Red Hat OpenShift Application Runtimes 1

Node.js Runtime Guide

For Use with Red Hat OpenShift Application Runtimes

Last Updated: 2019-01-16

Red Hat OpenShift Application Runtimes 1 Node.js Runtime Guide

For Use with Red Hat OpenShift Application Runtimes

Legal Notice

Abstract

This guide provides details on using the {NodeJS} runtime with Red Hat OpenShift Application Runtimes.

Table of Contents

PREFACE

This guide covers concepts as well as practical details needed by developers to use the Node.js runtime.

CHAPTER 1. WHAT IS NODE.JS

Node.js is based on the V8 JavaScript engine from Google and allows you to write server-side JavaScript applications. It provides an I/O model based on events and non-blocking operations that enables you to write efficient applications. Node.js also provides a large module ecosystem called npm. Check out Additional Resources for further reading on Node.js.

The Node.js runtime enables you to run Node.js applications and services on OpenShift while providing all the advantages and conveniences of the OpenShift platform such as rolling updates, continuous delivery pipelines, service discovery, and canary deployments. OpenShift also makes it easier for your applications to implement common microservice patterns such as externalized configuration, health check, circuit breaker, and failover.

CHAPTER 2. MISSIONS AND CLOUD-NATIVE DEVELOPMENT ON OPENSHIFT

When developing applications on OpenShift, you can use missions and boosters to kickstart your development.

Missions

Missions are working applications that showcase different fundamental pieces of building cloud native applications and services.

A mission implements a Microservice pattern such as:

- Creating REST APIs
- Interoperating with a database
- Implementing the Health Check pattern

You can use missions for a variety of purposes:

- A proof of technology demonstration
- A teaching tool, or a sandbox for understanding how to develop applications for your project
- They can also be updated or extended for your own use case

Boosters

A booster is the implementation of a mission in a specific runtime. Boosters are preconfigured, functioning applications that demonstrate core principles of modern application development and run in an environment similar to production.

Each mission is implemented in one or more runtimes. Both the specific implementation and the actual project that contains your code are called a booster.

For example, the REST API Level 0 mission is implemented for these runtimes:

- Node.js booster
- Spring Boot booster
- Eclipse Vert.x booster
- Thorntail booster

CHAPTER 3. AVAILABLE MISSIONS AND BOOSTERS FOR NODE.JS

The following boosters are available for Node.js.

3.1. REST API LEVEL 0 MISSION - NODE.JS BOOSTER

Mission proficiency level: **Foundational**.

What the REST API Level 0 Mission Does

The REST API Level 0 mission shows how to map business operations to a remote procedure call endpoint over HTTP using a REST framework. This corresponds to Level 0 in the Richardson Maturity Model. Creating an HTTP endpoint using REST and its underlying principles to define your API lets you quickly prototype and design the API flexibly.

This booster introduces the mechanics of interacting with a remote service using the HTTP protocol. It allows you to:

- Execute an HTTP **`GET`** request on the **`api/greeting`** endpoint.
- Receive a response in JSON format with a payload consisting of the **`Hello, World!`** String.
- Execute an HTTP **`GET`** request on the **`api/greeting`** endpoint while passing in a String argument. This uses the **`name`** request parameter in the query string.
- Receive a response in JSON format with a payload of **`Hello, $name!`** with **`$name`** replaced by the value of the **`name`** parameter passed into the request.

3.1.1. Viewing the booster source code and README

Prerequisites

One of the following:

- Access to developers.redhat.com/launch
- Fabric8 Launcher installed on a Single-node OpenShift Cluster

Procedure

1. Use the Fabric8 Launcher tool to generate your own version of the booster.
2. View the generated GitHub repository or download and extract the ZIP file that contains the booster source code.

Additional resources

- Using developers.redhat.com/launch
- Using the Fabric8 Launcher tool on a Single-node OpenShift Cluster

3.1.2. REST API Level 0 design tradeoffs

Table 3.1. Design Tradeoffs

Pros	Cons

• The booster enables fast prototyping.
• The API Design is flexible.
• HTTP endpoints allow clients to be language-neutral.
	• As an application or service matures, the REST API Level 0 approach might not scale well. It might not support a clean API design or use cases with database interactions.
	  ◦ Any operations involving shared, mutable state must be integrated with an appropriate backing datastore.
	  ◦ All requests handled by this API design are scoped only to the container servicing the request. Subsequent requests might not be served by the same container.

3.1.3. Deploying the REST API Level 0 booster to OpenShift Online

Use one of the following options to execute the REST API Level 0 booster on OpenShift Online.

- Use developers.redhat.com/launch
- Use the **oc** CLI client

Although each method uses the same **oc** commands to deploy your application, using developers.redhat.com/launch provides an automated booster deployment workflow that executes the **oc** commands for you.

3.1.3.1. Deploying the booster using developers.redhat.com/launch

Prerequisites

- An account at OpenShift Online.

Procedure

1. Navigate to the developers.redhat.com/launch URL in a browser and log in.
2. Follow on-screen instructions to create and launch your booster in Node.js.

3.1.3.2. Authenticating the oc CLI client

To work with boosters on OpenShift Online using the **oc** command-line client, you need to authenticate the client using the token provided by the OpenShift Online web interface.

Prerequisites

- An account at OpenShift Online.

Procedure

1. Navigate to the OpenShift Online URL in a browser.
2. Click on the question mark icon in the top right-hand corner of the Web console, next to your user name.
3. Select *Command Line Tools* in the drop-down menu.
4. Find the text box that contains the **`oc login …`** command with the hidden token, and click the button next to it to copy its content to your clipboard.
5. Paste the command into a terminal application. The command uses your authentication token to authenticate your **oc** CLI client with your OpenShift Online account.

```
$ oc login OPENSHIFT_URL --token=MYTOKEN
```

3.1.3.3. Deploying the REST API Level 0 booster using the `oc` CLI client

Prerequisites

- The booster application created using developers.redhat.com/launch. For more information, see Section 3.1.3.1, “Deploying the booster using developers.redhat.com/launch”.
- The **oc** client authenticated. For more information, see Section 3.1.3.2, “Authenticating the **oc** CLI client”.

Procedure

1. Clone your project from GitHub.

```
$ git clone git@github.com:USERNAME/MY_PROJECT_NAME.git
```

   Alternatively, if you downloaded a ZIP file of your project, extract it.

```
$ unzip MY_PROJECT_NAME.zip
```

2. Create a new project in OpenShift.

```
$ oc new-project MY_PROJECT_NAME
```

3. Navigate to the root directory of your booster.
4. Use **`npm`** to start the deployment to OpenShift.

```
$ npm install && npm run openshift
```

   These commands install any missing module dependencies, then using the Nodeshift module, deploy the booster on OpenShift.

5. Check the status of your booster and ensure your pod is running.

```
$ oc get pods -w
NAME                                READY     STATUS        RESTARTS
AGE
```

```
MY_APP_NAME-1-aaaaa               1/1       Running     0
58s
MY_APP_NAME-s2i-1-build           0/1       Completed   0
2m
```

The **MY_APP_NAME-1-aaaaa** pod should have a status of **Running** once it is fully deployed and started. Your specific pod name will vary. The number in the middle will increase with each new build. The letters at the end are generated when the pod is created.

6. Once your booster is deployed and started, determine its route.

Example Route Information

```
$ oc get routes
NAME                  HOST/PORT
PATH      SERVICES         PORT      TERMINATION
MY_APP_NAME          MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME
MY_APP_NAME      8080
```

The route information of a pod gives you the base URL which you use to access it. In the example above, you would use **http://MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME** as the base URL to access the application.

3.1.4. Deploying the REST API Level 0 booster to Single-node OpenShift Cluster

Use one of the following options to execute the REST API Level 0 booster locally on Single-node OpenShift Cluster:

- Using Fabric8 Launcher
- Using the **oc** CLI client

Although each method uses the same **oc** commands to deploy your application, using Fabric8 Launcher provides an automated booster deployment workflow that executes the **oc** commands for you.

3.1.4.1. Getting the Fabric8 Launcher tool URL and credentials

You need the Fabric8 Launcher tool URL and user credentials to create and deploy boosters on Single-node OpenShift Cluster. This information is provided when the Single-node OpenShift Cluster is started.

Prerequisites

- The Fabric8 Launcher tool installed, configured, and running. For more information, see the Install and Configure the Fabric8 Launcher Tool guide.

Procedure

1. Navigate to the console where you started Single-node OpenShift Cluster.
2. Check the console output for the URL and user credentials you can use to access the running Fabric8 Launcher:

Example Console Output from a Single-node OpenShift Cluster Startup

```
...
-- Removing temporary directory ... OK
-- Server Information ...
   OpenShift server started.
   The server is accessible via web console at:
       https://192.168.42.152:8443

   You are logged in as:
       User:     developer
       Password: developer

   To login as administrator:
       oc login -u system:admin
```

3.1.4.2. Deploying the booster using the Fabric8 Launcher tool

Prerequisites

- The URL of your running Fabric8 Launcher instance and the user credentials of your Single-node OpenShift Cluster. For more information, see Section 3.1.4.1, “Getting the Fabric8 Launcher tool URL and credentials”.

Procedure

1. Navigate to the Fabric8 Launcher URL in a browser.
2. Follow on-screen instructions to create and launch your booster in Node.js.

3.1.4.3. Authenticating the oc CLI client

To work with boosters on Single-node OpenShift Cluster using the **oc** command-line client, you need to authenticate the client using the token provided by the Single-node OpenShift Cluster web interface.

Prerequisites

- The URL of your running Fabric8 Launcher instance and the user credentials of your Single-node OpenShift Cluster. For more information, see Section 3.1.4.1, “Getting the Fabric8 Launcher tool URL and credentials”.

Procedure

1. Navigate to the Single-node OpenShift Cluster URL in a browser.
2. Click on the question mark icon in the top right-hand corner of the Web console, next to your user name.
3. Select *Command Line Tools* in the drop-down menu.
4. Find the text box that contains the **`oc login …`** command with the hidden token, and click the button next to it to copy its content to your clipboard.
5. Paste the command into a terminal application. The command uses your authentication token to authenticate your **oc** CLI client with your Single-node OpenShift Cluster account.

```
$ oc login OPENSHIFT_URL --token=MYTOKEN
```

3.1.4.4. Deploying the REST API Level 0 booster using the `oc` CLI client

Prerequisites

- The booster application created using Fabric8 Launcher tool on a Single-node OpenShift Cluster. For more information, see Section 3.1.4.2, “Deploying the booster using the Fabric8 Launcher tool”.
- Your Fabric8 Launcher tool URL.
- The **oc** client authenticated. For more information, see Section 3.1.4.3, “Authenticating the **oc** CLI client”.

Procedure

1. Clone your project from GitHub.

   ```
   $ git clone git@github.com:USERNAME/MY_PROJECT_NAME.git
   ```

   Alternatively, if you downloaded a ZIP file of your project, extract it.

   ```
   $ unzip MY_PROJECT_NAME.zip
   ```

2. Create a new project in OpenShift.

   ```
   $ oc new-project MY_PROJECT_NAME
   ```

3. Navigate to the root directory of your booster.

4. Use **npm** to start the deployment to OpenShift.

   ```
   $ npm install && npm run openshift
   ```

   These commands install any missing module dependencies, then using the Nodeshift module, deploy the booster on OpenShift.

5. Check the status of your booster and ensure your pod is running.

   ```
   $ oc get pods -w
   NAME                              READY     STATUS      RESTARTS
   AGE
   MY_APP_NAME-1-aaaaa               1/1       Running     0
   58s
   MY_APP_NAME-s2i-1-build           0/1       Completed   0
   2m
   ```

   The **`MY_APP_NAME-1-aaaaa`** pod should have a status of **`Running`** once it is fully deployed and started. Your specific pod name will vary. The number in the middle will increase with each new build. The letters at the end are generated when the pod is created.

6. Once your booster is deployed and started, determine its route.

Example Route Information

```
$ oc get routes
NAME                  HOST/PORT
PATH      SERVICES         PORT      TERMINATION
MY_APP_NAME          MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME
MY_APP_NAME      8080
```

The route information of a pod gives you the base URL which you use to access it. In the example above, you would use **`http://MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME`** as the base URL to access the application.

3.1.5. Deploying the REST API Level 0 booster to OpenShift Container Platform

The process of creating and deploying boosters to OpenShift Container Platform is similar to OpenShift Online:

Prerequisites

- The booster created using developers.redhat.com/launch or the Fabric8 Launcher tool.

Procedure

- Follow the instructions in Section 3.1.3, “Deploying the REST API Level 0 booster to OpenShift Online”, only use the URL and user credentials from the OpenShift Container Platform Web Console.

3.1.6. Interacting with the unmodified REST API Level 0 booster for Node.js

The booster provides a default HTTP endpoint that accepts GET requests.

Prerequisites

- Your application running
- The **`curl`** binary or a web browser

Procedure

1. Use **`curl`** to execute a **`GET`** request against the booster. You can also use a browser to do this.

```
$ curl http://MY_APP_NAME-
MY_PROJECT_NAME.OPENSHIFT_HOSTNAME/api/greeting
{"content":"Hello, World!"}
```

2. Use **`curl`** to execute a **`GET`** request with the **`name`** URL parameter against the booster. You can also use a browser to do this.

```
$ curl http://MY_APP_NAME-
MY_PROJECT_NAME.OPENSHIFT_HOSTNAME/api/greeting?name=Sarah
{"content":"Hello, Sarah!"}
```

NOTE

From a browser, you can also use a form provided by the booster to perform these same interactions. The form is located at the root of the project **`http://MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME`**.

3.1.7. REST resources

More background and related information on REST can be found here:

- Architectural Styles and the Design of Network-based Software Architectures - Representational State Transfer (REST)
- Richardson Maturity Model
- Express Web Framework
- REST API Level 0 mission - Spring Boot booster
- REST API Level 0 mission - Eclipse Vert.x booster
- REST API Level 0 mission - Thorntail booster

3.2. EXTERNALIZED CONFIGURATION MISSION - NODE.JS BOOSTER

Mission proficiency level: **Foundational**.

The Externalized Configuration mission provides a basic example of using a ConfigMap to externalize configuration. *ConfigMap* is an object used by OpenShift to inject configuration data as simple key and value pairs into one or more Linux containers while keeping the containers independent of OpenShift.

This mission shows you how to:

- Set up and configure a **`ConfigMap`**.
- Use the configuration provided by the **`ConfigMap`** within an application.
- Deploy changes to the **`ConfigMap`** configuration of running applications.

3.2.1. The externalized configuration design pattern

Whenever possible, externalize the application configuration and separate it from the application code. This allows the application configuration to change as it moves through different environments, but leaves the code unchanged. Externalizing the configuration also keeps sensitive or internal information out of your code base and version control. Many languages and application servers provide environment variables to support externalizing an application's configuration.

Microservices architectures and multi-language (polyglot) environments add a layer of complexity to managing an application's configuration. Applications consist of independent, distributed services, and each can have its own configuration. Keeping all configuration data synchronized and accessible creates a maintenance challenge.

ConfigMaps enable the application configuration to be externalized and used in individual Linux containers and pods on OpenShift. You can create a ConfigMap object in a variety of ways, including using a YAML file, and inject it into the Linux container. ConfigMaps also allow you to group and scale

sets of configuration data. This lets you configure a large number of environments beyond the basic *Development*, *Stage*, and *Production*. You can find more information about ConfigMaps in the OpenShift documentation.

3.2.2. Externalized Configuration design tradeoffs

Table 3.2. Design Tradeoffs

Pros	Cons
• Configuration is separate from deployments • Can be updated independently • Can be shared across services	• Adding configuration to environment requires additional step • Has to be maintained separately • Requires coordination beyond the scope of a service

3.2.3. Viewing the booster source code and README

Prerequisites

One of the following:

- Access to developers.redhat.com/launch
- Fabric8 Launcher installed on a Single-node OpenShift Cluster

Procedure

1. Use the Fabric8 Launcher tool to generate your own version of the booster.
2. View the generated GitHub repository or download and extract the ZIP file that contains the booster source code.

Additional resources

- Using developers.redhat.com/launch
- Using the Fabric8 Launcher tool on a Single-node OpenShift Cluster

3.2.4. Deploying the Externalized Configuration booster to OpenShift Online

Use one of the following options to execute the Externalized Configuration booster on OpenShift Online.

- Use developers.redhat.com/launch
- Use the **oc** CLI client

Although each method uses the same **oc** commands to deploy your application, using developers.redhat.com/launch provides an automated booster deployment workflow that executes the **oc** commands for you.

3.2.4.1. Deploying the booster using developers.redhat.com/launch

Prerequisites

- An account at OpenShift Online.

Procedure

1. Navigate to the developers.redhat.com/launch URL in a browser and log in.
2. Follow on-screen instructions to create and launch your booster in Node.js.

3.2.4.2. Authenticating the `oc` CLI client

To work with boosters on OpenShift Online using the **oc** command-line client, you need to authenticate the client using the token provided by the OpenShift Online web interface.

Prerequisites

- An account at OpenShift Online.

Procedure

1. Navigate to the OpenShift Online URL in a browser.
2. Click on the question mark icon in the top right-hand corner of the Web console, next to your user name.
3. Select *Command Line Tools* in the drop-down menu.
4. Find the text box that contains the **`oc login …`** command with the hidden token, and click the button next to it to copy its content to your clipboard.
5. Paste the command into a terminal application. The command uses your authentication token to authenticate your **oc** CLI client with your OpenShift Online account.

```
$ oc login OPENSHIFT_URL --token=MYTOKEN
```

3.2.4.3. Deploying the Externalized Configuration booster using the `oc` CLI client

Prerequisites

- The booster application created using developers.redhat.com/launch. For more information, see Section 3.2.4.1, “Deploying the booster using developers.redhat.com/launch”.
- The **oc** client authenticated. For more information, see Section 3.2.4.2, “Authenticating the **oc** CLI client”.

Procedure

1. Clone your project from GitHub.

```
$ git clone git@github.com:USERNAME/MY_PROJECT_NAME.git
```

Alternatively, if you downloaded a ZIP file of your project, extract it.

```
$ unzip MY_PROJECT_NAME.zip
```

2. Create a new OpenShift project.

```
$ oc new-project MY_PROJECT_NAME
```

3. Assign view access rights to the service account before deploying your booster, so that the booster can access the OpenShift API in order to read the contents of the ConfigMap.

```
$ oc policy add-role-to-user view -n $(oc project -q) -z default
```

4. Navigate to the root directory of your booster.

5. Deploy your ConfigMap configuration to OpenShift using **`app-config.yml`**.

```
$ oc create configmap app-config --from-file=app-config.yml
```

6. Verify your ConfigMap configuration has been deployed.

```
$ oc get configmap app-config -o yaml

apiVersion: v1
data:
  app-config.yml: |-
      message : "Hello, %s from a ConfigMap !"
      level : INFO
...
```

7. Use **`npm`** to start the deployment to OpenShift.

```
$ npm install && npm run openshift
```

These commands install any missing module dependencies, then using the Nodeshift module, deploy the booster on OpenShift.

8. Check the status of your booster and ensure your pod is running.

```
$ oc get pods -w
NAME                                      READY     STATUS
RESTARTS   AGE
MY_APP_NAME-1-aaaaa                1/1       Running     0
58s
MY_APP_NAME-s2i-1-build            0/1       Completed   0
2m
```

The **`MY_APP_NAME-1-aaaaa`** pod should have a status of **`Running`** once its fully deployed and started. Your specific pod name will vary. The number in the middle will increase with each new build. The letters at the end are generated when the pod is created.

9. Once your booster is deployed and started, determine its route.

Example Route Information

.

```
$ oc get routes
NAME                 HOST/PORT
PATH      SERVICES         PORT       TERMINATION
MY_APP_NAME          MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME
MY_APP_NAME       8080
```

The route information of a pod gives you the base URL which you use to access it. In the example above, you would use **`http://MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME`** as the base URL to access the application.

3.2.5. Deploying the Externalized Configuration booster to Single-node OpenShift Cluster

Use one of the following options to execute the Externalized Configuration booster locally on Single-node OpenShift Cluster:

- Using Fabric8 Launcher
- Using the **oc** CLI client

Although each method uses the same **oc** commands to deploy your application, using Fabric8 Launcher provides an automated booster deployment workflow that executes the **oc** commands for you.

3.2.5.1. Getting the Fabric8 Launcher tool URL and credentials

You need the Fabric8 Launcher tool URL and user credentials to create and deploy boosters on Single-node OpenShift Cluster. This information is provided when the Single-node OpenShift Cluster is started.

Prerequisites

- The Fabric8 Launcher tool installed, configured, and running. For more information, see the Install and Configure the Fabric8 Launcher Tool guide.

Procedure

1. Navigate to the console where you started Single-node OpenShift Cluster.
2. Check the console output for the URL and user credentials you can use to access the running Fabric8 Launcher:

Example Console Output from a Single-node OpenShift Cluster Startup

```
...
-- Removing temporary directory ... OK
-- Server Information ...
   OpenShift server started.
   The server is accessible via web console at:
       https://192.168.42.152:8443

   You are logged in as:
       User:     developer
       Password: developer
```

```
To login as administrator:
    oc login -u system:admin
```

3.2.5.2. Deploying the booster using the Fabric8 Launcher tool

Prerequisites

- The URL of your running Fabric8 Launcher instance and the user credentials of your Single-node OpenShift Cluster. For more information, see Section 3.2.5.1, “Getting the Fabric8 Launcher tool URL and credentials”.

Procedure

1. Navigate to the Fabric8 Launcher URL in a browser.
2. Follow on-screen instructions to create and launch your booster in Node.js.

3.2.5.3. Authenticating the `oc` CLI client

To work with boosters on Single-node OpenShift Cluster using the **oc** command-line client, you need to authenticate the client using the token provided by the Single-node OpenShift Cluster web interface.

Prerequisites

- The URL of your running Fabric8 Launcher instance and the user credentials of your Single-node OpenShift Cluster. For more information, see Section 3.2.5.1, “Getting the Fabric8 Launcher tool URL and credentials”.

Procedure

1. Navigate to the Single-node OpenShift Cluster URL in a browser.
2. Click on the question mark icon in the top right-hand corner of the Web console, next to your user name.
3. Select *Command Line Tools* in the drop-down menu.
4. Find the text box that contains the **`oc login …`** command with the hidden token, and click the button next to it to copy its content to your clipboard.
5. Paste the command into a terminal application. The command uses your authentication token to authenticate your **oc** CLI client with your Single-node OpenShift Cluster account.

```
$ oc login OPENSHIFT_URL --token=MYTOKEN
```

3.2.5.4. Deploying the Externalized Configuration booster using the `oc` CLI client

Prerequisites

- The booster application created using Fabric8 Launcher tool on a Single-node OpenShift Cluster. For more information, see Section 3.2.5.2, “Deploying the booster using the Fabric8 Launcher tool”.

- Your Fabric8 Launcher tool URL.
- The **oc** client authenticated. For more information, see Section 3.2.5.3, “Authenticating the **oc** CLI client”.

Procedure

1. Clone your project from GitHub.

```
$ git clone git@github.com:USERNAME/MY_PROJECT_NAME.git
```

   Alternatively, if you downloaded a ZIP file of your project, extract it.

```
$ unzip MY_PROJECT_NAME.zip
```

2. Create a new OpenShift project.

```
$ oc new-project MY_PROJECT_NAME
```

3. Assign view access rights to the service account before deploying your booster, so that the booster can access the OpenShift API in order to read the contents of the ConfigMap.

```
$ oc policy add-role-to-user view -n $(oc project -q) -z default
```

4. Navigate to the root directory of your booster.
5. Deploy your ConfigMap configuration to OpenShift using **`app-config.yml`**.

```
$ oc create configmap app-config --from-file=app-config.yml
```

6. Verify your ConfigMap configuration has been deployed.

```
$ oc get configmap app-config -o yaml

apiVersion: v1
data:
  app-config.yml: |-
      message : "Hello, %s from a ConfigMap !"
      level : INFO
...
```

7. Use **`npm`** to start the deployment to OpenShift.

```
$ npm install && npm run openshift
```

   These commands install any missing module dependencies, then using the Nodeshift module, deploy the booster on OpenShift.

8. Check the status of your booster and ensure your pod is running.

```
$ oc get pods -w
NAME                                      READY     STATUS
RESTARTS   AGE
```

```
MY_APP_NAME-1-aaaaa               1/1       Running     0
58s
MY_APP_NAME-s2i-1-build           0/1       Completed   0
2m
```

The **MY_APP_NAME-1-aaaaa** pod should have a status of **Running** once its fully deployed and started. Your specific pod name will vary. The number in the middle will increase with each new build. The letters at the end are generated when the pod is created.

9. Once your booster is deployed and started, determine its route.

Example Route Information

```
$ oc get routes
NAME                 HOST/PORT
PATH      SERVICES         PORT      TERMINATION
MY_APP_NAME          MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME
MY_APP_NAME      8080
```

The route information of a pod gives you the base URL which you use to access it. In the example above, you would use **http://MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME** as the base URL to access the application.

3.2.6. Deploying the Externalized Configuration booster to OpenShift Container Platform

The process of creating and deploying boosters to OpenShift Container Platform is similar to OpenShift Online:

Prerequisites

- The booster created using developers.redhat.com/launch or the Fabric8 Launcher tool.

Procedure

- Follow the instructions in Section 3.2.4, “Deploying the Externalized Configuration booster to OpenShift Online”, only use the URL and user credentials from the OpenShift Container Platform Web Console.

3.2.7. Interacting with the unmodified Externalized Configuration booster for Node.js

The booster provides a default HTTP endpoint that accepts **GET** requests.

Prerequisites

- Your application running
- The **curl** binary or a web browser

Procedure

1. Use **curl** to execute a **GET** request against the booster. You can also use a browser to do this.

```
$ curl http://MY_APP_NAME-
MY_PROJECT_NAME.OPENSHIFT_HOSTNAME/api/greeting
{"content":"Hello, World from a ConfigMap !"}
```

2. Update the deployed ConfigMap configuration.

```
$ oc edit configmap app-config
```

Change the value for the **`message`** key to **`Bonjour, %s from a ConfigMap !`** and save the file.

3. Update of the ConfigMap should be read by the application within an acceptable time (a few seconds) without requiring a restart of the application.

4. Execute a **`GET`** request using **`curl`** against the booster with the updated ConfigMap configuration to see your updated greeting. You can also do this from your browser using the web form provided by the application.

```
$ curl http://MY_APP_NAME-
MY_PROJECT_NAME.OPENSHIFT_HOSTNAME/api/greeting
{"content":"Bonjour, World from a ConfigMap !"}
```

3.2.8. Externalized Configuration resources

More background and related information on Externalized Configuration and ConfigMap can be found here:

- OpenShift ConfigMap Documentation
- Blog Post about ConfigMap in OpenShift
- Externalized Configuration mission - Spring Boot booster
- Externalized Configuration mission - Eclipse Vert.x booster
- Externalized Configuration mission - Thorntail booster

3.3. RELATIONAL DATABASE BACKEND MISSION - NODE.JS BOOSTER

Limitation: Run this booster on a Single-node OpenShift Cluster. You can also use a manual workflow to deploy this booster to OpenShift Online Pro and OpenShift Container Platform. This booster is not currently available on OpenShift Online Starter.

Mission proficiency level: **Foundational**.

What the Relational Database Backend Booster Does

The Relational Database Backend booster expands on the REST API Level 0 booster to provide a basic example of performing *create*, *read*, *update* and *delete* (*CRUD*) operations on a PostgreSQL database using a simple HTTP API. *CRUD* operations are the four basic functions of persistent storage, widely used when developing an HTTP API dealing with a database.

The booster also demonstrates the ability of the HTTP application to locate and connect to a database in OpenShift. Each runtime shows how to implement the connectivity solution best suited in the given case. The runtime can choose between options such as using *JDBC*, *JPA*, or accessing *ORM* APIs directly.

The booster application exposes an HTTP API, which provides endpoints that allow you to manipulate data by performing *CRUD* operations over HTTP. The *CRUD* operations are mapped to HTTP **`Verbs`**. The API uses JSON formatting to receive requests and return responses to the user. The user can also use an UI provided by the booster to use the application. Specifically, this booster provides an application that allows you to:

- Navigate to the application web interface in your browser. This exposes a simple website allowing you to perform *CRUD* operations on the data in the **`my_data`** database.
- Execute an HTTP **`GET`** request on the **`api/fruits`** endpoint.
- Receive a response formatted as a JSON array containing the list of all fruits in the database.
- Execute an HTTP **`GET`** request on the **`api/fruits/*`** endpoint while passing in a valid item ID as an argument.
- Receive a response in JSON format containing the name of the fruit with the given ID. If no item matches the specified ID, the call results in an HTTP error 404.
- Execute an HTTP **`POST`** request on the **`api/fruits`** endpoint passing in a valid **`name`** value to create a new entry in the database.
- Execute an HTTP **`PUT`** request on the **`api/fruits/*`** endpoint passing in a valid ID and a name as an argument. This updates the name of the item with the given ID to match the name specified in your request.
- Execute an HTTP **`DELETE`** request on the **`api/fruits/*`** endpoint, passing in a valid ID as an argument. This removes the item with the specified ID from the database and returns an HTTP code **`204`** (No Content) as a response. If you pass in an invalid ID, the call results in an HTTP error **`404`**.

This booster does not showcase a fully matured RESTful model (level 3), but it does use compatible HTTP verbs and status, following the recommended HTTP API practices.

3.3.1. Relational Database Backend design tradeoffs

Table 3.3. Design Tradeoffs

Pros	Cons

Pros	Cons
• Each runtime determines how to implement the database interactions. One can use a low-level connectivity API such as JDBC, some other can use JPA, and yet another can access ORM APIs directly. Each runtime decides what would be the best way. • Each runtime determines how the schema is created.	• The PostgreSQL database example provided with this mission is not backed up with persistent storage. Changes to the database are lost if you stop or redeploy the database pod. To use an external database with your mission's pod in order to preserve changes, see the Integrating External Services chapter of the OpenShift Documentation. It is also possible to set up persistent storage with database containers on OpenShift. (For more details about using persistent storage with OpenShift and containers, see the Persistent Storage, Managing Volumes and Persistent Volumes chapters of the OpenShift Documentation).

3.3.2. Viewing the booster source code and README

Prerequisites

One of the following:

- Access to developers.redhat.com/launch
- Fabric8 Launcher installed on a Single-node OpenShift Cluster

Procedure

1. Use the Fabric8 Launcher tool to generate your own version of the booster.
2. View the generated GitHub repository or download and extract the ZIP file that contains the booster source code.

Additional resources

- Using developers.redhat.com/launch
- Using the Fabric8 Launcher tool on a Single-node OpenShift Cluster

3.3.3. Deploying the Relational Database Backend booster to OpenShift Online

Use one of the following options to execute the Relational Database Backend booster on OpenShift Online.

- Use developers.redhat.com/launch
- Use the **oc** CLI client

Although each method uses the same **oc** commands to deploy your application, using developers.redhat.com/launch provides an automated booster deployment workflow that executes the **oc** commands for you.

3.3.3.1. Deploying the booster using developers.redhat.com/launch

Prerequisites

- An account at OpenShift Online.

Procedure

1. Navigate to the developers.redhat.com/launch URL in a browser and log in.
2. Follow on-screen instructions to create and launch your booster in Node.js.

3.3.3.2. Authenticating the oc CLI client

To work with boosters on OpenShift Online using the **oc** command-line client, you need to authenticate the client using the token provided by the OpenShift Online web interface.

Prerequisites

- An account at OpenShift Online.

Procedure

1. Navigate to the OpenShift Online URL in a browser.
2. Click on the question mark icon in the top right-hand corner of the Web console, next to your user name.
3. Select *Command Line Tools* in the drop-down menu.
4. Find the text box that contains the **`oc login …`** command with the hidden token, and click the button next to it to copy its content to your clipboard.
5. Paste the command into a terminal application. The command uses your authentication token to authenticate your **oc** CLI client with your OpenShift Online account.

```
$ oc login OPENSHIFT_URL --token=MYTOKEN
```

3.3.3.3. Deploying the Relational Database Backend booster using theoc CLI client

Prerequisites

- The booster application created using developers.redhat.com/launch. For more information, see Section 3.3.3.1, “Deploying the booster using developers.redhat.com/launch”.
- The **oc** client authenticated. For more information, see Section 3.3.3.2, “Authenticating the **oc** CLI client”.

Procedure

1. Clone your project from GitHub.

```
$ git clone git@github.com:USERNAME/MY_PROJECT_NAME.git
```

Alternatively, if you downloaded a ZIP file of your project, extract it.

```
$ unzip MY_PROJECT_NAME.zip
```

2. Create a new OpenShift project.

```
$ oc new-project MY_PROJECT_NAME
```

3. Navigate to the root directory of your booster.

4. Deploy the PostgreSQL database to OpenShift. Ensure that you use the following values for user name, password, and database name when creating your database application. The booster application is pre-configured to use these values. Using different values prevents your booster application from integrating with the database.

```
$ oc new-app -e POSTGRESQL_USER=luke -ePOSTGRESQL_PASSWORD=secret -
ePOSTGRESQL_DATABASE=my_data openshift/postgresql-92-centos7 --
name=my-database
```

5. Check the status of your database and ensure the pod is running.

```
$ oc get pods -w
my-database-1-aaaaa   1/1       Running   0         45s
my-database-1-deploy   0/1       Completed   0         53s
```

The **my-database-1-aaaaa** pod should have a status of **Running** and should be indicated as ready once it is fully deployed and started. Your specific pod name will vary. The number in the middle will increase with each new build. The letters at the end are generated when the pod is created.

6. Use **npm** to start the deployment to OpenShift.

```
$ npm install && npm run openshift
```

These commands install any missing module dependencies, then using the Nodeshift module, deploy the booster on OpenShift.

7. Check the status of your booster and ensure your pod is running.

```
$ oc get pods -w
NAME                                READY     STATUS      RESTARTS
AGE
MY_APP_NAME-1-aaaaa       1/1       Running     0          58s
MY_APP_NAME-s2i-1-build   0/1       Completed   0          2m
```

Your **MY_APP_NAME-1-aaaaa** pod should have a status of **Running** and should be indicated as ready once it is fully deployed and started.

8. Once your booster is deployed and started, determine its route.

Example Route Information

```
$ oc get routes
NAME                 HOST/PORT
```

```
PATH      SERVICES             PORT      TERMINATION
MY_APP_NAME   MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME
MY_APP_NAME   8080
```

The route information of a pod gives you the base URL which you use to access it. In the example above, you would use **`http://MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME`** as the base URL to access the application.

3.3.4. Deploying the Relational Database Backend booster to Single-node OpenShift Cluster

Use one of the following options to execute the Relational Database Backend booster locally on Single-node OpenShift Cluster:

- Using Fabric8 Launcher
- Using the **oc** CLI client

Although each method uses the same **oc** commands to deploy your application, using Fabric8 Launcher provides an automated booster deployment workflow that executes the **oc** commands for you.

3.3.4.1. Getting the Fabric8 Launcher tool URL and credentials

You need the Fabric8 Launcher tool URL and user credentials to create and deploy boosters on Single-node OpenShift Cluster. This information is provided when the Single-node OpenShift Cluster is started.

Prerequisites

- The Fabric8 Launcher tool installed, configured, and running. For more information, see the Install and Configure the Fabric8 Launcher Tool guide.

Procedure

1. Navigate to the console where you started Single-node OpenShift Cluster.
2. Check the console output for the URL and user credentials you can use to access the running Fabric8 Launcher:

Example Console Output from a Single-node OpenShift Cluster Startup

```
...
-- Removing temporary directory ... OK
-- Server Information ...
   OpenShift server started.
   The server is accessible via web console at:
       https://192.168.42.152:8443

   You are logged in as:
       User:     developer
       Password: developer

   To login as administrator:
       oc login -u system:admin
```

3.3.4.2. Deploying the booster using the Fabric8 Launcher tool

Prerequisites

- The URL of your running Fabric8 Launcher instance and the user credentials of your Single-node OpenShift Cluster. For more information, see Section 3.3.4.1, “Getting the Fabric8 Launcher tool URL and credentials”.

Procedure

1. Navigate to the Fabric8 Launcher URL in a browser.
2. Follow on-screen instructions to create and launch your booster in Node.js.

3.3.4.3. Authenticating the oc CLI client

To work with boosters on Single-node OpenShift Cluster using the **oc** command-line client, you need to authenticate the client using the token provided by the Single-node OpenShift Cluster web interface.

Prerequisites

- The URL of your running Fabric8 Launcher instance and the user credentials of your Single-node OpenShift Cluster. For more information, see Section 3.3.4.1, “Getting the Fabric8 Launcher tool URL and credentials”.

Procedure

1. Navigate to the Single-node OpenShift Cluster URL in a browser.
2. Click on the question mark icon in the top right-hand corner of the Web console, next to your user name.
3. Select *Command Line Tools* in the drop-down menu.
4. Find the text box that contains the **`oc login …`** command with the hidden token, and click the button next to it to copy its content to your clipboard.
5. Paste the command into a terminal application. The command uses your authentication token to authenticate your **oc** CLI client with your Single-node OpenShift Cluster account.

```
$ oc login OPENSHIFT_URL --token=MYTOKEN
```

3.3.4.4. Deploying the Relational Database Backend booster using theoc CLI client

Prerequisites

- The booster application created using Fabric8 Launcher tool on a Single-node OpenShift Cluster. For more information, see Section 3.3.4.2, “Deploying the booster using the Fabric8 Launcher tool”.
- Your Fabric8 Launcher tool URL.
- The **oc** client authenticated. For more information, see Section 3.3.4.3, “Authenticating the **oc** CLI client”.

Procedure

1. Clone your project from GitHub.

```
$ git clone git@github.com:USERNAME/MY_PROJECT_NAME.git
```

   Alternatively, if you downloaded a ZIP file of your project, extract it.

```
$ unzip MY_PROJECT_NAME.zip
```

2. Create a new OpenShift project.

```
$ oc new-project MY_PROJECT_NAME
```

3. Navigate to the root directory of your booster.

4. Deploy the PostgreSQL database to OpenShift. Ensure that you use the following values for user name, password, and database name when creating your database application. The booster application is pre-configured to use these values. Using different values prevents your booster application from integrating with the database.

```
$ oc new-app -e POSTGRESQL_USER=luke -ePOSTGRESQL_PASSWORD=secret -
ePOSTGRESQL_DATABASE=my_data openshift/postgresql-92-centos7 --
name=my-database
```

5. Check the status of your database and ensure the pod is running.

```
$ oc get pods -w
my-database-1-aaaaa   1/1       Running   0         45s
my-database-1-deploy   0/1       Completed   0         53s
```

   The **my-database-1-aaaaa** pod should have a status of **Running** and should be indicated as ready once it is fully deployed and started. Your specific pod name will vary. The number in the middle will increase with each new build. The letters at the end are generated when the pod is created.

6. Use **npm** to start the deployment to OpenShift.

```
$ npm install && npm run openshift
```

   These commands install any missing module dependencies, then using the Nodeshift module, deploy the booster on OpenShift.

7. Check the status of your booster and ensure your pod is running.

```
$ oc get pods -w
NAME                                READY     STATUS      RESTARTS
AGE
MY_APP_NAME-1-aaaaa       1/1       Running     0          58s
MY_APP_NAME-s2i-1-build   0/1       Completed   0          2m
```

   Your **MY_APP_NAME-1-aaaaa** pod should have a status of **Running** and should be indicated as ready once it is fully deployed and started.

8. Once your booster is deployed and started, determine its route.

   Example Route Information

   ```
   $ oc get routes
   NAME                 HOST/PORT
   PATH      SERVICES             PORT      TERMINATION
   MY_APP_NAME   MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME
   MY_APP_NAME   8080
   ```

   The route information of a pod gives you the base URL which you use to access it. In the example above, you would use **`http://MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME`** as the base URL to access the application.

3.3.5. Deploying the Relational Database Backend booster to OpenShift Container Platform

The process of creating and deploying boosters to OpenShift Container Platform is similar to OpenShift Online:

Prerequisites

- The booster created using developers.redhat.com/launch or the Fabric8 Launcher tool.

Procedure

- Follow the instructions in Section 3.3.3, “Deploying the Relational Database Backend booster to OpenShift Online”, only use the URL and user credentials from the OpenShift Container Platform Web Console.

3.3.6. Interacting with the Relational Database Backend API on Node.js

When you have finished creating your application booster, you can interact with it the following way:

Prerequisites

- Your application running
- The **`curl`** binary or a web browser

Procedure

1. Obtain the URL of your application by executing the following command:

   ```
   $ oc get route MY_APP_NAME
   ```

   ```
   NAME                 HOST/PORT
   PATH      SERVICES             PORT      TERMINATION
   MY_APP_NAME           MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME
   MY_APP_NAME           8080
   ```

2. To access the web interface of the database application, navigate to the *application URL* in your browser:

```
http://MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME
```

Alternatively, you can make requests directly on the **api/fruits/*** endpoint using **curl**:

List all entries in the database:

```
$ curl http://MY_APP_NAME-
MY_PROJECT_NAME.OPENSHIFT_HOSTNAME/api/fruits
```

```
[ {
  "id" : 1,
  "name" : "Apple",
  "stock" : 10
}, {
  "id" : 2,
  "name" : "Orange",
  "stock" : 10
}, {
  "id" : 3,
  "name" : "Pear",
  "stock" : 10
} ]
```

Retrieve an entry with a specific ID

```
$ curl http://MY_APP_NAME-
MY_PROJECT_NAME.OPENSHIFT_HOSTNAME/api/fruits/3
```

```
{
  "id" : 3,
  "name" : "Pear",
  "stock" : 10
}
```

Create a new entry:

```
$ curl -H "Content-Type: application/json" -X POST -d
'{"name":"Peach","stock":1}'  http://MY_APP_NAME-
MY_PROJECT_NAME.OPENSHIFT_HOSTNAME/api/fruits
```

```
{
  "id" : 4,
  "name" : "Peach",
  "stock" : 1
}
```

Update an Entry

```
$ curl -H "Content-Type: application/json" -X PUT -d
'{"name":"Apple","stock":100}'  http://MY_APP_NAME-
MY_PROJECT_NAME.OPENSHIFT_HOSTNAME/api/fruits/1
```

```
{
  "id" : 1,
  "name" : "Apple",
  "stock" : 100
}
```

Delete an Entry:

```
$ curl -X DELETE http://MY_APP_NAME-
MY_PROJECT_NAME.OPENSHIFT_HOSTNAME/api/fruits/1
```

Troubleshooting

- If you receive an HTTP Error code **503** as a response after executing these commands, it means that the application is not ready yet.

3.3.7. Relational database resources

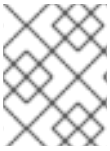
More background and related information on running relational databases in OpenShift, CRUD, HTTP API and REST can be found here:

- HTTP Verbs
- Architectural Styles and the Design of Network-based Software Architectures - Representational State Transfer (REST)
- The never ending REST API design debase
- REST APIs must be Hypertext driven
- Richardson Maturity Model
- Express Web Framework
- Relational Database Backend mission - Spring Boot booster
- Relational Database Backend mission - Eclipse Vert.x booster
- Relational Database Backend mission - Thorntail booster

3.4. HEALTH CHECK MISSION - NODE.JS BOOSTER

Mission proficiency level: **Foundational**.

When you deploy an application, its important to know if it is available and if it can start handling incoming requests. Implementing the *health check* pattern allows you to monitor the health of an application, which includes if an application is available and whether it is able to service requests.



NOTE

If you are not familiar with the health check terminology, see the Section 3.4.1, “Health check concepts” section first.

The purpose of this use case is to demonstrate the health check pattern through the use of probing.

Probing is used to report the liveness and readiness of an application. In this use case, you configure an application which exposes an HTTP **`health`** endpoint to issue HTTP requests. If the container is alive, according to the liveness probe on the **`health`** HTTP endpoint, the management platform receives **`200`** as return code and no further action is required. If the **`health`** HTTP endpoint does not return a response, for example if the thread is blocked, then the application is not considered alive according to the liveness probe. In that case, the platform kills the pod corresponding to that application and recreates a new pod to restart the application.

This use case also allows you to demonstrate and use a readiness probe. In cases where the application is running but is unable to handle requests, such as when the application returns an HTTP **`503`** response code during restart, this application is not considered ready according to the readiness probe. If the application is not considered ready by the readiness probe, requests are not routed to that application until it is considered ready according to the readiness probe.

3.4.1. Health check concepts

In order to understand the health check pattern, you need to first understand the following concepts:

Liveness

Liveness defines whether an application is running or not. Sometimes a running application moves into an unresponsive or stopped state and needs to be restarted. Checking for liveness helps determine whether or not an application needs to be restarted.

Readiness

Readiness defines whether a running application can service requests. Sometimes a running application moves into an error or broken state where it can no longer service requests. Checking readiness helps determine whether or not requests should continue to be routed to that application.

Fail-over

Fail-over enables failures in servicing requests to be handled gracefully. If an application fails to service a request, that request and future requests can then *fail-over* or be routed to another application, which is usually a redundant copy of that same application.

Resilience and Stability

Resilience and Stability enable failures in servicing requests to be handled gracefully. If an application fails to service a request due to connection loss, in a resilient system that request can be retried after the connection is re-established.

Probe

A probe is a Kubernetes action that periodically performs diagnostics on a running container.

3.4.2. Viewing the booster source code and README

Prerequisites

One of the following:

- Access to developers.redhat.com/launch
- Fabric8 Launcher installed on a Single-node OpenShift Cluster

Procedure

1. Use the Fabric8 Launcher tool to generate your own version of the booster.

2. View the generated GitHub repository or download and extract the ZIP file that contains the booster source code.

Additional resources

- Using developers.redhat.com/launch
- Using the Fabric8 Launcher tool on a Single-node OpenShift Cluster

3.4.3. Deploying the Health Check booster to OpenShift Online

Use one of the following options to execute the Health Check booster on OpenShift Online.

- Use developers.redhat.com/launch
- Use the **oc** CLI client

Although each method uses the same **oc** commands to deploy your application, using developers.redhat.com/launch provides an automated booster deployment workflow that executes the **oc** commands for you.

3.4.3.1. Deploying the booster using developers.redhat.com/launch

Prerequisites

- An account at OpenShift Online.

Procedure

1. Navigate to the developers.redhat.com/launch URL in a browser and log in.
2. Follow on-screen instructions to create and launch your booster in Node.js.

3.4.3.2. Authenticating the oc CLI client

To work with boosters on OpenShift Online using the **oc** command-line client, you need to authenticate the client using the token provided by the OpenShift Online web interface.

Prerequisites

- An account at OpenShift Online.

Procedure

1. Navigate to the OpenShift Online URL in a browser.
2. Click on the question mark icon in the top right-hand corner of the Web console, next to your user name.
3. Select *Command Line Tools* in the drop-down menu.
4. Find the text box that contains the **`oc login …`** command with the hidden token, and click the button next to it to copy its content to your clipboard.

5. Paste the command into a terminal application. The command uses your authentication token to authenticate your **oc** CLI client with your OpenShift Online account.

```
$ oc login OPENSHIFT_URL --token=MYTOKEN
```

3.4.3.3. Deploying the Health Check booster using the oc CLI client

Prerequisites

- The booster application created using developers.redhat.com/launch. For more information, see Section 3.4.3.1, “Deploying the booster using developers.redhat.com/launch”.
- The **oc** client authenticated. For more information, see Section 3.4.3.2, “Authenticating the **oc** CLI client”.

Procedure

1. Clone your project from GitHub.

```
$ git clone git@github.com:USERNAME/MY_PROJECT_NAME.git
```

   Alternatively, if you downloaded a ZIP file of your project, extract it.

```
$ unzip MY_PROJECT_NAME.zip
```

2. Create a new OpenShift project.

```
$ oc new-project MY_PROJECT_NAME
```

3. Navigate to the root directory of your booster.

4. Use **npm** to start the deployment to OpenShift.

```
$ npm install && npm run openshift
```

   These commands install any missing module dependencies, then using the Nodeshift module, deploy the booster on OpenShift.

5. Check the status of your booster and ensure your pod is running.

```
$ oc get pods -w
NAME                              READY     STATUS      RESTARTS
AGE
MY_APP_NAME-1-aaaaa               1/1       Running     0
58s
MY_APP_NAME-s2i-1-build           0/1       Completed   0
2m
```

   The **MY_APP_NAME-1-aaaaa** pod should have a status of **Running** once its fully deployed and started. You should also wait for your pod to be ready before proceeding, which is shown in the **READY** column. For example, **MY_APP_NAME-1-aaaaa** is ready when the **READY** column is **1/1**. Your specific pod name will vary. The number in the middle will increase with each new build. The letters at the end are generated when the pod is created.

6. Once your booster is deployed and started, determine its route.

Example Route Information

```
$ oc get routes
NAME                 HOST/PORT
PATH      SERVICES         PORT      TERMINATION
MY_APP_NAME          MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME
MY_APP_NAME      8080
```

The route information of a pod gives you the base URL which you use to access it. In the example above, you would use **`http://MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME`** as the base URL to access the application.

3.4.4. Deploying the Health Check booster to Single-node OpenShift Cluster

Use one of the following options to execute the Health Check booster locally on Single-node OpenShift Cluster:

- Using Fabric8 Launcher
- Using the **oc** CLI client

Although each method uses the same **oc** commands to deploy your application, using Fabric8 Launcher provides an automated booster deployment workflow that executes the **oc** commands for you.

3.4.4.1. Getting the Fabric8 Launcher tool URL and credentials

You need the Fabric8 Launcher tool URL and user credentials to create and deploy boosters on Single-node OpenShift Cluster. This information is provided when the Single-node OpenShift Cluster is started.

Prerequisites

- The Fabric8 Launcher tool installed, configured, and running. For more information, see the Install and Configure the Fabric8 Launcher Tool guide.

Procedure

1. Navigate to the console where you started Single-node OpenShift Cluster.
2. Check the console output for the URL and user credentials you can use to access the running Fabric8 Launcher:

Example Console Output from a Single-node OpenShift Cluster Startup

```
...
-- Removing temporary directory ... OK
-- Server Information ...
   OpenShift server started.
   The server is accessible via web console at:
       https://192.168.42.152:8443

   You are logged in as:
       User:     developer
```

```
        Password: developer

    To login as administrator:
        oc login -u system:admin
```

3.4.4.2. Deploying the booster using the Fabric8 Launcher tool

Prerequisites

- The URL of your running Fabric8 Launcher instance and the user credentials of your Single-node OpenShift Cluster. For more information, see Section 3.4.4.1, “Getting the Fabric8 Launcher tool URL and credentials”.

Procedure

1. Navigate to the Fabric8 Launcher URL in a browser.
2. Follow on-screen instructions to create and launch your booster in Node.js.

3.4.4.3. Authenticating the `oc` CLI client

To work with boosters on Single-node OpenShift Cluster using the **oc** command-line client, you need to authenticate the client using the token provided by the Single-node OpenShift Cluster web interface.

Prerequisites

- The URL of your running Fabric8 Launcher instance and the user credentials of your Single-node OpenShift Cluster. For more information, see Section 3.4.4.1, “Getting the Fabric8 Launcher tool URL and credentials”.

Procedure

1. Navigate to the Single-node OpenShift Cluster URL in a browser.
2. Click on the question mark icon in the top right-hand corner of the Web console, next to your user name.
3. Select *Command Line Tools* in the drop-down menu.
4. Find the text box that contains the **`oc login …`** command with the hidden token, and click the button next to it to copy its content to your clipboard.
5. Paste the command into a terminal application. The command uses your authentication token to authenticate your **oc** CLI client with your Single-node OpenShift Cluster account.

```
$ oc login OPENSHIFT_URL --token=MYTOKEN
```

3.4.4.4. Deploying the Health Check booster using the `oc` CLI client

Prerequisites

- The booster application created using Fabric8 Launcher tool on a Single-node OpenShift Cluster. For more information, see Section 3.4.4.2, “Deploying the booster using the Fabric8 Launcher tool”.
- Your Fabric8 Launcher tool URL.
- The **oc** client authenticated. For more information, see Section 3.4.4.3, “Authenticating the **oc** CLI client”.

Procedure

1. Clone your project from GitHub.

```
$ git clone git@github.com:USERNAME/MY_PROJECT_NAME.git
```

   Alternatively, if you downloaded a ZIP file of your project, extract it.

```
$ unzip MY_PROJECT_NAME.zip
```

2. Create a new OpenShift project.

```
$ oc new-project MY_PROJECT_NAME
```

3. Navigate to the root directory of your booster.
4. Use **npm** to start the deployment to OpenShift.

```
$ npm install && npm run openshift
```

   These commands install any missing module dependencies, then using the Nodeshift module, deploy the booster on OpenShift.

5. Check the status of your booster and ensure your pod is running.

```
$ oc get pods -w
NAME                                READY     STATUS      RESTARTS
AGE
MY_APP_NAME-1-aaaaa                 1/1       Running     0
58s
MY_APP_NAME-s2i-1-build             0/1       Completed   0
2m
```

   The **MY_APP_NAME-1-aaaaa** pod should have a status of **Running** once its fully deployed and started. You should also wait for your pod to be ready before proceeding, which is shown in the **READY** column. For example, **MY_APP_NAME-1-aaaaa** is ready when the **READY** column is **1/1**. Your specific pod name will vary. The number in the middle will increase with each new build. The letters at the end are generated when the pod is created.

6. Once your booster is deployed and started, determine its route.

   Example Route Information

```
$ oc get routes
NAME                 HOST/PORT
```

```
PATH      SERVICES        PORT      TERMINATION
MY_APP_NAME          MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME
MY_APP_NAME      8080
```

The route information of a pod gives you the base URL which you use to access it. In the example above, you would use **`http://MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME`** as the base URL to access the application.

3.4.5. Deploying the Health Check booster to OpenShift Container Platform

The process of creating and deploying boosters to OpenShift Container Platform is similar to OpenShift Online:

Prerequisites

- The booster created using developers.redhat.com/launch or the Fabric8 Launcher tool.

Procedure

- Follow the instructions in Section 3.4.3, “Deploying the Health Check booster to OpenShift Online”, only use the URL and user credentials from the OpenShift Container Platform Web Console.

3.4.6. Interacting with the unmodified Health Check booster

Once you have the booster deployed, you will have a service called **`MY_APP_NAME`** running that exposes the following REST endpoints:

/api/greeting

Returns a JSON containing greeting of **`name`** parameter (or World as default value).

/api/stop

Forces the service to become unresponsive as means to simulate a failure.

The following steps demonstrate how to verify the service availability and simulate a failure. This failure of an available service causes the OpenShift self-healing capabilities to be trigger on the service.

Alternatively, you can use the web interface to perform these steps.

1. Use **`curl`** to execute a **`GET`** request against the **`MY_APP_NAME`** service. You can also use a browser to do this.

```
$ curl http://MY_APP_NAME-
MY_PROJECT_NAME.OPENSHIFT_HOSTNAME/api/greeting
```

```
{"content":"Hello, World!"}
```

2. Invoke the **`/api/stop`** endpoint and verify the availability of the **`/api/greeting`** endpoint shortly after that.
   Invoking the **`/api/stop`** endpoint simulates an internal service failure and triggers the OpenShift self-healing capabilities. When invoking **`/api/greeting`** after simulating the failure, the service should return a HTTP status **`503`**.

```
$ curl http://MY_APP_NAME-
MY_PROJECT_NAME.OPENSHIFT_HOSTNAME/api/stop
```

```
Stopping HTTP server, Bye bye world !
```

(followed by)

```
$ curl http://MY_APP_NAME-
MY_PROJECT_NAME.OPENSHIFT_HOSTNAME/api/greeting
```

```
Not online
```

3. Use **`oc get pods -w`** to continuously watch the self-healing capabilities in action.
While invoking the service failure, you can watch the self-healing capabilities in action on OpenShift console, or with the **`oc`** client tools. You should see the number of pods in the **READY** state move to zero (**`0/1`**) and after a short period (less than one minute) move back up to one (**`1/1`**). In addition to that, the **RESTARTS** count increases every time you you invoke the service failure.

```
$ oc get pods -w
NAME                              READY     STATUS    RESTARTS   AGE
MY_APP_NAME-1-26iy7   0/1       Running   5          18m
MY_APP_NAME-1-26iy7   1/1       Running   5         19m
```

4. Optional: Use the web interface to invoke the service.
Alternatively to the interaction using the terminal window, you can use the web interface provided by the service to invoke the different methods and watch the service move through the life cycle phases.

```
http://MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME
```

5. Optional: Use the web console to view the log output generated by the application at each stage of the self-healing process.

    1. Navigate to your project.
    2. On the sidebar, click on *Monitoring*.
    3. In the upper right-hand corner of the screen, click on *Events* to display the log messages.
    4. Optional: Click *View Details* to display a detailed view of the Event log.

    The health check application generates the following messages:

Message	Status
Unhealthy	Readiness probe failed. This message is expected and indicates that the simulated failure of the **`/api/greeting`** endpoint has been detected and the self-healing process starts.

Message	Status
Killing	The unavailable Docker container running the service is being killed before being re-created.
Pulling	Downloading the latest version of docker image to re-create the container.
Pulled	Docker image downloaded successfully.
Created	Docker container has been successfully created
Started	Docker container is ready to handle requests

3.4.7. Health check resources

More background and related information on health checking can be found here:

- Overview of Application Health in OpenShift
- Health Checking in Kubernetes
- Kubernetes Liveness and Readiness Probes
- Kubernetes Probes
- Health Check mission - Spring Boot booster
- Health Check mission - Eclipse Vert.x booster
- Health Check mission - Thorntail booster

3.5. CIRCUIT BREAKER MISSION - NODE.JS BOOSTER

Limitation: Run this booster on a Single-node OpenShift Cluster. You can also use a manual workflow to deploy this booster to OpenShift Online Pro and OpenShift Container Platform. This booster is not currently available on OpenShift Online Starter.

Mission proficiency level: **Foundational**.

The *Circuit Breaker* mission demonstrates a generic pattern for reporting the failure of a service and then limiting access to the failed service until it becomes available to handle requests. This helps prevent cascading failure in other services that depend on the failed services for functionality.

This mission shows you how to implement a Circuit Breaker and Fallback pattern in your services.

3.5.1. The circuit breaker design pattern

The Circuit Breaker is a pattern intended to:

- Reduce the impact of network failure and high latency on service architectures where services synchronously invoke other services.
  If one of the services:
  - becomes unavailable due to network failure, or
  - incurs unusually high latency values due to overwhelming traffic,

  other services attempting to call its endpoint may end up exhausting critical resources in an attempt to reach it, rendering themselves unusable.
- Prevent the condition also known as cascading failure, which can render the entire microservice architecture unusable.
- Act as a proxy between a protected function and a remote function, which monitors for failures.
- Trip once the failures reach a certain threshold, and all further calls to the circuit breaker return an error or a predefined fallback response, without the protected call being made at all.

The Circuit Breaker usually also contain an error reporting mechanism that notifies you when the Circuit Breaker trips.

Circuit breaker implementation

- With the Circuit Breaker pattern implemented, a service client invokes a remote service endpoint via a proxy at regular intervals.
- If the calls to the remote service endpoint fail repeatedly and consistently, the Circuit Breaker trips, making all calls to the service fail immediately over a set timeout period and returns a predefined fallback response.
- When the timeout period expires, a limited number of test calls are allowed to pass through to the remote service to determine whether it has healed, or remains unavailable.
  - If the test calls fail, the Circuit Breaker keeps the service unavailable and keeps returning the fallback responses to incoming calls.
  - If the test calls succeed, the Circuit Breaker closes, fully enabling traffic to reach the remote service again.

3.5.2. Circuit Breaker design tradeoffs

Table 3.4. Design Tradeoffs

Pros	Cons
• Enables a service to handle the failure of other services it invokes.	• Optimizing the timeout values can be challenging ◦ Larger-than-necessary timeout values may generate excessive latency. ◦ Smaller-than-necessary timeout values may introduce false positives.

3.5.3. Viewing the booster source code and README

Prerequisites

One of the following:

- Access to developers.redhat.com/launch
- Fabric8 Launcher installed on a Single-node OpenShift Cluster

Procedure

1. Use the Fabric8 Launcher tool to generate your own version of the booster.
2. View the generated GitHub repository or download and extract the ZIP file that contains the booster source code.

Additional resources

- Using developers.redhat.com/launch
- Using the Fabric8 Launcher tool on a Single-node OpenShift Cluster

3.5.4. Deploying the Circuit Breaker booster to OpenShift Online

Use one of the following options to execute the Circuit Breaker booster on OpenShift Online.

- Use developers.redhat.com/launch
- Use the **oc** CLI client

Although each method uses the same **oc** commands to deploy your application, using developers.redhat.com/launch provides an automated booster deployment workflow that executes the **oc** commands for you.

3.5.4.1. Deploying the booster using developers.redhat.com/launch

Prerequisites

- An account at OpenShift Online.

Procedure

1. Navigate to the developers.redhat.com/launch URL in a browser and log in.
2. Follow on-screen instructions to create and launch your booster in Node.js.

3.5.4.2. Authenticating the oc CLI client

To work with boosters on OpenShift Online using the **oc** command-line client, you need to authenticate the client using the token provided by the OpenShift Online web interface.

Prerequisites

- An account at OpenShift Online.

Procedure

1. Navigate to the OpenShift Online URL in a browser.
2. Click on the question mark icon in the top right-hand corner of the Web console, next to your user name.
3. Select *Command Line Tools* in the drop-down menu.
4. Find the text box that contains the **`oc login …`** command with the hidden token, and click the button next to it to copy its content to your clipboard.
5. Paste the command into a terminal application. The command uses your authentication token to authenticate your **oc** CLI client with your OpenShift Online account.

```
$ oc login OPENSHIFT_URL --token=MYTOKEN
```

3.5.4.3. Deploying the Circuit Breaker booster using the `oc` CLI client

Prerequisites

- The booster application created using developers.redhat.com/launch. For more information, see Section 3.5.4.1, “Deploying the booster using developers.redhat.com/launch”.
- The **oc** client authenticated. For more information, see Section 3.5.4.2, “Authenticating the **oc** CLI client”.

Procedure

1. Clone your project from GitHub.

```
$ git clone git@github.com:USERNAME/MY_PROJECT_NAME.git
```

   Alternatively, if you downloaded a ZIP file of your project, extract it.

```
$ unzip MY_PROJECT_NAME.zip
```

2. Create a new OpenShift project.

```
$ oc new-project MY_PROJECT_NAME
```

3. Navigate to the root directory of your booster.
4. Use the provided **`start-openshift.sh`** script to start the deployment to OpenShift.

```
$ chmod +x start-openshift.sh
$ ./start-openshift.sh
```

   These commands use the Nodeshift **`npm`** module to install your dependencies, launch the S2I build process on OpenShift, and start the services.

5. Check the status of your booster and ensure your pod is running.

```
$ oc get pods -w
NAME                                READY     STATUS      RESTARTS
AGE
MY_APP_NAME-greeting-1-aaaaa     1/1       Running   0
17s
MY_APP_NAME-greeting-1-deploy    0/1       Completed 0
22s
MY_APP_NAME-name-1-aaaaa         1/1       Running   0
14s
MY_APP_NAME-name-1-deploy        0/1       Completed 0
28s
```

Both the **MY_APP_NAME-greeting-1-aaaaa** and **MY_APP_NAME-name-1-aaaaa** pods should have a status of **Running** once they are fully deployed and started. You should also wait for your pods to be ready before proceeding, which is shown in the **READY** column. For example, **MY_APP_NAME-greeting-1-aaaaa** is ready when the **READY** column is **1/1**. Your specific pod names will vary. The number in the middle will increase with each new build. The letters at the end are generated when the pod is created.

6. Once your booster is deployed and started, determine its route.

Example Route Information

```
$ oc get routes
NAME                 HOST/PORT
PATH      SERVICES        PORT      TERMINATION
MY_APP_NAME-greeting   MY_APP_NAME-greeting-
MY_PROJECT_NAME.OPENSHIFT_HOSTNAME            MY_APP_NAME-greeting
8080                    None
MY_APP_NAME-name       MY_APP_NAME-name-
MY_PROJECT_NAME.OPENSHIFT_HOSTNAME            MY_APP_NAME-name
8080                    None
```

The route information of a pod gives you the base URL which you use to access it. In the example above, you would use **http://MY_APP_NAME-greeting-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME** as the base URL to access the application.

3.5.5. Deploying the Circuit Breaker booster to Single-node OpenShift Cluster

Use one of the following options to execute the Circuit Breaker booster locally on Single-node OpenShift Cluster:

- Using Fabric8 Launcher
- Using the **oc** CLI client

Although each method uses the same **oc** commands to deploy your application, using Fabric8 Launcher provides an automated booster deployment workflow that executes the **oc** commands for you.

3.5.5.1. Getting the Fabric8 Launcher tool URL and credentials

You need the Fabric8 Launcher tool URL and user credentials to create and deploy boosters on Single-node OpenShift Cluster. This information is provided when the Single-node OpenShift Cluster is started.

Prerequisites

- The Fabric8 Launcher tool installed, configured, and running. For more information, see the Install and Configure the Fabric8 Launcher Tool guide.

Procedure

1. Navigate to the console where you started Single-node OpenShift Cluster.
2. Check the console output for the URL and user credentials you can use to access the running Fabric8 Launcher:

   Example Console Output from a Single-node OpenShift Cluster Startup

```
...
-- Removing temporary directory ... OK
-- Server Information ...
   OpenShift server started.
   The server is accessible via web console at:
       https://192.168.42.152:8443

   You are logged in as:
       User:     developer
       Password: developer

   To login as administrator:
       oc login -u system:admin
```

3.5.5.2. Deploying the booster using the Fabric8 Launcher tool

Prerequisites

- The URL of your running Fabric8 Launcher instance and the user credentials of your Single-node OpenShift Cluster. For more information, see Section 3.5.5.1, “Getting the Fabric8 Launcher tool URL and credentials”.

Procedure

1. Navigate to the Fabric8 Launcher URL in a browser.
2. Follow on-screen instructions to create and launch your booster in Node.js.

3.5.5.3. Authenticating the oc CLI client

To work with boosters on Single-node OpenShift Cluster using the **oc** command-line client, you need to authenticate the client using the token provided by the Single-node OpenShift Cluster web interface.

Prerequisites

- The URL of your running Fabric8 Launcher instance and the user credentials of your Single-node OpenShift Cluster. For more information, see Section 3.5.5.1, “Getting the Fabric8 Launcher tool URL and credentials”.

Procedure

1. Navigate to the Single-node OpenShift Cluster URL in a browser.
2. Click on the question mark icon in the top right-hand corner of the Web console, next to your user name.
3. Select *Command Line Tools* in the drop-down menu.
4. Find the text box that contains the **`oc login …`** command with the hidden token, and click the button next to it to copy its content to your clipboard.
5. Paste the command into a terminal application. The command uses your authentication token to authenticate your **oc** CLI client with your Single-node OpenShift Cluster account.

```
$ oc login OPENSHIFT_URL --token=MYTOKEN
```

3.5.5.4. Deploying the Circuit Breaker booster using the `oc` CLI client

Prerequisites

- The booster application created using Fabric8 Launcher tool on a Single-node OpenShift Cluster. For more information, see Section 3.5.5.2, “Deploying the booster using the Fabric8 Launcher tool”.
- Your Fabric8 Launcher tool URL.
- The **oc** client authenticated. For more information, see Section 3.5.5.3, “Authenticating the **oc** CLI client”.

Procedure

1. Clone your project from GitHub.

```
$ git clone git@github.com:USERNAME/MY_PROJECT_NAME.git
```

   Alternatively, if you downloaded a ZIP file of your project, extract it.

```
$ unzip MY_PROJECT_NAME.zip
```

2. Create a new OpenShift project.

```
$ oc new-project MY_PROJECT_NAME
```

3. Navigate to the root directory of your booster.
4. Use the provided **`start-openshift.sh`** script to start the deployment to OpenShift.

```
$ chmod +x start-openshift.sh
$ ./start-openshift.sh
```

These commands use the Nodeshift **npm** module to install your dependencies, launch the S2I build process on OpenShift, and start the services.

5. Check the status of your booster and ensure your pod is running.

```
$ oc get pods -w
NAME                                READY     STATUS      RESTARTS
AGE
MY_APP_NAME-greeting-1-aaaaa     1/1       Running   0
17s
MY_APP_NAME-greeting-1-deploy    0/1       Completed 0
22s
MY_APP_NAME-name-1-aaaaa         1/1       Running   0
14s
MY_APP_NAME-name-1-deploy        0/1       Completed 0
28s
```

Both the **MY_APP_NAME-greeting-1-aaaaa** and **MY_APP_NAME-name-1-aaaaa** pods should have a status of **Running** once they are fully deployed and started. You should also wait for your pods to be ready before proceeding, which is shown in the **READY** column. For example, **MY_APP_NAME-greeting-1-aaaaa** is ready when the **READY** column is **1/1**. Your specific pod names will vary. The number in the middle will increase with each new build. The letters at the end are generated when the pod is created.

6. Once your booster is deployed and started, determine its route.

Example Route Information

```
$ oc get routes
NAME                 HOST/PORT
PATH      SERVICES         PORT      TERMINATION
MY_APP_NAME-greeting   MY_APP_NAME-greeting-
MY_PROJECT_NAME.OPENSHIFT_HOSTNAME            MY_APP_NAME-greeting
8080                    None
MY_APP_NAME-name       MY_APP_NAME-name-
MY_PROJECT_NAME.OPENSHIFT_HOSTNAME            MY_APP_NAME-name
8080                    None
```

The route information of a pod gives you the base URL which you use to access it. In the example above, you would use **http://MY_APP_NAME-greeting-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME** as the base URL to access the application.

3.5.6. Deploying the Circuit Breaker booster to OpenShift Container Platform

The process of creating and deploying boosters to OpenShift Container Platform is similar to OpenShift Online:

Prerequisites

- The booster created using developers.redhat.com/launch or the Fabric8 Launcher tool.

Procedure

- Follow the instructions in Section 3.5.4, “Deploying the Circuit Breaker booster to OpenShift Online”, only use the URL and user credentials from the OpenShift Container Platform Web Console.

3.5.7. Interacting with the unmodified Node.js Circuit Breaker booster

Once you have the Node.js booster deployed, you have the following services running:

`MY_APP_NAME-name`

Exposes the following endpoints:

- the **`/api/name`** endpoint, which returns a name when this service is working, and an error when this service is set up to demonstrate failure.
- the **`/api/state`** endpoint, which controls the behavior of the **`/api/name`** endpoint and determines whether the service works correctly or demonstrates failure.

`MY_APP_NAME-greeting`

Exposes the following endpoints:

- the **`/api/greeting`** endpoint that you can call to get a personalized greeting response. When you call the **`/api/greeting`** endpoint, it issues a call against the **`/api/name`** endpoint of the **`MY_APP_NAME-name`** service as part of processing your request. The call made against the **`/api/name`** endpoint is protected by the Circuit Breaker.

  If the remote endpoint is available, the **`name`** service responds with an HTTP code **`200`** (**`OK`**) and you receive the following greeting from the **`/api/greeting`** endpoint:

  ```
  {"content":"Hello, World!"}
  ```

  If the remote endpoint is unavailable, the **`name`** service responds with an HTTP code **`500`** (**`Internal server error`**) and you receive a predefined fallback response from the **`/api/greeting`** endpoint:

  ```
  {"content":"Hello, Fallback!"}
  ```

- the **`/api/cb-state`** endpoint, which returns the state of the Circuit Breaker. The state can be:
  - *open* : the circuit breaker is preventing requests from reaching the failed service,
  - *closed*: the circuit breaker is allowing requests to reach the service.

The following steps demonstrate how to verify the availability of the service, simulate a failure and receive a fallback response.

1. Use **`curl`** to execute a **`GET`** request against the **`MY_APP_NAME-greeting`** service. You can also use the **`Invoke`** button in the web interface to do this.

```
$ curl http://MY_APP_NAME-greeting-
MY_PROJECT_NAME.LOCAL_OPENSHIFT_HOSTNAME/api/greeting
{"content":"Hello, World!"}
```

2. To simulate the failure of the **MY_APP_NAME-name** service you can:
   - use the **Toggle** button in the web interface.
   - scale the number of replicas of the pod running the **MY_APP_NAME-name** service down to 0.
   - execute an HTTP **PUT** request against the **/api/state** endpoint of the **MY_APP_NAME-name** service to set its state to **fail**.

     ```
     $ curl -X PUT -H "Content-Type: application/json" -d '{"state":
     "fail"}' http://MY_APP_NAME-name-
     MY_PROJECT_NAME.LOCAL_OPENSHIFT_HOSTNAME/api/state
     ```

3. Invoke the **/api/greeting** endpoint. When several requests on the **/api/name** endpoint fail:
   a. the Circuit Breaker opens,
   b. the state indicator in the web interface changes from **CLOSED** to **OPEN**,
   c. the Circuit Breaker issues a fallback response when you invoke the **/api/greeting** endpoint:

      ```
      $ curl http://MY_APP_NAME-greeting-
      MY_PROJECT_NAME.LOCAL_OPENSHIFT_HOSTNAME/api/greeting
      {"content":"Hello, Fallback!"}
      ```

4. Restore the name **MY_APP_NAME-name** service to availability. To do this you can:
   - use the **Toggle** button in the web interface.
   - scale the number of replicas of the pod running the **MY_APP_NAME-name** service back up to 1.
   - execute an HTTP **PUT** request against the **/api/state** endpoint of the **MY_APP_NAME-name** service to set its state back to **ok**.

     ```
     $ curl -X PUT -H "Content-Type: application/json" -d '{"state":
     "ok"}' http://MY_APP_NAME-name-
     MY_PROJECT_NAME.LOCAL_OPENSHIFT_HOSTNAME/api/state
     ```

5. Invoke the **/api/greeting** endpoint again. When several requests on the **/api/name** endpoint succeed:
   a. the Circuit Breaker closes,
   b. the state indicator in the web interface changes from **OPEN** to **CLOSED**,
   c. the Circuit Breaker issues a returns the **Hello World!** greeting when you invoke the **/api/greeting** endpoint:

```
$ curl http://MY_APP_NAME-greeting-
MY_PROJECT_NAME.LOCAL_OPENSHIFT_HOSTNAME/api/greeting
{"content":"Hello, World!"}
```

3.5.8. Circuit breaker resources

Follow the links below for more background information on the design principles behind the Circuit Breaker pattern

- microservices.io: Microservice Patterns: Circuit Breaker
- Martin Fowler: CircuitBreaker
- Circuit Breaker mission - Spring Boot booster
- Circuit Breaker mission - Eclipse Vert.x booster
- Circuit Breaker mission - Thorntail booster

3.6. SECURED MISSION - NODE.JS BOOSTER

Limitation: Run this booster on a Single-node OpenShift Cluster. You can also use a manual workflow to deploy this booster to OpenShift Online Pro and OpenShift Container Platform. This booster is not currently available on OpenShift Online Starter.

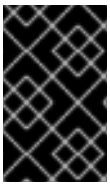
Mission proficiency level: **Advanced**.

The Secured booster secures a REST endpoint using Red Hat SSO. (This booster expands on the REST API Level 0 booster).

Red Hat SSO:

- Implements the Open ID Connect protocol which is an extension of the OAuth 2.0 specification.
- Issues access tokens to provide clients with various access rights to secured resources.

Securing an application with SSO enables you to add security to your applications while centralizing the security configuration.



IMPORTANT

This mission comes with Red Hat SSO pre-configured for demonstration purposes, it does not explain its principles, usage, or configuration. Before using this mission, ensure that you are familiar with the basic concepts related to Red Hat SSO.

3.6.1. The Secured project structure

The SSO booster project contains:

- the sources for the Greeting service, which is the one which we are going to to secure
- a template file (**`service.sso.yaml`**) to deploy the SSO server
- the Keycloak adapter configuration to secure the service

3.6.2. Viewing the booster source code and README

Prerequisites

One of the following:

- Access to developers.redhat.com/launch
- Fabric8 Launcher installed on a Single-node OpenShift Cluster

Procedure

1. Use the Fabric8 Launcher tool to generate your own version of the booster.
2. View the generated GitHub repository or download and extract the ZIP file that contains the booster source code.

Additional resources

- Using developers.redhat.com/launch
- Using the Fabric8 Launcher tool on a Single-node OpenShift Cluster

3.6.3. Red Hat SSO deployment configuration

The `service.sso.yaml` file in this booster contains all OpenShift configuration items to deploy a pre-configured Red Hat SSO server. The SSO server configuration has been simplified for the sake of this exercise and does provide an out-of-the-box configuration, with pre-configured users and security settings. The `service.sso.yaml` file also contains very long lines, and some text editors, such as gedit, may have issues reading this file.

WARNING

It is not recommended to use this SSO configuration in production. Specifically, the simplifications made to the booster security configuration impact the ability to use it in a production environment.

Table 3.5. SSO Booster Simplifications

Change	Reason	Recommendation
The default configuration includes both public and **private keys in the yaml configuration files**.	We did this because the end user can deploy Red Hat SSO module and have it in a usable state without needing to know the internals or how to configure Red Hat SSO.	In production, do not store private keys under source control. They should be added by the server administrator.

Change	Reason	Recommendation
The configured **clients accept any callback url**.	To avoid having a custom configuration for each runtime, we avoid the callback verification that is required by the OAuth2 specification.	An application-specific callback URL should be provided with a valid domain name.
Clients do not require SSL/TLS and the secured applications are not exposed over HTTPS.	The boosters are simplified by not requiring certificates generated for each runtime.	In production a secure application should use HTTPS rather than plain HTTP.
The token timeout has been increased to 10 minutes from the default of 1 minute.	Provides a better user experience when working with the command line examples	From a security perspective, the window an attacker would have to guess the access token is extended. It is recommended to keep this window short as it makes it much harder for a potential attacker to guess the current token.

3.6.4. Red Hat SSO realm model

The **`master`** realm is used to secure this booster. There are two pre-configured application client definitions that provide a model for command line clients and the secured REST endpoint.

There are also two pre-configured users in the Red Hat SSO **`master`** realm that can be used to validate various authentication and authorization outcomes: **`admin`** and **`alice`**.

3.6.4.1. Red Hat SSO users

The realm model for the secured boosters includes two users:

admin

The **`admin`** user has a password of **`admin`** and is the realm administrator. This user has full access to the Red Hat SSO administration console, but none of the role mappings that are required to access the secured endpoints. You can use this user to illustrate the behavior of an authenticated, but unauthorized user.

alice

The **`alice`** user has a password of **`password`** and is the canonical application user. This user will demonstrate successful authenticated and authorized access to the secured endpoints. An example representation of the role mappings is provided in this decoded JWT bearer token:

```
{
  "jti": "0073cfaa-7ed6-4326-ac07-c108d34b4f82",
  "exp": 1510162193,
  "nbf": 0,
  "iat": 1510161593,
  "iss": "https://secure-sso-
```

```
sso.LOCAL_OPENSHIFT_HOSTNAME/auth/realms/master", (1)
  "aud": "demoapp",
  "sub": "c0175ccb-0892-4b31-829f-dda873815fe8",
  "typ": "Bearer",
  "azp": "demoapp",
  "nonce": "90ff5d1a-ba44-45ae-a413-50b08bf4a242",
  "auth_time": 1510161591,
  "session_state": "98efb95a-b355-43d1-996b-0abcb1304352",
  "acr": "1",
  "client_session": "5962112c-2b19-461e-8aac-84ab512d2a01",
  "allowed-origins": [
    "*"
  ],
  "realm_access": {
    "roles": [ (2)
      "booster-admin"
    ]
  },
  "resource_access": { (3)
    "secured-booster-endpoint": {
      "roles": [
        "booster-admin" (4)
      ]
    },
    "account": {
      "roles": [
        "manage-account",
        "view-profile"
      ]
    }
  },
  "name": "Alice InChains",
  "preferred_username": "alice", (5)
  "given_name": "Alice",
  "family_name": "InChains",
  "email": "alice@keycloak.org"
}
```

(1) The **`iss`** field corresponds to the Red Hat SSO realm instance URL that issues the token. This must be configured in the secured endpoint deployments in order for the token to be verified.

(2) The **`roles`** object provides the roles that have been granted to the user at the global realm level. In this case **`alice`** has been granted the **`booster-admin`** role. We will see that the secured endpoint will look to the realm level for authorized roles.

(3) The **`resource_access`** object contains resource specific role grants. Under this object you will find an object for each of the secured endpoints.

(4) The **`resource_access.secured-booster-endpoint.roles`** object contains the roles granted to **`alice`** for the **`secured-booster-endpoint`** resource.

(5) The **`preferred_username`** field provides the username that was used to generate the access token.

3.6.4.2. The application clients

The OAuth 2.0 specification allows you to define a role for application clients that access secured resources on behalf of resource owners. The **master** realm has the following application clients defined:

demoapp

This is a **confidential** type client with a client secret that is used to obtain an access token that contains grants for the **alice** user which enable **alice** to access the Thorntail, Eclipse Vert.x, Node.js and Spring Boot based REST booster deployments.

secured-booster-endpoint

The **secured-booster-endpoint** is a bearer-only type of client that requires a **booster-admin** role for accessing the associated resources, specifically the Greeting service.

3.6.5. Node.js SSO adapter configuration

The SSO adapter is the *client side*, or client to the SSO server, component that enforces security on the web resources. In this specific case, it is the Greeting service.

Enacting Security Example Node.js Code

```
const express = require('express');
const Keycloak = require('keycloak-connect'); (1)
const kc = new Keycloak({}); (2)

const app = express();

app.use(kc.middleware()); (3)

app.use('/api/greeting', kc.protect('booster-admin'), callback); (4)
```

(1) **npm** module keycloak-connect must be installed and **required**. The keycloak-connect module acts as connect middleware, which provides integration with **express**.

(2) Instantiate a new **Keycloak** object and pass in an empty configuration object.

(3) Tells **express** to use Keycloak as middleware.

(4) Enforces that a user must be authenticated and part of the booster-admin role before accessing a resource.

Enacting Security in Keycloak Adapter using `keycloak.json`

```
{
  "realm": "master", (1)
  "resource": "secured-booster-endpoint", (2)
  "realm-public-key": "...", (3)
  "auth-server-url": "${env.SSO_AUTH_SERVER_URL}", (4)
  "ssl-required": "external",
  "disable-trust-manager": true,
  "bearer-only": true, (5)
  "use-resource-role-mappings": true
}
```

1. The security realm to be used.
2. The actual Keycloak *client* configuration.
3. PEM format of the realm public key. You can obtain this from the administration console.
4. The address of the Red Hat SSO server (Interpolation at build time).
5. If enabled the adapter will not attempt to authenticate users, but only verify bearer tokens.

The example Node.js code enables Keycloak and enforces protection of the Greeting service web resource endpoint. The **`keycloak.json`** configures the security adapter to interact with Red Hat SSO.

Additional resources

- For more information about the Node.js Keycloak adapter, see the Keycloak documentation.

3.6.6. Deploying the Secured booster to Single-node OpenShift Cluster

3.6.6.1. Getting the Fabric8 Launcher tool URL and credentials

You need the Fabric8 Launcher tool URL and user credentials to create and deploy boosters on Single-node OpenShift Cluster. This information is provided when the Single-node OpenShift Cluster is started.

Prerequisites

- The Fabric8 Launcher tool installed, configured, and running. For more information, see the Install and Configure the Fabric8 Launcher Tool guide.

Procedure

1. Navigate to the console where you started Single-node OpenShift Cluster.
2. Check the console output for the URL and user credentials you can use to access the running Fabric8 Launcher:

   Example Console Output from a Single-node OpenShift Cluster Startup

```
...
-- Removing temporary directory ... OK
-- Server Information ...
   OpenShift server started.
   The server is accessible via web console at:
       https://192.168.42.152:8443

   You are logged in as:
       User:     developer
       Password: developer

   To login as administrator:
       oc login -u system:admin
```

3.6.6.2. Creating the Secured booster using Fabric8 Launcher

Prerequisites

- The URL and user credentials of your running Fabric8 Launcher instance. For more information, see Section 3.6.6.1, “Getting the Fabric8 Launcher tool URL and credentials”.

Procedure

- Navigate to the Fabric8 Launcher URL in a browser and log in.
- Follow the on-screen instructions to create your booster in Node.js. When asked about which deployment type, select *I will build and run locally.*
- Follow on-screen instructions.
  When done, click the **Download as ZIP file** button and store the file on your hard drive.

3.6.6.3. Authenticating the `oc` CLI client

To work with boosters on Single-node OpenShift Cluster using the **oc** command-line client, you need to authenticate the client using the token provided by the Single-node OpenShift Cluster web interface.

Prerequisites

- The URL of your running Fabric8 Launcher instance and the user credentials of your Single-node OpenShift Cluster. For more information, see Section 3.6.6.1, “Getting the Fabric8 Launcher tool URL and credentials”.

Procedure

1. Navigate to the Single-node OpenShift Cluster URL in a browser.
2. Click on the question mark icon in the top right-hand corner of the Web console, next to your user name.
3. Select *Command Line Tools* in the drop-down menu.
4. Find the text box that contains the **`oc login …`** command with the hidden token, and click the button next to it to copy its content to your clipboard.
5. Paste the command into a terminal application. The command uses your authentication token to authenticate your **oc** CLI client with your Single-node OpenShift Cluster account.

```
$ oc login OPENSHIFT_URL --token=MYTOKEN
```

3.6.6.4. Deploying the Secured booster using the `oc` CLI client

Prerequisites

- The booster application created using the Fabric8 Launcher tool on a Single-node OpenShift Cluster. For more information, see Section 3.6.6.2, “Creating the Secured booster using Fabric8 Launcher”.
- Your Fabric8 Launcher URL.

- The **oc** client authenticated. For more information, see Section 3.6.6.3, “Authenticating the **oc** CLI client”.

Procedure

1. Clone your project from GitHub.

```
$ git clone git@github.com:USERNAME/MY_PROJECT_NAME.git
```

   Alternatively, if you downloaded a ZIP file of your project, extract it.

```
$ unzip MY_PROJECT_NAME.zip
```

2. Create a new OpenShift project.

```
$ oc new-project MY_PROJECT_NAME
```

3. Navigate to the root directory of your booster.
4. Deploy the Red Hat SSO server using the **`service.sso.yaml`** file from your booster ZIP file:

```
$ oc create -f service.sso.yaml
```

5. Use **`npm`** to start the deployment to Single-node OpenShift Cluster.

```
$ npm install && npm run openshift -- \
      -d SSO_AUTH_SERVER_URL=$(oc get route secure-sso -o
jsonpath='{"https://"}{.spec.host}{"/auth\n"}')
```

   These commands install any missing module dependencies, then using the Nodeshift module, deploy the booster on OpenShift.

3.6.7. Deploying the Secured booster to OpenShift Container Platform

In addition to the Single-node OpenShift Cluster, you can create and deploy the booster on OpenShift Container Platform with only minor differences. The most important difference is that you need to create the booster application on Single-node OpenShift Cluster before you can deploy it with OpenShift Container Platform.

Prerequisites

- The booster created using Single-node OpenShift Cluster.

3.6.7.1. Authenticating the oc CLI client

To work with boosters on OpenShift Container Platform using the **oc** command-line client, you need to authenticate the client using the token provided by the OpenShift Container Platform web interface.

Prerequisites

- An account at OpenShift Container Platform.

Procedure

1. Navigate to the OpenShift Container Platform URL in a browser.
2. Click on the question mark icon in the top right-hand corner of the Web console, next to your user name.
3. Select *Command Line Tools* in the drop-down menu.
4. Find the text box that contains the **`oc login …`** command with the hidden token, and click the button next to it to copy its content to your clipboard.
5. Paste the command into a terminal application. The command uses your authentication token to authenticate your **oc** CLI client with your OpenShift Container Platform account.

```
$ oc login OPENSHIFT_URL --token=MYTOKEN
```

3.6.7.2. Deploying the Secured booster using the `oc` CLI client

Prerequisites

- The booster application created using the Fabric8 Launcher tool on a Single-node OpenShift Cluster.
- The **oc** client authenticated. For more information, see Section 3.6.7.1, “Authenticating the **oc** CLI client”.

Procedure

1. Clone your project from GitHub.

```
$ git clone git@github.com:USERNAME/MY_PROJECT_NAME.git
```

   Alternatively, if you downloaded a ZIP file of your project, extract it.

```
$ unzip MY_PROJECT_NAME.zip
```

2. Create a new OpenShift project.

```
$ oc new-project MY_PROJECT_NAME
```

3. Navigate to the root directory of your booster.
4. Deploy the Red Hat SSO server using the **`service.sso.yaml`** file from your booster ZIP file:

```
$ oc create -f service.sso.yaml
```

5. Use **`npm`** to start the deployment to OpenShift Container Platform.

```
$ npm install && npm run openshift -- \
      -d SSO_AUTH_SERVER_URL=$(oc get route secure-sso -o
jsonpath='{"https://"}{.spec.host}{"/auth\n"}')
```

These commands install any missing module dependencies, then using the Nodeshift module, deploy the booster on OpenShift.

3.6.8. Authenticating to the Secured booster API endpoint

The Secured booster provides a default HTTP endpoint that accepts **GET** requests if the caller is authenticated and authorized. The client first authenticates against the Red Hat SSO server and then performs a **GET** request against the Secured booster using the access token returned by the authentication step.

3.6.8.1. Getting the Secured booster API endpoint

When using a client to interact with the booster, you must specify the Secured booster endpoint, which is the *PROJECT_ID* service.

Prerequisites

- The Secured booster deployed and running.
- The **oc** client authenticated.

Procedure

1. In a terminal application, execute the **`oc get routes`** command.
   A sample output is shown in the following table:

Example 3.1. List of Secured endpoints

Name	Host/Port	Path	Services	Port	Termination
secure-sso	secure-sso-myproject.LOCAL_OPENSHIFT_HOSTNAME		secure-sso	<all>	passthrough
PROJECT_ID	PROJECT_ID-myproject.LOCAL_OPENSHIFT_HOSTNAME		PROJECT_ID	<all>	
sso	sso-myproject.LOCAL_OPENSHIFT_HOSTNAME		sso	<all>	

In the above example, the booster endpoint would be **`http://PROJECT_ID-myproject.LOCAL_OPENSHIFT_HOSTNAME`**. **`PROJECT_ID`** is based on the name you entered when generating your booster using developers.redhat.com/launch or the Fabric8 Launcher tool.

3.6.8.2. Authenticating HTTP requests using the command line

Request a token by sending a HTTP POST request to the Red Hat SSO server. In the following example, the jq CLI tool is used to extract the token value from the JSON response.

Prerequisites

- The secured booster endpoint URL. For more information, see Section 3.6.8.1, “Getting the Secured booster API endpoint”.
- The **`jq`** command-line tool (optional). To download the tool and for more information, see https://stedolan.github.io/jq/.

Procedure

1. Request an access token with **`curl`**, the credentials, and **`<SSO_AUTH_SERVER_URL>`** and extract the token from the response with the **`jq`** command:

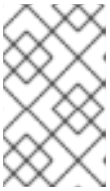
```
curl -sk -X POST
https://<SSO_AUTH_SERVER_URL>/auth/realms/master/protocol/openid-
connect/token \
  -d grant_type=password \
  -d username=alice\
  -d password=password \
  -d client_id=demoapp \
  -d client_secret=1daa57a2-b60e-468b-a3ac-25bd2dc2eadc \
  | jq -r '.access_token'

eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJRek1nbXhZMUhrQnpx
TnR0SnkwMm5jNTNtMGNiWDQxV1hNSTU1MFo4MGVBIn0.eyJqdGkiOiI0NDA3YTliNC04
YWRhLTRlMTctODQ2ZS03YjI5MjMyN2RmYTIiLCJleHAiOjE1MDc3OTM3ODcsIm5iZiI6
MCwiaWF0IjoxNTA3NzkzNzI3LCJpc3MiOiJodHRwczovL3NlY3VyZS1zc28tc3NvLWRl
bW8uYXBwcy5jYWZlLWJhYmUub3JnL2F1dGgvcmVhbG1zL21hc3RlciIsImF1ZCI6ImRl
bW9hcHAiLCJzdWIiOiJjMDE3NWNjYi0wODkyLTRiMzEtODI5Zi1kZGE4NzM4MTVmZTgi
LCJ0eXAiOiJCZWFyZXIiLCJhenAiOiJkZW1vYXBwIiwiYXV0aF90aW1lIjowLCJzZXNz
aW9uX3N0YXRlIjoiMDFjOTkzNGQtNmZmOS00NWYzLWJkNWUtMTU4NDI5ZDZjNDczIiwi
YWNyIjoiMSIsImNsaWVudF9zZXNzaW9uIjoiMzM3Yzk0MTYtYTdlZS00ZWUzLThjZWQt
ODhlODI0MGJjNTAyIiwiYWxsb3dlZC1vcmlnaW5zIjpbIioiXSwicmVhbG1fYWNjZXNz
Ijp7InJvbGVzIjpbImJvb3N0ZXItYWRtaW4iXX0sInJlc291cmNlX2FjY2VzcyI6eyJz
ZWN1cmVkLWJvb3N0ZXItZW5kcG9pbnQiOnsicm9sZXMiOlsiYm9vc3Rlci1hZG1pbiJd
fSwiYWNjb3VudCI6eyJyb2xlcyI6WyJtYW5hZ2UtYWNjb3VudCIsInZpZXctcHJvZmls
ZSJdfX0sIm5hbWUiOiJBbGljZSBJbkNoYWlucyIsInByZWZlcnJlZF91c2VybmFtZSI6
ImFsaWNlIiwiZ2l2ZW5fbmFtZSI6IkFsaWNlIiwiZmFtaWx5X25hbWUiOiJJbkNoYWlu
cyIsImVtYWlsIjoiYWxpY2VAa2V5Y2xvYWsub3JnIn0.mjmZe37enHpigJv0BGuIitOj
-
kfMLPNwYzNd3n0Ax4Nga7KpnfytGyuPSvR4KAG8rzkfBNN9klPYdy7pJEeYlfmnFUkM4
EDrZYgn4qZAznP1Wzy1RfVRdUFi0-
GqFTMPb37o5HRldZZ09QljX_j3GHnoMGXRtYW9RZN4eKkYkcz9hRwgfJoTy2CuwFqeJw
ZYUyXifrfA-JoTr0UmSUed-0NMksGrtJjjPggUGS-
qOn6OgKcmN2vaVAQlxW32y53JqUXctfLQ6DhJzIMYTmOflIPy0sgG1mG7sovQhw1xTg0
vTjdx8zQ-EJcexkj7IivRevRZsslKgqRFWs67jQAFQA
```

<SSO_AUTH_SERVER_URL> is the url of the **secure-sso** service.

The attributes, such as **username**, **password**, and **client_secret** are usually kept secret, but the above command uses the default provided credentials with this booster for demonstration purpose.

If you do not want to use **jq** to extract the token, you can run just the **curl** command and manually extract the access token.



NOTE

The **-sk** option tells curl to ignore failures resulting from self-signed certificates. Do not use this option in a production environment. On macOS, you must have **curl** version **7.56.1** or greater installed. It must also be built with OpenSSL.

1. Invoke the Secured service. Attach the access (bearer) token to the HTTP headers:

```
$ curl -v -H "Authorization: Bearer <TOKEN>"
http://<SERVICE_HOST>/api/greeting

{
    "content": "Hello, World!",
    "id": 2
}
```

Example 3.2. A sample GET Request Headers with an Access (Bearer) Token

```
> GET /api/greeting HTTP/1.1
> Host: <SERVICE_HOST>
> User-Agent: curl/7.51.0
> Accept: */*
> Authorization: Bearer <TOKEN>
```

<SERVICE_HOST> is the URL of the secured booster endpoint. For more information, see Section 3.6.8.1, “Getting the Secured booster API endpoint”.

2. Verify the signature of the access token.
The access token is a JSON Web Token, so you can decode it using the JWT Debugger:

   a. In a web browser, navigate to the JWT Debugger website.

   b. Select **RS256** from the *Algorithm* drop down menu.

NOTE

Make sure the web form has been updated after you made the selection, so it displays the correct RSASHA256(…) information in the Signature section. If it has not, try switching to HS256 and then back to RS256.

   c. Paste the following content in the topmost text box into the *VERIFY SIGNATURE* section:

```
-----BEGIN PUBLIC KEY-----
```

```
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAoETnPmN55xBJjRzN/cs30
OzJ9olkteLVNRjzdTxFOyRtS2ovDfzdhhO9XzUcTMbIsCOAZtSt8K+6yvBXypOSYv
I75EUdypmkcK1KoptqY5KEBQ1KwhWuP7IWQ0fshUwD6jI1QWDfGxfM/h34FvEn/0t
J71xN2P8TI2YanwuDZgosdobx/PAvlGREBGuk4BgmexTOkAdnFxIUQcCkiEZ2C41u
CrxiS4CEe5OX91aK9HKZV4ZJX6vnqMHmdDnsMdO+UFtxOBYZio+a1jP4W3d7J5fGe
iOaXjQCOpivKnP2yU2DPdWmDMyVb67l8DRA+jh0OJFKZ5H2fNgE3II59vdsRwIDAQ
AB
-----END PUBLIC KEY-----
```

NOTE

This is the master realm public key from the Red Hat SSO server deployment of the Secured booster.

d. Paste the **`token`** output from the client output into the *Encoded* box.
The *Signature Verified* sign is displayed on the debugger page.

3.6.8.3. Authenticating HTTP requests using the web interface

In addition to the HTTP API, the secured endpoint also contains a web interface to interact with.

The following procedure is an exercise for you to see how security is enforced, how you authenticate, and how you work with the authentication token.

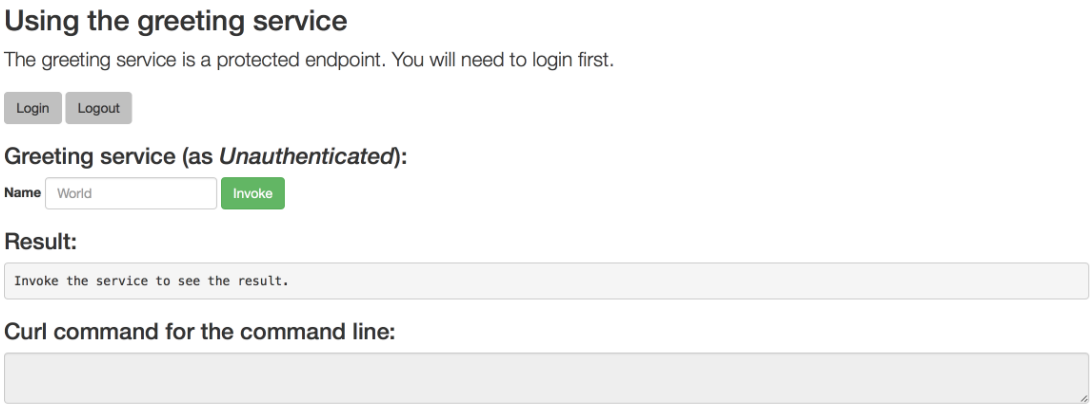
Prerequisites

- The secured endpoint URL. For more information, see Section 3.6.8.1, “Getting the Secured booster API endpoint”.

Procedure

1. In a web browser, navigate to the endpoint URL.
2. Perform an unauthenticated request:
   a. Click the *Invoke* button.

Figure 3.1. Unauthenticated Secured Booster Web Interface



The services responds with an **`HTTP 403 Forbidden`** status code.

NOTE

This is not the correct status code. It should be **`HTTP 401 Unauthorized`**. This issue has been identified and this booster will be updated as soon as it is resolved.

3. Perform an authenticated request as a user:

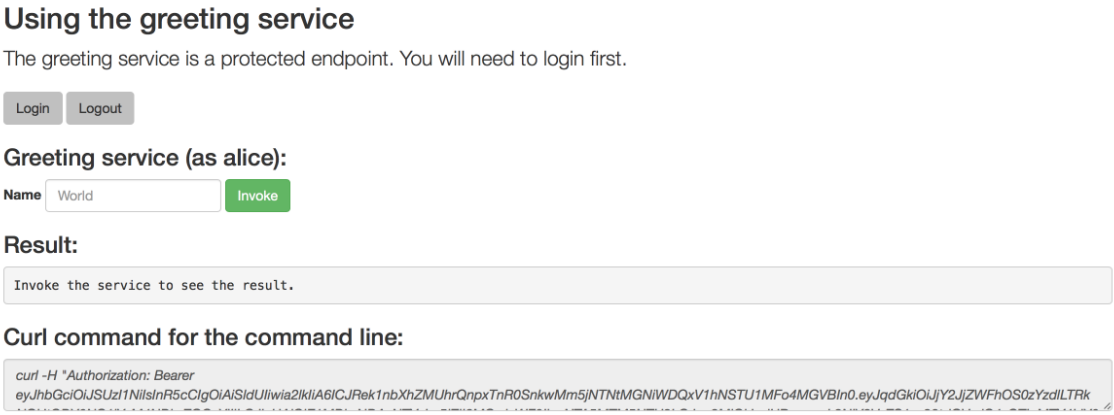
   a. Click the *Login* button to authenticate against Red Hat SSO. You will be redirected to the SSO server.

   b. Log in as the *Alice* user. You will be redirected back to the web interface.

NOTE

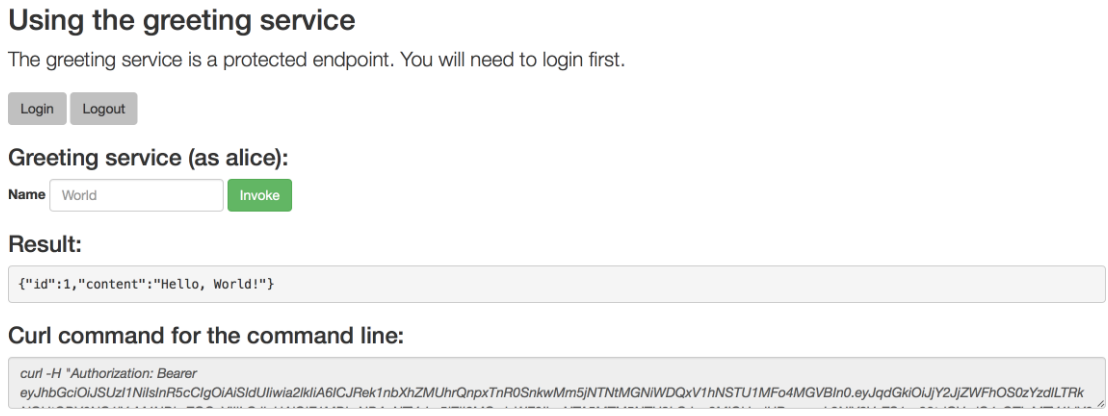
You can see the access (bearer) token in the command line output at the bottom of the page.

Figure 3.2. Authenticated Secured Booster Web Interface (as Alice)



   c. Click *Invoke* again to access the Greeting service.
   Confirm that there is no exception and the JSON response payload is displayed. This means the service accepted your access (bearer) token and you are authorized access to the Greeting service.

Figure 3.3. The Result of an Authenticated Greeting Request (as Alice)



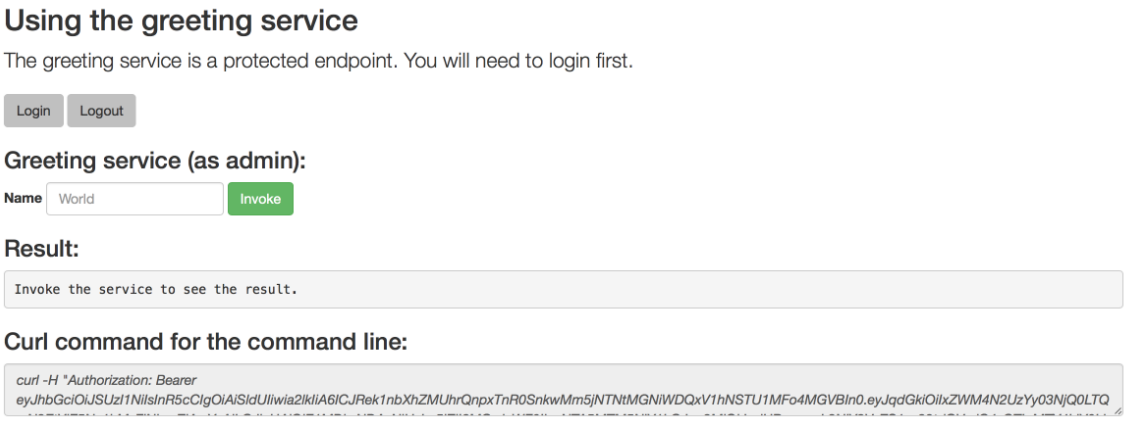
   d. Log out.

4. Perform an authenticated request as an admininstrator:

a. Click the *Invoke* button.
Confirm that this sends an unauthenticated request to the Greeting service.

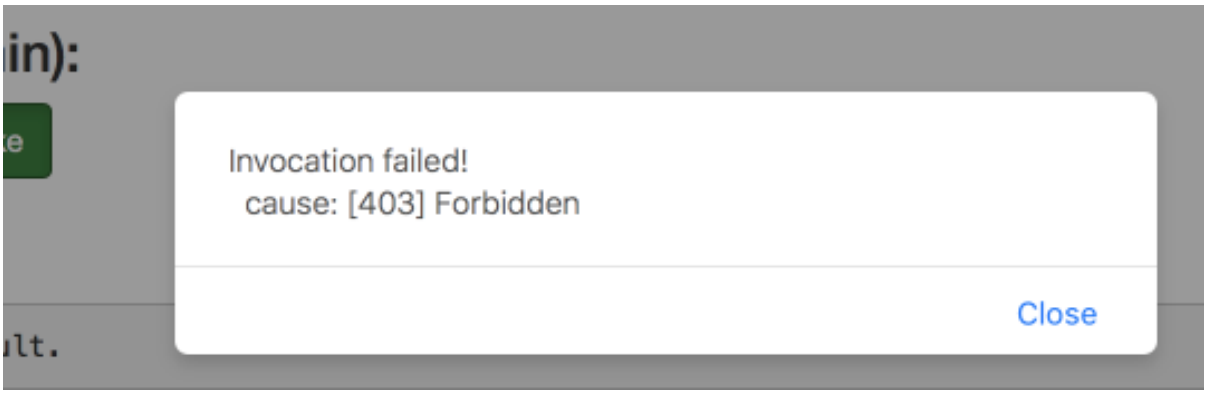
b. Click the *Login* button and log in as the *admin* user.

Figure 3.4. Authenticated Secured Booster Web Interface (as admin)



5. Click the *Invoke* button.
The service responds with an **`HTTP 403 Forbidden`** status code because the *admin* user is not authorized to access the Greeting service.

Figure 3.5. Unauthorized Error Message



3.6.9. Secured SSO resources

Follow the links below for additional information on the principles behind the OAuth2 specification and on securing your applications using Red Hat SSO and Keycloak:

- Aaron Parecki: OAuth2 Simplified
- Red Hat SSO 7.1 Documentation
- Keycloak 3.2 Documentation
- Secured mission - Spring Boot booster
- Secured mission - Eclipse Vert.x booster
- Secured mission - Thorntail booster

3.7. CACHE MISSION - NODE.JS BOOSTER

Limitation: Run this booster on a Single-node OpenShift Cluster. You can also use a manual workflow to deploy this booster to OpenShift Online Pro and OpenShift Container Platform. This booster is not currently available on OpenShift Online Starter.

Mission proficiency level: **Advanced**.

The Cache mission demonstrates how to use a cache to increase the response time of applications.

This mission shows you how to:

- Deploy a cache to OpenShift.
- Use a cache within an application.

3.7.1. How caching works and when you need it

Caches allows you to store information and access it for a given period of time. You can access information in a cache faster or more reliably than repeatedly calling the original service. A disadvantage of using a cache is that the cached information is not up to date. However, that problem can be reduced by setting an *expiration* or TTL (time to live) on each value stored in the cache.

> **Example 3.3. Caching example**
>
> Assume you have two applications: *service1* and *service2*:
>
> - *Service1* depends on a value from *service2*.
>   - If the value from *service2* infrequently changes, *service1* could cache the value from *service2* for a period of time.
>   - Using cached values can also reduce the number of times *service2* is called.
> - If it takes *service1* 500 ms to retrieve the value directly from *service2*, but 100 ms to retrieve the cached value, *service1* would save 400 ms by using the cached value for each cached call.
> - If *service1* would make uncached calls to *service2* 5 times per second, over 10 seconds, that would be 50 calls.
> - If *service1* started using a cached value with a TTL of 1 second instead, that would be reduced to 10 calls over 10 seconds.

How the Cache mission works

1. The *cache*, *cute name*, and *greeting* services are deployed and exposed.
2. User accesses the web frontend of the *greeting* service.
3. User invokes the *greeting* HTTP API using a button on the web frontend.
4. The *greeting* service depends on a value from the *cute name* service.
   - The *greeting* service first checks if that value is stored in the *cache* service. If it is, then the cached value is returned.

- If the value is not cached, the *greeting* service calls the *cute name* service, returns the value, and stores the value in the *cache* service with a TTL of 5 seconds.

5. The web front end displays the response from the *greeting* service as well as the total time of the operation.
6. User invokes the service multiple times to see the difference between cached and uncached operations.
   - Cached operations are significantly faster than uncached operations.
   - User can force the cache to be cleared before the TTL expires.

3.7.2. Viewing the booster source code and README

Prerequisites

One of the following:

- Access to developers.redhat.com/launch
- Fabric8 Launcher installed on a Single-node OpenShift Cluster

Procedure

1. Use the Fabric8 Launcher tool to generate your own version of the booster.
2. View the generated GitHub repository or download and extract the ZIP file that contains the booster source code.

Additional resources

- Using developers.redhat.com/launch
- Using the Fabric8 Launcher tool on a Single-node OpenShift Cluster

3.7.3. Deploying the Cache booster to OpenShift Online

Use one of the following options to execute the Cache booster on OpenShift Online.

- Use developers.redhat.com/launch
- Use the **oc** CLI client

Although each method uses the same **oc** commands to deploy your application, using developers.redhat.com/launch provides an automated booster deployment workflow that executes the **oc** commands for you.

3.7.3.1. Deploying the booster using developers.redhat.com/launch

Prerequisites

- An account at OpenShift Online.

Procedure

1. Navigate to the developers.redhat.com/launch URL in a browser and log in.
2. Follow on-screen instructions to create and launch your booster in Node.js.

3.7.3.2. Authenticating the oc CLI client

To work with boosters on OpenShift Online using the **oc** command-line client, you need to authenticate the client using the token provided by the OpenShift Online web interface.

Prerequisites

- An account at OpenShift Online.

Procedure

1. Navigate to the OpenShift Online URL in a browser.
2. Click on the question mark icon in the top right-hand corner of the Web console, next to your user name.
3. Select *Command Line Tools* in the drop-down menu.
4. Find the text box that contains the **`oc login …`** command with the hidden token, and click the button next to it to copy its content to your clipboard.
5. Paste the command into a terminal application. The command uses your authentication token to authenticate your **oc** CLI client with your OpenShift Online account.

```
$ oc login OPENSHIFT_URL --token=MYTOKEN
```

3.7.3.3. Deploying the Cache booster using the oc CLI client

Prerequisites

- The booster application created using developers.redhat.com/launch. For more information, see Section 3.7.3.1, “Deploying the booster using developers.redhat.com/launch”.
- The **oc** client authenticated. For more information, see Section 3.7.3.2, “Authenticating the **oc** CLI client”.

Procedure

1. Clone your project from GitHub.

```
$ git clone git@github.com:USERNAME/MY_PROJECT_NAME.git
```

Alternatively, if you downloaded a ZIP file of your project, extract it.

```
$ unzip MY_PROJECT_NAME.zip
```

2. Create a new project.

```
$ oc new-project MY_PROJECT_NAME
```

3. Navigate to the root directory of your booster.
4. Deploy the cache service.

```
$ oc apply -f service.cache.yml
```

5. Use **`start-openshift.sh`** to start the deployment to OpenShift.

```
$ ./start-openshift.sh
```

6. Check the status of your booster and ensure your pod is running.

```
$ oc get pods -w
NAME                                READY     STATUS       RESTARTS
AGE
cache-server-123456789-aaaaa             1/1       Running     0
8m
MY_APP_NAME-cutename-1-bbbbb       1/1       Running     0
4m
MY_APP_NAME-cutename-s2i-1-build   0/1       Completed   0
7m
MY_APP_NAME-greeting-1-ccccc       1/1       Running     0
3m
MY_APP_NAME-greeting-s2i-1-build   0/1       Completed   0
3m
```

Your 3 pods should have a status of **`Running`** once they are fully deployed and started.

7. Once your booster is deployed and started, determine its route.

Example Route Information

```
$ oc get routes
NAME                 HOST/PORT
PATH      SERVICES        PORT      TERMINATION
MY_APP_NAME-cutename   MY_APP_NAME-cutename-
MY_PROJECT_NAME.OPENSHIFT_HOSTNAME             MY_APP_NAME-cutename
8080                    None
MY_APP_NAME-greeting   MY_APP_NAME-greeting-
MY_PROJECT_NAME.OPENSHIFT_HOSTNAME             MY_APP_NAME-greeting
8080                    None
```

The route information of a pod gives you the base URL which you use to access it. In the example above, you would use **`http://MY_APP_NAME-greeting-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME`** as the base URL to access the greeting service.

3.7.4. Deploying the Cache booster to Single-node OpenShift Cluster

Use one of the following options to execute the Cache booster locally on Single-node OpenShift Cluster:

- Using Fabric8 Launcher
- Using the **`oc`** CLI client

Although each method uses the same **oc** commands to deploy your application, using Fabric8 Launcher provides an automated booster deployment workflow that executes the **oc** commands for you.

3.7.4.1. Getting the Fabric8 Launcher tool URL and credentials

You need the Fabric8 Launcher tool URL and user credentials to create and deploy boosters on Single-node OpenShift Cluster. This information is provided when the Single-node OpenShift Cluster is started.

Prerequisites

- The Fabric8 Launcher tool installed, configured, and running. For more information, see the Install and Configure the Fabric8 Launcher Tool guide.

Procedure

1. Navigate to the console where you started Single-node OpenShift Cluster.
2. Check the console output for the URL and user credentials you can use to access the running Fabric8 Launcher:

   Example Console Output from a Single-node OpenShift Cluster Startup

```
...
-- Removing temporary directory ... OK
-- Server Information ...
   OpenShift server started.
   The server is accessible via web console at:
       https://192.168.42.152:8443

   You are logged in as:
       User:     developer
       Password: developer

   To login as administrator:
       oc login -u system:admin
```

3.7.4.2. Deploying the booster using the Fabric8 Launcher tool

Prerequisites

- The URL of your running Fabric8 Launcher instance and the user credentials of your Single-node OpenShift Cluster. For more information, see Section 3.7.4.1, “Getting the Fabric8 Launcher tool URL and credentials”.

Procedure

1. Navigate to the Fabric8 Launcher URL in a browser.
2. Follow on-screen instructions to create and launch your booster in Node.js.

3.7.4.3. Authenticating the oc CLI client

To work with boosters on Single-node OpenShift Cluster using the **oc** command-line client, you need to authenticate the client using the token provided by the Single-node OpenShift Cluster web interface.

Prerequisites

- The URL of your running Fabric8 Launcher instance and the user credentials of your Single-node OpenShift Cluster. For more information, see Section 3.7.4.1, “Getting the Fabric8 Launcher tool URL and credentials”.

Procedure

1. Navigate to the Single-node OpenShift Cluster URL in a browser.
2. Click on the question mark icon in the top right-hand corner of the Web console, next to your user name.
3. Select *Command Line Tools* in the drop-down menu.
4. Find the text box that contains the **`oc login …`** command with the hidden token, and click the button next to it to copy its content to your clipboard.
5. Paste the command into a terminal application. The command uses your authentication token to authenticate your **oc** CLI client with your Single-node OpenShift Cluster account.

```
$ oc login OPENSHIFT_URL --token=MYTOKEN
```

3.7.4.4. Deploying the Cache booster using the `oc` CLI client

Prerequisites

- The booster application created using Fabric8 Launcher tool on a Single-node OpenShift Cluster. For more information, see Section 3.7.4.2, “Deploying the booster using the Fabric8 Launcher tool”.
- Your Fabric8 Launcher tool URL.
- The **oc** client authenticated. For more information, see Section 3.7.4.3, “Authenticating the **oc** CLI client”.

Procedure

1. Clone your project from GitHub.

```
$ git clone git@github.com:USERNAME/MY_PROJECT_NAME.git
```

Alternatively, if you downloaded a ZIP file of your project, extract it.

```
$ unzip MY_PROJECT_NAME.zip
```

2. Create a new project.

```
$ oc new-project MY_PROJECT_NAME
```

3. Navigate to the root directory of your booster.
4. Deploy the cache service.

```
$ oc apply -f service.cache.yml
```

5. Use **`start-openshift.sh`** to start the deployment to OpenShift.

```
$ ./start-openshift.sh
```

6. Check the status of your booster and ensure your pod is running.

```
$ oc get pods -w
NAME                                READY     STATUS      RESTARTS
AGE
cache-server-123456789-aaaaa             1/1       Running     0
8m
MY_APP_NAME-cutename-1-bbbbb       1/1       Running     0
4m
MY_APP_NAME-cutename-s2i-1-build   0/1       Completed   0
7m
MY_APP_NAME-greeting-1-ccccc       1/1       Running     0
3m
MY_APP_NAME-greeting-s2i-1-build   0/1       Completed   0
3m
```

Your 3 pods should have a status of **`Running`** once they are fully deployed and started.

7. Once your booster is deployed and started, determine its route.

Example Route Information

```
$ oc get routes
NAME                 HOST/PORT
PATH      SERVICES        PORT      TERMINATION
MY_APP_NAME-cutename   MY_APP_NAME-cutename-
MY_PROJECT_NAME.OPENSHIFT_HOSTNAME             MY_APP_NAME-cutename
8080                    None
MY_APP_NAME-greeting   MY_APP_NAME-greeting-
MY_PROJECT_NAME.OPENSHIFT_HOSTNAME             MY_APP_NAME-greeting
8080                    None
```

The route information of a pod gives you the base URL which you use to access it. In the example above, you would use **`http://MY_APP_NAME-greeting-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME`** as the base URL to access the greeting service.

3.7.5. Deploying the Cache booster to OpenShift Container Platform

The process of creating and deploying boosters to OpenShift Container Platform is similar to OpenShift Online:

Prerequisites

- The booster created using developers.redhat.com/launch or the Fabric8 Launcher tool.

Procedure

- Follow the instructions in Section 3.7.3, “Deploying the Cache booster to OpenShift Online”, only use the URL and user credentials from the OpenShift Container Platform Web Console.

3.7.6. Interacting with the unmodified Cache booster

Prerequisites

- Your application deployed

Procedure

1. Navigate to the **`greeting`** service using your browser.
2. Click *Invoke the service* once.
   Notice the **`duration`** value is above **`2000`**. Also notice the cache state has changed form **`No cached value`** to **`A value is cached`**.
3. Wait 5 seconds and notice cache state has changed back to **`No cached value`**.
   The TTL for the cached value is set to 5 seconds. When the TTL expires, the value is no longer cached.
4. Click *Invoke the service* once more to cache the value.
5. Click *Invoke the service* a few more times over the course of a few seconds while cache state is **`A value is cached`**.
   Notice a significantly lower **`duration`** value since it is using a cached value. If you click *Clear the cache*, the cache is emptied.

3.7.7. Caching resources

More background and related information on caching can be found here:

- Cache mission - Spring Boot booster
- Cache mission - Eclipse Vert.x booster
- Cache mission - Thorntail booster

CHAPTER 4. DEBUGGING

This section contains information about debugging your Node.js–based application and using debug logging in both local and remote deployments.

4.1. REMOTE DEBUGGING

To remotely debug an application, you need to start it in a debugging mode and attach a debugger to it.

4.1.1. Starting your application locally and attaching the native debugger

The native debugger enables you to debug your Node.js–based application using the built-in debugging client.

Prerequisites

- An application you want to debug.

Procedure

1. Start the application with the debugger enabled.
   The native debugger is automatically attached and provides a debugging prompt.

   Example application with the debugger enabled

   ```
   $ node inspect app.js
   < Debugger listening on ws://127.0.0.1:9229/12345678-aaaa-bbbb-cccc-
   0123456789ab
   < For help see https://nodejs.org/en/docs/inspector
   < Debugger attached.
   ...
   debug>
   ```

   If you have a different entry point for your application, you need to change the command to specify that entry point:

   ```
   $ node inspect path/to/entrypoint
   ```

   For example, when using the express generator to create your application, the entry point is set to **`./bin/www`** by default. Some of the boosters, such as the REST API Level 0 booster, use **`./bin/www`** as an entry point.

2. Use the debugger prompt to perform debugging commands.

4.1.2. Starting your application locally and attaching the V8 inspector

The V8 inspector enables you to debug your Node.js–based application using other tools, such as Chrome DevTools, that use the Chrome Debugging Protocol.

Prerequisites

- An application you want to debug.

- The V8 inspector installed, such as the one provided in the Google Chrome Browser.

Procedure

1. Start your application with the V8 inspector integration enabled.

```
$ node --inspect app.js
```

   If you have a different entry point for your application, you need to change the command to specify that entry point:

```
$ node --inspect path/to/entrypoint
```

   For example, when using the express generator to create your application, the entry point is set to **`./bin/www`** by default. Some of the boosters, such as the REST API Level 0 booster, use **`./bin/www`** as an entry point.

2. Attach the V8 inspector and perform debugging commands.
   For example, if using Google Chrome:

   a. Navigate to **`chrome://inspect`**.

   b. Select your application from below *Remote Target*.

   c. You can now see the source of your application and can perform debugging actions.

4.1.3. Starting your application on OpenShift in debugging mode

To debug your Node.js-based application on OpenShift remotely, you must set the **`NODE_ENV`** environment variable inside the container to **`development`** and configure port forwarding so that you can connect to your application from a remote debugger.

Prerequisites

- Your application running on OpenShift.
- The **oc** binary installed on your machine.
- The ability to execute the **`oc port-forward`** command in your target OpenShift environment.

Procedure

1. Using the **oc** command, list the available deployment configurations:

```
$ oc get dc
```

2. Set the **`NODE_ENV`** environment variable in the deployment configuration of your application to **`development`** to enable debugging. For example:

```
$ oc set env dc/MY_APP_NAME NODE_ENV=development
```

3. Redeploy the application if it is not set to redeploy automatically on configuration change. For example:

```
$ oc rollout latest dc/MY_APP_NAME
```

4. Configure port forwarding from your local machine to the application pod:

   a. List the currently running pods and find one containing your application:

   ```
   $ oc get pod
   NAME                              READY     STATUS      RESTARTS
   AGE
   MY_APP_NAME-3-1xrsp          0/1       Running     0          6s
   ...
   ```

   b. Configure port forwarding:

   ```
   $ oc port-forward MY_APP_NAME-3-1xrsp $LOCAL_PORT_NUMBER:5858
   ```

   Here, **$LOCAL_PORT_NUMBER** is an unused port number of your choice on your local machine. Remember this number for the remote debugger configuration.

5. Attach the V8 inspector and perform debugging commands.
   For example, if using Google Chrome:

   a. Navigate to **chrome://inspect**.

   b. Click *Configure*.

   c. Add **127.0.0.1:$LOCAL_PORT_NUMBER**.

   d. Click *Done*.

   e. Select your application from below *Remote Target*.

   f. You can now see the source of your application and can perform debugging actions.

6. When you are done debugging, unset the **NODE_ENV** environment variable in your application pod. For example:

   ```
   $ oc set env dc/MY_APP_NAME NODE_ENV-
   ```

4.2. DEBUG LOGGING

Debug logging is a way to add detailed information to the application log when debugging. This allows you to:

- Keep minimal logging output during normal operation of the application for improved readability and reduced disk space usage.
- View detailed information about the inner workings of the application when resolving issues.

4.2.1. Add debug logging

This example uses the debug package, but there are also other packages available that can handle debug logging.

Prerequisites

- An application you want to debug. For example, a booster.

Procedure

1. Add the **debug** logging definition.

```
const debug = require('debug')('mybooster');
```

2. Add debug statements.

```
app.use('/api/greeting', (request, response) => {
  const name = request.query ? request.query.name : undefined;
  //log name in debugging
  debug('name: '+name);
  response.send({content: `Hello, ${name || 'World'}`});
});
```

3. Add the debug module to **`package.json`**.

```
...
"dependencies": {
    "debug": "^3.1.0"
  }
```

Depending on your application, this module may already be included. For example, when using the express generator to create your application, the **debug** module is already added to **`package.json`**. Some of the boosters, such as the REST API Level 0 booster, already have the **debug** module in the **`package.json`**.

4. Install the application dependencies.

```
$ npm install
```

4.2.2. Accessing debug logs on localhost

Use the **`DEBUG`** environment variable when starting your application to enable debug logging.

Prerequisites

- An application with debug logging.

Procedure

1. Set the **`DEBUG`** environment variable when starting your application to enable debug logging.

```
$ DEBUG=mybooster npm start
```

The **debug** module can use wildcards to filter debugging messages. This is set using the **`DEBUG`** environment variable.

2. Test your application to invoke debug logging.

For example, when debug logging in the REST API Level 0 booster is set to log the **name** variable in the **/api/greeting** method:

```
$ curl http://localhost:8080/api/greeting?name=Sarah
```

3. View your application logs to see your debug messages.

```
mybooster name: Sarah +3m
```

4.2.3. Accessing Node.js debug logs on OpenShift

Use the the **DEBUG** environment variable in your application pod in OpenShift to enable debug logging.

Prerequisites

- An application with debug logging.
- The **oc** CLI client installed.

Procedure

1. Use the **oc** CLI client to log into your OpenShift instance.

```
$ oc login ...
```

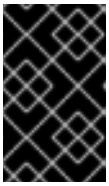
2. Deploy your application to OpenShift.

```
$ npm run openshift
```

This runs the **openshift** npm script, which wraps direct calls to nodeshift.

3. Find the name of your pod and follow the logs to watch it start.

```
$ oc get pods
....
$ oc logs -f pod/POD_NAME
```



IMPORTANT

After your pod has started, leave this command running and execute the remaining steps in a new terminal window. This allows you to *follow* the logs and see new entries made to it.

4. Test your application.
For example, if you had debug logging in the REST API Level 0 booster to log the **name** variable in the **/api/greeting** method:

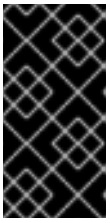
```
$ oc get routes
...
$ curl $APPLICATION_ROUTE/api/greeting?name=Sarah
```

5. Return to your pod logs and notice there are no debug logging messages in the logs.
6. Set the **DEBUG** environment variable to enable debug logging.

```
$ oc get dc
...
$ oc set env dc DC_NAME DEBUG=mybooster
```

7. Return to your pod logs to watch the update roll out.
   After the update has rolled out, your pod will stop and you will no longer be following the logs.
8. Find the name of your new pod and follow the logs.

```
$ oc get pods
....
$ oc logs -f pod/POD_NAME
```



IMPORTANT

After your pod has started, leave this command running and execute the remaining steps in a different terminal window. This allows you to *follow* the logs and see new entries made to it. Specifically, the logs will show your debug messages.

9. Test the application to invoke debug logging.

```
$ oc get routes
...
$ curl $APPLICATION_ROUTE/api/greeting?name=Sarah
```

10. Return to your pod logs to see the debug messages.

```
...
mybooster name: Sarah +3m
```

To disable debug logging, remove the **DEBUG** environment variable from the pod:

```
$ oc set env dc DC_NAME DEBUG-
```

Additional resources

More details on environment variables are available in the OpenShift documentation.

CHAPTER 5. DEVELOPING AN APPLICATION FOR THE NODE.JS RUNTIME

5.1. NODE.JS RUNTIME API

The Node.js runtime provides the core Node.js API which is documented in the Node.js API documentation.

5.2. CREATING A NODE.JS APPLICATION

In addition to using a booster, you can also create new Node.js applications from scratch and deploy them to OpenShift.

5.2.1. Creating an application

Prerequisites

- **npm** installed.

Procedure

1. Create the application folder.

```
$ mkdir myApp
```

2. Initialize your application with npm.
   The rest of this example assumes the entry point is **app.js**, which you are prompted to set when running **npm init**.

```
$ cd myApp
$ npm init
```

3. Create the entry point in a new file called **app.js**.

   Example app.js

```
const http = require('http');

const server = http.createServer((request, response) => {
  response.statusCode = 200;
  response.setHeader('Content-Type', 'application/json');

  const greeting = {content: 'Hello, World!'};

  response.write(JSON.stringify(greeting));
  response.end();
});

server.listen(8080, () => {
  console.log('Server running at http://localhost:8080');
});
```

4. Start your application.

```
$ node app.js
Server running at http://localhost:8080
```

5. Using **curl** or your browser, verify your application is running at **http://localhost:8080**.

```
$ curl http://localhost:8080
{"content":"Hello, World!"}
```

5.2.2. Deploying an application to OpenShift

Prerequisites

- The **oc** CLI client installed.
- **npm** installed.

Procedure

1. Add **nodeshift** to your application.

```
$ npm install nodeshift --save
```

2. Add the **openshift** and **start** entries to the **scripts** section in **package.json**.

```
{
  "name": "myApp",
  "version": "1.0.0",
  "description": "",
  "main": "app.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "openshift": "nodeshift --strictSSL=false --
metadata.out=deployment-metadata.json --build.forcePull=true --
dockerImage=registry.access.redhat.com/rhoar-nodejs/nodejs-8",
    "start": "node app.js",
    ...
  }
  ...
}
```

The **openshift** script uses **nodeshift** to deploy the application to OpenShift.

3. *Optional*: Add a **files** section in **package.json**.

```
{
  "name": "myApp",
  "version": "1.0.0",
  "description": "",
  "main": "app.js",
  "scripts": {
    ...
```

```
  },
  "files": [
    "package.json",
    "app.js"
  ]
  ...
}
```

The **files** section tells **nodeshift** what files and directories to include when deploying to OpenShift. **nodeshift** uses the **node-tar** module to create a tar file based on the files and directories you list in the **files** section. This tar file is used when **nodeshift** deploys your application to OpenShift. If the **files** section is not specified, **nodeshift** will send the entire current directory, excluding:

- **node_modules/**
- **.git/**
- **tmp/**
  It is recommended that you include a **files** section in **package.json** to avoid including unnecessary files when deploying to OpenShift.

4. Log in to your OpenShift instance with the **oc** client.

```
$ oc login ...
```

5. Use **nodeshift** to deploy the application to OpenShift.

```
$ npm run openshift
```

6. Expose the application in OpenShift using a route.

Example for exposing a service using a route

```
$ oc expose service myapp
```

Optionally, you can create a **.nodeshift** directory at the root of your project to include deployment yaml files. These files will create items such as routes when deploying your application to OpenShift using nodeshift. For example, to add a route during deployment, you could create a **.nodeshift/route.yaml** with the following:

Example .nodeshift/route.yaml

```
apiVersion: v1
kind: Route
metadata:
  name: myapp
spec:
  port:
    targetPort: 8080
  to:
    kind: Service
    name: myapp
```

5.3. DEPLOYING AN EXISTING NODE.JS APPLICATION TO OPENSHIFT

You can deploy an existing Node.js application to OpenShift using the steps in Section 5.2.2, “Deploying an application to OpenShift”.

In addition, you must verify the following items when deploying an existing application:

- Ensure the **`files`** section of **`package.json`** lists all files and directories you need to deploy with your application.
- Ensure the **`start`** entry in the **`scripts`** section of **`package.json`** correctly starts your application.
- Ensure all the ports used by your application are correctly exposed when configuring your routes.

APPENDIX A. ABOUT NODESHIFT

Nodeshift is a module for running OpenShift deployments with Node.js projects.

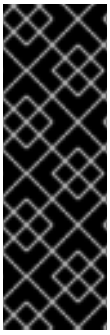
IMPORTANT

Nodeshift assumes you have the **oc** CLI client installed, and you are logged into your OpenShift cluster. Nodeshift also uses the current project the **oc** CLI client is using.

Nodeshift uses resource files in the `.nodeshift` folder located at the root of the project to handle creating OpenShift Routes, Services and DeploymentConfigs. More details on Nodeshift are available on the Nodeshift project page.

APPENDIX B. UPDATING THE DEPLOYMENT CONFIGURATION OF A BOOSTER

The deployment configuration for a booster contains information related to deploying and running the booster in OpenShift, such as route information or readiness probe location. The deployment configuration of a booster is stored in a set of YAML files. For boosters that use the Fabric8 Maven Plugin, the YAML files are located in the **`src/main/fabric8/`** directory. For boosters using Nodeshift, the YAML files are located in the **`.nodeshift`** directory.



IMPORTANT

The deployment configuration files used by the Fabric8 Maven Plugin and Nodeshift do not have to be full OpenShift resource definitions. Both Fabric8 Maven Plugin and Nodeshift can take the deployment configuration files and add some missing information to create a full OpenShift resource definition. The resource definitions generated by the Fabric8 Maven Plugin are available in the **`target/classes/META-INF/fabric8/`** directory. The resource definitions generated by Nodeshift are available in the **`tmp/nodeshift/resource/`** directory.

Prerequisites

- An existing booster project.
- The **oc** CLI client installed.

Procedure

1. Edit an existing YAML file or create an additional YAML file with your configuration update.
   - For example, if your booster already has a YAML file with a **`readinessProbe`** configured, you could change the **`path`** value to a different available path to check for readiness:

```
spec:
  template:
    spec:
      containers:
        readinessProbe:
          httpGet:
            path: /path/to/probe
            port: 8080
            scheme: HTTP
...
```

   - If a **`readinessProbe`** is not configured in an existing YAML file, you can also create a new YAML file in the same directory with the **`readinessProbe`** configuration.
2. Deploy the updated version of your booster using Maven or npm.
3. Verify that your configuration updates show in the deployed version of your booster.

```
$ oc export all --as-template='my-template'

apiVersion: v1
kind: Template
```

```
metadata:
  creationTimestamp: null
  name: my-template
objects:
- apiVersion: v1
  kind: DeploymentConfig
  ...
  spec:
    ...
    template:
      ...
      spec:
        containers:
          ...
          livenessProbe:
            failureThreshold: 3
            httpGet:
              path: /path/to/different/probe
              port: 8080
              scheme: HTTP
            initialDelaySeconds: 60
            periodSeconds: 30
            successThreshold: 1
            timeoutSeconds: 1
          ...
```

Additional resources

If you updated the configuration of your application directly using the web-based console or the **oc** CLI client, export and add these changes to your YAML file. Use the **`oc export all`** command to show the configuration of your deployed application.

APPENDIX C. CONFIGURING A JENKINS FREESTYLE PROJECT TO DEPLOY YOUR NODE.JS APPLICATION WITH NODESHIFT

Similar to using nodeshift from your local host to deploy a Node.js application, you can configure Jenkins to use nodeshift to deploy a Node.js application.

Prerequisites

- Access to an OpenShift cluster.
- The Jenkins container image running on same OpenShift cluster.
- The Node.js plugin installed on your Jenkins server.
- A Node.js application configured to use nodeshift and the Red Hat base image.

  Example using the Red Hat base image with nodeshift

  ```
  $ nodeshift --dockerImage=registry.access.redhat.com/rhoar-
  nodejs/nodejs-8 ...
  ```

- The source of the application available in GitHub.

Procedure

1. Create a new OpenShift project for your application:
   a. Open the OpenShift Web console and log in.
   b. Click *Create Project* to create a new OpenShift project.
   c. Enter the project information and click *Create*.
2. Ensure Jenkins has access to that project.
   For example, if you configured a service account for Jenkins, ensure that account has **`edit`** access to the project of your application.
3. Create a new freestyle Jenkins project on your Jenkins server:
   a. Click *New Item*.
   b. Enter a name, choose *Freestyle project*, and click *OK*.
   c. Under *Source Code Management*, choose *Git* and add the GitHub url of your application.
   d. Under *Build Environment*, make sure *Provide Node & npm bin/ folder to PATH*is checked and the Node.js environment is configured.
   e. Under *Build*, choose *Add build step* and select **`Execute Shell`**.
   f. Add the following to the *Command* area:

   ```
   npm install -g nodeshift
   nodeshift --strictSSL=false --
   ```

```
dockerImage=registry.access.redhat.com/rhoar-nodejs/nodejs-8 --
namespace=MY_PROJECT
```

Substitute **MY_PROJECT** with the name of the OpenShift project for your application.

g. Click *Save*.

4. Click *Build Now* from the main page of the Jenkins project to verify your application builds and deploys to the OpenShift project for your application.
You can also verify that your application is deployed by opening the route in the OpenShift project of the application.

Next steps

- Consider adding GITSCM polling or using the **Poll SCM** build trigger. These options enable builds to run every time a new commit is pushed to the GitHub repository.
- Consider adding nodeshift as a global package when configuring the Node.js plugin. This allows you to omit **npm install -g nodeshift** when adding your **Execute Shell** build step.
- Consider adding a build step that executes tests before deploying.

APPENDIX D. BREAKDOWN OF PACKAGE.JSON PROPERTIES

nodejs-rest-http/package.json

```
{
  "name": "nodejs-rest-http",
  "version": "1.1.1",
  "author": "Red Hat, Inc.",
  "license": "Apache-2.0",
  "scripts": {
    "test": "tape test/*.js | tap-spec", (1)
    "lint": "eslint test/*.js app.js bin/*",
    "prepare": "nsp check",
    "coverage": "nyc npm test",
    "coveralls": "nyc npm test && nyc report --reporter=text-lcov |
coveralls",
    "ci": "npm run lint && npm run coveralls",
    "dependencyCheck": "szero . --ci",
    "release": "standard-version",
    "openshift": "nodeshift --strictSSL=false --nodeVersion=8.x", (2)
    "postinstall": "license-reporter report && license-reporter save --xml
licenses.xml",
    "start": "node ." (3)
  },
  "main": "./bin/www", (4)
  "repository": {
    "type": "git",
    "url": "git://github.com/nodeshift-starters/nodejs-rest-http.git"
  },
  "files": [ (5)
    "package.json",
    "app.js",
    "public",
    "bin",
    "LICENSE",
    "licenses"
  ],
  "bugs": {
    "url": "https://github.com/nodeshift-starters/nodejs-rest-http/issues"
  },
  "homepage": "https://github.com/nodeshift-starters/nodejs-rest-http",
  "devDependencies": { (6)
    "coveralls": "^3.0.0",
    "nodeshift": "^1.3.0",
    "nsp": "~3.1.0",
    "nyc": "~11.4.1",
    "standard-version": "^4.2.0",
    "supertest": "^3.0.0",
    "szero": "^1.0.0",
    "tap-spec": "~4.1.1",
    "tape": "~4.8.0",
    "xo": "~0.20.3"
  },
```

```
  "dependencies": { 7
    "body-parser": "^1.18.2",
    "debug": "^3.1.0",
    "express": "^4.16.0",
    "license-reporter": "^1.1.3"
  }
}
```

1. A **npm** script for running unit tests. Run with **npm run test**.
2. A **npm** script for deploying this application to Single-node OpenShift Cluster. Run with **npm run openshift**. The **strictSSL** option allows us to deploy to Single-node OpenShift Cluster instances with self-signed certificates.
3. A **npm** script for starting this application. Run with **npm start**.
4. The primary entrypoint for the application when run with **npm start**.
5. Specifies the files to be included in the binary that is uploaded to Single-node OpenShift Cluster.
6. A list of development dependencies to be installed from the **npm** registry. These are used for testing and deployment to Single-node OpenShift Cluster.
7. A list of dependencies to be installed from the **npm** registry.

APPENDIX E. ADDITIONAL NODE.JS RESOURCES

- Node.js Home Page
- npm Home Page

APPENDIX F. APPLICATION DEVELOPMENT RESOURCES

For additional information on application development with OpenShift see:

- OpenShift Interactive Learning Portal
- Red Hat OpenShift Application Runtimes Overview

APPENDIX G. THE SOURCE-TO-IMAGE (S2I) BUILD PROCESS

Source-to-Image (S2I) is a build tool for generating reproducible Docker-formatted container images from online SCM repositories with application sources. With S2I builds, you can easily deliver the latest version of your application into production with shorter build times, decreased resource and network usage, improved security, and a number of other advantages. OpenShift supports multiple build strategies and input sources.

For more information, see the Source-to-Image (S2I) Build chapter of the OpenShift Container Platform documentation.

You must provide three elements to the S2I process to assemble the final container image:

- The application sources hosted in an online SCM repository, such as GitHub.
- The S2I Builder image, which serves as the foundation for the assembled image and provides the ecosystem in which your application is running.
- Optionally, you can also provide environment variables and parameters that are used by S2I scripts.

The process injects your application source and dependencies into the Builder image according to instructions specified in the S2I script, and generates a Docker-formatted container image that runs the assembled application. For more information, check the S2I build requirements, build options and how builds work sections of the OpenShift Container Platform documentation.

APPENDIX H. PROFICIENCY LEVELS

Each available mission teaches concepts that require certain minimum knowledge. This requirement varies by mission. The minimum requirements and concepts are organized in several levels of proficiency. In addition to the levels described here, you might need additional information specific to each mission.

Foundational

The missions rated at Foundational proficiency generally require no prior knowledge of the subject matter; they provide general awareness and demonstration of key elements, concepts, and terminology. There are no special requirements except those directly mentioned in the description of the mission.

Advanced

When using Advanced missions, the assumption is that you are familiar with the common concepts and terminology of the subject area of the mission in addition to Kubernetes and OpenShift. You must also be able to perform basic tasks on your own, for example configure services and applications, or administer networks. If a service is needed by the mission, but configuring it is not in the scope of the mission, the assumption is that you have the knowledge to to properly configure it, and only the resulting state of the service is described in the documentation.

Expert

Expert missions require the highest level of knowledge of the subject matter. You are expected to perform many tasks based on feature-based documentation and manuals, and the documentation is aimed at most complex scenarios.

APPENDIX I. GLOSSARY

I.1. PRODUCT AND PROJECT NAMES

developers.redhat.com/launch

developers.redhat.com/launch is a standalone getting started experience offered by Red Hat for jumpstarting cloud-native application development on OpenShift. It provides a hassle-free way of creating functional example applications, called missions, as well as an easy way to build and deploy those missions to OpenShift.

Fabric8 Launcher

The Fabric8 Launcher is the upstream project on which developers.redhat.com/launch is based.

Single-node OpenShift Cluster

An OpenShift cluster running on your machine using Minishift.

I.2. TERMS SPECIFIC TO FABRIC8 LAUNCHER

Booster

A language-specific implementation of a particular mission on a particular runtime. Boosters are listed in a booster catalog.
For example, a booster is a web service with a REST API implemented using the Thorntail runtime.

Booster Catalog

A Git repository that contains information about boosters.

Mission

An application specification, for example *a web service with a REST API.*
Missions generally do not specify which language or platform they should run on; the description only contains the intended functionality.

Runtime

A platform that executes boosters. For example, Thorntail or Eclipse Vert.x.