OpenShift Online 3

Getting Started

Getting Started with OpenShift Online

Last Updated: 2018-09-15

OpenShift Online 3 Getting Started

Getting Started with OpenShift Online

Legal Notice

Abstract

Quickly build, launch, and scale container-based web apps in a public cloud environment using OpenShift Online 3.

Table of Contents

CHAPTER 1. OVERVIEW

OpenShift Online 3 is Red Hat's application hosting platform that makes it easy for developers to quickly build, launch, and scale container-based web apps in a public cloud environment.

Check out the following topics to get started as an **application developer** trying out OpenShift Online 3:

- Step through a **basic walkthrough** using the web console and create your first project and application.
- Go **beyond the basics** and get hands-on with the CLI.
- Connect to OpenShift Online using **Eclipse tooling**.
- If you are familiar with OpenShift Online 2, learn about some **architectural and terminology changes** introduced with OpenShift Online 3.

CHAPTER 2. CHOOSE A PLAN

2.1. REVIEW PLANS AND REGISTER

2.1.1. Red Hat OpenShift Online

Review the differences between OpenShift Online Starter and OpenShift Online Pro plans, then complete the registration process.

When registering, you can redeem valid, one-time-use promotional coupons, if applicable.

Once registered, customize your account.

2.1.2. Red Hat Fuse Online

Red Hat Fuse Online is a low-code integration platform to help business users build integration solutions faster.

Red Hat Fuse Online is hosted on Red Hat OpenShift Online, so you can get up and running quickly. It can also be deployed anywhere OpenShift is supported—Red Hat managed, public cloud, or private cloud. Learn more and sign up for the technology preview.

2.2. ADDITIONAL LEARNING

You can explore Red Hat OpenShift's Interactive Learning Portal and Kubernetes By Example to learn more about OpenShift technology.

2.3. NEXT UP: BASIC WALKTHROUGH

Once you are registered, go to Basic Walkthrough to learn how to get a simple project up and running.

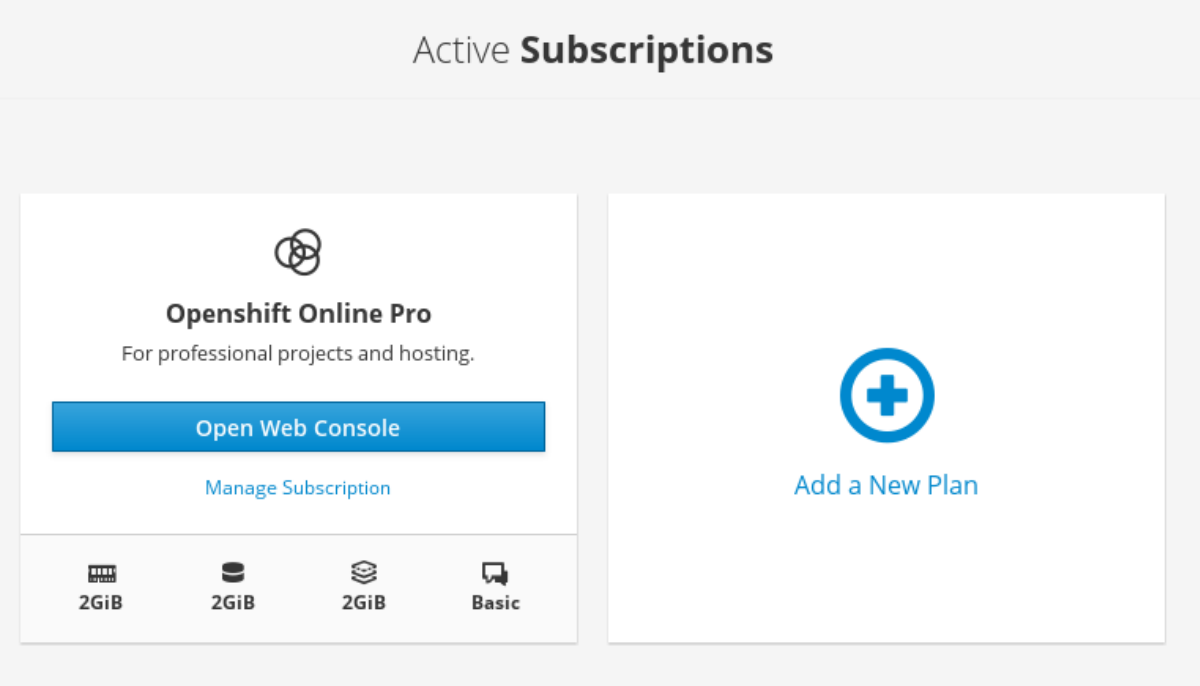
CHAPTER 3. MANAGE YOUR PLAN

3.1. MANAGE YOUR OPENSHIFT ONLINE PLAN

3.1.1. Customize Your Plan with Add-ons

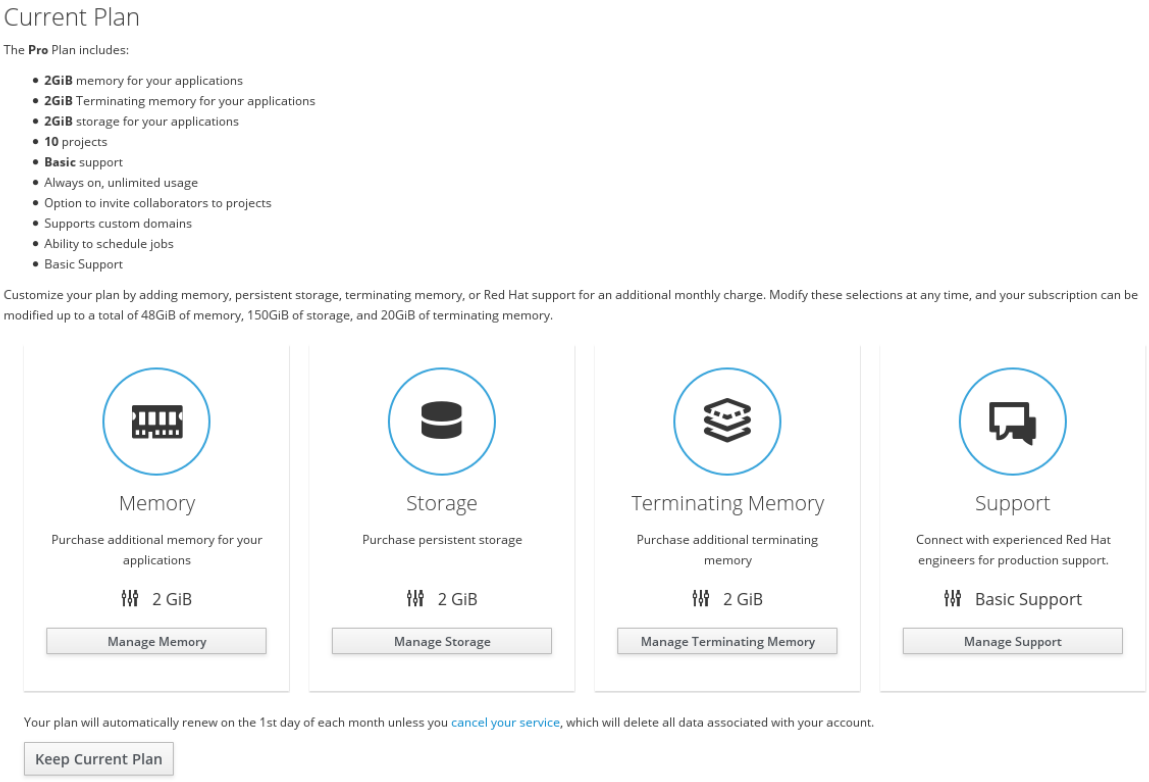
You can customize your plan with add-ons.

1. In the **Active Subscriptions** page, click the **Manage Subscription** link.



2. Click **Change** by the **Current Plan** heading

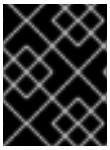
3. From the **Current Plan** page, manage memory, storage, terminating memory, and support options. Add on resources for an additional cost.
   Click **Manage Memory**, **Manage Storage**, **Manage Terminating Memory**, or **Manage Support** and follow the workflow for each.



3.1.2. Upgrading Your Plan

To upgrade your plan from OpenShift Online Starter to OpenShift Online Pro, visit **`manage.openshift.com`**.

See Creating a Template from Existing Objects for guidance on exporting all of your existing objects.



IMPORTANT

OpenShift Online Starter users, if you violate the Terms of Service Agreement, then your account will be terminated and you will be banned from creating subscriptions.

3.1.3. Apply Coupons

You can redeem valid, one-time-use promotional coupons, if applicable.

You can also redeem coupons after the registration process:

1. From the **Active Subscriptions** page, click **Manage Subscriptions**.
2. On the next **Subscriptions** page, click **Manage** next to the **Coupons** heading.
3. On the next **Subscription Coupons** page, you will see (if applicable) coupons that are already in use. To apply a new coupon, click **Apply New Coupon** and complete the workflow.

CHAPTER 4. BASIC WALKTHROUGH

4.1. OVERVIEW

This guide demonstrates how to get a simple project up and running on OpenShift Online.

The following sections guide you through creating a project that contains a sample Node.js application that will serve a welcome page and the current hit count (stored in a database) using the OpenShift Online web console. This involves creating two pods:

- one to host the Node.js application
- one to host the MongoDB database

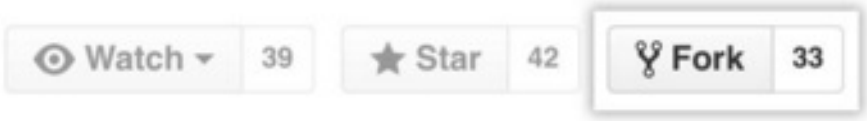
The tutorial assumes that you have:

- an OpenShift Online account.
- a GitHub account.
- Git installed locally.

4.2. SETUP

In this section, you will *fork* the OpenShift Node.js sample application on GitHub and clone the repository to your local machine so that you can deploy and edit the app.

1. On GitHub, navigate to the **openshift/nodejs-ex** repository. In the top-right corner of the page, click **Fork**:



2. Next, execute the following commands on your local machine to clone the sample application and change to the new directory:

```
$ git clone https://github.com/<your_github_username>/nodejs-ex
$ cd nodejs-ex
```

That's it! Now, you have a fork of the original **openshift/nodejs-ex** example application Git repository and a copy on your local machine.

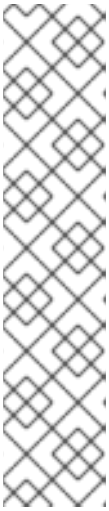
4.3. GET ORIENTED WITH THE WEB CONSOLE

Visit the Web Console topic to learn about its components.

4.4. CREATING A NEW APPLICATION

In this section, you will deploy your first application to OpenShift Online using the web console.

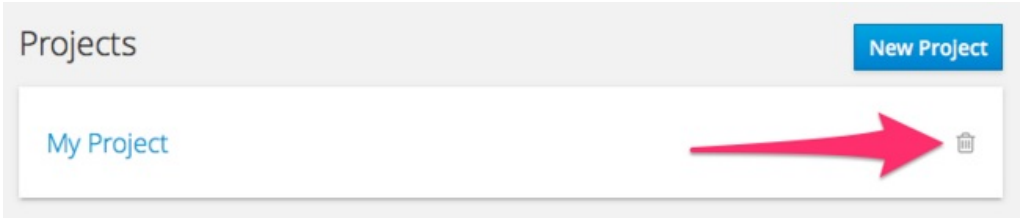
1. Navigate to the welcome page of the OpenShift Online web console and click **Create Project** to create your first project:



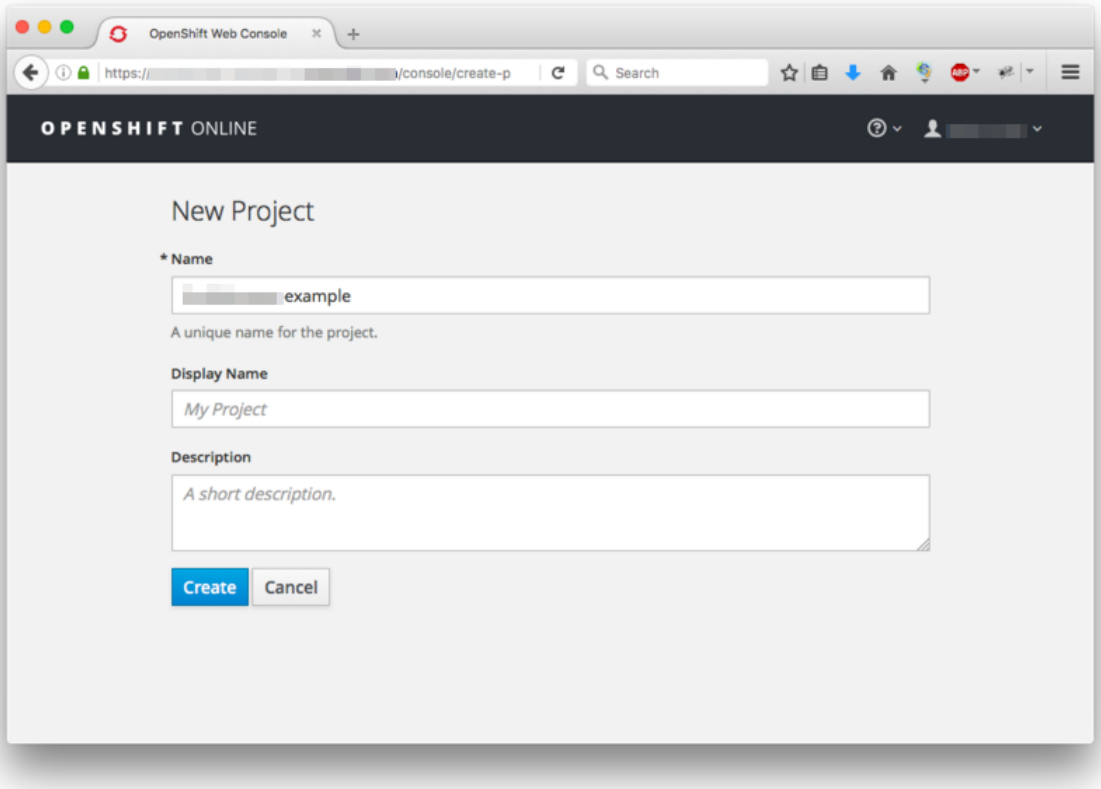
NOTE

OpenShift Online Starter only allows you to create a single project at this time. If you already have a project, you must delete it in order to continue.

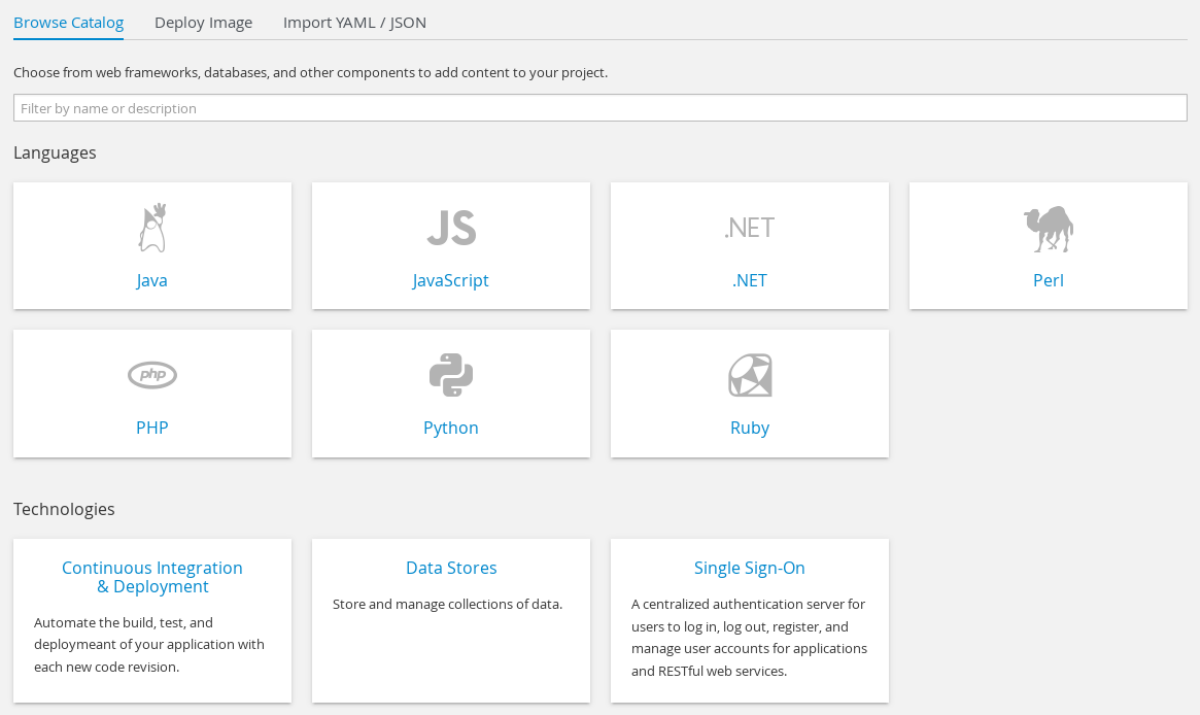
To delete your existing project, click the trash can icon next to the project name on the welcome page:



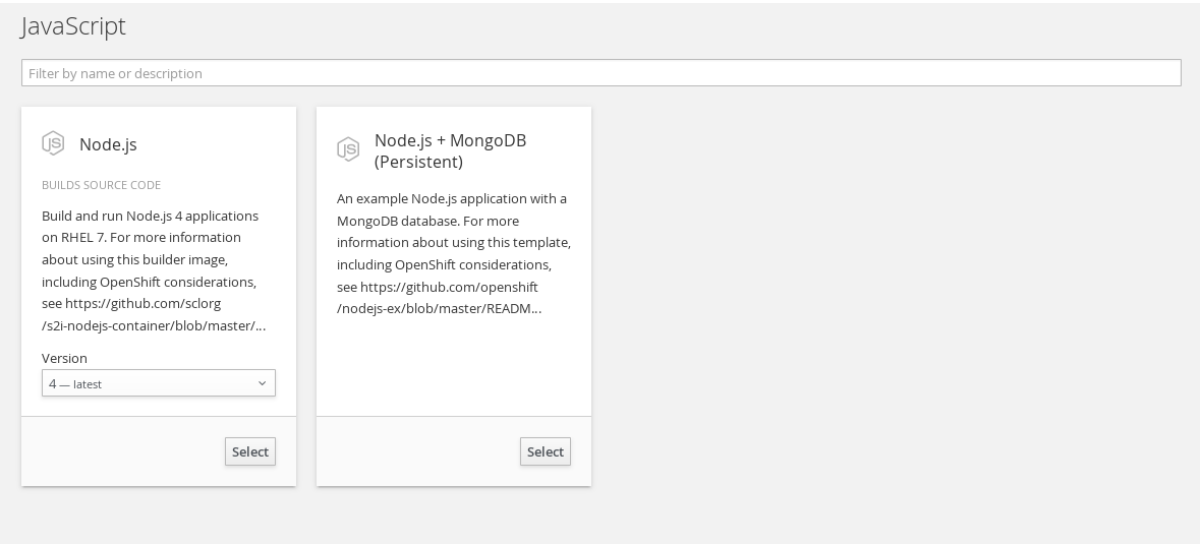
2. Replace **`my-project`** with a unique name for your project, such as **`<your_github_username>-example`**. You can leave the display name and description blank.



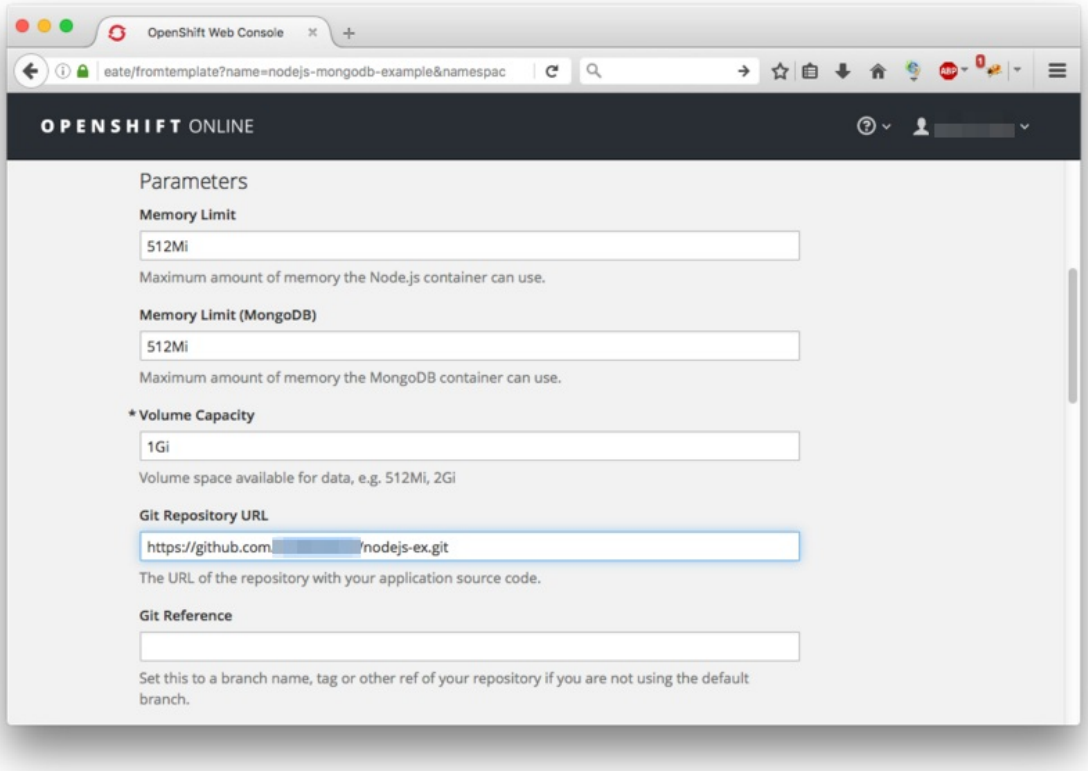
3. Click on the **JavaScript** option:



4. Select the **nodejs-mongodb-example** Quickstart template:



5. On the next screen, replace the user name in the **Git Repository URL** parameter with your GitHub user name. Use the default values provided for all other parameters:



6. Finally, scroll to the bottom of the page and click **Create** to deploy your application.

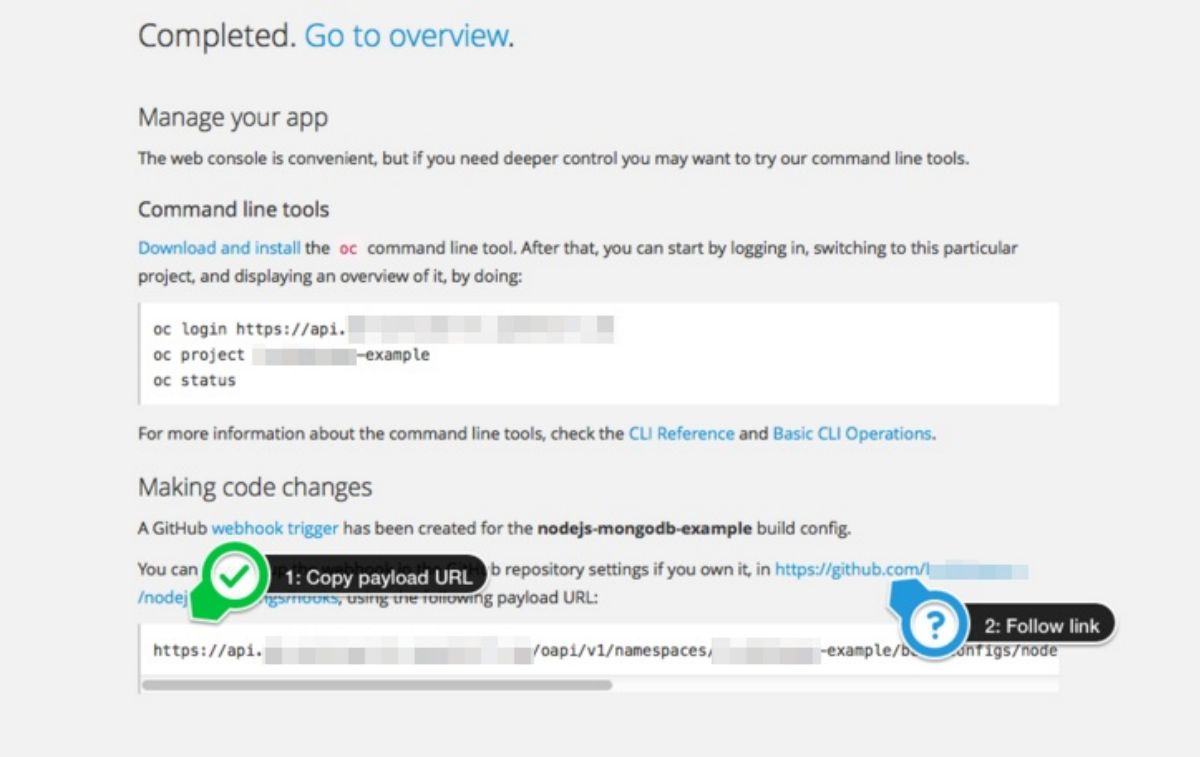
NOTE

You can follow along on the **Overview** page of the web console to see the new resources being created, and watch the progress of the build and deployment. While the MongoDB pod is being created, its status is shown as pending. The MongoDB pod then starts up and displays its newly-assigned IP address.

4.5. CONFIGURING AUTOMATED BUILDS

In this section, you will configure a GitHub webhook to automatically trigger a rebuild of your application whenever you push code changes to your forked repository. This involves adding the Github webhook URL from your application into your Github repository. You obtain this webhook from these locations:

- At the bottom of **Next Steps** page shown after creating your app, you will see a section titled **Making code changes**. Copy the payload URL from the bottom of the page and follow the link to the GitHub project webhook settings page provided:



- In the OpenShift Online web console:

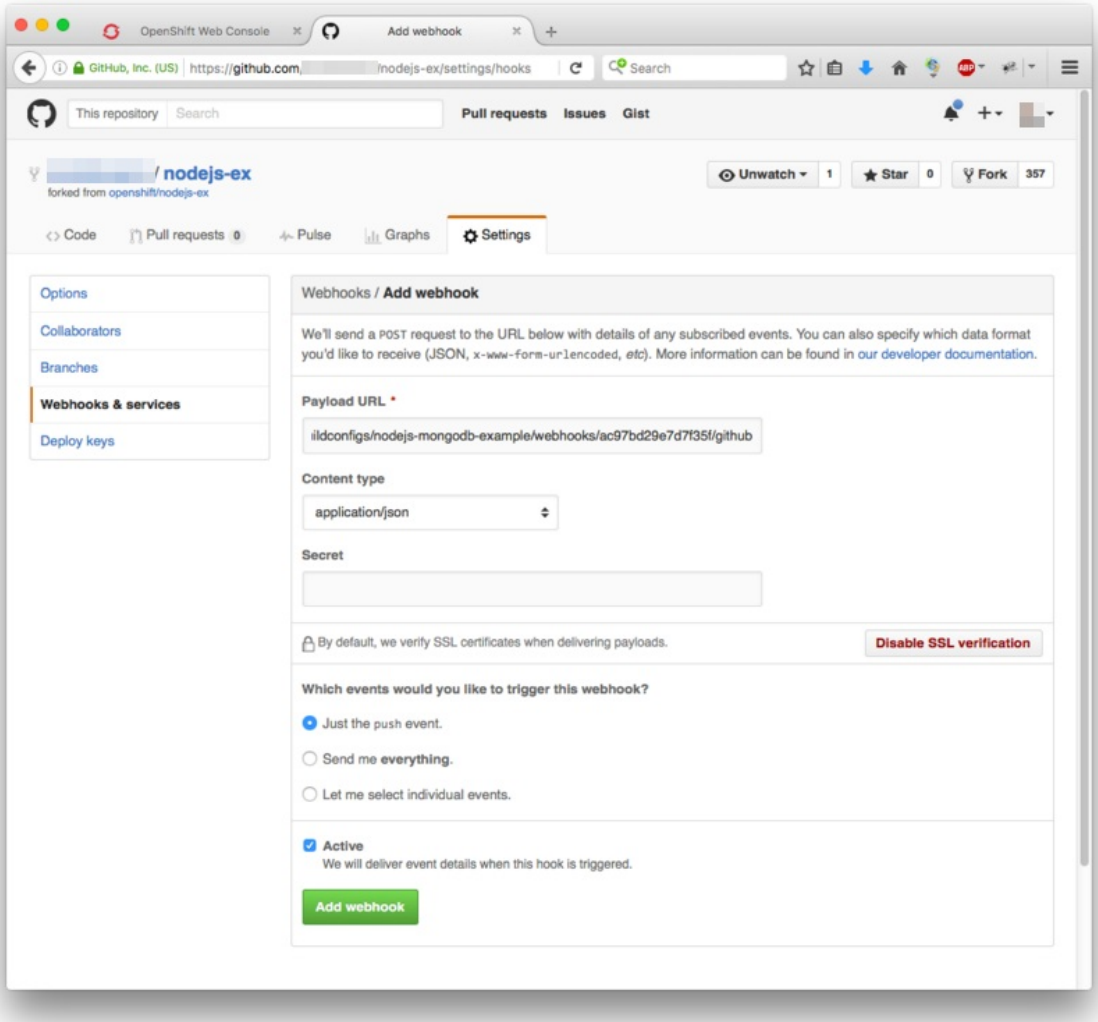
  a. Navigate to the project containing your application.

  b. Click the **Browse** tab, then click **Builds**, then click the name of the build for your Node.js application.

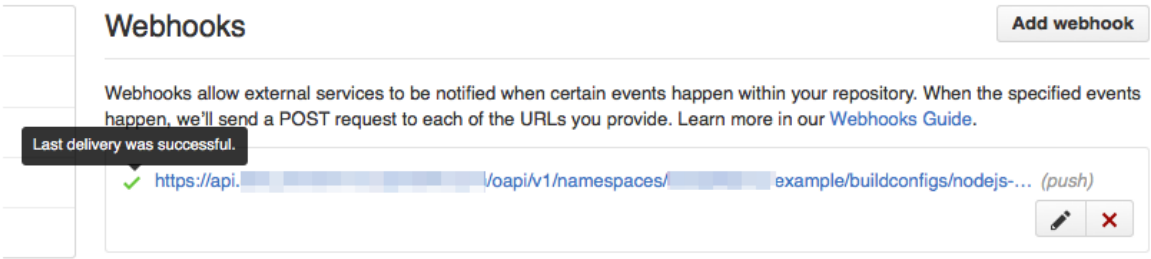
  c. From the **Configuration** tab, click  next to **GitHub webhook URL** to copy your GitHub webhook.

Next, add the webhook to the Github repository:

1. In GitHub, click **Add webhook** in the GitHub Webhook settings for your project. Paste the payload URL into the **Payload URL** field. Ensure **Content type** field is set to **application/json** instead of the default **application/x-www-form-urlencoded**. Then, click **Add webhook** to finish adding the webhook to your project:



2. GitHub now attempts to ping the OpenShift Online server to ensure that communication is successful. If it is correctly configured, you will see a green check mark next to your new webhook URL in GitHub. Hover your mouse over the check mark to see the status of the last delivery:

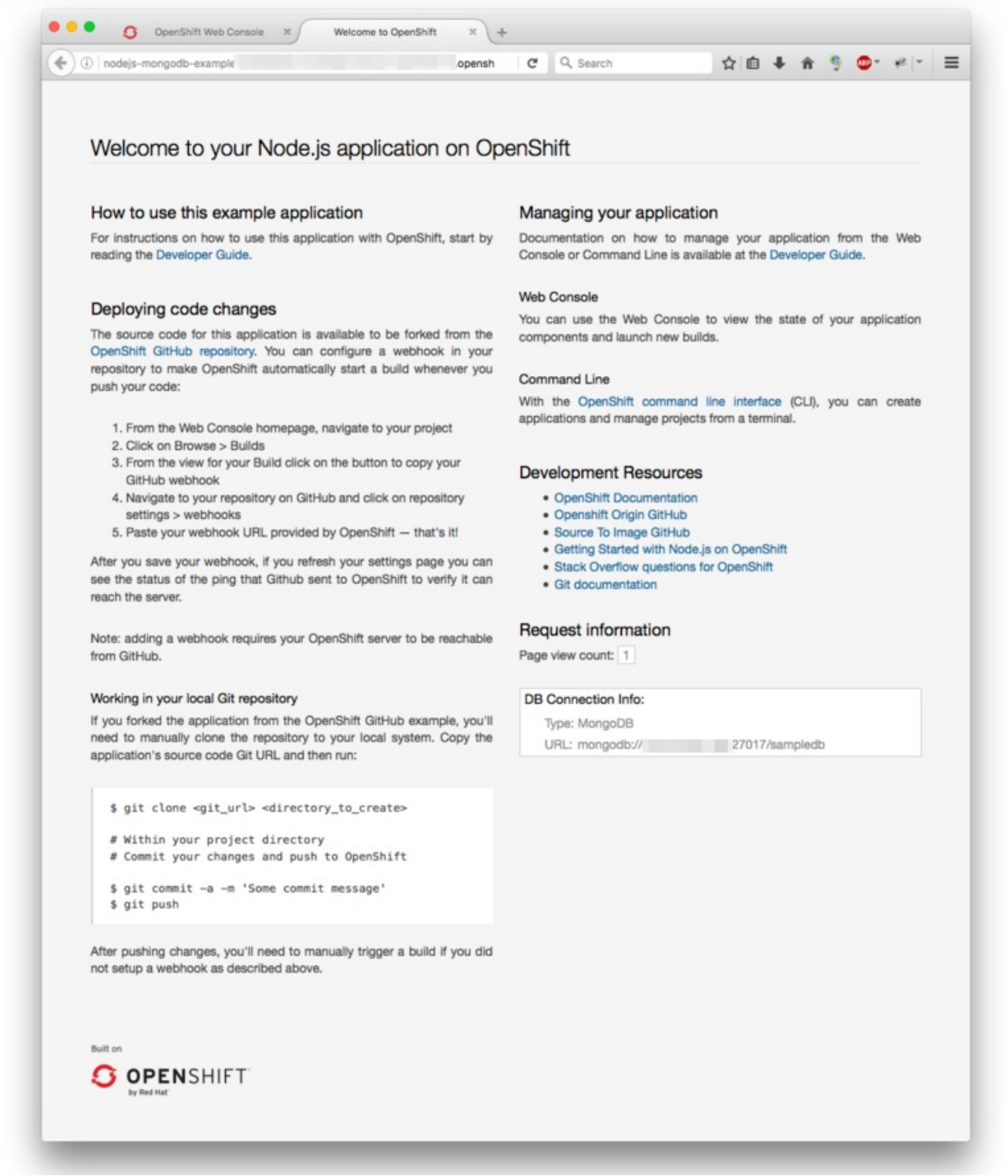


The next time you push a code change to your forked repository, your application will automatically rebuild.

4.6. VIEWING YOUR RUNNING APPLICATION

In this section, you will view your running application using a web browser.

In the web console, view the **Overview** page for your project to determine the web address for your application. Click the web address displayed underneath the **NODEJS-MONGODB-EXAMPLE** service to open your application in a new browser tab:



NOTE

You can find all routes configured for your project at any time in the web console:

1. From the web console, navigate to the project containing your application.
2. Click the **Browse** tab, then click **Routes**.
3. Click the host name to open your application in a browser new tab.

4.7. PUSHING A CODE CHANGE

In this section, you will learn how to push a local code change to the application.

1. On your local machine, use a text editor to open the sample application's source for the file ***nodejs-ex/views/index.html***.

2. Make a code change that will be visible from within your application. For example, change the title on line 219:

views/index.html

```
...  ...  @@ -216,7 +216,7 @@ pre {
216  216
217  217  <section class='container'>
218  218            <hgroup>
219       -            <h1>Welcome to your Node.js application on OpenShift</h1>
     219  +            <h1>This is my awesome Node.js application on OpenShift</h1>
220  220            </hgroup>
221  221
222  222
```

3. Commit the changes in Git, and push the change to your GitHub repository:

```
$ git add views/index.html
$ git commit -m "Updates heading on welcome page"
$ git push origin master
```

4. If your webhook is correctly configured, your application will immediately rebuild itself based on your changes. View your application using a web browser to see your changes.

Now going forward, all you need to do is push code updates and OpenShift Online handles the rest.

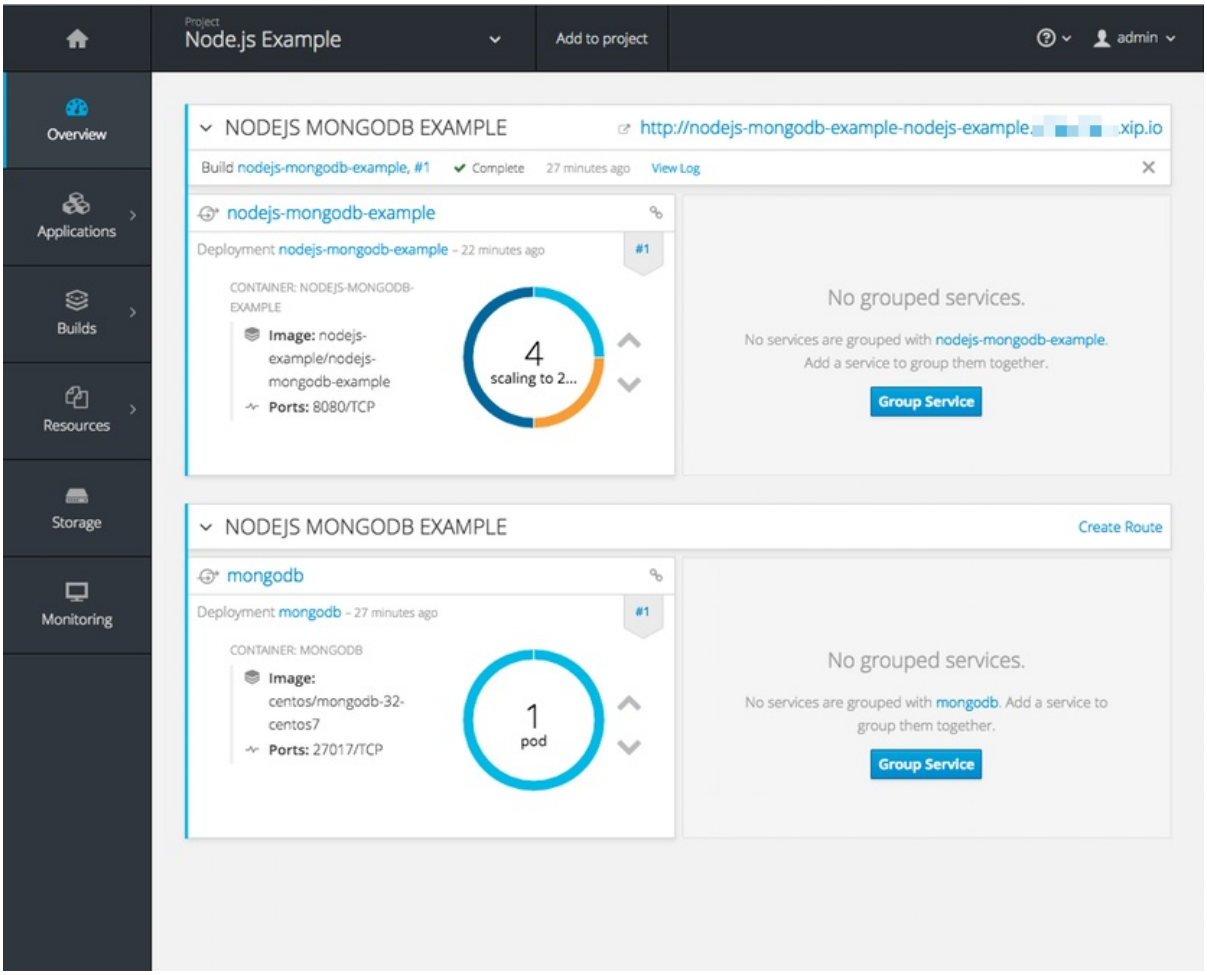
4.8. SCALING THE APPLICATION

IMPORTANT

OpenShift Online Starter users are unable to scale applications. Only OpenShift Online Pro users have this ability.

In this section, you will add additional instances of your Node.js service so that your application can handle additional traffic volume.

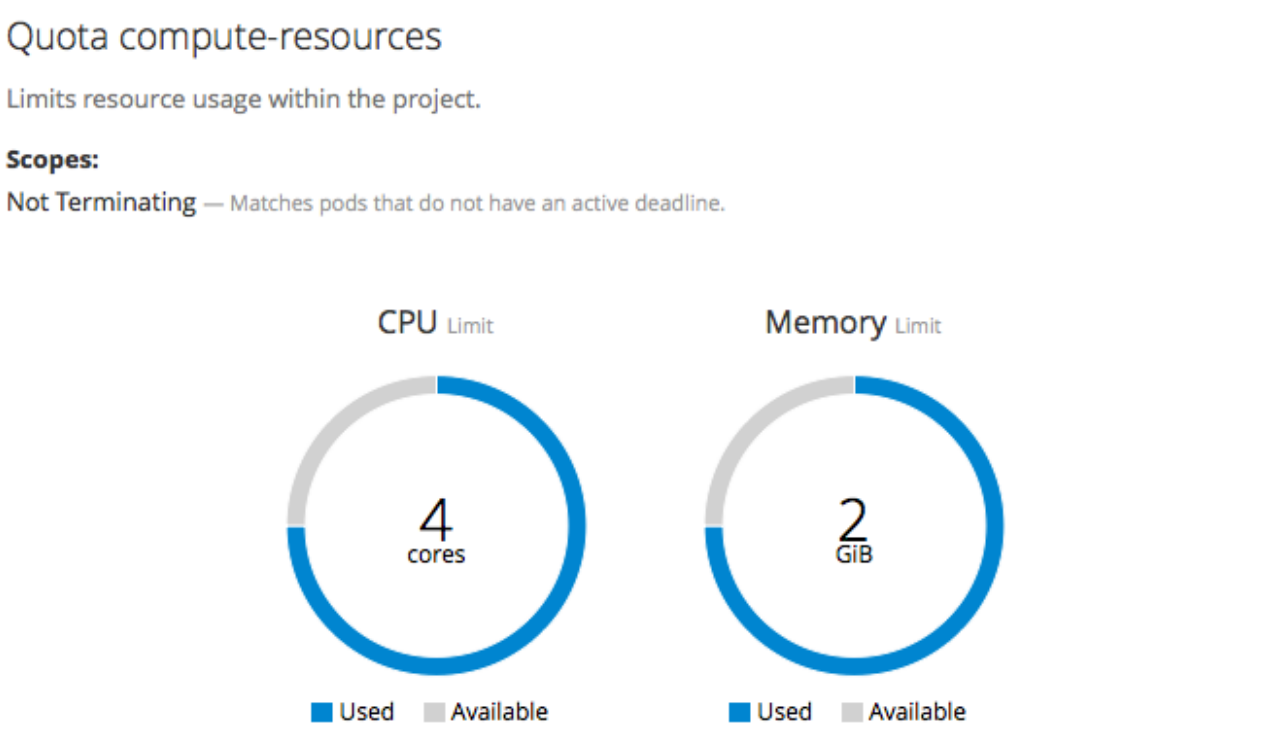
1. In the web console, view the **Overview** page for your project. Click the up arrow under the **NODEJS-MONGODB-EXAMPLE** service to add an additional replica of your Node.js application:



The **nodejs-mongodb-example** Quickstart is configured to use 512 MiB of memory per pod. Your OpenShift Online Pro quota will allow up to 3 replicas of the **nodejs-mongodb-example** pod in addition to the MongoDB database (for a total of 2 GiB).

You can check your quota usage at any time in the web console:

1. From the web console, navigate to the project containing your application.
2. Click the **Settings** tab and scroll to the section titled **Quota compute-resources** to view usage:



Resource type	Used	Max
CPU (Limit)	3 cores	4 cores
Memory (Limit)	1536 MiB	2 GiB

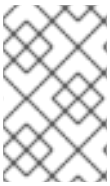
4.9. NEXT UP: BEYOND THE BASICS

Next, we'll go beyond the basics using the OpenShift Online CLI to compose this same application using individual images.

CHAPTER 5. BEYOND THE BASICS

5.1. OVERVIEW

This getting started experience walks you through the “long way” of getting the same sample project from the Basic Walkthrough topic up and running on OpenShift Online.



NOTE

If you are unfamiliar with the core concepts of OpenShift version 3, you might want to start by reading about what’s new. This version of OpenShift is significantly different from version 2.

The following sections guide you through creating a project that contains a sample Node.js application that will serve a welcome page and the current hit count (stored in a database). This involves creating two pods:

- one to host the Node.js application
- one to host the MongoDB database

The tutorial assumes that you have:

- a free OpenShift Online 3 account.
- a free GitHub account.
- Git installed locally.

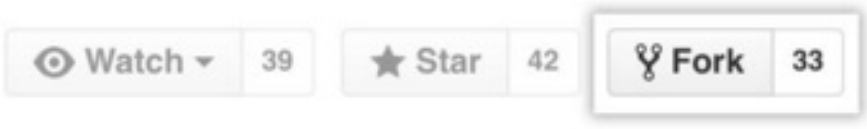
5.2. SETUP

In this section, you will *fork* the OpenShift Node.js sample application on GitHub and clone the repository to your local machine so that you can deploy and edit the app.

NOTE

You can skip this step if you already forked the **openshift/nodejs-ex** repository when following the Basic Walkthrough topic.

1. On GitHub, navigate to the **openshift/nodejs-ex** repository. In the top-right corner of the page, click **Fork**:



2. Next, execute the following commands on your local machine to clone the sample application and change to the new directory:

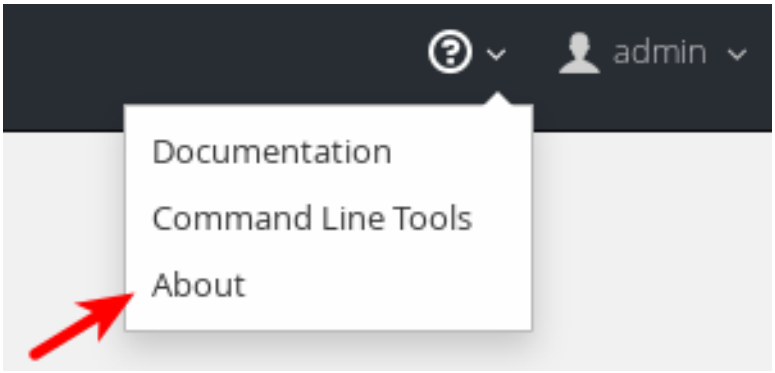
```
$ git clone https://github.com/<your_github_username>/nodejs-ex
$ cd nodejs-ex
```

That's it! Now, you have a fork of the original **openshift/nodejs-ex** example application Git repository and a copy on your local machine.

5.3. INSTALLING THE OPENSHIFT CLI

In this section, you will install the OpenShift CLI. The OpenShift CLI exposes commands for managing your applications, as well as lower level tools to interact with each component of your system.

1. First, download the OpenShift Online CLI from the **About** page in the OpenShift Online web console.



   The CLI is available for Linux (32- or 64-bit), Mac OS X, and Windows. After you have downloaded the CLI, return to these steps.

2. Next, unpack or unzip the archive and move the **oc** binary to a directory on your **PATH**.

NOTE

To check your **PATH** on Linux or Mac OS X, open the Terminal and run:

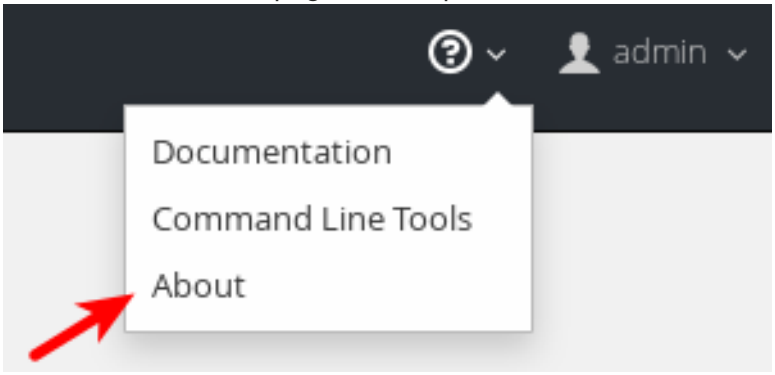
```
$ echo $PATH
```

To check it on Windows, open the Command Prompt and run:

```
C:\> path
```

   After it is installed, you can use the **oc** command from your command shell.

3. Then, visit the **About** page in the OpenShift Online web console.



4. Copy the **`oc login`** command shown with your current session token to log in to OpenShift Online from the CLI:

```
$ oc login https://<your_cluster_ID>.openshift.com --token=
<your_session_token>
```

The **oc login** command is the best way to initially set up the OpenShift Online CLI. The information is automatically saved in a CLI configuration file that is then used for subsequent commands.

5.4. CREATING A NEW APPLICATION FROM SOURCE CODE

In this section, you will deploy your first application to OpenShift Online using the web console.

1. First, create a new project. Replace **<project_name>** below with a unique name for your project, such as **<your_github_username>-example**:

```
$ oc new-project <project_name>
```

After creating the new project, you will be automatically switched to the new project namespace.

If you followed the Basic Walkthrough topic, you already created your first project. You must switch to your project namespace and clear out the original sample application.

a. Use the following command to find the name of your existing project(s):

```
$ oc get projects
```

b. Next, switch to your project namespace:

```
$ oc project <your_project_name>
```

c. Then, delete all existing objects in your project:

```
$ oc delete all --all
```

d. Use the following command to find the name of your existing persistent volume claims:

```
$ oc get pvc
```

e. Finally, delete your existing persistent volume claims with:

```
$ oc delete pvc mongodb
```

2. Next, create a new application from your forked copy of the **nodejs-ex** source code file:

```
$ oc new-app https://github.com/<your_github_username>/nodejs-ex --
name nodejs-mongodb-example
```

NOTE

The **--name** option will apply a name of **nodejs-mongodb-example** to all the resources created by the **oc new-app** command, for easy management later.

The tool will inspect the source code, locate an appropriate image that can build the source code, create an image stream for the new application image that will be built, then create the correct build configuration, deployment configuration and service definition.

The **`oc new-app`** command kicks off a build after all required dependencies are confirmed and automatically deploys the application after the image is available.

TIP

You can follow along on the **Overview** page for your project in the web console to see the new resource being created and watch the progress of the build and deployment. When the Node.js pod is running, the build is complete.

You can also use the **`oc status`** command to check the status of your new nodejs app, as well as **`oc get pods`** to check when the pod is up and running.

The **`oc get services`** command tells you what IP address the service is running; the default port it deploys to is **`8080`**.

5.5. CONFIGURING ROUTES

In this section, you will configure a route to expose your Node.js service to external requests.

1. First, find your service name (which should be **`nodejs-mongodb-example`** with:

```
$ oc get services
```

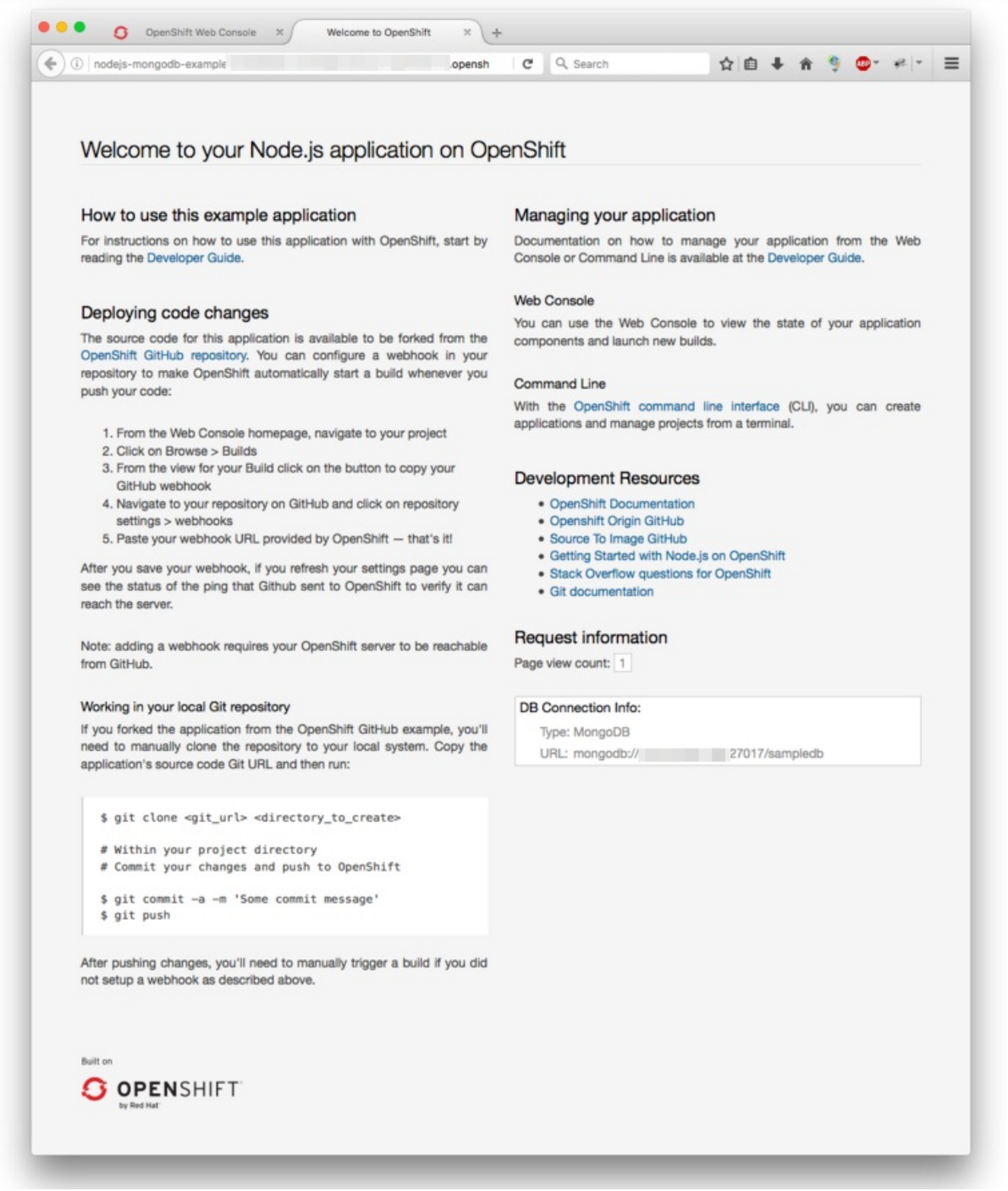
2. Next, create a route to expose your service to external requests:

```
$ oc expose service/nodejs-mongodb-example
```

3. Now you can find the external host/port for your service with:

```
$ oc get routes
```

4. Finally, copy the route **HOST/PORT** for your application and paste it in the browser to view your application:



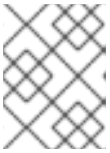
5.6. PROVISIONING A DATABASE

In this section, you will add a MongoDB service to your project.

You may have noticed the **`No database configured`** under **Request information** when you viewed the index page of your application. Let's fix that by adding a MongoDB service.

1. Add the OpenShift Online-provided MongoDB database to your project with:

```
$ oc new-app mongodb-persistent \
-p MONGODB_USER=admin \
-p MONGODB_PASSWORD=secret \
-p MONGODB_ADMIN_PASSWORD=super-secret
```



NOTE

The **-p** flag sets the parameter values used by the **mongodb-persistent** database template.

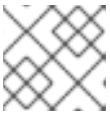
2. Note the name of the MongoDB service before heading to the next section.

5.7. SETTING ENVIRONMENT VARIABLES

In this section, you will configure the Node.js service to connect to your new MongoDB service.

1. You must add the environment variable **MONGO_URL** to your Node.js web service so that it will utilize the MongoDB service, and enable the "Page view count" feature. Run:

```
$ oc set env dc/nodejs-mongodb-example
MONGO_URL='mongodb://admin:secret@<MongoDB-service-
name>:27017/sampledb'
```



NOTE

Services in the same project have automatically resolvable DNS names.

For example:

```
$ oc set env dc/nodejs-mongodb-example
MONGO_URL='mongodb://admin:secret@mongodb-persistent:27017/sampledb'
```

2. Next, run **oc status** to confirm that an updated deployment has been kicked off. After the deployment completes, you will now have a Node.js welcome page showing the current hit count, as stored in a MongoDB database.

NOTE

Use the following to get a list of environment variables set for all pods in the project:

```
$ oc set env pods --all --list
```

5.8. CONFIGURING AUTOMATED BUILDS

In this section, you will configure a GitHub webhook to automatically trigger a rebuild of your application whenever you push code changes to your forked repository.

1. First, run the following command to display the webhook URLs associated with your build configuration:

```
$ oc describe buildConfig nodejs-mongodb-example
```

2. Copy the webhook GitHub URL output by the above command. The webhook URL will be in the following format:

```
http://<openshift_api_host:port>/osapi/v1/namespaces/<namespace>/buildconfigs/frontend/webhooks/<your_secret_key>/github
```

3. Next, navigate to your forked repository on GitHub, then:
   a. Click **Settings**.
   b. Click **Webhooks & Services**.
   c. Click **Add webhook**
   d. Paste your webhook URL into the **Payload URL** field and click **Add webhook** to save.

That's it! Your application will now automatically rebuild when you push code changes to your forked GitHub repository.

5.9. PUSHING A CODE CHANGE

In this section, you will learn how to push a local code change to the application.

1. On your local machine, use a text editor to open the sample application's source for the file ***nodejs-ex/views/index.html***.
2. Make a code change that will be visible from within your application. For example, change the title on line 219:

views/index.html

```
...  ...  @@ -216,7 +216,7 @@ pre {
216  216
217  217  <section class='container'>
218  218            <hgroup>
219       -            <h1>Welcome to your Node.js application on OpenShift</h1>
     219  +            <h1>This is my awesome Node.js application on OpenShift</h1>
220  220            </hgroup>
221  221
222  222
```

3. Commit the changes in Git, and push the change to your GitHub repository:

```
$ git add nodejs-ex/views/index.html
$ git commit -m "Updates heading on welcome page"
$ git push origin master
```

4. If your webhook is correctly configured, your application will immediately rebuild itself based on your changes. You can follow along on the **Overview** page for your project in the web console to see watch the progress of the build and deployment. View your application using a web browser to see your changes once the deployment is completed.

Now all you need to do is push code updates, and OpenShift Online handles the rest.

5.10. FAILURE NOTIFICATIONS

For each of your projects, you can choose to receive email notifications about various failures, including dead or failed deployments, dead builds, and dead or failed persistent volume claims (PVCs).

5.11. WHAT'S NEXT?

The following sections provide some next steps now that you have finished your initial walkthrough of OpenShift Online 3.

5.11.1. OpenShift Online Usage Considerations

5.11.2. Other Quickstarts

Similar to OpenShift Online 2, OpenShift Online 3 provides out of the box a set of languages and databases for developers with corresponding implementations and tutorials that allow you to kickstart your application development. Language support centers around the Quickstart templates, which in turn leverage builder images.

Check out the Creating New Applications topic and try out Quickstart templates for the following languages:

Language	Implementations and Tutorials
Ruby	Rails
Python	Django
Node.js	Node.js
PHP	CakePHP
Perl	Dancer
Java	Maven

Other images provided by OpenShift Online include:

- MySQL
- MongoDB
- PostgreSQL
- Jenkins

In addition, JBoss Middleware has put together a broad range of OpenShift Online templates as well as images as part of their xPaaS services.

The technologies available with the xPaaS services in particular include:

- Java EE 6 Application Server provided by JBoss EAP 6
- Integration and Messaging Services provided by JBoss Fuse and JBoss A-MQ

- Data Grid Service provided by JBoss Data Grid
- Real Time Decision Service provided by JBoss BRMS
- Java Web Server 3.0 provided by Tomcat 7 and Tomcat 8

With each of these offerings, a series of combinations are provided:

- HTTP only versus HTTP and HTTPS
- No database required, or the use of either MongoDB, PostgreSQL, or MySQL
- If desired, integration with A-MQ

5.11.3. Using rsync

See Copying Files for steps on using **`oc rsync`** to copy local files to or from a remote directory in a container.

5.11.4. Configuring Autoscaling

See Pod Autoscaling for steps on automatically increasing or decreasing the scale of a replication controller or deployment configuration, based on metrics.

You can also check out the OpenShift blog for an article on autoscaling.

5.11.5. Explore the Developer Guide

Further explore the Developer Guide. For example, start with the Planning Your Development Process and Creating New Applications topics.

5.11.6. Troubleshooting

Review common tips and suggestions.

CHAPTER 6. COMPARING OPENSHIFT ONLINE 2 AND 3

6.1. OVERVIEW

OpenShift Online 3 is based on the OpenShift version 3 (v3) architecture, which is very different product than OpenShift version 2 (v2). Many of the same terms from OpenShift v2 are used in v3, and the same functions are performed, but the terminology can be different, and behind the scenes things may be happening very differently. Still, OpenShift remains an application platform.

This topic discusses these differences in detail, in an effort to help OpenShift users in the transition from OpenShift v2 to OpenShift v3.

6.2. ARCHITECTURE CHANGES

Gears Versus Containers

Gears were a core component of OpenShift v2. Technologies such as kernel namespaces, cGroups, and SELinux helped deliver a highly-scalable, secure, containerized application platform to OpenShift users. Gears themselves were a form of container technology.

OpenShift v3 takes the gears idea to the next level. It uses Docker as the next evolution of the v2 container technology. This container architecture is at the core of OpenShift v3.

Kubernetes

As applications in OpenShift v2 typically used multiple gears, applications on OpenShift v3 will expectedly use multiple containers. In OpenShift v2, gear orchestration, scheduling, and placement was handled by the OpenShift broker host. OpenShift v3 integrates Kubernetes into the master host to drive container orchestration.

6.3. APPLICATIONS

Applications are still the focal point of OpenShift. In OpenShift v2, an application was a single unit, consisting of one web framework of no more than one cartridge type. For example, an application could have one PHP and one MySQL, but it could not have one Ruby, one PHP, and two MySQLs. It also could not be a database cartridge, such as MySQL, by itself.

This limited scoping for applications meant that OpenShift performed seamless linking for all components within an application using environment variables. For example, every web framework knew how to connect to MySQL using the **`OPENSHIFT_MYSQL_DB_HOST`** and **`OPENSHIFT_MYSQL_DB_PORT`** variables. However, this linking was limited to within an application, and only worked within cartridges designed to work together. There was nothing to help link across application components, such as sharing a MySQL instance across two applications.

While most other PaaSes limit themselves to web frameworks and rely on external services for other types of components, OpenShift v3 makes even more application topologies possible and manageable.

OpenShift v3 uses the term "application" as a concept that links services together. You can have as many components as you desire, contained and flexibly linked within a project, and, optionally, labeled to provide grouping or structure. This updated model allows for a standalone MySQL instance, or one shared between JBoss components.

Flexible linking means you can link any two arbitrary components together. As long as one component can export environment variables and the second component can consume values from those

environment variables, and with potential variable name transformation, you can link together any two components without having to change the images they are based on. So, the best containerized implementation of your desired database and web framework can be consumed directly rather than you having to fork them both and rework them to be compatible.

This means you can build anything on OpenShift. And that is OpenShift's primary aim: to be a container-based platform that lets you build entire applications in a repeatable lifecycle.

6.4. CARTRIDGES VERSUS IMAGES

In OpenShift v3, an image has replaced OpenShift v2's concept of a cartridge.

Cartridges in OpenShift v2 were the focal point for building applications. Each cartridge provided the required libraries, source code, build mechanisms, connection logic, and routing logic along with a preconfigured environment to run the components of your applications.

However, cartridges came with disadvantages. With cartridges, there was no clear distinction between the developer content and the cartridge content, and you did not have ownership of the home directory on each gear of your application. Also, cartridges were not the best distribution mechanism for large binaries. While you could use external dependencies from within cartridges, doing so would lose the benefits of encapsulation.

From a packaging perspective, an image performs more tasks than a cartridge, and provides better encapsulation and flexibility. However, cartridges also included logic for building, deploying, and routing, which do not exist in images. In OpenShift v3, these additional needs are met by Source-to-Image (S2I) and configuring the template.

Dependencies

In OpenShift v2, cartridge dependencies were defined with **`Configure-Order`** or **`Requires`** in a cartridge manifest. OpenShift v3 uses a declarative model where pods bring themselves in line with a predefined state. Explicit dependencies that are applied are done at runtime rather than just install time ordering.

For example, you might require another service to be available before you start. Such a dependency check is always applicable and not just when you create the two components. Thus, pushing dependency checks into runtime enables the system to stay healthy over time.

Collection

Whereas cartridges in OpenShift v2 were colocated within gears, images in OpenShift v3 are mapped 1:1 with containers, which use pods as their colocation mechanism.

Source Code

In OpenShift v2, applications were required to have at least one web framework with one Git repository. In OpenShift v3, you can choose which images are built from source and that source can be located outside of OpenShift itself. Because the source is disconnected from the images, the choice of image and source are distinct operations with source being optional.

Build

In OpenShift v2, builds occurred in application gears. This meant downtime for non-scaled applications due to resource constraints. In v3, builds happen in separate containers. Also, OpenShift v2 build results used rsync to synchronize gears. In v3, build results are first committed as an immutable image and

published to an internal registry. That image is then available to launch on any of the nodes in the cluster, or available to rollback to at a future date.

Routing

In OpenShift v2, you had to choose up front as to whether your application was scalable, and whether the routing layer for your application was enabled for high availability (HA). In OpenShift v3, routes are first-class objects that are HA-capable simply by scaling up your application component to two or more replicas. There is never a need to recreate your application or change its DNS entry.

The routes themselves are disconnected from images. Previously, cartridges defined a default set of routes and you could add additional aliases to your applications. With OpenShift v3, you can use templates to set up any number of routes for an image. These routes let you modify the scheme, host, and paths exposed as desired, with no distinction between system routes and user aliases.

6.5. BROKER VERSUS MASTER

A master in OpenShift v3 is similar to a broker host in OpenShift v2. However, the MongoDB and ActiveMQ layers used by the broker in OpenShift v2 are no longer necessary, because **etcd** is typically installed with each master host.

6.6. DOMAIN VERSUS PROJECT

A project is essentially a v2 domain.