1. Launch the OpenShift cluster

You are using **Code-Ready Containers**, a single-server OpenShift cluster that Red Hat makes available for free. It's the successor of minishift.

To start the cluster, run `crc start`. After about five minutes, the cluster is up and running.

Copy the last few output lines into a file:

```
Started the OpenShift cluster.

The server is accessible via web console at:
  https://console-openshift-console.apps-crc.testing

Log in as administrator:
  Username: kubeadmin
  Password: vNd8o-Ryg8J-hYLSP-AoGpU

Log in as user:
  Username: developer
  Password: developer

Use the 'oc' command line interface:
  $ eval $(crc oc-env)
  $ oc login -u developer https://api.crc.testing:6443
```

The kubeadmin password is particularly important.

2. Deploy an app from an image

First contact with OpenShift

```
$ oc login -u developer
error: couldn't get https://192.168.49.2:8443/.well-known/oauth-authorization-server: unexpected
response status 403
```

Before you can use the kubectl or oc command, you need to set the kubeconfig context to CRC.

```
$ oc config get-contexts
CURRENT   NAME                                  CLUSTER                AUTHINFO                          NAMESPACE
          /api-crc-testing:6443/developer       api-crc-testing:6443   developer/api-crc-testing:6443
          crc-admin                             api-crc-testing:6443   kubeadmin                         default
          crc-developer                         api-crc-testing:6443   developer                         default
          default/api-...ng:6443/kubeadmin      api-crc-testing:6443   kubeadmin/api-crc-testing:6443    default
          devproject/api-...ng:6443/developer   api-crc-testing:6443   developer/api-crc-testing:6443    devproject
          devproject/api-...ng:6443/kubeadmin   api-crc-testing:6443   kubeadmin/api-crc-testing:6443    devproject
*         minikube                              minikube               minikube                          default
```

Currently, the kubeconfig file (`$HOME/.kube/config`) uses minikube as context, since this is what you have been using so far. Change it to one of the crc contexts. It doesn't matter which.

```
$ oc config use-context crc-admin
Switched to context "crc-admin".
$ oc login -u developer
Password: developer
Login successful.

You don't have any projects. You can try to create a new project, by running

    oc new-project <projectname>

$ oc whoami
```

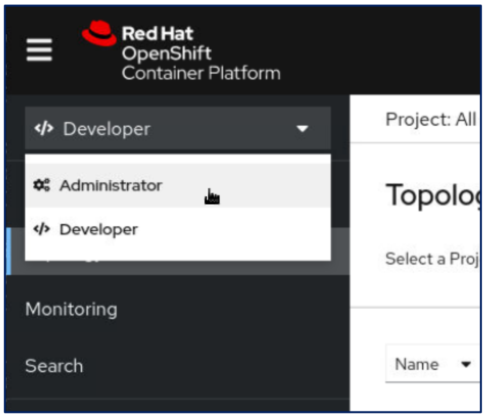
```
developer
```

Create a project:

```
$ oc new-project devproject
```

Projects are OpenShift's method to group applications. They have access control and quotas for developers or teams. Projects are based on Kubernetes namespaces.

Access the OpenShift web console at https://console-openshift-console.apps-crc.testing. Log on as developer, password developer.



The Topology page is shown, with the project you just created.

If you forgot to create the project, do it from the console: Open the **Project** menu, then click on **create a Project**. Only the project's name has to be provided.

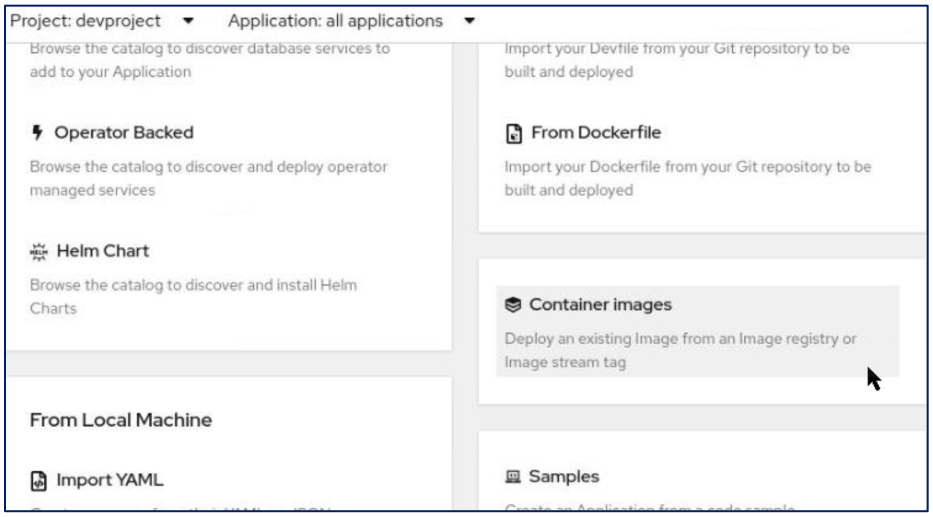
Your perspective is *Developer*. Switch to the *Administrator* perspective (screenshot on the left) and note how the menu changes. Go back to the *Developer* perspective.

The Administrator Perspective consists of: Operators**,** Workloads**,** Networking**,** Storage**,** Builds**,** User Mgmt and Administration menus.

The Developer Perspective includes: Topology, Monitoring, Search, Builds, Helm, Project, ConfigMaps, Secrets.

Deploy an app

In the developer perspective, enter *+Add*. This section allows you to create applications from various sources and using various methods. View the available options. Find the **Container images** tile and click it. You will now deploy an app from an image.



Project: devproject
Application: all applications

Browse the catalog to discover database services to add to your Application

Import your Devfile from your Git repository to be built and deployed

Operator Backed

Browse the catalog to discover and deploy operator managed services

From Dockerfile

Import your Dockerfile from your Git repository to be built and deployed

Helm Chart

Browse the catalog to discover and install Helm Charts

Container images

Deploy an existing Image from an Image registry or Image stream tag

From Local Machine

Import YAML

Samples

Under *Image name from external registry*, enter image name `docker.io/openshiftroadshow/parksmap-katacoda:1.2.0`. The image will be validated.

If you like, you can select a different icon to identify your application.

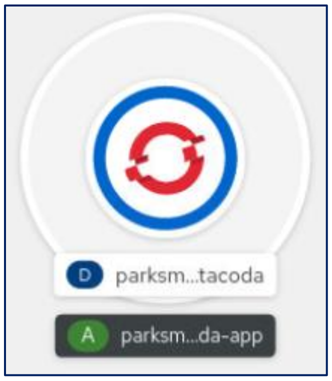
Scroll down. Two names were set automatically; the app is named `parksmap-katacod-app`, and the name `parksmap-katacoda` will be used as a basis for naming various application components.

Under **Resources**, you have the choice of **Deployment** or **DeploymentConfig**. Select **Deployment** for now.

Under **Advanced Options**, the **route** box is checked. Since you will create the route later, u**ncheck it.**

There are advanced options for **Health Checks, Deployment, Scaling, Resource Limits and Labels**. Open them all. Most of them were covered to some extent during the Kubernetes course module; try to remember their meaning. Feel free to set some options.

Click **Create**. The browser goes back to *Topology*, where this picture is shown:



The outer white ring is the app, the inner blue ring is a Deployment.

Switch to a terminal window and list your resources

```
$ oc get all
NAME                                       READY   STATUS    RESTARTS   AGE
pod/parksmap-katacoda-6849877f77-xs6s9    1/1     Running   0          118s

NAME                           TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)    AGE
service/parksmap-katacoda   ClusterIP   10.217.5.149   <none>        8080/TCP   118s

NAME                                    READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/parksmap-katacoda   1/1     1            1           118s

NAME                                             DESIRED   CURRENT   READY   AGE
replicaset.apps/parksmap-katacoda-6849877f77   1         1         1       118s

NAME                                                  IMAGE REPOSITORY
TAGS    UPDATED
imagestream.image.openshift.io/parksmap-katacoda   default-route-openshift-image-registry.apps-
crc.testing/devproject/parksmap-katacoda   1.2.0   About a minute ago
```

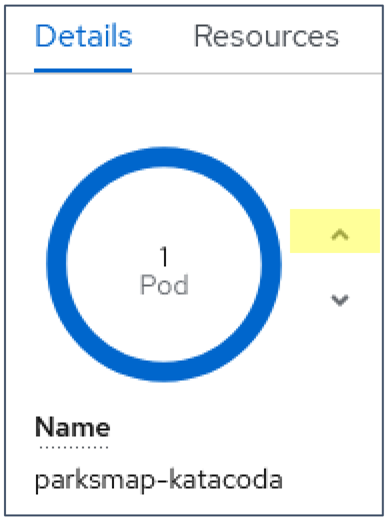
There is a deployment, a replicaset, a pod, a service and the OpenShift-specific resource imagestream.

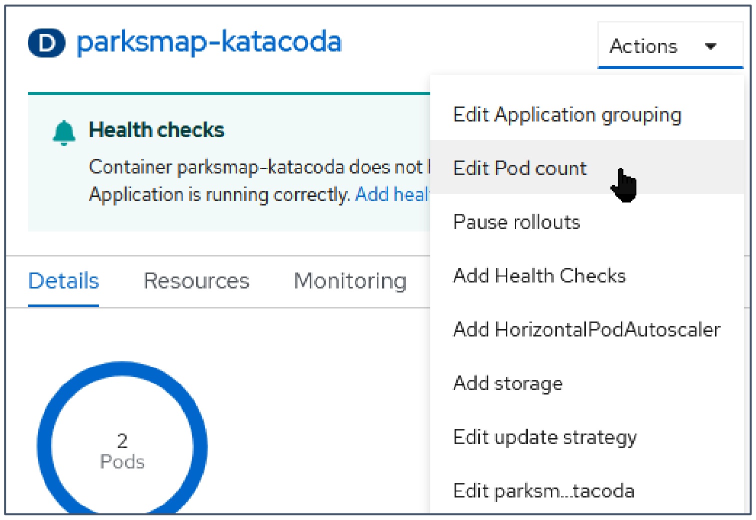
View details in the browser

Click on the blue ring to open the Deployment menu. The Resources tab lists the pod and the service.

Open the *Details* tab. This seems to be a graphical representation of `oc describe deployment`. Click on the dash-underlined headers such as **Update strategy** to see a short explanation of their meaning.

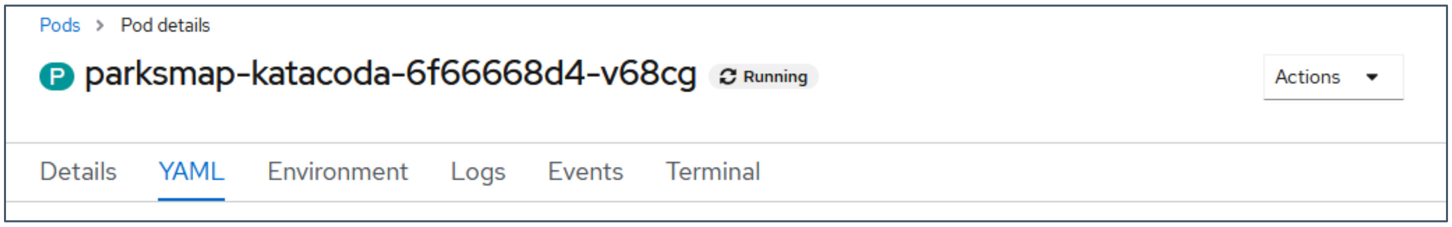
Scale the application up by clicking on the up arrow, or Actions → Edit Pod Count





Go back to the **resources** tab. There are two pods now. Click on a pod.

Lots of pod details are shown, including metrics. The **YAML** tab displays the pod's manifest and allows you to download it.



Select action **delete pod**. It will be deleted, but since the replica count is 2, a new pod is created immediately.

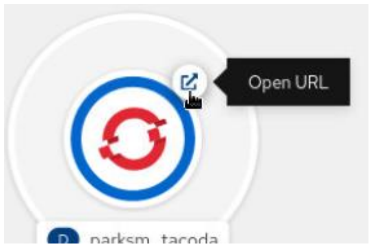
Scale back down to 1. This removes one of the two pods.

Create a route

Switch to the administrator perspective. Select **myproject**. In the left menu, select **Networking→Routes**. Create a route and set a name, select the parksmap service and the 8080→8080 port mapping; leave **secure route** unchecked.

After clicking Create, you are in the route's details page. The route's URL should be similar to http://parkmap-route-devproject.apps-crc.testing.

Go back to **developer** perspective and select Topology. The route is indicated by this icon:



Click on the route icon to open the app. Currently, it only shows a world map that can be moved and zoomed.

Building from source code

You will add a second deployment to your app which provides locations of national parks around the world. This time, you will use S2i to generate the app.

Ensure that you are in the developer perspective.

Click **+Add** in the menu. Click **Catalog→All Services**.

Notice the type of objects in the catalog:

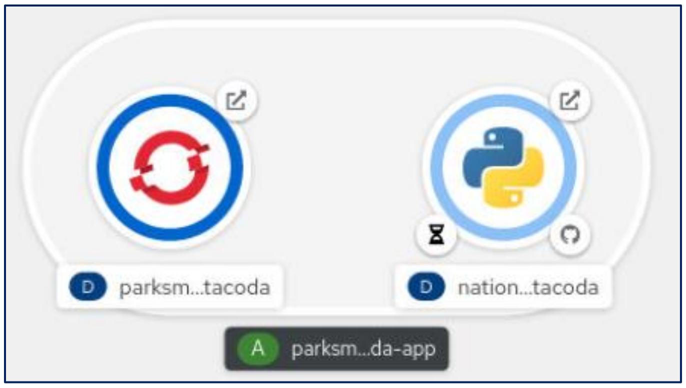
> **Builder Images;** S2i-enabled container images for building source code for with a particular language or framework.
>
> **Helm Charts**
>
> **Templates**; sample applications that you can tweak to build your own.
>
> **Devfiles**; YAML code that describes a development environment.
>
> **Not shown here: Operator backed**; infrastructure services managed by Kubernetes controllers.

Select **Languages** on the left menu, then **Python**. There are three options; click on the builder image **Python**, read the description, then **Create App**.

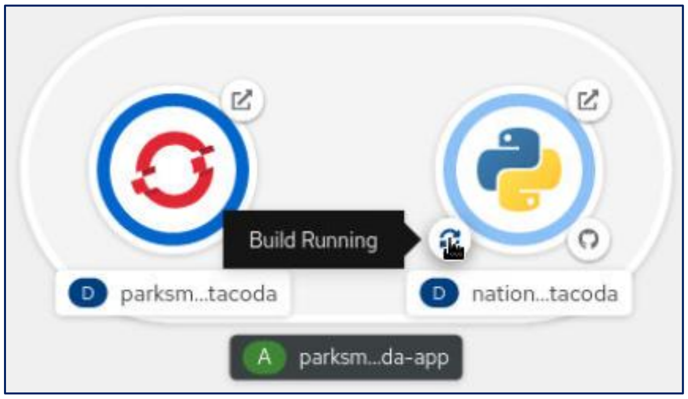
Enter **Git Repo URL** https://github.com/openshift-roadshow/nationalparks-katacoda. OpenShift performs a quick validation: Is this a valid repo, is it accessible.

Leave all other options unchanged and click **Create**. This sends you back to the **Topology** page.

The parksmap app now has two deployments, one being the Python code.



The light blue ring and the hourglass icon indicate that the application is waiting to build. The Github icon links to the source code's repo. The hourglass will eventually change to an icon indicating that a build is in progress.



Click on the Python ring and select **Resources** → **Builds**. The build might be running but will eventually complete. **View logs** shows the build messages. The last message should be *Push successful*.

Back to the **Topology** page: The Python deployment has a check mark now, meaning that the build is complete. The ring is a darker blue, also indicating that the application is ready. Click on the route icon of the left deployment to see locations of national parks.

Explore resources on the command line

Go to a terminal and list all resources.

```
$ oc get all
NAME                                          READY   STATUS      RESTARTS   AGE
pod/nationalparks-katacoda-1-build            0/1     Completed   0          7m54s
...

NAME                                               DESIRED   CURRENT   READY   AGE
replicaset.apps/nationalparks-katacoda-74ff6bdbbf  1         1         1       2m44s
replicaset.apps/nationalparks-katacoda-85898c9f8f  0         0         0       7m54s
replicaset.apps/parksmap-katacoda-6849877f77       2         2         2       40m

NAME                                                   TYPE     FROM   LATEST
buildconfig.build.openshift.io/nationalparks-katacoda  Source   Git    1

NAME                                               TYPE     FROM          STATUS     STARTED
DURATION
build.build.openshift.io/nationalparks-katacoda-1  Source   Git@43842c2   Complete   7 minutes ago
5m10s
...
NAME                                              HOST/PORT
PATH    SERVICES                  PORT       TERMINATION   WILDCARD
route.route.openshift.io/nationalparks-katacoda   nationalparks-katacoda-devproject.apps-crc.testing
nationalparks-katacoda    8080-tcp                  None
```

```
route.route.openshift.io/parkmap-route              parkmap-route-devproject.apps-crc.testing
parksmap-katacoda         8080-tcp                  None
```

Many more resources exist now.

One of the pods was used for building the app and is now complete. It is associated with the Build resource:

```
$ oc describe pod/nationalparks-katacoda-1-build|grep Controlled
Controlled By:  Build/nationalparks-katacoda-1
```

List the details of the build. A few points worth noting: The name and some details of the build config are included, the reason why the build was started, and there is a long list of events.

```
$ oc describe build nationalparks-katacoda-1 | more
```

Also display the details of the build config. They should match what you say in the build.

There are two versions of the same replicaset because the corresponding deployment was upgraded:

```
$ oc rollout history deploy nationalparks-katacoda
deployment.apps/nationalparks-katacoda
REVISION  CHANGE-CAUSE
1         <none>
2         <none>
```

To understand why there are two revisions, display their details. The image was updated from revision 1 to 2.

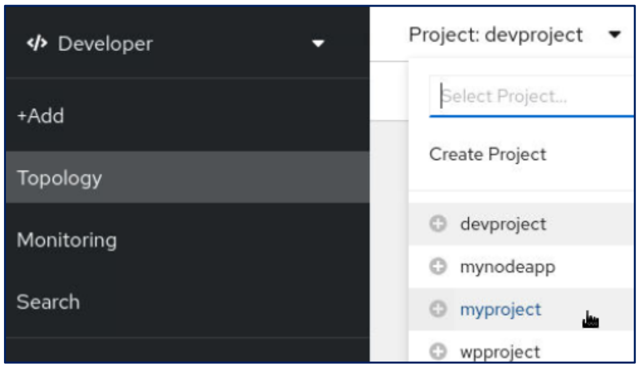
```
$ oc rollout history deploy nationalparks-katacoda --revision=1
$ oc rollout history deploy nationalparks-katacoda --revision=2
```

3. Deploy a second app from source

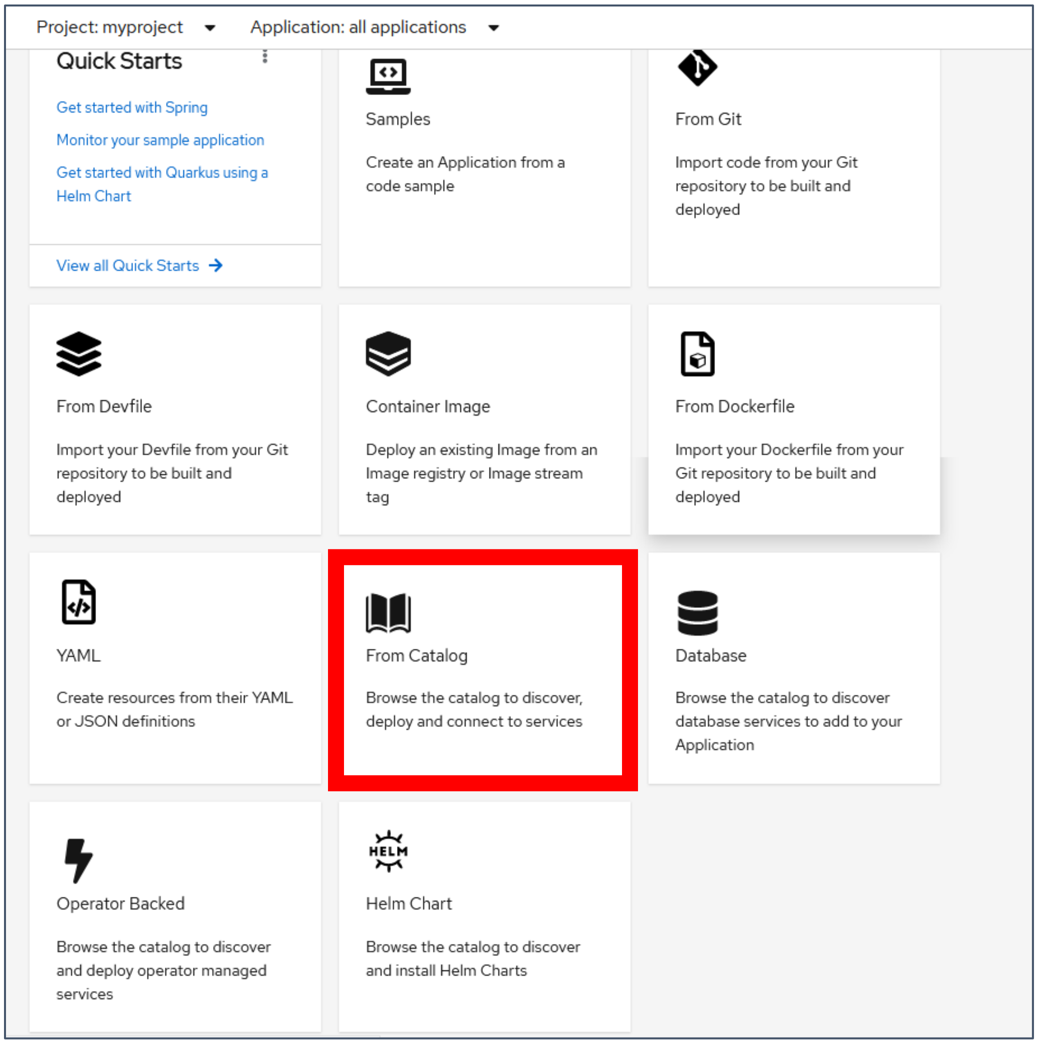
Create a second project from the command line or the webconsole.

```
$ oc new-project myproject
Now using project "myproject" on server "https://openshift:6443".

You can add applications to this project with the 'new-app' command. For example, try:

    oc new-app rails-postgresql-example

to build a new example application in Ruby. Or use kubectl to deploy a simple Kubernetes application:

    kubectl create deployment hello-node --image=k8s.gcr.io/serve_hostname
```

In the console, ensure you are developer, go to the **Topology** page and select the new project from the drop-down menu.



Click on **myproject**. The Topology page is empty. Go to **+Add** and select **From Catalog**.



Project: myproject
Application: all applications

Quick Starts
Get started with Spring
Monitor your sample application
Get started with Quarkus using a Helm Chart
View all Quick Starts

Samples
Create an Application from a code sample

From Git
Import code from your Git repository to be built and deployed

From Devfile
Import your Devfile from your Git repository to be built and deployed

Container Image
Deploy an existing Image from an Image registry or Image stream tag

From Dockerfile
Import your Dockerfile from your Git repository to be built and deployed

YAML
Create resources from their YAML or JSON definitions

From Catalog
Browse the catalog to discover, deploy and connect to services

Database
Browse the catalog to discover database services to add to your Application

Operator Backed
Browse the catalog to discover and deploy operator managed services

Helm Chart
Browse the catalog to discover and install Helm Charts

Select **Language** → **Python**. Click on the **Python** tile.

The Python page describes the Python builder image. **Create Application**.

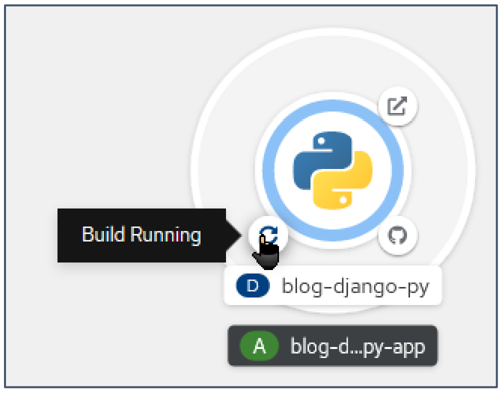
Git Repo URL: Enter https://github.com/openshift-katacoda/blog-django-py.

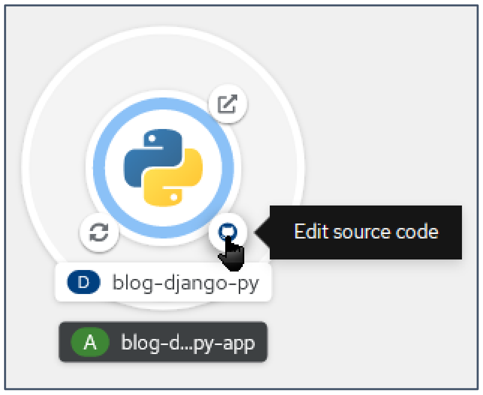
General: see app name and name.

Leave **Route** checkbox checked.

Other **Advanced Settings**: Routing, Health Checks, Build config, Deployment, Scaling, Resource limits.

Leave everything unchanged and **Create**.





The ring color goes from white (build pending) to light blue (build running) to a darker blue (app running)

Clicking on the **Build Running** icon shows the logs. **Edit Source Code** links to the Git repo.

Delete app

Switch to the command line.

Show all resources. Assuming this is your only currently active app, you can simply list all resources:

```
$ oc get all -o name
pod/blog-django-py-1-build
pod/blog-django-py-6d77767b8c-ctj8m
service/blog-django-py
deployment.apps/blog-django-py
replicaset.apps/blog-django-py-6d77767b8c
replicaset.apps/blog-django-py-7fc8bdfb
buildconfig.build.openshift.io/blog-django-py
build.build.openshift.io/blog-django-py-1
imagestream.image.openshift.io/blog-django-py
route.route.openshift.io/blog-django-py
```

Notice that the app comprises pods, a service, a deployment, replicasets, a buildconfig, a build, an imagestream and the route.

In case several apps are deployed, use a label to filter the right app.

```
$ oc get all -o name -l app=
```

Display details of the route:

```
$ oc describe route blog-django-py
Name:                   blog-django-py
Namespace:              myproject
Created:                8 minutes ago
Labels:                 app=blog-django-py
                        app.kubernetes.io/component=blog-django-py
                        app.kubernetes.io/instance=blog-django-py
                        app.kubernetes.io/name=python
                        app.kubernetes.io/part-of=blog-django-py-app
                        app.openshift.io/runtime=python
                        app.openshift.io/runtime-version=3.8-ubi7
Annotations:            openshift.io/host.generated=true
Requested Host:         blog-django-py-myproject.2886795282-80-hazel04.environments.katacoda.com
                           exposed on router default (host apps-crc.testing) 8 minutes ago
Path:                   <none>
TLS Termination:        <none>
Insecure Policy:        <none>
Endpoint Port:          8080-tcp

Service:        blog-django-py
Weight:         100 (100%)
Endpoints:      10.217.1.13:8080
```

The label `app=blog-django-py` was set by the web console. List all resources with this label.

```
$ oc get all --selector app=blog-django-py -o name
$ oc delete all -o name --selector app=blog-django-py
pod/blog-django-py-7fc8bdfb-6jjmw
service/blog-django-py
deployment.apps/blog-django-py
replicaset.apps/blog-django-py-7fc8bdfb
buildconfig.build.openshift.io/blog-django-py
build.build.openshift.io/blog-django-py-1
imagestream.image.openshift.io/blog-django-py
route.route.openshift.io/blog-django-py
```

The label in this example is not universally correct. Always use `oc describe` to verify what labels have been applied. Before deleting, check with `oc get all --selector`.

Deploying using the command line

Ensure you are in devproject.

```
$ oc projects
$ oc project
$ oc project devproject
```

The new-app command below uses IMAGE~CODE syntax. It uses builder image python:latest and clones the source code from Github.

```
$ oc new-app python:latest~https://github.com/openshift-katacoda/blog-django-py
--> Found image f742fc9 (5 months old) in image stream "openshift/python" under tag "latest" for
"python:latest"
...
--> Creating resources ...
...
--> Success
    Build scheduled, use 'oc logs -f buildconfig/blog-django-py' 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/blog-django-py'
    Run 'oc status' to view your app.
```

The application is not exposed, which means there is no route. In contrast to the web console build process, the new-app command does not create a route by default..

```
$ oc logs bc/blog-django-py --follow
```

bc is the abbreviation for **buildconfig**. This shows the s2i output including buildah output. It will take several minutes to complete. You can interrupt and continue with the next commands. The following command shows the status of services, deployment configs, build configurations, and deployments in the current project. It looks like the Django blog is still building. The national parks app consists of two deployments, which are in progress.

```
$ oc status
In project devproject on server https://api.crc.testing:6443

svc/blog-django-py - 10.217.5.48:8080
  deployment/blog-django-py deploys istag/blog-django-py:latest <-
    bc/blog-django-py source builds https://github.com/openshift-katacoda/blog-django-py on
openshift/python:latest
      build #1 running for 2 minutes - 9c4f338: Merge pull request #9 from mixpix3ls/master (ryan
jarvinen <ryan.jarvinen@gmail.com>)
    deployment #1 running for 2 minutes - 0/1 pods growing to 1
...
3 infos identified, use 'oc status --suggest' to see details.
$ oc get all
```

This is too much information. Use a label to filter for the Django app.

```
$ oc get all -l app=blog-django-py
NAME                     TYPE        CLUSTER-IP    EXTERNAL-IP   PORT(S)    AGE
service/blog-django-py   ClusterIP   10.217.5.48   <none>        8080/TCP   10m

NAME                             READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/blog-django-py   1/1     1            1           10m

NAME                                            TYPE     FROM   LATEST
buildconfig.build.openshift.io/blog-django-py   Source   Git    1

NAME                                        TYPE     FROM          STATUS     STARTED          DURATION
build.build.openshift.io/blog-django-py-1   Source   Git@9c4f338   Complete   10 minutes ago   4m39s

NAME                                            IMAGE REPOSITORY
TAGS     UPDATED
```

```
imagestream.image.openshift.io/blog-django-py   default-route-openshift-image-registry.apps-
crc.testing/devproject/blog-django-py   latest   6 minutes ago
```

Replicasets and pods have a different label.

```
$ oc get all -l deployment=blog-django-py
NAME                               READY   STATUS    RESTARTS   AGE
pod/blog-django-py-b87967fbd-trmp9   1/1     Running   0          6m32s

NAME                                          DESIRED   CURRENT   READY   AGE
replicaset.apps/blog-django-py-78b7478997   0         0         0       11m
replicaset.apps/blog-django-py-b87967fbd    1         1         1       6m32s
```

Use the `expose` command to create a route.

```
$ oc expose service blog-django-py
route.route.openshift.io/blog-django-py exposed
$ oc get route
NAME             HOST/PORT                                                                PATH
SERVICES         PORT       TERMINATION   WILDCARD
blog-django-py   blog-django-py-myproject.2886795286-80-elsy07.environments.katacoda.com          blog-
django-py   8080-tcp                 None
...
```

Check the Topology page. The Django deployment is there but has no app. This is so because it lacks a label. Add the required label, then check the topology again.

```
$ oc label deploy blog-django-py app.kubernetes.io/part-of=blog-django-app
```

Try the route icon.

Trigger a new build

Go back to the command line and manually trigger a build. This command requires a BuildConfig as argument.

```
$ oc get bc
NAME                       TYPE     FROM   LATEST
blog-django-py             Source   Git    1
nationalparks-katacoda     Source   Git    1
$ oc start-build blog-django-py
build.build.openshift.io/blog-django-py-2 started
```

Follow the build.

```
$ oc get build --watch
NAME                         TYPE     FROM          STATUS     STARTED          DURATION
nationalparks-katacoda-1     Source   Git@43842c2   Complete   2 hours ago      5m10s
blog-django-py-1             Source   Git@9c4f338   Complete   28 minutes ago   4m39s
blog-django-py-2             Source   Git@9c4f338   Running    15 seconds ago
```

The logs have more detail.

```
$ oc logs buildconfig/blog-django-py
...
$ oc describe buildconfig blog-django-py
...
URL:              https://github.com/openshift-katacoda/blog-django-py
...
Webhook GitHub:
        URL:
https://openshift:6443/apis/build.openshift.io/v1/namespaces/myproject/buildconfigs/blog-
django-py/webhooks/<secret>/github
Webhook Generic:
        URL:
https://openshift:6443/apis/build.openshift.io/v1/namespaces/myproject/buildconfigs/blog-
django-py/webhooks/<secret>/generic
        AllowEnv:       false
```

Webhooks are used to detect that a new build must be performed, e.g. automatically when updated source is pushed to repo. This is possible for repos that you own.

4. Build locally

Use local source code to build an image. Start by cloning the Django blog code.

```
$ git clone https://github.com/openshift-katacoda/blog-django-py
$ cd blog-django-py
```

Make a change to the code. The default color is red. You add an environment variable to the S2i configuration.

```
$ echo 'BLOG_BANNER_COLOR=blue' >> .s2i/environment
```

New build:

```
$ oc start-build blog-django-py --from-dir=. --wait
```

Note the `from-dir` option. This build uses the local directory instead of the git repo. S2i calls this a *binary build*, although this term might be a bit misleading for a Python program.

Also check progress on the web console under the Builds menu: **Builds** → **blog-django-py** → **Builds tab**, then select the running build and the **logs** tab.

Accessing the app after the build shows that the banner has turned from red to blue.

To revert to the git source, run `oc start-build blog-django-py`

Since you are impatient, cancel the ongoing build: `oc cancel-build blog-django-py-NUMBER`

List the deployment's update history with `oc rollout history deploy blog-django-py`. Each build causes a new rollout.

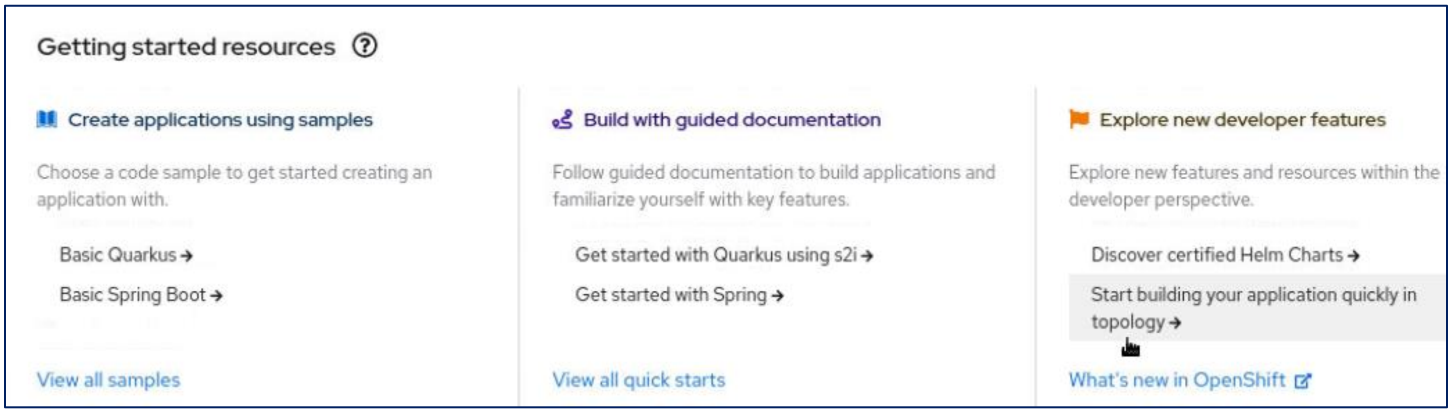
5. Optionally, build the Nodejs app

This is the same nodejs application that you used when building Docker images using Docker and S2i. You will clone the code from Github.

Log on to the web console as developer. Use the developer's perspective.

Go to the **+Add** section. Explore two ways of launching a Github-based build. Find the **Git Repository** tile and click on **From Git**. Explore the options, but don't fill any fields and go back to **+Add**.

Under **Getting started resources**, and **Explore new developer features**, select **Start building your application quickly in topology**.


Getting started resources

Create applications using samples

Choose a code sample to get started creating an application with.

Basic Quarkus →

Basic Spring Boot →

View all samples

Build with guided documentation

Follow guided documentation to build applications and familiarize yourself with key features.

Get started with Quarkus using s2i →

Get started with Spring →

View all quick starts

Explore new developer features

Explore new features and resources within the developer perspective.

Discover certified Helm Charts →

Start building your application quickly in topology →

What's new in OpenShift

An input field "*Add to Project*" is displayed. Enter the text **node**. You will be presented with a list of NodeJS images. Select **Nodejs Builder Images**, then click the blue **Create Application** button.

Select Builder Image Version **14-ubi8**. This is nodejs version 14 on RHEL 8.

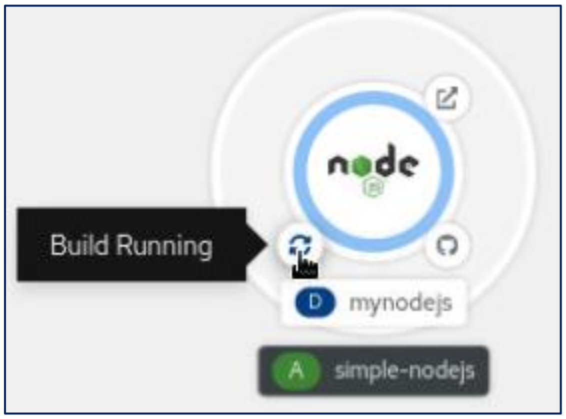
Enter the Git repo https://github.com/berndbausch/openshift-nodejs.

Enter **Application Name** *simple-nodejs* and **Name** *mynodejs*. Your BuildConfig will be named *mynodejs*, and other elements will contain *mynodejs* as part of their names.

Scroll down. Leave the **Create a route** box checked, but open the advanced route options and enter **awesome-site.apps-crc.testing** in the hostname field. To be reachable in this CRC environment, all routes must use *apps-crc.testing* as their domain component.

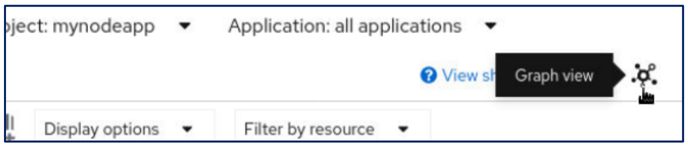
Click the blue **Create** box.

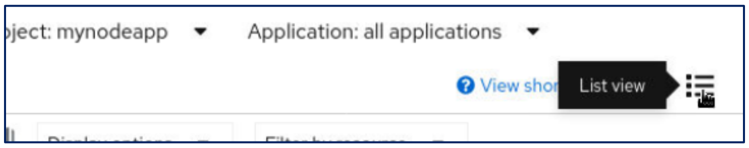
This sends you back to the Topology screen, where you see your app. Click on the build icon to see live log messages from the build.



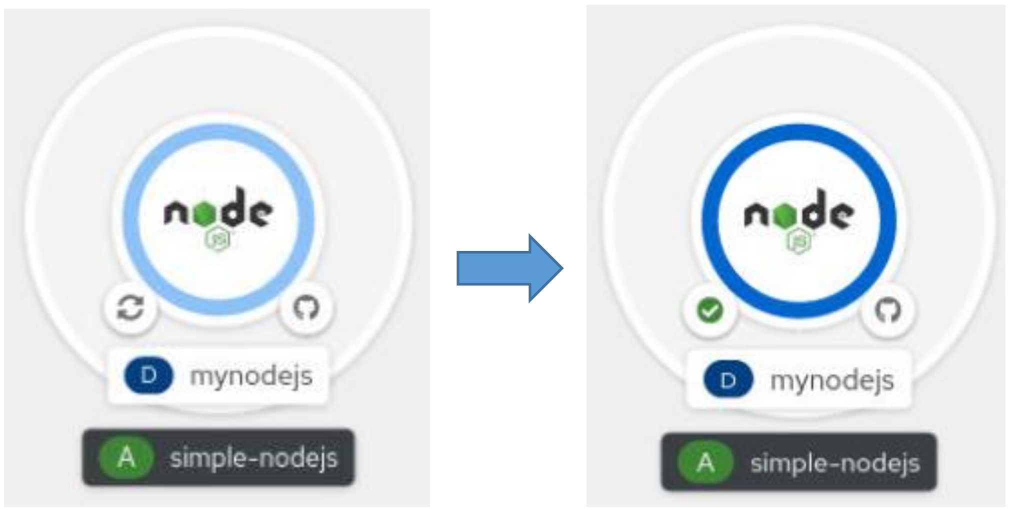
Eventually, the build will complete. Click on the route icon to see the app in action.

Switch between graphical and list mode by clicking the **Graph View** and **List View** buttons:





Go to the Graph View. Initially, your Deployment is marked with a light blue ring, which will eventually turn into a darker blue. When this is the case, the app is ready. If your mouse has a scroll wheel, you can use it to zoom in and out (there are also zoom buttons).



Light blue means that the app is currently building. This is also symbolized by the small icon in the lower left. This icon turns into a checkmark at the end of the build. Notice the Github icon; clicking it opens the code's Github repo.

Hover your mouse over the dark blue ring. How many pods are running?

Explore the interface by clicking on various elements of this graph, but don't spend too much time trying to understand what they mean.

When you are happy with your exploration, go to **Builds** in the left menu. Your BuildConfigs (labeled with a blue BC) are listed here. Click on the *mynodejs* BuildConfig.

The **Details** tab shows a description of the BuildConfig object, including the source location, the build image, the output image, run policy and triggers.Apart from the source location and the build image, these are default values.

Feel free to click the dash-underlined details for a short description of their meaning.

Go to the **YAML** tab to see the manifest. Download it; it will be saved to **buildconfig-mynodejs.yaml** in your Downloads directory.

Go to the **Builds** tab. It lists the builds that have been performed so far.

Back to the command line, check what project you are in.