

Lab 1 - Welcome

Welcome to OpenShift!

This lab provides a quick tour of the console to help you get familiar with the user interface along with some key terminology we will use in subsequent lab content. If you are already familiar with the basics of OpenShift simply ensure you can login and create the project.

Key Terms

We will be using the following terms throughout the workshop labs so here are some basic definitions you should be familiar with. You'll learn more terms along the way, but these are the basics to get you started.

- Container - Your software wrapped in a complete filesystem containing everything it needs to run
- Image - We are talking about docker images; read-only and used to create containers
- Pod - One or more docker containers that run together
- Service - Provides a common DNS name to access a pod (or replicated set of pods)
- Project - A project is a group of services that are related logically
- Deployment - an update to your application triggered by an image change or config change
- Build - The process of turning your source code into a runnable image
- BuildConfig - configuration data that determines how to manage your build



- Route - a labeled and DNS mapped network path to a service from outside OpenShift
- Master - The foreman of the OpenShift architecture, the master schedules operations, watches for problems, and orchestrates everything
- Node - Where the compute happens, your software is run on nodes

Log into the web console

OpenShift provides a web console that allows you to perform various tasks via a web browser. Additionally, you can utilize a command line tool to perform tasks. We will be focusing on the web console today.

1. Use your browser to navigate to the URI provided by your instructor.
2. Login with the user/password provided.

Create an empty project

First let's create a new project to do our workshop work in. We will use the student number you were given to ensure you don't clash with classmates, so in the steps below replace 'YOUR#' with your student number (if applicable).

1. Click the Create Project button
2. Populate Name with demo-YOUR#
3. Populate the Display Name and Description with whatever you'd like
4. Click Create



Summary

You should now be ready to get hands-on with our workshop labs.

Lab 2 - BYO Container Image

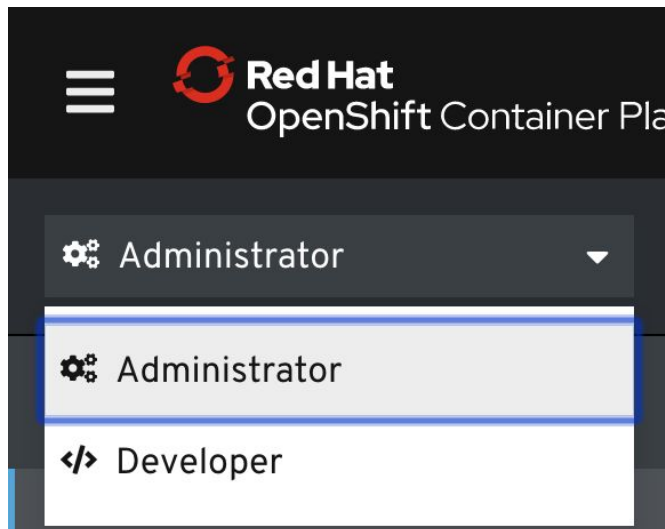
Bring your own docker

It's easy to get started with OpenShift whether you're using our app templates or bringing your existing assets. In this quick lab we will deploy an application using an existing container image. OpenShift will create an image stream for the image as well as deploy and manage containers based on that image. And we will dig into the details to show how all that works.

For this lab, we will be working in the Developer view rather than the Cluster Administrator view we are currently using.

Let's point OpenShift to an existing built container image

1. Switch to the *Developer View*



2. Click the *+Add* button in the left hand menu to Add a workload.

3. All of the options in this menu are available to help fast track your application to the Kubernetes platform. For now, select *Container Image*.
4. For the image name, enter “*sonatype/nexus:oss*” and click the magnifying glass on the right side of the text box
5. Observe the default values that are populated from the search.
6. Leave *Create route to the application* checked. This will make the application accessible outside the cluster.
7. Click *Create*

Deploy Image

Namespace *

Deploy an existing image from an image registry.

Image Name *

To deploy an image from a private repository, you must [create an image pull secret](#) with your image registry credentials.



sonatype/nexus:oss  Aug 9, 11:37 am, 203.7 MiB, 5 layers

- Image Stream **nexus:oss** will track this image.
- This image will be deployed in Deployment Config **nexus**.
- Port 8081/TCP will be load balanced by Service **nexus**.
Other containers can access this service through the hostname **nexus**.

This image declares volumes and will default to use non-persistent, host-local storage.
You can add persistent storage later to the deployment config.

Name *

Identifies the resources created for this image.

Is anything running?

1. Click on your new *nexus* application in the topology view
2. Click on the *Resources* tab and note the active pods.
3. Watch it move from *Container Creating* status to *Running*.

Does this pod do anything?

We created a route to this application when we added the application. Routes are load balanced domain names that are available outside the cluster for accessing your application. They reference service records which are load balanced domain names that are only available inside the cluster. Remember: routes are for external to internal communication, services are for internal to internal communication.

1. If you haven't already, click the *nexus* application in the topology view.
2. Select the URL listed under *Routes*. This will open a page in a new tab displaying a 404 Error.

HTTP ERROR: 404

Problem accessing /. Reason:

Not Found

Powered by Jetty://

3. This 404 error is actually good! We are getting it because Nexus hosts its content under a */nexus* path. Append “*/nexus*” to the end of your route's URL.
4. You should now see the Nexus Repository Manager.



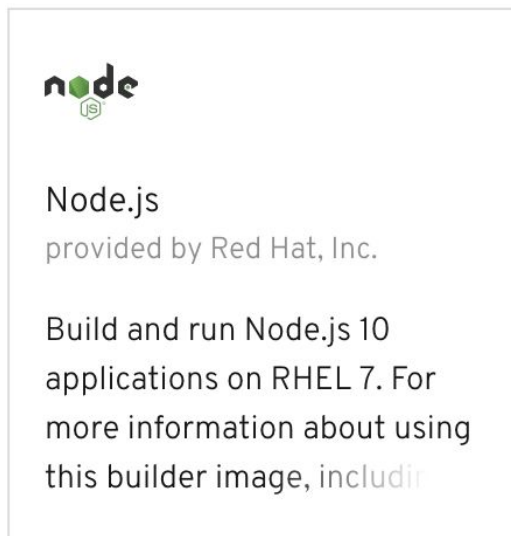
Lab 3 - Deploying an App with Source to Image (S2I)

Source to Image

One of the useful components of OpenShift is its source-to-image capability. S2I is a framework that makes it easy to turn your source code into runnable images. The main advantage of using S2I for building reproducible container images is the ease of use for developers. You'll see just how simple it can be in this lab.

Let's build a node.js web server using S2I

1. Click the *+Add* button in the left hand menu
2. Select *From Catalog*
3. Select the Node.js option. Using the search bar can help.



4. Select *Create Application*
5. Under the *Git Repo URL* field, click *try sample*.
6. In the *Application* drop down, select *Create Application*.



7. For the *Application Name*, type “nodejs-ex”
8. Click *Create*

Application Build

1. Notice the new application in the topology. Click on it to bring up the details pane. Go to the *Resources* tab.
2. Notice under *Builds* that you have a new build running. Click *View Logs* to monitor the build progress. Once the build is complete, you'll see a line that says, “Push successful”.
3. Go back to the topology view and select your Node.js application again. Again, go to the *Resources* tab. Now under *Pods*, you should see a running pod that is hosting your application.
4. In the *Routes* group, you should see one active route. Click the URL to see your application.

Welcome to your Node.js application on OpenShift

How to use this example application

For instructions on how to use this application with OpenShift, start by reading the [Developer Guide](#).

Deploying code changes

The source code for this application is available to be forked from the [OpenShift GitHub repository](#). You can configure a webhook in your repository to make OpenShift automatically start a build whenever you push your code:

1. From the Web Console homepage, navigate to your project
2. Click on Browse > Builds
3. Click the link with your BuildConfig name
4. Click the Configuration tab
5. Click the "Copy to clipboard" icon to the right of the "GitHub webhook URL" field
6. Navigate to your repository on GitHub and click on repository settings > webhooks > Add webhook
7. Paste your webhook URL provided by OpenShift in the "Payload URL" field
8. Change the "Content type" to 'application/json'
9. Leave the defaults for the remaining fields — that's it!

After you save your webhook, if you refresh your settings page you can see the status of

Managing your application

Documentation on how to manage your application from the Web Console or Command Line is available at the [Developer Guide](#).

Web Console

You can use the Web Console to view the state of your application components and launch new builds.

Command Line

With the [OpenShift command line interface \(CLI\)](#), you can create applications and manage projects from a terminal.

Development Resources

- [OpenShift Documentation](#)
- [OpenShift Origin GitHub](#)
- [Source To Image GitHub](#)
- [Getting Started with Node.js on OpenShift](#)
- [Stack Overflow questions for OpenShift](#)
- [Git documentation](#)

Lab 4 - Developing and Managing your Application

Developing and managing your application in OpenShift

In this lab we will explore some of the common activities undertaken by developers working in OpenShift. You will become familiar with how to use environment variables, secrets, build configurations, and more. Let's look at some of the basic things a developer might care about for a deployed app.

Pod logs

In the S2I lab we looked at a build log to inspect the process of turning source code into an image. Now let's inspect the log for a running pod - in particular let's see the web application's logs.

1. Go to the *Topology* view, click on our Node.js application to get the details pane, and go to the *Resources* tab.
2. Under *Pods*, click *View Logs* next to the running pod. This shows you all of the STDOUT/STDERR logs for the running pod.

These logs are only retained while the pod is active. An operator is available to easily install a support EFK stack to enable persistent logs.

Modifying the run time environment

Configuration of containers is typically controlled through environment variables. These variables are typically used to control connections to other services, like databases, or even to modify



debugging levels for different environments. Let's modify the environment variables for this running pod.

1. Go to the *Topology* view, click on our Node.js application to get the details pane.
2. From the *Actions* menu, select *Edit Deployment Config*. A *DeploymentConfig* is the Kubernetes resource type that controls the creation of your pod.
3. A view of the YAML definition for this *DeploymentConfig* is shown. All resources in Kubernetes are defined with a YAML document. For fine-grained control of your resources, you can submit and edit this YAML from the portal or the CLI. However, most common tasks can be done without editing the YAML directly. Click the *Environment* tab.
4. Two sections appear on this page for controlling the environment variables. The first allows for setting environment variables manually, the second allows a *ConfigMap* or a *Secret* to be imported. Although *ConfigMaps* and *Secrets* are very important to proper application control, we won't be using them today.
5. In the *Single Values* section, create a new variable named "TEST" and give it a value of 1.
6. Click *Save*
7. Go back to the *Overview* tab. In the circle status widget, notice that one application is *Pending* and another is *Running*. This will change to *Terminating* and *Running*. Finally, this will show only one Pod that is running. Because we changed the *DeploymentConfig*, an updated pod was deployed to the cluster. Once the new pod was healthy, the old pod was terminated. This is called a rolling deployment.

What about passwords and private keys?

Passwords and Private Keys should be configured using Secrets and not set directly as environment variables. Secrets help prevent exposure of confidential information. This is out of the scope of this workshop, but an important detail.

Getting into a pod

In traditional Linux environments, we use SSH to connect to remote servers. In a container world, it would be an anti-pattern to install an SSH server into every container as that SSH server is very likely not a dependency of the application. Instead, Kubernetes allows us to get a shell into the running container.

1. Go to the *Topology* view, click on our Node.js application to get the details pane, and go to the *Resources* tab.
2. Click on the name of the running pod under *Pods*.
3. Select the *Terminal* tab to be given a shell in the container. For pods with more than one container, you will have to choose a container to enter.
4. Try something out. Perhaps enter, “echo \$TEST” to see the value we set in the DeploymentConfig.

```
sh-4.2$ echo $TEST
1
sh-4.2$
```

Lab 5 - Build Triggers

Build triggers: manual, webhooks, and patching - Oh My!

Once you have an app deployed in OpenShift you can take advantage of some continuous capabilities that help to enable DevOps and automate your management process. We will cover some of those in this lab: Build triggers, webhooks, and rollbacks.

Build Triggers

When using S2I builds in OpenShift, there are four built-in ways to trigger an application be rebuilt.

- **Manual** - Manually tell OpenShift to start a new build. This can either be done from the portal or the CLI. This is great for building S2I images into your existing pipelines.
- **Webhook** - OpenShift hosts a secure API endpoint that can be called to trigger a build. This is great for connecting OpenShift to your Git system (GitHub, Gitlab, Microsoft TFS, etc).
- **Image Change** - The upstream source image is updated. This is great for automatically applying minor patches to your containers. All health checks are run before the new image is pushed into production with a rolling deployment style.
- **Configuration Change** - The DeploymentConfig configuration is updated. We did this one earlier!

Let's trigger a manual build:

1. Go to the *Topology* view, click on our Node.js application to get the details pane, and go to the *Resources* tab.



2. Click on *Start Build*. You'll see that Build #2 will go from started, to running, to complete. You can view the logs if you wish. Once the Build is finished, you can watch the new pod deploy from the details pane as well.

Let's discuss webhooks:

1. Go to the *Topology* view, click on our Node.js application to get the details pane, and go to the *Resources* tab.
2. In the *Builds* section, select the name of the build.

Builds

A screenshot of the OpenShift console's 'Builds' section for a service named 'nodejs'. The 'nodejs' label is highlighted with a red rectangle. To its right is a 'Start Build' button. Below this, a table lists two builds. The first row shows 'Build #2 is complete (less than a minute ago)' with a green checkmark icon and a 'View Logs' link. The second row shows 'Build #1 is complete (an hour ago)' with a green checkmark icon and a 'View Logs' link.

BC nodejs	Start Build
✓ Build #2 is complete (less than a minute ago)	View Logs
✓ Build #1 is complete (an hour ago)	View Logs

3. Scroll down to the bottom of the page and note the *Webhooks* section. You can click the link here to copy the URL to the webhook that would be needed by your version control system. This webhook can typically be configured in any Git based version control system. When a new commit is pushed to your repository, your VCS will notify OpenShift and automatically trigger a rebuild and redeploy. This behavior can, of course, be changed and tuned.

More control

OpenShift allows for additional control over deployments through the command line and YAML definitions. By default, OpenShift will perform canary, rolling deployments. But perhaps it is more appropriate to simply stop your old version and then start the new



one. This can be done as well with a *recreate* update strategy. Strategies can even get more detailed with the ability to create your own strategy for updating your application. If a deployment passes health checks, but still proves to be undesirable, a rollback can easily be initiated from the command line to restore to the last known working version. Later in this lab, we will walk through a Blue/Green deployment scenario.

Lab 6 - Replication and Recovery

Things will go wrong, and that's why we have replication and recovery

Things will go wrong with your software, or your hardware, or from something out of your control. But we can plan for that failure, and planning for it lets us minimize the impact. OpenShift supports this via what we call replication and recovery.

Replication

Let's walk through replicating, or horizontally scaling, our application. Replication allows for faster recovery after a pod failure and proper load balancing between available replicas.

1. Go to the *Topology* view, click on our Node.js application to get the details pane, and go to the *Overview* tab.
2. At the top of this page is a circle gauge showing 1 pod as active. Just to the right are up and down arrows. Click the up arrow twice.
3. The status should now show "1 - Scaling to 3". Continue watching the status until it shows "3 pods".
4. Go to the *Resources* tab and note that there are now three active pods.
5. Click the *Route* URL and note that behavior has not changed. The Route traffic (and Service traffic) are automatically load balanced between all of the active and healthy pods.

Recovery

If one pod were to fail, another will automatically be created to take its place. We can explore this, but we'll have to be quick... faster than OpenShift. Read through these directions before executing.

1. Go to the *Topology* view, click on our Node.js application to get the details pane, and go to the *Resources* tab.
2. Select the name of any of the active pods.

NOW YOU HAVE TO BE QUICK







3. In the *Actions* menu, select *Delete Pod* and confirm by pressing the *Delete* button.
4. Click *Topology* and select your Node.js application. If you were fast enough, you'll be able to see the original container being destroyed. You should also be able to see the new container being created. Through all of these actions, the load balancers for routes and services have automatically stayed up-to-date.

Application health

Pods can also enter a state called *CrashLoopBackoff*. This occurs when a particular pod crashes continually. If a pod crashes too many times, Kubernetes will wait before reattempting a deployment. You can emulate this by running “`pkill -9 node`” inside of the same pod a few times, but it is a bit tough to do in real time. If you are quick, you can see this in the details pane.



Pods

 nodejs-5-sxh8n	 Running	View Logs
 nodejs-5-vglcd	 Running	View Logs
 nodejs-5-kg97z	 Crash Loop Back Off	View Logs

Clean up

Scale your application back down to one replica before moving on with this lab.

Lab 7 - Labels

Labels

This is a pretty simple lab, we are going to explore labels. You can use labels to organize, group, or select API objects.

For example, pods are "tagged" with labels, and then services use label selectors to identify the pods they proxy to. This makes it possible for services to reference groups of pods, even treating pods with potentially different containers as related entities.

Labels on a pod

1. Go to the Topology view, click on our Node.js application to get the details pane, and go to the *Resources* tab.
2. Click on the name from the running pod under *Pods*.
3. From the *Actions* menu, select *Edit Labels*.
4. Add a label of "env=prod" and hit enter.
5. Click *Save*.

One such advantage of labelling our resources is to find them later using the built in search.

1. On the left hand menu, under *Advanced*, select *Search*.
2. In drop down menu to the left of the search bar, select *Pod*.
3. Search for env=prod in the search bar.

Search





4. Click Enter.
5. You can now see all of your production pods that are running.

Going further

Remember, everything in Kubernetes is simply a resource that was created through an API call to post the YAML definition. Everything. Use this to your advantage to stay organized. Any relevant metadata can be applied as a label.

Lab 8 - CI/CD Pipelines

CI/CD Defined

In modern software projects many teams utilize the concept of Continuous Integration (CI) and Continuous Delivery (CD). By setting up a tool chain that continuously builds, tests, and stages software releases, a team can ensure that their product can be reliably released at any time. OpenShift can be an enabler in the creation and management of this tool chain.

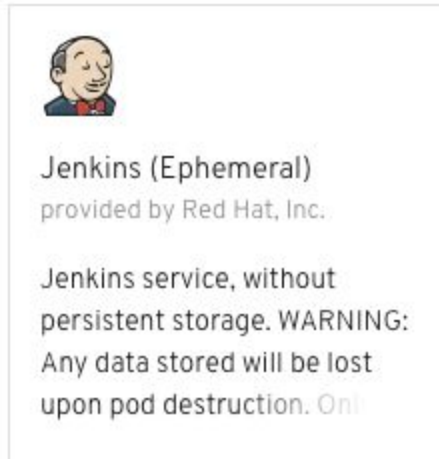
In this lab we walk through creating a simple example of a CI/CD pipeline utilizing Jenkins, all running on top of OpenShift! The Jenkins job will trigger OpenShift to build and deploy a test version of the application, validate that the deployment works, and then tag the test version into production.

In OpenShift 4.2, we have also introduced Tekton pipelines. However, they are still in Tech Preview so they will not be covered today. Tekton pipelines are meant to be far more Kubernetes native than other pipeline tools today.

Create a Jenkins server in your project

1. Click *+Add* from the left hand menu.
2. Select *From Catalog*

3. Search for “Jenkins” and select *Jenkins (Ephemeral)*.



4. Click *Instantiate Template*
5. All of the default options are appropriate for this workshop, select *Create* at the bottom of this page.
6. Go back to the *Topology* view and wait for the Jenkins server install and initial configuration to complete. This takes a while. If you are watching the logs, you'll eventually see a message that says, “INFO: Jenkins is fully up and running”. This means that you are getting close.

7. Once Jenkins is finished, you'll be able to go to the published route and get a screen like this:



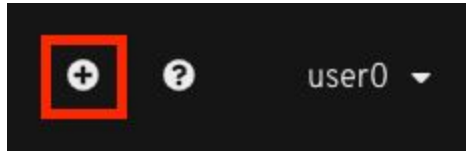
8. Click *Log in with OpenShift* and sign in with your OpenShift credentials.
9. Select *Allow selected permissions*
10. It will take a while for the next page to load. This is Jenkins instantiating your account for the first time. Fair warning, because of the load on this demo environment, it may time out. You can watch the Jenkins log to observe your account being added. Once you see a message like, "INFO: OpenShift OAuth: adding permissions for user user0, stored in the matrix as user0-admin-edit-view, based on OpenShift roles [admin, edit, view]", you are close.

Create a new application self-service template

To create a new application, we are going to add a template to the template catalog and then order that template for our project. This is one way of shortcutting common, repeatable deployments and enabling self-service.

1. Go back to the OpenShift console.

2. Go to the following URL:
https://raw.githubusercontent.com/rmkraus/ocp4-workshop/master/hello_world_template.json
3. Copy the entire contents of the file.
4. Click the plus sign on the right of the top bar.



5. Paste the YAML pipeline definition into the text box and click create.

Create an instance of our new application

1. From the left hand menu, click *+Add*
2. Select *From Catalog*
3. Search for “helloworld”
4. Select *nodejs-helloworld-sample*
5. Select *Instantiate Template*
6. The default options are fine, select *Create*.
7. In the topology view, you should now see two new DeploymentConfigs: frontend and frontend-prod.

Create a Jenkins pipeline using OpenShift

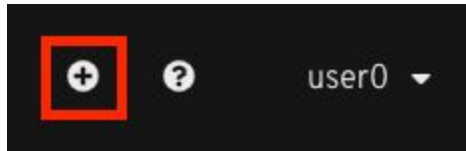
We will be creating the following very simple (4) stage Jenkins pipeline.

1. Build the application from source.
2. Deploy the test version of the application.
3. Submit for approval, then tag the image for production, otherwise abort.
4. Scale the application.



The first step is to create a build configuration that is based on a Jenkins pipeline strategy. The pipeline is written in the GROOVY language using a Jenkins file format.

1. Go to the following URL:
https://raw.githubusercontent.com/rmkraus/ocp4-workshop/master/jenkins_buildconfig.yml
2. Copy all of the YAML file contents
3. Click the plus sign on the right of the top bar.



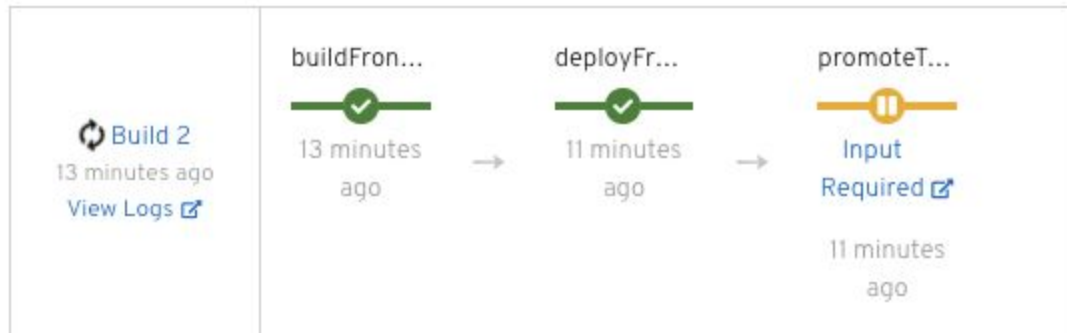
4. Paste the YAML pipeline definition into the text box and click create.

Start the pipeline

1. On the left hand menu, select *Builds*.
2. Select your new build definition called *pipeline*.
3. From the *Actions* menu, select *Start Build*
4. The *Build Overview* will update with the progress directly from Jenkins.

5. Watch the steps complete until it gets to the step that requires input. Then select *Input Required*.

Build Overview



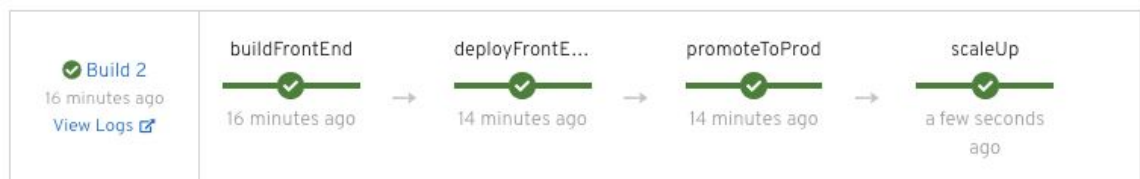
6. This will bring you to the Jenkins UI. In the Jenkins left hand menu, there should be an option called *Paused for Input*. Select that.
7. Select the *Promote* button to promote the project to production.

Promote to PROD?



8. Go back to the OpenShift page to see the completed pipeline.

Build Overview



9. Go to the topology page to see pods have been deployed to frontend and frontend-prod.



More complex pipelines could be created to fine tune the behaviors here. The takeaway is that OpenShift natively has full supported integration with Jenkins pipelines, even when using S2I builds!

Lab 9 - Blue / Green Deployments

Blue/Green Deployments

When implementing continuous delivery for your software one very useful technique is called Blue/Green deployments. It addresses the desire to minimize downtime during the release of a new version of an application to production. Essentially, it involves running two production versions of your app side-by-side and then switching the routing from the last stable version to the new version once it is verified. Using OpenShift, this can be very seamless because using containers we can easily and rapidly deploy a duplicate infrastructure to support alternate versions and modify routes as a service. In this lab, we will walk through a simple Blue/Green workflow with a simple web application on OpenShift.

Let's deploy some applications

1. Using the OpenShift expertise you have gained so far, deploy two Node.js applications using Node.js 10 and the example Git repository. Name one *blue* and the other *green*. Add them to a new application called "bluegreen". Only create a route for the blue application.
2. In the topology view, you should see both DeploymentConfigs in a single application bubble. Wait for both DeploymentConfigs

to go blue.



3. Notice the blue DC has a route as noted by the icon on the top right. It is the current production pod.

Let's make some changes

Imagine we have made some changes to the source code for our application. We would want to rebuild our green deployment, switch the traffic over, verify there are no issues, and then go home. Let's do

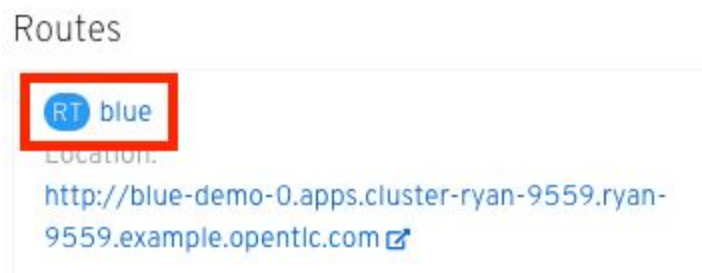


just that. We won't actually make any changes to the code, but we can walk through the rest of the procedure.

1. Click on the *green* deployment config to pull up the status pane. Go to the *Resources* tab.
2. Click *Start Build*. (Note, for Blue/Green deployments, you would likely not want to setup Git Webhooks to automate building. You could however control all this behavior in Jenkins pipelines!)
3. Once *Build #2* is complete, you should see OpenShift roll out the new build automatically.
4. Now is the time that you may do any final QA testing to your service before going live. We won't do this today, but being careful can be a good thing.

Now that we are satisfied that the green deployment has been updated and is operational, we will move our route from the blue deployment to the green deployment.

1. In the *Topology* view, click on the *blue* deployment.
2. In the *Routes* section, click on the name of the active route. It should be *blue*.

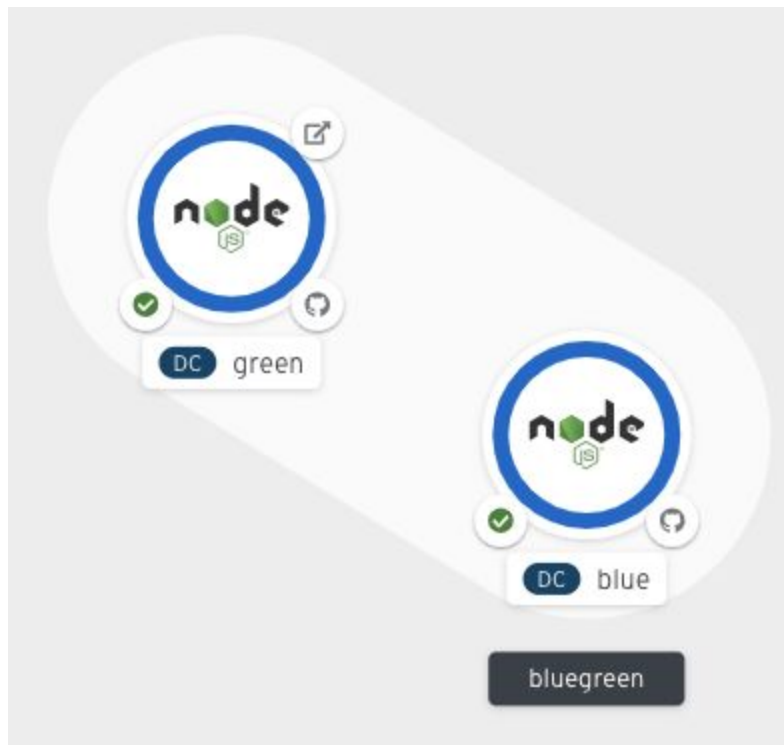


3. Go to the *YAML* tab as we will want to update this route's definition.

4. Line 27 of this YAML document, in the *spec* section defines the service to which this route points. Let's change the *name* from "blue" to "green".

```
22 spec:
23   host: blue-demo-0.apps.cluster-ryan-9559.ryan-9559.example.opentlc.com
24   subdomain: ''
25   to:
26     kind: Service
27     name: green
28     weight: 100
29   port:
```

5. Click *Save* and return to the *Topology* view.
6. Notice that our *green* deployment now has the route.



Success! Now all traffic will be directed to our green deployment. If we need to rollback, we could do this by simply updating the route again. This version of rollback can be popular as traffic can be redirected very quickly and in real time.

A/B Testing

A natural progression from Blue/Green deployments is A/B deployments. This involves splitting traffic across the old and new service for a time. Reducing the load on the old while increasing the load on the new. This can be done by adding an Alternate Backend to the route and applying weights. In our example, to route 70% to *green* and 30% to *blue* would look something like this.

```
spec:
  host: blue-demo-0.apps.cluster-ryan-9559.ryan-9559.example.opentlc.com
  subdomain: ''
  to:
    kind: Service
    name: green
    weight: 70
  alternateBackends:
  - kind: Service
    name: blue
    weight: 30
  port:
```

Note that the topology view is not currently able to display this correctly and will only show the route as being attached to *green*.

Service meshes

While outside the scope of this workshop, OpenShift does have a Service Mesh operator for easy management and supported installation of Istio and Kiali for even finer grained control of traffic. There is a performance penalty for using service meshes so they should probably not be universally applied, but only used when reasonable.



Fin

That's it!

Hopefully, these labs provided you some idea of how to perform common tasks within the OpenShift environment. And hopefully, you have a deeper understanding of how containers and container orchestration works. Please feel free to continue to "kick the tires" in the demo environment we've setup and explore both the web console and the oc command line client.

Learn more

If you would like your own developer version of OpenShift to continue exploring, check out our [Code Ready Containers](#). You'll need some good hardware, but it will setup a single node OpenShift cluster for testing and development. Also, feel free to reach out to your sales team with any additional questions.

Thank you and happy coding!