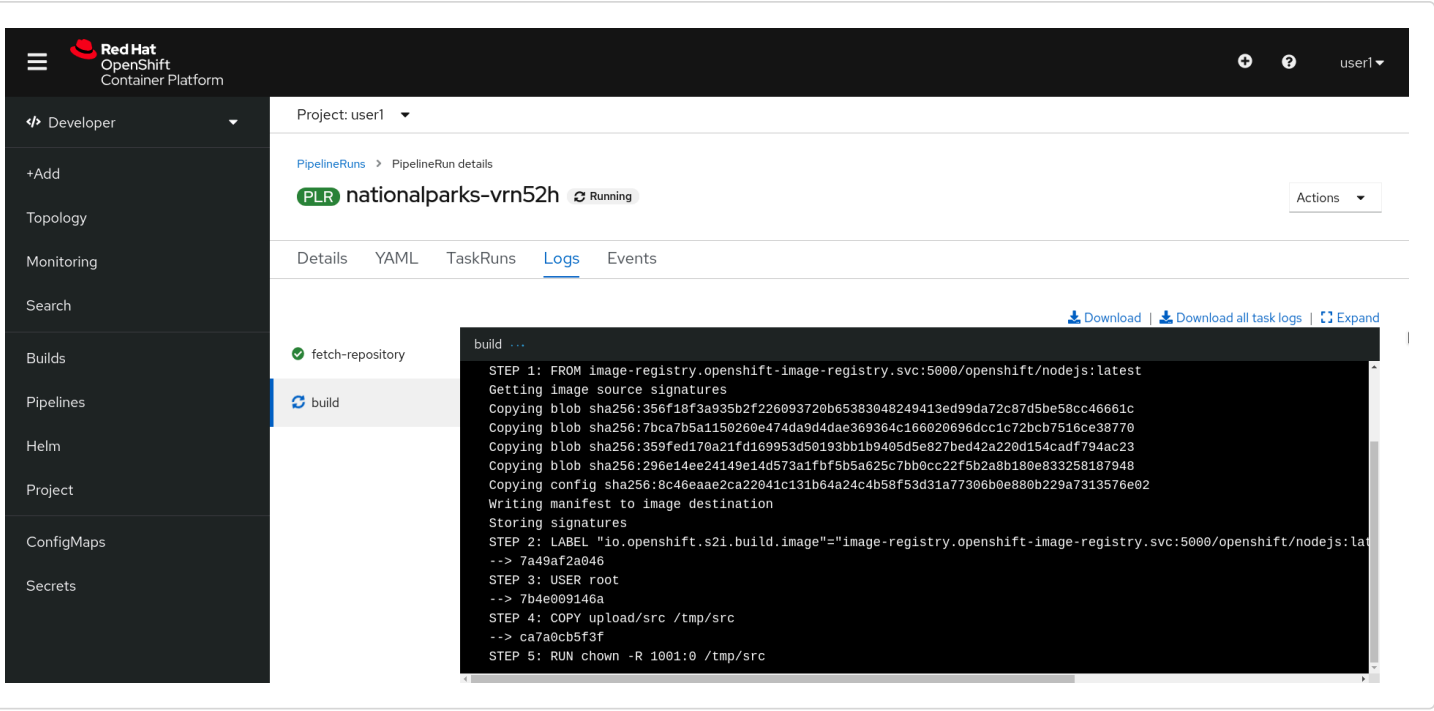
At this point, OpenShift will build the app and create a container through the Pipeline we just added.

> We will discuss more in details about OpenShift Pipelines in the **Continuous Integration and Pipelines** module.

To see the build logs, in the Topology view, click the `nationalparks` entry. Inside **Resources** tab, go to **PipelineRuns** section and click to **View Logs** link next to running pipeline.



Your newly created pipeline is running to build the backend from he source code and push the resulting container image to the OpenShift Registry.



The initial build will take a few minutes to download all of the dependencies needed for the application.

After the build has completed and successfully:

- The S2I process will push the resulting container image to the internal OpenShift registry
- The **Deployment** (D) will detect that the image has changed, and this will cause a new deployment to happen.
- A **ReplicaSet** (RS) will be spawned for this new deployment.
- The RC will detect no **Pods** are running and will cause one to be deployed, as our default replica count is just 1.

In the end, when issuing the `oc get pods` command, you will see that the each step of the pipeline has been executed inside a Pod (Completed) and that an application **Pod** is in a ready and running state:

```
NAME                    READY     STATUS      RESTARTS   AGE
nationalparks-757df44bd4-hnrxc                         1/1     Running     0          2m23s
```

```
nationalparks-vrn52h-build-m5nmf-pod-r4p2z              0/4     Completed   0          4m26s
nationalparks-vrn52h-deploy-pv6nx-pod-vwx62             0/1     Completed   0          2m22s
nationalparks-vrn52h-fetch-repository-4wjkm-pod-4zxm6   0/1     Completed   0          5m27s
```

If you look again at the web console, you will notice that, when you create the application this way, OpenShift also creates a **Route** for you. You can see the URL in the web console, or via the command line:

```
oc get routes
```

Where you should see something like the following:

```
NAME            HOST/PORT                                                   PATH      SERVICES
PORT       TERMINATION       WILDCARD
nationalparks   nationalparks-user4.apps.rosa-7s42b.rfax.p1.openshiftapps.com            nationalparks
8080-tcp
parksmap        parksmap-user4.apps.rosa-7s42b.rfax.p1.openshiftapps.com                 parksmap
8080-tcp        edge        none
```

In the above example, the URL is:

```
http://nationalparks-user4.apps.rosa-7s42b.rfax.p1.openshiftapps.com
```

Since this is a backend application, it doesn't actually have a web interface. However, it can still be used with a browser. All backends that work with the parksmap frontend are required to implement a `/ws/info/` endpoint. To test, visit this URL in your browser:

National Parks Info Page

> The trailing slash is **required**.

You will see a simple JSON string:

```
{"id":"nationalparks-py","displayName":"National Parks (PY)","center":
{"latitude":"47.039304","longitude":"14.505178"},"zoom":4}
```

Earlier we said:

```
This is a Python application that performs 2D geo-spatial queries
against a MongoDB database
```

But we don't have a database. Yet.

Continue