Chapter 1. Configuring a Cloud Application Developer Environment

Objectives

After completing this section, you should be able to edit application source code with Virtual Studio Code (VS Code).

Integrated Development Environments

Software developers execute many different types of tasks during the software development life cycle of an application:

- compile and build the source code
- correct syntax errors
- debug runtime errors
- maintain version control changes to the source code
- refactor source code
- execute source code tests

In the past, developers used a collection of separate tools, such as editors, compilers, interpreters, and debuggers, to develop software applications. Inefficiencies arose from using separate tools, leading to the creation of
Integrated Development Environments, or IDEs. IDEs improve developer productivity by integrating common software development tools into a single application. IDEs often integrate:

- language-specific editors
- code completion capabilities
- syntax highlighting
- programming language documentation
- code debugging

- source control management tools, such as Git, SVN, or Mercurial

Many modern IDEs support multiple programming languages. Using these IDEs, developers create applications in a variety of different languages, without needing to learn language-specific tooling, such as compilers and interpreters.

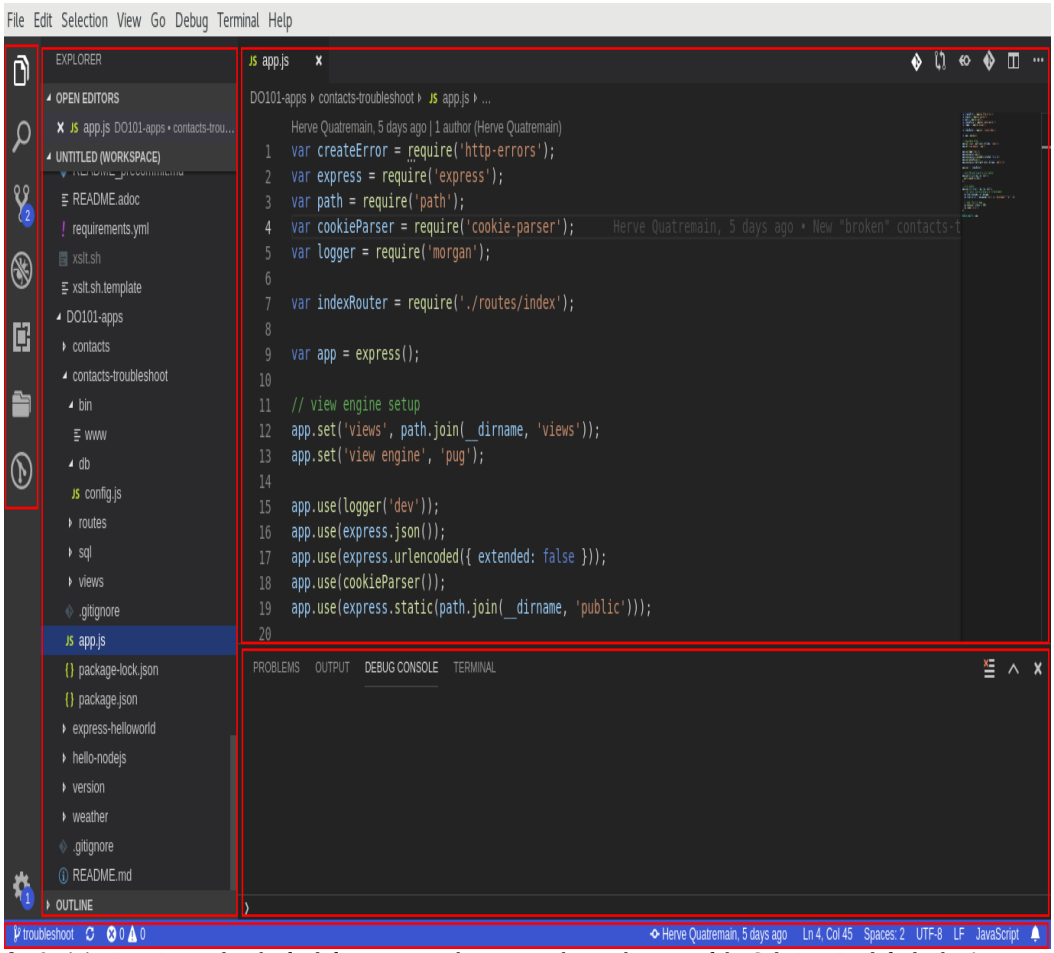
Developing Software with VS Code

VS Code is a popular open source IDE that supports multiple languages, including JavaScript, Python, Go, C#, TypeScript, and more. It provides syntax highlighting, code completion, debugging, and code snippet capabilities, along with a plug-in framework that allows you to install additional functionality from a plug-in marketplace.

In this course, you use VS Code to create, edit, and manage source code projects.

Overview of the VS Code Interface

The VS Code interface contains five primary components:



1. **The Activity Bar**. Located at the far-left, it contains shortcuts to change the view of the Side Bar. By default, the Activity Bar contains shor menu.
2. **Side Bar**. Located immediately to the right of the **Activity Bar**, this area displays the selected Activity view, such as the Explorer view.
3. **Editor Groups**. The top-right region of VS Code contains one or more groups of editors.

   By default, there is only one editor group. Any active editor takes up the entire region of the editor group.

   Click the `Split Editor Right` icon in the upper-right to create a second editor group. With two editor groups, you can edit two different fi
4. **Panels**. Located below the editors, individual panels provide different output or debugging information for activities in VS Code. By default

   - Problems
   - Output
   - Debug Console
   - Terminal

5. **Status Bar**. Located at the bottom, this area provides information about the open project and files as you edit.

VS Code Workspaces

VS Code organizes a collection of related software projects and configuration settings in a **workspace**. Configuration data for each workspace is stored in a file with a `.code-workspace` extension.

From the `File` menu, you can close, save, and open a workspace.

For example, consider a web application, which is named `myapp`, with the following components:

- JavaScript code for the front end user interface of the application.
- Python code for the back-end application logic.
- Configuration and scripts for the application database.
- AsciiDoc files for the application documentation.

If you maintain each of these components in separate code repositories or folders, then you can add each folder to a `myapp` VS Code workspace that you dedicate for the application:

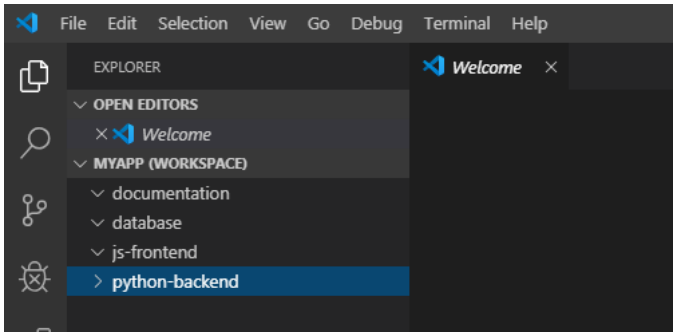


Figure 1.1: A VS Code workspace with multiple source code folders

To add a source code folder to a workspace, click **File** → **Add Folder to Workspace...**.

To remove a source code folder from a workspace, access the Explorer view (**View** → **Explorer**). Right-click a selected top level workspace folder, and then select `Remove Folder from Workspace` to remove the folder from your workspace.

VS Code Integrated Terminal

Although VS Code integrates many development tools, you might need access to external development tools or applications. VS Code integrates the terminal from your operating system, enabling you to execute arbitrary terminal commands in VS Code. To view the integrated terminal, click **View** → **Terminal**.

You can open multiple terminals in the VS Code terminal panel. The terminal panel contains a list of open terminals. To display a terminal, select it from the list. To close a terminal, click **Kill Terminal**.

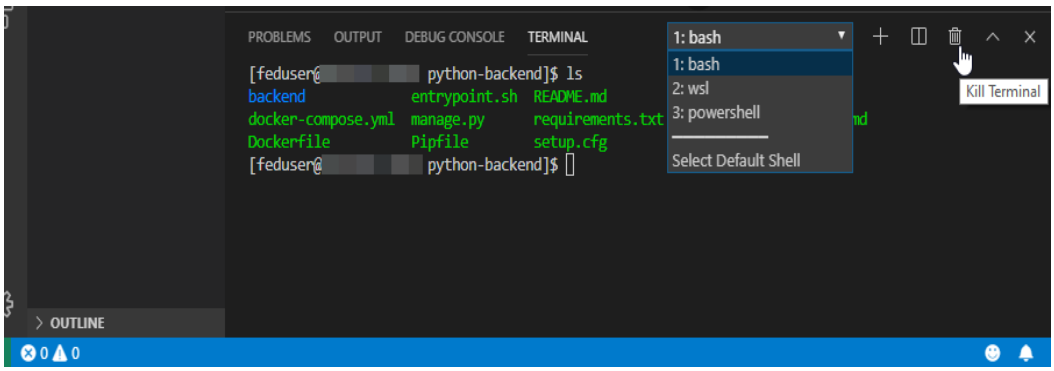


Figure 1.2: The VS Code integrated terminal

VS Code Extensions

Although the VS Code integrated terminal aids arbitrary command execution, you must install and learn to use any needed external commands. Additionally, terminal commands do not take advantage of common usage patterns in VS Code.

VS Code provides an extension framework to encourage the integration of software development capabilities into VS Code. Any user or organization can contribute extensions to VS Code. After an extension is developed, it is advertised and published on the VS Code marketplace.

You can search, download, and install extensions from the VS Code marketplace in VS Code. Click **View** → **Extensions** to access the Extensions view in the Side Bar.

NOTE

In this course, you do not install any additional extensions for VS Code.

Developing a Node.js Application Using VS Code

Many of the example web applications in the course exercises are Node.js applications. If an exercise in this course requires you to edit code, then use VS Code to make the changes.

Node.js is an open source runtime engine that executes JavaScript outside of a browser. It is designed to efficiently handle many concurrent connections for network applications. Additionally, Node.js enables you to write both front end and back end code in one language, JavaScript. For these reasons, Node.js is a popular runtime engine for web application development.

Installing Node.js

To install Node.js, navigate to https://nodejs.org/en/download in a browser. Click the appropriate link for your operating system, and then follow the instructions.

After you install Node.js, you use the `node` command to execute Node.js applications.

Node.js Modules

All modern programming languages support code reuse through shared libraries, packages, and modules.

Modules are the smallest unit of reusable code in Node.js. Node.js provides several built-in modules. You can also download and use third party Node.js modules.

When you create a complex Node.js application, you write custom Node.js modules to group related logical code. Your custom Node.js modules are defined in JavaScript text files.

Use the `require` keyword to load a Node.js module. Consider the following example:

```
var http = require('http');
var mymodule = require('./mymodule');
```

The `http` variable contains a reference to the built-in `http` module, while the `mymodule` variable contains a reference to the module defined in the `mymodule.js` file.

Node.js Packages

Like other programming languages, Node.js provides a way to define and manage application dependencies. A Node.js application dependency is called a
package. A package is collection of one or more Node.js modules, or application code, that you download from a Node.js package repository.

Application dependencies are defined in the `packages.json` file, located at the root of the Node.js project folder. An example `packages.json` file follows:

```
{
    "name": "hello",
    "version": "0.1.0",
    "description": "An example package.json",
    "author": "Student <student@example.com>",
    scripts: {
        "start": "node app.js"
    },
    "dependencies": {
        "cookie-parser": "1.4.*",
        "http-errors": ">=1.6.3",
    },
    "license": "MIT"
}
```

In this example, the `hello` application requires specific versions of the `cookie-parser` and `http-errors` packages.

The `packages.json` file also defines other metadata for a Node.js application, such as the name, version, and author of the application.

The Node Package Manager

The Node Package Manager (NPM) is a command line tool used to create, install, and publish Node.js packages. For most operating systems, the
`npm` command (short for Node Package Manager) is installed as part of the Node.js installation process.

To install the dependencies for a Node.js application, use the `npm install` command.

To initialize an empty directory as a Node.js project, use the `npm init` command to create a `packages.json` file for a new Node.js application.

To manage the application life cycle, use the `npm` command to start, stop or restart the application.

The Express Web Application Framework

Express is a common Node.js framework that aims to simplify the creation of web services. Because Express is a Node.js package, use the
`npm install` command to install Express:

```
$> npm install express
```

After installing the `express` Node.js package, the `express` command is available on your system. Use the `express` command to create a set of initial project files for a new Express application.

Consider the example that follows:

```
$> express /path/to/project/folder/myapp
```

The command creates a `myapp` folder that contains a `packages.json` file:

```
{
  "name": "myapp",
  "version": "0.0.0",
  "private": true,
  "scripts": {
    "start": "node ./bin/www"
  },
  "dependencies": {
    "cookie-parser": "~1.4.4",
    "debug": "~2.6.9",
    "express": "~4.16.1",
    "http-errors": "~1.6.3",
    "jade": "~1.11.0",
    "morgan": "~1.9.1"
  }
}
```

The `express` package is a dependency for the `myapp` Node.js application. Start the `myapp` application with the `node ./bin/www` command.

The source code in the `./bin/www` file loads the`app` Node.js module:

```
...output omitted...
/**
 * Module dependencies.
 */

var app = require('../app');
...output omitted...
```

The `app` module source code is contained in the `app.js` file, located in the root of the `myapp` project directory. The `app.js` file contains the primary application logic.

The `app.js` file for a simple "Hello, World!" Express application contains the following:

```
var express = require('express');
var app = express();

app.get('/', function (req, res) {
  res.send('Hello, World!\n');
});

app.listen(8080, function () {
  console.log('Example app listening on port 8080!');
});

module.exports = app;
```

The `app` variable references an instance of an Express application. The application is configured to listen for requests on port 8080. When you access the application endpoint, the applications sends a response of `Hello, World!`.

Guided Exercise: Developing Applications with Visual Studio Code

In this exercise, you will use Visual Studio Code (VS Code) to create a simple Node.js application.

Outcomes

You should be able to:

- Download and install Node.js.
- Download and install VS Code.
- Create a workspace in VS Code.
- Add a project folder to a VS Code workspace.

To perform this exercise, ensure that you have access to a Linux (Debian or Fedora-based), macOS, or Windows system, including the required permissions to install software on that system.

Procedure 1.1. Steps

1. Download and install `Node.js`.
    1. Windows Installation.
        - In a browser, go to https://nodejs.org/en/download. Click **Windows Installer** to download the `Node.js` installer for Windows. Click **Save File** to launch a file window. Click **Save** in the file window to accept the default filename and location for the file.
        - Navigate to the downloaded file and open it to display a Setup Wizard window. Click **Next** on the Welcome screen to begin installation.
        - Check `I accept the terms in the License Agreement` and then click **Next**.
        - Click **Next** to accept the default installation location.
        - Click **Next** to accept the default settings on the `Custom Setup` screen.
        - Click **Next** to skip the installation of tools you must use to compile native modules.
        - Click **Install** to begin installation. Click **Yes** to allow the application to make changes to your system. Wait for the installation to finish.
        - Click **Finish** to exit the Setup Wizard.
    2. Linux Installation.
        - Open a new command line terminal.
        - To install `Node.js` on Ubuntu and Debian systems, use the following command:

          ```
          yourname@yourhost:~$ sudo apt install nodejs
          ```

          The command may ask for your password to install the package.

          A later exercise requires the Node Package Manager (NPM). Install the package:

          ```
          yourname@yourhost:~$ sudo apt install npm
          ```

          The Ubuntu `npm` package installs several different software development packages. You can skip this `npm` installation if you need to minimize the number of packages on your Ubuntu system.
        - To install `Node.js` on Fedora and Red Hat Enterprise Linux 8 systems, use the following command:

          ```
          [yourname@yourhost ~]$ sudo dnf install nodejs
          ```

The command may ask for your password to install the package.

3. macOS Installation.

- Navigate to https://nodejs.org/en/download/ in a web browser.
- Download the stable long term support (LTS) 64-bit Node.js installer (.pkg) for macOS
- Run the .pkg installer. In recent versions of macOS (10.14 and higher), software that is installed from sources other than the App Store are blocked by default.

  Open **System Preferences** → **Security & Privacy**, and in the `General` tab, click `Open anyway` to continue. Relaunch the Node.js installer if required.
- Click `Continue`. The installer will install `node`, the Node.js runtime, and `npm`, the Node.js package manager to `/usr/local/bin`.
- Click `Continue` in the `Software License Agreement` window.
- Click `Agree` to accept the terms of the license.
- Finally, click `Install` to begin the installation. You may be prompted for your macOS password. Enter your macOS password to continue.

2. Download and install VS Code.

WARNING

This course is designed for VS Code version v1.39, but the instructions that follow show you how to install the newest version of VS Code.

If you wish to match the version of VS Code in this course, go to https://code.visualstudio.com/updates/v1_39 and follow the installation instructions for your operating system.

1. Windows Installation.

- In a browser on your Windows system, go to https://code.visualstudio.com/download
- Click **Windows** to download VS Code for Windows.
- Click **Save File** to save the file. In the window that displays, click **Save** to accept the default file name and download location.
- Navigate to the downloaded file and open it to display the VS Code Setup Wizard. Select `I accept the agreement`, and then click **Next**.
- Click **Next** to accept the default location for the installation. If the folder already exists, click **Yes** to install to the folder anyway.
- Click **Next** to add a Start Menu folder for VS Code. If you do not need a Start Menu folder for VS Code, then select `Don't create a Start Menu folder` before you click **Next**.
- Review the list of additional tasks to perform during VS Code installation. Select any tasks you need for your system and then click **Next**.
- Click **Install** to install VS Code.
- Click **Finish** to close the Setup Wizard.

2. Linux Installation.

- In a browser, navigate to: https://code.visualstudio.com/download
- For Ubuntu and Debian systems, click `.deb`. For Fedora and Red Hat Enterprise Linux systems, click `.rpm`.
- Select `Open with`, and then select `Software Install (default)`. Click **OK**.
- In the installation window, click **Install**.
- In the `Authentication Required` window, enter your password and then click **Authenticate**.
- When the installation completes, close the window.

3. macOS Installation.
    - Navigate to https://code.visualstudio.com/download in a web browser.
    - Click the apple icon (Mac) to download the Visual Studio Code zip file.
    - Uncompress the zip file, and then copy the `Visual Studio Code` executable binary to your `Applications` folder using the macOS `Finder`.
    - Double-click the `Visual Studio Code` binary to start Visual Studio Code.

3. Open VS Code and create a workspace to host your projects.
    1. Open the VS Code application according to your operating system. Click **View** → **Explorer** to display the Explorer view.
    2. If you installed and used VS Code on your system previous to this course, then click **File** → **Close Workspace**.
    If **File** → **Close Workspace** is not available, then skip this step.

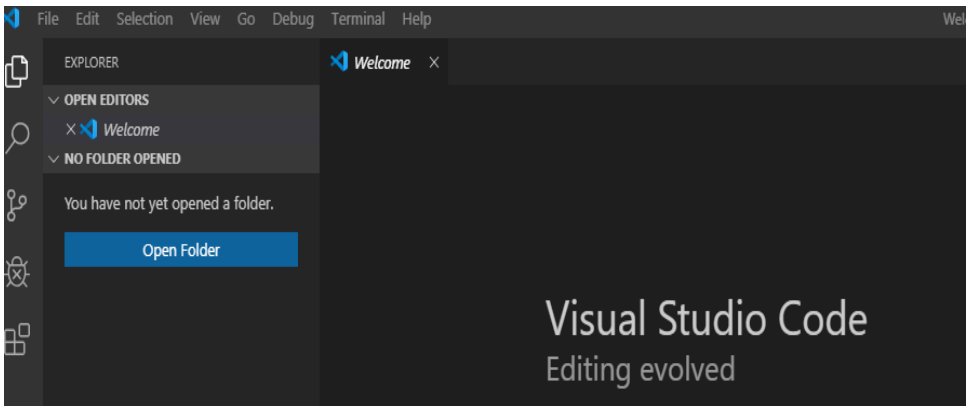


Figure 1.3: The VS Code application

    3. Click **File** → **Save Workspace As ...**. In the window that displays, navigate to your home directory. Type `My Projects` as the file name and then click **Save**. The Explorer view displays a **Add Folder** button to add project folders to your workspace.

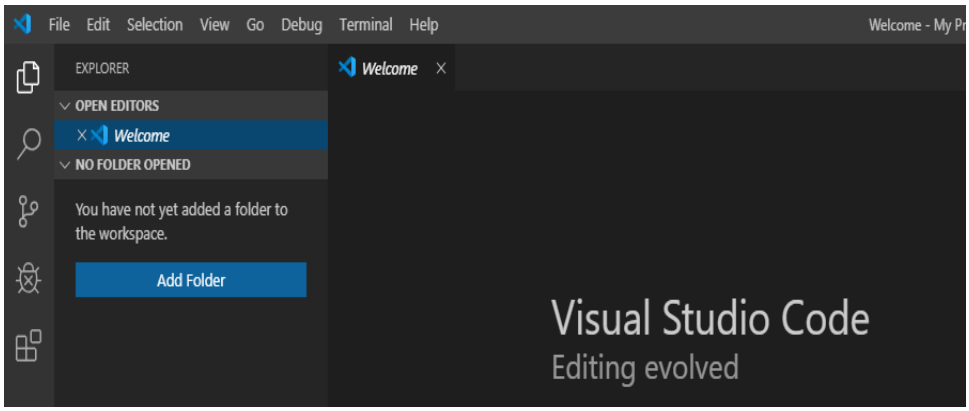


Figure 1.4: The VS Code workspace.

4. Create a `hello-nodejs` project folder, and then add it to your workspace.
    1. In VS Code, click **File** → **Add Folder to Workspace...**. In the window that displays, navigate to your home directory. Create a new folder named `hello-nodejs`. Click **Add** to add this new folder to your workspace.

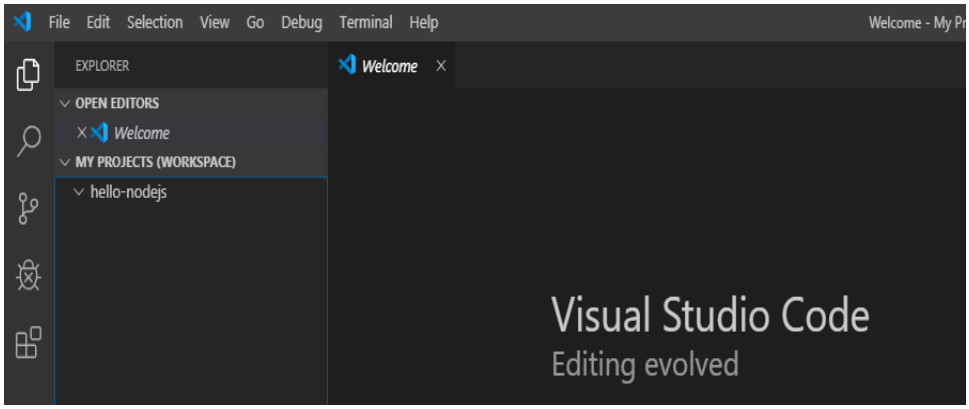


Figure 1.5: Add a folder to the VS Code workspace.

5. Create a `app.js` file in the project. Save the file with the following content:

    1. Right-click on `hello-nodejs` in the workspace, and then select `New File`. Enter `app.js` for the file name to launch a VS Code tab for the new file.

    2. Add the text `console.log("Hello World!\n");` to the `app.js` editor tab, and then save the file (**File → Save**).

6. Right-click on `hello-nodejs` in the workspace, and then select `Open in Terminal` to access the `hello-nodejs` project from the VS Code integrated terminal.

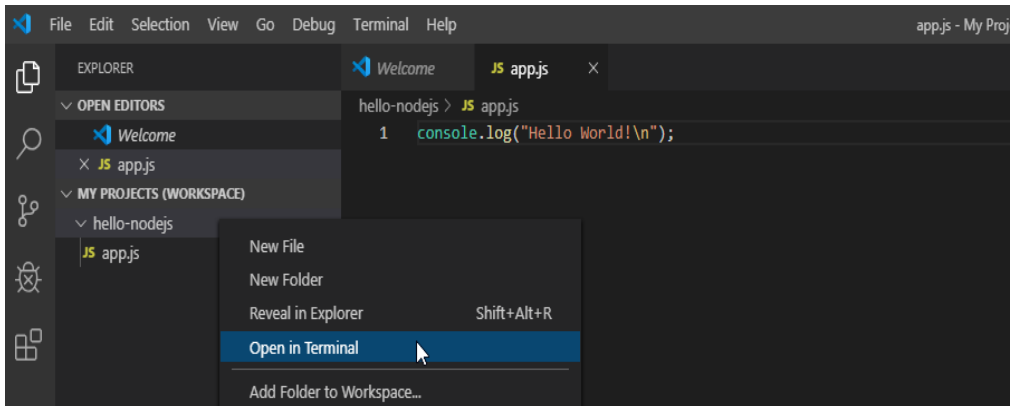


Figure 1.6: Open a project folder in the VS Code integrated terminal.

7. In the integrated terminal, execute `node app.js` to test your sample code and your Node.js installation.

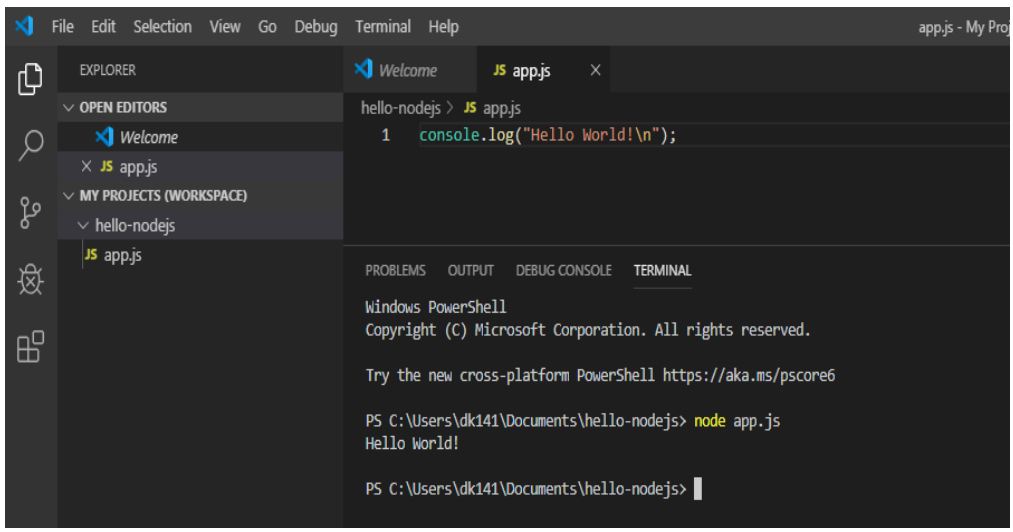


Figure 1.7: Execute a Node.js application in the VS Code intergrated terminal.

8. To clean up your work, click the **Kill Terminal** icon to close the integrated terminal window.

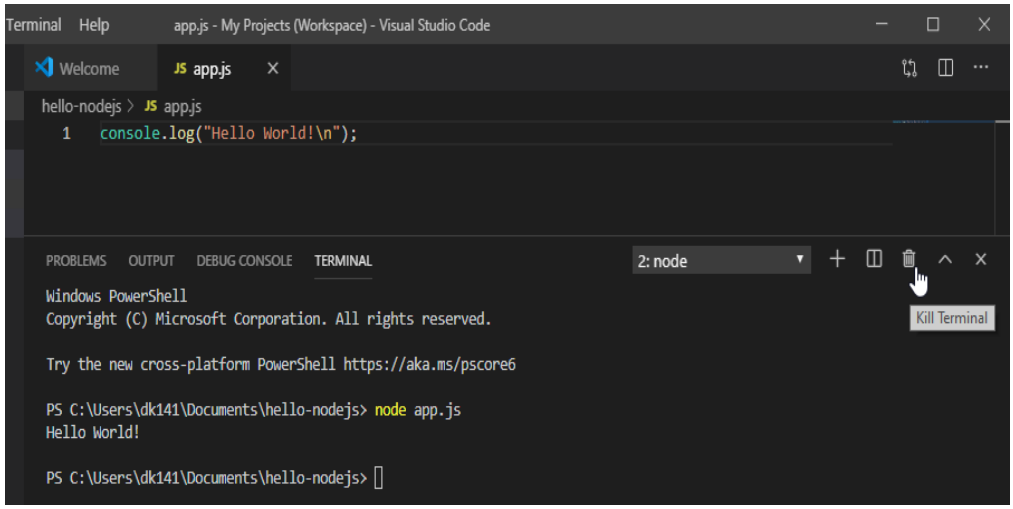


Figure 1.8: Closing the integrated terminal window in VS Code.

This concludes the guided exercise.

Initializing a Git Repository

Objectives

After completing this section, you should be able to create a Git repository.

Software Version Control

A Version Control System (VCS) enables you to efficiently manage and collaborate on code changes with others. Version control systems provide many benefits, including:

- The ability to review and restore old versions of files.
- The ability to compare two versions of the same file to identify changes.
- A record or log of who made changes at a particular time.
- Mechanisms for multiple users to collaboratively modify files, resolve conflicting changes, and merge the changes together.

There are several open source version control systems available including:

- CVS
- SVN
- Git
- Mercurial

Introducing Git

Git is the most popularly used version control system. For this reason, you use Git as the version control system for all the exercises in this course.

Git can convert any local system folder into a Git repository. Although you have many of the benefits of version control, your Git repository only exists on your local system.

To share your repository with another collaborator, you must host the repository on a code repository platform.

There are many free code repository platforms, including:

- GitHub
- GitLab
- BitBucket

- SourceForge

In this course, you use GitHub to host and share your Git repositories.

Git Workflow Overview

To retrieve the project files for an existing software project with Git, you **clone** the Git repository for the project.

When you clone a project, a complete copy of the original remote repository is created locally on your system. Your local copy of the repository contains the entire history of the project files, not just the latest version of project files. You can switch to different versions of the application, or compare two different versions of a file, without connecting to the remote repository. This allows you to continue implementing code changes when the remote repository is not available.

Git does not automatically synchronize local repository changes to the remote repository, nor does it automatically download new remote changes to your local copy of the repository. You control when Git downloads commits from the remote repository, and when local commits are uploaded to the remote repository. Git provides several mechanisms to safely synchronize your local repository with the remote project repository.

To track file changes in Git, you create a series of project snapshots as you make changes to your project. Each snapshot is called a **commit**. When you commit code changes to your repository, you create a new code snapshot in your Git repository.

Each commit contains metadata to help you find and load this snapshot at a later time:

- **commit message** - A high level summary of the file changes in the commit.
- **timestamp** - The date and time that the commit was created.
- **author** - A field that describes who created the commit.
- **commit hash** - A unique identifier for the commit. A commit hash consists of 40 hexadecimal numbers. If a Git command requires a commit hash to perform an operation, then you can abbreviate the commit to seven characters.

After you create commits in a local repository on your system, you must **push** your changes to the remote repository. When you push changes to a remote Git repository, you upload local commits to the remote repository. After a push, your commits are available for others to download.

When other contributors push commits to a remote repository, those commits are not present in your local repository. To download new commits from other contributors, you
pull changes from the remote Git repository.

Committing Code Changes to a Branch

This animation demonstrates how code changes are committed to the master branch of a Git repository.

Volume 90%

Installing Git

Git is an open source version control system that is available for Linux, MacOS, and Windows systems. Before you can use Git, you must install it.

In a browser, navigate to https://git-scm.com/downloads and follow the directions for your operating system.

After installing Git on your system, you can use VS Code to manage your Git source code repositories.

To test your Git installation, open VS Code and access the integrated terminal (**View** → **Terminal**). At the terminal prompt, execute `git --version`. If Git is correctly installed, then a version number prints in the terminal:

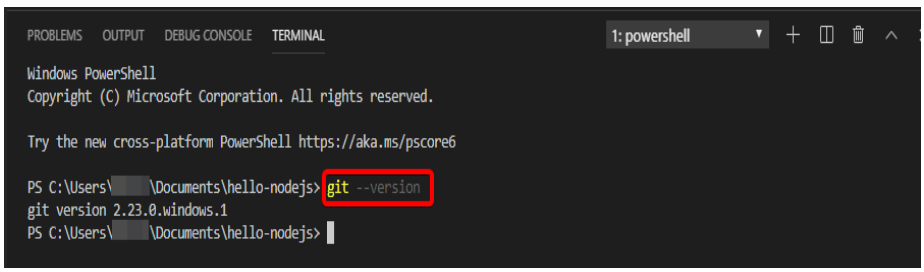


Figure 1.9: Testing the installation of Git in VS Code

Configure the Source Control view in VS Code

Use the VS Code Command Palette (**View → Command Palette...**) and the Source Control view (**View → SCM**) to execute Git operations, such as cloning a repository or committing code changes.

By default, the VS Code Source Control view is different when you have one Git repository in your workspace, compared to when you have multiple Git repositories in your workspace.

When you have multiple Git repositories in your workspace, then the Source Control view displays a `SOURCE CONTROL PROVIDERS` list:

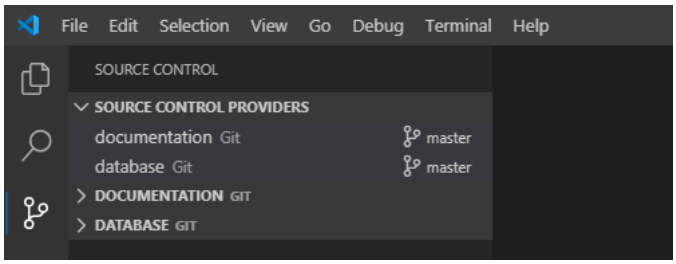


Figure 1.10: The Source Control Providers list in the Source Control view of VS Code.

By default, when there is only a single Git repository in your workspace, then the Source Control view does not display the `SOURCE CONTROL PROVIDERS` list.

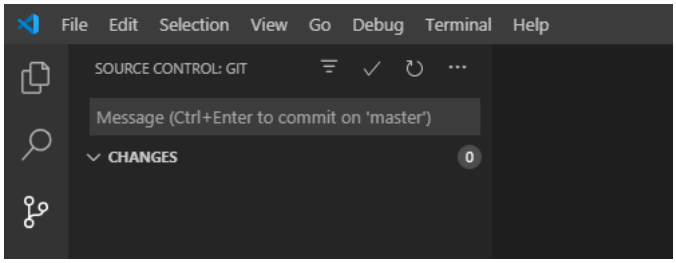


Figure 1.11: Source Control view for a single Git repository.

For a consistent user interface, independent of the number of Git repositories in your workspace, you must enable the `Always Show Providers` source control management option.

To enable this option, access the Command Palette (**View → Command Palette...**) and type `settings`. Select `Preferences: Open Settings (UI)` from the list of options. VS Code displays a settings window:

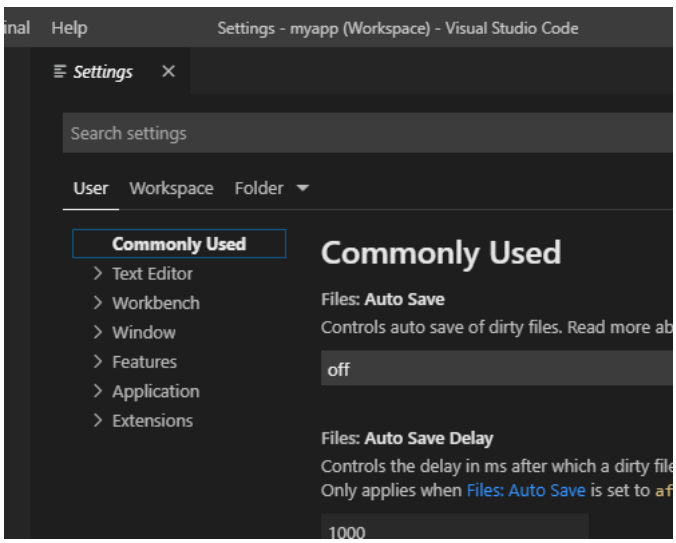


Figure 1.12: The Settings window for VS Code.

Click **User**, and then click **Features → SCM**. VS Code displays Source Control Management (SCM) options for VS Code. Select `Always Show Providers`

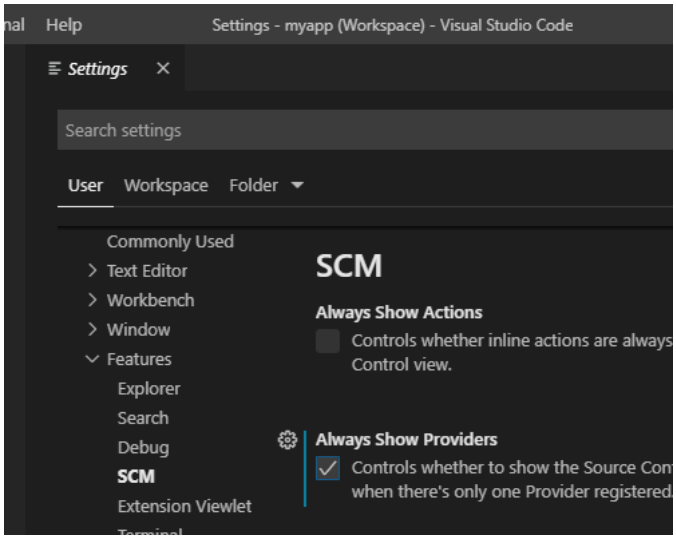


Figure 1.13: Option to always show the Source Control Providers list in the Source Control view of VS Code.

When you enable this option, then VS Code displays the `SOURCE CONTROL PROVIDERS` list in the Source Control view for any number of workspace Git repositories, including only one repository.

Cloning a Git Repository

Use the VS Code Command Palette (**View** → **Command Palette...**) and the Source Control view (**View** → **SCM**) to execute Git operations, such as cloning a repository or committing code changes.

To clone a remote Git repository in VS Code, access the Command Palette (**View** → **Command Palette...**). Type `clone` at the prompt, and then select `Git: Clone` from the list of options.

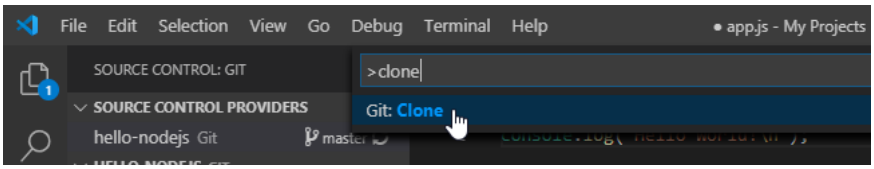


Figure 1.14: Using the command palette to clone a repository.

VS Code prompts you for the URL of the remote repository, and then prompts for the desired location of the local repository on your file system.

After VS Code clones the repository, add the cloned repository folder to your VS Code workspace.

Committing Code Changes

After adding a cloned repository to your VS Code workspace, you can edit project files like any other workspace files. As you edit project files, Git assigns a status to each file:

- **modified** - The file contains saved differences from the most recent version. Modified files are not automatically added or committed to your Git repository.
- **staged** - A staged file is a modified file that you flag to be included as part of your next code commit to the repository.

When you commit code to the repository, only those files with a **staged** status are included in the commit. If your project contains modified files that are not staged, then those files are not included in the commit. This feature enables you to control the file changes that are included in each commit.

In VS Code, the Source Control view (**View** → **SCM**) displays all modified and staged repository files. After you save edits to a file, the file name displays in the `CHANGES` list.

To add a modified file to your next code commit, click the modified file in the `CHANGES` list, from the Source Control view. VS Code displays a new tab to highlight the changes to the file:

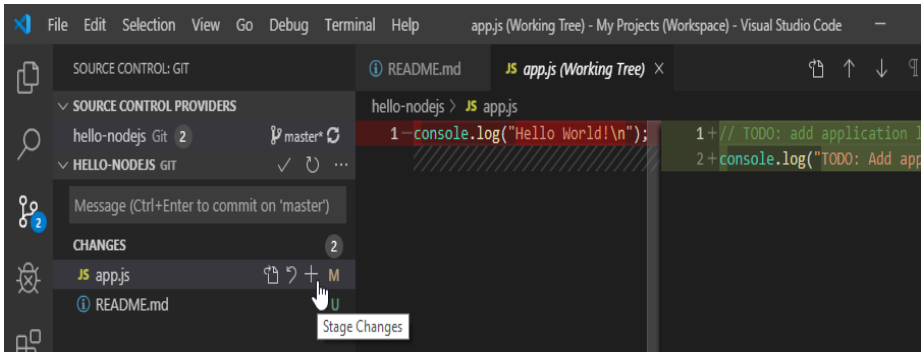


Figure 1.15: Review file changes before staging a file.

If the file changes are accurate and complete, click **Stage Changes** to add the file changes to your next code commit.

After all of your desired file changes are staged, provide a commit message in the Source Control view. Then, select the check box to commit all of the staged file changes to your local repository.

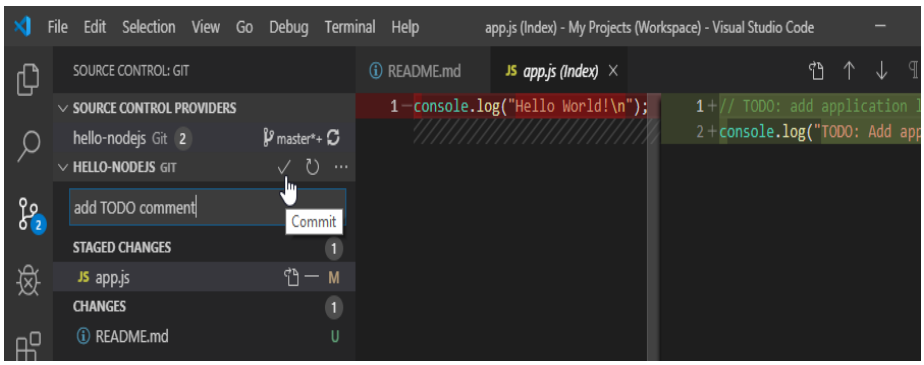


Figure 1.16: Commit staged files in VS Code.

Using a Remote Repository

When you commit code changes, you only commit code to your local repository. No changes are made to the remote repository.

When you are ready to share your work, synchronize your local repository to the remote repository. To retrieve commits from the remote repository, Git performs a pull operation. To publish local commits to the remote repository, Git performs a push operation. VS Code handles the pull and push Git operations when you synchronize your local repository to the remote repository.

The Source Control view compares your local repository with the corresponding remote repository. If there are commits to download from the remote repository, then the number of commits displays with a download arrow icon. If there are local commits that you have not published to the remote repository, then the number of commits appears next to an upload arrow icon.

In the figure that follows, there are zero commits to download and one commit to upload to the remote repository:

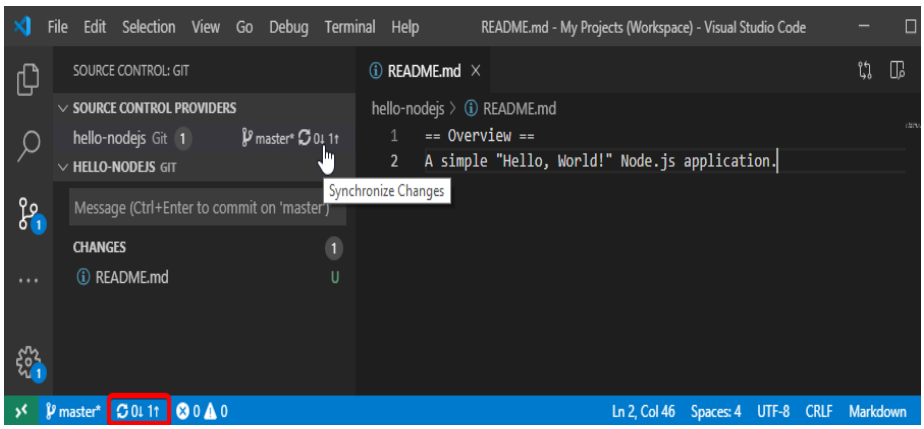


Figure 1.17: Git repository status in the Source Control view and status bar.

Click **Syncrhonize Changes** to publish your local code commits to the remote repository. Alternatively, you can click the same icon in the status bar to publish your code commits.

IMPORTANT

If the Source Control view (**View** → **SCM**) does not contain a `SOURCE CONTROL PROVIDERS` heading, then you will not see a **Synchronize Changes** icon.

To enable it, right-click `SOURCE CONTROL: GIT` . If a check mark does not display to the left of `Source Control Providers` , then click `Source Control Providers` .

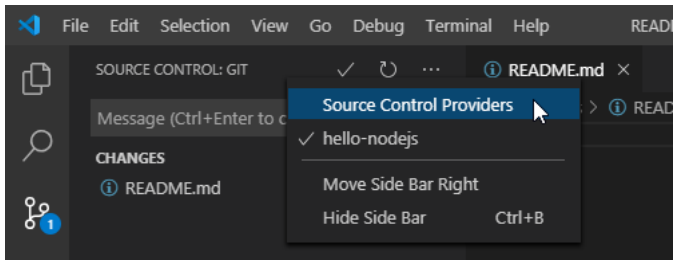


Figure 1.18: Enable the source control providers listing.

Initializing a New Git Repository

If you have an existing software project that needs version control, then you can convert it to a Git repository.

To convert any file system folder into a Git repository, access the VS Code Command Palette (**View** → **Command Palette...**). Type `intialize` at the prompt, and then select `Git: Initialize Repository` from the list of options.

VS Code displays a list of project folders in the workspace.

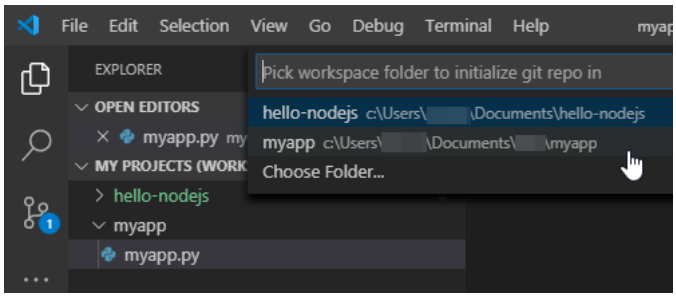


Figure 1.19: Using VS Code to initialize a Git repository.

After you select a project folder, Git initializes the folder as an empty Git repository. The Source Control view displays an entry for the new repository. Because the folder is initialized as an empty repository, every file in the project folder is marked as an untracked file.

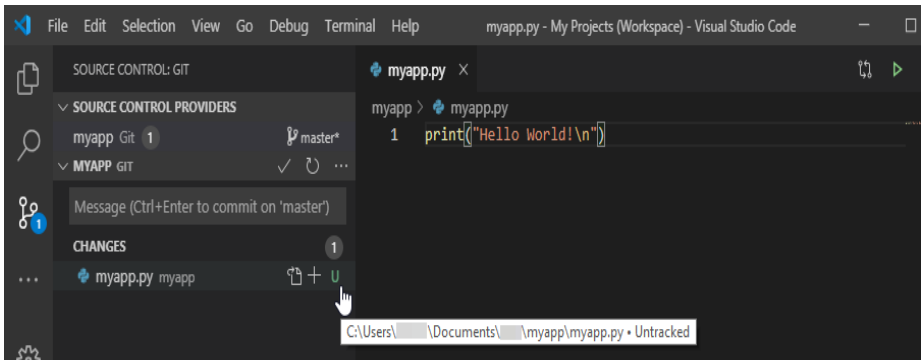


Figure 1.20: Untracked files in a Git repository.

For any project file that requires version control, click the plus icon to add the file to the local repository. Each added file displays in the `STAGED CHANGES` list in the Source Control view. After you stage all of the project files, provide a commit message, and then click the check mark icon to create the first commit in the repository.

When you create a new repository from a local file system folder, the new repository is not associated with a remote repository. If you need to share your repository:

1. Create a new Git repository on a code hosting platform, such as GitHub.
2. Associate your local repository to the new remote repository, and then synchronize changes.

Adding a Remote Repository to a Local Repository

After you create a new repository on a code hosting platform, the platform provides you with a HTTPS and SSH URL to access the repository. Use this URL to add this hosted repository as a remote repository for your local repository.

> **NOTE**
>
> In this course, you only use HTTPS URLs to access remote code repositories. HTTPS access to a Git repository requires very little additional configuration, but does require that you provide credentials for the code hosting platform.
>
> You can configure your Git installation to cache your credentials. This helps minimize re-entering credentials each time you connect to the remote repository.
>
> SSH access to Git repository requires the configuration of your SSH keys with the code hosting platform.
>
> SSH key configuration is beyond the scope of this course.

Access the VS Code Command Palette to add a remote repository to your local repository. Type `add remote` at the prompt, and then select `Git: Add Remote` from the list of options. If you are prompted to select a local repository, then make an appropriate selection.

At the prompt, enter `origin` for the remote name; `origin` is the conventional name given to the remote repository that is designated as the central code repository.

At the prompt, enter the HTTPS URL for your remote repository. If you have commits in your local repository, a `Publish Changes` icon displays in the `SOURCE CONTROL PROVIDERS` list.

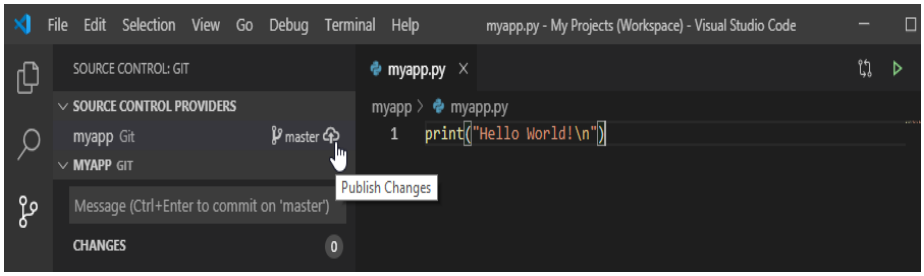


Figure 1.21: Publish a local Git repository to a remote repository.

Click the `Publish Changes` icon to push your local commits to the remote repository. If you are prompted, then provide the necessary remote repository credentials.

Guided Exercise: Initializing a Git Repository

In this exercise, you will use VS Code to push your project source code to a remote Git repository.

Outcomes

You should be able to:

- Install Git.
- Initialize a local folder as a Git repository.

- Stage a file in a Git repository.
- Commit staged files to a local Git repository.
- Push commits in a local Git repository to a remote repository.
- You have access to a Linux (Debian or Fedora-based), macOS, or Windows system and the required permissions to install software on that system.
- Visual Studio Code (VS Code) is installed on your system.

Procedure 1.2. Steps

1. Download and install `Git`.
    - Linux Installation.
        - Open a new command line terminal.
        - To install `Git` on Ubuntu and Debian systems, use the following command:

          ```
          yourname@yourhost:~$ sudo apt install git
          ```

          The command may prompt for your password to install the package.
        - To install `Git` on Fedora and Red Hat Enterprise Linux 8 systems, use the following command:

          ```
          [yourname@yourhost ~]$ sudo dnf install git
          ```

          The command may prompt for your password to install the package.
    - macOS Installation.
        - Git is installed by default on the latest macOS versions. To verify the Git installation, open a new command line terminal and enter the following command:

          ```
          $ git --version
          git version 2.22.0
          ```
    - Windows Installation.
        - In a browser on your Windows system, navigate to https://git-scm.com/download/win and save the executable file to your system.
        - In Windows Explorer, navigate to the downloaded file. Double-click the file to start the setup wizard. If prompted, click **Yes** to allow the installer to make changes to your system.
        - Click **Next** to accept the license agreement.
        - Click **Next** to accept the default installation location for Git. If a window displays a warning about the installation location, click **Yes** to continue the installation of Git to that location.
        - Click **Next** to accept the installation of the default set of components.
        - Click **Next** to accept the default Start Menu Folder.
        - Select `Use Visual Studio Code as Git's default editor` from the editor list to use VS Code as the default editor. Click **Next**.
        - At the `Adjusting your PATH environment` prompt, click **Next**.
        - Make an appropriate choice for the HTTPS transport back-end. If you are unsure of which option to select, then accept the default selection. Click **Next**.
        - At the `Configuring the line-ending conversions` prompt, accept the default selection and click **Next**.
        - Click **Next** to accept the default terminal emulator settings.
        - At the `Configuring extra options` prompt, click **Next** to accept the defaults.
        - Click **Install** to accept the default experimental features and begin installation. Wait for installation to complete, and then proceed to the next step.
        - Click **Finish** to exit the setup wizard.

2. Use VS Code to test your Git installation. Configure your Git installation identity with your GitHub credentials.
   - Open VS Code.
   - Click **Terminal** → **New Terminal** to open an integrated terminal.
   - Execute `git --version` in the integrated terminal to test the installation of Git. The command prints the version of the Git installation on your system.

     NOTE

     VS Code depends on the configuration options selected during the Git installation process. If the `git --version` command fails in the integrated terminal, try restarting VS Code. Then, repeat this step to check the installation of Git.
   - In a browser, navigate to https://github.com. If you do not have a GitHub account, then create one. Log in to GitHub.
   - In the VS Code integrated terminal, execute `git config --global user.name` *`yourgituser`*, replacing *`yourgituser`* with your GitHub user name.
   - In the VS Code integrated terminal, execute `git config --global user.email` *`user@example.com`*, replacing *`user@example.com`* with the email address associated with your GitHub account.

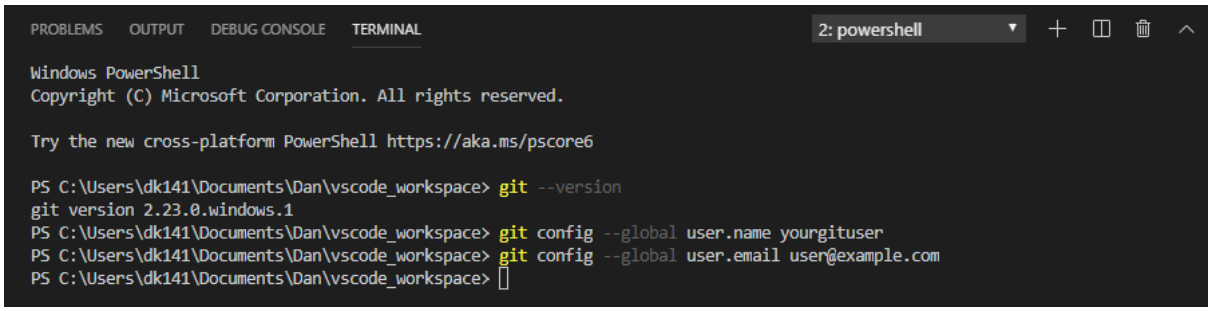


Figure 1.22: The VS Code integrated terminal.

NOTE

Git requires your GitHub user name and password for certain transactions with remote repositories.

On Windows systems, Git manages these credentials by default. You are only prompted for credentials the first time you connect to a remote repository.

By default on Linux and macOS systems, Git does not manage your remote repository credentials. Git prompts for your credentials each time you connect to GitHub.

To cache your credentials on Linux or macOS systems, execute the following command from a system terminal:

```
$> git config --global credential.helper cache
```

3. Enable the `Always Show Providers` source control management option in VS Code.
   - Access the Command Palette (**View** → **Command Palette...**) and type `settings`. Select `Preferences: Open Settings (UI)` from the list of options.
   - When the `Settings` window displays, click **User** → **Features** → **SCM**.
   - VS Code displays Source Control Management (SCM) options for VS Code. Select `Always Show Providers`.

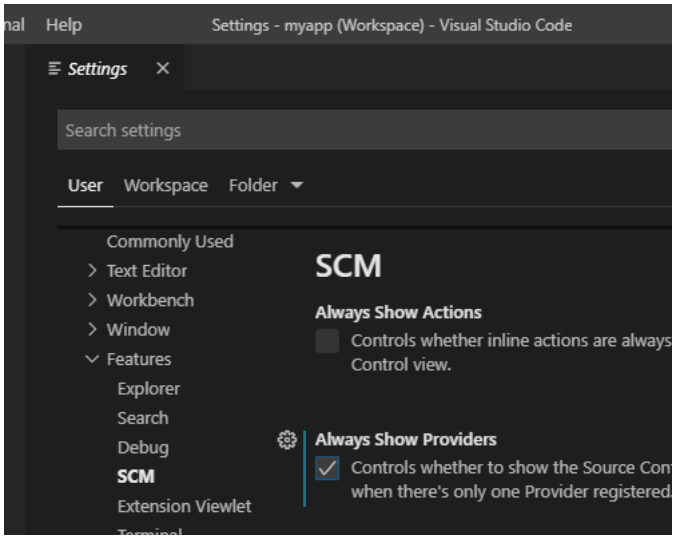


Figure 1.23: Option to always show the Source Control Providers list in the Source Control view of VS Code.

- Close the `Settings` tab.

4. Ensure that you have a `hello-nodejs` project folder in your VS Code workspace. If you already have a `hello-nodejs` project folder in your VS Code workspace from a previous exercise, then skip this step.

   - Download the following zip file to your system:

     https://github.com/RedHatTraining/DO101-apps/releases/download/v0.1/hello-nodejs.zip.

     Unzip the file, which creates a `hello-nodejs` folder on your system. The `hello-nodejs` folder contains a single file, `app.js`. Note the location of the `hello-nodejs` folder. You use this folder in a later step.

   - Click **File** → **Add Folder to Workspace…**

   - In the file window, navigate to the location of the unzipped `hello-nodejs` folder. Select the `hello-nodejs` folder and click **Add**.

5. Initialize the `hello-nodejs` project as a Git repository.

   - Access the VS Code Command Palette (**View** → **Command Palette…**).

   - Type `initialize`. VS Code provides a list of possible commands that match what you type. Select `Git: Initialize Repository` from the list of Command Palette options.

Figure 1.24: Git repository initialization using the Command Palette.

   - Select `hello-nodejs` from the list of workspace folders.

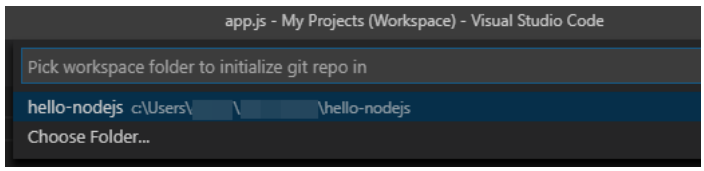


Figure 1.25: Selection prompt to initialize a local Git repository.

6. Create a commit from the `app.js` file.

   - Click **View** → **SCM** to access the `Source Control` view in the Activity Bar.

   - Hover over the `app.js` entry under `CHANGES`. VS Code displays a message that the `app.js` file is untracked. Click the plus sign for the `app.js` entry to add the file to the repository.

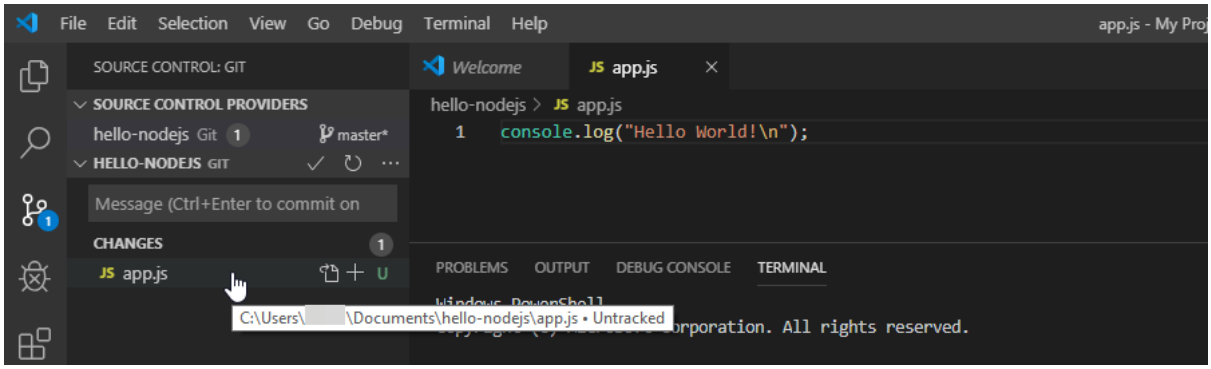


Figure 1.26: List of changed files.

This stages the `app.js` file for the next commit. The file now appears under the `STAGED CHANGES` heading.

- Click in the `Message (press Ctrl+Enter to commit)` field. Type `add initial app.js code` in the message field. Click the check mark icon to commit the changes.

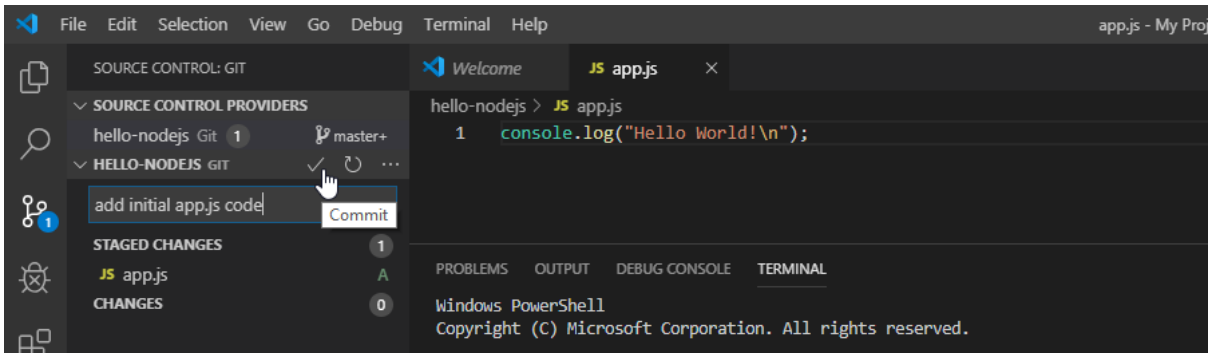


Figure 1.27: VS Code commit message box.

7. Create a new GitHub repository to host your project files. Add the GitHub repository as a remote repository for your local `hello-nodejs` project. Publish your local repository to GitHub.

   - In a browser, navigate to https://github.com. If you are not logged in to Github, then log in.
   - Click the `+` on the upper-right, and then select `New repository` from the list displayed.

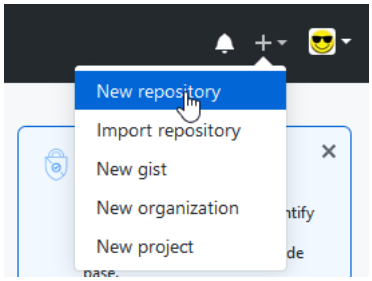


Figure 1.28: Create a new Git repository on GitHub

   - Type `hello-nodejs` in the `Repository name` field. By default, the repository is publicly accessible. If you need a private repository, then select the **Private** check box.

WARNING

Do not select `Initialize the repository with a README`. Also, do not add a `.gitignore` file nor a license to your repository.

Create an empty repository to avoid a merge conflict in a later step.

Click **Create Repository** to create the new GitHub repository. A summary page provides Git commands for a variety of project initialization scenarios:

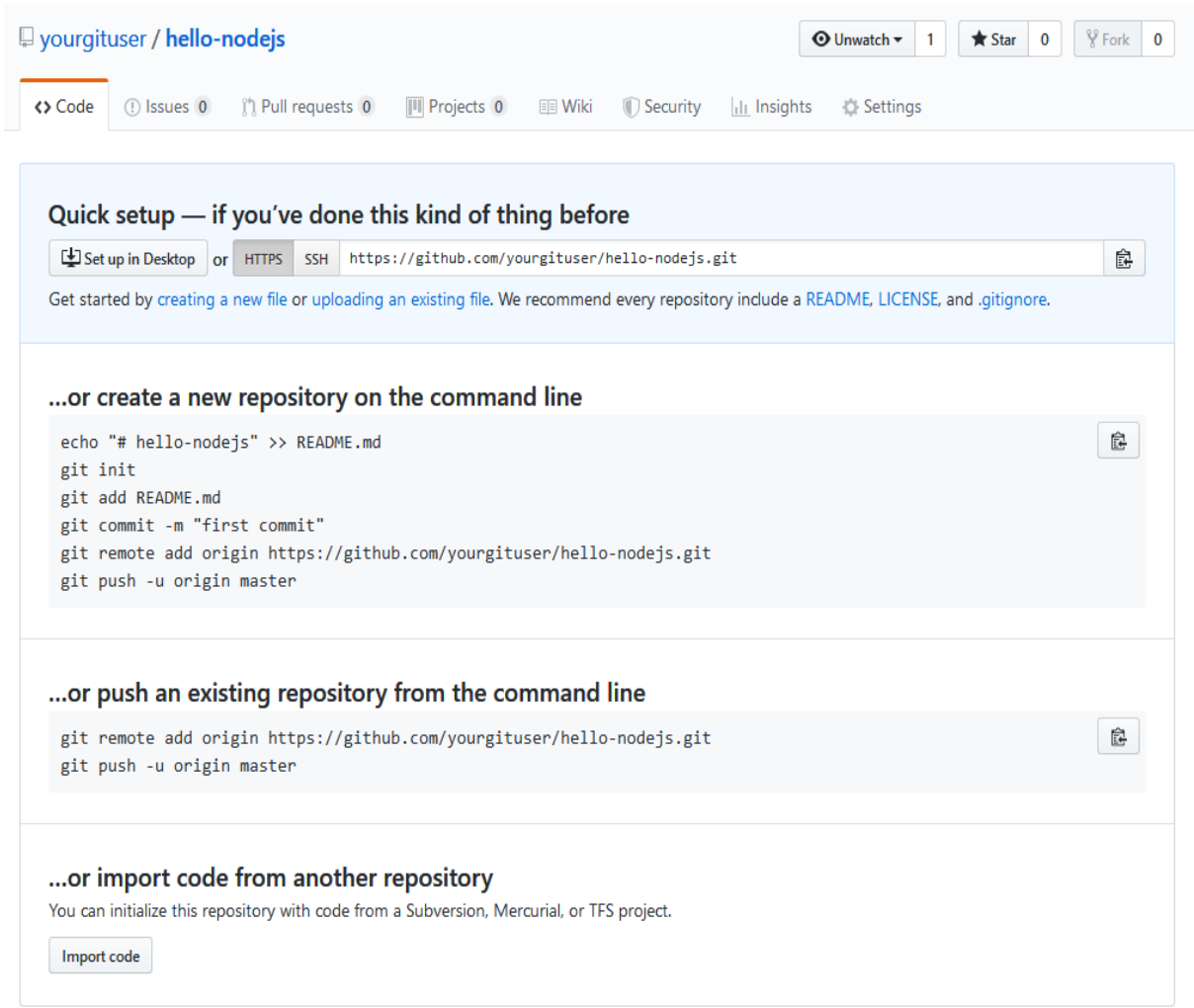


Figure 1.29: The summary page for a new GitHub repository

8. Add your new GitHub repository as a remote repository for the `hello-nodejs` project.

   - In VS Code, type `Git: Add` in the Command Palette (**View** → **Command Palette…**). Then, select `Git: Add Remote` from the list of options.

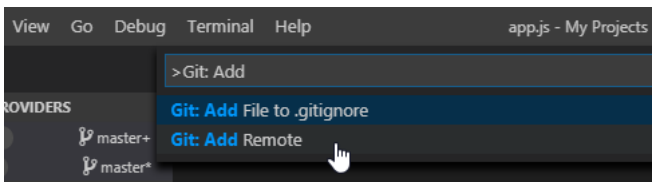


Figure 1.30: The remote URL prompt to add a remote Git repository

   - If you have more that one local Git repository in VS Code, then select `hello-nodejs` from the list of options.

     When prompted for a remote name, enter `origin`.

NOTE

> A Git repository can interact with multiple remote repositories. A remote name of `origin` is a Git convention to indicate the originating repository for a local Git repository.

- At the next prompt, enter the HTTPS URL of your `hello-nodejs` GitHub repository. The URL form is: `https://github.com/yourgituser/hello-nodejs`.

9. Publish your local repository commits to the GitHub repository.

   - Locate the `hello-nodejs` entry in the `SOURCE CONTROL PROVIDERS` section, and then click the "Publish Changes" icon.

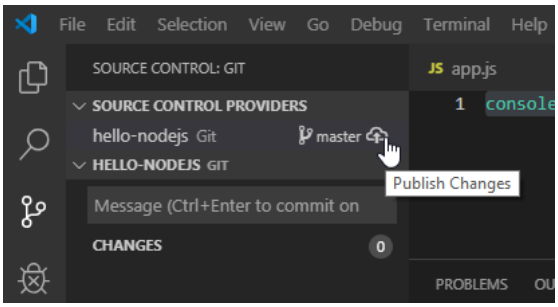


Figure 1.31: Source control view with multiple workspace Git repositories.

The first time VS Code connects to GitHub, a prompt for your GitHub credentials displays. When prompted, provide your GitHub user name and password.

- If this is your first time publishing commits in VS Code, then an additional prompt displays:

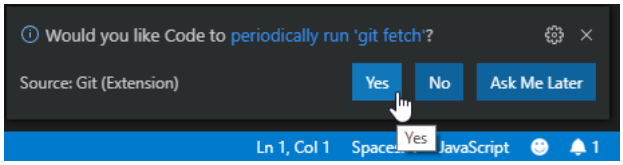


Figure 1.32: VS Code prompt to periodically fetch new commits

Click **Yes** to configure VS Code to periodically check the remote repository for new commits.

10. In a browser, navigate to `https://github.com/yourgituser/hello-nodejs`, replacing *`yourgituser`* with your GitHub user name. Verify that your source code is present in your GitHub repository.

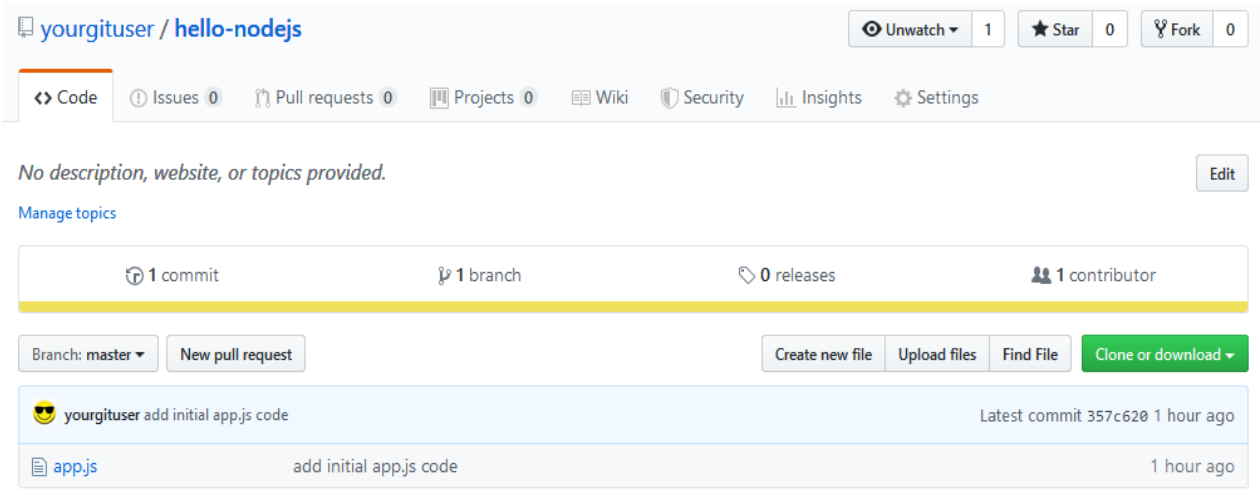


Figure 1.33: Local project files are present on GitHub

11. To clean up your work, click the **Kill Terminal** icon to close the integrated terminal window.

This concludes the guided exercise.

Managing Application Source Code with Git

Objectives

After completing this section, you should be able to use version control to collaborate and manage application source code.

Overview of Git Branching

Git version control features a branching model to track code changes. A **branch** is a named reference to a particular sequence of commits.

All Git repositories have a base branch named `master`. By default, when you create a commit in your repository, the `master` branch is updated with the new commit.

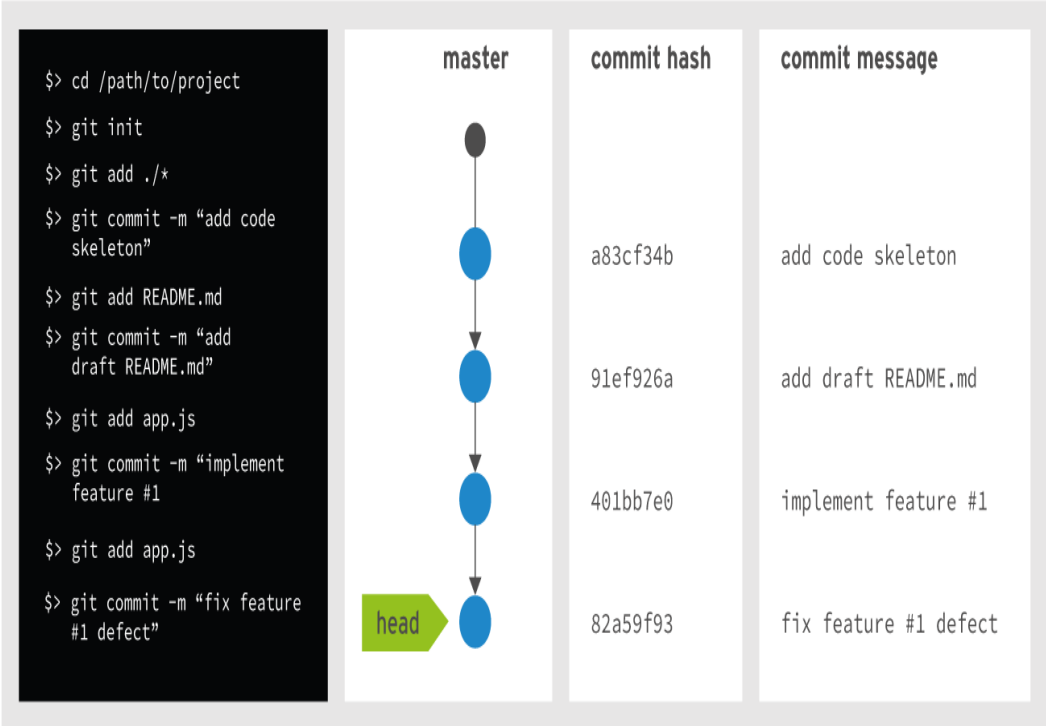


Figure 1.34: Commits to the master branch of a Git repository.

By convention, the `master` branch in a Git repository contains the latest, stable version of the application source code. To implement a new feature or functionality, create a new branch from the `master` branch. This new branch, called a **feature branch**, contains commits corresponding to code changes for the new feature. The `master` branch is not affected by commits to the feature branch.

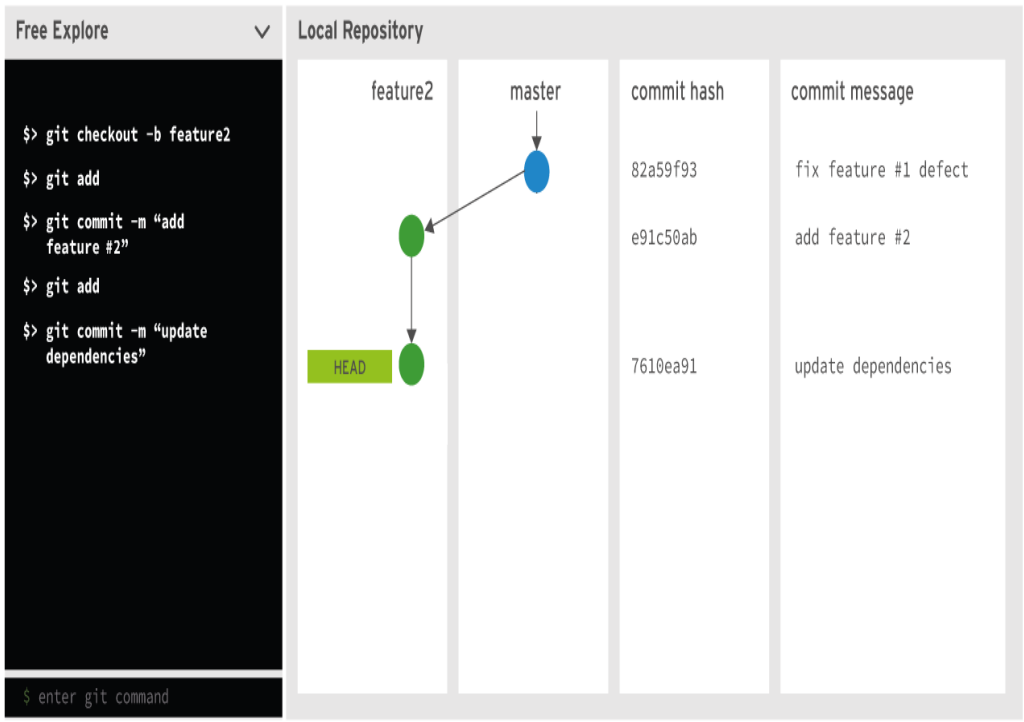


Figure 1.35: Commits to a feature branch of a Git repository.

When you use a branch for feature development, you can commit and share your code frequently without impacting the stability of code in the
`master` branch. After ensuring the code in the feature branch is complete, tested, and reviewed, you are ready to merge the branch into another branch, such as the `master` branch. **Merging** is the process of combining the commit histories from two separate branches into a single branch.

Branching and Merging

This animation uses a feature branch workflow to demonstrate a sequence of code changes.

Volume 90%

Merge Conflicts

Git has sophisticated mechanisms to merge code changes from one branch into another branch. However, if there are changes to the same file in both branches, then a
merge conflict can occur.

A merge conflict indicates that Git is not able to automatically determine how to integrate changes from both branches. When this happens, Git labels each affected file as a conflict. VS Code displays an entry for each file with a merge conflict in the Source Control view, beneath the
`MERGE CHANGES` heading. Each file with a merge conflict entry contains a `C` to the right of the entry.

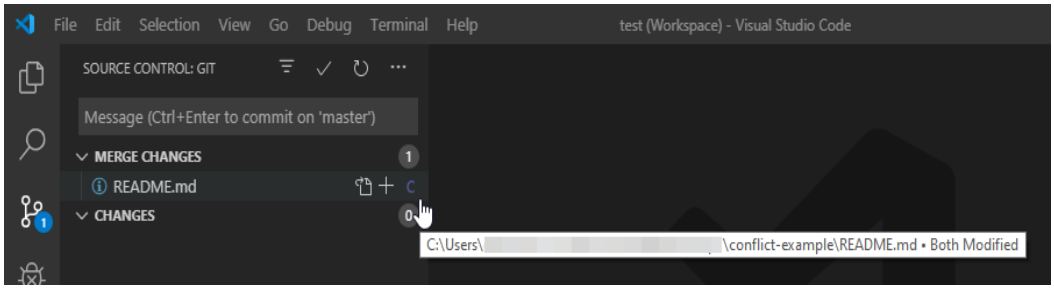


Figure 1.36: A merge conflict in the Source Control view of VS Code.

Git also inserts markers in each affected file to indicate the sections that contain content conflicts from both branches. If you click on the merge conflict entry in the VS Code Source Control view, then an editor tab displays and highlights the sections of the file that conflict.

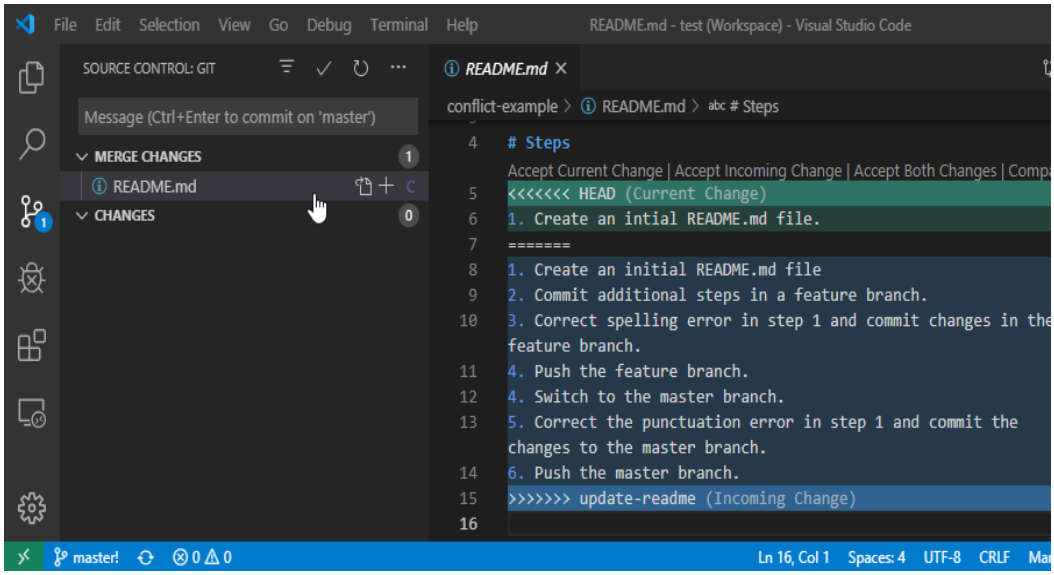


Figure 1.37: VS Code highlights the conflicts in a merge conflict file.

In the preceding figure, the green highlighted section contains content from the current branch, in this case the `master` branch. The blue highlighted text contains changes from the `update-readme` branch, which is being merged into the `master` branch.

There are many options to resolve merge conflicts that happen as a result of the normal merging process.

For each conflict in a conflicted file, replace all of the content between the merge conflict markers (<<<<<<<< and >>>>>>>> inclusive) with the correct content from one branch or the other, or an appropriate combination of content from both branches.

For example, the following figure shows that the conflict section from the preceding example is replaced with content from both the `master` and `update-readme` branches.

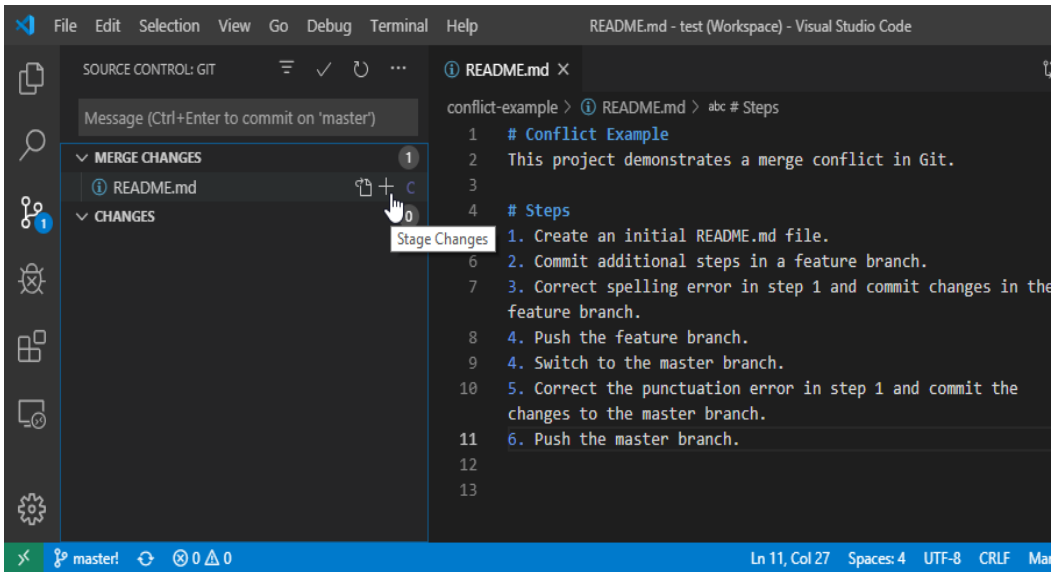


Figure 1.38: Resolving a merge conflict in VS Code.

After you reconcile the content for each conflict in a file, then save, stage, and commit the file changes.

NOTE

Managing merge conflicts is beyond the scope of this course. For more information on merge conflicts, see **Basic Merge Conflicts** at https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging

Collaboration Strategies Using Git

Git provides development teams with the flexibility to implement code collaboration processes and branching strategies that fit the size and dynamics of each team. When a team agrees on a Git workflow, then code changes are managed consistently and team productivity increases.

Centralized Workflow

A centralized Git workflow uses a central Git repository as the single source of record for application code. The central repository is typically hosted on a repository hosting platform, such as GitHub. In this workflow, the development team agrees that the central repository will only contain a single branch,
`master`. Developers push code changes directly to the `master` branch, and do not push commits in other branches to the central repository.

Developers clone the central repository, creating a local copy of the repository for each developer. Developers make code changes and commit to the master branch of their local repository. Developers merge in the latest changes from the remote repository, before pushing new changes to the remote repository.

Because the workflow results in commits to a single branch only, team members are prone to merge conflicts. Merge conflicts are mitigated when teams clearly identify separate code changes for each team member that can be implemented in distinct files.

In addition to merge conflicts, this workflow allows you to commit partial or incomplete code changes to the `master` branch. When incomplete code changes are committed, the stability of code in the `master` branch is compromised. Teams that use a centralized workflow must adopt additional processes and communication strategies to mitigate risks to the stability of project code on the `master` branch.

One way teams mitigate code stability issues with a centralized worfklow is to implement Git tags in the code repository. A **tag** is a human-readable reference to a particular commit in a Git repository, independent of any branch. Teams can implement a tag nomenclature to identify commits corresponding to stable versions of the project source code.

> **NOTE**
>
> Git tags are beyond the scope of this course. See **Tagging** in the Git documentation at https://git-scm.com/book/en/v2/Git-Basics-Tagging.

The centralized Git workflow works well for small teams collaborating on a small project with infrequent code changes, but becomes difficult to manage with larger teams and large code projects.

Feature Branch Workflow

A feature branch workflow implements safety mechanisms to protect the stability of code on the `master` branch. The aim of this workflow is to always have deployable and stable code for every commit on the `master` branch, but still allow team members to develop and contribute new features to the project.

In a feature branch workflow, each new feature is implemented in a dedicated branch. Multiple contributors collaborate on the feature by committing code to the feature branch. After a feature is complete, tested, and reviewed in the feature branch, then the feature is merged into the
`master` branch.

The feature branch workflow is an extension of the centralized workflow. A central repository is the source of record for all project files, including feature branches.

When a developer is ready to merge a feature branch into the master branch, the developer pushes the local feature branch to the remote repository. Then, the developer submits a request to the team, such as a pull request or merge request, to have the code changes in the feature branch reviewed by a team member. After a team member approves the request, the repository hosting platform merges the feature branch into the master branch.

> **NOTE**
>
> A **pull request** is a feature of several repository hosting platforms, including GitHub and BitBucket.
>
> A pull request lets you submit code for inclusion to a project code base. Pull requests often provide a way for team members to provide comments, questions, and suggestions about submitted code changes. Pull requests also provide a mechanism to approve the merging of the code changes into another branch, such as
>
> `master` .
>
> On other platforms, such as GitLab and SourceForge, this code review feature is called a **merge request**.

For example, the following figure shows the GitHub user interface after pushing the `feature1` branch. GitHub displays a notification that you recently pushed the `feature1` branch. To submit a pull request for the `feature1` branch, click **Compare & pull request** in the notification.

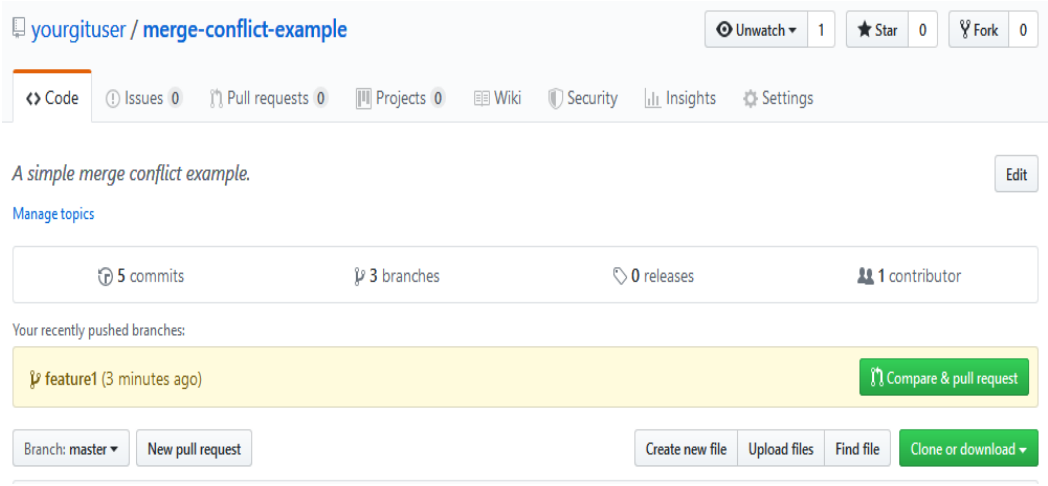


Figure 1.39: GitHub user inteface after you push a feature branch.

GitHub displays a form that you must complete to submit the pull request.

By default, you request that the `feature1` branch be merged (or pulled) into the `master` branch. If you want to merge the feature branch into a different branch, then you select the correct base branch from the list of options. GitHub displays a message to indicate if the merge operation will cause conflicts. In the figure that follows, GitHub indicates that the two branches will merge without any conflicts and can be automatically merged.

The GitHub pull request form allows you to provide a title and description for the pull request. The form also allows you to assign the pull request to a GitHub user, and also specify a set of GitHub users to review the code changes in the pull request.

After completing the form, click **Create pull request** to create the pull request.

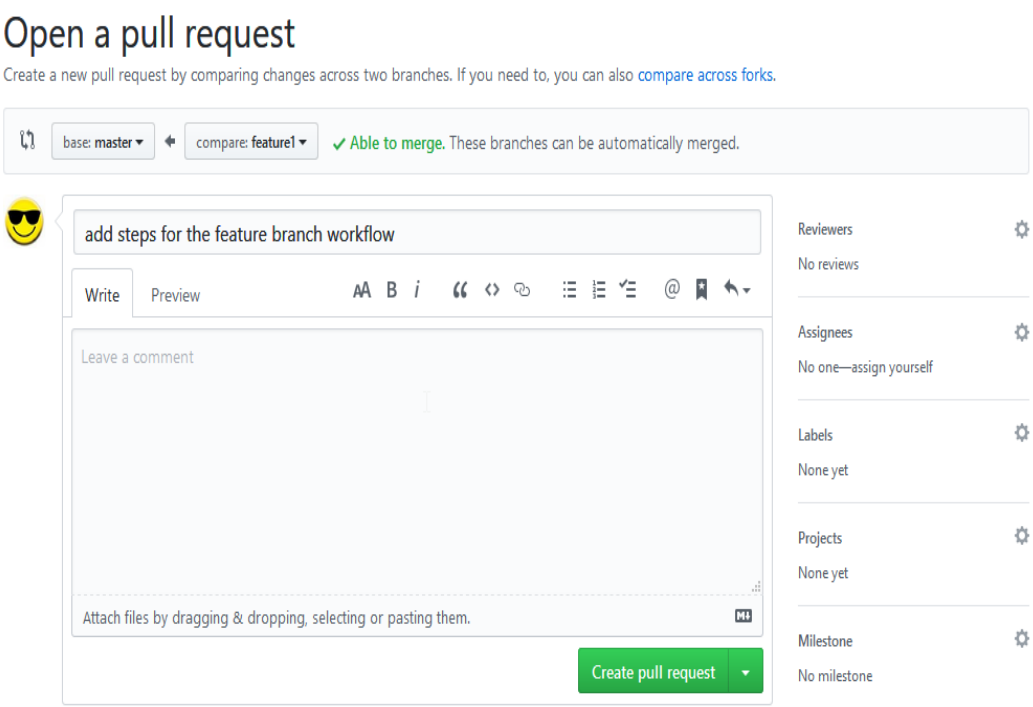


Figure 1.40: The GitHub form to submit a pull request.

After a reviewer grants approval, you can merge the pull request.

Click **Pull requests** for the GitHub repository to display a list of pull requests. Select your pull request from the list. GitHub displays information about the pull request, including comments, questions, or suggestions by other reviewers. When you are ready to merge the

changes to the `master` branch, click **Merge pull request**.

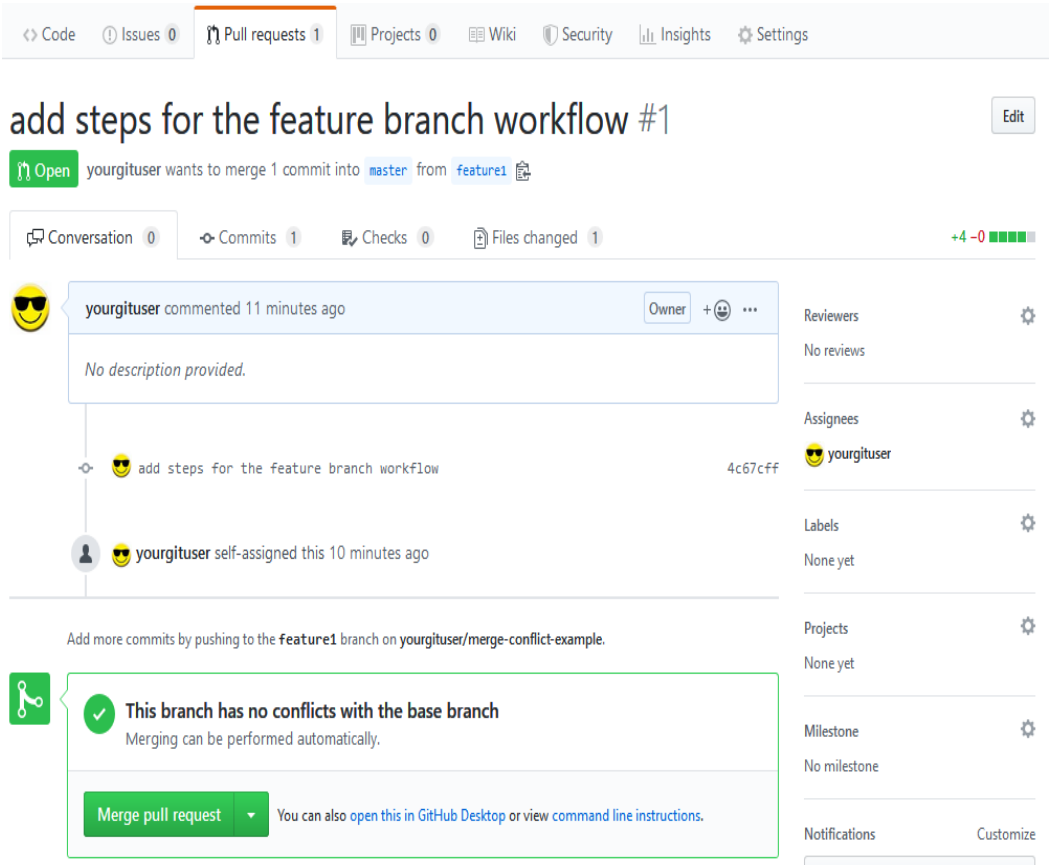


Figure 1.41: A GitHub pull request.

GitHub displays a message for the merge operation. Click **Confirm merge** to merge the `feature1` branch into the `master` branch. GitHub then displays a **Delete branch** button.

Finally, in the feature branch workflow you delete a feature branch after it is merged into the `master` branch. Click **Delete branch** to delete the branch.

Forked Repository Workflow

The Forked repository workflow is often used with large open source projects. With a large number of contributors, managing feature branches in a central repository is difficult. Additionally, the project owner may not want to allow contributors to create branches in the code repository. In this scenario, branch creation on the central repository is limited to a small number of team members. The forked repository workflow allows a large number of developers to contribute to the project while maintaining the stability of the project code.

In a forked repository workflow, you create a **fork** of the central repository, which becomes your personal copy of the repository on the same hosting platform.

After creating a fork of the repository, you clone the fork. Then, use the feature branch workflow to implement code changes and push a new feature branch to your fork of the official repository.

Next, open a pull request for the new feature branch in your fork. After a representative of the official repository approves the pull request, the feature branch from your fork is merged into the original repository.

For example, consider the workflow of the OpenShift Origin GitHub repository.

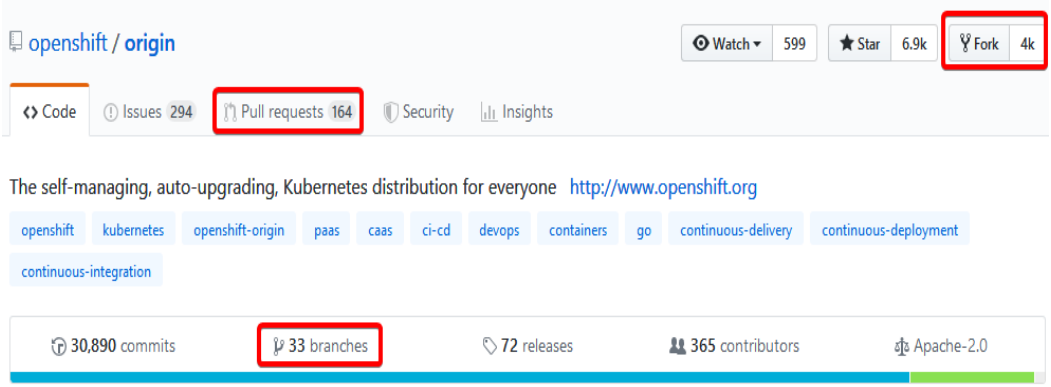


Figure 1.42: The OpenShift Origin GitHub repository.

There are more pull requests than branches in the OpenShift Origin GitHub repository. Many (if not all) of the pull requests contain code from one of the forks of the repository, instead of from a repository branch.

Git Branching in VS Code

In VS Code, use the Source Control view (**View → SCM**) to access the branching features in Git. The `SOURCE CONTROL PROVIDERS` section contains an entry for each Git repository in your VS Code workspace. Each entry displays the current branch for the repository.

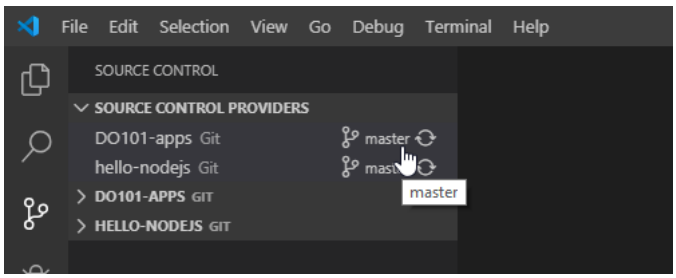


Figure 1.43: The Source Control view shows the current branch for each workspace Git repository.

To switch to a different branch in a repository, click the current branch name for the repository entry under the `SOURCE CONTROL PROVIDERS` heading. VS Code displays two options to create a new branch, and also displays a list of existing branches and tags in the repository:

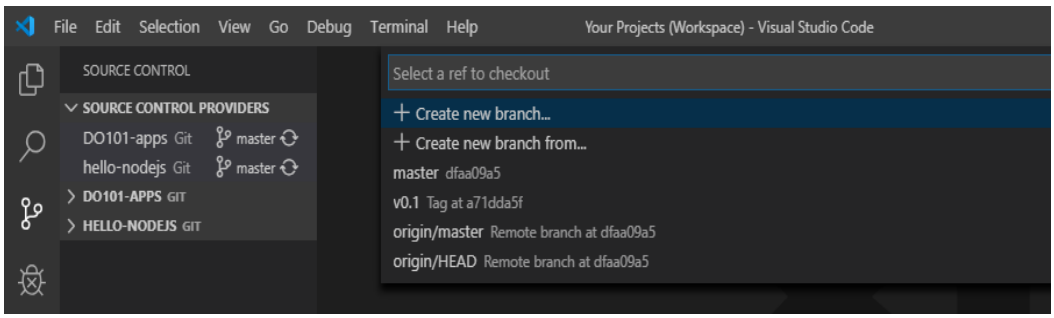


Figure 1.44: Checkout a branch in the Source Control view.

When you select either of the two options to create a new branch, VS Code prompts you for a name to assign to the new branch. If you selected the
`Create new branch...` option, VS Code creates a new branch from the current repository branch.

If you selected the `Create a new branch from...` option, then VS Code also provides you list of existing repository tags and branches. After you select an item, VS Code creates a new branch that begins from the tag or branch that you select.

NOTE

Often software projects adopt branch naming conventions or standards. Branch naming standards help you summarize the code changes contained in a branch.

> The following are examples of branch name templates for a branch naming standard:

- `feature/`*`feature-id`*`/description`
- `hotfix/issue-number/`*`description`*
- `release/release-string`

> A branch naming standard also defines the set of allowable characters. Branch names are often limited to alphanumeric characters and field seperators (such as `/`, `_`, or `-` characters).

After VS Code creates the new local branch, the Source Control view updates the repository entry with the name of the new branch. When you click the `Publish Changes` icon, VS Code publishes the new local branch to the remote repository.

Any new code you commit is added to the new local branch. When you are ready to push your local commits to the remote repository, click the `Synchronize Changes` icon in the Source Control view. The `Synchronize Changes` icon displays:

- the number of commits in the local repository to upload
- the number of commits in the remote repository to download

To download commits from a remote repository, VS Code first fetches the remote commits and merges them into your local repository. Then, VS Code pushes your local commits to the remote repository.

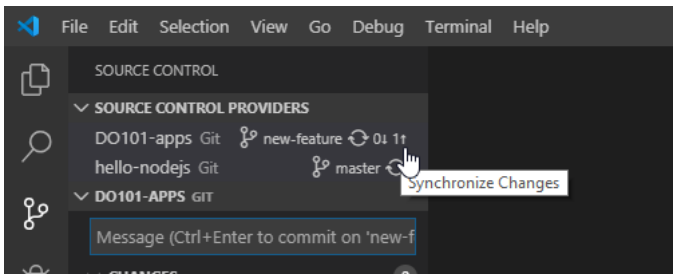


Figure 1.45: Push a local commit to the remote repository.

GUIDED EXERCISE: MANAGING APPLICATION SOURCE CODE WITH GIT

In this exercise, you will use VS Code to push code changes in a new branch to a remote Git repository.

Outcomes

You should be able to:

- Fork the sample applications repository for this course to your personal GitHub account.
- Clone the sample applications repository for this course from your personal GitHub account to your system.
- Commit code changes to a new branch.
- Push a new branch to a remote repository.
- Visual Studio Code (VS Code) is installed on your system.
- Node.js and Node Package Manager (NPM) are installed on your system.
- Git is installed on your system and configured with your user name and email address.

Procedure 1.3. Steps

1. Fork the sample applications for this course into your personal GitHub account.

1. Open a web browser, and navigate to https://github.com/RedHatTraining/DO101-apps. If you are not logged in to GitHub, then click **Sign in** in the upper-right corner.

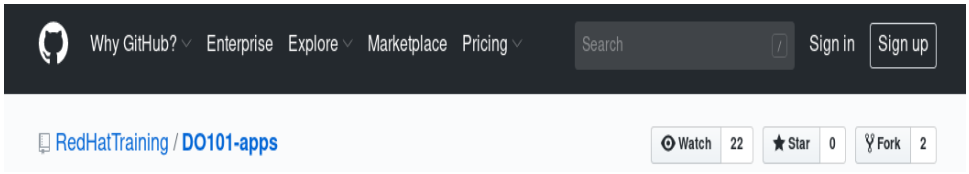


Figure 1.46: Sign-in page for the course Git repository.

2. Log in to GitHub using your personal user name and password.

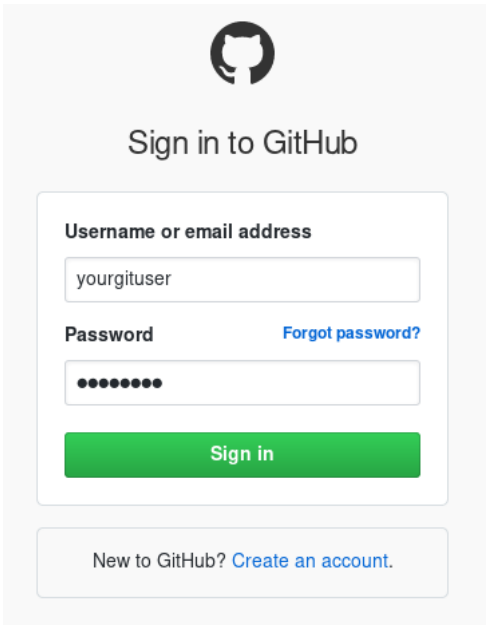


Figure 1.47: GitHub Sign-in page.

3. Return to https://github.com/RedHatTraining/DO101-apps and click **Fork** in the upper-right corner.

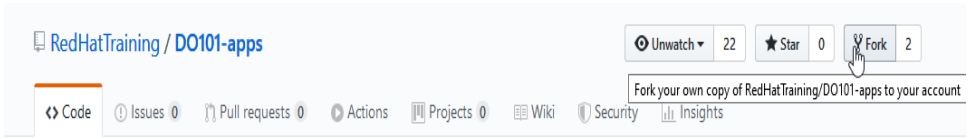


Figure 1.48: Fork the Red Hat Training DO101-apps repository.

4. In the `Fork DO101-apps` window, click *`yourgituser`* to select your personal GitHub project.

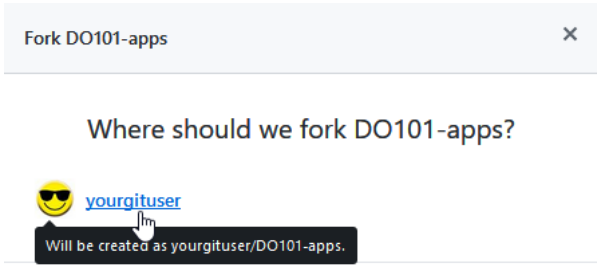


Figure 1.49: Fork the Red Hat Training DO101-apps repository.

IMPORTANT

While it is possible to rename your personal fork of the https://github.com/RedHatTraining/DO101-apps repository, the example output in this course assumes that you retain the name `DO101-apps` when you fork the repository.

5. After a few minutes, the GitHub web interface displays your new *`yourgituser`*`/DO101-apps` repository.

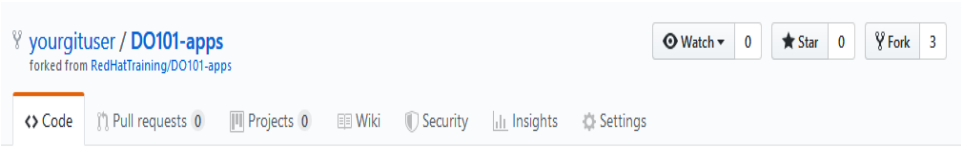


Figure 1.50: A personal fork of the Red Hat Training DO101-apps repository.

2. Clone the sample applications for this course from your personal GitHub account using VS Code.

   1. If you do not have VS Code open from a previous exercise, the open it.
   2. In the Command Palette (**View** → **Command Palette…**), type `clone`. Select **Git: Clone** from the list of options.

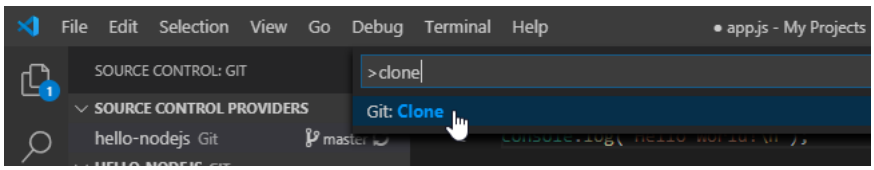


Figure 1.51: Using the command palette to clone a repository.

   3. In the prompt that displays, enter the HTTPS URL for your repository, `https://github.com/yourgituser/DO101-apps`.
   4. In the file window that opens, choose a local folder to store the repository clone. VS Code creates a `DO101-apps` subfolder in the folder you select. The default location is your home folder.

      Click **Select Repository Location** to select your location.

   5. After VS Code clones the repository, a prompt displays in the lower-right corner of the VS Code window. Click **Add to Workspace** to add the cloned repository to your VS Code workspace.

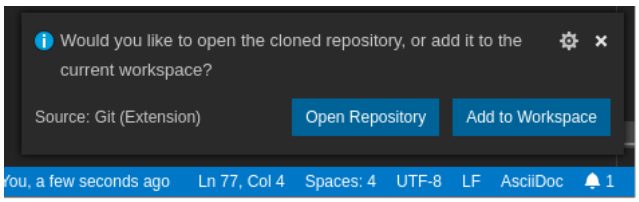


Figure 1.52: Add a cloned repository to the current workspace.

NOTE

This prompt remains active for only a few seconds. If the prompt closes, add the cloned repository folder to your workspace (**File** → **Add Folder to Workspace…**). In the file window that displays, navigate to the location of the cloned `DO101-apps` folder. Select the `DO101-apps` folder, and then click **Add**.

3. Add a feature to the `express-helloworld` application to display `Hello Mars!` when a user accesses the `/mars` application endpoint. Implement the new feature in the `devenv-versioning` branch.

   1. From the Source Control view (**View** → **SCM**), click **master** in the `DO101-apps` entry under `SOURCE CONTROL PROVIDERS`.

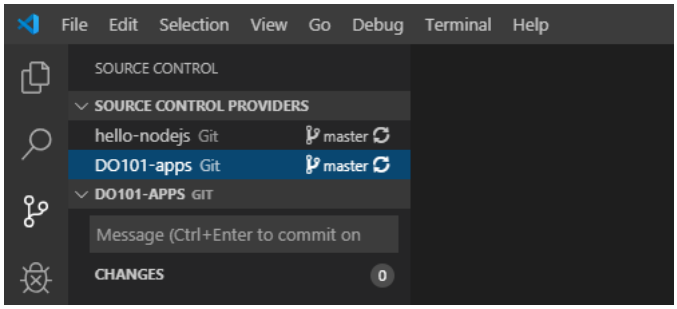


Figure 1.53: Checkout a new branch for a repository.

NOTE

If the Source Control view does not display the `SOURCE CONTROL PROVIDERS` heading, right-click `SOURCE CONTROL` at the top of the Source Control view and select `Source Control Providers`.

2. Select `Create new branch…` from the list of options.
3. When prompted, enter `devenv-versioning` for the branch name. The `DO101-apps` entry in the Source Control view updates to the `devenv-versioning` branch.
4. In the Explorer view (**View** → **Explorer**), click the `DO101-apps/express-helloworld/app.js` file. VS Code opens an editor tab for the file.
5. To implement the new feature, add the following lines to the `app.js` file:

   Insert these lines before the line that begins with `app.listen`:

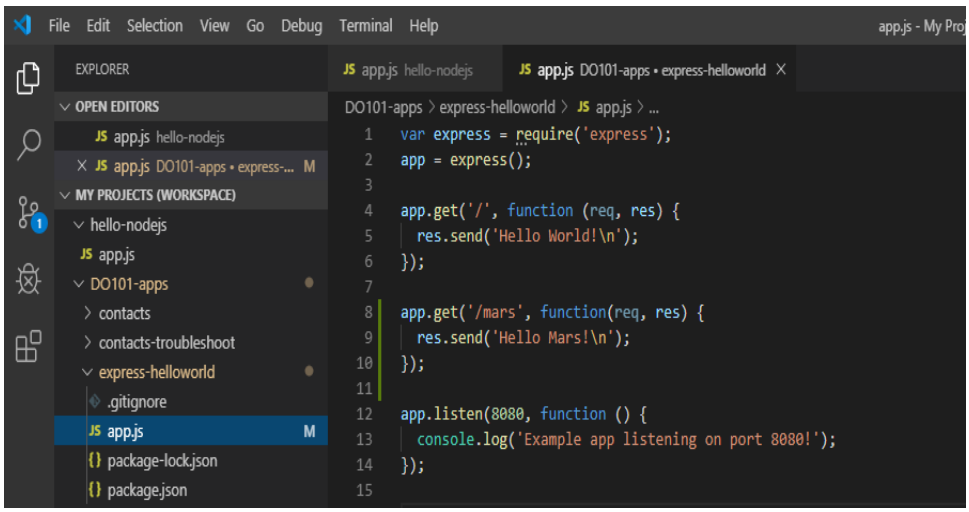


Figure 1.54: An added feature to the Express application.

   Save the `app.js` file (**File** → **Save**).

4. Install the application dependencies and execute the application. Verify that the application returns a `Hello Mars!` message when you access the `/mars` end point.

NOTE

This step requires the Node Package Manager (NPM). When you install the Node.js package on Ubuntu systems, NPM is not installed as a dependency. Thus, you must also install the NPM package on Ubuntu systems.

> The Ubuntu NPM package installs several different software development packages. You can skip this step if you need to minimize the number of installed packages on your Ubuntu system. Skipping this step does not prevent you from completing this exercise.

1. Right-click `express-helloworld` in the Explorer view (**View** → **Explorer**) and then select `Open in Terminal`.
2. Execute the `npm install` command in the integrated terminal to install the application dependencies.
3. Execute the `npm start` command in the integrated terminal to start the application.

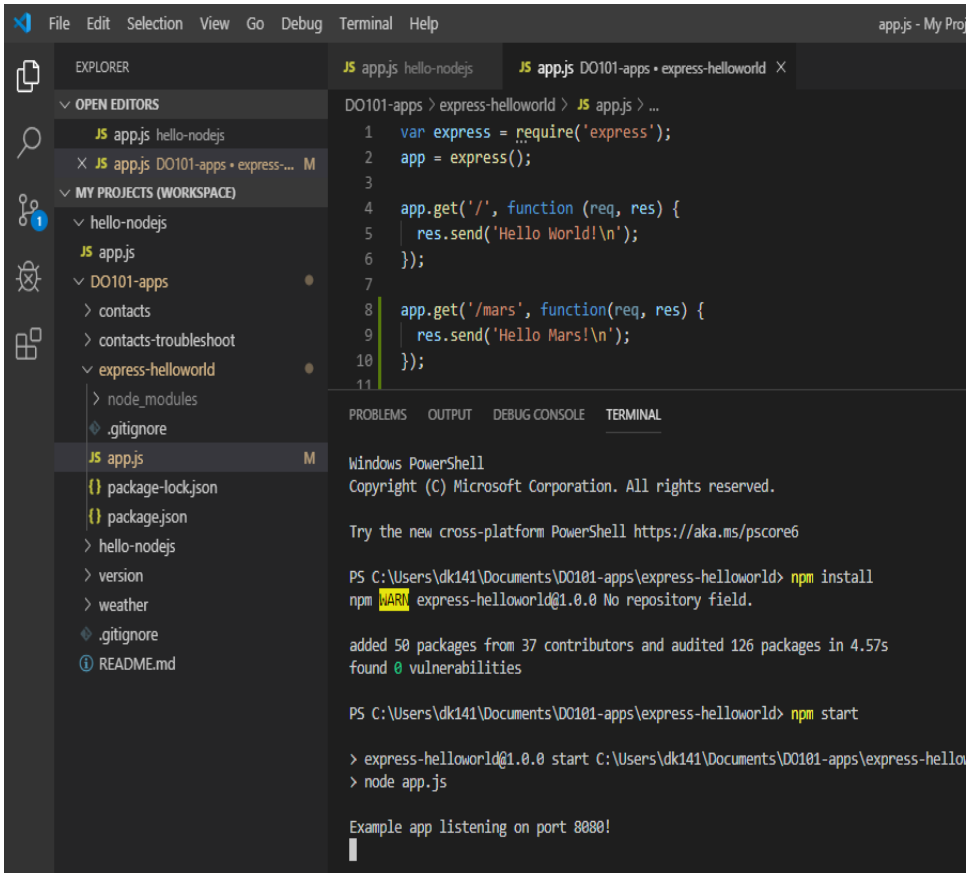


Figure 1.55: Executing a Node.js application from the integrated terminal.

4. In a browser, navigate to http://localhost:8080/. Verify that the application responds with a `Hello World!` message.
5. In a browser, navigate to http://localhost:8080/mars. Verify that the application responds with a `Hello Mars!` message.
6. To stop the application, click in the integrated terminal and type **Ctrl+C**. If a prompt displays, then provide an appropriate response to terminate the process.
7. To clean up, click the **Kill Terminal** icon to close the integrated terminal window.

5. Commit your changes locally, and then push the new commit to your GitHub repository.
   1. Access the Source Control view (**View** → **SCM**) and then click `DO101-apps` in the `SOURCE CONTROL PROVIDERS` list.
   2. Hover over the entry for the `app.js` file under the `CHANGES` heading for the `DO101-apps` repository. Click the `Stage Changes` icon to add the changes to the `app.js` file to your next commit.
   3. Add a commit message of `add /mars endpoint` in the message prompt, and then click the check mark button to commit the staged changes:

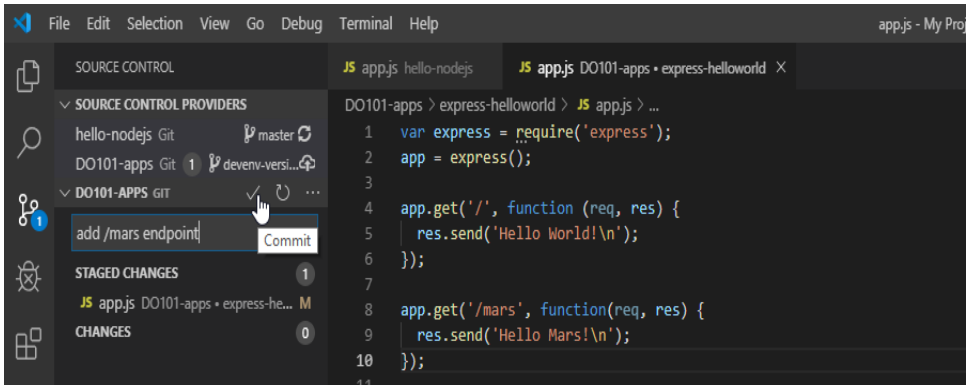


Figure 1.56: A commit message for a new feature.

6. Publish the `devenv-versioning` branch to your GitHub repository. Verify that your changes are present on GitHub.

    1. In the Source Control view (**View → SCM**), locate the `DO101-apps` entry under the `SOURCE CONTROL PROVIDERS` heading. Click the `Publish Changes` icon to publish the `devenv-versioning` branch to the remote repository. If a prompt displays, then provide your GitHub user name and password.

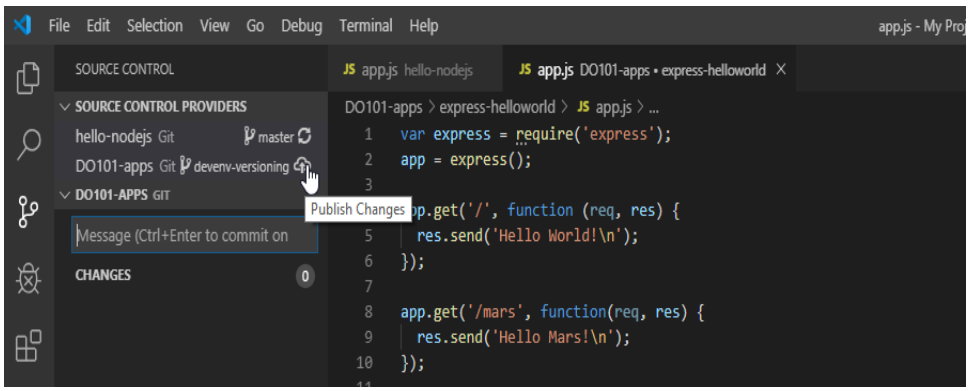


Figure 1.57: Publish a local branch to a remote repository.

    2. In a browser, navigate to your personal `DO101-apps` repository at `https://github.com/yourgituser/DO101-apps`. The GitHub interface indicates that you recently pushed code changes to the `devenv-versioning` branch.

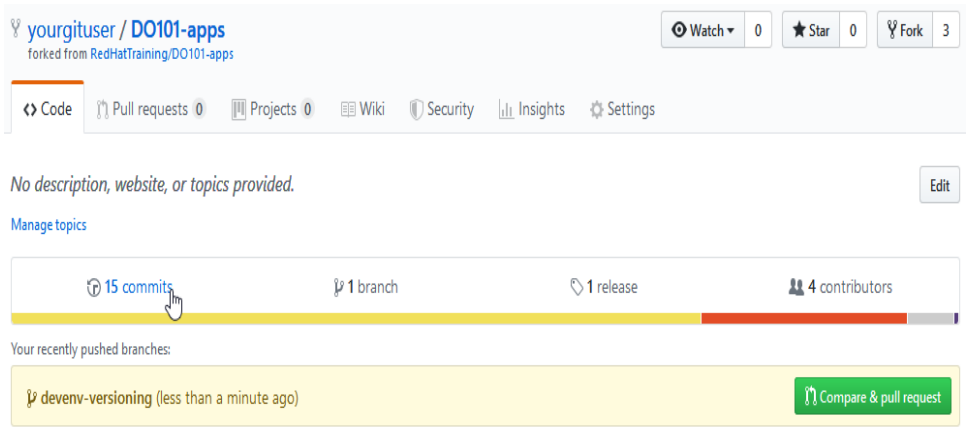


Figure 1.58: A GitHub repository indicates the presence of a new branch.

    3. Click `commits` below the `Code` tab to display a list of recent commits.

    4. From the branch list, select `devenv-versioning`. An entry for the `add /mars endpoint` commit displays at the top of the commit history.

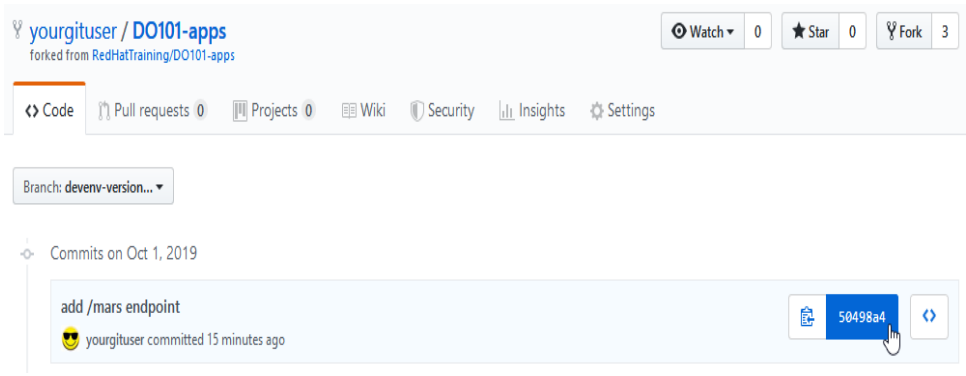


Figure 1.59: The commit history for a GitHub repository branch.

5. To the right of the `add /mars endpoint` entry, click the seven character commit hash. In the example shown, this value is *`50498a4`*.

   GitHub displays the four lines of code that you added to the `express-helloworld/app.js` file.

Figure 1.60: GitHub displays commit code changes.

You successfully pushed your code changes in a new branch to your GitHub repository.

This concludes the guided exercise.

Chapter 2. Deploying Applications to Red Hat OpenShift Container Platform

Deploying an Application to Red Hat OpenShift Container Platform

OBJECTIVES

After completing this section, you should be able to deploy an application to OpenShift.

Introducing OpenShift Container Platform

Red Hat OpenShift Container Platform is a self-service platform where development teams can deploy their applications. The platform integrates the tools to build and run applications, and manages the complete application life cycle from initial development to production.

OpenShift offers several deployment scenarios. One typical workflow starts when a developer provides the Git repository URL for an application to OpenShift.

The platform automatically retrieves the source code from Git, and then builds and deploys the application. The developer can also configure OpenShift to detect new Git commits, and then automatically rebuild and redeploy the application.

By automating the build and deployment processes, OpenShift allows developers to focus on application design and development. By rebuilding your application with every change you commit, OpenShift gives you immediate feedback. You can detect and fix errors early in the development process, before they become an issue in production.

OpenShift provides the building mechanisms, libraries, and runtime environments for the most popular languages, such as Java, Ruby, Python, PHP, .NET, Node.js, and many more. It also comes with a collection of additional services that you can directly use for your application, such as databases.

As traffic and load to your web application increases, OpenShift can rapidly provision and deploy new instances of the application components. For the Operations team, it provides additional tools for logging and monitoring.

OpenShift Container Platform Architecture

Red Hat OpenShift Online, at https://www.openshift.com/, is a public OpenShift instance run by Red Hat. With that cloud platform, customers can directly deploy their applications online, without needing to install, manage, and update their own instance of the platform.

Red Hat also provides the Red Hat OpenShift Container Platform that companies can deploy on their own infrastructure. By deploying your own instance of OpenShift Container Platform, you can fine tune the cluster performance specific to your needs. In this classroom, you will be provided access to a private OpenShift cluster.

Application Architecture

Several development teams or customers usually share the same OpenShift platform. For security and isolation between projects and customers, OpenShift builds and runs applications in isolated `containers`.

A container is a way to package an application with all its dependencies, such as runtime environments and libraries.

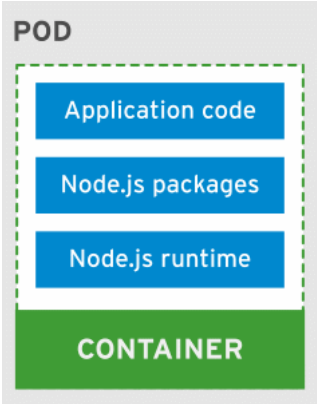


Figure 2.1: A container in a pod

The previous diagram shows a container for a Node.js application. The container groups the Node.js runtime, the Node.js packages required by the application, and the application code itself.

To manage the containerized applications, OpenShift adds a layer of abstraction known as the `pod`.

Pods are the basic unit of work for OpenShift. A pod encapsulates a container, and other parameters, such as a unique IP address or storage. A pod can also group several related containers that share resources.

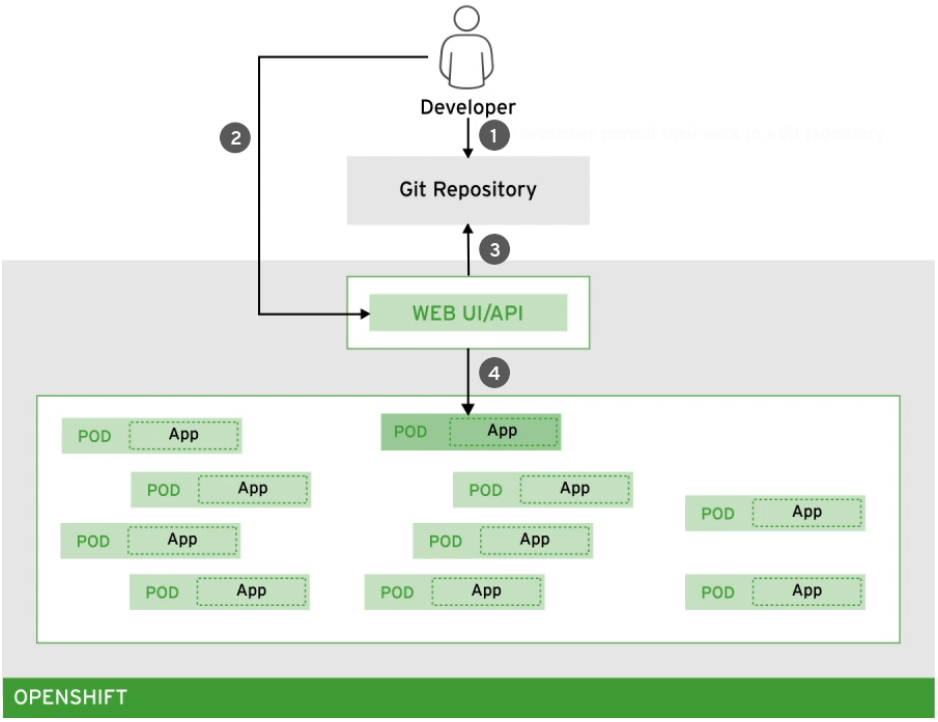


Figure 2.2: Pods running on OpenShift

The preceding diagram shows an OpenShift platform, hosting several applications, and running in pods. To deploy a new application, use the following workflow:

1. The developers commit work to a Git repository.
2. When ready to deploy their code, the developers use the OpenShift `web console` to create the application. The URL to the Git repository is one of the required parameters.
3. OpenShift retrieves the code from the Git repository and builds the application.
4. OpenShift deploys the application in a pod.

OpenShift Resource Types

OpenShift uses `resources` to describe the components of an application. When you deploy a new application, OpenShift creates those resources for you, and you can view and edit them through the web console.

For example, the Pod resource type represents a container running on the platform. A Route resource associates a public URL to the application, so your customers can reach it from outside OpenShift.

INTRODUCING THE DEVELOPER WEB CONSOLE FOR OPENSHIFT

The OpenShift web console is a browser-based user interface that provides a graphical alternative to the command-line tools. With the web UI, developers can easily deploy and manage their applications.

Logging in and Accessing the Developer Perspective

To access the web console, use the URL of your OpenShift platform. To use OpenShift, each developer must have an account. The following screen capture shows the login page.

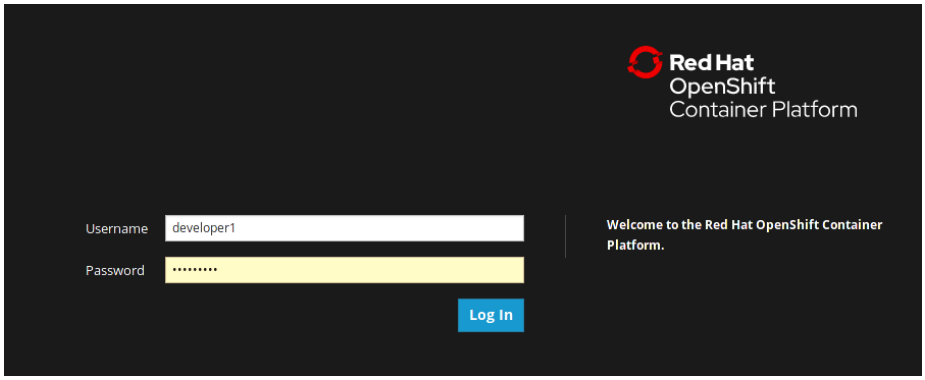


Figure 2.3: Logging in to OpenShift

The web console provides two perspectives, one for developers, and the other for administrators and operators. As shown in the following screen capture, the perspective is selected from the menu on the left. As a developer, you usually select the `Developer` perspective.

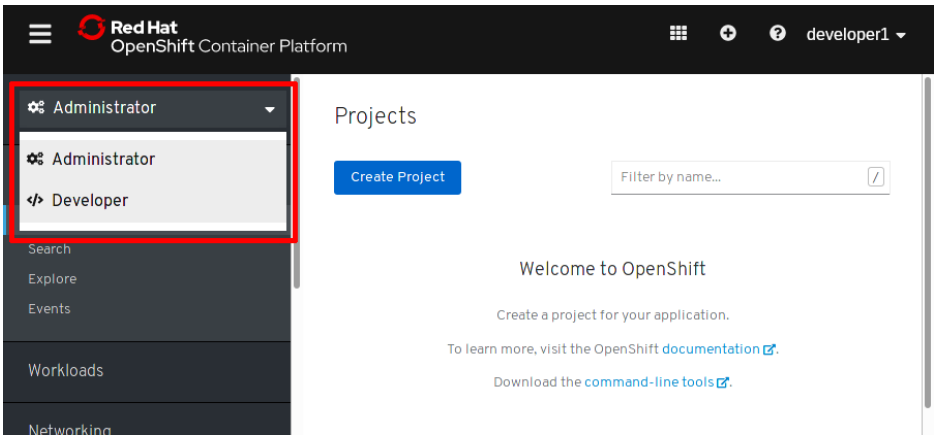


Figure 2.4: The Administrator and Developer perspectives

Creating a Project

OpenShift groups applications in `projects`. Using projects, developer teams can organize content in isolation from other teams. Projects enable both grouping the individual components of an application (front end, back end, and database), and creating life cycle environments (development, QA, production).

To create a project, use the **Advanced → Projects** menu.

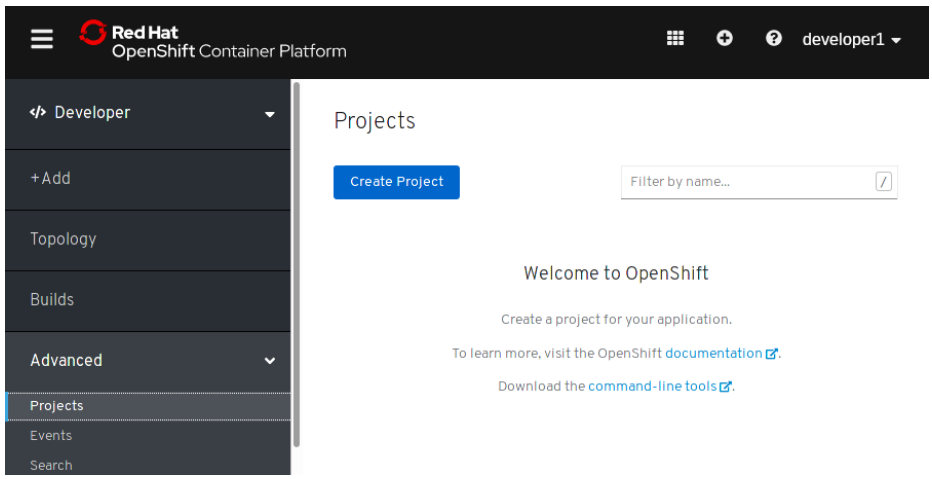


Figure 2.5: Creating a project

Deploying a New Application

OpenShift provides several methods to add a new application. The `Add` option is the entry point to an assistant that allows you to choose between the available methods to deploy an application to the OpenShift cluster as part of a specific project.

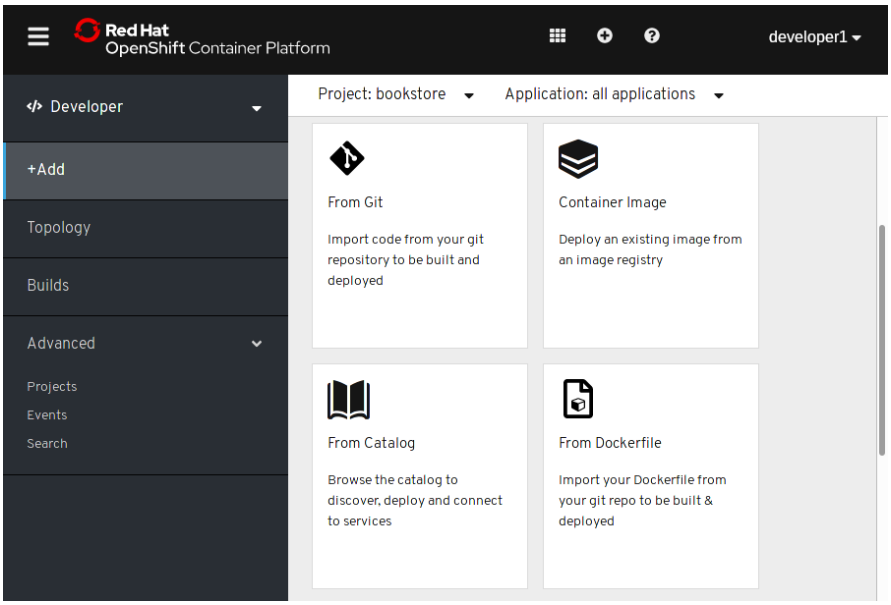


Figure 2.6: Selecting a method to deploy a new application in OpenShift

With the `From Catalog` method, you can list and deploy the available ready to use applications, for example, a MariaDB database. You can also select the language of your application, and provide its source code from a Git repository.

The following screen capture shows some of the available languages.

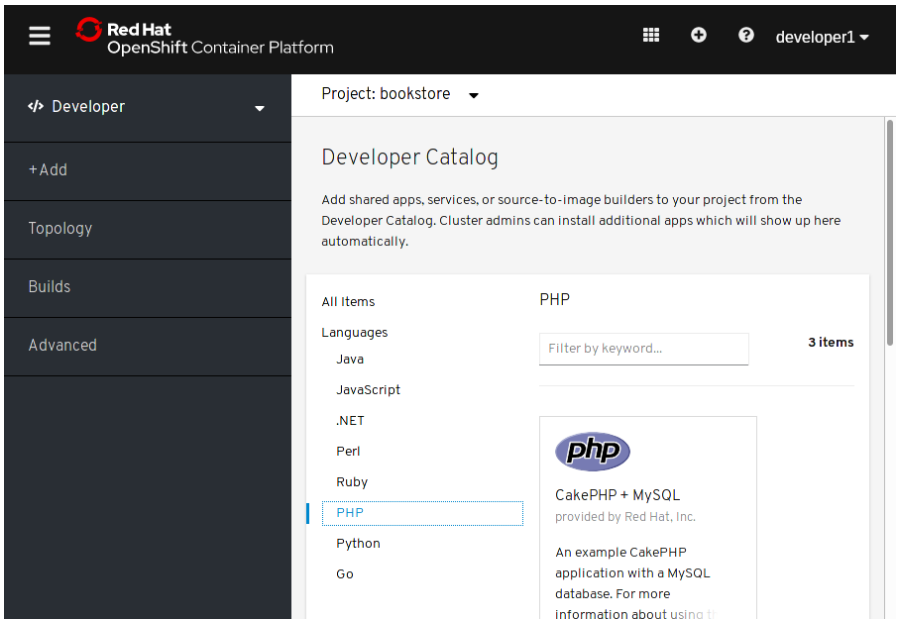


Figure 2.7: Available languages in the catalog

After selecting the application language from the catalog, OpenShift provides a form to collect the application details.

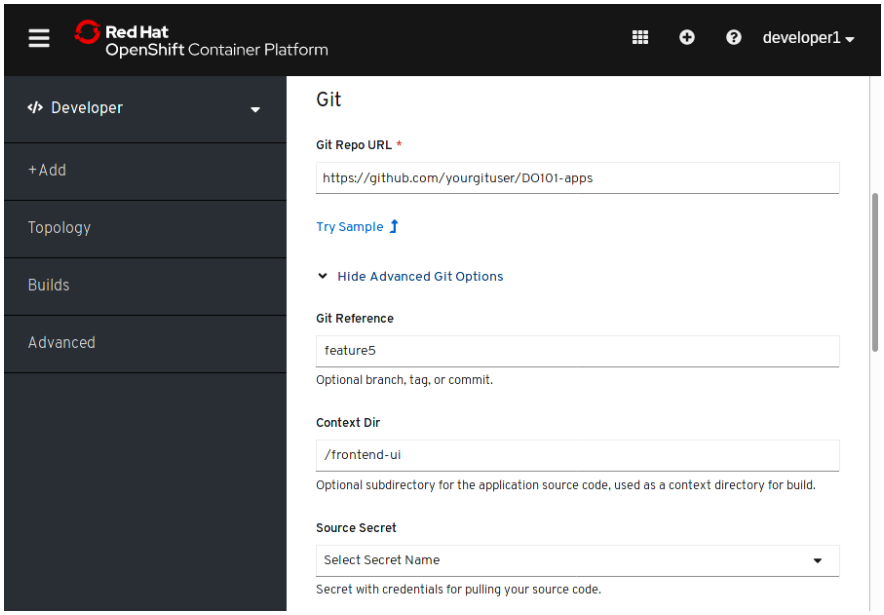


Figure 2.8: Deploying an application — Git details

Use the fields in the first section of the form to provide Git repository details for the application.

If the source code to deploy is not in the Git `master` branch, then provide the branch name in the `Git Reference` field. Use the `Context Dir` field when the application to deploy is in a subdirectory, and not at the root of the Git repository.

In the preceding screen capture, the application is stored in the `frontend-ui` subdirectory, under the https://github.com/yourgituser/DO101-apps Git repository. Also note that OpenShift should fetch the `feature5` branch, rather than the `master` branch.

Under the `General` section of the form, use the `Application Name` field to name the application. OpenShift uses this name to refer to your application in the web console pages. The `Name` field is an internal name that OpenShift uses to identify all the resources it creates during build and deployment. That name is used, for example, for the route resource that associates a public URL to the application.

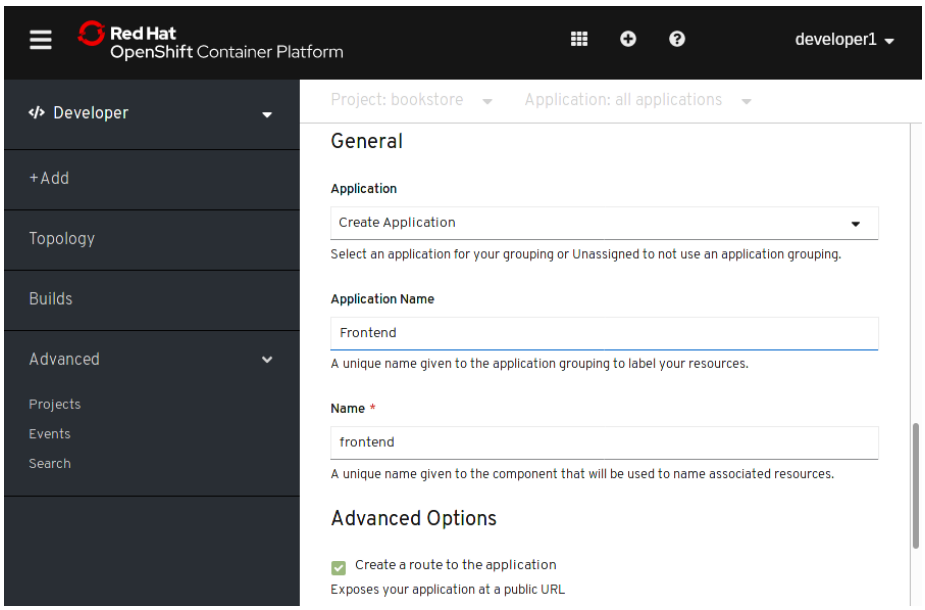


Figure 2.9: Naming the application

Reviewing an Application in the Web Console

The `Topology` option provides an overview of the applications in a project. The following screen capture shows two applications running in the `bookstore` project: the `Frontend` PHP application and a database.

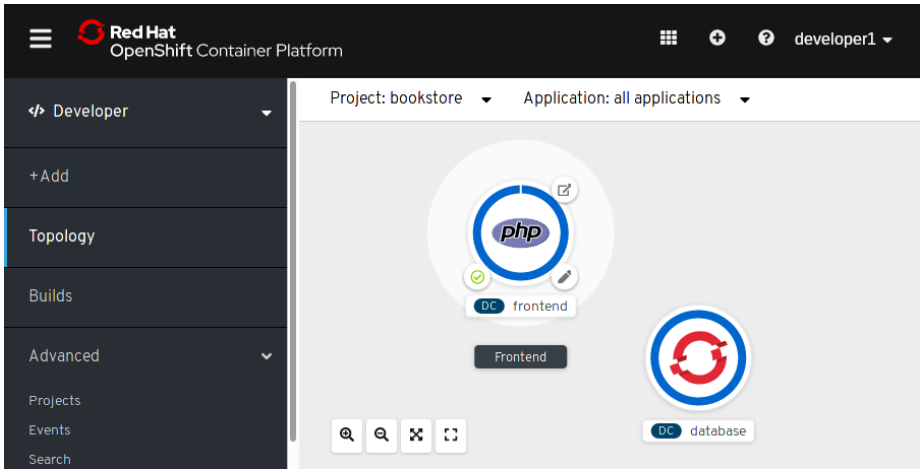


Figure 2.10: Overview of the applications in a project

Click the **application** icon to access application details. The following screen capture shows the resources that OpenShift creates for the application. Notice that one pod is running, the build is complete, and a route provides external access to the application.

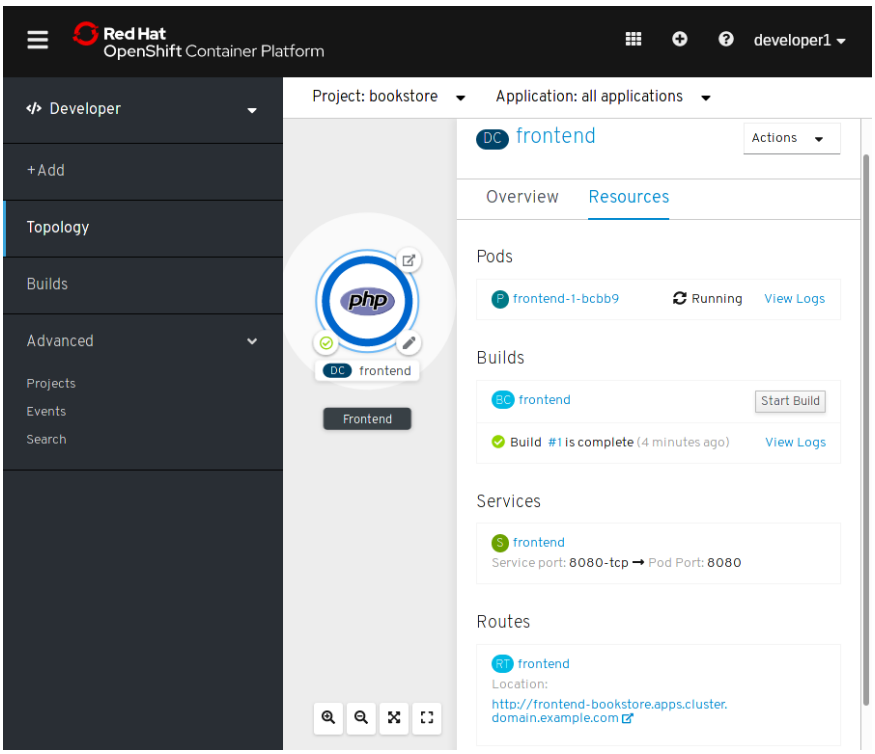


Figure 2.11: Resources of an application

Usually, when the web console displays a resource name, it is a link to the details page for that resource. The following screen capture displays the details of the Route resource.

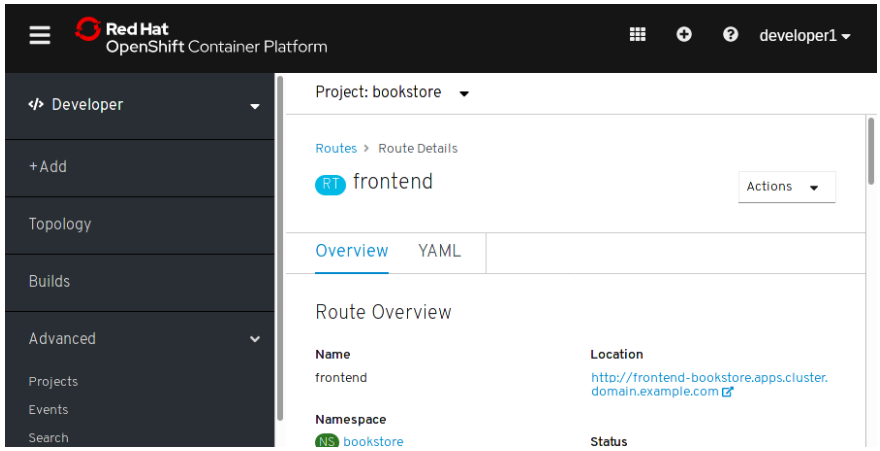


Figure 2.12: Application's route details

Editing OpenShift Resources

Most resource details pages from the OpenShift web console provide an **Actions** button that displays a menu.

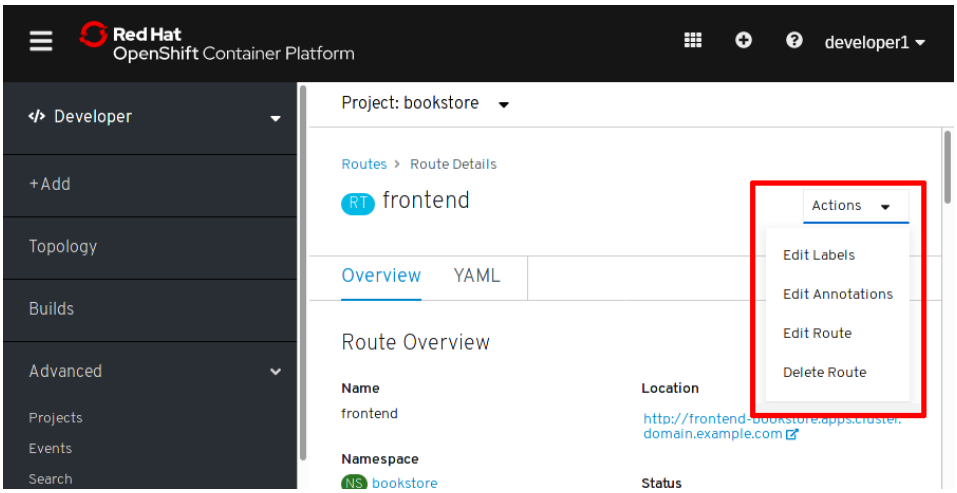


Figure 2.13: Available actions for a route resource

This menu usually provides options to edit and delete the resource.

Guided Exercise: Deploy an Application to Red Hat OpenShift Container Platform

In this exercise, you will use the OpenShift web console to deploy a Node.js application.

Outcomes

You should be able to use the OpenShift web console to:

- Create a new project.
- Add a new application from a Git repository.
- Inspect the resources that OpenShift creates during build and deployment.

- To perform this exercise, ensure that the `express-helloworld` Node.js application is in your GitHub `DO101-app` repository from the previous activity. Your changes should be in the `devenv-versioning` branch.
- Red Hat Training manages an OpenShift cluster dedicated to this course. Your Red Hat Training Online Learning environment provides access to this platform.

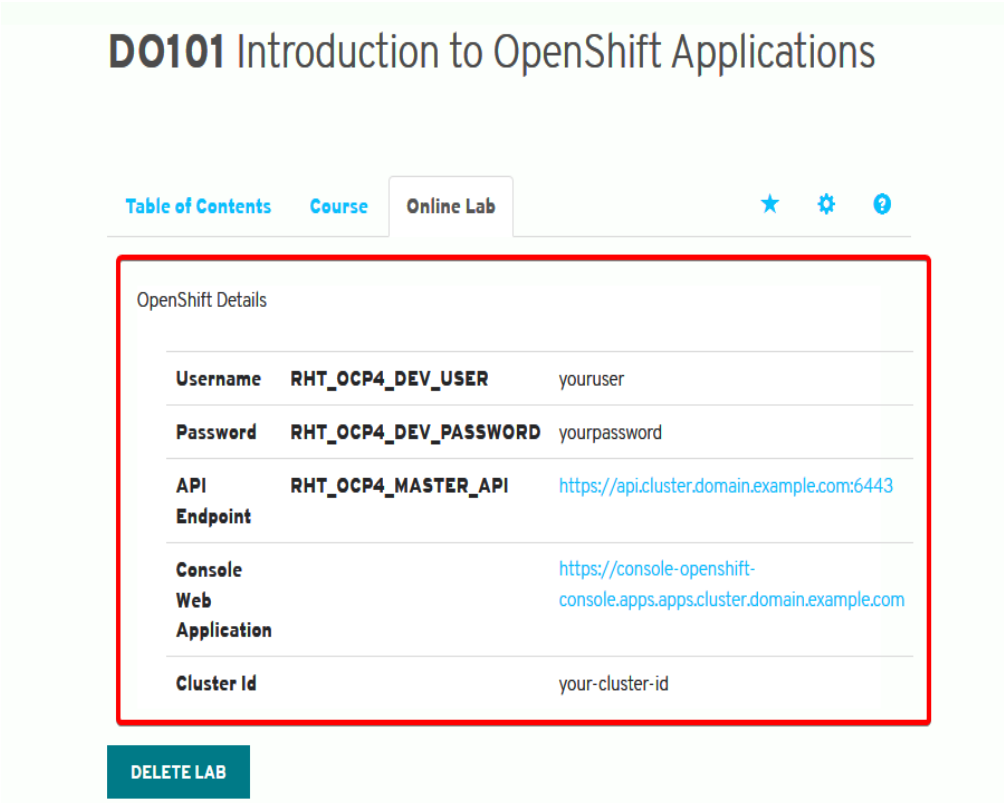


Figure 2.14: Details of the Red Hat Training Online Learning environment

In the exercise, use the `Console Web Application` URL and the provided user name and password to access the OpenShift web console.

Procedure 2.1. Steps

1. Open a web browser and access the OpenShift web console. Log in and switch to the `Developer` perspective.
    1. Open a web browser and navigate to *`https://console-openshift-console.apps.cluster.domain.example.com`* to access the OpenShift web console. Replace the URL with the value from your Red Hat Training Online Learning environment. The login page for the web console displays.
    2. Log in with the user name and password from your Red Hat Training Online Learning environment.
    3. From the list at the top of the menu on the left, select the `Developer` perspective.

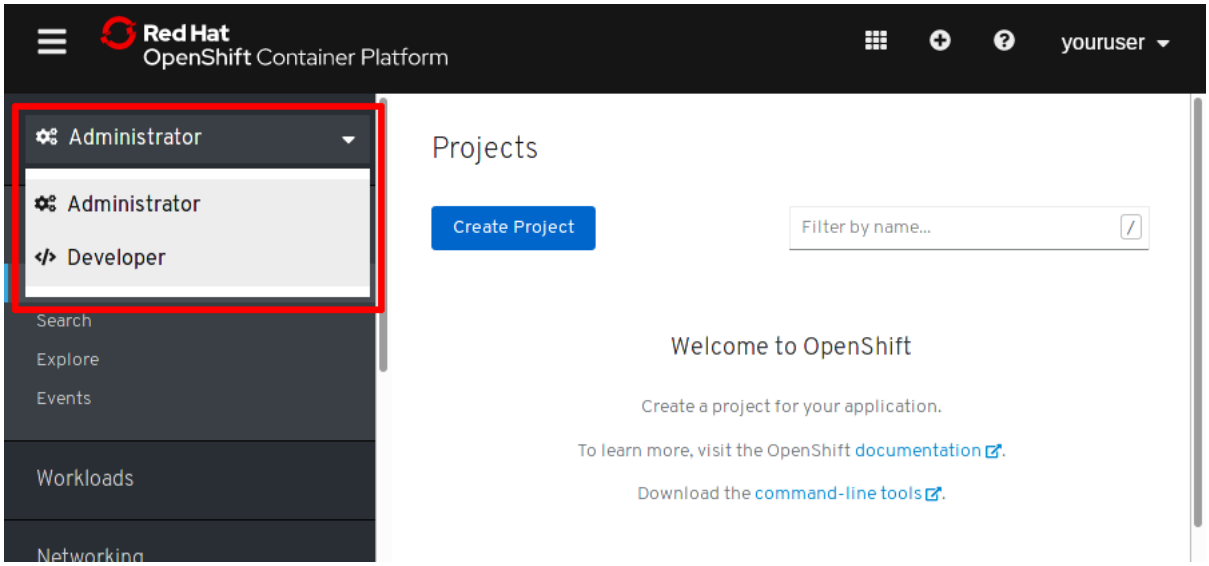


Figure 2.15: Accessing the Developer perspective

The page may show a restricted access warning because you do not yet have access to a project. You create a project in the next step.

2. Create a new project named *youruser*`-deploy-app`. Replace `youruser` with your user name.

    1. Use the **Advanced** → **Projects** menu and click **Create Project**.

    2. Enter *youruser*`-deploy-app` in the `Name` field. Replace `youruser` with your user name. Leave the other fields empty and click **Create**.

3. Create a new JavaScript Node.js application named `helloworld`. Use the code you pushed to the Git repository in the previous exercise. The code is in the `express-helloworld` subdirectory. The branch name is `devenv-versioning`.

    1. Click the `Add` tab on the left side of the page and then click **From Catalog**.

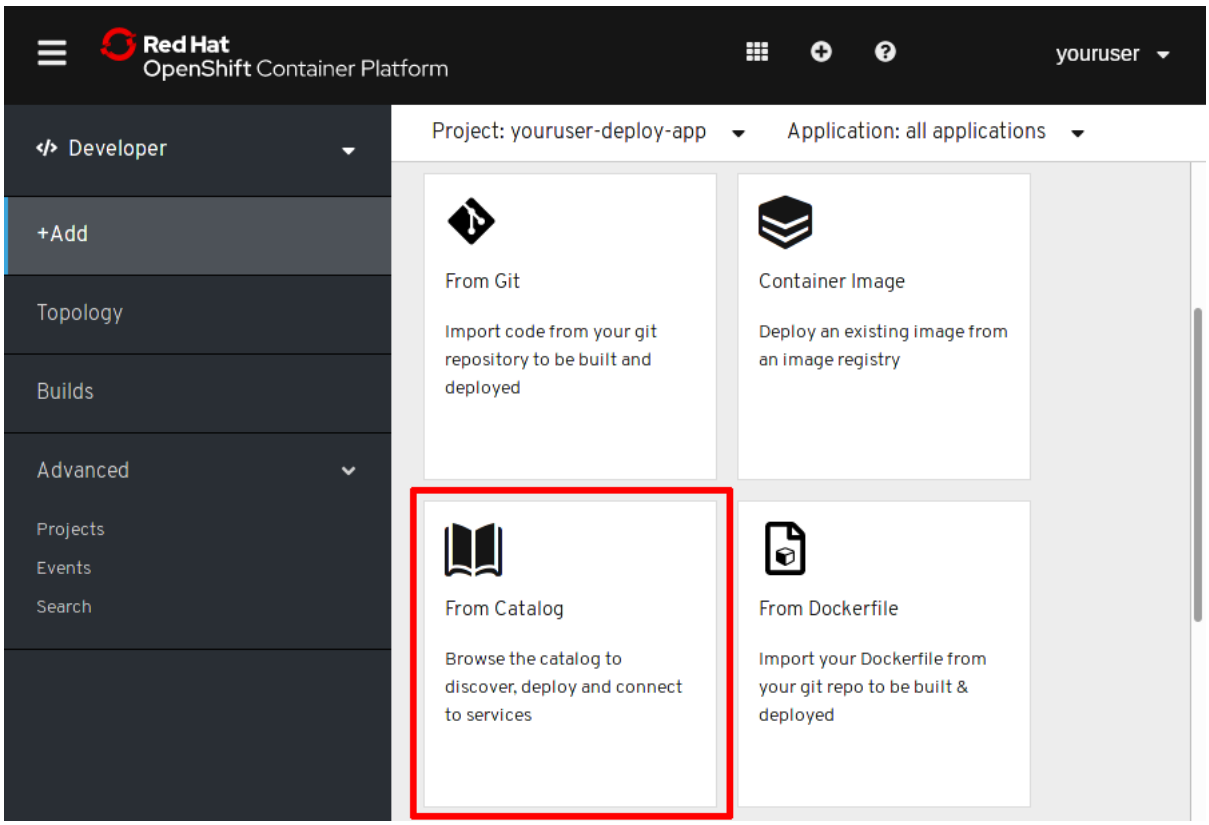


Figure 2.16: Adding a new application

2. Click **Languages** → **JavaScript** and then click the first option, `Node.js`. Click **Create Application** to enter the details of the application.

3. Complete the form according to the following table. To access the Git parameters, click **Show Advanced Git Options**.

   Table 2.1. New Application Parameters

Parameter	Value
`Git Repo URL`	https://github.com/*`yourgituser`*/DO101-apps
`Git Reference`	`devenv-versioning`
`Context Dir`	`/express-helloworld`
`Application Name`	`helloworld`
`Name`	`helloworld`
`Create a route to the application`	Make sure the box is checked

   To avoid deployment errors in the following steps, review the values you entered in the form before proceeding. Click **Create** to start the build and deployment processes.

4. Wait for OpenShift to build and deploy your application.

   1. The web console should automatically show the `Topology` page. If necessary, click the `Topology` tab on the left side of the page.

   2. Wait for the application icon to change to the build complete status.

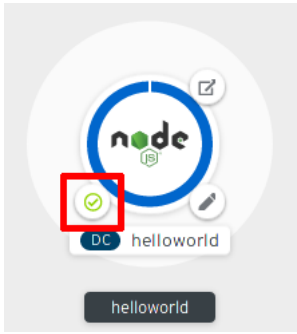


Figure 2.17: Application deployment complete

   The build and deploy processes can take up to two minutes to complete.

5. Review the application resources and confirm that you can access the application from the internet.

   1. Click the `Node.js` icon to access the application details.

   2. Click the `Resources` tab to list the resources that OpenShift created during the application deployment.

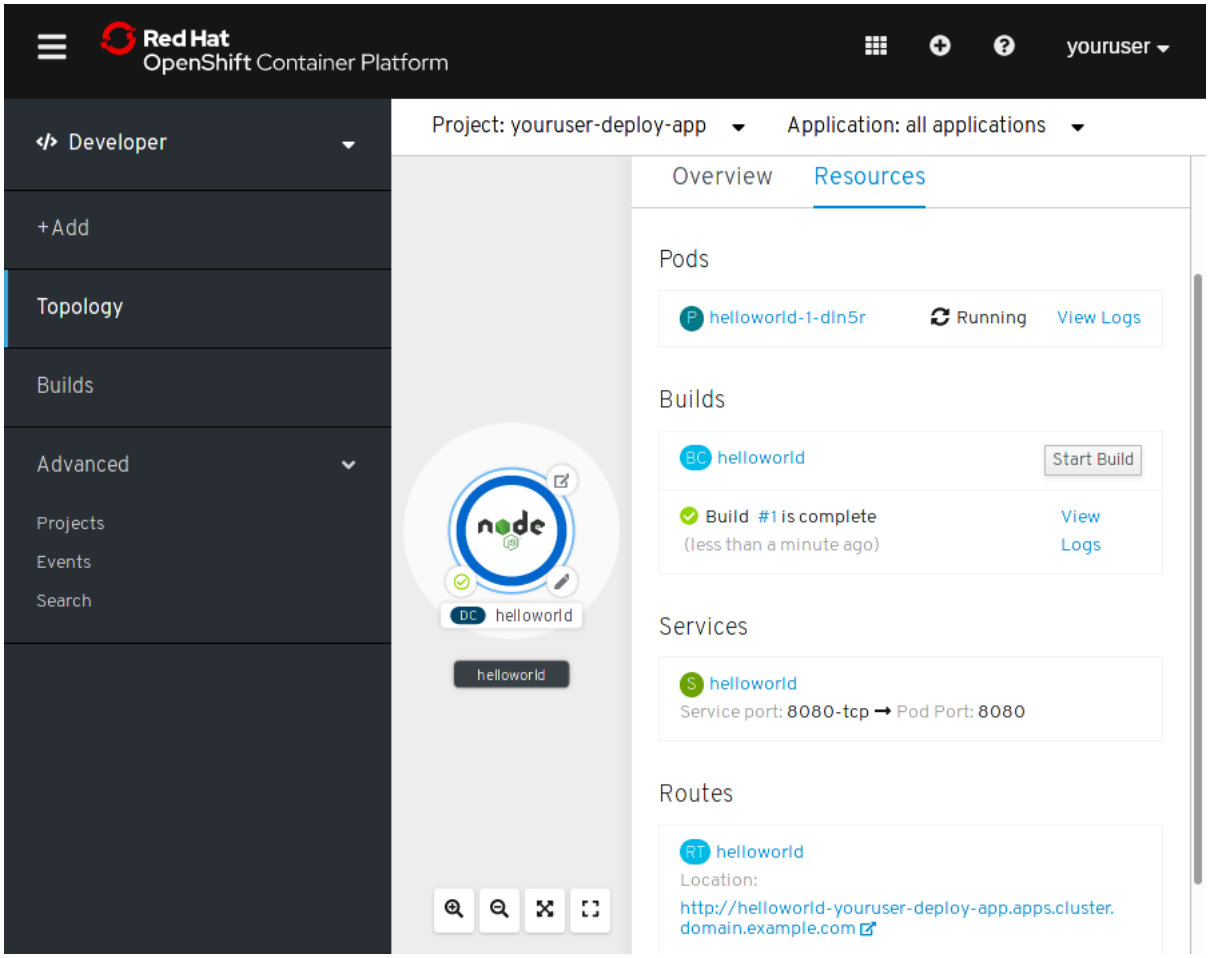


Figure 2.18: Reviewing the application resources

Notice the pod resource. This pod hosts the application running on the platform.

The build resource collects details of the application build process. Notice that the build is complete.

The route resource associates a public URL to the application.

3. Click the `Location` link in the route resource. Your web browser opens a new tab and accesses the application public URL.

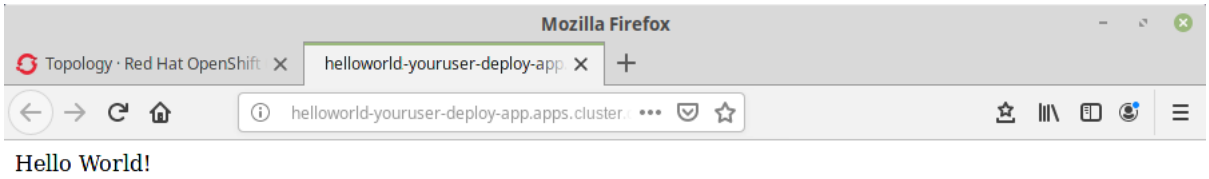


Figure 2.19: Accessing the helloworld application

When done, close the browser tab.

6. To clean up your work, delete the `youruser-deploy-app` project. When you delete a project, OpenShift automatically removes all its resources.

    1. Use the **Advanced** → **Projects** menu to list all your projects.

    2. Click the menu button at the end of the `youruser-deploy-app` project row, and then click `Delete Project`.

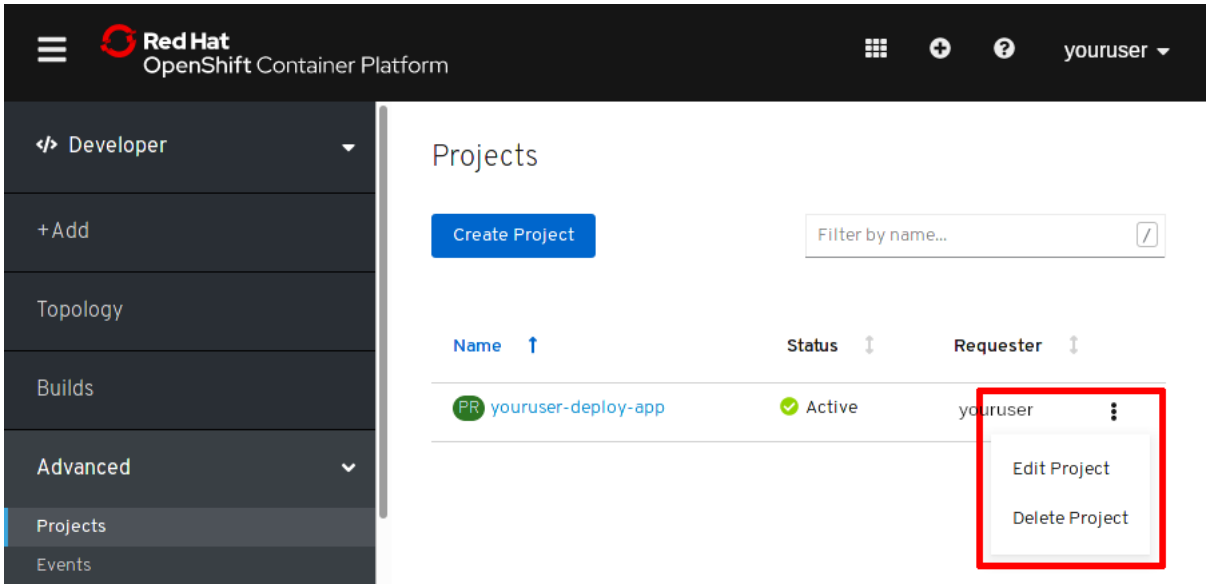


Figure 2.20: Deleting a project

To confirm the deletion of the project, enter the project name in the confirmation window, and then click **Delete**.

3. Log out of the web console. To log out, click your login name in the top right corner and then click **Log out**.

This concludes the guided exercise.

Chapter 3. Configuring Application Builds in OpenShift Updating an Application

Objectives

After completing this section, you should be able to update an application deployed on Red Hat OpenShift Container Platform.

Building and Updating Applications

To deploy applications on OpenShift, you must create a *container image*. A container image is a binary package containing an application and all of its dependencies, including the operating system.

In OpenShift, a *build* is the process of creating a runnable container image from application source code. A *BuildConfig* resource defines the entire build process.

OpenShift can create container images from source code without the need for tools such as Docker or Podman. After they are built, application container images are stored and managed from a built-in *container registry* that comes bundled with the OpenShift platform.

OpenShift supports many different ways to build a container image. The most common method is called *Source to Image* (S2I). In an S2I build, application source code is combined with an *S2I builder image*, which is a container image containing the tools, libraries, and frameworks required to run the application.

For example, to run Node.js applications on OpenShift, you will use a Node.js S2I builder image. The Node.js S2I builder image is a container image configured with the Node.js runtime, package management tools (NPM), and other libraries required for running Node.js applications.

OpenShift can automatically detect the type of application, and choose an appropriate S2I builder image to build the final application container image.

For example, if the root of the application source code tree contains a `package.json` file, OpenShift will automatically select the latest Node.js S2I builder image for the build. If desired, you can override this default selection and choose your own S2I builder image for the build process.

Manually Triggering Builds

After an application is deployed on OpenShift, then OpenShift can rebuild and redeploy a new container image anytime the application source code is modified. Use either the
`oc` command line client, or the OpenShift web console to trigger a new build of the updated application.

The workflow for an application deployed from GitHub when using the manual build process is as follows:

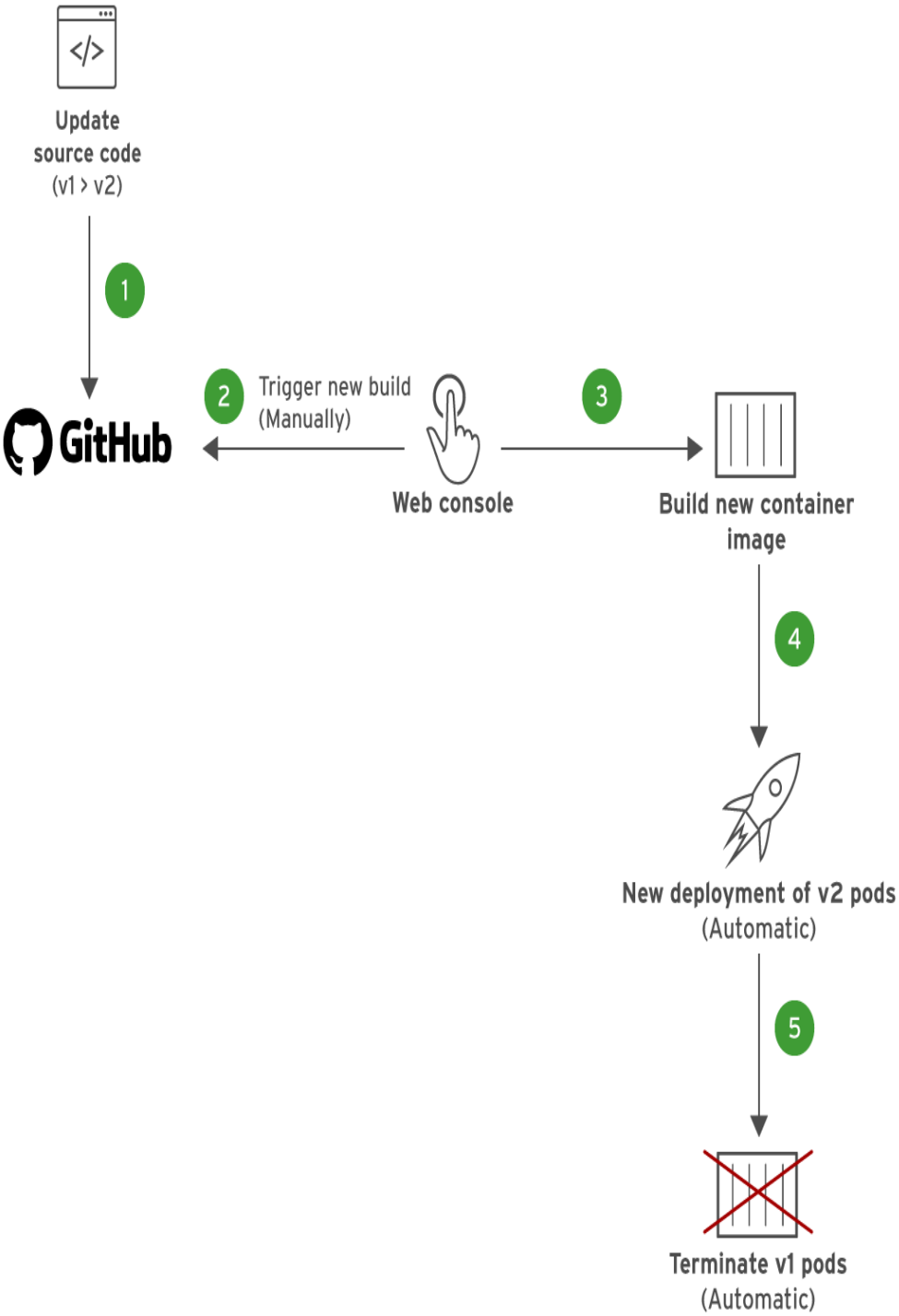


Figure 3.1: Manual Builds

1. Update source code for an existing application, such as from v1 to v2, and then commit the changes to GitHub.
2. Manually trigger a new build using the OpenShift web console, or the OpenShift client command line interface (CLI).
3. OpenShift builds a new container image with updated code.
4. OpenShift rolls out new pods based on the new container image (v2).
5. After the new pods based on v2 are rolled out, OpenShift directs new requests to the new pods, and terminates the pods based on the older version (v1).

Automatic Builds using Webhooks

A *Webhook* is a mechanism to subscribe to events from a source code management system, such as GitHub. OpenShift generates unique webhook URLs for applications that are built from source stored in Git repositories. Webhooks are configured on a Git repository. Based on the webhook configuration, GitHub will send a HTTP POST request to the webhook URL, with details that include the latest commit information.

The OpenShift REST API listens for webhook notifications at this URL, and then triggers a new build automatically. You must configure your webhook to point to this unique URL.

The workflow for an automatic rebuild triggered by webhooks is as follows:

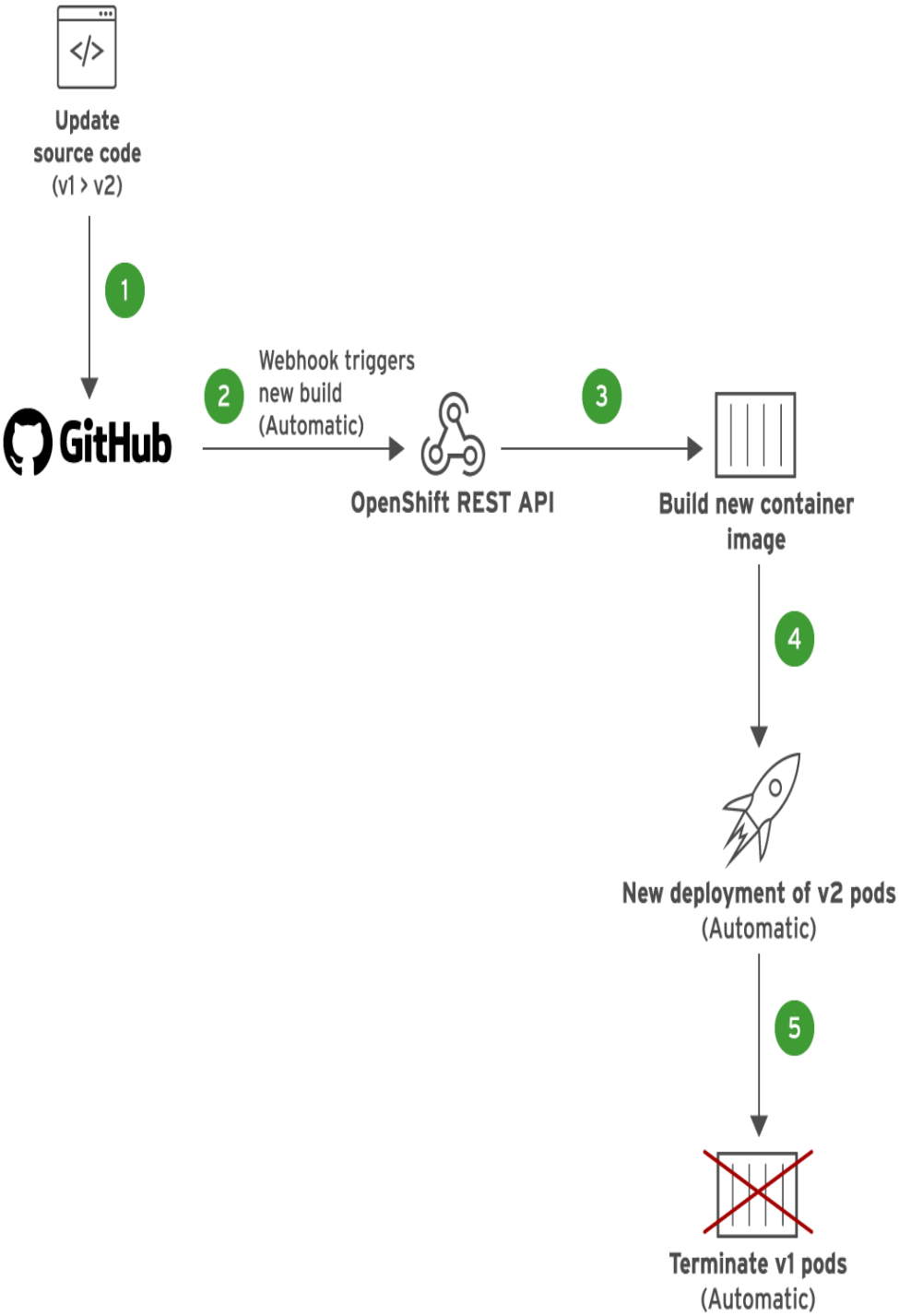


Figure 3.2: Automatic builds using Webhooks

1. Update source code for an existing application (from v1 to v2) and commit the changes to GitHub.
2. The GitHub webhook sends an event notification to the OpenShift REST API.
3. OpenShift builds a new container image with the updated code.
4. OpenShift rolls out new pods based on the new container image (v2).
5. After the new pods based on v2 are rolled out, OpenShift directs new requests to the new pods, and terminates the pods based on the older v1.

Guided Exercise: Updating an Application

In this exercise, you will update the source code for a Node.js application deployed on OpenShift.

Outcomes

You should be able to:

- Create a new OpenShift application using the `oc new-app` command.
- Trigger a new build manually from the OpenShift web console, after updating the source code of an application.
- Set up webhooks in GitHub to automatically trigger new builds when there are new commits to the Git repository.

To perform this exercise, ensure that you have access to:

- A running Red Hat OpenShift Container Platform cluster.
- The source code for the `version` application in the `DO101-apps` Git repository on your local system.

Procedure 3.1. Steps

1. Install the OpenShift command line interface (CLI).
    1. Download the OpenShift CLI binary for your platform from https://mirror.openshift.com/pub/openshift-v4/clients/ocp/latest/.
    2. Download the compressed files of the form `openshift-client-<platform>-<version>.*` .

    WARNING

    Do not download the `openshift-install-*` files.

    3. Unzip the compressed archive file, and then copy the `oc` binary to a directory of your choice. Ensure that this directory is in the `PATH` variable for your system. On macOS and Linux, copy the `oc` binary to `/usr/local/bin`:

    ```
    $ sudo cp oc /usr/local/bin/
    $ sudo chmod +x /usr/local/bin/oc
    ```

    For Windows 10 systems, decompress the downloaded archive, and then follow the instructions at https://www.architectryan.com/2018/03/17/add-to-the-path-on-windows-10/ to edit the `PATH` environment variable and add the full path to the directory where the `oc` binary is located.

    4. Verify that the `oc` binary works for your platform. Open a new command line terminal and run the following:

    ```
    $ oc version
    Client Version: openshift-clients-4.2.0-201910041700
    ```

    Your output may be slightly different based on the version of the OpenShift client that you downloaded.

    NOTE

    To use the `oc` command from within the VS Code embedded terminal, restart the IDE after installing it.

2. Deploy the `version` application to OpenShift using the `oc` tool.
    1. Log in to your OpenShift account using the API end point URL that was provided to you:

    You will be prompted for a user name and password. Use the same user name and password that you used to log in to the OpenShift web console.

    WARNING

The API end point URL and the OpenShift web console URL are **not** the same. The API end point URL is used mainly to interact with OpenShift, using command line tools like `oc` .

2. Create a new project called *youruser*`-version`.
3. Launch the Visual Studio Code (VS Code) editor. In the Explorer view (**View** → **Explorer**), open the `DO101-apps` folder in the `My Projects` workspace. The source code for the `version` application is in the `version` directory.
4. In the source control view in VS Code (**View** → **SCM**), ensure that the `DO101-apps` entry under `SOURCE CONTROL PROVIDERS` shows the `master` branch. If you were working with another branch for a different exercise, click on the current branch and then select `master` in the `Select a ref to checkout` window to switch to the `master` branch.

   WARNING

   Each exercise uses a unique branch. Always create a new branch using `master` as the base.

5. Click **master** in the `DO101-apps` entry under `SOURCE CONTROL PROVIDERS`.

   Select `Create new branch…` from the list of options.

6. At the prompt, enter `update-app` for the branch name. The Source Control view updates the `DO101-apps` entry with the new branch name.
7. Push the `update-app` branch to your `DO101-apps` GitHub repository.

   Click the Publish Changes cloud icon next to the `update-app` branch to push your local branch to your remote Git repository. If prompted, then provide your GitHub user name and password.

8. Deploy the `version` application from your `DO101-apps` Git repository.

   Switch to a command line terminal. Enter the command below in a single line with no line breaks. Do not type the \ character at the end of the lines. The command is broken into multiple lines in this example for clarity and formatting purposes only:

   NOTE

   The `#update-app` indicates that OpenShift should use the `update-app` Git branch, which was created in the previous step. The `--context-dir` parameter indicates the directory under the `DO101-apps` repository where the application source code is stored.

3. Test the application.

   1. Open a web browser and access the OpenShift web console. Log in to the OpenShift web console using your developer account.
   2. Switch to the `Developer` perspective.
   3. Click `Topology` in the navigation pane. From the `Project:` list, select the `yourname-version` project.
   4. Click the `version` deployment config, and then click the `Overview` tab. Verify that a single pod is running. You may have to wait for a minute or two for the application to be fully deployed.

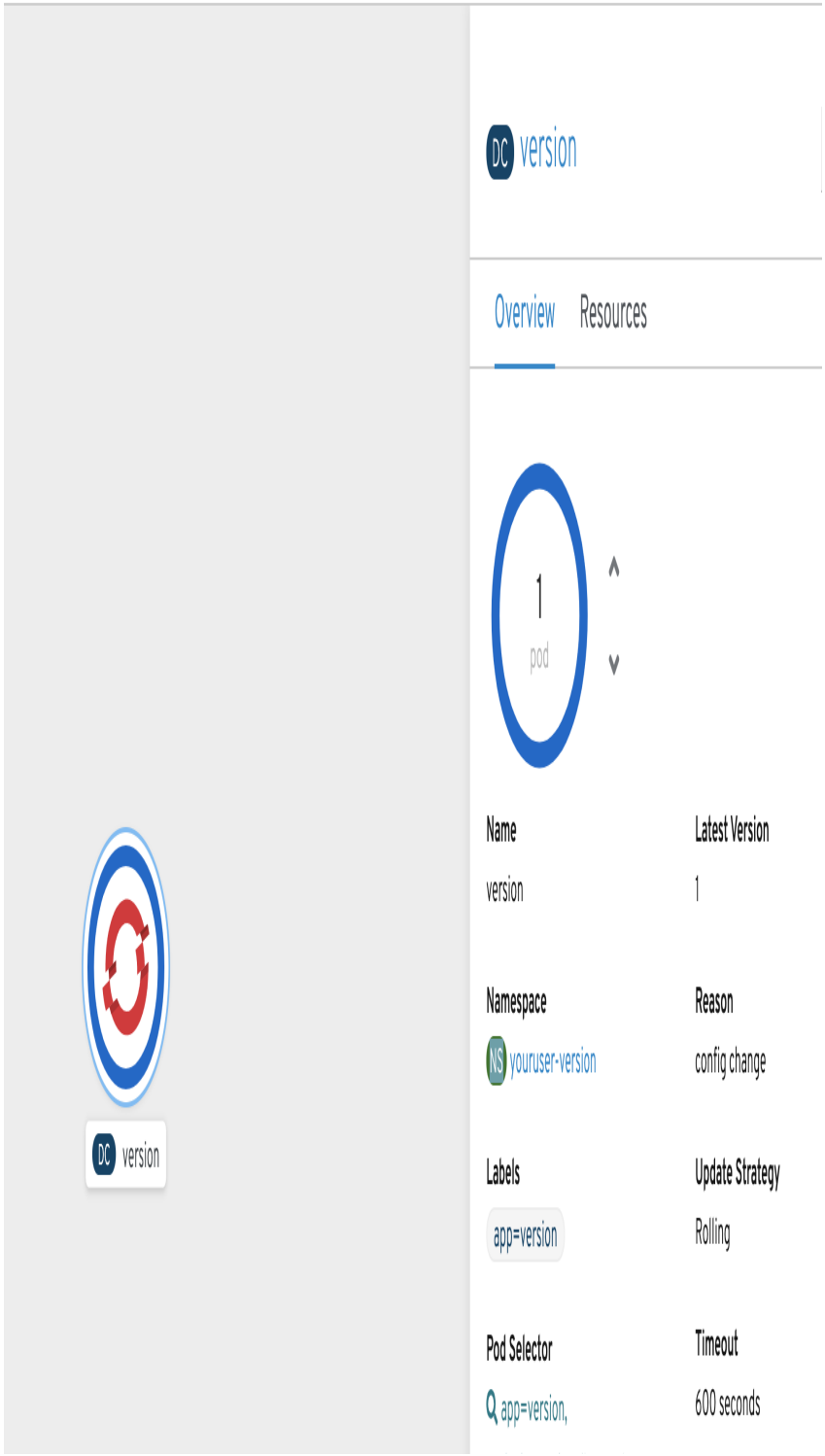


Figure 3.3: Application deployment complete

5. Applications created using the `oc new-app` command do not create a route resource by default. Run the following command to create a route and allow access to the application from the internet:

```
$ oc expose svc/version
route.route.openshift.io/version exposed
```

NOTE

You can also create a route using the OpenShift web console; select the `Administrator` perspective, and then click on **Networking → Routes**.

6. View the `Topology` page again, and observe that the `version` deployment now displays an icon to open a URL.

   Click `Open URL` to view the application.

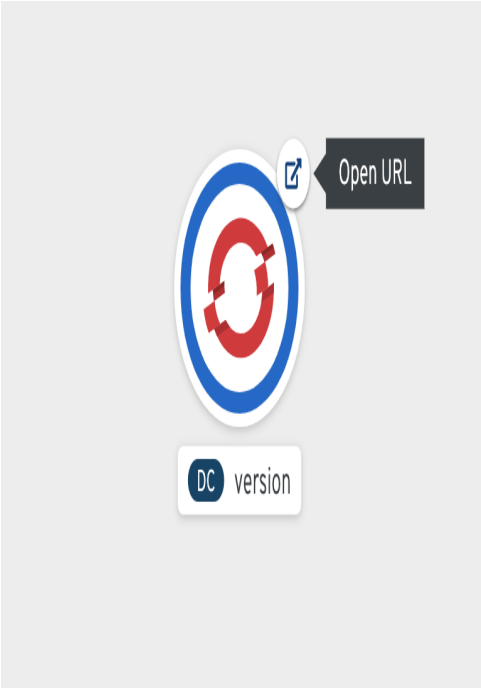


Figure 3.4: Route URL

7. The application opens in a new browser tab. You should see version 1 of the application as output.

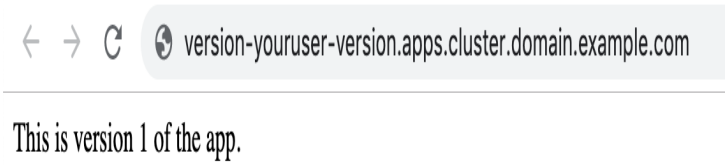


Figure 3.5: Application version 1

Close the browser tab.

4. Update the `version` application.

1. In the VS Code explorer view (**View** → **Explorer**), click the `app.js` file in the `version` application folder of the `DO101-apps` folder. VS Code opens an editor tab for the `app.js` file.

2. Change the `response` variable to version 2 as follows:

3. Save the changes to the file.

4. Commit your changes locally, and then push the new commit to your GitHub repository.

   Access the Source Control view (**View** → **SCM**). Locate the entry for the `app.js` file under the `CHANGES` heading for the `DO101-apps` directory.

   Click the plus (+) button to add the `app.js` file changes to your next commit.

5. Add a commit message of `updated app to v2` in the message prompt, and then click the check mark button to commit the staged changes.

6. Locate the `DO101-apps` entry in the `SOURCE CONTROL PROVIDERS` heading.

   Click the `Synchronize Changes` icon to publish the changes to the remote repository.

   If prompted that this action will push and pull commits to and from the origin, then click `OK` to continue. If prompted, then provide your GitHub user name and password.

1. Trigger a new build manually using the OpenShift web console.

   1. Click on `Builds` in the navigation pane to view the `Build Configs` page.

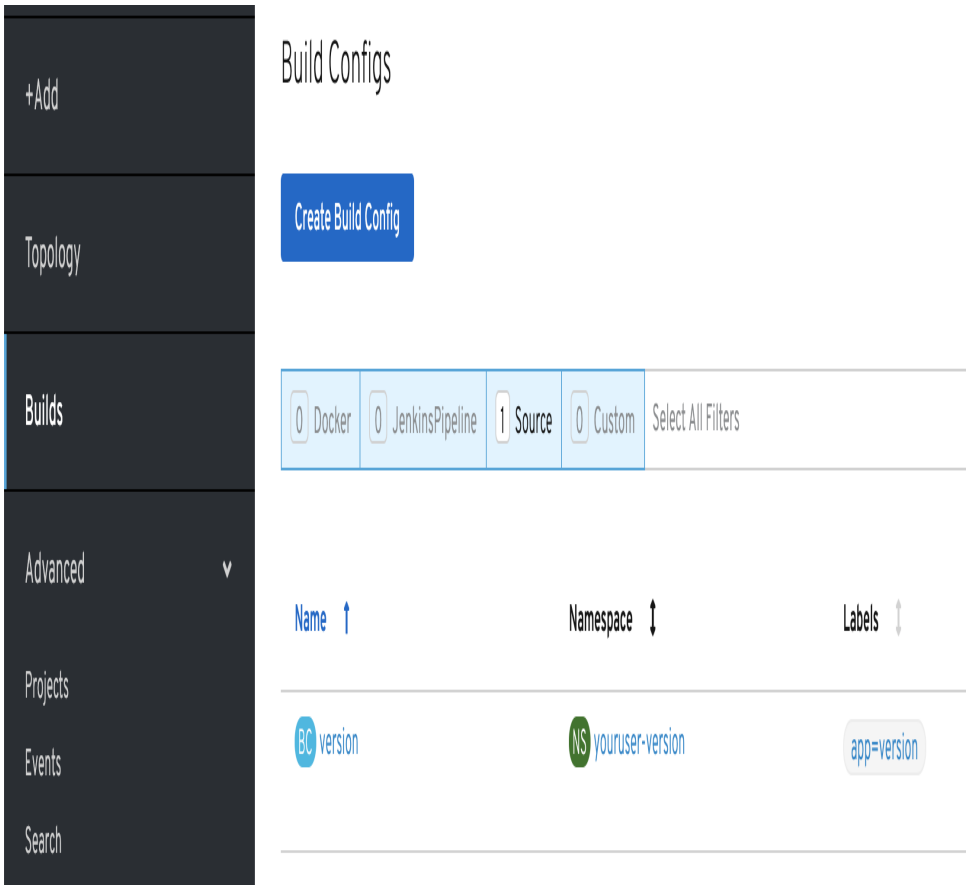


Figure 3.6: Build Configs Page

   2. Expand the menu to the right of the `version` build config, and then click `Start Build`.

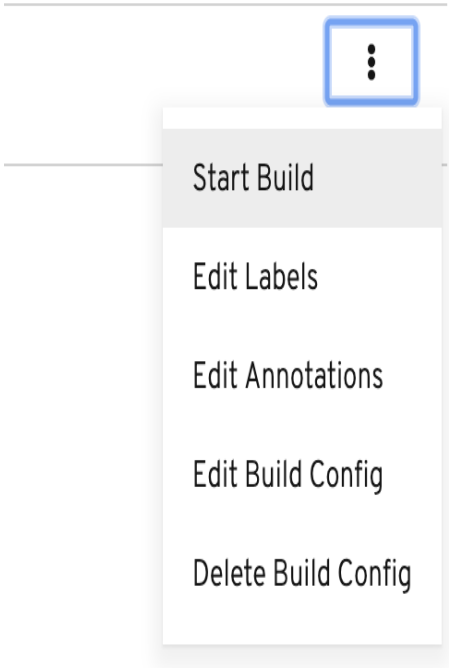


Figure 3.7: Start Build

3. You should now see the details page for the build `version-2`.

   Click on the `Logs` tab to see the progress of the build. A new container image with the updated source code is built and pushed to the OpenShift image registry. Wait until the application container image is built and deployed. You should see a `Push successful` message in the logs before proceeding to the next step.

4. Click on `Topology` in the navigation pane, and then click `Open URL` from the `version` deployment. Version 2 of the application is displayed.

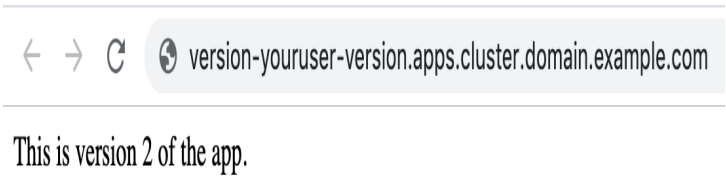


Figure 3.8: Version 2 of the application

   Close the browser tab.

2. Set up webhooks in GitHub to automatically trigger a new build when you commit changes to the repository.

1. Navigate to your `DO101-apps` repository (https://github.com/*`yourgituser`*/DO101-apps) on GitHub using a web browser. Log in to your GitHub account if prompted.

2. Click on the `Settings` tab to open the settings page for the repository.

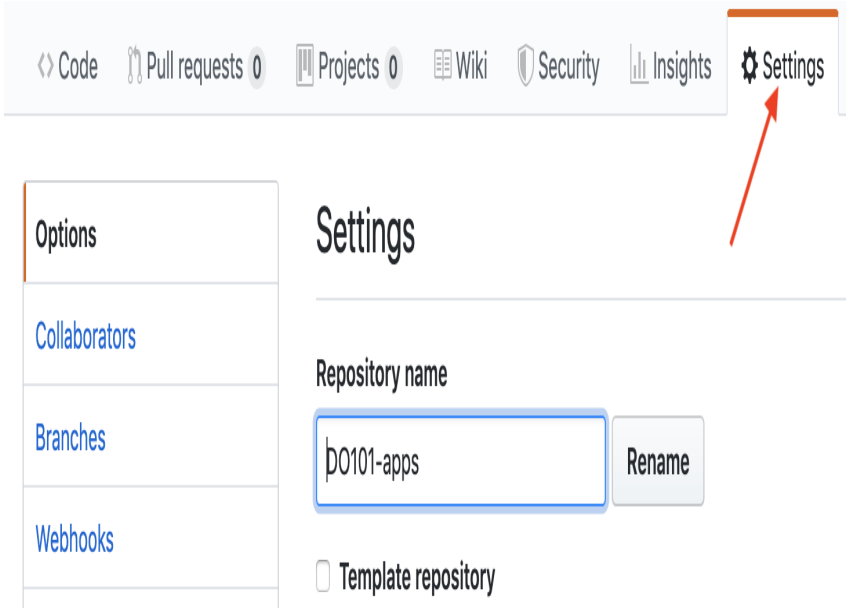


Figure 3.9: GitHub repository settings

3. Click on `Webhooks` in the left menu to open the webhooks page.

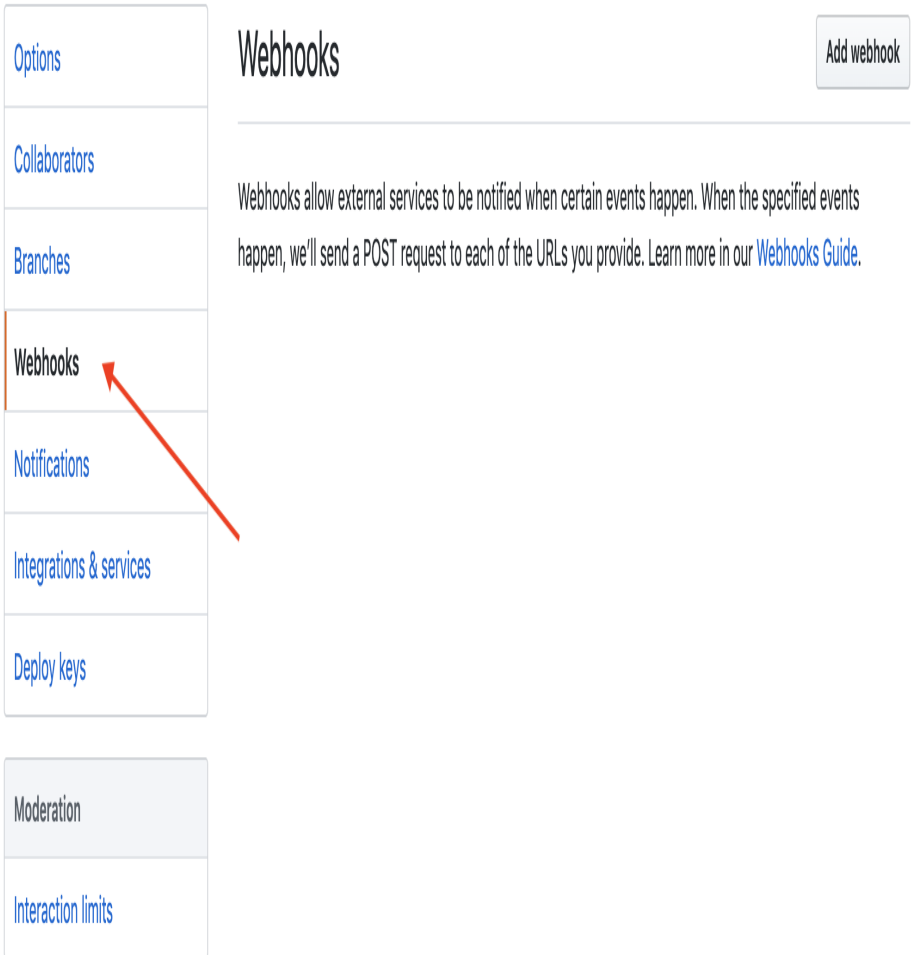


Figure 3.10: Webhooks page

4. Click **Add webhook** to add a new webhook. You may be asked to reenter your GitHub password. After your password is confirmed, the `Add Webhook` page displays.

5. Next, you must provide a payload URL to the webhook. The webhook is triggered (sends a HTTP POST request to the URL) when you commit some changes to the Git repository.

   To get the payload URL, switch to the OpenShift web console in your browser, and then click on `Builds` in the navigation pane.

6. Click the `version` build config to bring up the `Build Config Overview` page.

   Scroll to the bottom of this page and locate the `Copy URL with Secret` link next to the `GitHub` type.

   Click on the notepad icon to copy the payload URL.

Figure 3.11: Payload URL

7. Switch back to the GitHub webhooks page.

   Paste the payload URL into the `Payload URL` field.

   Change the `Content type` to `application/json`.

   Do not change any other fields.

   Click **Add webhook** to add the webhook.

   GitHub sends a test ping to the payload URL to verify availability, and displays a green check mark for the webhook, if it was successful.

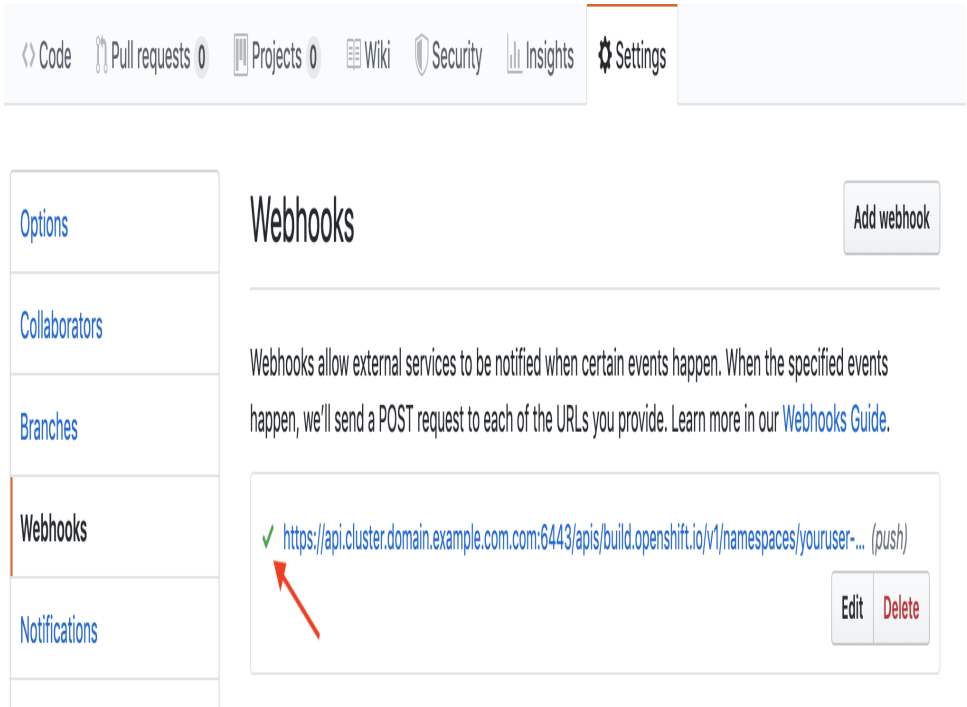


Figure 3.12: Webhook active

3. Push changes to the Git repository to automatically trigger a new build.

    1. Edit the `DO101-apps/version/app.js` file in VS Code to change the `response` variable to version 3, as follows:

    2. Save the changes to your file.

    3. Stage the changes, commit, and then push the changes to your Git repository using steps similar those used for version 2 of the application.

    4. After your changes are committed and pushed to the remote Git repository, a new build is automatically triggered.

        In the OpenShift web console, click on **Builds** → **version**, and then click the `Builds` tab to view the list of builds for the `version` application. Notice that a new build, `version-3` has started.

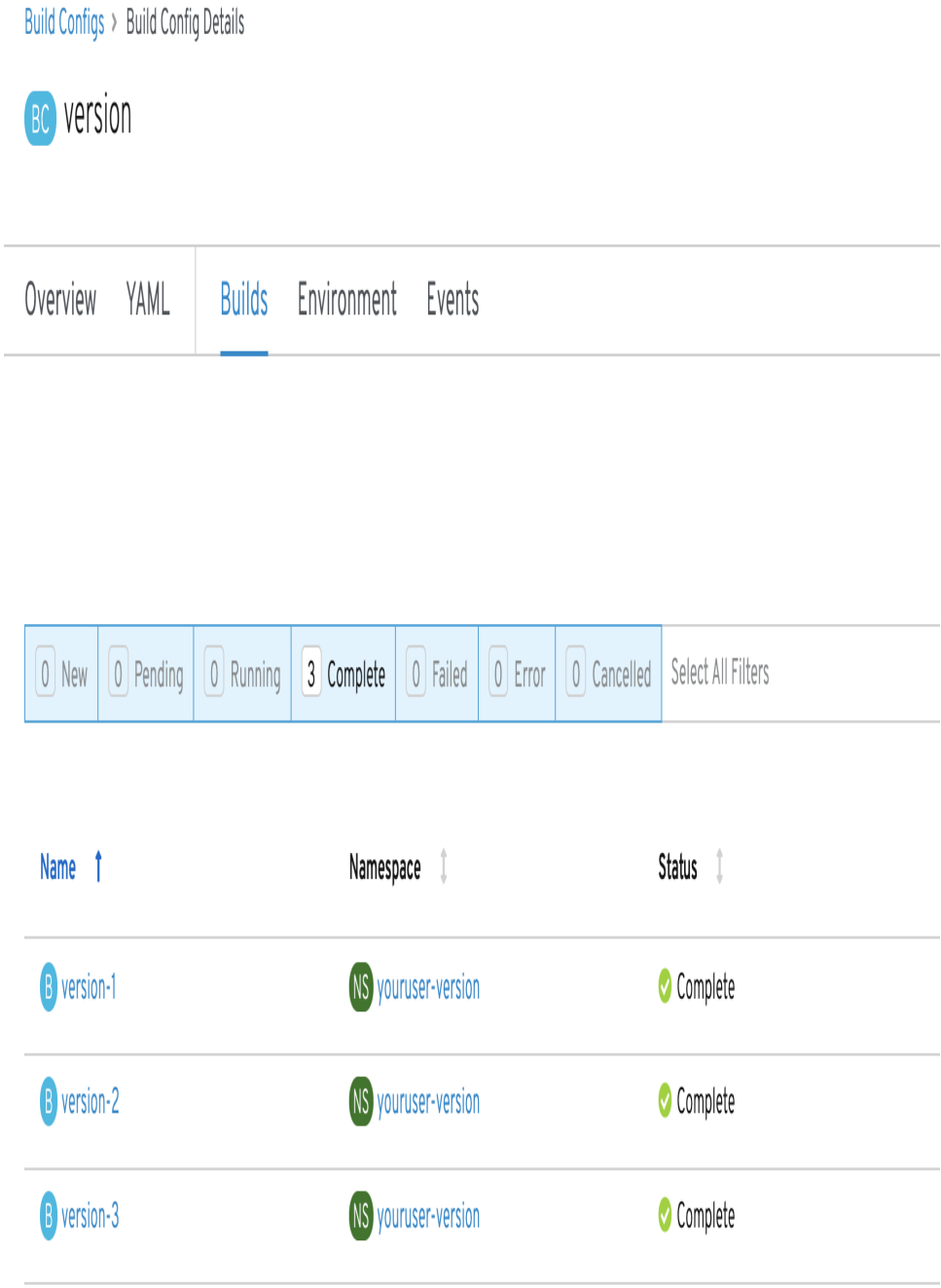


Figure 3.13: Build list

5. Wait for the build to complete successfully, and for the application to be redeployed.

   Click `Topology`, and then click `Open URL` to view the updated application. It should now display version 3 in the output.

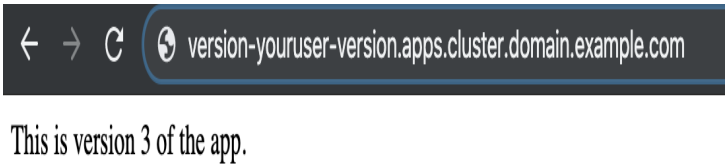


Figure 3.14: Version 3 of app

Close the browser tab.

1. Clean up. Delete the *youruser*-version project and remove the webhooks from GitHub.

    1. Navigate to the Settings page of the D0101-apps repository in GitHub.

       Click on Webhooks to view the webhooks page.

    2. Click **Delete** next to the active webhook, and then confirm the deletion in the resulting confirmation window. You might be prompted for your GitHub password.

    3. In the OpenShift web console, click on **Advanced** → **Projects** to view the list of projects, and then delete the youruser-version project. When prompted, confirm the deletion.

       Enter *youruser*-version in the confirmation window, and then delete the project.

Figure 3.15: Delete version app

This concludes the guided exercise.

Configuring Application Secrets

Objectives

After completing this section, you should be able to add and adjust application configuration and secrets for applications deployed on Red Hat OpenShift Container Platform.

Externalizing Application Configuration in OpenShift

Cloud-native applications store application configuration as environment variables rather than coding the configuration values directly in the application source code. The advantage of this approach is that it creates a separation between the application configuration and the environment in which the application is running. Configuration usually varies from one environment to another, whereas source code does not.

For example, suppose you want to promote an application from a development environment to a production environment, with intermediate stages such as testing and user acceptance. The source code remains the same, but the configuration details specific to each environment, such as connection details to a non-production database, must not be static and must be managed separately.

Configuring Applications Using Secrets and Configuration Maps

Red Hat OpenShift Container Platform provides `Secret` and `Configuration Map` resources to externalize configuration for an application.

Secrets are used to store sensitive information, such as passwords, database credentials, and any other data that should not be stored in clear text.

Configuration maps, also commonly called `config maps`, are used to store non-sensitive application configuration data in clear text.

You can store data in secrets and configuration maps as key-value pairs or you can store an entire file (for example, configuration files) in the secret. Secret data is base64 encoded and stored on disk, while configuration maps are stored in clear text. This provides an added layer of security to secrets to ensure that sensitive data is not stored in plain text that humans can read.

The following is an example configuration map definition in YAML:

```
apiVersion: v1
data:
    username: myuser (1)
    password: mypass (2)
kind: ConfigMap (3)
metadata:
    name: myconf (4)
```

(1) Data stored in the configuration map
(2)
(3) OpenShift resource type (configuration map)
(4) Name for the configuration map

The following is an example secret definition in YAML:

```
apiVersion: v1
data:
    username: cm9vdAo= (1)
    password: c2VjcmV0Cg== (2)
kind: Secret (3)
metadata:
    name: mysecret (4)
    type: Opaque
```

(1) Data stored in the secret in base64 encoded format
(2)
(3) OpenShift resource type (secret)
(4) Name for the secret

Note that the data in the secret (`username` and `password`) is encoded in base64 format, and not stored in plain text like the configuration map data.

After creating secrets and configuration maps, you must associate the resources with applications by referencing them in the deployment configuration for each application.

OpenShift automatically redeploys the application and makes the data available to the application.

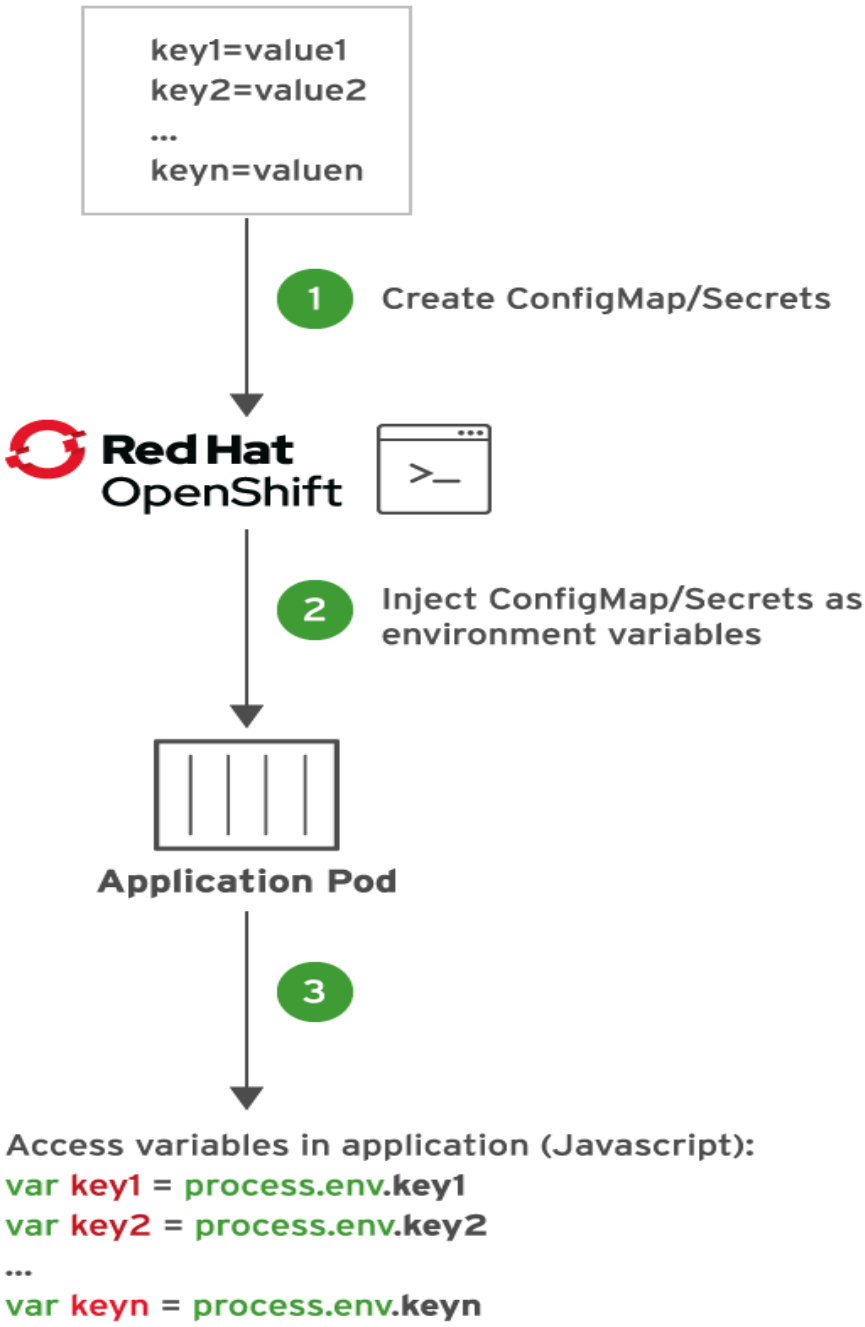


Figure 3.16: Configuration Maps and Secrets

The workflow for creating and using secrets and configuration maps on OpenShift is as follows:

1. Create configuration maps and secrets using the OpenShift web console, or the OpenShift command line client (`oc`). You can store key-value pairs, or an entire file.

2. After you edit the deployment configuration for the application, and map the environment variables to use the secrets and configuration maps configured in the project, OpenShift injects the secrets and configuration maps into application pods .

3. The application accesses the values at run time, using key based look-ups. OpenShift converts the base64 encoded data into a format that the application can read.

In addition to exposing secrets and configuration maps as environment variables, you can also expose them as files. This is useful when you store entire configuration files in secrets or configuration maps.

For example, if your application configuration is in an XML file, then you can store the file in a configuration map or secret and mount it inside the container, where your application can then parse it at start-up. To make changes to the configuration, you edit the configuration map or secret, and then OpenShift automatically redeploys the application container and picks up the changes. You do not need to rebuild your application or the container image.

Note that if you choose to mount the secrets and configuration maps as files inside the application pods, OpenShift mounts the file using a read-only temporary file system, with a file for each key, and the key value as the content of the corresponding file.

Guided Exercise: Configuring Application Secrets

In this exercise, you will deploy a Node.js application that uses configuration maps and secrets on Red Hat OpenShift Container Platform.

Outcomes

You should be able to:

- Create configuration maps and secrets using the OpenShift web console.
- Update the deployment configuration for an application to use the configuration maps and secrets.
- Deploy an application that uses the configuration maps and secrets.

To perform this exercise, ensure you have access to:

- A running Red Hat OpenShift cluster.
- The source code for the `weather` application in the `DO101-apps` Git repository on your local system.

Procedure 3.2. Steps

1. In this exercise, you will use the OpenWeatherMap API to fetch the weather forecast for cities around the world. To invoke the OpenWeatherMap API, you need a unique API key.
    1. Create a new account for the OpenWeatherMap API. Navigate to the website https://openweathermap.org/ using a web browser.
    2. Click `Sign up` to create a new account.

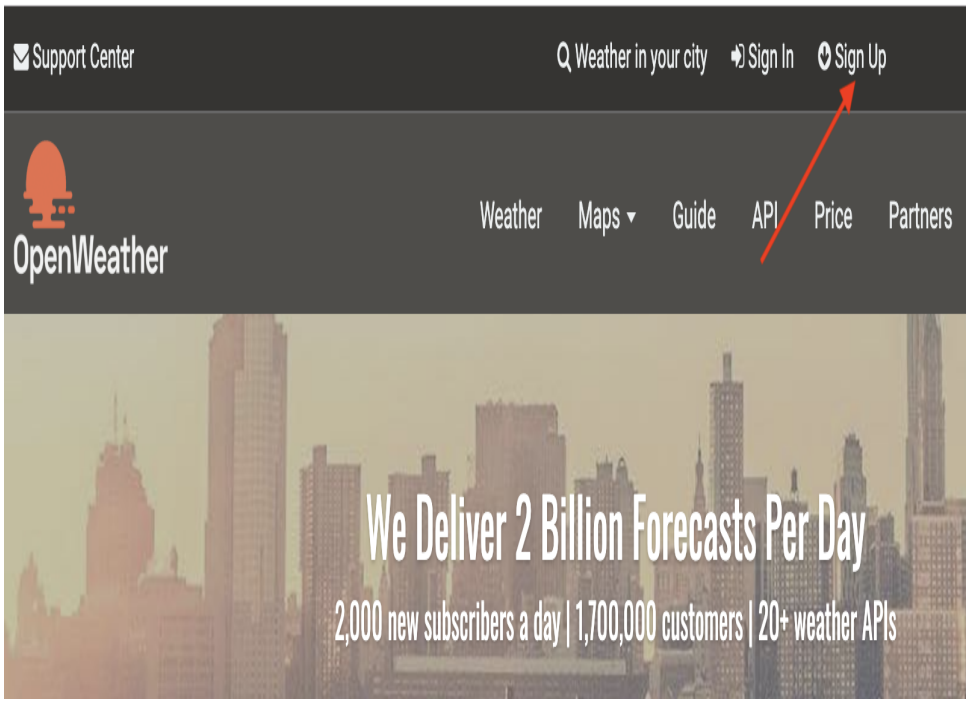


Figure 3.17: Sign up for OpenWeatherMap account

3. Enter a `username`, `email`, and `password` for your account. Select the check boxes to confirm your age, and agree to the terms and conditions.

   Do not select any of the three options related to receiving communications from OpenWeather. Select the `I'm not a robot` option, and then click **Create Account**.

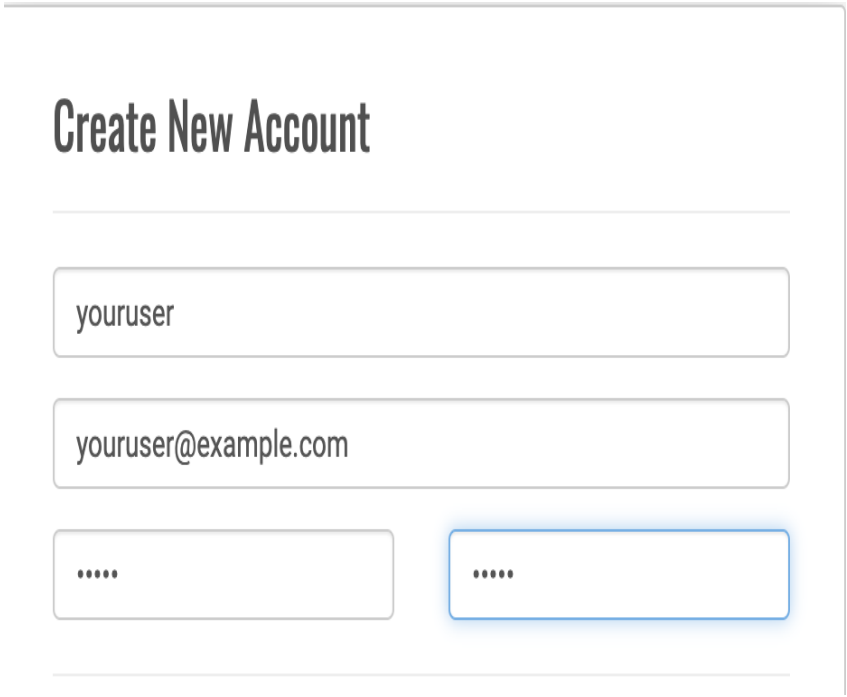


Figure 3.18: Create a new OpenWeatherMap account

4. When asked to provide details about the usage of the API, enter your name and select `Other` in the `Purpose` field.

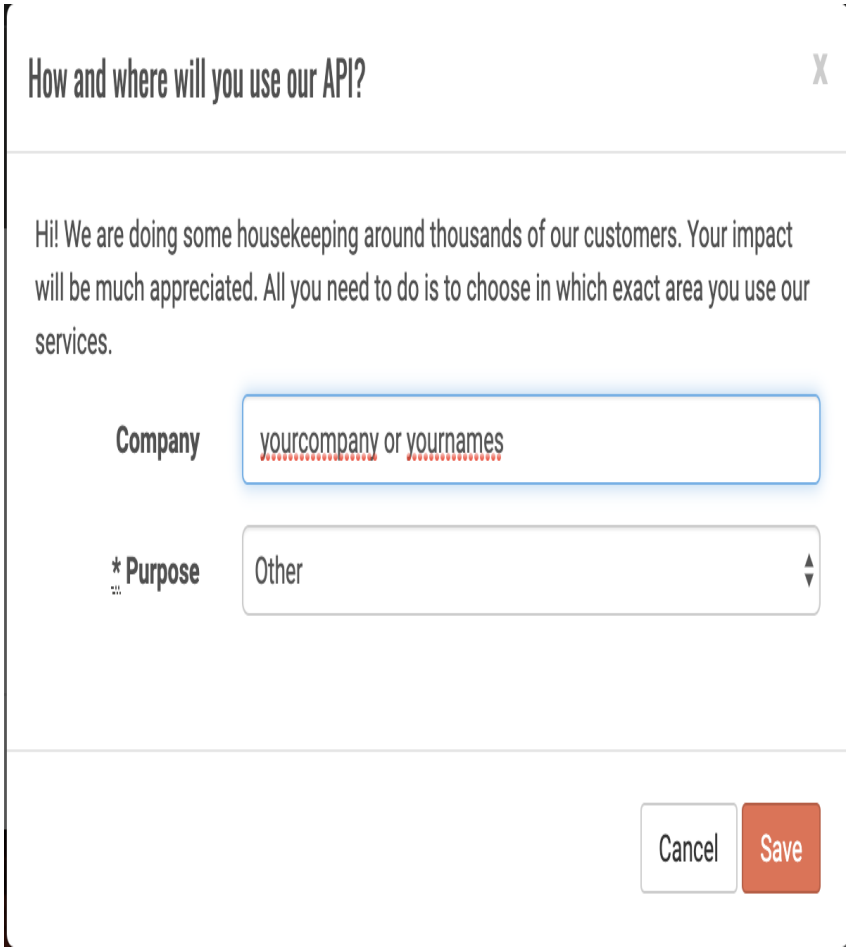


Figure 3.19: API Usage Dialog

5. After you have logged in, click `Services` to view the services offered for your free account. There are restrictions on free accounts that limit the number of API calls you can make in a given duration.

Figure 3.20: API Services

6. Click `API keys` to view the API keys for your account.

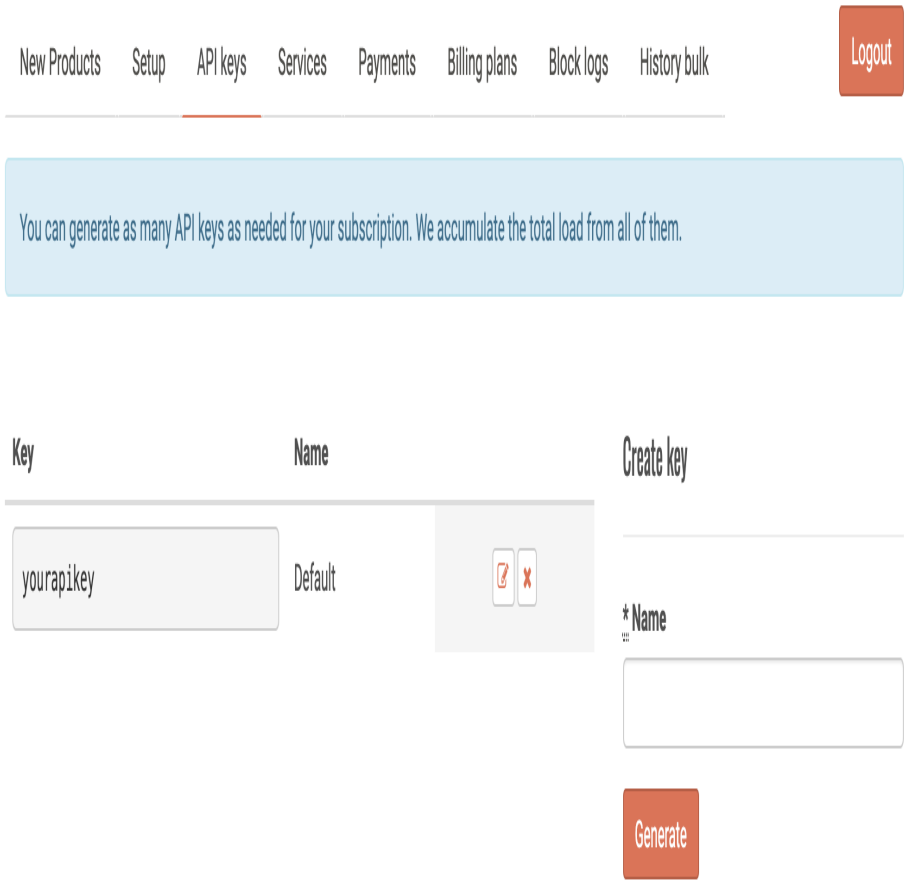


Figure 3.21: API keys

7. A default API key is generated for your account. Copy this API key to a temporary file, or keep the browser tab open. You need the API key to create an OpenShift secret for the `weather` application. You can also generate more keys by clicking **Generate**.

   It may take a few minutes for your API key to be activated before you can invoke the OpenWeatherMap API. To verify that your API key is active, invoke the URL `http://api.openweathermap.org/data/2.5/weather?q=London&appid=api_key` using a browser.

   Replace `api_key` with your default API key from the previous step. If the API key is activated, you will get a JSON response like the following with forecast data for London:

   If the key is not yet activated, you will get a code 401 `Invalid API key` message. Continue with the next steps in the exercise while the API key gets activated, and retest it after the application is deployed in OpenShift.

2. Inspect the source code for the `weather` application.

   1. Launch the Visual Studio Code (VS Code) editor, and then open the `DO101-apps` folder in the `My Projects` workspace. The source code for the `weather` application is in the `weather` directory.

   2. Inspect the `DO101-apps/weather/package.json` file to view the package dependencies for this Node.js application. The `weather` application is a simple web application that is based on the popular Express.js framework. The `weather` application uses the `node-fetch` HTTP client package to access the OpenWeatherMap API and display weather forecasts for numerous cities around the world.

      ```
      "dependencies": {
          "cookie-parser": "~1.4.4",
          "debug": "~2.6.9",
          "dotenv": "^8.1.0",
          "express": "~4.16.1",
          "http-errors": "~1.6.3",
      ```

```
    "morgan": "~1.9.1",
    "node-fetch": "^2.6.0",
    "package.json": "^2.0.1",
    "pug": "2.0.0-beta11"
  }
```

3. Inspect the `DO101-apps/weather/app.js` file, which is the main entry point for the application. There is a single Express.js route definition called `indexRouter`:

4. The code for the `indexRouter` route is defined in the `DO101-apps/weather/routes/index.js` file. Open this file in VS Code. This file contains the main business logic in the application.

5. The first method handles HTTP GET requests to the '/' URL:

   All HTTP GET requests to the '/' URL are redirected to a page with an HTML form that allows you to enter a city name for which you want the weather forecast.

   The code for the HTML form is in the `DO101-apps/weather/views/index.pug` file.

6. The second method handles HTTP POST requests from the HTML form by invoking the OpenWeatherMap API, and then passes the resulting JSON response to the HTML front end:

7. Note that invoking the OpenWeatherMap API requires an API key. The API key is injected as an environment variable at run time. You will create an OpenShift secret to store the API key:

8. You will also create a configuration map to store application specific configuration in plain text. The `UNITS` environment variable controls if the weather forecast is displayed in metric (degree celsius), or imperial units (fahrenheit):

3. Create a new branch in your Git repository for the `weather` application.

   1. In the source control view in VS Code (**View → SCM**), ensure that the `DO101-apps` entry under `SOURCE CONTROL PROVIDERS` shows the `master` branch. If you were working with another branch for a different exercise, then switch to the `master` branch by clicking on the current branch and then selecting `master` in the `Select a ref to checkout` window. If you were working with another branch for a different exercise, then click on the current branch and select `master` in the `Select a ref to checkout` window to switch to the `master` branch.

      WARNING

      Each exercise uses a unique branch. Always create a new branch using `master` as the base.

   2. Click **master** in the `DO101-apps` entry under `SOURCE CONTROL PROVIDERS`.

      Select `Create new branch…` from the list of options.

   3. At the prompt, enter `weather` for the branch name. The Source Control view updates the `DO101-apps` entry with the new branch name.

   4. Push the `weather` branch to your `DO101-apps` GitHub repository.

      Click the cloud icon next to the `weather` branch to push your local branch to your remote Git repository. When prompted, provide your GitHub user name and password.

4. Create a new project for the `weather` application.

   1. Log in to the OpenShift web console using your developer account. Select the `Developer` perspective.

   2. Create a new project named `youruser-weather`. Replace `youruser` with your user name.

5. Create a secret to store the API key for the OpenWeatherMap service.

   1. Click **Advanced → Search** in the navigation pane.

   2. Click on `Service` to expand the menu and search for `secret`. Select the `Secret` option.

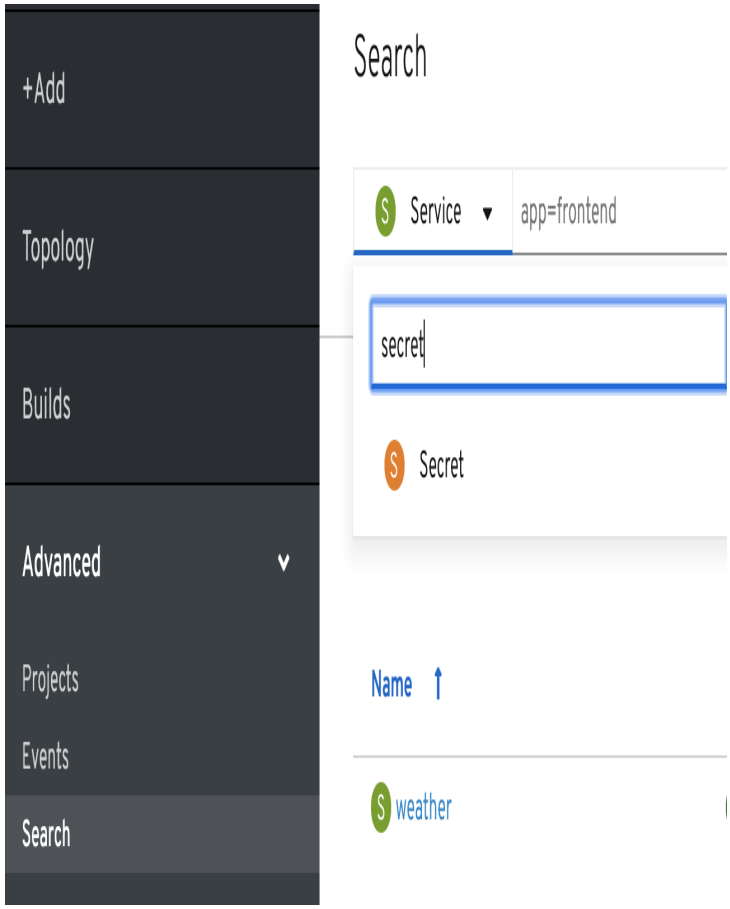


Figure 3.22: Search for Secrets

Ensure that your current project is *youruser*-weather.

3. Click **Create** → **Key/Value Secret** to create a new secret.

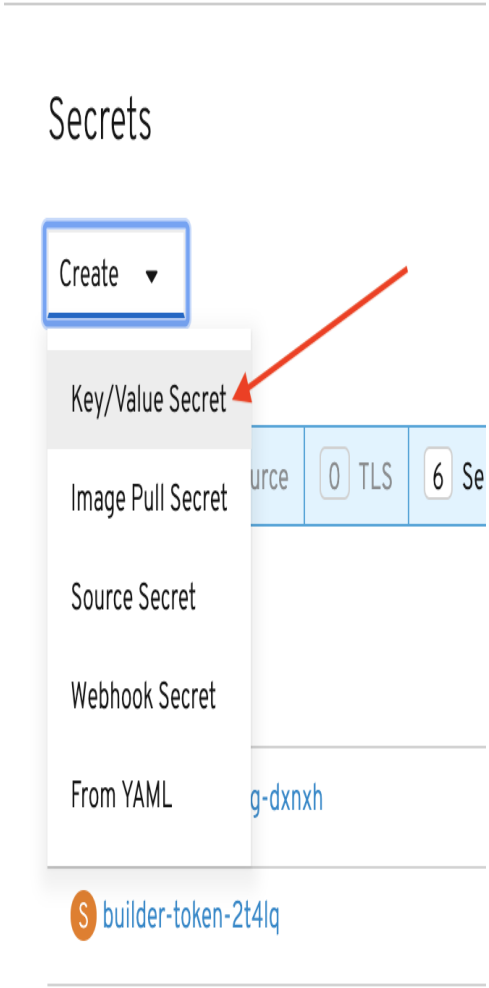


Figure 3.23: Create Secret

4. On the `Create Key/Value Secret` page, enter `owm-api-secret` in the `Secret Name` field, `OWM_API_KEY` in the `Key` field, and then copy the default API key that was generated for your OpenWeatherMap API account to the `Value` field.

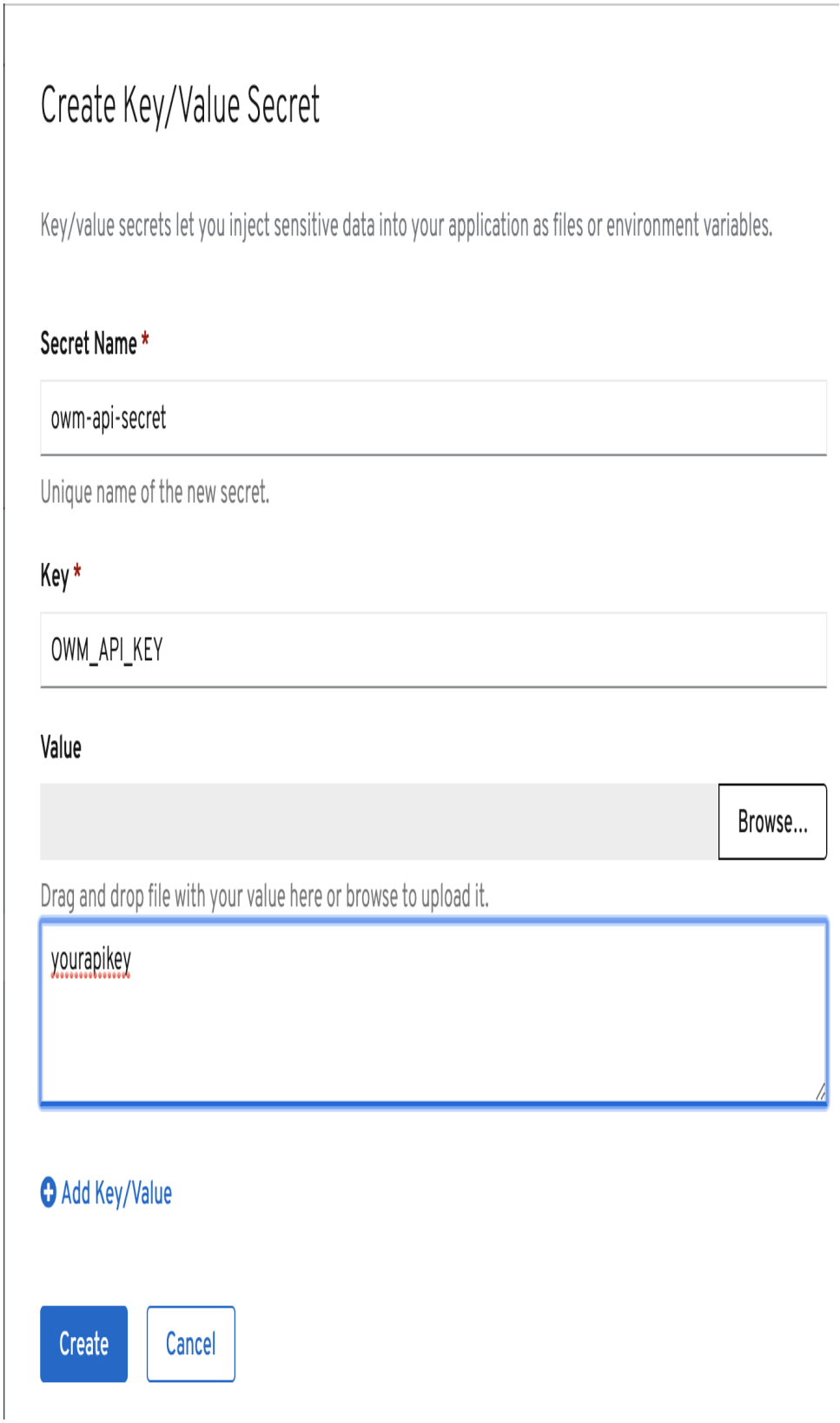


Figure 3.24: Secret Details

Click **Create** to create the secret.

6. Create a configuration map to store the configuration related to the `weather` application.

1. For the `weather` application, create a configuration map to hold a variable that indicates if the weather forecast should be in imperial (Fahrenheit) or metric units (Celsius).

   Click **Advanced** → **Search**.

2. Click on `Service` to expand the menu and search for `configmap`. Select the `ConfigMap` option.

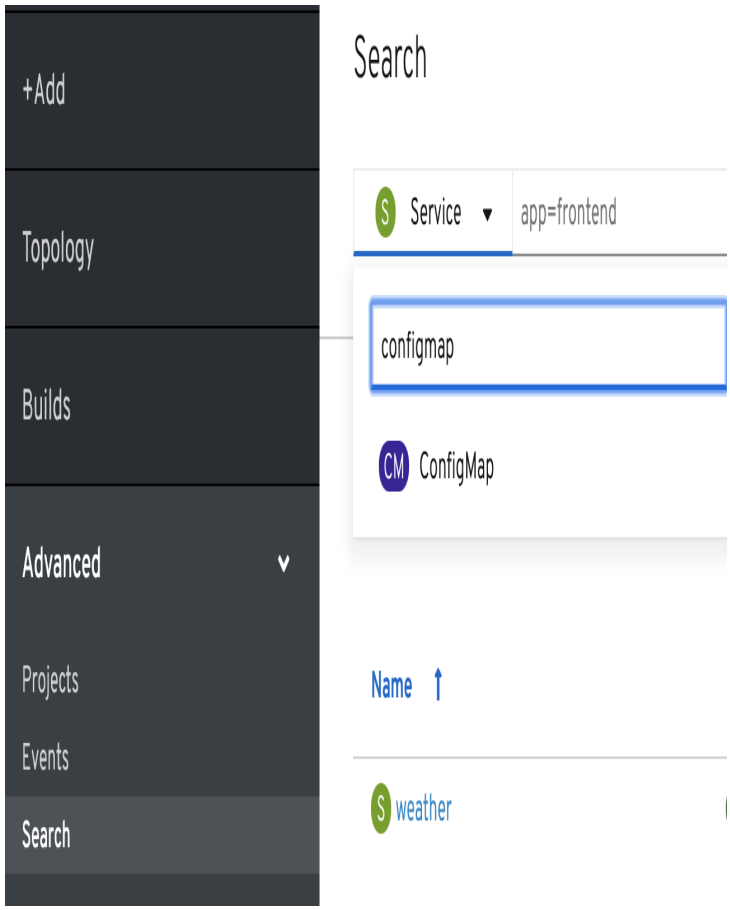


Figure 3.25: Search for ConfigMap

3. Click **Create Config Map** to create a new configuration map.

   The `Create Config Map` page shows a YAML editor and sample code to create key value pairs.

4. Delete the existing YAML content. Add the following YAML to the editable area. Replace *`youruser`* with your user name to ensure that the `namespace` attribute matches your current project:

   IMPORTANT

   Ensure that you maintain correct indentation as shown in the snippet. YAML files are indentation sensitive.

   Click **Create** to create the configuration map.

The snippet creates a configuration map named `weather-config` in the `youruser-weather` project. The configuration map stores a single variable (key) named `UNITS` with a string value `metric`.

6. Deploy the `weather` application to OpenShift.

   1. Select `Add` in the left menu, and then click **From Catalog**.

   2. Select **Languages** → **JavaScript** and click the first option, `Node.js`. Click **Create Application** to enter the details of the application.

   3. Complete the form according to the following table. To access the Git parameters, click **Show Advanced Git Options**.

      Table 3.1. New Application Parameters

Form Field	Value
`Git Repo URL`	https://github.com/`yourgituser`/DO101-apps
`Git Reference`	`weather`
`Context Dir`	`/weather`
`Application Name`	`weather`
`Name`	`weather`

Do not click `Create` yet. You must first customize the deployment configuration.

4. Click `Deployment Configuration` to customize the deployment configuration. Reference the secret and configuration map that you created earlier.

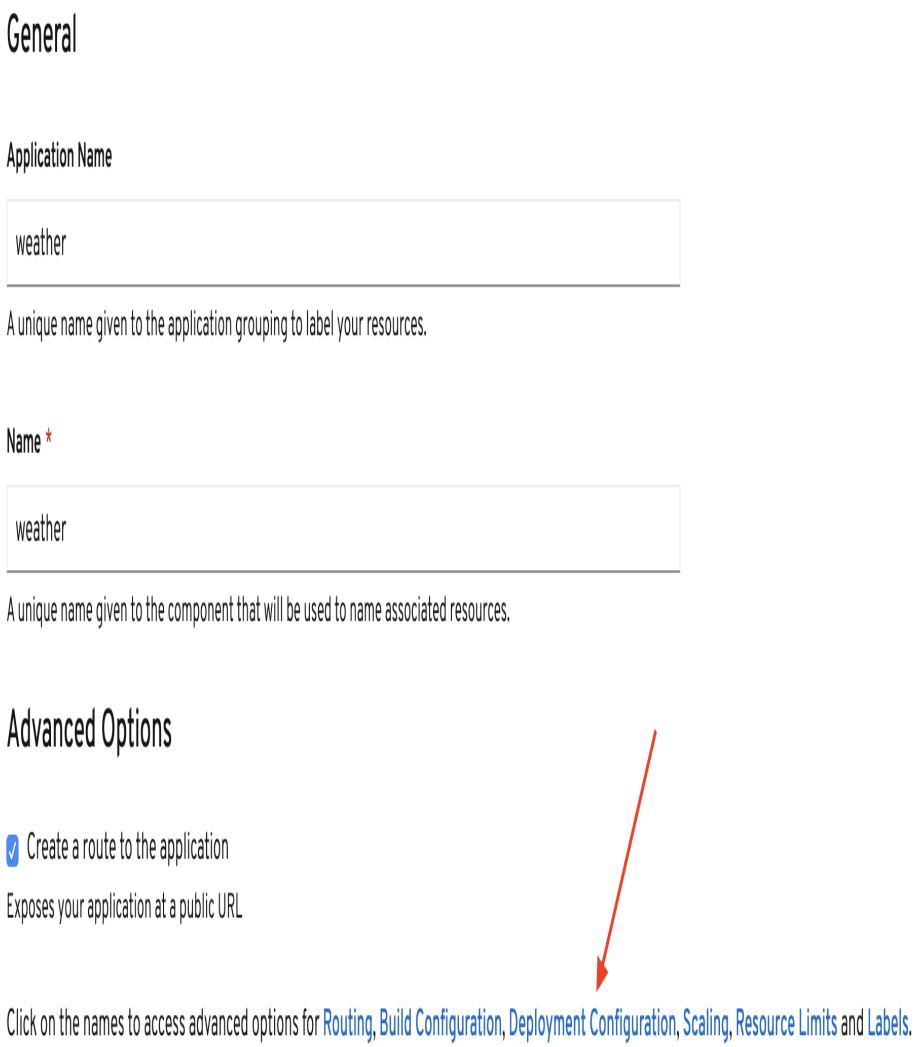


Figure 3.26: Deployment Configuration for weather application

Click **Add from Config Map or Secret**

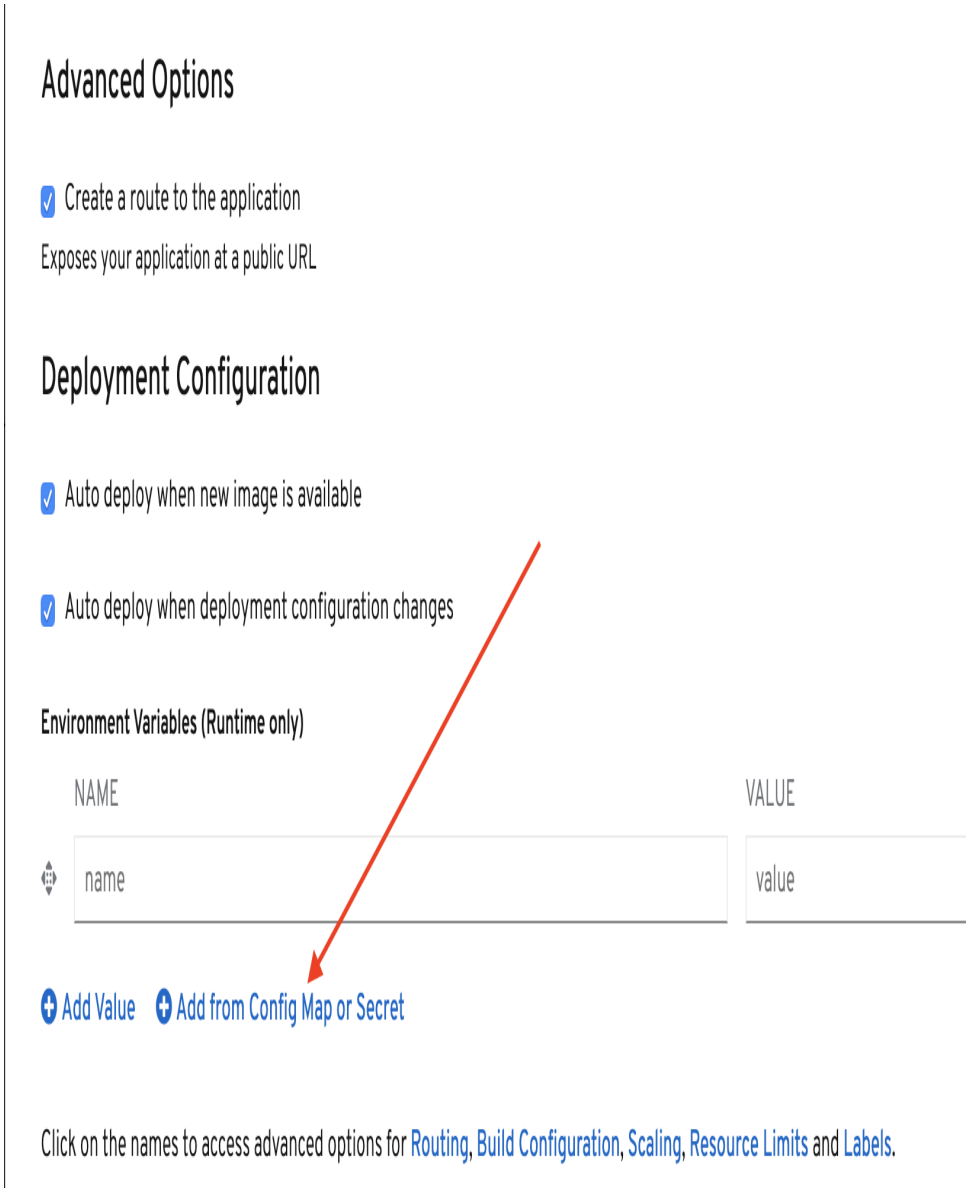


Figure 3.27: Add Environment Variables from Config Map or Secret

5. A new row is added to the `Environment Variables` table. Add a key called `OWM_API_KEY`. Click the `Select a resource` list, search for `owm-api-secret`, and then select the `owm-api-secret` secret.

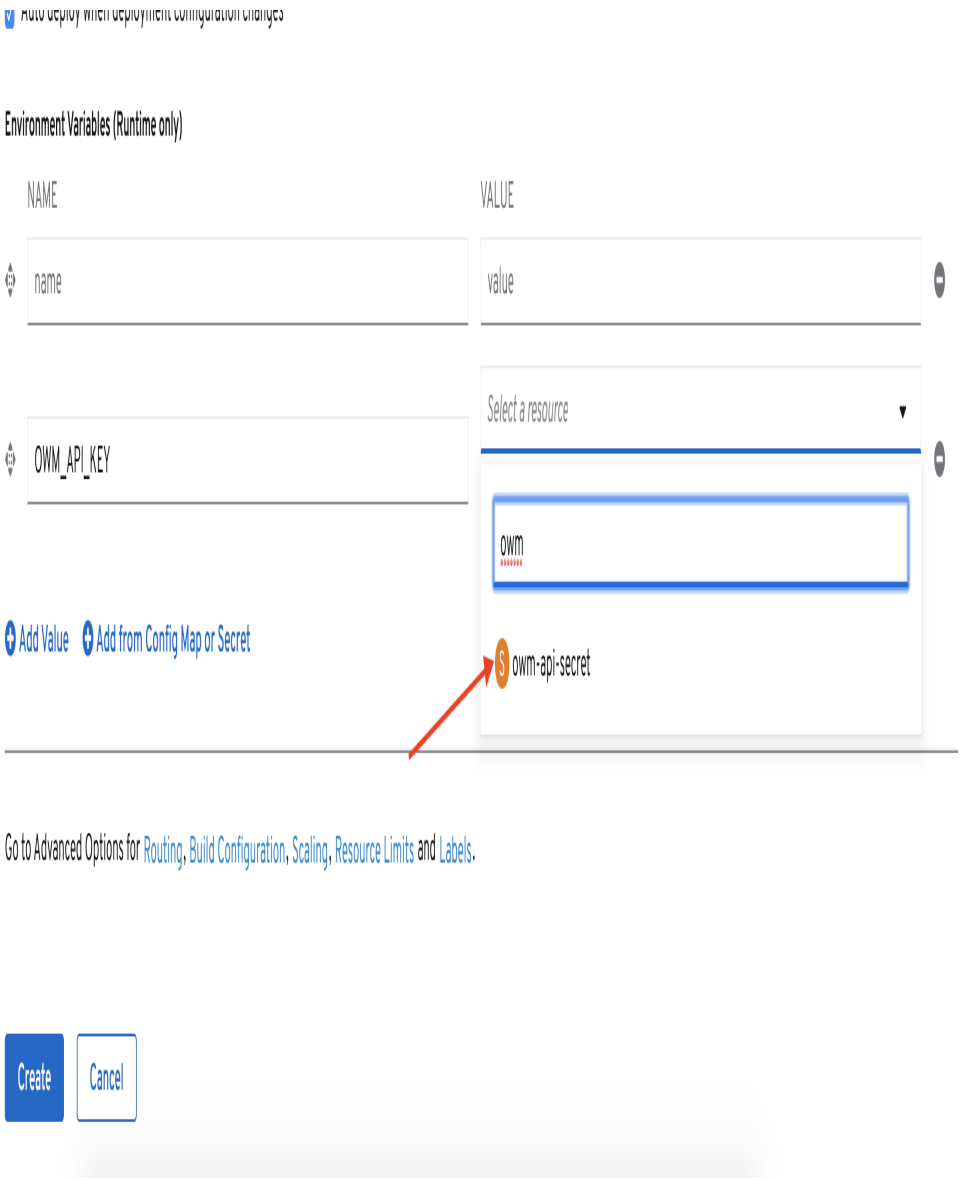


Figure 3.28: Select Secret

Click `Select a key`, and then select `OWM_API_KEY`.

6. Click **Add from Config Map or Secret** again. A third row is added to the `Environment Variables` table.

   Add a key called `UNITS`. Click `Select a resource`, search for `weather`, and then select the `weather-config` configuration map. Click `Select a key`, and the select `UNITS`.

7. Remove the first row in the `Environment Variables` table by clicking the minus icon next to the empty `value` field.

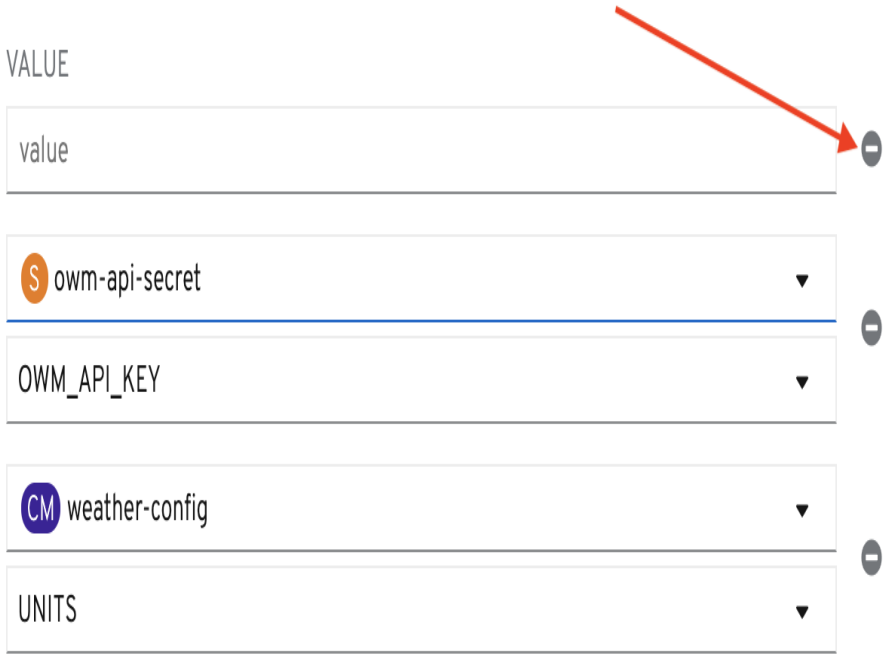


Figure 3.29: Remove empty environment variable

Your final `Environment Variables` table should display as follows:

Figure 3.30: Final Environment Variables table

8. To avoid deployment errors in the following steps, review the values you entered in the form before proceeding. Click **Create** to start the build and deployment processes.

7. Test the application.

    1. Wait for a few minutes while the application container image is built and deployed. When deployment is complete, a green tick mark displays for the `weather` deployment on the `Topology` page.

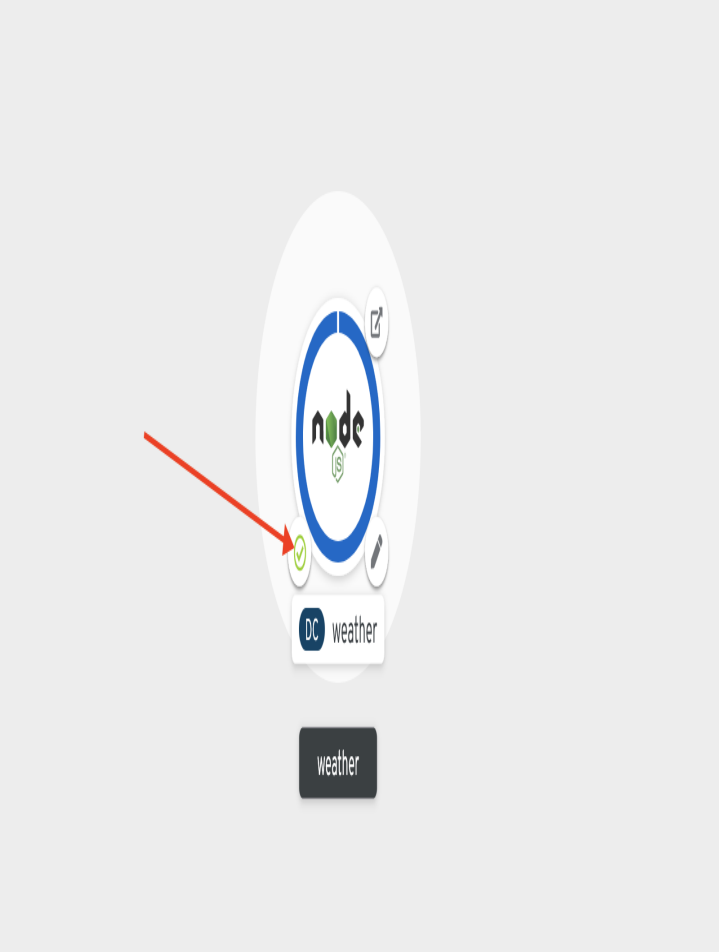


Figure 3.31: Weather app built successfully

2. Click the `Open URL` link to open the route URL for the `weather` application.

Figure 3.32: Weather route URL

3. The home page for the `weather` application displays.

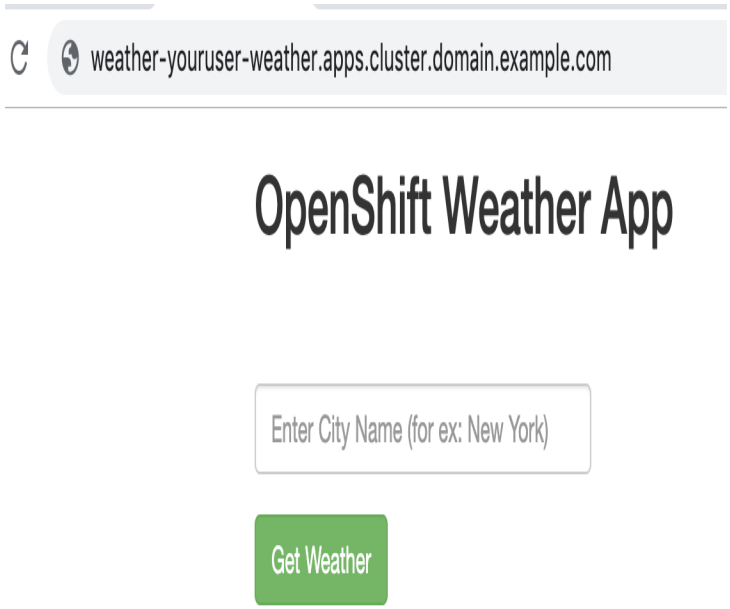


Figure 3.33: Weather app home page

4. Enter a city name in the field, such as "New York", and then click **Get Weather**.

5. The current weather forecast for the city displays in metric units.

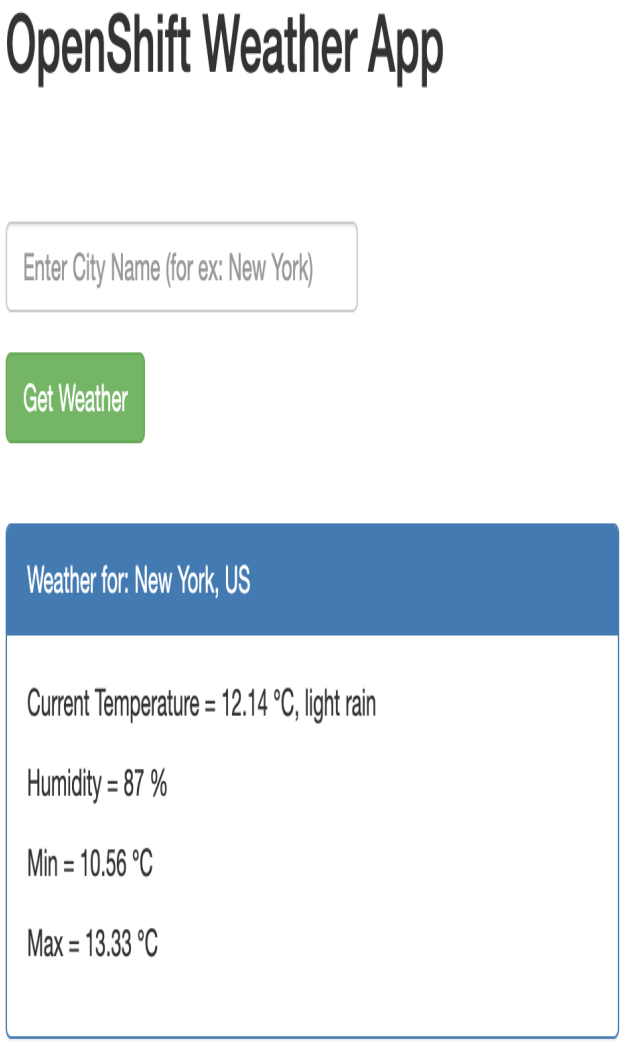


Figure 3.34: Weather forecast

Close the browser tab.

8. Switch back to the OpenShift web console to view the logs for the `weather` application.
    1. Click `Topology`, and then click on the `weather` deployment.
    2. Click `Resources`, and then click `View Logs` for the `weather` pod to view the logs for the `weather` application.

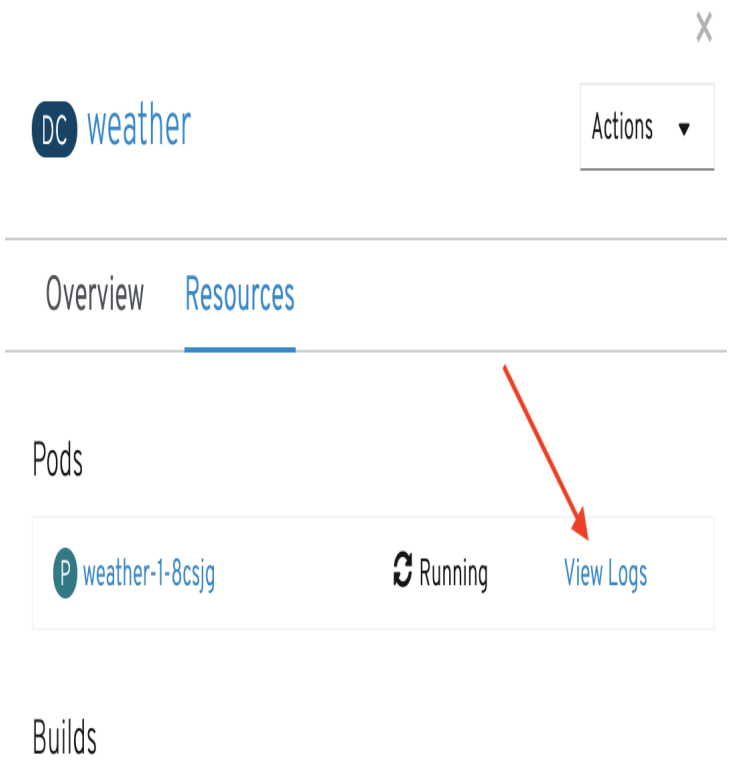


Figure 3.35: View logs for the weather application

The raw JSON response from the OpenWeatherMap API should be displayed. The application user interface filters the response and displays a subset of the data on the screen.

```
{ coord: { lon: -73.99, lat: 40.73 },
  weather:
   [ { id: 500, main: 'Rain', description: 'light rain', icon: '10d' },
     { id: 701, main: 'Mist', description: 'mist', icon: '50d' },
     { id: 300,
       main: 'Drizzle',
       description: 'light intensity drizzle',
       icon: '09d' } ],
  base: 'stations',
  main:
   { temp: 12.14,
     pressure: 1020,
     humidity: 87,
     temp_min: 10.56,
     temp_max: 13.33 },
  visibility: 16093,
  wind: { speed: 6.2, deg: 70 },
```


Figure 3.36: Weather application logs

9. Update the deployment configuration for the `weather` application to change the value of the `UNITS` key in the `weather-config` configuration map to display the forecast in imperial units.

    1. Click **Advanced** → **Search**. Click on `Service` to expand the menu, search for `configmap`, and then select `ConfigMap`.

    2. Click the three dots to the right of `weather-config`, and then click `Edit Config Map`.

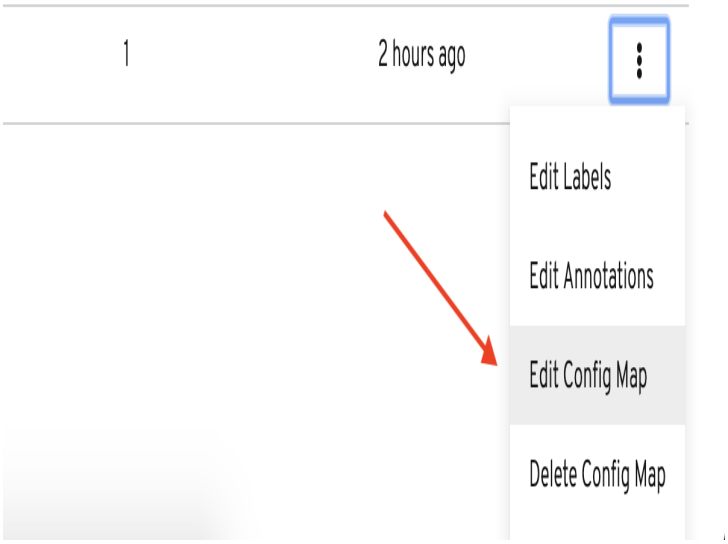


Figure 3.37: Edit Config Map

    3. A page displays with an editable YAML snippet that contains the existing data in the `weather-config` configuration map.

Change the value of the `UNITS` key to `imperial` as shown below:

4. Click **Save** to save your changes.

1. Retest the application and verify that the weather forecast is displayed in imperial units.

    1. Click `Topology`, and then click the `weather` deployment.

    Click on **Actions** → **Start Rollout**. It may take a minute to redeploy the application.

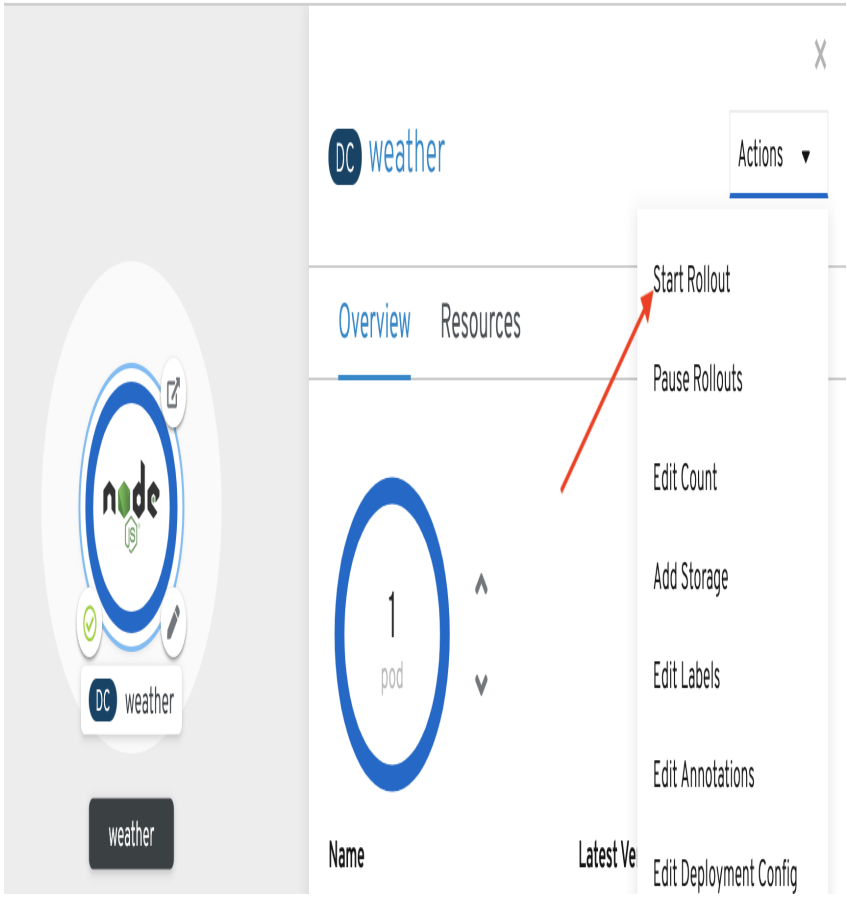


Figure 3.38: Rollout Deployment

2. Click the `Open URL` icon for the `weather` deployment on the `Topology` page, represented by a blue circle diagram with the Node.js logo.

Figure 3.39: Weather route URL

3. Enter a city name, and then click `Get Weather`. The weather forecast displays in imperial units.

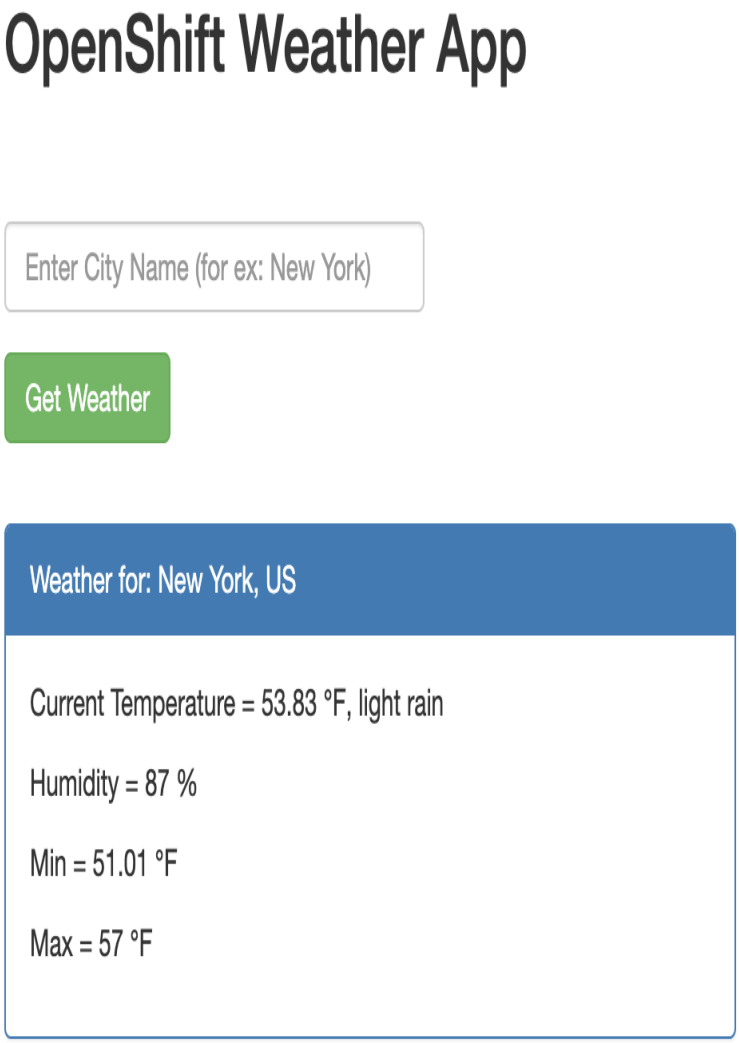


Figure 3.40: Weather forecast in imperial units

Close the browser tab.

2. Clean up. Delete the `youruser-weather` project.

   In the OpenShift web console, click **Advanced** → **Projects** to view the list of projects, and then delete the *`youruser`*`-weather` project. Confirm the deletion when prompted.

   Enter `youruser-weather` in the confirmation window and delete the project.

This concludes the guided exercise.

Connecting an Application to a Database

Objectives

After completing this section, you should be able to deploy an application that connects to a database on the Red Hat OpenShift Container Platform.

Connecting to Databases

The Red Hat OpenShift Container Platform supports the deployment of a number of databases, such as MySQL, PostgreSQL, and MongoDB, using the OpenShift web console
`Developer` view, or the OpenShift command line client (`oc`).

After deploying a database, you can deploy other applications to OpenShift to access, store, and manage data in the database. Store the database credentials in an OpenShift secret, and then connect to the database from applications using environment variables. OpenShift injects the secret data, as environment variables, into application pods at run time.

NOTE

When deploying a database using one of the built-in templates provided by OpenShift, a secret containing the database user name, password, and database name is created automatically. The name of the secret is the same as the database service name.

Although you can use of this secret to connect to the database, you might want to create your own secret to store more details about the database, including application specific flags.

You can create a single secret that encapsulate all the configuration details for the database. You can safely delete the default, generated secret if you do not need it.

By externalizing the database configuration and storing it in a secret, you avoid storing sensitive information in plain text configuration files. Another advantage of this approach is support for switching between different environments, like development, staging, QA, and production, without rebuilding the application.

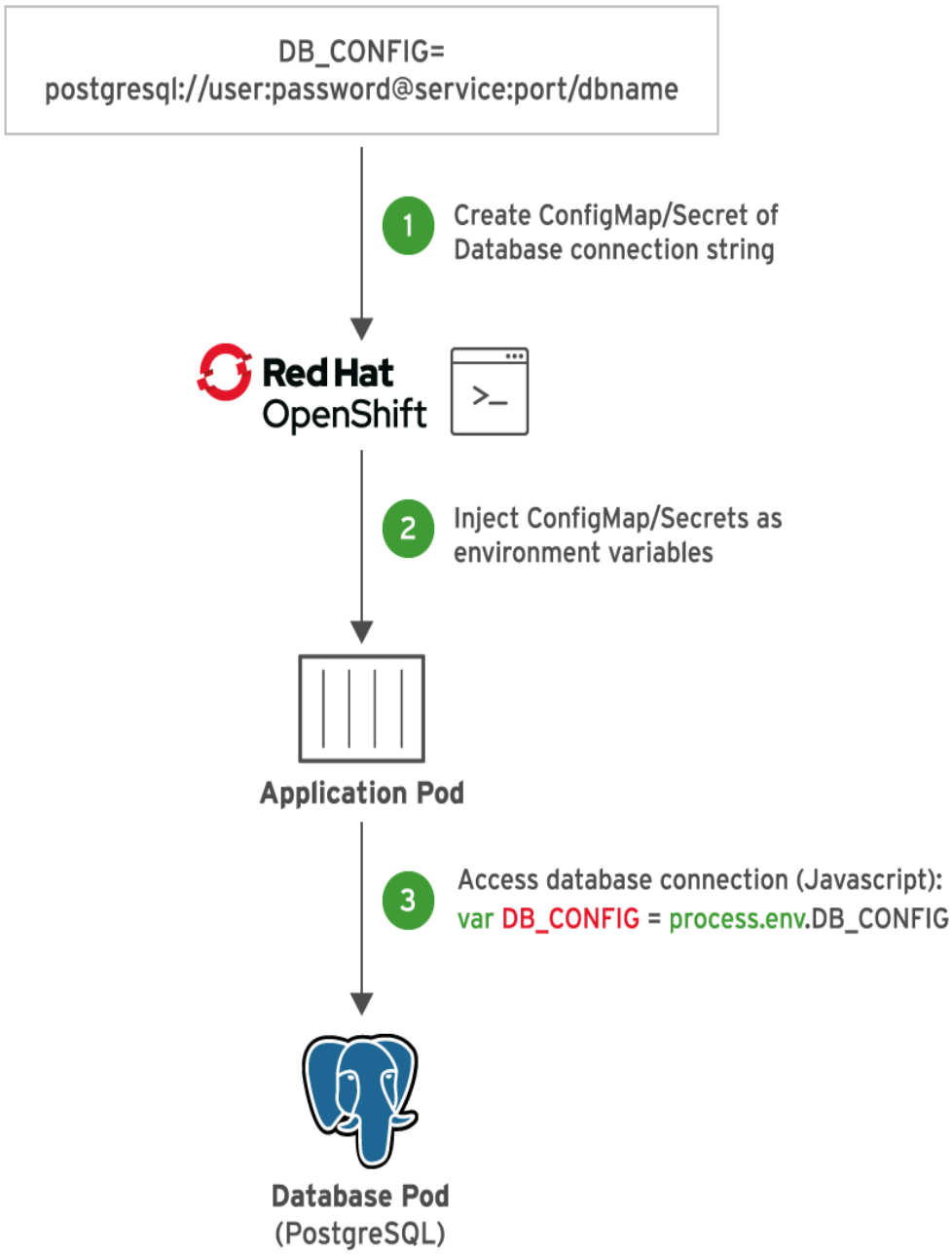


Figure 3.41: Connecting to a PostgreSQL Database

The workflow for accessing databases from applications deployed on OpenShift is as follows:

1. Create a secret to store the database access configuration using the OpenShift web console, or the OpenShift command line client (`oc`).
2. OpenShift injects the secret into application pods after you edit the deployment configuration for the application, and map the environment variables to use the secret.
3. The application accesses the values at run time, using key based look-ups. OpenShift converts the base64 encoded data back into a format that is readable by the application.

For Node.js based JavaScript applications, the general format for the PostgreSQL database connection string is of the form:

```
postgresql://username:password@dbservice:port/dbname
```

where,

- `username` = database user name for accessing the database
- `password` = password for accessing the database
- `dbservice` = service name for the database deployed on OpenShift
- `port` = TCP port where the database server listens for incoming connections. Default is "5432" for PostgreSQL
- `dbname` = database name

For example, deploy a PostgreSQL database on OpenShift as shown in following table:

Table 3.2. PostgreSQL Database Details

Form Field	Value
Database Service Name	mydbservice
PostgreSQL Connection Username	myapp
PostgreSQL Connection Password	mypass
Port	5432
PostgreSQL Database Name	mydb

The resulting database connection string for Node.js based applications is:

```
postgresql://myapp:mypass@mydbservice:5432/mydb
```

In scenarios where the Node.js application is deployed on OpenShift, but the database is external to the cluster, the database connection string remains the same; the one exception is that the database service name is replaced by the hostname or IP address of the external database server.

For example, if your PostgreSQL database server is running on a server called `mydbhost.example.com`, then the database connection string (assuming all other details are similar to values listed in the preceding table) becomes:

```
postgresql://myapp:mypass@mydbhost.example.com:5432/mydb
```

To create a secret with the database connection string as data, and to access it from a Node.js application, use the following steps:

1. Navigate to the **Advanced → Search** in the OpenShift web console.
2. Click on `Service` to expand the menu, and then search for `secret`. Select the `Secret` option.

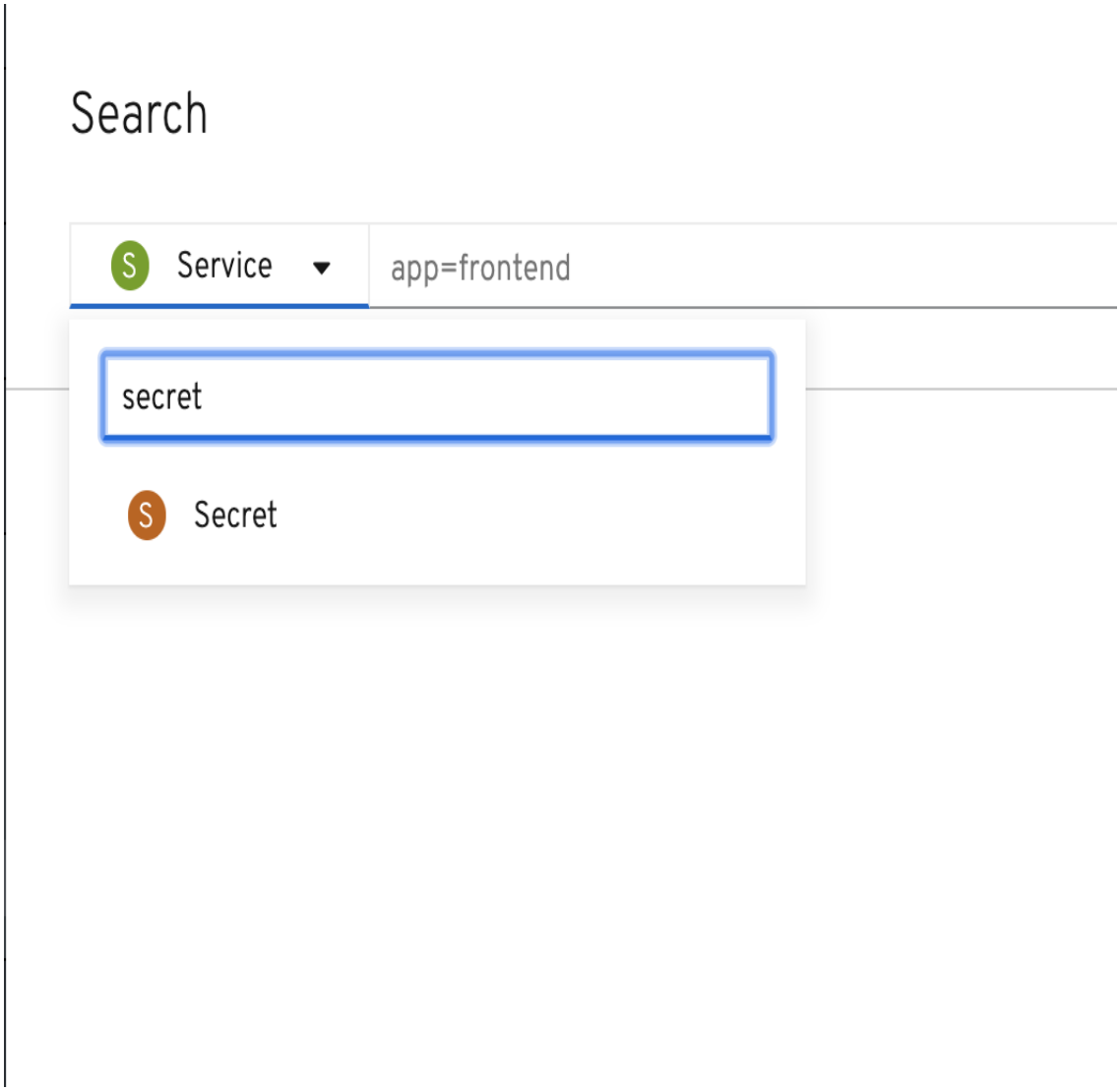


Figure 3.42: Search for Secret type

3. Click **Create** → **Key/Value Secret** to create a new secret.

4. Create a new key-value secret using the database connection string as the value.

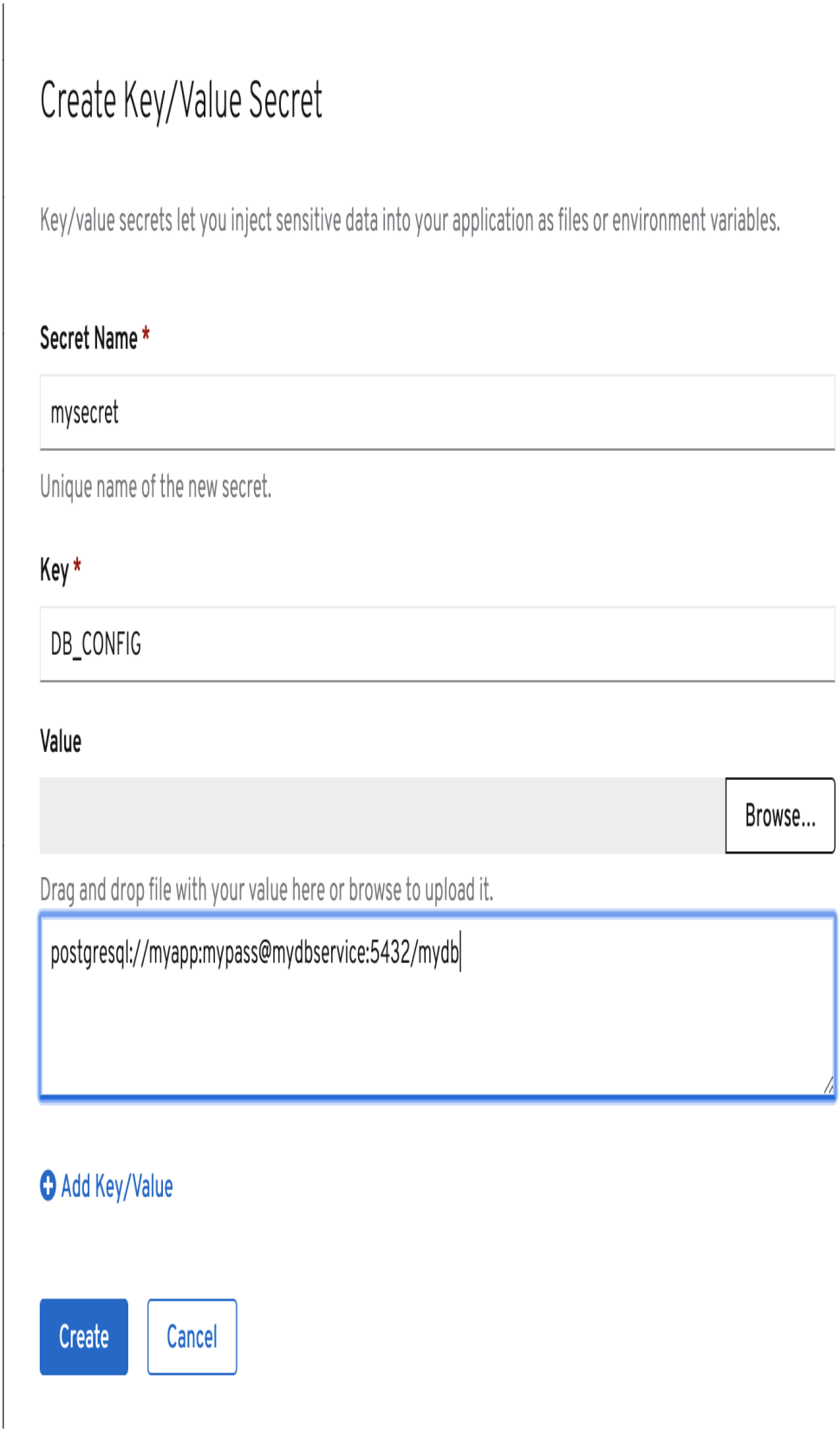


Figure 3.43: Secret Details

5. After creating the secret, edit the deployment configuration for the application and map the secret to an environment variable accessible from the application.

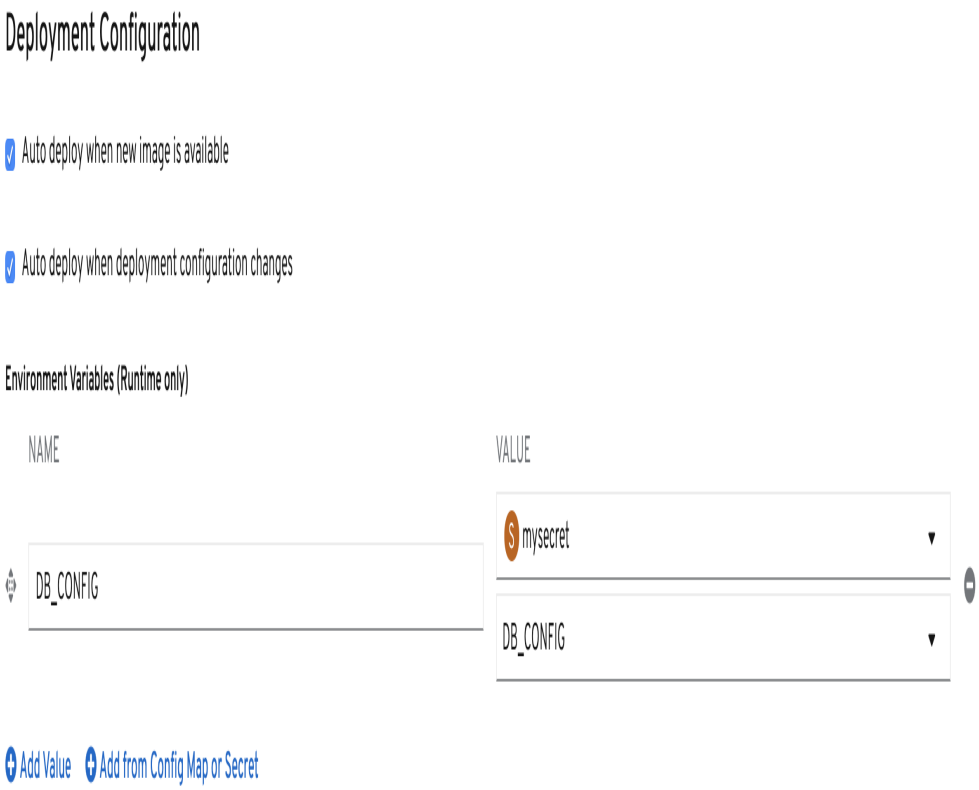


Figure 3.44: Map Secret to Environment Variable

6. Finally, access the environment variable from a Node.js application as follows:

```
...output omitted...
const DB_CONFIG = process.env.DB_CONFIG ...output omitted...
const { Pool } = require('pg');

`const pgconn = new Pool({
    connectionString: DB_CONFIG,
    ssl: false,
});`
...output omitted...
```

Guided Exercise: Connecting to a Database

In this exercise, you will deploy a Node.js application that connects to a PostgreSQL database on OpenShift.

Outcomes

You should be able to:

- Store the database connection information in a secret.
- Integrate the Node.js application with the PostgreSQL database using OpenShift services.
- Populate and fetch data from the PostgreSQL database using the Node.js application.

To perform this exercise, ensure you have access to:

- A running Red Hat OpenShift Container Platform cluster.
- The source code for the `contacts` application in the `D0101-apps` Git repository on your local system.

Procedure 3.3. Steps

1. Inspect the source code for the `contacts` application.

    1. Launch the Visual Studio Code (VS Code) editor, and then open the `DO101-apps` folder in the `My Projects` workspace. The source code for the `contacts` application is in the `contacts` directory.

    2. Inspect the `DO101-apps/contacts/package.json` file to view the package dependencies for this Node.js application. The `contacts` application uses the popular Express.js web application framework, and it stores and fetches contact information from a PostgreSQL database.

    ```
    ...output omitted...
    "dependencies": {
        "connect-flash": "^0.1.1",
        "cookie-parser": "~1.4.4",
        "debug": "~2.6.9",
        "dotenv": "^8.1.0",
        "express": "~4.16.1",
        "http-errors": "~1.6.3",
        "morgan": "~1.9.1",
        "pg": "^7.12.1",
        "pug": "2.0.0-beta11"
      },
    ...output omitted...
    ```

    3. Inspect the `DO101-apps/contacts/app.js` file, which is the main entry point for the application. There is a single Express.js route definition called `indexRouter`:

    ```
    app.use('/', indexRouter);
    ```

    4. The code for the `indexRouter` route is defined in the `DO101-apps/contacts/routes/index.js` file. Open this file in VS Code. The file contains the code to insert and fetch contact information from a PostgreSQL database.

    5. The first method handles HTTP GET requests to the '/' URL. This method checks for the existence of the `contacts` table in the database. If the table does not exist, it renders an HTML page with a button to seed data in the database.

    If the `contacts` table exists, the list of contacts is fetched as a JSON array and passed to the front-end view layer, which displays the contacts in an HTML table.

    The code for the HTML front-end is in the `DO101-apps/contacts/views/index.pug` file.

    ```
    ...output omitted...
    router.get('/', function(req, res) {
    ...output omitted...
      res.render('index', { error: null, contacts: contacts, title: 'Contact List' });
    });
    ...output omitted...
    ```

    6. The second method handles HTTP POST requests to the URL `/seed`. This method creates the `contacts` table, and populates it with a list of contacts. After the data is stored in PostgreSQL, the method redirects the request to the `/` URL, which renders the HTML table with the contacts.

    ```
    ...output omitted...
    router.post('/seed', function(req,res) {
      pgconn.query("drop table if exists contacts; create table contacts ...output omitted...

        // redirect to the index page
        else {
          res.redirect('/');
        }
    ...output omitted...
    ```

    7. The database connection information is configured in the `DO101-apps/contacts/db/config.js` file. The database URL, consisting of the host name, port, database name, user name, and password is injected as an environment variable (`DB_CONFIG`) at run time. You will create an OpenShift secret to store the database URL:

    ```
    ...output omitted...
    const DB_CONFIG = process.env.DB_CONFIG
    ...output omitted...
    ```

```
const { Pool } = require('pg');

const pgconn = new Pool({
    connectionString: DB_CONFIG,
    ssl: false,
});
...output omitted...
```

2. Create a new branch in your Git repository for the `contacts` application.

    1. In the source control view in VS Code (**View** → **SCM**), ensure that the `DO101-apps` entry under `SOURCE CONTROL PROVIDERS` shows the `master` branch. If you are working with another branch for a different exercise, click on the current branch and then select `master` in the `Select a ref to checkout` window to switch to the `master` branch.

        WARNING

        Each exercise uses a unique branch. Always create a new branch using `master` as the base.

    2. Click **master** in the `DO101-apps` entry, for `SOURCE CONTROL PROVIDERS`.

        Select `Create new branch…` from the list of options.

    3. At the prompt, enter `contacts` for the branch name. The Source Control view updates the `DO101-apps` entry with the new branch name.

    4. Push the `contacts` branch to your `DO101-apps` GitHub repository.

        Click the cloud icon for the `contacts` branch to push your local branch to your remote Git repository. When prompted, provide your GitHub user name and password.

3. Create a new project for the `contacts` application.

    1. Log in to the OpenShift web console using your developer account. Select the `Developer` perspective.

    2. Create a new project named `youruser-contacts`. Replace *`youruser`* with your user name.

4. Deploy a PostgreSQL database in the *`youruser`*`-contacts` project.

    1. Click `+Add`, and then ensure that the current project is set to *`youruser`*`-contacts`.

    2. Click **Database**.

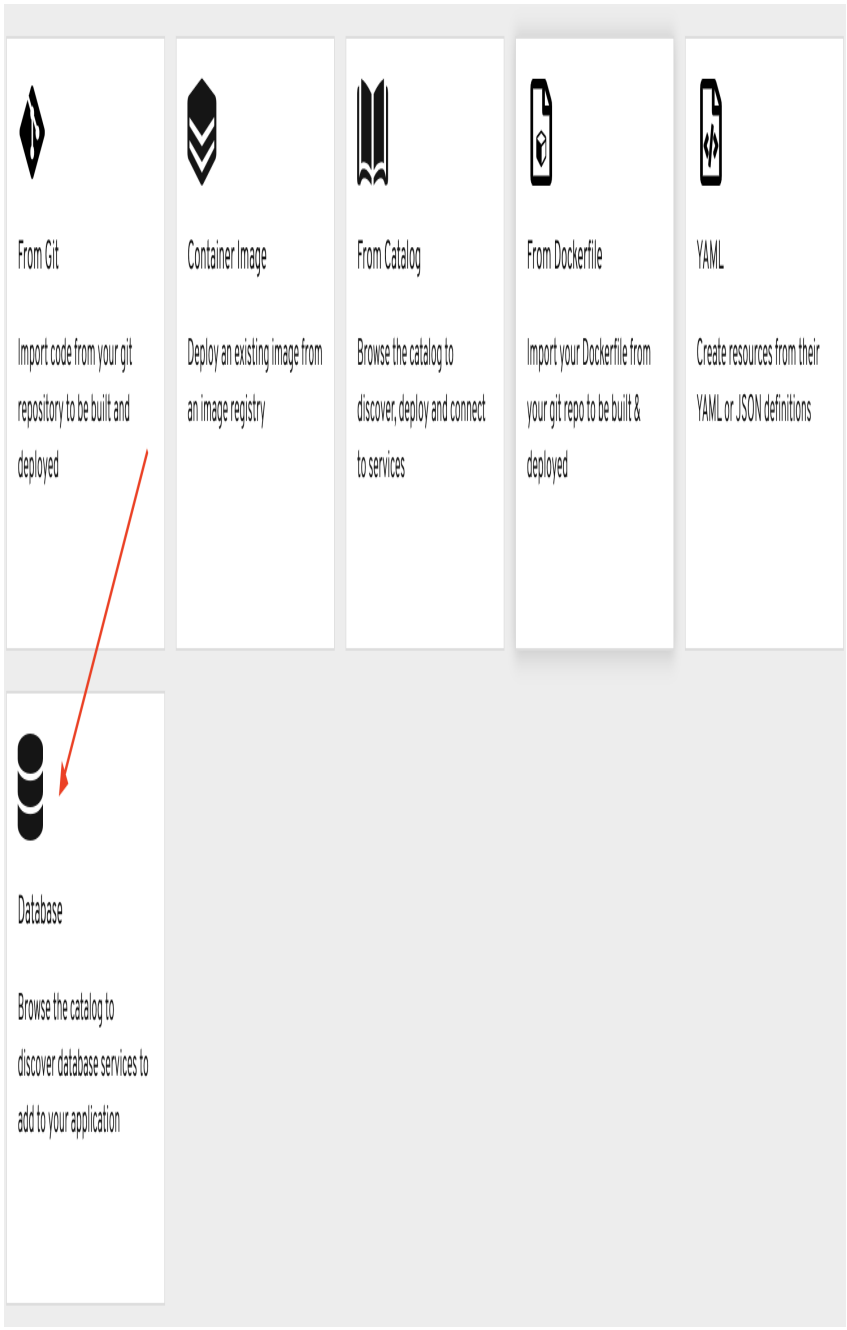


Figure 3.45: Add Database

3. On the `Developer Catalog` page, click **Databases** → **Postgres**.

4. Click the second `PostgreSQL (Ephemeral)` option.

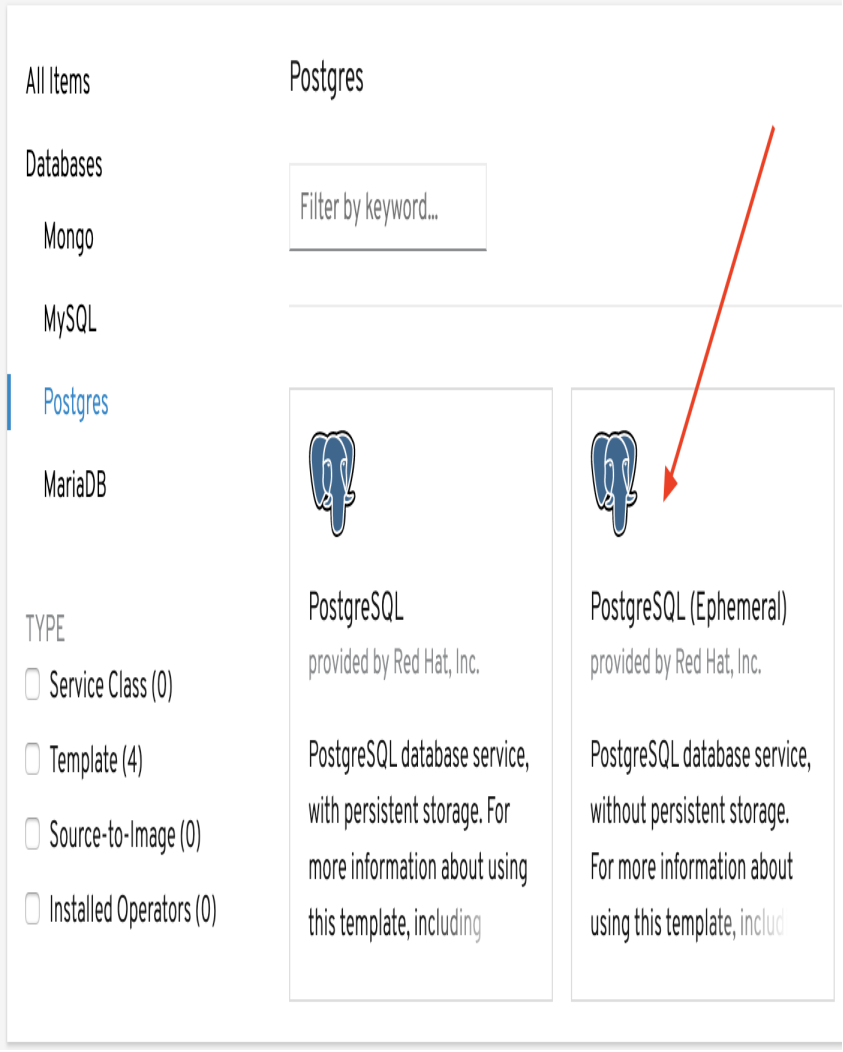


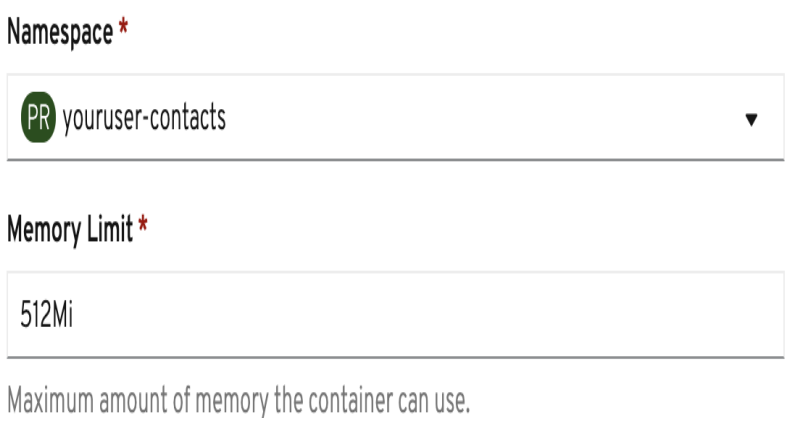
Figure 3.46: Add PostgreSQL Database

5. Click `Instantiate Template`, and then complete the form according to the following table:

Table 3.3. New Database Form

Form Field	Value
Database Service Name	contactsdb
PostgreSQL Connection Username	contacts
PostgreSQL Connection Password	contacts
PostgreSQL Database Name	contactsdb

Leave all other fields at their default values.



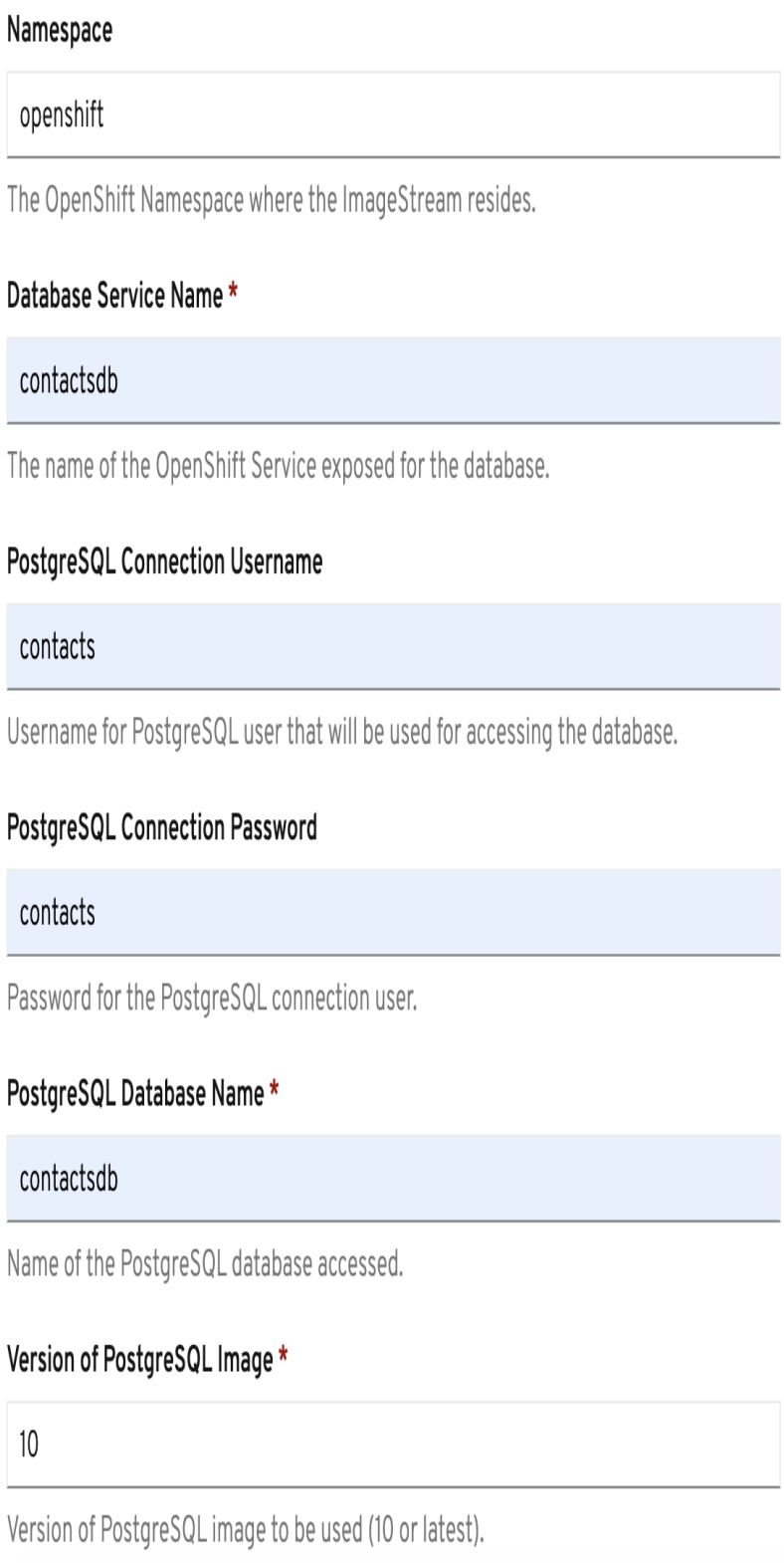


Figure 3.47: New PostgreSQL Database Form

Click **Create** to start deploying the database.

6. Click `Topology`, and then click the `contactsdb` deployment. Click `Resources`, and then verify that a single PostgreSQL database pod is running.

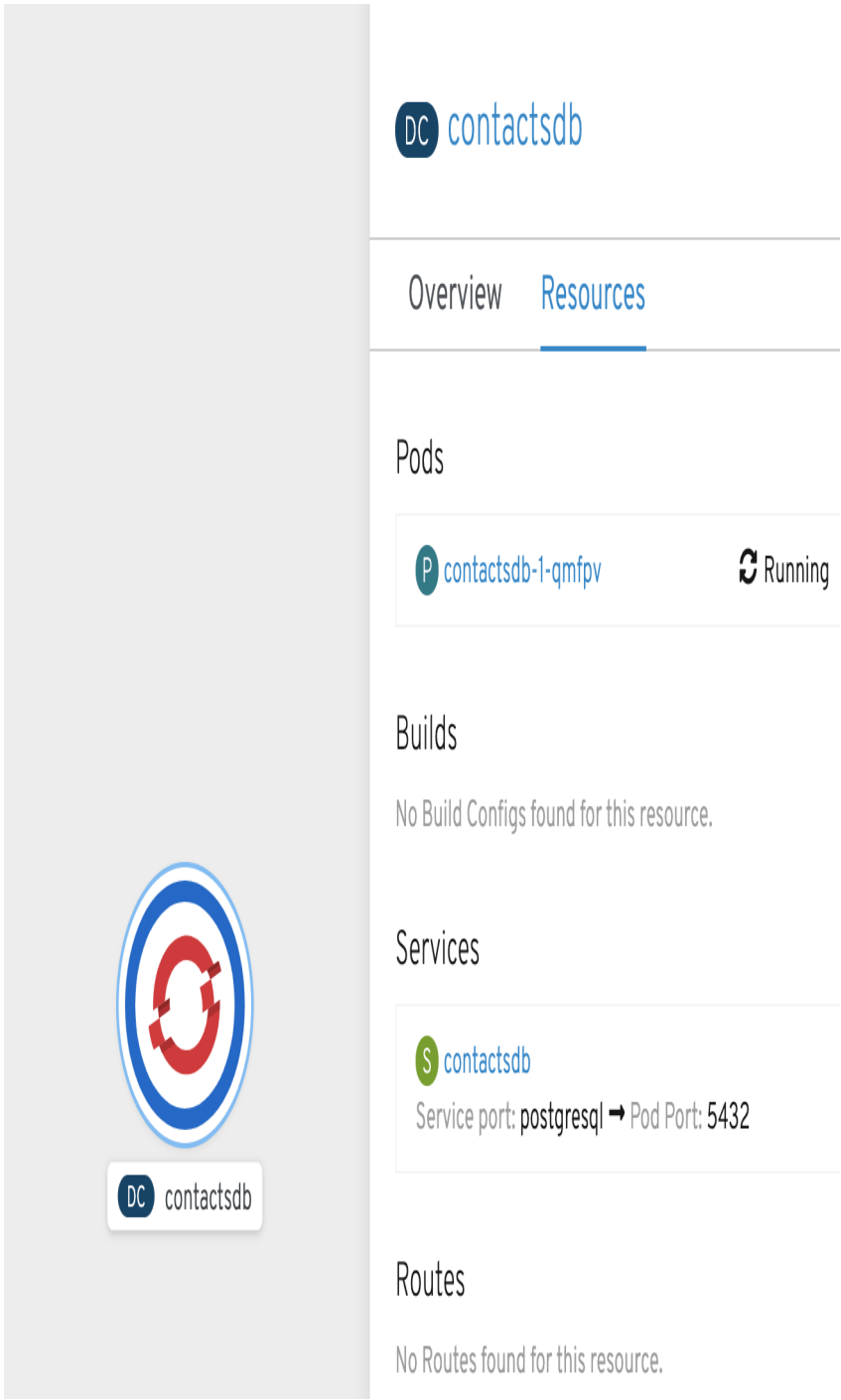


Figure 3.48: PostgreSQL Database Deployment

5. Create a secret to store the database connection information.

    1. Click **Advanced** → **Search**.

    2. Click on `Service`, and then search for `secret`. Select the `Secret` option.

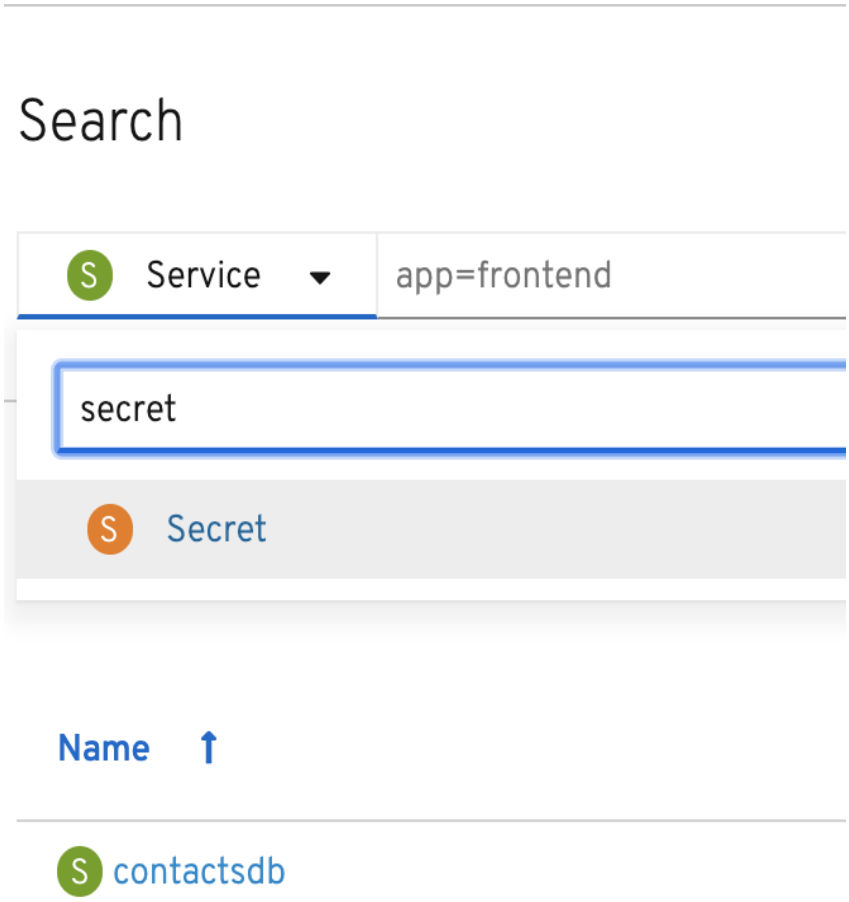


Figure 3.49: Search for Secrets

Ensure that your current project is `youruser-contacts`.

3. Click **Create** → **Key/Value Secret** to create a new secret.

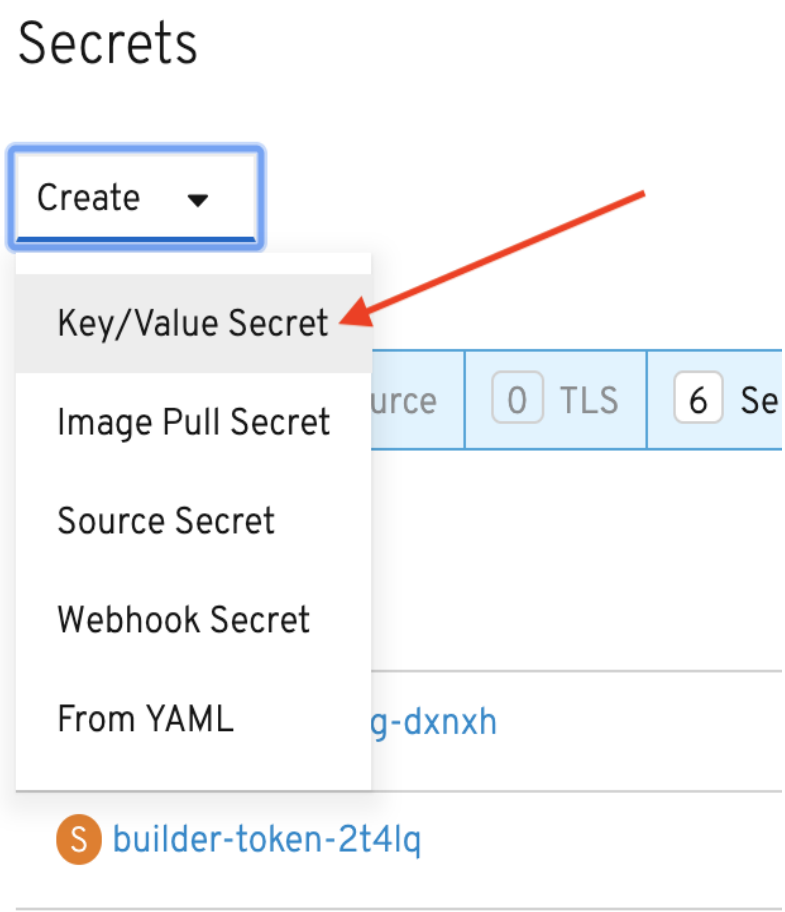


Figure 3.50: Create Secret

4. On the `Create Key/Value Secret` page, enter `contactsdb-secret` in the `Secret Name` field, `DB_CONFIG` in the `Key` field, and `postgresql://contacts:contacts@contactsdb:5432/contactsdb` in the `Value` field.

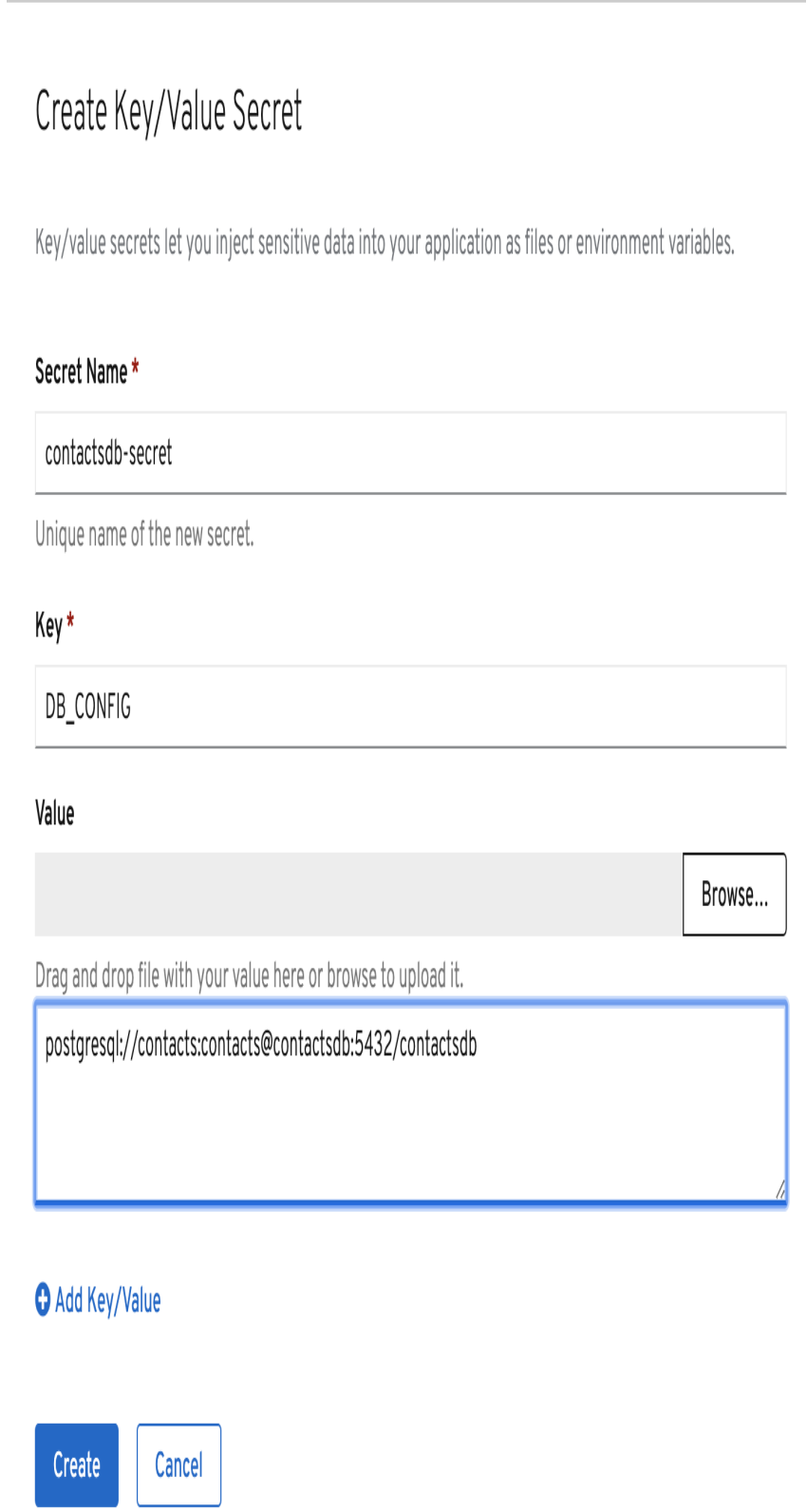


Figure 3.51: Secret Details

Recall that the `DB_CONFIG` environment variable is mapped to the `connectionString` attribute in the `DO101-apps/contacts/db/config.js` file. The PostgreSQL database driver module for Node.js expects the database URL to be of the form:

```
postgresql://<username>:<password>@<host>:<port>/<database>
```

5. Click **Create** to create the secret.

6. Deploy the `contacts` application to OpenShift.

    1. Select `Add`, and then click **From Catalog**.

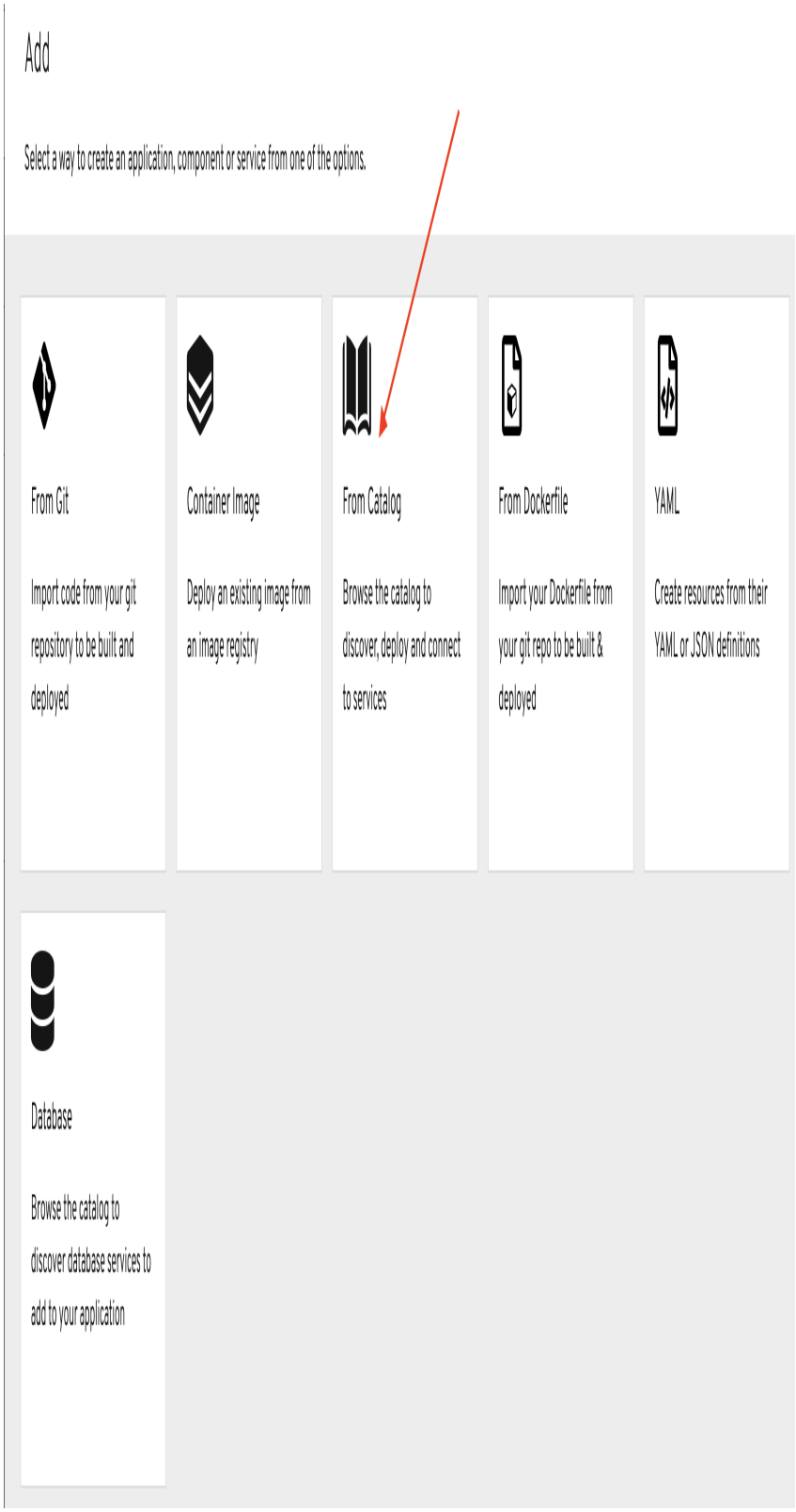


Figure 3.52: Add from Catalog

2. Select **Languages** → **JavaScript**, and then click the first option, `Node.js`. Click **Create Application** to enter the details of the application.

3. Complete the form according to the following table. To access the Git parameters, click **Show Advanced Git Options**.

   Table 3.4. New Application Parameters

Form Field	Value
`Git Repo URL`	https://github.com/`yourgituser`/DO101-apps
`Git Reference`	`contacts`
`Context Dir`	`/contacts`
`Application Name`	`contacts`
`Name`	`contacts`

   Do not click **Create** yet. First, you must customize the deployment configuration for the application to make use of the secret that you created in a previous step.

4. Click `Deployment Configuration` to customize the deployment configuration. Reference the secret that you created earlier.

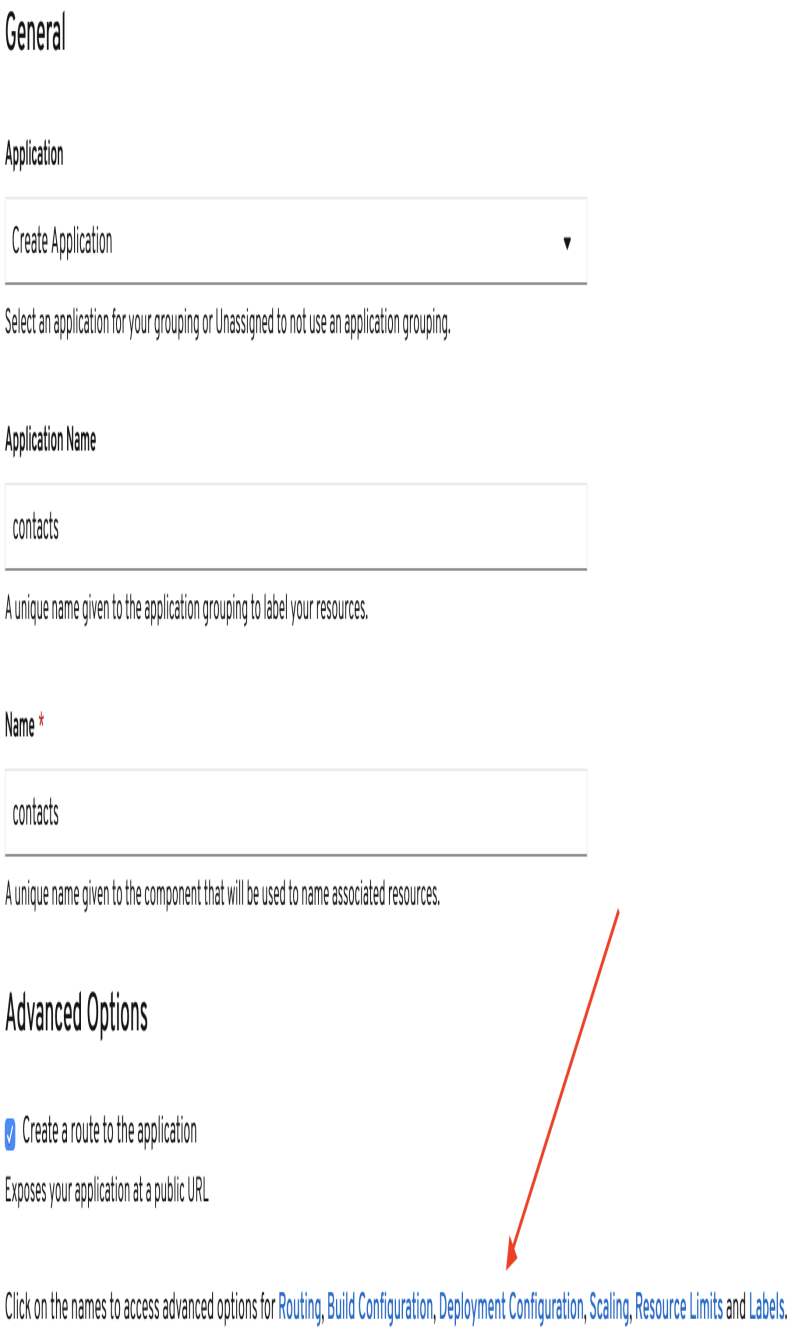


Figure 3.53: Deployment Configuration for contacts application

Click **Add from Config Map or Secret**

Figure 3.54: Add Environment Variables from Config Map or Secret

5. A new row is added to the `Environment Variables` table. Enter `DB_CONFIG` in the `NAME` field in the new row. Click the `Select a resource` list, search for `contacts`, and then select the `contactsdb-secret` secret.

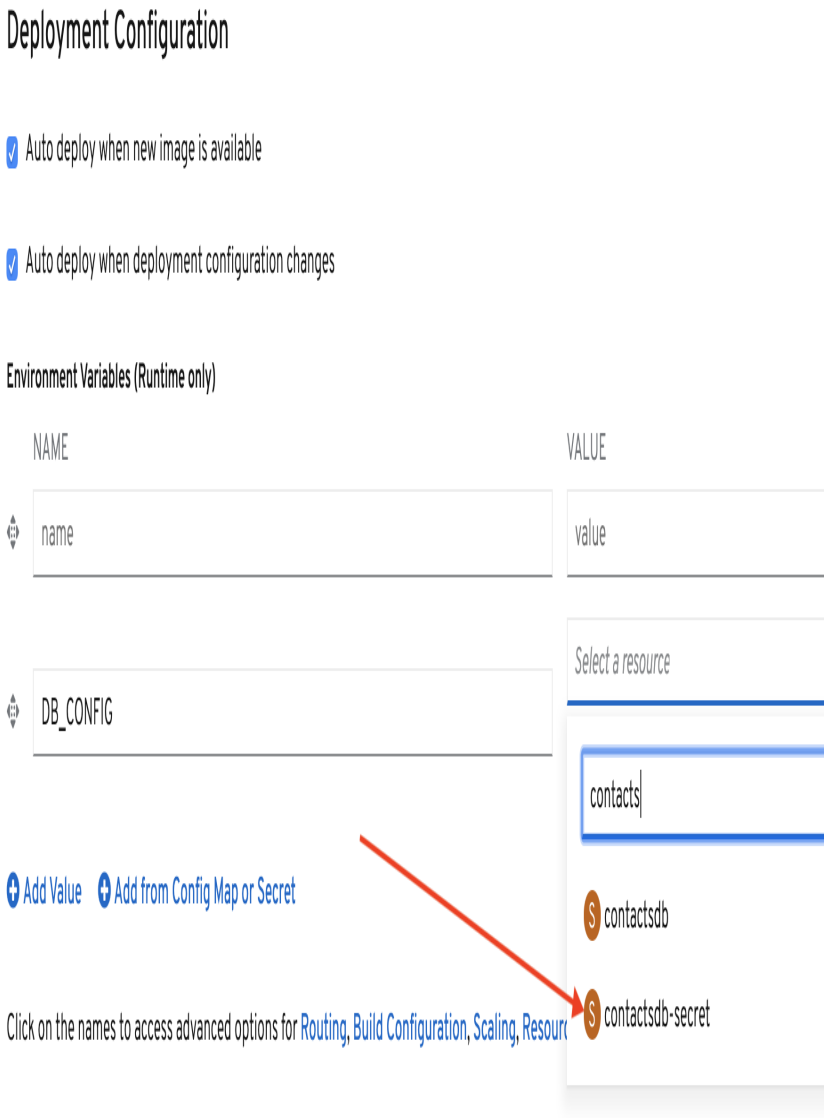


Figure 3.55: Select Secret

6. Click the `Select a key` list, and then select `DB_CONFIG`.

Figure 3.56: Select DB_CONFIG key from secret

7. Click the minus (-) symbol on the first row of the `Environment Variables (Runtime only)` table to delete the unused environment variable.

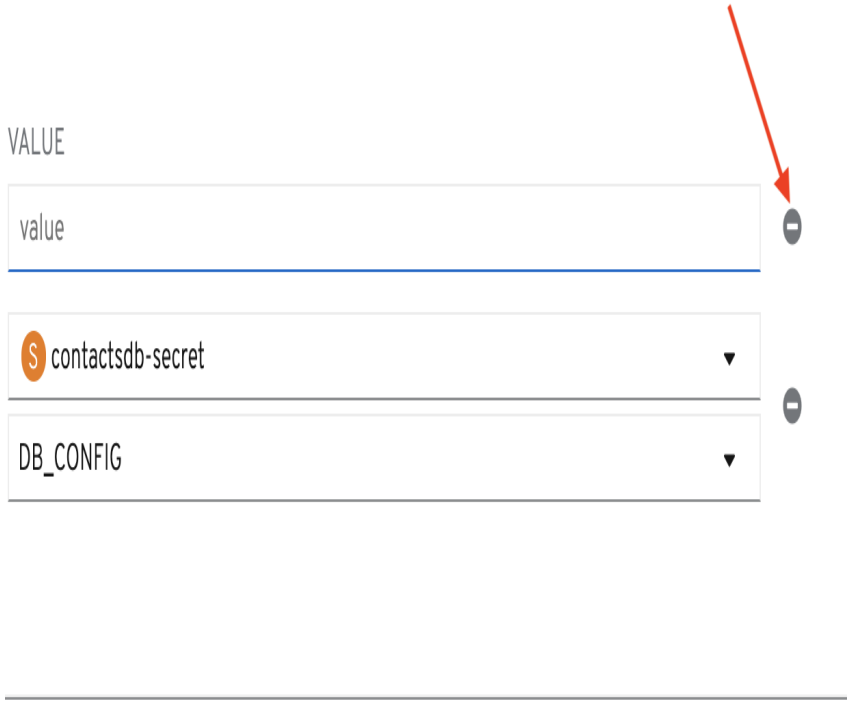


Figure 3.57: Delete Row

8. To avoid deployment errors in the following steps, review the values you entered in the form before proceeding. Click **Create** to start the build and deployment processes.

7. Test the application.

    1. Wait for a few minutes while the application container image is built and deployed. You should see a green tick mark for the `contacts` deployment on the `Topology` page.

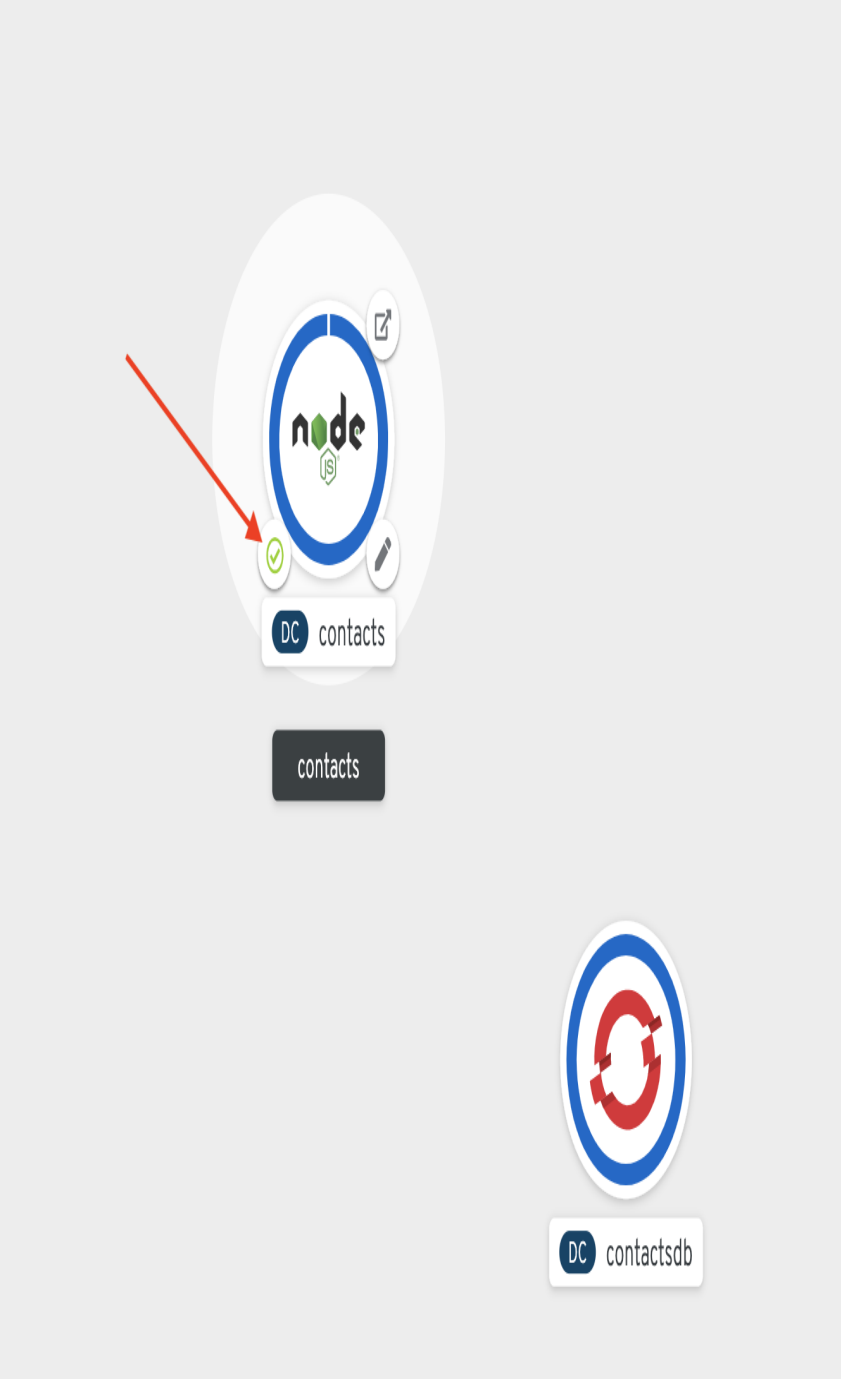


Figure 3.58: Contacts app built successfully

2. Click `Open URL` to open the route URL for the `contacts` application.

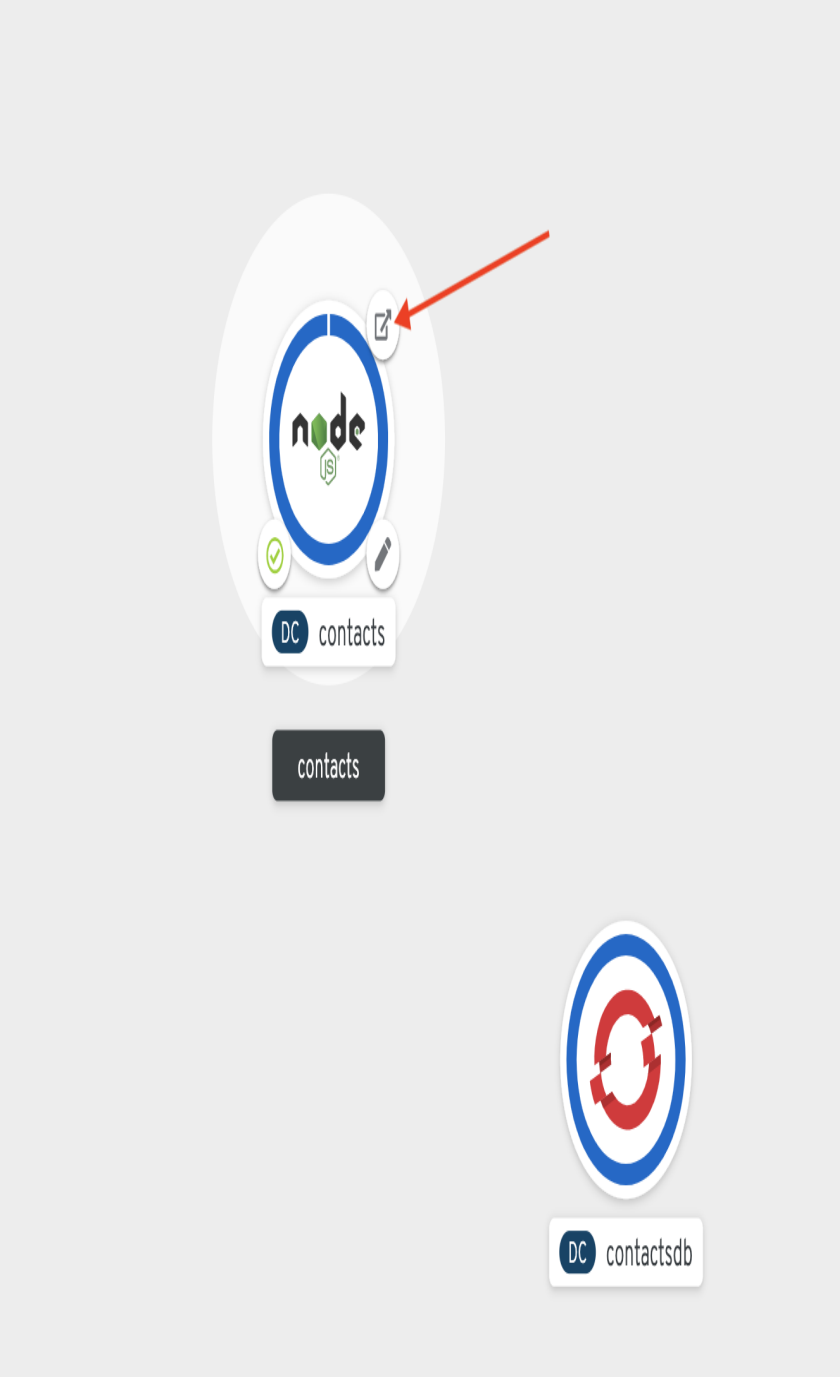


Figure 3.59: Contacts app route URL

3. The home page for the `contacts` application displays.

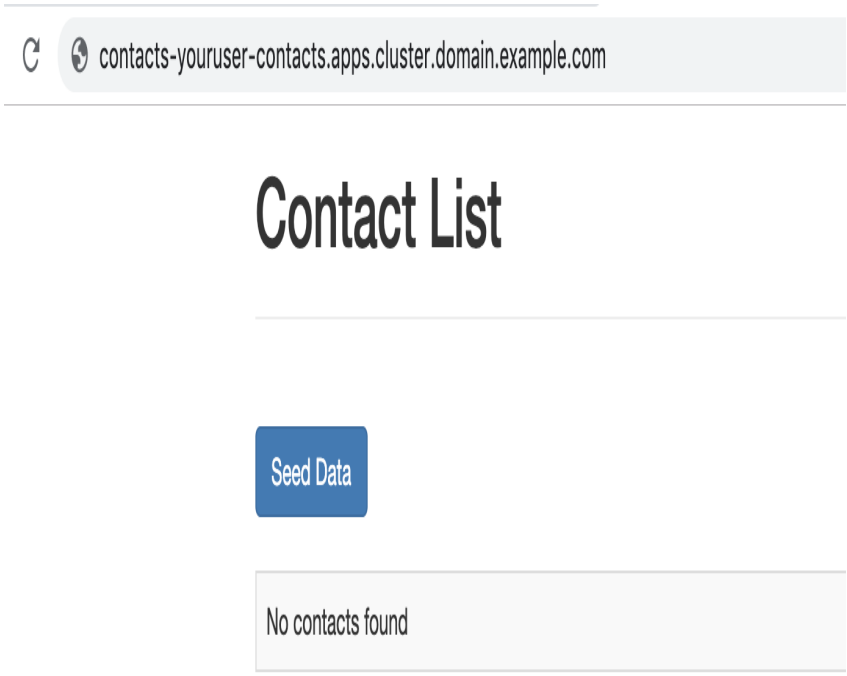


Figure 3.60: Contacts app Home Page

4. Click **Seed Data** to populate the database with sample contacts.
5. A table with five sample contacts that are fetched from the database displays.

Figure 3.61: Sample Contacts

Close the browser tab.

8. Verify contacts data in the PostgreSQL database using the OpenShift web console.
    1. Click `Topology`.
    2. Click on the `contactsdb` deployment, and then click `Resources`.
    3. Click the `contactsdb` pod, listed in the `Pods` section. The pod name will differ from this example.

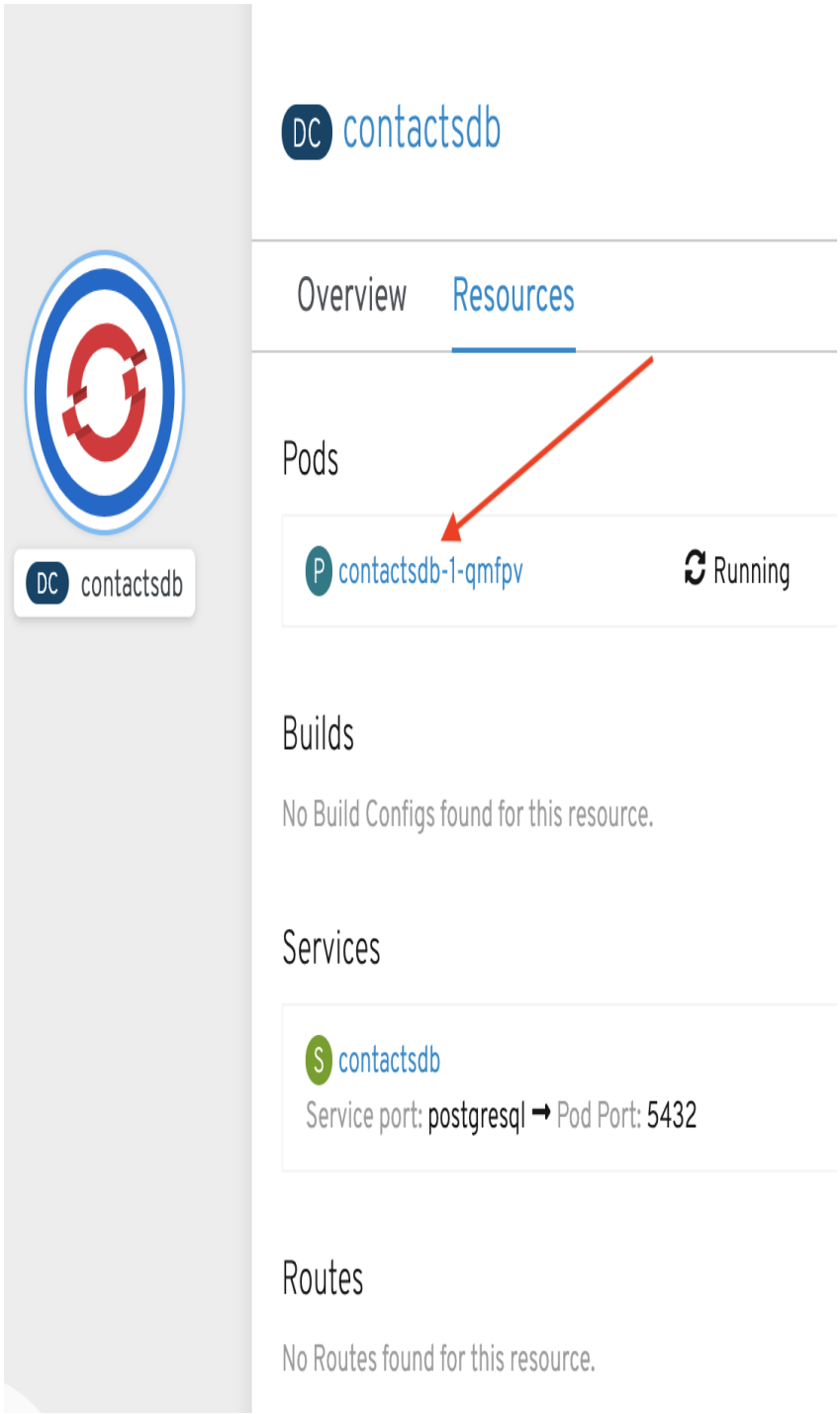


Figure 3.62: Click contactsdb Pod

4. On the pod details page, click on `Terminal` to open a command line terminal session inside the running PostgreSQL database pod.

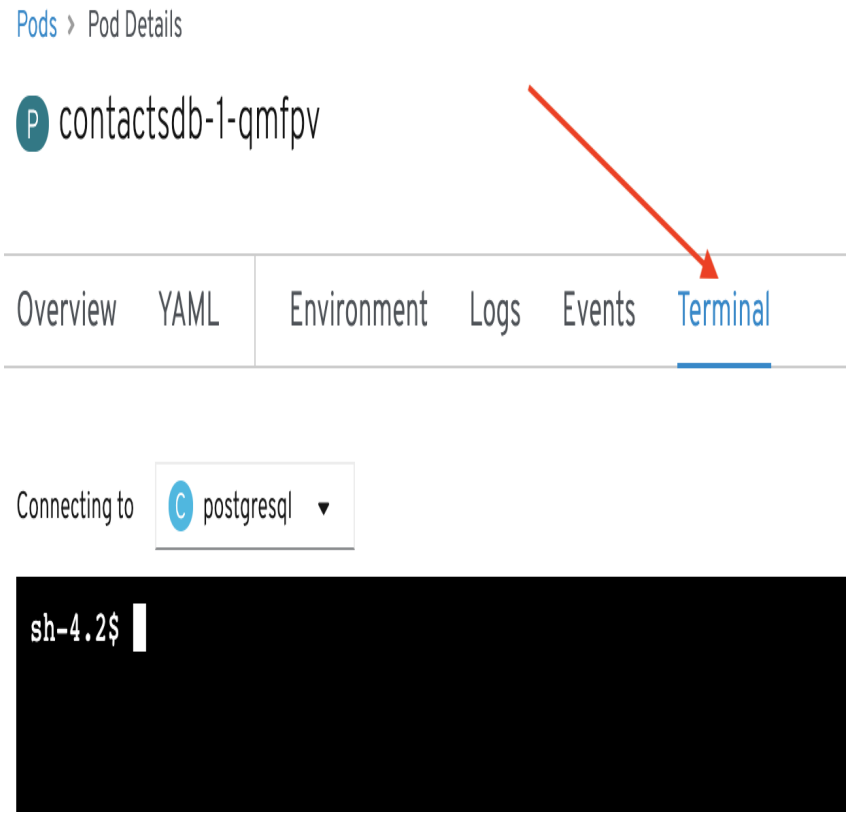


Figure 3.63: Click Terminal Tab

5. The PostgreSQL database client program `psql` is available to access the database. Run the following commands in the terminal to verify that the `contacts` table exists in the `contactsdb` database, and has five rows in it.

   First, connect to the `contactsdb` database.

   Next, list the tables in the database and verify that a table called `contacts` is displayed.

   Finally, run a select query to list the data in the `contacts` table.

   The following output displays.

Figure 3.64: Database Output

```
sh-4.2$ psql -U contacts contactsdb
contactsdb=> \dt
contactsdb=> select * from contacts;
```

9. Clean up. Delete the *youruser*-contacts project.

   In the OpenShift web console, click on **Advanced** → **Projects** to view the list of projects, and then delete the youruser-contacts project. A prompt to confirm the deletion displays.

Enter *youruser*-contacts in the confirmation window and delete the project.

This concludes the guided exercise.

Chapter 4. Scaling Applications in OpenShift

Scaling an Application

Objectives

After completing this section, you should be able to scale an application deployed on Red Hat OpenShift Container Platform to meet load demand.

Describing Pod Scaling

Most real-world applications do not run only in a single pod. They often need to run in several pods to meet growing user demand. By duplicating the application in multiple pods, the application is scaling to meet user demand. When a user makes a request to access the application, OpenShift automatically directs the service request to an available pod so that more users can concurrently access the application. If more pods running the application are available, then users are less likely to experience an outage or unavailable application.

Some applications receive a large number of concurrent requests only during certain periods, which makes it very difficult to size the number of pods up front before running the application. However, there are extra costs associated with running more pods than required when traffic is not at its peak.

Red Hat OpenShift Container Platform refers to the action of changing the number of pods for an application as scaling. `Scaling up` refers to increasing the number of pods for an application. `Scaling down` refers to decreasing that number. Scaling up allows the application to handle more client requests, and scaling down provides cost savings when the load goes down.

Remember that for customers to reach an application from outside OpenShift, you must create a route resource that associates a public URL to the application. When scaling your application, OpenShift automatically configures that route to distribute client requests among member pods.

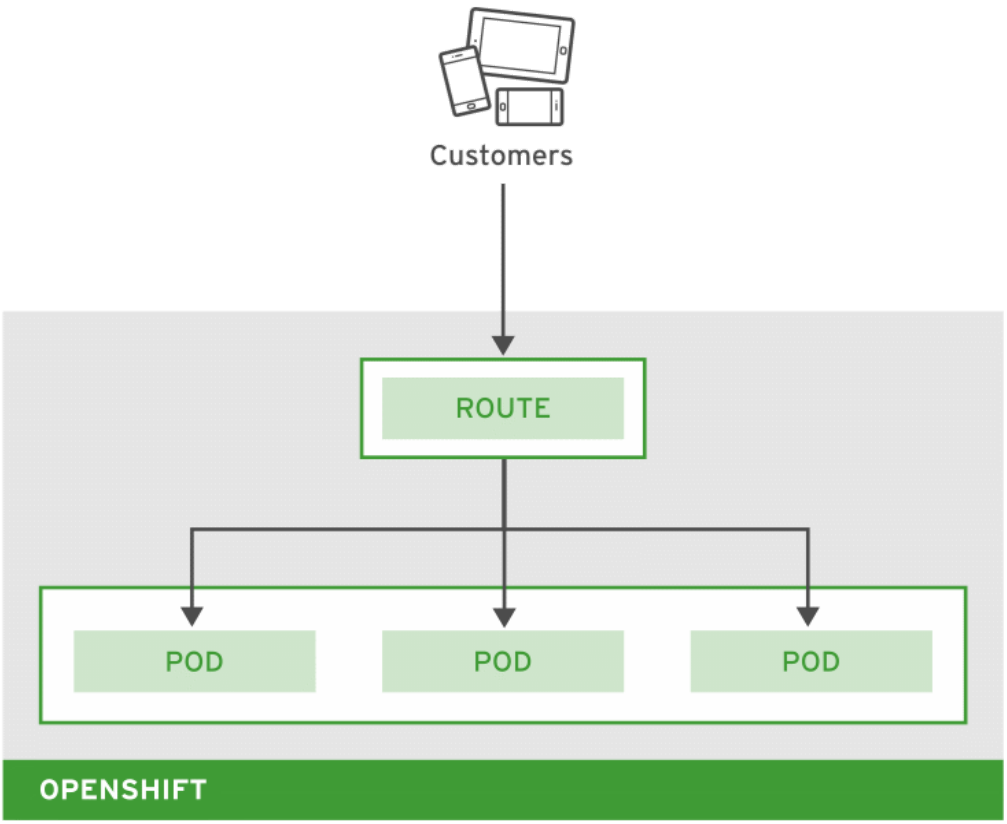


Figure 4.1: The route resource distributes client requests

When scaling up an application, the OpenShift platform first deploys a new pod and then waits for the pod to be ready. Only after the new pod becomes available does the OpenShift platform configure the route to also send traffic to the new pod. When scaling down, OpenShift reconfigures the route to stop sending traffic to the pod, and then deletes the pod.

Preparing your Application for Scaling

OpenShift allows any application to scale by creating multiple pods, but this does not mean that every application automatically becomes scalable because it is running in OpenShift. The application must be able to work correctly with multiple instances of itself.

Some web applications maintain user state using some kind of HTTP session abstraction, usually with HTTP cookies. Because OpenShift can direct a user to any running pod, that session information must be available globally, and not only in the first pod that the user reaches. Therefore, these applications must keep the session information in a central store, such as a database or in-memory shared store like Redis or memcached, and not only in the pod local file system.

Databases, such as MariaDB and PostgreSQL, do not usually support running in multiple pods. For these scenarios, there are alternative solutions. For example, the MariaDB operator at https://access.redhat.com/containers/?tab=overview#/registry.connect.redhat.com/mariadb/operator provides a deployment that can scale on multiple pods.

For PostgreSQL, refer to the PostgreSQL Operator on OpenShift documentation at https://blog.openshift.com/leveraging-the-crunchy-postgresql/ for more details.

Configuring the Route Resource

By default, OpenShift tries to direct all requests from a client to the same pod. You can configure the route resource to modify this behavior. For example, if the round-robin algorithm is selected, OpenShift distributes, or load balances, the requests in a round-robin fashion between the pods. With two pods, for example, OpenShift sends the first request to the first pod, the second request to the second pod, the third request to the first pod again, and so on.

To access the route details, from the `Topology` page, click the application icon, then click the `Overview` tab, and then select the route resource.

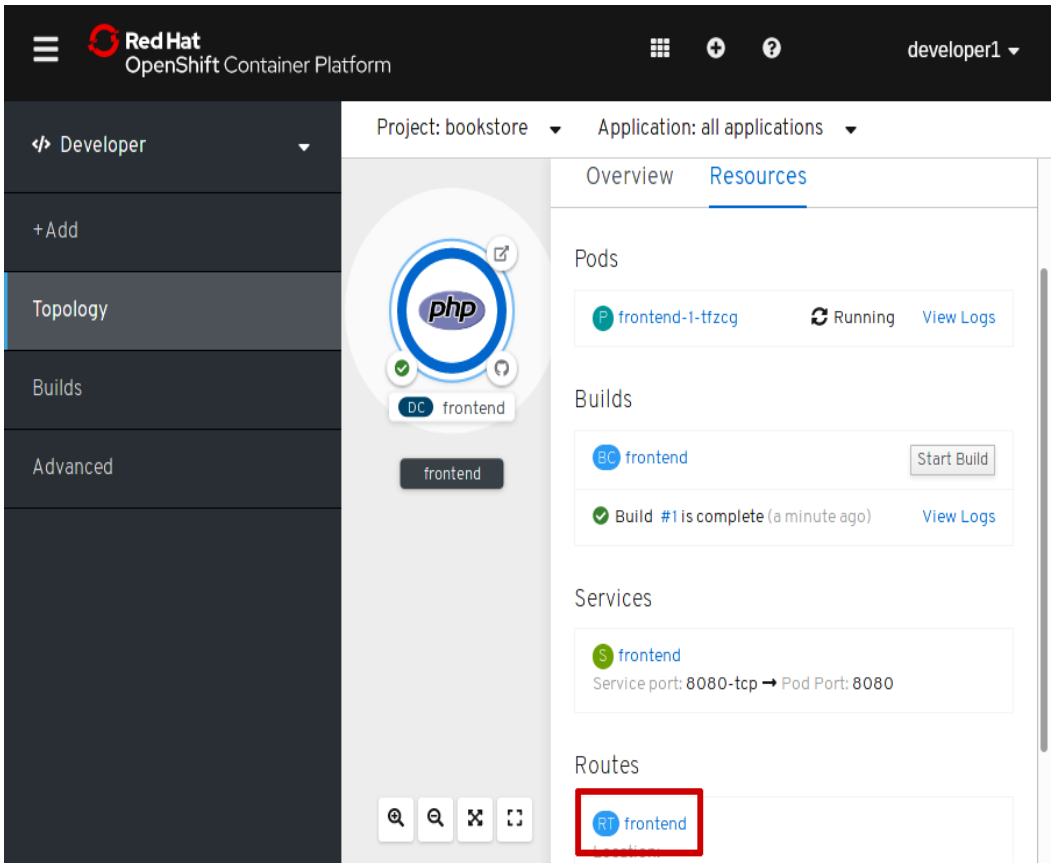


Figure 4.2: Accessing the route resource details

Modify the `annotations` section in the YAML file to edit the route parameters.

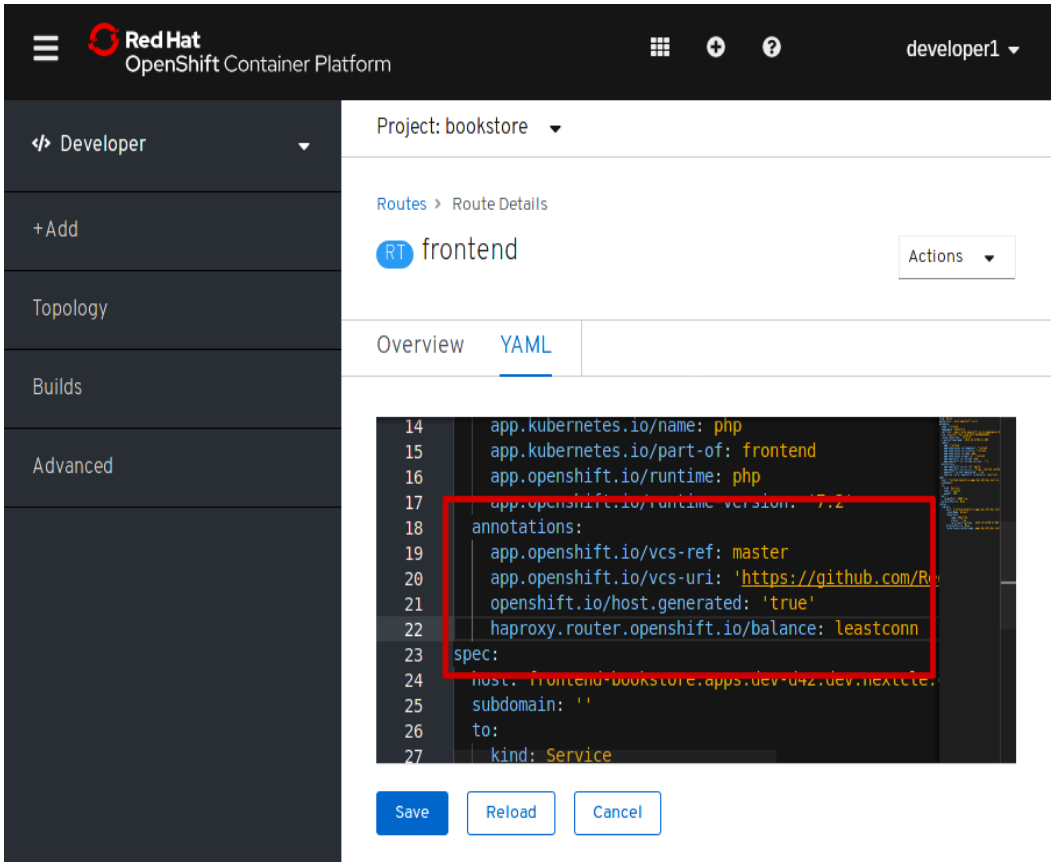
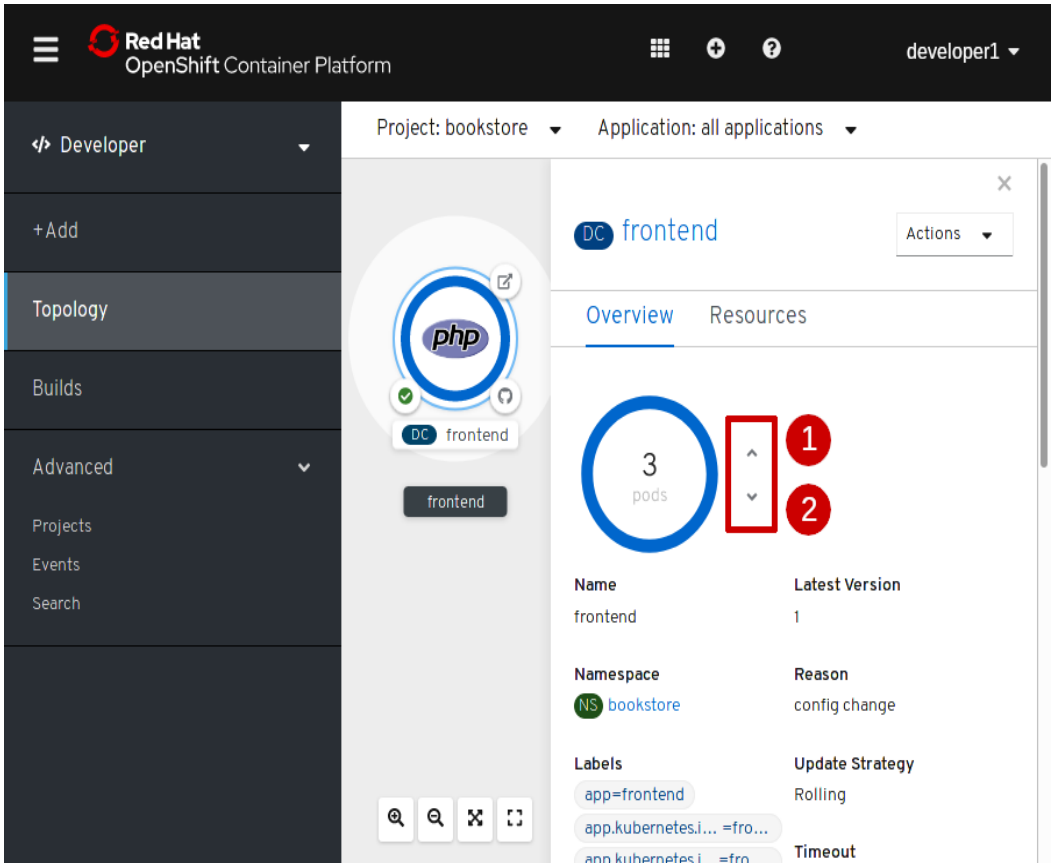


Figure 4.3: Updating the route resource

Scaling an Application Manually

With the OpenShift web console, developers can manually scale their applications. For example, the developer may increase the number of pods when anticipating a surge in the application load.



1. Increases the pod count.
2. Decreases the pod count.

Configuring the Horizontal Pod Autoscaler

In addition to manual scaling, OpenShift provides the `Horizontal Pod Autoscaler (HPA)` feature. HPA automatically increases or decreases the number of pods depending on average CPU utilization. Developers can also configure HPA to use custom application metrics for scaling. Although this advanced configuration is outside the scope of this course, the reference section provides a link for more information on this subject.

The OpenShift web console does not provide an interface to enable HPA. Therefore, developers must use the `oc` command-line client. In the following example, the command enables and configures HPA for the `frontend` deployment configuration (`dc`):

```
$ oc autoscale dc/frontend --min=1 --max=5 --cpu-percent=80
```

The options are as follows:

`dc/frontend`

Name of the application deployment configuration resource

`--min=1`

Minimum number of pods

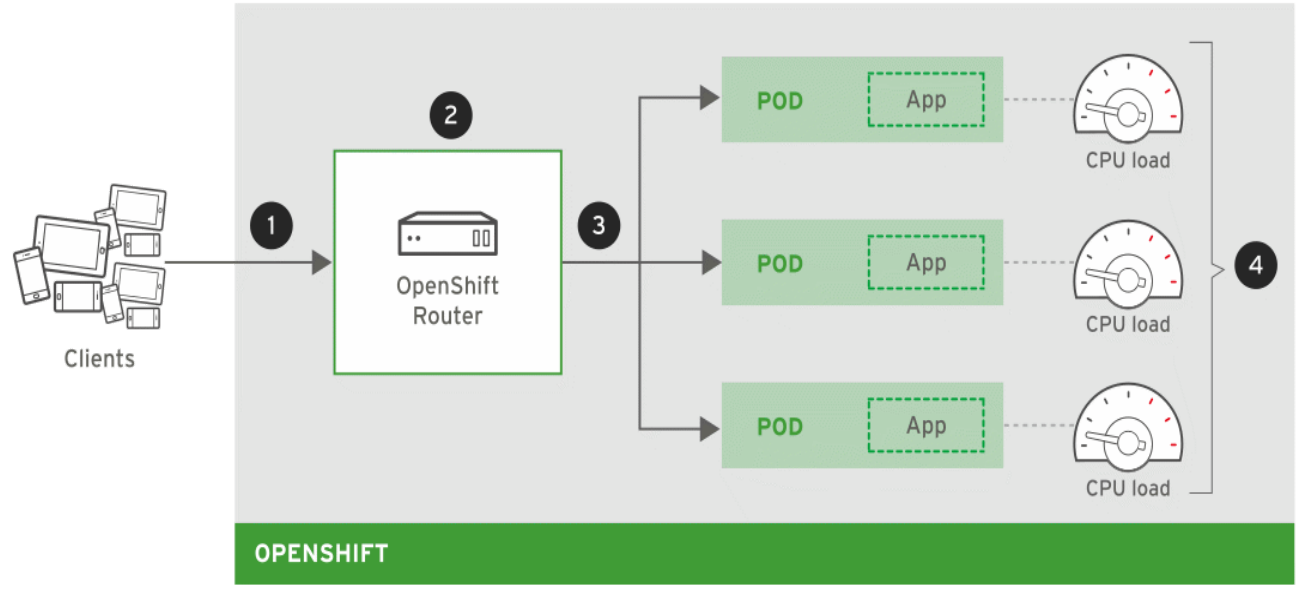
`--max=5`

Maximum number of pods. HPA does not scale up the application beyond this limit, even if the load continues to increase.

`--cpu-percent=80`

Ideal average CPU utilization for each pod. If the global average CPU utilization is above that value, then HPA starts a new pod. If the global average CPU utilization is below the value, then HPA deletes a pod.

The following diagram shows how HPA scales an application based on CPU utilization.



1. Clients access the application.
2. The OpenShift router distributes the requests to the pods.
3. The application in the pods processes the request.
4. When the number of clients increases, the CPU utilization increases in each pod. When the CPU utilization is above the configured value, HPA s pod.

Guided Exercise: Scaling an Application

In this exercise, you will configure OpenShift to automatically scale up an application to meet load demand.

Outcomes

You should be able to use OpenShift to:

- Manually scale an application using the web console.
- Configure OpenShift to automatically scale an application when the CPU load increases.

To perform this exercise, ensure that you have access to a running OpenShift cluster, VS Code, Git, and the `oc` command are installed on your system, and you have cloned your GitHub `DO101-app` repository in VS Code.

Procedure 4.1. Steps

1. In Visual Studio Code (VS Code), in the `DO101-apps` repository, create a new branch named `scale`. Push the branch to your GitHub repository.

    1. Open VS Code. Click **View** → **SCM** to access the source control view. Ensure that the `DO101-apps` entry under `SOURCE CONTROL PROVIDERS` shows the `master` branch.

    NOTE

    If the Source Control view does not display the `SOURCE CONTROL PROVIDERS` heading, then right-click `SOURCE CONTROL` at the top of the Source Control view and select `Source Control Providers`.

    If you were working with another branch for a different exercise, click the current branch and then select `master` in the `Select a ref to checkout` window.

WARNING

Each exercise uses a unique branch. Always create a new branch using `master` as the base.

2. Click **master** in the `DO101-apps` entry under `SOURCE CONTROL PROVIDERS`, and then select `Create new branch`. Type `scale` to name the branch.

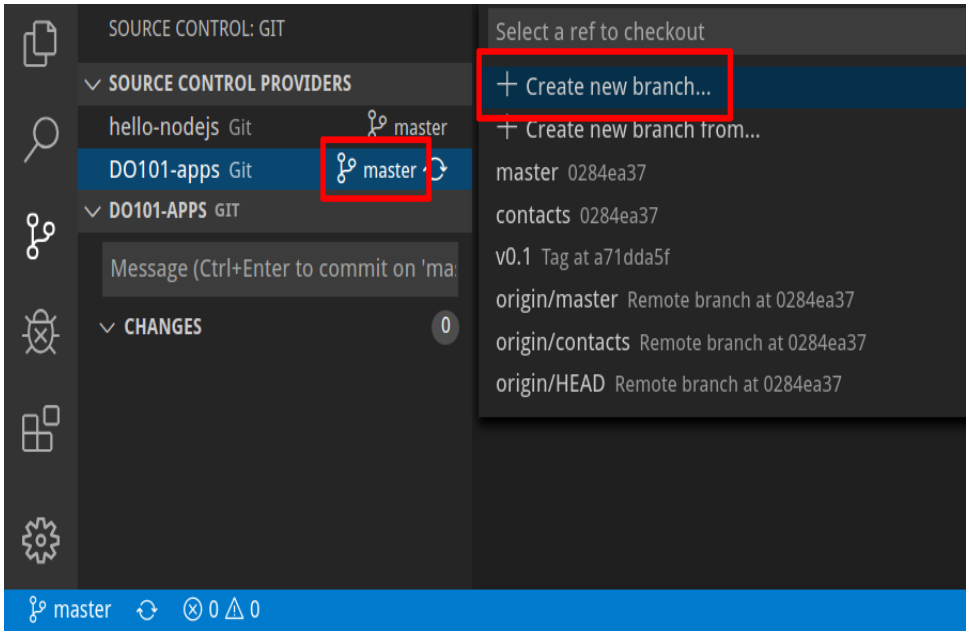


Figure 4.4: Creating a new branch with VS Code

3. To push the new branch to GitHub, click the `Publish Changes` icon for the `DO101-apps` entry. VS Code may ask for your GitHub login name and password.

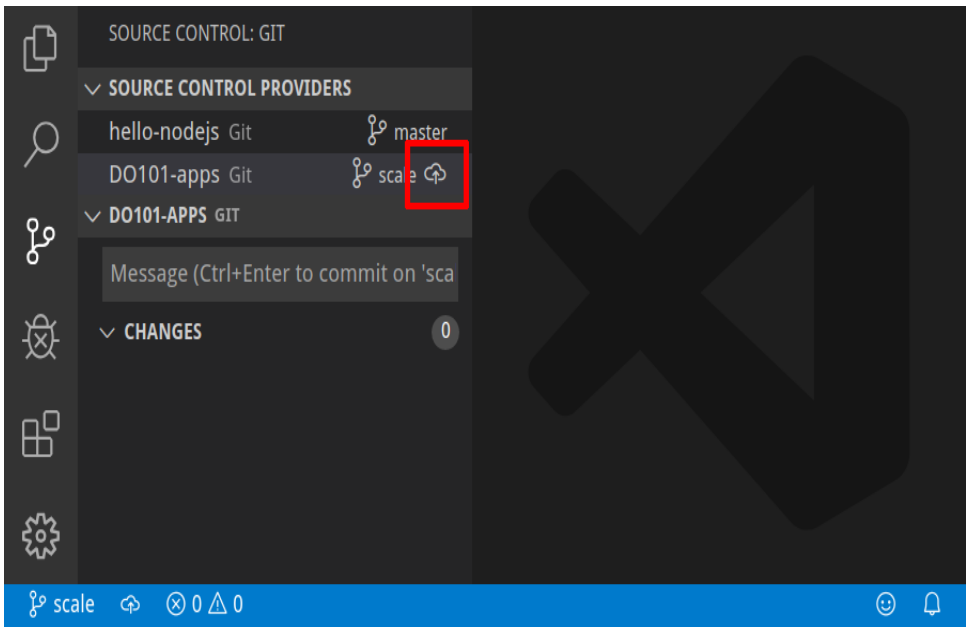


Figure 4.5: Pushing the new branch to GitHub

2. Access the Red Hat OpenShift Container Platform web console and create a new project named *`youruser`*`-scale`. Replace *`youruser`* with your user name.

    1. Log in to the OpenShift web console using your developer account.

2. Select the `Developer` perspective from the navigation pane.
3. On the **Advanced** → **Projects** menu, click **Create Project**.
4. Enter *`youruser`*`-scale` in the `Name` field. Replace *`youruser`* with your user name. Leave the other fields empty and click **Create**.

3. Deploy the `scale` PHP application. When accessed from the internet, this application displays the name and IP address of its pod. The source code for this application is in the `php-scale` subdirectory of your Git repository. Use the `scale` branch to deploy the application to OpenShift.
   1. Click the `Add` tab on the left side of the page and then click **From Catalog**.
   2. Click **Languages** → **PHP** and then click `PHP`.

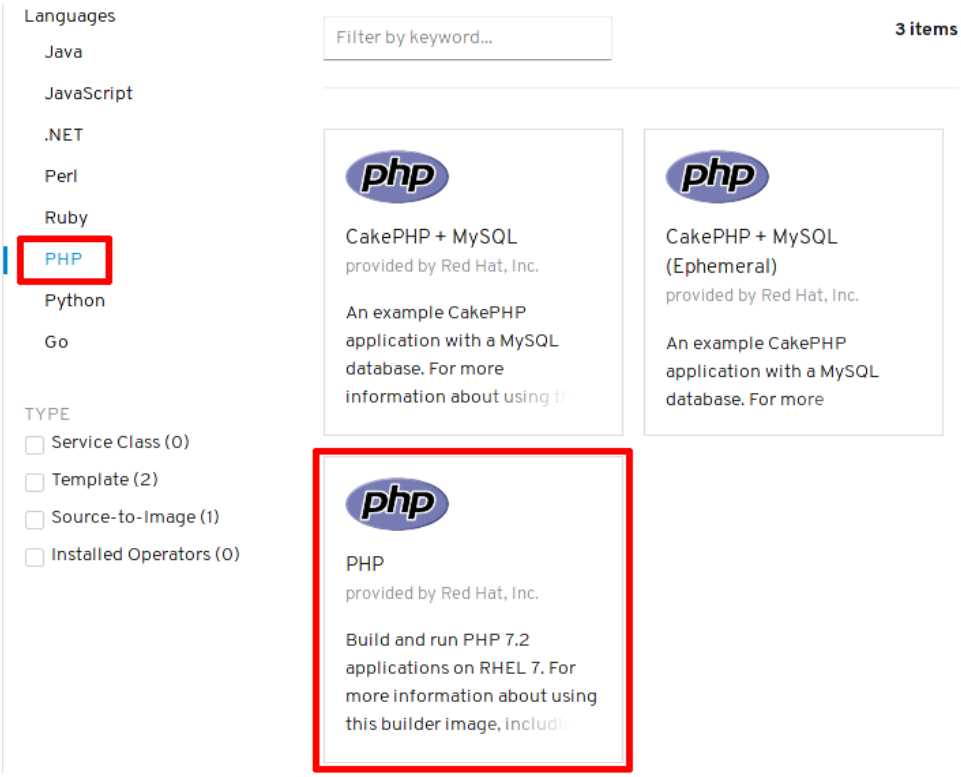


Figure 4.6: Creating a PHP application

Click **Create Application** to enter the details of the application.

3. Complete the form according to the following table. To access the Git parameters, click **Show Advanced Git Options**.

Table 4.1. New Application Parameters

Parameter	Value
`Git Repo URL`	https://github.com/*`yourgituser`*/DO101-apps
`Git Reference`	`scale`
`Context Dir`	`/php-scale`
`Application Name`	`scale`
`Name`	`scale`

To avoid unexpected errors, review the values that you entered in the form before proceeding:

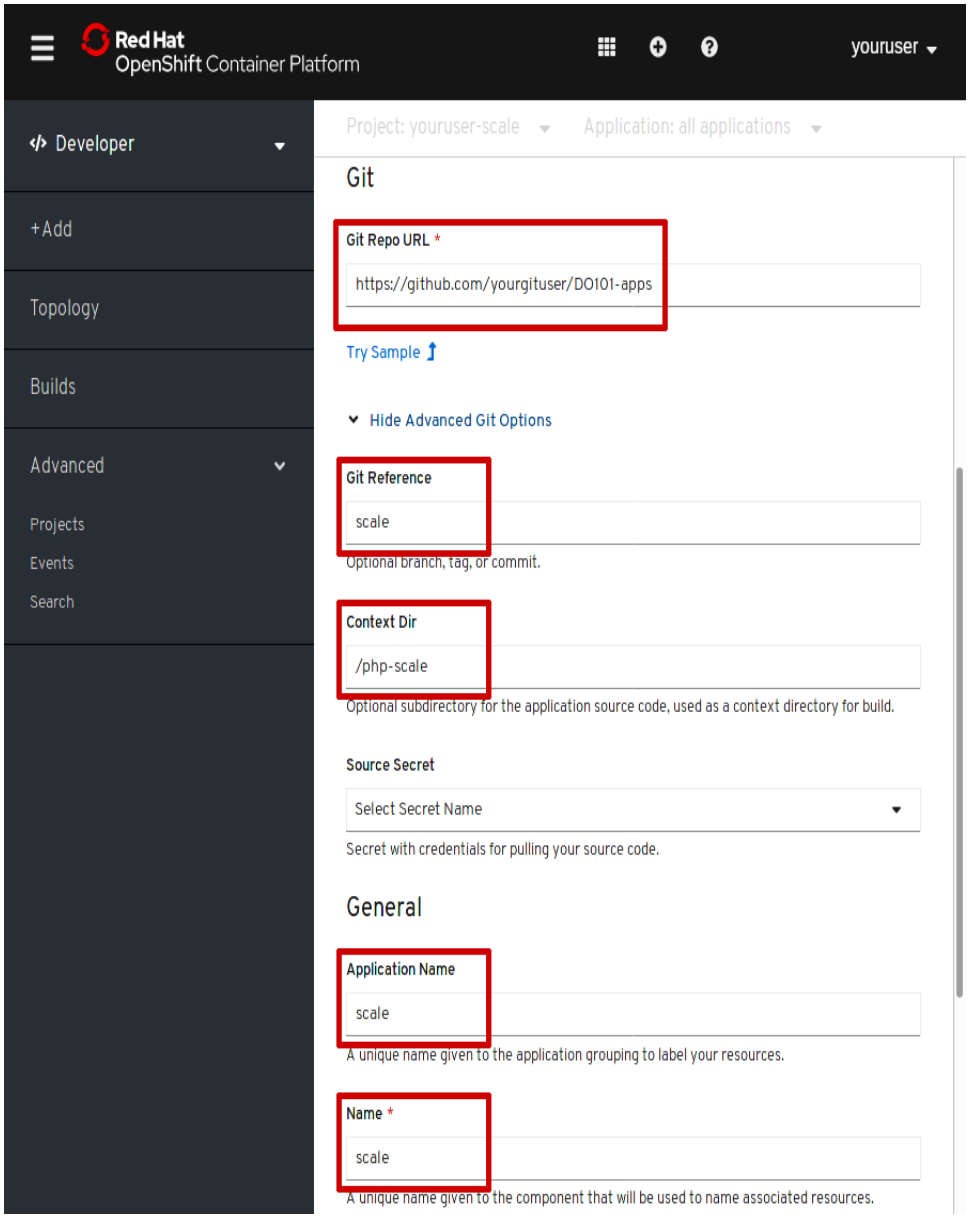


Figure 4.7: Completing the application creation form

Click **Create** to start the build and deployment processes.

4. Wait for OpenShift to build and deploy your application and verify that you can access it from the internet.
    1. The web console should automatically show the `Topology` page. If necessary, click the `Topology` tab.
    2. Wait for the application icon to show that the deployment is finished, and then click the **Open URL** icon to access the application.

Figure 4.8: Accessing the scale application

A new browser tab displays the application.

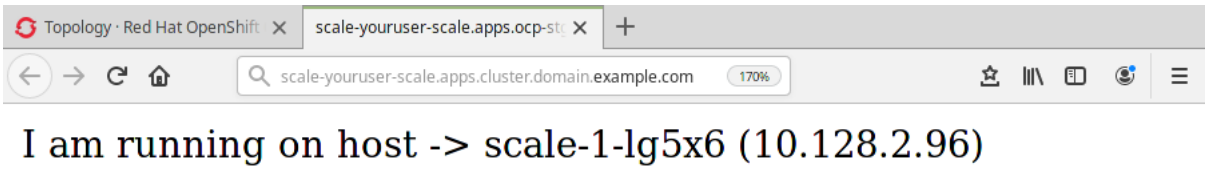


Figure 4.9: Displaying the scale application

The host name and IP address in the preceding screen capture are probably different on your system.

3. Close the application browser tab.

5. Scale the application to two pods and confirm that you can still access it from the internet.

    1. From the `Topology` page, click the `PHP` icon, and then click the `Overview` tab.
    2. Click the `Increase the pod count` arrow to scale the application to two pods.

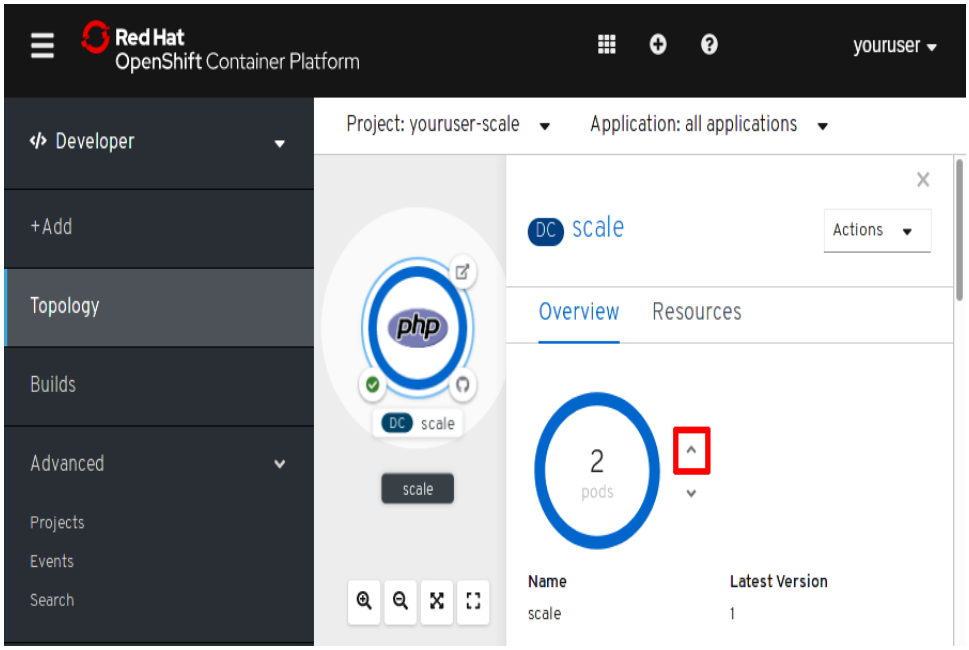


Figure 4.10: Scaling up the application

3. Wait a few seconds for the second pod to start, and then click the **Open URL** icon to confirm that you can still access the application.

4. Close the application browser tab.

6. By default, the load balancer redirects all requests from a particular client to the same pod. Use your web browser to test that OpenShift distributes the requests between the two pods, configure the route resource to disable the affinity between clients and pods.

    1. From the `Topology` page, click the `PHP` icon and then the `Resources` tab.

    2. Click the `scale` route resource.

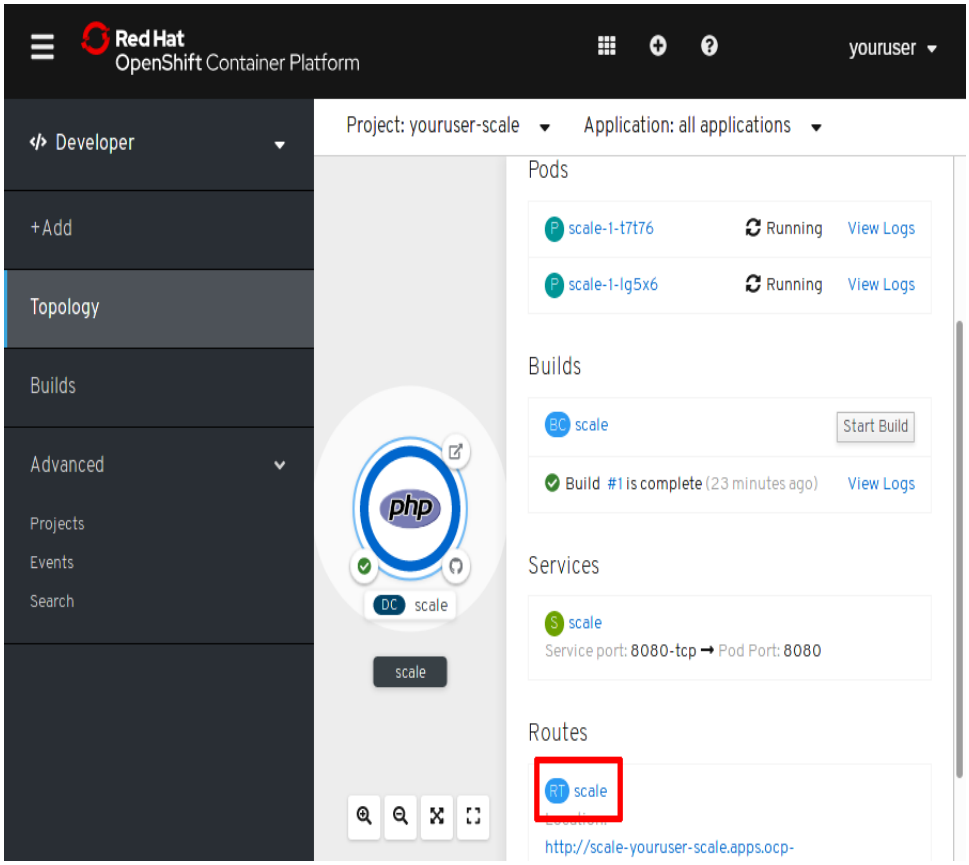


Figure 4.11: Accessing the route resource

3. Click the `YAML` tab, and then use the editor to add the following two lines in the `annotations` section:

> **IMPORTANT**
>
> Ensure that you maintain correct indentation as shown in the example. YAML files are indentation sensitive.

4. Click **Save**. OpenShift automatically reconfigures the load balancer with the new configuration.
5. Use your web browser to confirm that OpenShift distributes the requests to the PHP application between the two pods. From the `Topology` page, click the PHP application **Open URL** button. A new browser tab displays the application. Refresh the page several times. Notice that the pod name and IP address alternate with every request.
6. Close the application browser tab.

1. Scale down the application to one pod.
   1. From the `Topology` page, click the `PHP` icon, and then click the `Overview` tab.
   2. Click the `Decrease the pod count` arrow to scale the application to one pod.
   3. Wait for the second pod to terminate, and then click the PHP application **Open URL** button to access the application again. Refresh the application page several times. Notice that the page always displays the same pod name and IP address. OpenShift directs all requests to the remaining pod.
   4. Close the application browser tab.
2. In the second part of this exercise, you configure OpenShift to automatically scale up your application when the CPU load is above 20%.

   You cannot configure the autoscaler from the web console, so use the `oc` command on your system instead.

   1. Open a new command-line terminal and log in to your OpenShift account using the API endpoint URL. In the following command, replace the URL with the value from your Red Hat Training Online Learning environment, and use the same user name and password that you use to log in to the OpenShift web console.

WARNING

The API endpoint URL and the OpenShift web console URL are **not** the same. The API endpoint URL is used mainly to interact with OpenShift, using command-line tools such as `oc`.

2. Select the *youruser*`-scale` project:
3. Configure the autoscaler for the application. Set the maximum number of pods to three and the CPU load to 20%.

9. To test your configuration, deploy the `stress` application to trigger the autoscaler.

NOTE

The `stress` application has been specially developed for this course. It uses the Apache HTTP server benchmarking tool, or `ApacheBench` (`ab`), to send a lot of concurrent requests to your PHP application.

The `ab` command is called from the `DO101-apps/stress/Dockerfile` file:

```
ab -dSrk -c 20 -n 50000000 \
  http://${SCALE_SERVICE_HOST}:${SCALE_SERVICE_PORT}/index.php
```

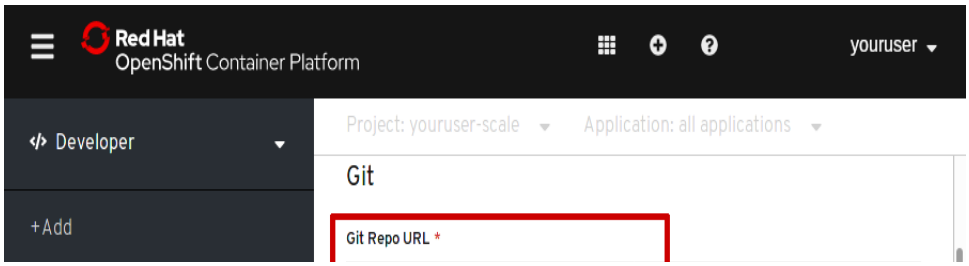
Notice the use of the `SCALE_SERVICE_HOST` and `SCALE_SERVICE_PORT` variables to refer to the `scale` application. OpenShift automatically sets those variables in all the pods in the project.

1. In the OpenShift web console, click the `Add` tab, and then click **From Dockerfile**.
2. Complete the form according to the following table. To access the Git parameters, click **Show Advanced Git Options**.

Table 4.2. Stress Application Parameters

Parameter	Value
`Git Repo URL`	https://github.com/*yourgituser*/DO101-apps
`Git Reference`	`scale`
`Context Dir`	`/stress`
`Application`	Select `Create application` from the list.
`Application Name`	`stress`
`Name`	`stress`
`Create a route to the application`	Clear the check box

To avoid unexpected errors, review the values that you entered in the form before proceeding.



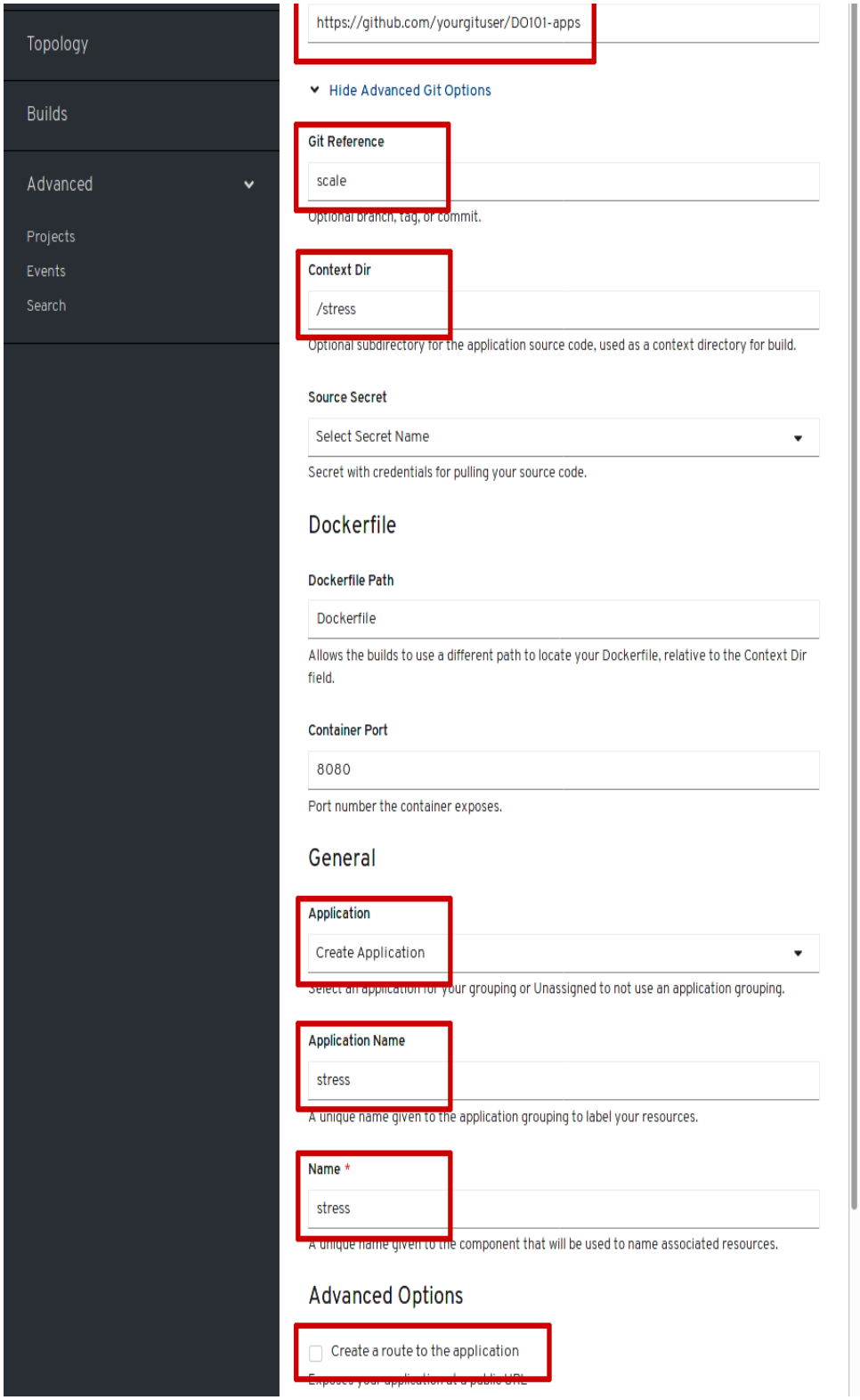


Figure 4.12: Completing the stress application creation form

Click **Create** to start the build and deployment processes.

3. Wait for the `stress` application to deploy, and then consult the logs of the pod to confirm that the `ab` command is sending requests. From the `Topology` page, click the `stress` icon, and then click the `Resources` tab. Click `View Logs` near the pod.

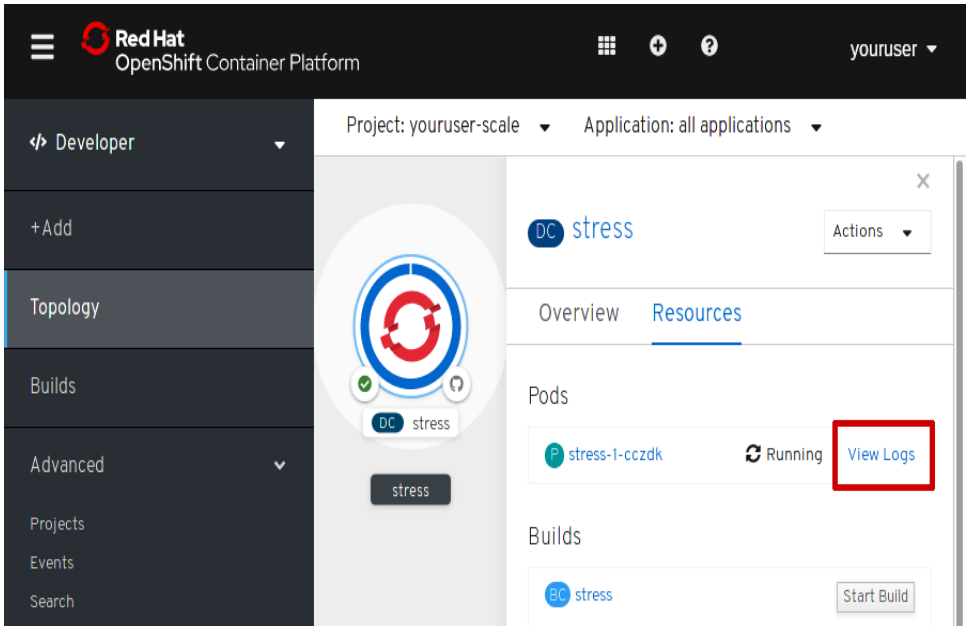


Figure 4.13: Accessing the logs of the stress pod

Notice that `ApacheBench` is running:

The IP address displayed in the preceding output is probably different on your system.

10. Inspect the `scale` application and confirm that the number of pods automatically increases to three.
    1. From the `Topology` page, click the `PHP` icon, and then click the `Overview` tab.
    2. Notice that the number of pods for the `scale` application increases to three. You may have to wait a few minutes.

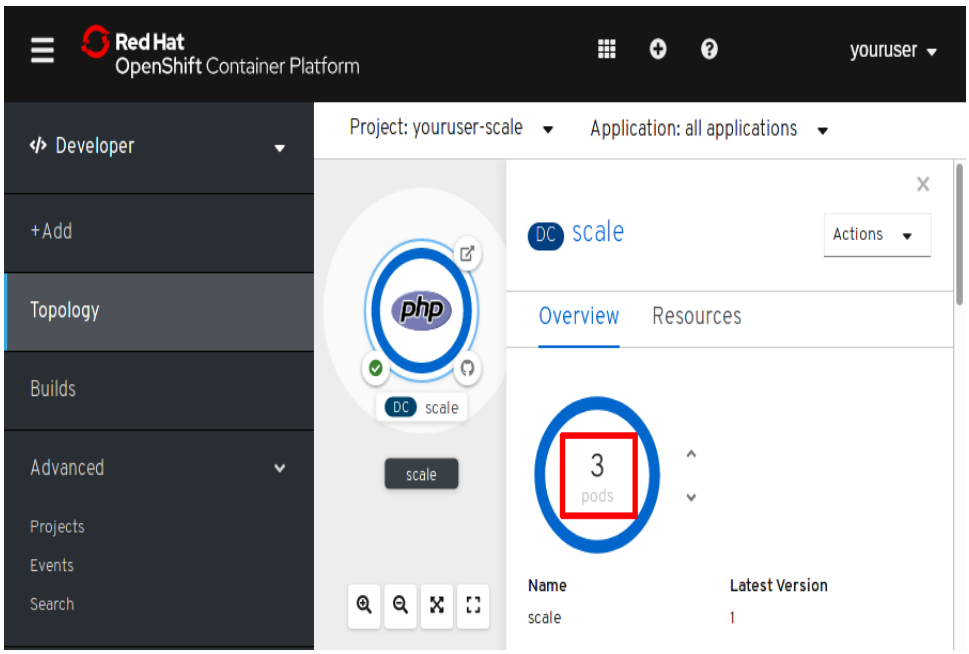


Figure 4.14: Autoscaling the application

11. Stop the `stress` application by scaling it down to zero pods. Notice that the `scale` application automatically scales down to one pod.
    1. From the `Topology` page, click the `stress` icon, and then click the `Overview` tab.
    2. Click the `Decrease the pod count` arrow to scale the application to zero pods.

3. Click the `PHP` icon, and then click the `Overview` tab. It may take up to 10 minutes for the number of pods in the `scale` application to scale down to one.

12. To clean up your work, delete the `youruser-scale` project. When you delete a project, OpenShift automatically removes all its resources.

    1. Click the **Advanced** → **Projects** menu to list all of your projects.

    2. Click the menu button at the end of the `youruser-scale` project row, and then click `Delete Project`. To confirm the deletion of the project, enter the project name in the confirmation window, and then click **Delete**.

    3. Log out of the web console. To log out, click your login name in the top right corner and then click **Log out**.

This concludes the guided exercise.

Chapter 5. Troubleshooting Applications in OpenShift

Troubleshooting and Fixing an Application

Objectives

After completing this section, you should be able to troubleshoot an application by observing OpenShift logs and events.

Troubleshooting an Application with the OpenShift Web Console

The Source-to-Image (S2I) process is a simple way to automatically build and deploy an application from its source code.

This process is often a convenient way to deploy applications quickly. However, if either the build or the deployment operation fail, then you must troubleshoot and resolve any issues before the S2I process can successfully execute.

To identify and troubleshoot the error, it is helpful to understand that the S2I process is composed of two major steps:

- Build step — Compiles source code, downloads library dependencies, and packages the application as a container image. Red Hat OpenShift Container Platform uses the `BuildConfig` resource for the build step.

- Deployment step — Starts a pod and makes the application available. If the build step succeeds, then this step executes.

  OpenShift uses the `DeploymentConfig` resource for the deployment step.

If you identify which step failed, then you will more easily identify why your application is not available as expected.

Inspecting Logs with the OpenShift Web Console

When an application fails, the web console helps developers identify the part of the deployment process in error. At each step, OpenShift maintains logs that developers can consult for troubleshooting.

You can consult the status of each application from the `Topology` page. The application icon provides a quick overview of its state.

A mark near the icon indicates the build status, as shown in the following screen capture. The icon on the left indicates a successful build. The icon on the right indicates a failed build.

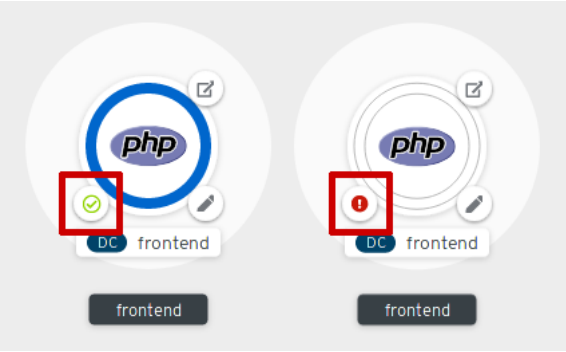


Figure 5.1: Successful and failed builds

If the build is successful, then OpenShift deploys the application. In the following screen capture, the icon on the left shows a successful deployment. The icon on the right shows a failed deployment.

Figure 5.2: Successful and failed deployments

To get more details and access the logs, click the icon and navigate to the `Resources` tab.

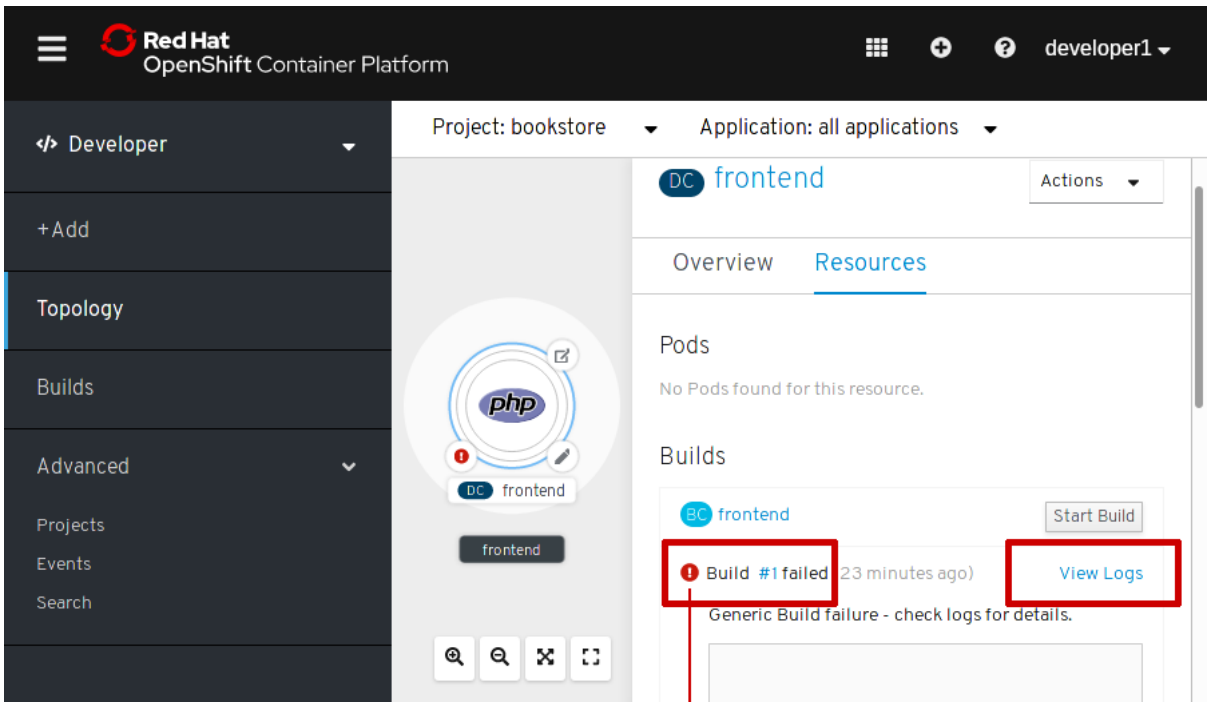


Figure 5.3: Details of a failed build

To identify any build issues, evaluate and analyze the logs for a build by clicking `View Logs`.

The detail page also provides access to the deployment logs.

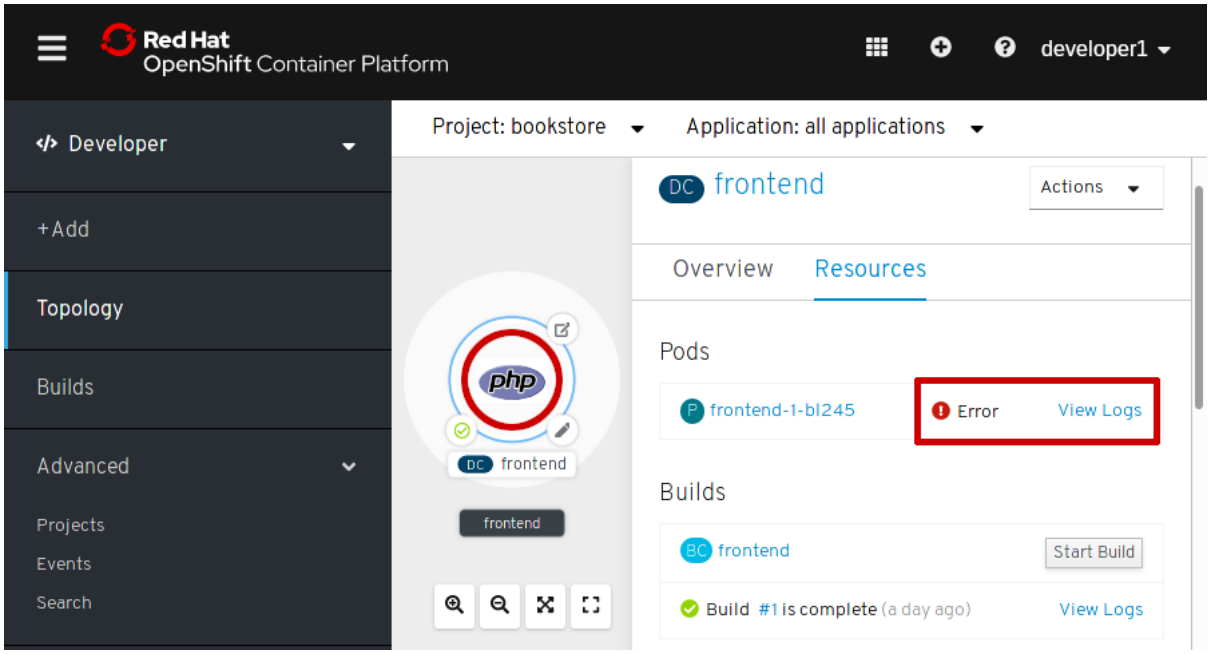


Figure 5.4: Details of a failed deployment

Accessing a Running Pod

Sometimes, OpenShift successfully builds and deploys the application, but due to bugs in the application source code the application does not behave as expected. In this situation, developers can access the logs of the running pod. For advanced troubleshooting, a console in the pod is available to run commands, and access the runtime environment inside the pod.

From the `Topology` page, click the pod name to access all of its parameters.

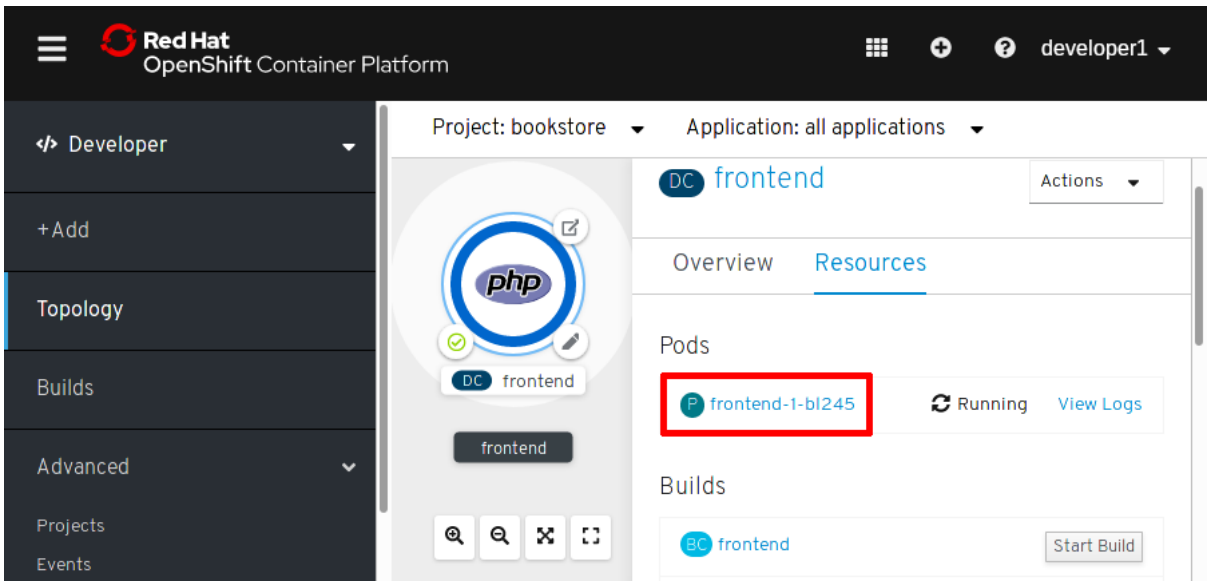


Figure 5.5: Accessing the pod details

The page that displays provides a tab to access the logs of the running pod. These logs show the output of the application that is running inside the pod.

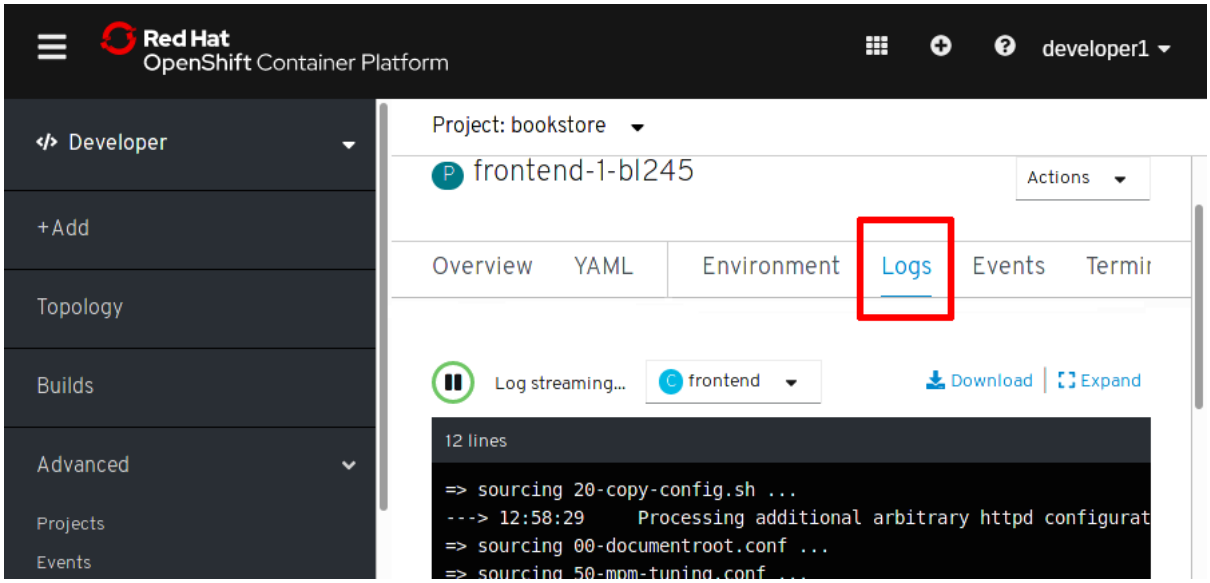


Figure 5.6: Consulting the pod logs

A console inside the container that is running the application is accessible on the `Terminal` tab. Expert developers can use this console for advanced troubleshooting.

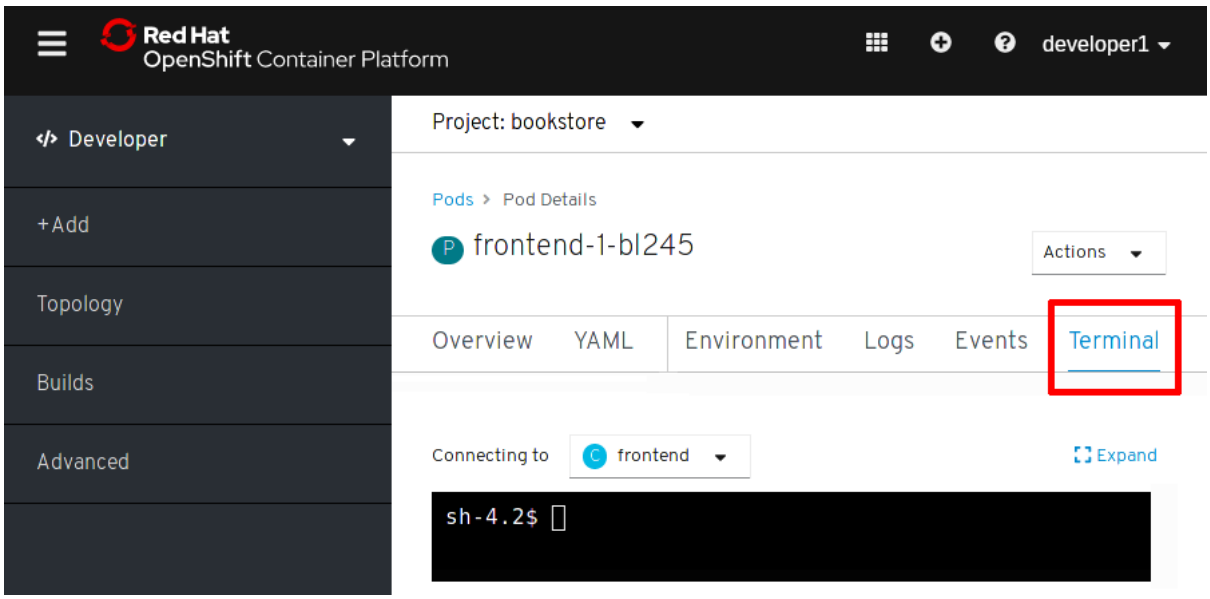


Figure 5.7: Accessing the pod terminal

Debugging an Application Running on OpenShift

For some languages, such as Node.js and Java, OpenShift provides features to attach a debugger to a running pod.

The debugger usually runs on a local workstation and connects to the application running on OpenShift through a remote debug port.

Remote debugging of an application is beyond the scope of this course. For more information, consult the following documentation:

- How to Debug Your Node.js Application on OpenShift with Chrome DevTools at
- Remote Debugging of Java Applications on OpenShift at

Redeploying an Application after Fixing an Issue

After the source code has been fixed, committed, and pushed to the Git repository, developers can use the OpenShift web console to start a new build. When the build is complete, OpenShift automatically redeploys the application.

Remember, you can also configure webhooks for OpenShift to automatically start a new build with each commit. This way, as soon as the developer commits a fix, OpenShift redeploys the application without delay or manual intervention.

Accessing OpenShift Events

To simplify troubleshooting, OpenShift provides a high-level logging and auditing facility called `events`. OpenShift events signal significant actions, such as starting or destroying a pod.

To read the OpenShift events in the web console, select the **Advanced** → **Events** menu.

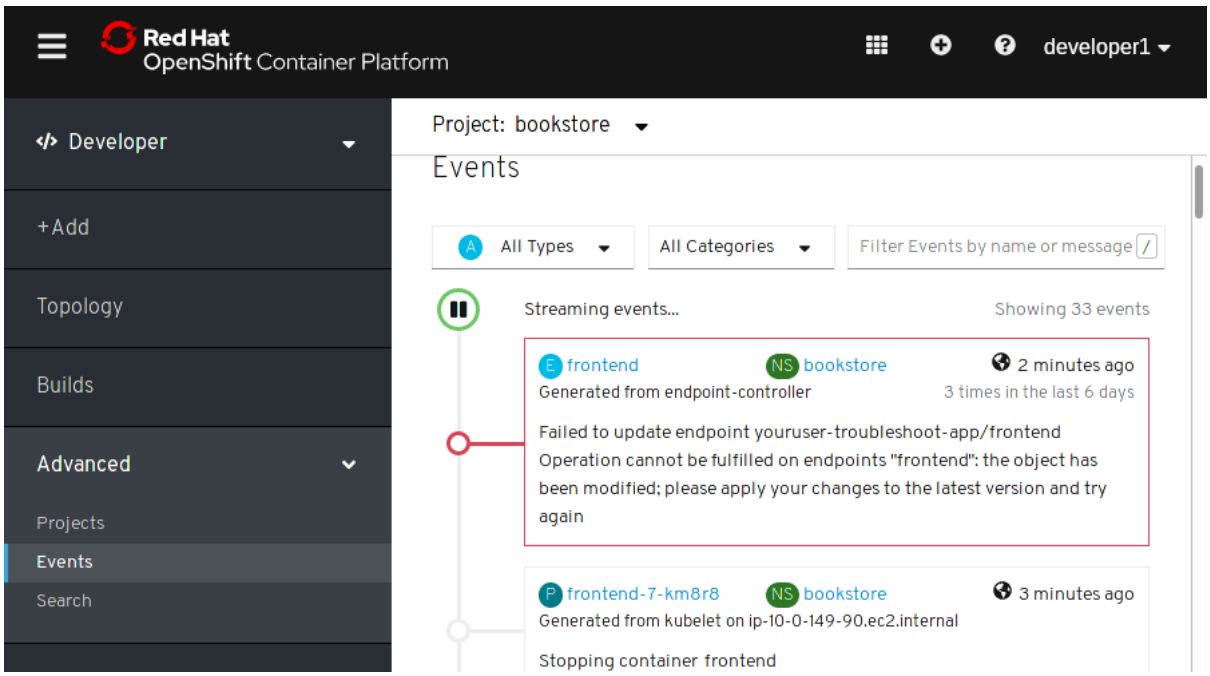


Figure 5.8: Listing the project events

From this page, developers have an overall view of the project events. Viewing the OpenShift events associated with a project is an important step to understanding what an application is doing.

Because OpenShift automatically refreshes the page with new events, developers can follow the progress of deployment in real time. In particular, when a build or an application deployment fails, there often are critical hints to the root cause of the problem provided in the project events. Sometimes, the provided information is enough for the developer to fix the application without having to inspect the more detailed OpenShift logs.

OpenShift uses events to report errors, such as when a build or a pod creation fails. It also notifies normal conditions through events, such as when OpenShift automatically scales an application by adding a new pod.

Troubleshooting Missing Environment Variables

Sometimes, the source code requires customization that is unavailable in containerized environments, such as database credentials, file system access, or message queue information. Those values are usually provided by leveraging environment variables within the pod. For example, a MySQL instance will require certain environment variables to be configured in the environment prior to starting. If these variables are missing, the service will fail to start. Developers using the S2I process might need to access or manage this information if a service or application behaves unexpectedly.

The OpenShift logs can indicate missing values or options that must be enabled, incorrect parameters or flags, or environment incompatibilities.

Troubleshooting Invalid Parameters

Multi-tiered applications typically require sharing certain parameters, such as login credentials for a back-end database. As a developer, it is important to ensure that the same values for parameters reach all pods in the application. For example, if a Node.js application runs in one pod, connected with another pod running a database, then make sure that the two pods use the same user name and password for the database. Usually, logs from the application pod provide a clear indication of these problems and how to solve them.

A good practice to centralize shared parameters is to store them in Configuration Map or in Secret resources. Remember that those resources can be injected into pods as environment variables, through the Deployment Configuration. Injecting the same Configuration Map or Secret resource into different pods ensures that not only the same environment variables are available, but also the same values.

Guided Exercise: Troubleshooting an Application

In this exercise, you will use OpenShift logs to troubleshoot an application.

Outcomes

You should be able to use the OpenShift web console to:

- Identify build and deployment issues.
- Analyze application source code and OpenShift logs to diagnose problems.
- Fix issues and redeploy applications.

To perform this exercise, ensure you have access to a running Red Hat OpenShift Container Platform cluster and ensure that Visual Studio Code (VS Code) and Git are installed on your system, and that you have cloned your GitHub `DO101-app` repository in VS Code.

Procedure 5.1. Steps

1. In VS Code, in the `DO101-apps` repository, create a new branch named `troubleshoot` for recording your work in this exercise. Push that branch to your GitHub repository.

    1. If you do not have VS Code open from a previous exercise, then open it.
    2. In the source control view in VS Code (**View → SCM**), ensure that the `DO101-apps` entry under `SOURCE CONTROL PROVIDERS` shows the `master` branch.

    NOTE

    If the Source Control view does not display the `SOURCE CONTROL PROVIDERS` heading, then right-click `SOURCE CONTROL` at the top of the Source Control view and select `Source Control Providers`.

    If necessary, click on the current branch and select `master` in the `Select a ref to checkout` window to switch to the `master` branch.

    WARNING

    Each exercise uses a unique branch. Always create a new branch using `master` as the base.

    3. Click **master** in the `DO101-apps` entry under `SOURCE CONTROL PROVIDERS`, and then select `Create new branch`. Type `troubleshoot` to name the branch.

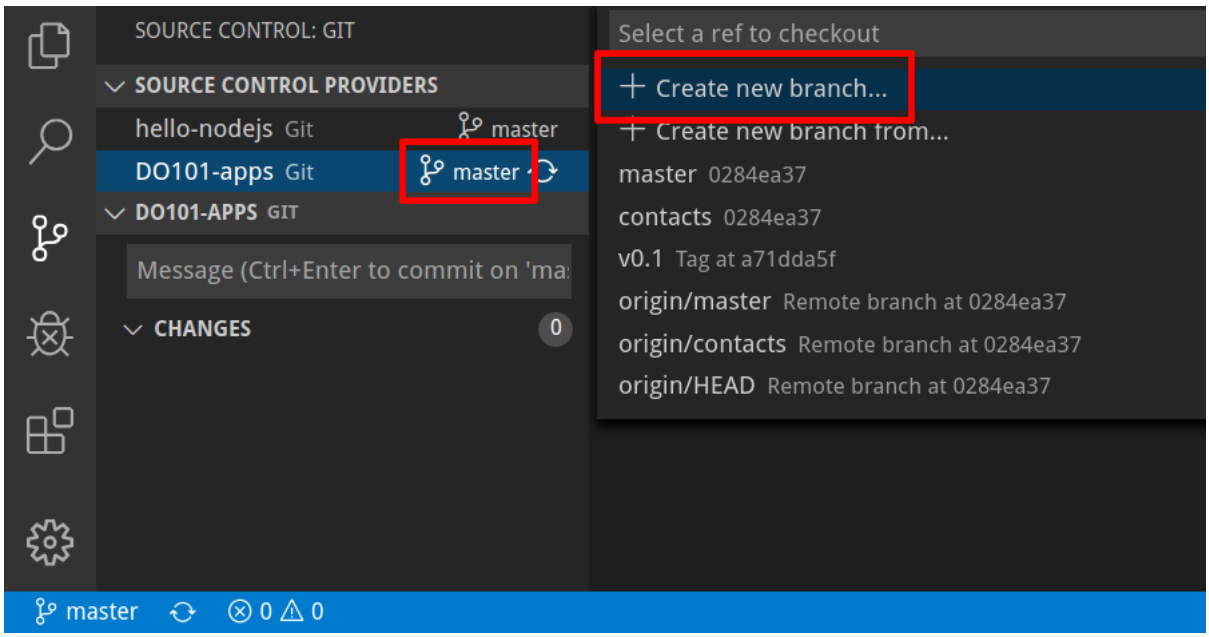


Figure 5.9: Creating a new branch with VS Code

4. To push the new branch to GitHub, click the `Publish Changes` icon for the `DO101-apps` entry. VS Code may ask for your GitHub login name and password.

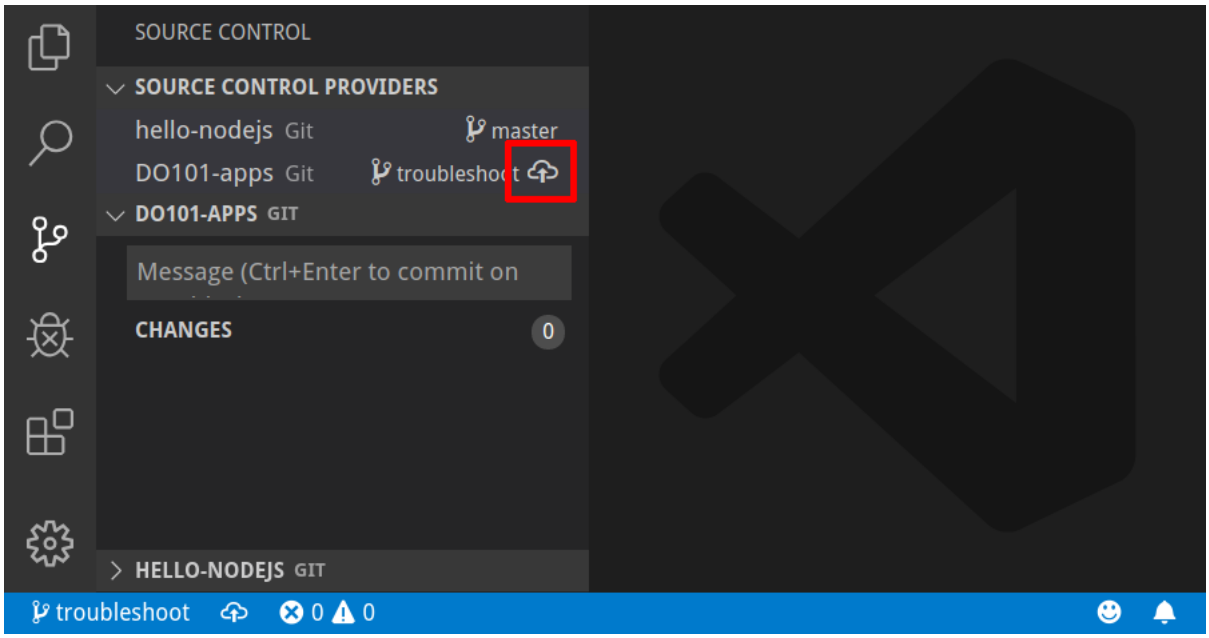


Figure 5.10: Pushing the new branch to GitHub

2. Access the OpenShift web console and create a new project named *`youruser`*`-troubleshoot-app`. Replace `youruser` with your user name.

    1. Log in to the OpenShift web console using your developer account.
    2. From the menu on the left, select the `Developer` perspective.
    3. On the **Advanced** → **Projects** menu, click **Create Project**.
    4. Enter *`youruser`*`-troubleshoot-app` in the `Name` field. Replace `youruser` with your user name. Leave the other fields empty and click **Create**.

3. The `contacts-troubleshoot` application you deploy in this exercise requires a PostgreSQL database. Deploy that database from the catalog.

    1. In the web console, click `Add`, and then click **From Catalog**.
    2. Click **Databases** → **Postgres**, and then click `PostgreSQL (Ephemeral)`. Click **Instantiate Template**, and then complete the form according to the following table:

        Table 5.1. PostgreSQL Parameters

Parameter	Value
`Database Service Name`	`postgresql`
`PostgreSQL Database Name`	`contactsdb`

        Leave the other fields with their default values and click **Create** to deploy the database. When deploying the database with these default parameters, OpenShift generates a random user name and password to access the database, and stores them in a secret resource.

    3. To inspect the secret resource that OpenShift creates to store the database authentication parameters, click the **Advanced** → **Search** menu. Click the `Service` menu and then type `secret`. Select `Secret` from the list.

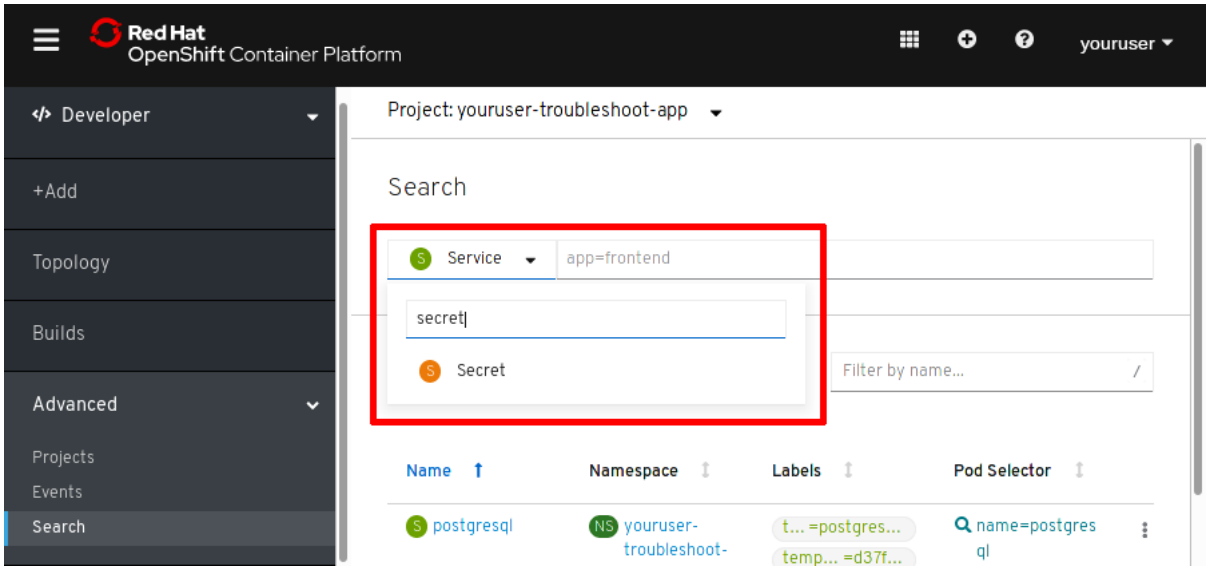


Figure 5.11: Searching for the secret resources

4. From the resulting list of secrets, click the `postgresql` secret.

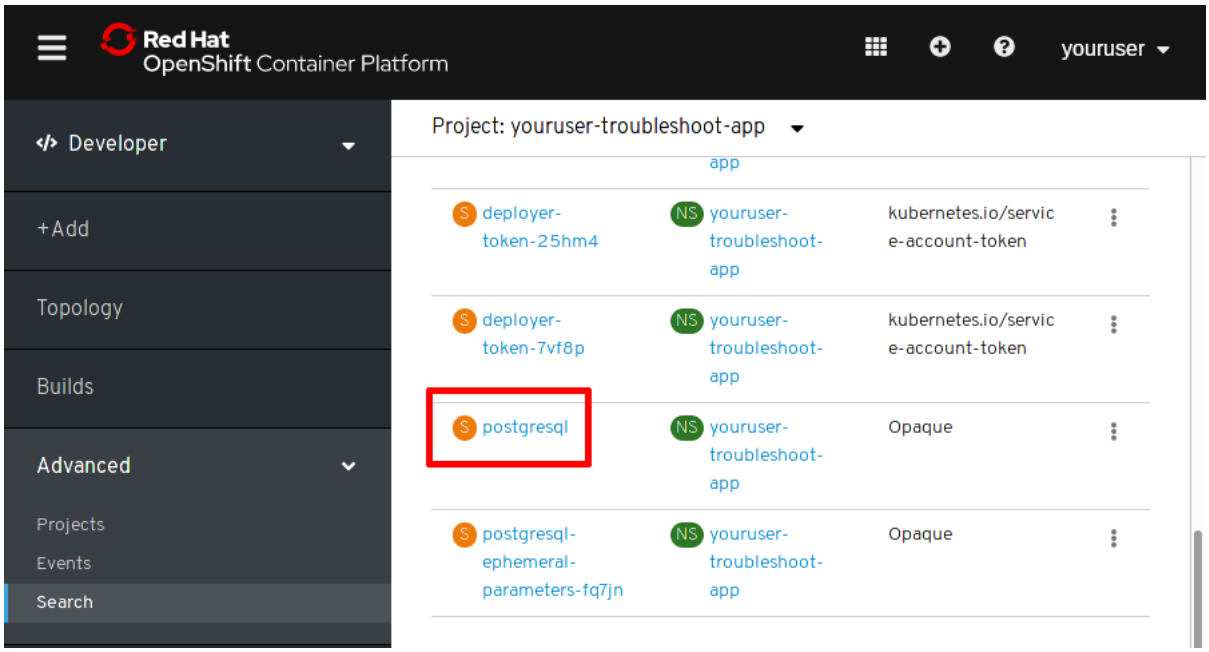


Figure 5.12: Accessing the secret resource

Under the `Data` section, notice that the secret stores the database name in the `database-name` entry, the password in the `database-password` entry, and the user name in the `database-user` entry.

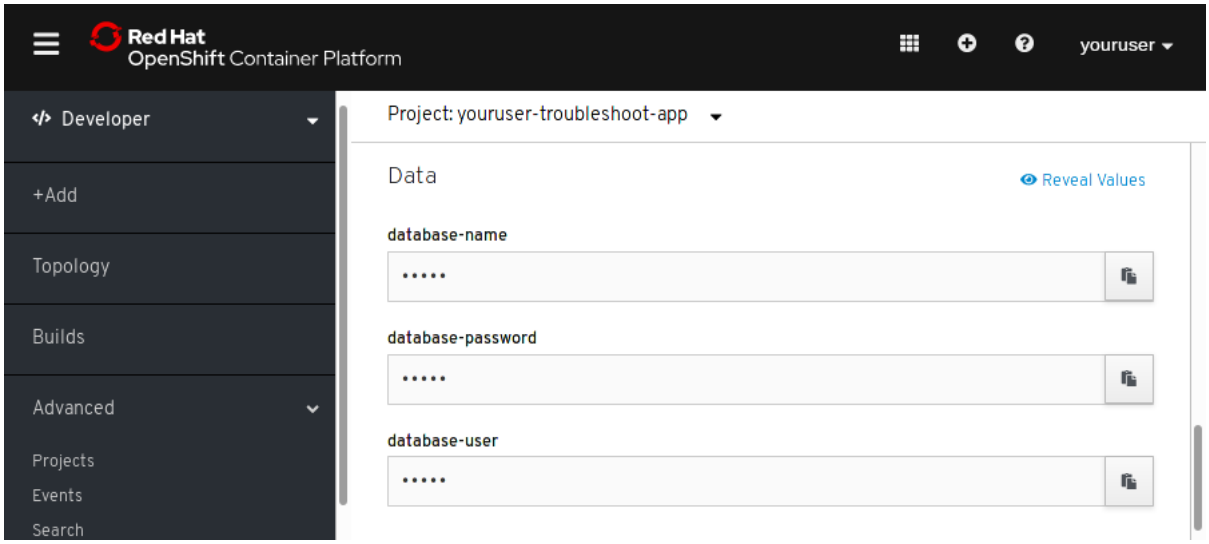


Figure 5.13: Data stored in the secret resource

4. Deploy the `contacts` JavaScript Node.js application. The code is in the `contacts-troubleshoot` subdirectory of your Git repository. Use the `troubleshoot` branch to deploy the application to OpenShift.

    1. On the `Add` tab, click **From Catalog**.

    2. Click **Languages** → **JavaScript**, and then click on the first option, `Node.js`. Click **Create Application** to enter the details of the application.

    3. Complete the form according to the following table. To access the Git parameters, click **Show Advanced Git Options**.

        Table 5.2. New Application Parameters

Parameter	Value
`Git Repo URL`	https://github.com/*`yourgituser`*/DO101-apps
`Git Reference`	`troubleshoot`
`Context Dir`	`/contacts-troubleshoot`
`Application Name`	`contacts`
`Name`	`contacts`

        To avoid unexpected errors when completing the following steps, review the values that you entered in the form before proceeding. Click **Create** to start the build and deployment processes.

5. OpenShift fails to build or deploy your application. Use the OpenShift web console to locate the error. Access the log to get more details.

    1. The web console should automatically show the `Topology` page. If this page is not displayed, then click the `Topology` tab.

    2. After a few minutes, the application displays an error state.

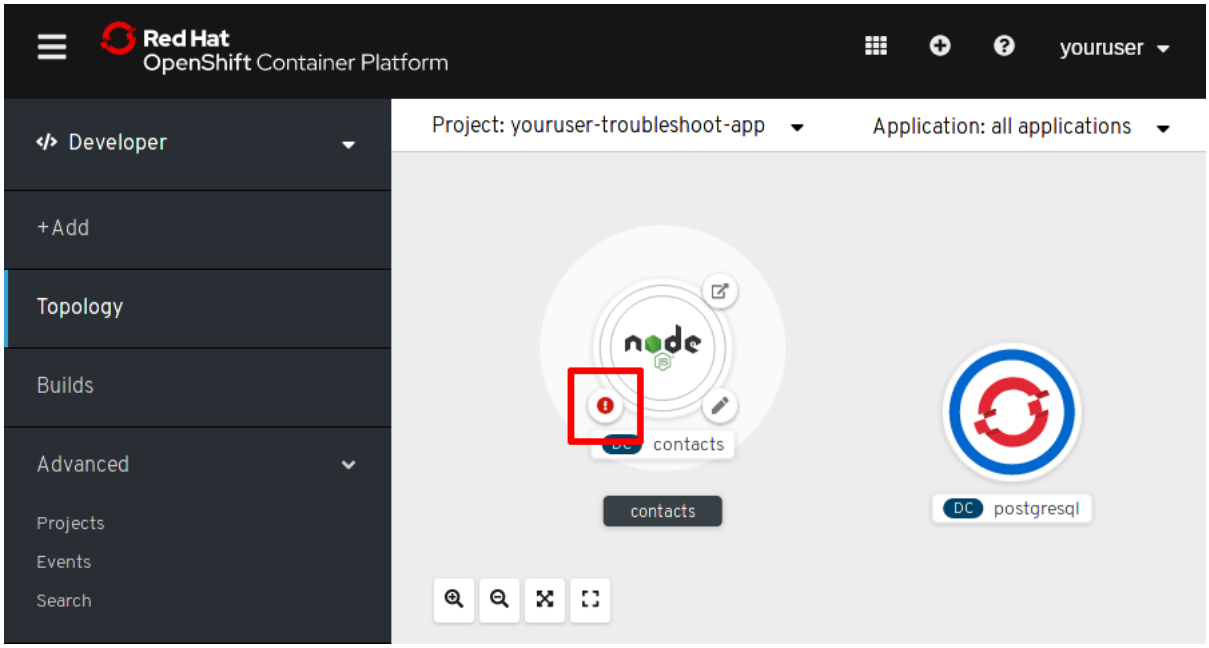


Figure 5.14: Error in the build step

Click the `Node.js` icon to access the application details (do not click the error check mark). Click the `Resources` tab to list the state of the resources.

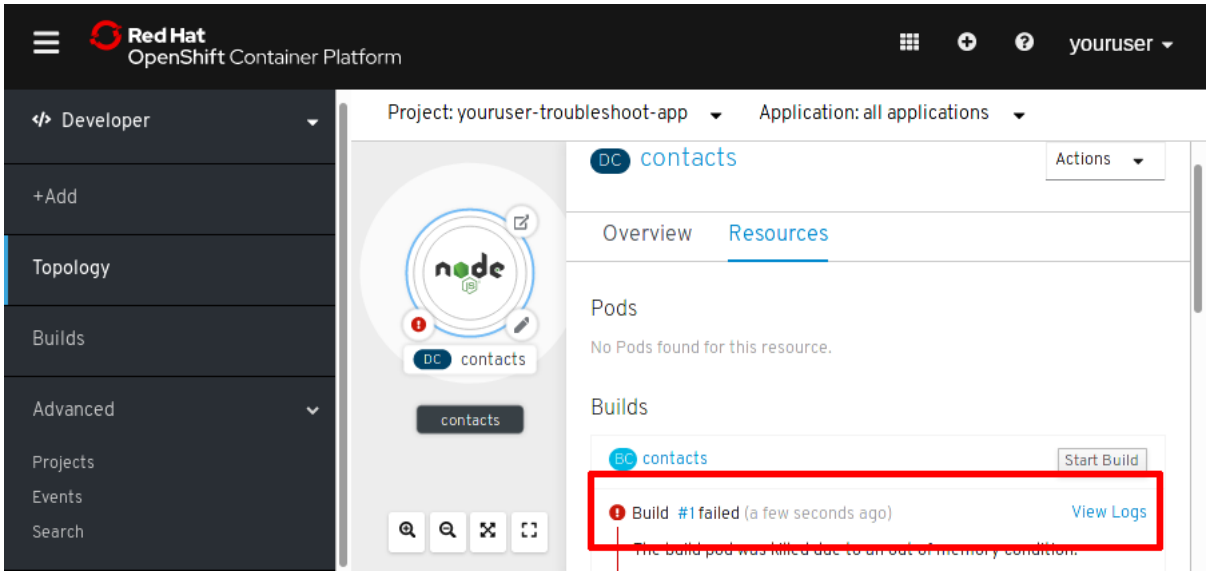


Figure 5.15: Accessing the error details

Notice that the build is in the error state. OpenShift is not able to deploy the application, and no pod is running.

3. Click `View Logs` to access the build log. Notice that the error occurs during the dependencies installation, and is caused by a syntax error in the `package.json` file.

6. Open the `DO101-apps/contacts-troubleshoot/package.json` file to identify and fix the issue. When finished, commit and push your change to GitHub.

    1. In the VS Code Explorer view (**View** → **Explorer**), open the `DO101-apps/contacts-troubleshoot/package.json` file.

    2. The `dependencies` keyword must be enclosed in double quotes. Notice that the opening double quote is missing.

    3. Add the missing double quote and then save the file.

    4. Commit your change. From the Source Control view (**View** → **SCM**), click the `+` icon for the `package.json` entry to stage the file for the next commit. Click in the `Message (press Ctrl+Enter to commit)` field. Type `Fix syntax error` in the message box.

Click the check mark icon to commit the change.

5. To push your change to the GitHub repository, click the `Synchronize Changes` icon for the `DO101-apps` entry, under `SOURCE CONTROL PROVIDERS`. If VS Code displays a message saying that this action will push and pull commits to and from `origin/troubleshoot`, then click **OK**. VS Code may ask for your GitHub login name and password.

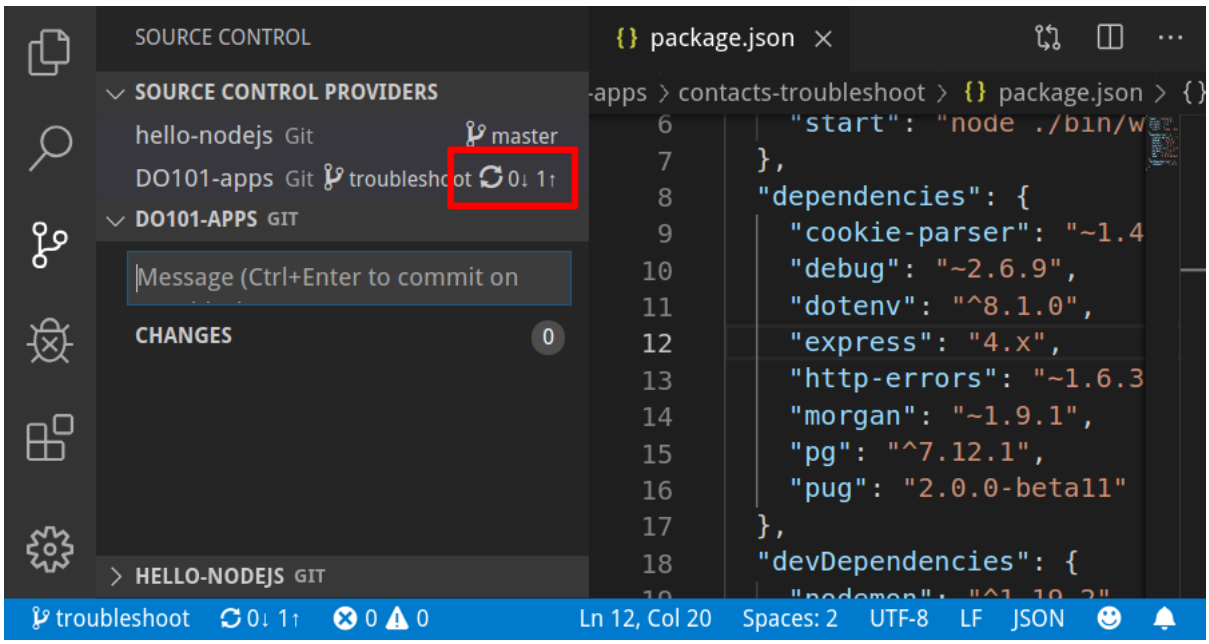


Figure 5.16: Pushing changes to GitHub

1. Use the OpenShift web console to initiate a new build and deployment. Click the `Topology` tab. Click the `Node.js` icon, and then click on the `Resources` tab. Click **Start Build** to initiate a build. For this new build, OpenShift retrieves your fixed version of the application from GitHub.

2. OpenShift fails again to build or deploy your application. Use the OpenShift web console to locate the error. Access the log to get more details.

    1. Wait for the build to finish and notice that the build is successful. The pod, however, is in the error state.

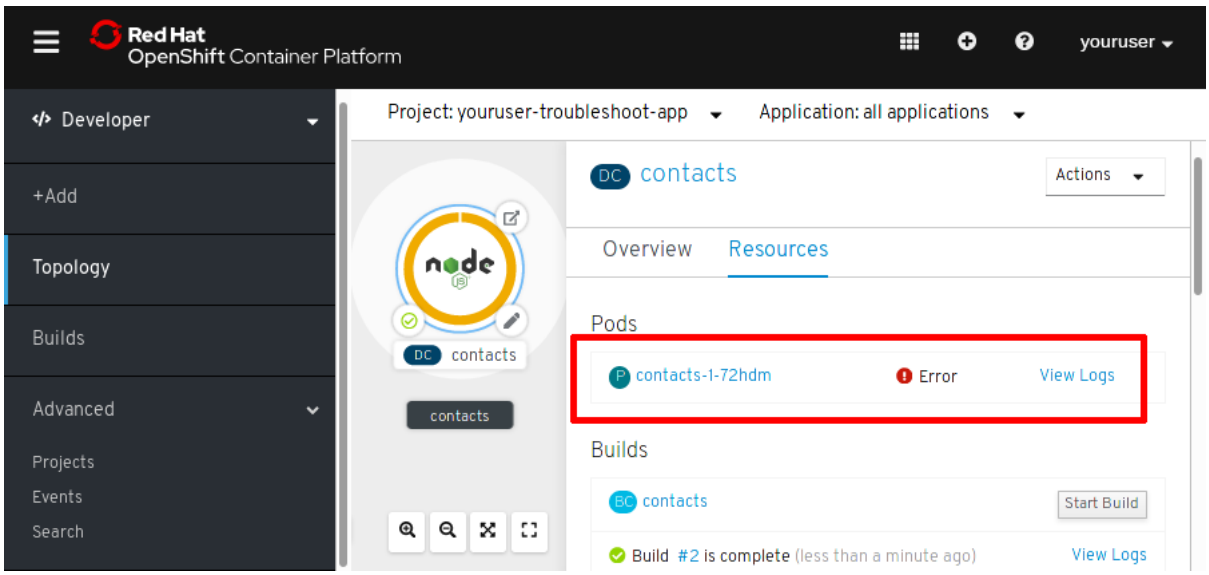


Figure 5.17: Deployment in error

The error status may alternate between `Error` and `CrashLoopBackOff`.

2. Click `View Logs` near the pod in error to access its log.

During startup, Node.js cannot find the program to run.

9. The Node.js application declares the program to start in the `package.json` file. Identify and fix the issue in the `DO101-apps/contacts-troubleshoot/package.json` file. When finished, commit and push your change to GitHub.

    1. Open the `DO101-apps/contacts-troubleshoot/package.json` file in VS Code.

    2. The file defines the program to run in the `scripts` section. The program is `./bin/www.js`.

    3. In VS Code, navigate to the `./bin/` directory (`DO101-apps/contacts-troubleshoot/bin/`). In this directory, the program file is `www`, not `www.js`.

    4. In `package.json`, replace `www.js` by `www`, and then save the file.

    5. Commit your change. From the Source Control view (**View → SCM**), click the `+` icon for the `package.json` entry to stage the file for the next commit. Click in the `Message (press Ctrl+Enter to commit)` field. Type `Fix wrong program name` in the message box. Click the check mark icon to commit the change.

    6. To push your change to the GitHub repository, click the `Synchronize Changes` icon for the `DO101-apps` entry, under `SOURCE CONTROL PROVIDERS`.

1. Use the OpenShift web console to initiate a new build and deployment. Click the `Topology` tab. Click the `Node.js` icon and then the `Resources` tab. Click **Start Build** to start a build.

2. Wait a few minutes for the build and deployment to complete. This time, the build is successful and a pod is running. Access the application and notice the error message.

    1. Click the **Open URL** button to access the application.

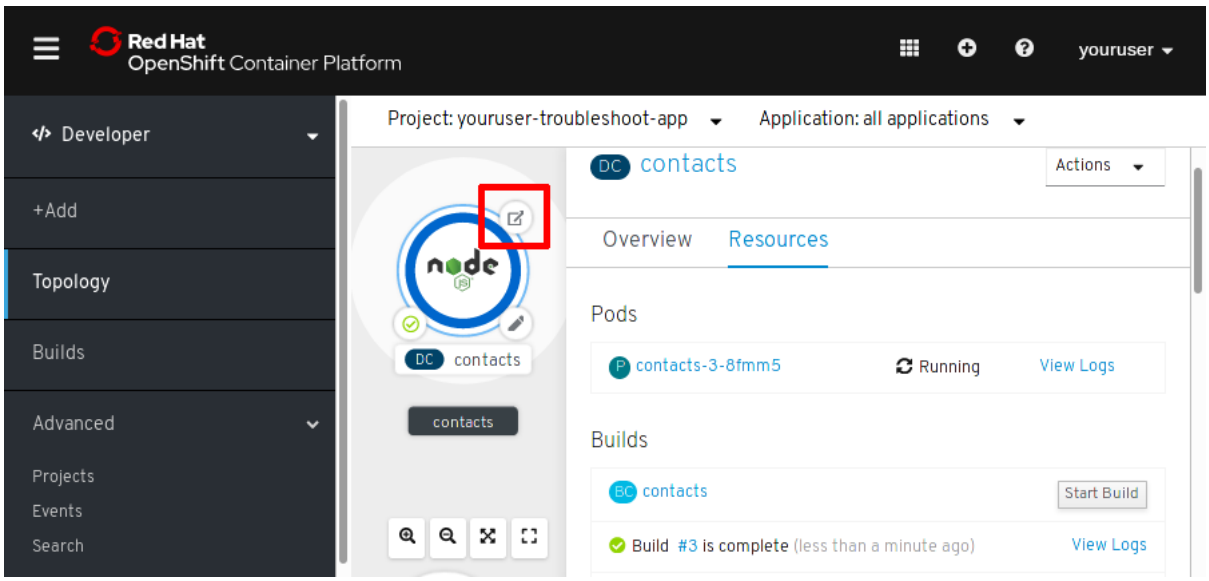


Figure 5.18: Accessing the contacts application

    2. The application displays an error message regarding the database connection.

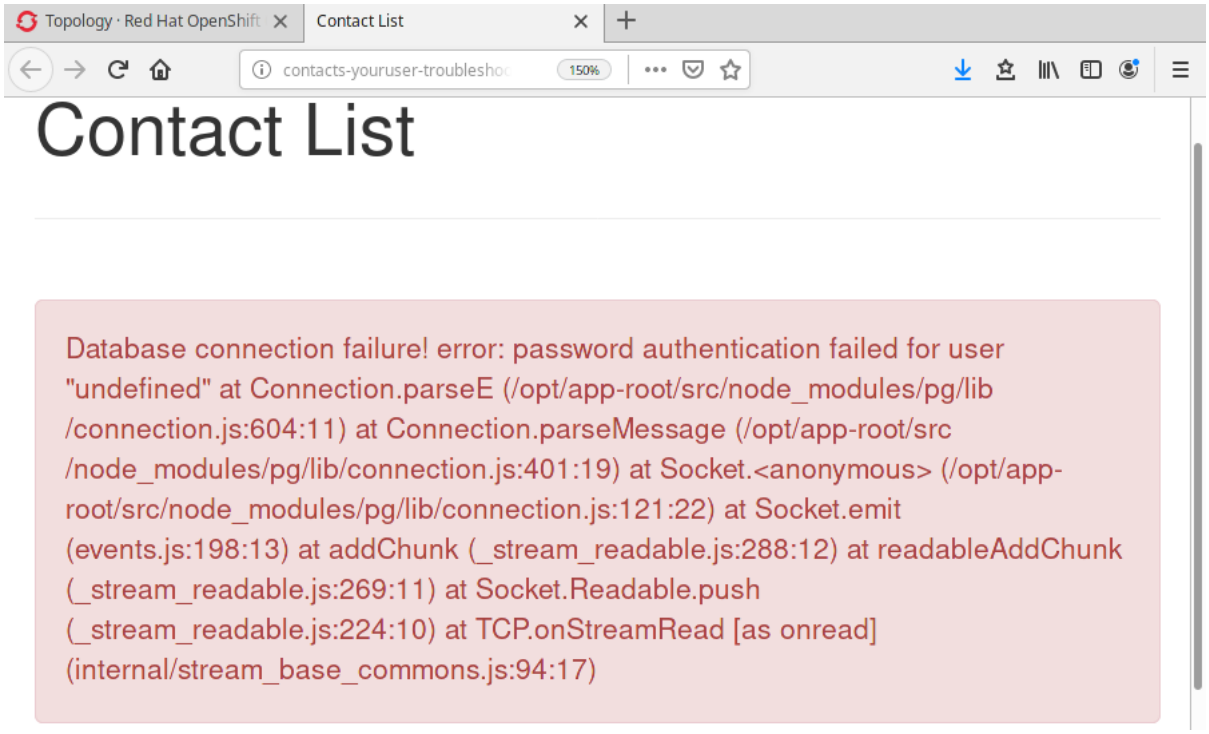


Figure 5.19: Database error

This message indicates that the provided user name for database authentication is not defined. When finished, close the browser tab.

3. The Node.js application defines the database parameters in the `DO101-apps/contacts-troubleshoot/db/config.js` file. Open this file with VS Code.

   The application retrieves the database connection parameters from the `database-user`, `database-password`, and `database-name` environment variables, but those variables are not available in the running pod.

13. Use the OpenShift web console to associate the `postgresql` secret to the `contacts` deployment configuration resource. With this association, OpenShift defines each entry in the secret as environment variables in the pod. When done, confirm that the application is finally working as expected.

    1. Click the `contacts` deployment configuration link.

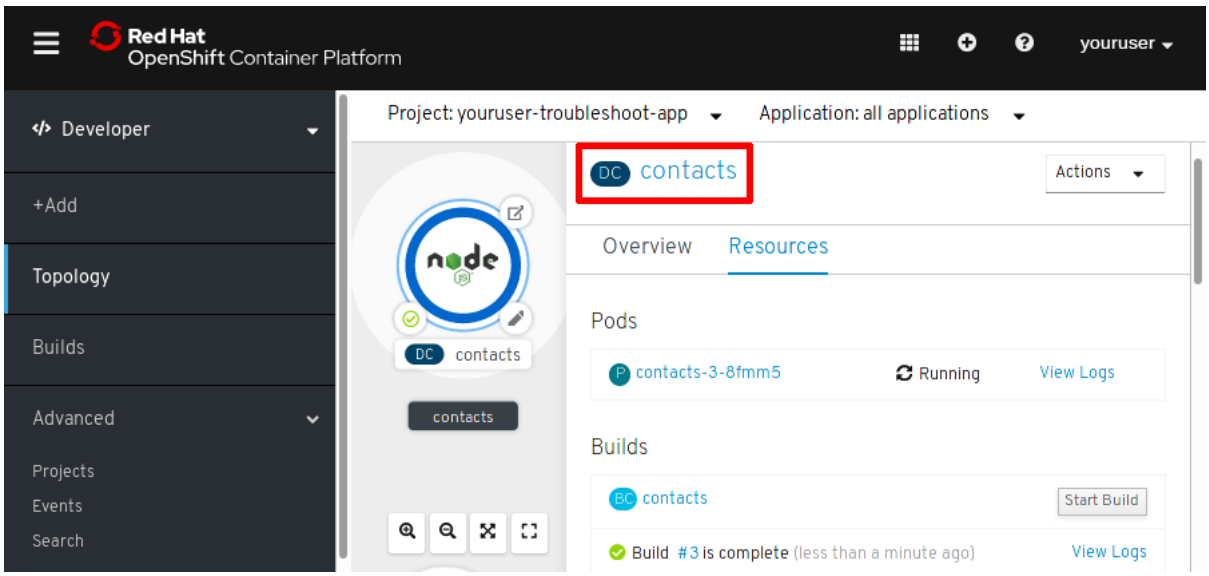


Figure 5.20: Accessing the deployment configuration details

2. Click the `Environment` tab. In the `All values from existing config maps or secrets (envFrom)` section, set the `Config Map/Secret` field to `postgresql`, and then click **Save**.

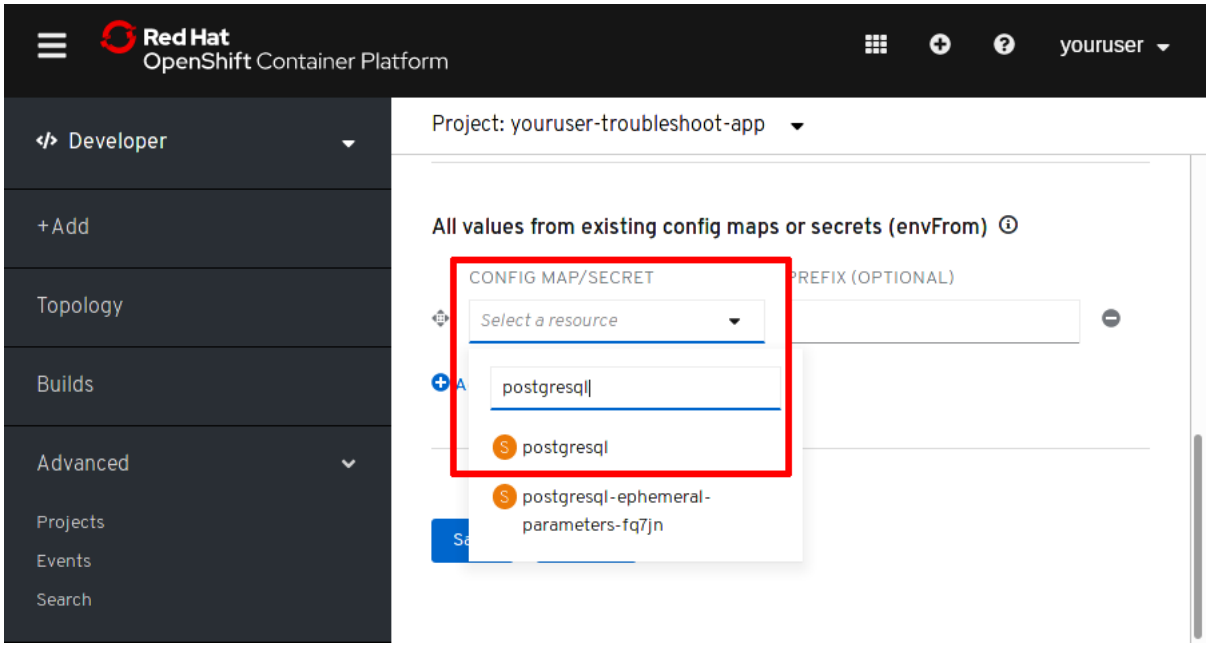


Figure 5.21: Associating the secret with the deployment configuration

When you update the deployment configuration resource, OpenShift automatically redeploys the application.

3. Wait a minute for the pod to redeploy and then test the application. To test the application, click `Topology`, and then click `Node.js` **Open URL**. The application displays the rows retrieved from the database.

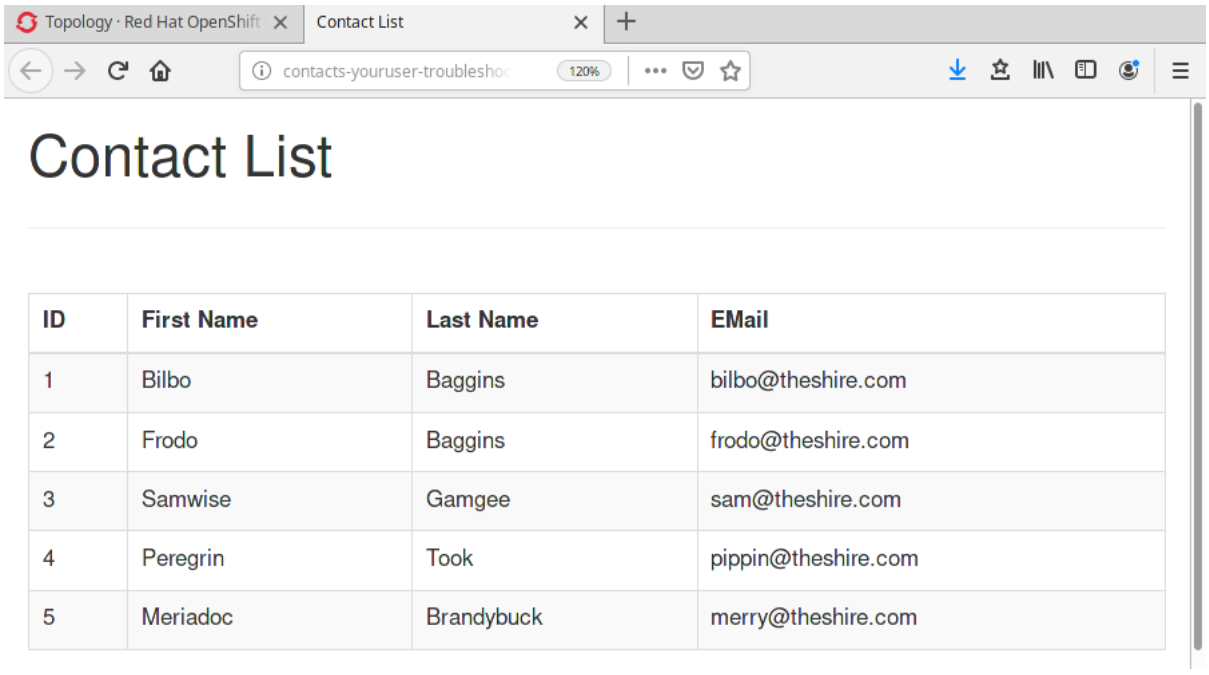


Figure 5.22: The contacts application

When done, close the browser tab.

14. To clean up your work, delete the `youruser-troubleshoot-app` project. When you delete a project, OpenShift automatically removes all its resources.

    1. Use the **Advanced** → **Projects** menu to list all of your projects.

2. Click the menu button at the end of the `youruser-troubleshoot-app` project row, and then click `Delete Project`. To confirm the deletion of the project, enter the project name in the confirmation window, and then click **Delete**.

3. Log out of the web console. To log out, click your login name in the top right corner and then click **Log out**.

This concludes the guided exercise.