



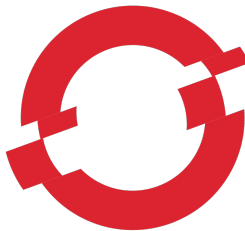
redhat®

THE NEW PaaS:

Using Docker and Containers to Simplify Your
Life and Accelerate Development on AWS

LAB 3: OPENSIFT

Version 1.2



OPENSIFT



Copyright © 2014 Red Hat, Inc.

Red Hat, the Shadowman logo, and the OpenShift logo are trademarks of Red Hat, Inc. in the United States and other countries. All other trademarks are the property of their respective owners.

Table of Contents

Introduction	4
Overview	4
What is OpenShift?	4
Technical Knowledge Prerequisites	4
Topics Covered	4
Sign in to the AWS Management Console	4
Using qwikLABS™ to sign in to the AWS Management Console	4
SSH Access.....	6
Module 1: Set Up Source Code in GitHub.....	6
1. Launch OpenShift.....	6
2. Fork the Application Code	7
3. Set up a Webhook for this Source Code	7
Module 2: Getting Under the Hood of OpenShift 3.....	11
1. Start a Private Docker Registry	11
2. Define a Build Configuration.....	12
3. Trigger a Build of Your Application Image	13
4. Configure and Launch an Application	15
Module 3: Redeploying with an Updated Image	18
1. Make Another Code Change.....	19
2. Trigger the App Redeployment	19
Module 4: Extra Credit.....	20
Check out the OpenShift structures in etcd.....	20
Conclusion.....	20
End Your Lab	21
Additional Resources	22

Introduction

Overview

In this lab we will explore the OpenShift 3 Platform-as-a-Service system as it builds upon Kubernetes and Docker to provide complete source-to-production application management¹.

What is OpenShift?

OpenShift is Red Hat's Platform-as-a-Service system. For developers, OpenShift automates the provisioning, management and scaling of applications so that you can focus on writing the code. For administrators, OpenShift provides automatic application stack provisioning and application scaling so that you can focus on managing systems, not developer resources.

Technical Knowledge Prerequisites

To successfully complete this lab, you should be familiar with SSH or PuTTY, the Linux command-line environment, Docker, and Kubernetes. For introductions to Docker and Kubernetes, refer to **Lab 1: Docker** and **Lab 2: Kubernetes** in this series.

Topics Covered

This lab will help you understand the OpenShift 3 system by walking you through key features:

- builds and build configurations
- Application configs and parameters
- Leveraging replicationControllers as part of a basic deployment strategy

Sign in to the AWS Management Console

If you are taking the entire three-lab course, you should still be logged in to the AWS Console from **Lab 1: Docker** and **Lab 2: Kubernetes**. If so, you can skip this section.

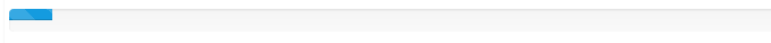
Using qwikLABS[™] to sign in to the AWS Management Console

Welcome to this self-paced lab! The first step is for you to sign in to Amazon Web Services.

1. To the right of the lab title, click **Start Lab**. If you are prompted for a token, use the one you received or purchased.

Note: A status bar shows the progress of the lab environment creation process. The AWS Management Console is accessible during lab resource creation, but your AWS resources may not be fully available until the process is complete.

 *Create in progress...*



¹ This lab is based on the simple-ruby-app example application from the OpenShift code base: <https://github.com/openshift/origin/tree/master/examples/simple-ruby-app>

2. On the lab details page, notice the lab properties.
 - a. **Duration** - The time the lab will run before automatically shutting down.
 - b. **Setup Time** - The estimated time to set up the lab environment.
 - c. **AWS Region** - The AWS Region in which the lab resources are created.

Duration (minutes): 600
Setup Time (minutes): 0
AWS Region: [us-east-1] US East (N. Virginia)

Note: The AWS Region for your lab will differ depending on your location and the lab setup.

3. In the AWS Management Console section of the qwikLAB™ page, copy the Password to the clipboard.

AWS Management Console

User Name:

Password:

4. Click the Open Console button.



5. Log into the AWS Management Console using the following steps.
 - a. In the **User Name** field type **awsstudent**.
 - b. In the **Password** field, paste the password copied from the lab details page.
 - c. Click **Sign in using our secure server**.

Amazon Web Services Sign In

Please enter the AWS Identity & Access Management (IAM) User name and password assigned by your system administrator to sign in.

AWS Account: 832809622232

User Name:

Password:

[Sign in using our secure server](#)

Please contact your system administrator if you have forgotten your user credentials.

[Sign in using AWS Account credentials](#)

Note: The AWS account is automatically generated by *qwikLAB™*. Also, the login credentials for the *awsstudent* account are provisioned by *qwikLAB™* using AWS Identity Access Management.

SSH Access

The **Lab 1: Docker** guide contains detailed information on how to establish an SSH connection to an EC2 instance. Refer there for information on how to connect to the `lab3_openshift` instance.

Module 1: Set Up Source Code in GitHub

In order to work with the OpenShift 3 system, first we need to have some source code that we want to build and publish as an application. We will also be taking advantage of a notification service called a “webhook”, which is going to automatically notify our OpenShift system when we update our app.

1. Launch OpenShift

In a terminal window, log into the OpenShift host and run the following command to start the service:

```
$ openshift start --listenAddr="0.0.0.0:8080"
```

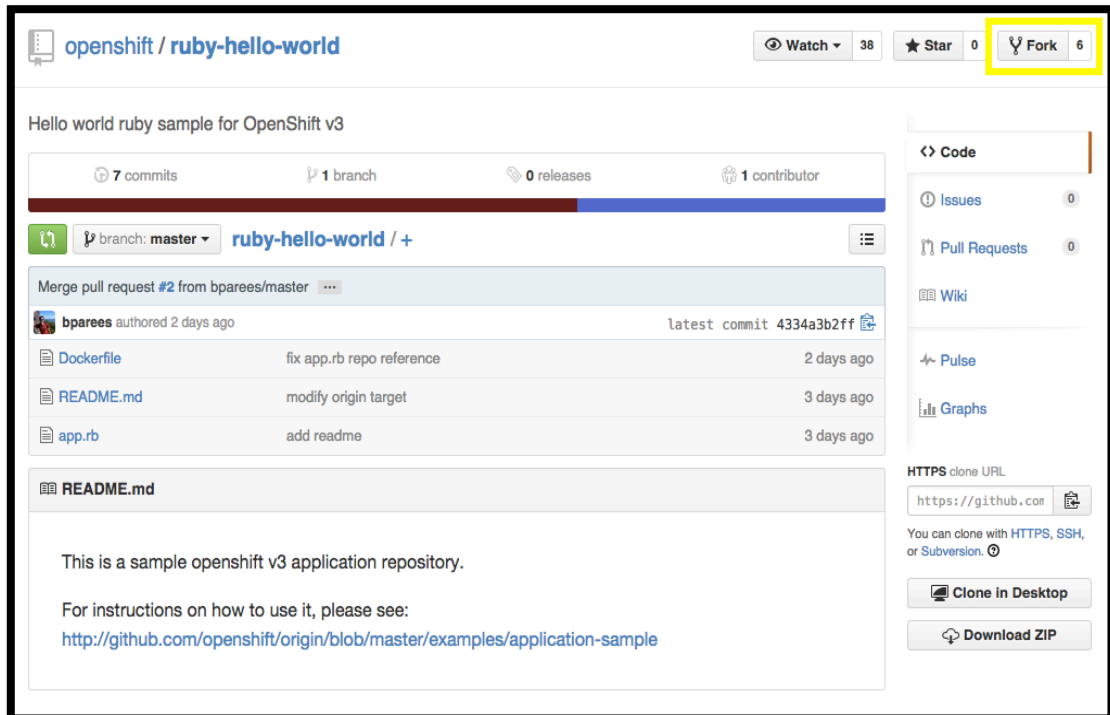
OpenShift will run as a foreground process in this terminal, and any log data that is generated by OpenShift will be visible here, too.

2. Fork the Application Code

Open up a web browser to the following URL:

```
https://github.com/openshift/ruby-hello-world
```

In the upper right-hand corner of the page, find and press the Fork button:



Specify your own GitHub account as the target location.

3. Set up a Webhook for this Source Code

The webhook that we create in this step will automatically notify our OpenShift system whenever we push a code update to our GitHub repository.

1. Leave the OpenShift system running in one terminal ("terminal 1"). Open a second terminal ("terminal 2") and log in to the OpenShift host there.
2. In order to generate your webhook, you need to know the public IP of the OpenShift host. First, from terminal 2, try running the shell function `lab_generate_webhook`:

```
$ lab_generate_webhook  
http://54.90.90.36:8080/osapi/v1beta1/buildConfigHooks/build10  
0/secret101/github
```

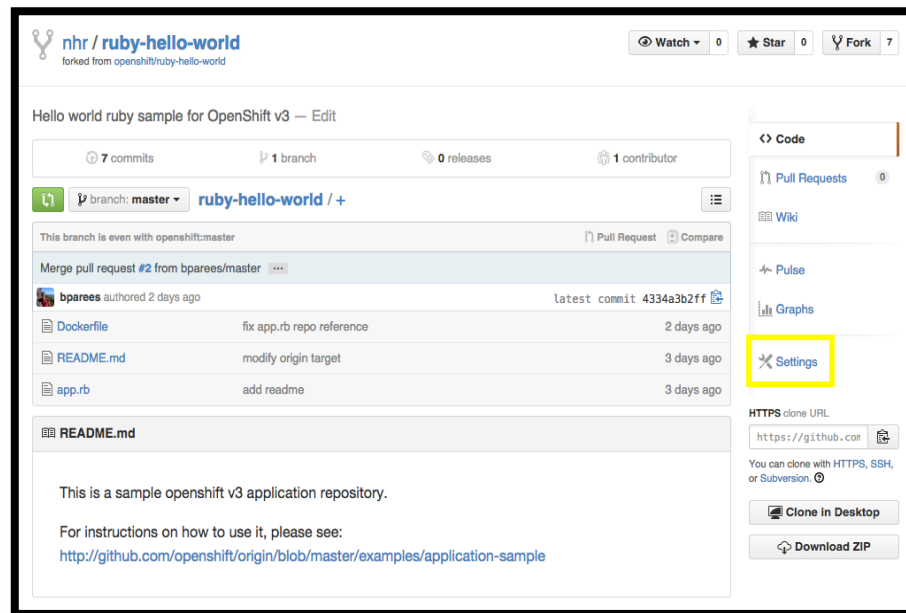
If that doesn't work, you will need to find the public IP address of this AWS instance in the EC2 dashboard. Then the formula for the webhook is:

```
http://<PUBLIC_IP>:8080/osapi/v1beta1/buildConfigHooks/build10  
0/secret101/github
```

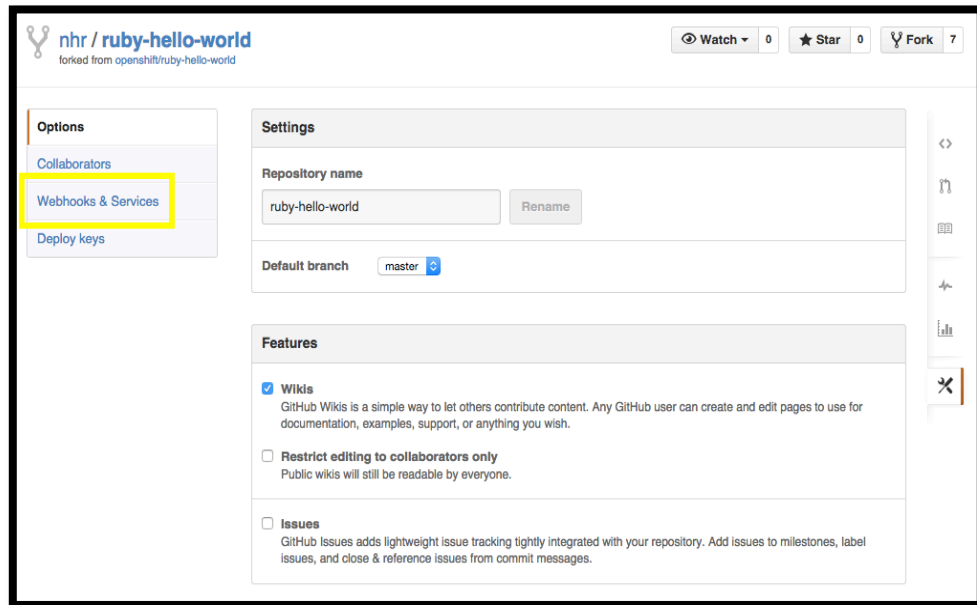
3. Now, over in your web browser, first make sure you are looking at your fork of the ruby-hello-world application. The URL should be:

```
https://github.com/<YOUR_GITHUB_UNAME>/ruby-hello-world
```

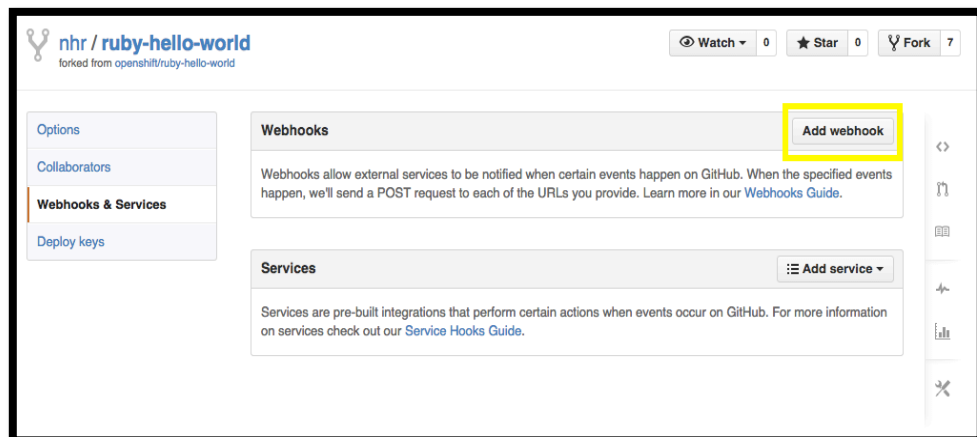
4. Now, find and click the Settings icon on the right side of the page:



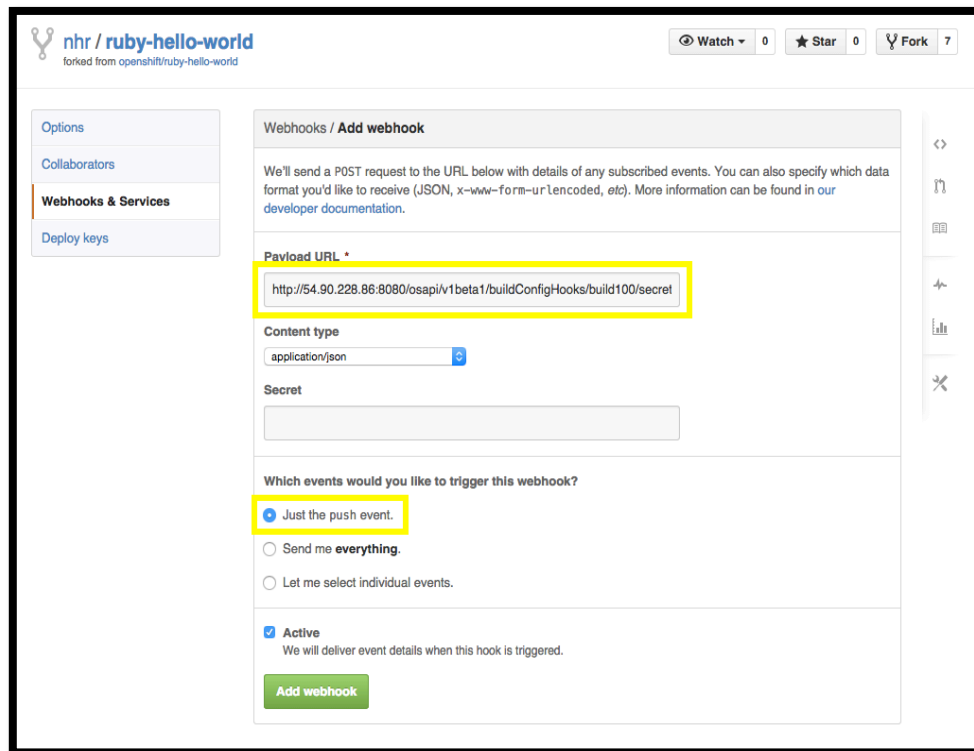
5. And from the Settings page, find and click the Webhooks & Services link on the left:



6. Almost there! Now, from the Webhooks & Services page, find and press the Add a Webhook button:



7. Now, in the Payload URL field, copy the Webhook URL that you generated in step #2. Leave the other fields exactly as you seem them here:



The screenshot shows the GitHub 'Add webhook' interface. On the left is a sidebar with links: Options, Collaborators, Webhooks & Services (highlighted), and Deploy keys. The main content area is titled 'Webhooks / Add webhook'. It contains a description of webhooks, a 'Payload URL' field with the value 'http://54.90.228.86:8080/osapi/v1beta1/buildConfigHooks/build100/secret' (highlighted with a yellow box), a 'Content type' dropdown set to 'application/json', a 'Secret' text input field, and a section 'Which events would you like to trigger this webhook?' with three radio button options: 'Just the push event.' (selected and highlighted with a yellow box), 'Send me everything.', and 'Let me select individual events.'. Below this is a checked 'Active' checkbox with a note 'We will deliver event details when this hook is triggered.'. At the bottom is a green 'Add webhook' button.

8. Finally, press the Add webhook button.

Once the webhook is created, GitHub will try to validate it by sending a test notification to our OpenShift system. This won't work yet, so don't panic if you notice any failed webhook attempts.

Okay, all set! You are now ready to play around with OpenShift 3!

Module 2: Getting Under the Hood of OpenShift 3

In this portion of the lab, we will walk through the process of defining and deploying an app in the OpenShift PaaS at the system level.

This walkthrough assumes that you are still running the OpenShift process in one terminal window ("terminal 1") and that you are also logged into the OpenShift host with another terminal window ("terminal 2").

1. Start a Private Docker Registry

When OpenShift builds an image for you, it will automatically store that image in a private registry that is managed by the OpenShift system. We will launch the private registry now, using the same tools that we would use to launch any other application.

First, move into the `/root/simple-ruby-app` directory and have a look at `registry_config/registry_config.json`:

```
{
  ...
  "description": "Creates a private docker registry",
  "id": "docker-registry-config",
  "items": [
    {
      "containerPort": 0,
      "id": "registryservice",
      "kind": "Service",
      "port": 5000,
      "selector": {
        "name": "registryPod"
      },
      ...
    },
    {
      "id": "registryController",
      "kind": "ReplicationController",
      "desiredState": {
        "podTemplate": {
          "desiredState": {
            "manifest": {
              "containers": [
                ...
              ]
            }
          }
        }
      }
    }
  ]
}
```

This file describes a processed OpenShift application `config` object. We'll dig into `config` objects more later in the lab, but for now recall that an application config is a collection of related Kubernetes objects. In this case, the registry consists of a service and a replicationController.

Fire up this config with the `apply` command:

```
$ openshift kube apply -c registry_config/registry_config.json
```

Then watch the pod list with `openshift kube list pods` until the “registry” pod status changes to “running”:

```
$ openshift kube list pods
ID          Image(s)      Host          Labels          Status
-----
a09fc80d    registry      127.0.0.1/    name=registryPod,replicationController=... Running
```

Once the registry is running, head on to the next section or try the exercise below.

EXERCISE

Looking for a challenge? `registry_config/registry_config.json` describes a very simple but complete app configuration consisting of a `replicationController` and a `service`. Use it as the basis for another application based on the `nginx:latest` Docker image. You’ll need to:

- Change the pod info within the `replicationController` definition
- Update port mappings
- Change the `labels` -> `name` values and the corresponding selector and `replicaSelector` settings

2. Define a Build Configuration

A build configuration tells OpenShift:

- The type of build associated with an app (“docker” or “sti”²)
- Where the source code can be found
- A name for the resulting Docker image

Have a look at `buildcfg/buildcfg.json`:

```
{
  "id": "build100",
  "desiredInput": {
    "type": "docker",
    "sourceURI": "git://github.com/openshift/ruby-hello-world.git",
    "imageTag": "openshift/origin-ruby-sample",
  },
  "secret": "secret101"
}
```

² “source-to-image”. This walkthrough uses the `docker` build method, meaning that the source code must include a `Dockerfile`. By contrast, a source-to-image build would not require a `Dockerfile`; instead it would use a specific, purpose-built Docker image to pull and potentially compile the source code into a new app image.

Notice the “secret” parameter. This value may look familiar from when we set up a webhook for our application on GitHub. This value will be passed by GitHub as part of the webhook to validate the notification.

Before we can create the build configuration, we need to edit this file so that it uses our fork of the app code on GitHub.

1. Modify line 6 of the file, replacing ‘openshift’ with your GitHub username:

```
"sourceURI": "git://github.com/openshift/ruby-hello-world.git",
```

becomes

```
"sourceURI": "git://github.com/<GITHUB_USERNAME>/ruby-hello-  
world.git",
```

2. Now that this is done, you can submit the buildConfig definition for your application:

```
$ openshift kube create buildConfigs -c buildcfg/buildcfg.json
```

The buildConfig is created immediately, but you can verify it with the list command:

```
$ openshift kube list buildConfigs
```

3. Trigger a Build of Your Application Image

Now that we have created a buildConfig, we can trigger a build of our Docker image by modifying and committing an update of the application's source code.

1. In your terminal window, run a watch of the OpenShift builds list:

```
$ watch openshift kube list builds
```

You won't see anything here yet.

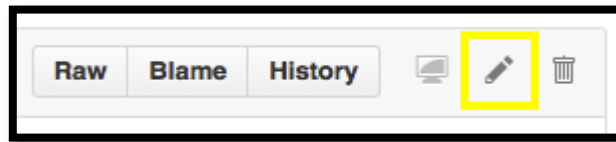
2. Now, in your web browser, navigate to your fork of the application code at:

```
https://github.com/<GITHUB_USERNAME>/ruby-hello-world
```

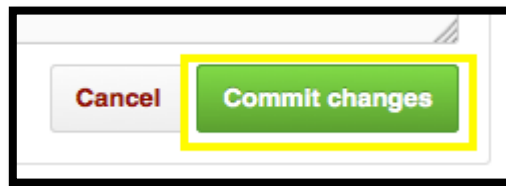
3. Edit your application code directly on the GitHub site:

- a) Click on the app.rb file link

- b) In the file view, next to the Raw, Blame and History buttons, find and click on the Pencil icon:



- c) Modify line 6 of the file to say something other than "Hello World!"
- d) In the lower right-hand corner of the screen, find and click the Commit changes button:



4. Now, quickly jump back to the terminal where you are watching the build list. You should see a new build appear, with a state value that goes from 'new' to 'pending', then to 'running' and eventually to 'complete'.

ID	Status	Pod ID
8586cfde-427b-11e4-8380	running	build-docker-8586cfde-427b-11e4-8380

If you break out of the `watch` with `<CTRL>+C`, then you can run a `docker ps` to see that OpenShift has fired up a `docker-builder` container where your new docker image is being constructed:

```
$ docker ps
CONTAINER ID        IMAGE
9e70c4155cfa        nhripps/docker-builder:lab
```

5. Once the status of the build is 'complete' you should have a new Docker image. To confirm it, run `docker images` and look for an image with a name that ends in `origin-ruby-sample`

The Docker image that was built by OpenShift is automatically pushed to the private Docker repository that we started earlier. You can see a listing of images specifically stored on that registry with the `curl` command:

```
$ curl -XGET http://${DOCKER_REGISTRY}/v1/search
```

You should see output similar to:

```
{"num_results": 1, "query": "", "results": [{"description": "",  
"name": "openshift/origin-ruby-sample"}]}
```

4. Configure and Launch an Application

In order to start our private registry, we used the `apply` command to cause OpenShift to create a few different Kubernetes objects. Now we're going to do the same thing again, but this time we're launching an application based on the Docker image that we just built from source.

Have a look at `template/template.json`:

```
{  
  "id": "ruby-helloworld-sample-template",  
  "kind": "Template",  
  "name": "ruby-hello-world-template",  
  "description": "A simple ruby application in openshift origin v3",  
  "parameters": [  
    {  
      "name": "ADMIN_USERNAME",  
      "description": "administrator username",  
      "type": "string",  
      "expression": "admin[A-Z0-9]{3}"  
    },  
    {  
      "name": "ADMIN_PASSWORD",  
      "description": "administrator password",  
      "type": "string",  
      "expression": "[a-zA-Z0-9]{8}"  
    },  
    {  
      "name": "DB_PASSWORD",  
      "description": "",  
      "type": "string",  
      "expression": "[a-zA-Z0-9]{8}"  
    }  
  ],  
  "items": [  
    {  
      "id": "frontend",  
      "kind": "Service",  
      "apiVersion": "v1beta1",  
      "port": 5432,  
      "selector": {  
        "name": "frontend"  
      }  
    },  
    {  
      "id": "frontendController",  
      "kind": "ReplicationController",  
      ...  
    }  
  ],  
}
```

In addition to defining a list of items, this template also defines a list of parameters.

What are parameters?

These are values that can be used elsewhere in the config. Parameters can be randomized or hard-coded; it all depends on the expression definition for the parameter's value.

How are parameters used?

In the frontendController definition, we can see that the parameters defined above will be passed to the associated Docker images as environment variables:

```
{
  "id": "frontendController",
  "kind": "ReplicationController",
  "apiVersion": "v1beta1",
  "desiredState": {
    "replicas": 1,
    "replicaSelector": {"name": "frontend"},
    "podTemplate": {
      "desiredState": {
        "manifest": {
          "version": "v1beta1",
          "id": "frontendController",
          "containers": [{
            "name": "ruby-helloworld",
            "image": "localhost:5000/openshift/origin-ruby-sample",
            "env": [
              {
                "name": "ADMIN_USERNAME",
                "value": "${ADMIN_USERNAME}"
              },
              {
                "name": "ADMIN_PASSWORD",
                "value": "${ADMIN_PASSWORD}"
              },
              {
                "name": "DB_PASSWORD",
                "value": "${DB_PASSWORD}"
              }
            ],
            "ports": [{"containerPort": 8080}]
          }
        ]
      },
      "labels": {"name": "frontend"}
    },
    "labels": {"name": "frontend"}
  }
}
```

Before we can use this template, we have to do a few things:

1. In your terminal window, run `echo $DOCKER_REGISTRY` and take note of the IP address. It will be the value that precedes `:.5000`:


```
$ echo $DOCKER_REGISTRY
10.99.182.25:5000
```

(In case you are curious, the value you will see is the eth0 IP address)

2. Now edit `template/template.json`. On line 50, change `localhost` to the IP address that you found in step 1:

```
"image": "localhost:5000/openshift/origin-ruby-sample",
```

becomes

```
"image": "<YOUR_ETH0_IP>:5000/openshift/origin-ruby-sample",
```

3. Finally, before the template can be applied, it has to be processed. This is accomplished with the `process` command:

```
$ openshift kube process -c template/template.json >
processed/template.processed.json
```

Once you have processed the template, have a look at the newly generated version at `processed/template.processed.json`. There are a few things to take note of:

- The `parameters` section is no longer present. The processing step removes this section after generating and applying the values that were defined there.
- In the `env` list of the `frontendController` object, the placeholder values have been replaced with values generated by the template processor.

Now you can start up the application with the `apply` command:

```
$ openshift kube apply -c processed/template.processed.json
```

Once the command exits, you can start to see the Kubernetes objects that have been created with the `list` commands:

- `openshift kube list services`
- `openshift kube list replicationControllers`
- `openshift kube list pods`

Finally, to connect to the application itself, use `curl localhost:5432`. You should see output similar to the following:

```
$ curl localhost:5432
Hello World! Welcome to OpenShift!
User is adminS5G
Password is 6E7MiTv2
DB password is qLq3dLI1
```

That's the app that you created from some simple source code, compiled into a Docker image and deployed into a Kubernetes cluster. OpenShift took care of building the source code and deploying the necessary pods and services. Nice work!

Following the Parameter Trail

Notice the User, Password and DB password values. Do they look familiar?

- `template/template.json` - Rules for generating these values
- `processed/template.processed.json` - Generated values applied to template
- `ruby-hello-world/app.rb` - Our app picks up the values as environment variables

This workflow is essential to making reusable applications. By leveraging environment variables in the application itself, we keep secrets (like passwords) out of the source code, and we also avoid vendor lock-in. The same app can run on its own outside of the context of a PaaS.

EXERCISE

Using the `registry_config/registry_config.json` file as a start (or your own `config.json` file from the previous exercise), replace some of the hard-coded values in the file (like port numbers, for instance) with parameters, and then add a parameter section to define rules for generating the values. Refer to the `template/template.json` file for guidance on how to format the parameters list.

Review

In this part of the lab, we looked over the components of OpenShift that take application source code, build it into a Docker image, and then deploy the image as part of a complete application on Kubernetes. The steps that we did by hand will be fully automated by OpenShift 3.

But once an application is already deployed, how does OpenShift handle the building and re-deployment of that app when the code is changed again? Read on...

Module 3: Redeploying with an Updated Image

Having manually gone through all of the steps that OpenShift 3 performs automatically gives you a sense for how the system works. However, it doesn't give you a good sense of what the developer experience will be. Typically, a developer will be iterating over the app code with code commits and seeing those updates occur almost instantly in the app itself.

In this part, you will walk through a simple demonstration of the redeployment mechanism that OpenShift uses for app code updates.

1. Make Another Code Change

In your web browser, navigate back to your sample app and make another change to app.rb:

<> Code **Preview**

```
1  require 'sinatra'
2
3  set :bind, '0.0.0.0'
4  set :port, 8080
5  get '/' do
6    "Hello World! Welcome to Las Vegas!\n"+
7    # ENV values are generated during template processing
8    # and then passed to the container when openshift launches it.
9    "User is #{ENV['ADMIN_USERNAME']}\n"+
10   "Password is #{ENV['ADMIN_PASSWORD']}\n"+
11   "DB password is #{ENV['DB_PASSWORD']}\n"
12  end
13
```

Then press the Commit changes button to commit the new code, and watch the OpenShift builds until the new image is completed:

```
$ openshift kube list builds
```

Now if you run `docker images`, you will see that the original app image has been untagged, and the newly built app image is now labeled with `origin-ruby-sample`:

```
$ docker images
REPOSITORY                                TAG      IMAGE ID
10.139.85.181:5000/openshift/origin-ruby-sample  latest   53a8558b3c53
<none>                                       <none>   da553ecc3b5c
```

2. Trigger the App Redeployment

The first part of this operation was pretty familiar: change some code, trigger a build, watch a new image get created. However, in this step we will watch as OpenShift leverages the Kubernetes replicationController concept to automatically redeploy rebuilt app images.

To see how this is done, simply kill the currently running app container:

```
$ docker kill `docker ps | grep helloworld | awk '{print $1}'`
```

And now watch the currently running Docker container list:

```
$ watch docker ps
```

Within a few moments, you should see a new container appear. This is your app, redeployed from the rebuilt app image1. To confirm this, use `curl` to reconnect to the app:

```
$ curl localhost:5432
Hello World! Welcome to Las Vegas!
User is admin0KT
Password is 262yMqKP
DB password is 52QuCp2Q
```

The behavior represents the most basic deployment strategy: take down the running image and let Kubernetes restart the pod with the newly updated image. As OpenShift 3 matures, other deployment strategies will emerge to provide other behaviors, like zero-downtime upgrades. Great job!

Module 4: Extra Credit

Time permitting, here's another thing to try:

Check out the OpenShift structures in `etcd`

Using the `etcdctl` utility, you can see how OpenShift uses the `etcd` service to add PaaS-specific objects to the cluster. Start exploring by listing out the contents of the root level of the `etcd` keystore:

```
$ etcdctl ls /
```

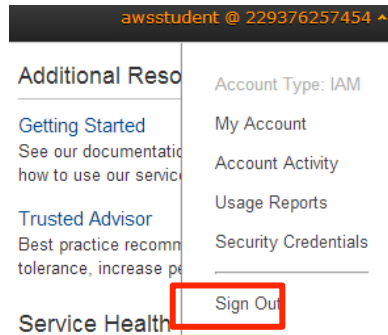
Conclusion

Congratulations! You now have successfully:

- Explored the OpenShift extensions to Kubernetes: builds, buildConfigs and config templates.
- Used OpenShift to build and deploy a Docker container starting with app source code.
- Updated the app source and manually traversed the process of app update deployment.

End Your Lab

1. To log out of the AWS Management Console, from the menu, click **awsstudent @ [YourAccountNumber]** and choose **Sign out** (where [YourAccountNumber] is the AWS account generated by *qwikLAB™*).



2. Close any active SSH client sessions or remote desktop sessions.
3. Click the **End Lab** button on the *qwikLAB™* lab details page.



4. When prompted for confirmation, click **OK**.
5. For **My Rating**, rate the lab (using the applicable number of stars), optionally type a **Comment**, and click **Submit**.

My Rating: ★★★★★ None

Comment:

Note: The number of stars indicates the following:

- 1 star = Very dissatisfied
- 2 stars = Dissatisfied
- 3 stars = Neutral
- 4 stars = Satisfied
- 5 stars = Very satisfied.

You may close the dialog if you do not wish to provide feedback.



Additional Resources

For more information about Docker, Kubernetes and OpenShift, go to the [OpenShift Origin repo on GitHub](#).