



# **Red Hat**

## **Training and Certification**

### **Student Workbook**

Kubernetes 1.22 DO100A

### **Foundations of Red Hat Cloud-native Development**

Edition 2







## Join a community dedicated to learning open source

The Red Hat® Learning Community is a collaborative platform for users to accelerate open source skill adoption while working with Red Hat products and experts.



**Network** with tens of thousands of community members



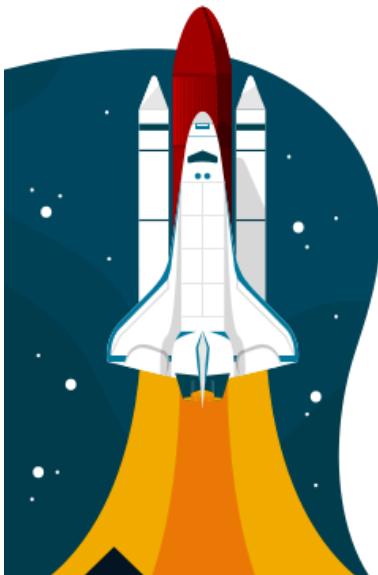
**Engage** in thousands of active conversations and posts



**Join and interact** with hundreds of certified training instructors



**Unlock** badges as you participate and accomplish new goals



This knowledge-sharing platform creates a space where learners can connect, ask questions, and collaborate with other open source practitioners.

**Access** free Red Hat training videos

**Discover** the latest Red Hat Training and Certification news

**Connect** with your instructor - and your classmates - before, after, and during your training course.

**Join** peers as you explore Red Hat products

Join the conversation [learn.redhat.com](https://learn.redhat.com)



Copyright © 2020 Red Hat, Inc. Red Hat, Red Hat Enterprise Linux, the Red Hat logo, and Ansible are trademarks or registered trademarks of Red Hat, Inc. or its subsidiaries in the United States and other countries.



# **Foundations of Red Hat Cloud-native Development**



**Kubernetes 1.22 DO100A**  
**Foundations of Red Hat Cloud-native Development**  
**Edition 2 r00000000**  
**Publication date 20220000**

Authors: Enol Alvarez de Prado, Manuel Aude Morales,  
Guy Bianco IV, Marek Czernek, Natalie Lind, Michael Phillips,  
Eduardo Ramirez Ronco, Alejandro Serna Borja, Jordi Sola Alaball  
Course Architects: Zachary Guterman, Fernando Lozano  
DevOps Engineer: Richard Allred  
Editor: Sam Ffrench

Copyright © 2022 Red Hat, Inc.

The contents of this course and all its modules and related materials, including handouts to audience members, are  
Copyright © 2022 Red Hat, Inc.

No part of this publication may be stored in a retrieval system, transmitted or reproduced in any way, including, but not limited to, photocopy, photograph, magnetic, electronic or other record, without the prior written permission of Red Hat, Inc.

This instructional program, including all material provided herein, is supplied without any guarantees from Red Hat, Inc. Red Hat, Inc. assumes no liability for damages or legal action arising from the use or misuse of contents or details contained herein.

If you believe Red Hat training materials are being used, copied, or otherwise improperly distributed, please send email to [training@redhat.com](mailto:training@redhat.com) or phone toll-free (USA) +1 (866) 626-2994 or +1 (919) 754-3700.

Red Hat, Red Hat Enterprise Linux, the Red Hat logo, JBoss, OpenShift, Fedora, Hibernate, Ansible, CloudForms, RHCA, RHCE, RHCSA, Ceph, and Gluster are trademarks or registered trademarks of Red Hat, Inc. or its subsidiaries in the United States and other countries.

Linux® is the registered trademark of Linus Torvalds in the United States and other countries.

Java® is a registered trademark of Oracle American, Inc. and/or its affiliates.

XFS® is a registered trademark of Hewlett Packard Enterprise Development LP or its subsidiaries in the United States and/or other countries.

MySQL® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js® is a trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack word mark and the Square O Design, together or apart, are trademarks or registered trademarks of OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. Red Hat, Inc. is not affiliated with, endorsed by, or sponsored by the OpenStack Foundation or the OpenStack community.

All other trademarks are the property of their respective owners.

<b>Document Conventions</b>	<b>ix</b>
	ix
<b>Introduction</b>	<b>xi</b>
Foundations of Red Hat Cloud-native Development .....	xi
Orientation to the Classroom Environment .....	xii
<b>1. Introducing Containers and Kubernetes</b>	<b>1</b>
Introducing Containers and Container Engines .....	2
Introducing Kubernetes and Container Orchestration .....	5
Summary .....	7
<b>2. Running Containerized Applications</b>	<b>9</b>
Contrasting Kubernetes Distributions .....	10
Guided Exercise: Contrasting Kubernetes Distributions .....	14
Introducing Kubectl .....	27
Guided Exercise: Connecting Kubectl To Your Cluster .....	32
Running and Interacting with Your First Application .....	39
Guided Exercise: Running and Interacting with Your First Application .....	42
Summary .....	46



# Document Conventions

---

This section describes various conventions and practices used throughout all Red Hat Training courses.

## Admonitions

Red Hat Training courses use the following admonitions:



### References

These describe where to find external documentation relevant to a subject.



### Note

These are tips, shortcuts, or alternative approaches to the task at hand. Ignoring a note should have no negative consequences, but you might miss out on something that makes your life easier.



### Important

These provide details of information that is easily missed: configuration changes that only apply to the current session, or services that need restarting before an update will apply. Ignoring these admonitions will not cause data loss, but may cause irritation and frustration.



### Warning

These should not be ignored. Ignoring these admonitions will most likely cause data loss.

## Inclusive Language

Red Hat Training is currently reviewing its use of language in various areas to help remove any potentially offensive terms. This is an ongoing process and requires alignment with the products and services covered in Red Hat Training courses. Red Hat appreciates your patience during this process.



# Introduction

## Foundations of Red Hat Cloud-native Development

Foundations of Red Hat Cloud-native Development (DO100a) is designed for IT professionals without previous cloud application deployment experience to learn basic Kubernetes skills. This course is a part of a three-course specialization. In this specialization, you will run, deploy, and test containerized applications with zero-downtime releases.

### Course Objectives

- Introduction to deploying Cloud Native Applications on a Kubernetes cluster.

### Audience

- Developers who wish to deploy cloud applications.
- Administrators who are new to Kubernetes.
- Architects who are considering using container technologies in software architectures.
- Site Reliability Engineers who are considering using Kubernetes.

### Prerequisites

- Experience running commands and navigating the file system on Linux, MacOS, and Windows.
- Experience with web application development and corresponding technologies.

# Orientation to the Classroom Environment

---

DO100a is a **Bring Your Developer Workstation (BYDW)** class, where you use your own internet-enabled system to access the shared OpenShift cluster. The following operating systems are supported:

- Red Hat Enterprise Linux 8 or Fedora Workstation 34 or later
- Ubuntu 20.04 LTS or later
- Microsoft Windows 10
- macOS 10.15 or later

## BYDW System Requirements

Attribute	Minimum Requirements	Recommended
CPU	1.6 GHz or faster processor	Multi-core i7 or equivalent
Memory	8 GB	16 GB or more
Disk	10 GB free space HD	10 GB or more free space SSD
Display Resolution	1024x768	1920x1080 or greater

You must have permissions to install additional software on your system. Some hands-on learning activities in DO100a provide instructions to install the following programs:

- Git 2.18 or later (Git Bash for Windows systems)
- The Kubernetes CLI (`kubectl`) v1.21 or later
- Minikube (Optional)

You might already have these tools installed. If you do not, then wait until the day you start this course to ensure a consistent course experience.



### Important

Hands-on activities also require that you have a personal account on GitHub and a public, free internet service.

## BYDW Systems Support Considerations

Depending on your system, you might see differences between your command-line shell and the examples given in this course.

### **Red Hat Enterprise Linux or Fedora Workstation**

- If you use Bash as the default shell, then your prompt might match the [user@host ~]\$ prompt used in the course examples, although different Bash configurations can produce different results.
- If you use another shell, such as zsh, then your prompt format will differ from the prompt used in the course examples.
- When performing the exercises, interpret the [user@host ~]\$ prompt used in the course as a representation of your system prompt.
- All the commands from the exercises should be functional.

### **Ubuntu**

- You might find differences in the prompt format.
- In Ubuntu, your prompt might be similar to user@host:~\$.
- When performing the exercises, interpret the [user@host ~]\$ prompt used in the course as a representation of your Ubuntu prompt.
- All the commands from the exercises should be functional.

### **macOS**

- You might find differences in the prompt format.
- In macOS, your prompt might be similar to host:~ user\$.
- When performing the exercises, interpret the [user@host ~]\$ prompt used in the course as a representation of your macOS prompt.
- All the commands from the exercises should be functional.
- You might need to grant execution permissions to the installed runtimes.

### **Microsoft Windows**

- Windows does not support Bash natively. Instead, you must use PowerShell.
- In Windows PowerShell, your prompt should be similar to PS C:\Users\user>.
- When performing the exercises, interpret the [user@host ~]\$ Bash prompt as a representation of your Windows PowerShell prompt.
- For some commands, Bash syntax and PowerShell syntax are similar, such as cd or ls. You can also use the slash character (/) in file system paths.
- For other commands, the course provides help to transform Bash commands into equivalent PowerShell commands.
- This course only provides support for Windows PowerShell.
- The Windows firewall might ask for additional permissions in certain exercises.

## **Executing Long Commands**

This course breaks long commands into multiple lines by using the backslash character (\), for example:

```
[user@host ~]$ long command \
    argument_1 \
    argument_2
```

The preceding example works on the Linux and macOS systems.

On Windows, use the backtick character (`), for example:

```
[user@host ~]$ long command ` \
    argument_1 ` \
    argument_2
```

Alternatively, you can type commands in one line on all systems, such as:

```
[user@host ~]$ long command argument_1 argument_2
```

## Chapter 1

# Introducing Containers and Kubernetes

### Goal

Describing containers and container orchestration with Kubernetes.

### Objectives

- Get an overview of what containers are, how they improve the software life cycle, and samples of different container runtimes.
- Recognize Kubernetes as a container orchestration tool.

### Sections

- Introducing Containers and Container Engines
- Introducing Kubernetes and Container Orchestration

# Introducing Containers and Container Engines

## Objectives

After completing this section, you should be able to get an overview of what containers are, how they improve the software life cycle, and samples of different container runtimes.

## Traditional Applications

Traditional software applications typically depend on other libraries, configuration files, or services that are provided by the runtime environment. The application runtime environment is a physical host or virtual machine (VM) and application dependencies are installed as part of the host.

For example, consider a Python application that requires access to a common shared library that implements the TLS protocol. A system administrator installs the required package that provides the shared library before installing the Python application.

The major drawback to a traditional deployment is that the application's dependencies are mixed with the runtime environment. Because of this, an application might break when any updates or patches are applied to the base operating system (OS).

For example, an update to the TLS shared library removes TLS 1.0 as a supported protocol. Updating the library breaks a Python application that is strictly dependent on TLS 1.0. The system administrator must downgrade the library to keep the application running, but this prevents other applications from using the benefits of the updated package.

To alleviate potential breakages, a company might maintain a full test suite to guarantee OS updates do not affect applications.

Furthermore, applications must be stopped before updating the associated dependencies. To minimize downtime, organizations use complex systems to provide high availability. Maintaining multiple applications on a single host often becomes cumbersome, and any update has the potential to break one of the applications.

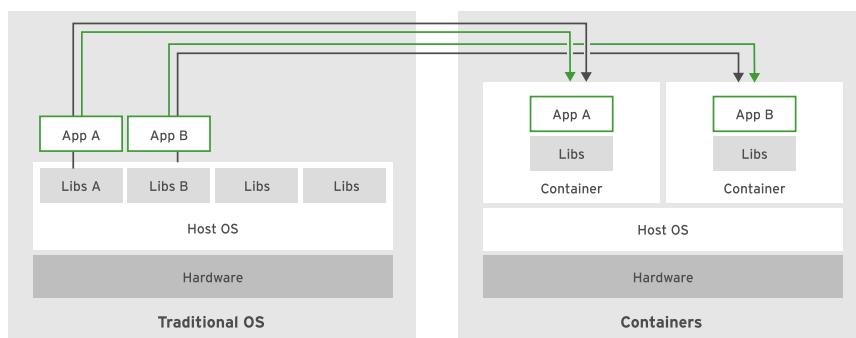


Figure 1.1: Container versus operating system differences

## Containerized Applications

Deploying applications using containers is an alternative to the traditional methods. A container is a set of one or more processes that are isolated from the rest of the system.

Containers provide many of the same benefits as virtual machines, such as security, storage, and network isolation. Containers require fewer hardware resources and are quick to start and terminate. They also isolate the libraries and the runtime resources, such as CPU and storage, and minimize the impact of OS updates.

Beyond improving efficiency, elasticity, and reusability of hosted applications, container usage improves application portability. The Open Container Initiative (OCI) provides a set of industry standards that define a container runtime specification and a container image specification. The image specification defines the format for the bundle of files and metadata that form a container image. When you build a container image compliant with the OCI standard, you can use any OCI-compliant container engine to execute the contained application.

There are many container engines available to manage and execute containers, including Rocket, Drawbridge, LXC, Docker, and Podman.

The following are other major advantages to using containers:

### **Low hardware footprint**

Containers use OS-internal features to create an isolated environment where resources are managed using OS facilities such as namespaces and cgroups. This approach minimizes the amount of CPU and memory overhead compared to a virtual machine hypervisor. Running an application in a VM isolates the application from the running environment, but it requires a heavy service layer to achieve the level of isolation provided by containers.

### **Environment isolation**

Containers work in a closed environment where changes made to the host OS or other applications do not affect the container. Because the libraries needed by a container are self-contained, the application can run without disruption. For example, each application can exist in its own container with its own set of libraries. An update made to one container does not affect other containers.

### **Quick deployment**

Containers deploy quickly because there is no need to install the entire underlying operating system. Normally, to support isolation, a host requires a new OS installation, and any update might require a full OS restart. A container restart does not require stopping any services on the host OS.

### **Multiple environment deployment**

In a traditional deployment scenario using a single host, any environment differences could break the application. By using containers, all application dependencies and environment settings are encapsulated in the container image.

### **Reusability**

The same container can be reused without the need to set up a full OS. For example, the same database container that provides a production database service can be used by each developer to create a development database during application development. By using containers, there is no longer a need to maintain separate production and development database servers. A single container image is used to create instances of the database service.

Often, a software application with all of its dependent services (databases, messaging, file systems) are made to run in a single container. This can lead to the same problems associated with traditional software deployments to virtual machines or physical hosts. In these instances, a multicontainer deployment might be more suitable.

Furthermore, containers are an ideal approach when using microservices for application development. Each service is encapsulated in a lightweight and reliable container environment

that can be deployed to a production or development environment. The collection of containerized services required by an application can be hosted on a single machine, removing the need to manage a machine for each service.

In contrast, many applications are not well suited for a containerized environment. For example, applications accessing low-level hardware information, such as memory, file systems, and devices may be unreliable due to container limitations.



## References

### **Home - Open Containers Initiative**

<https://www.opencontainers.org/>

# Introducing Kubernetes and Container Orchestration

---

## Objectives

After completing this section, you should be able to recognize Kubernetes as a container orchestration tool.

## Limitations of Containers

Containers provide an easy way to package and run services. As the number of containers managed by an organization grows, the manual work of managing them grows disproportionately.

When using containers in a production environment, enterprises often require the following capabilities:

- Easy communication between a large number of services
- Resource limits on applications
- Ability to respond to application usage spikes by increasing or decreasing replicas
- Gradual rollout of a new release to different users

Enterprises often require a container orchestration technology because container runtimes, by themselves, do not adequately address the above requirements.

## Kubernetes Overview

Kubernetes is a container orchestration platform that simplifies the deployment, management, and scaling of containerized applications.

A pod is the smallest manageable unit in Kubernetes, and consists of at least one container. Kubernetes also uses pods to manage the containers within and their resource limits as a single unit.

## Kubernetes Features

Kubernetes offers the following features on top of a container engine:

### Service discovery and load balancing

Kubernetes enables inter-service communication by assigning a single DNS entry to each set of containers. This way, the requesting service only needs to know the target's DNS name, allowing the cluster to change the container's location and IP address. This permits load-balancing requests across the pool of container replicas.

### Horizontal scaling

Applications can scale up and down manually or automatically with a configuration set, by using either the command-line interface or the web UI.

### Self-healing

Kubernetes can use user-defined health checks to monitor containers to restart and reschedule them in case of failure.

## Automated rollout

Kubernetes can gradually release updates to your application's containers while checking their status. If something goes wrong during the rollout, Kubernetes can roll back to the previous version of the application.

## Secrets and configuration maps

You can manage the configuration settings and secrets of your applications without rebuilding containers. Configuration maps store these settings in a way that decouples them from the pods and containers using them. Application secrets can include any configuration setting that must be kept private, such as user names, passwords, and service endpoints.

## Operators

Operators are packaged Kubernetes applications that bring the knowledge of application lifecycles into the Kubernetes cluster. Applications packaged as Operators use the Kubernetes API to update the cluster's state by reacting to changes in the application state.



### References

#### Production-Grade Container Orchestration - Kubernetes

<https://kubernetes.io/>

# Summary

---

In this chapter, you learned:

- Applications running in containers are decoupled from the host operating system's libraries.
- Among other features, container orchestration platforms provide tooling to automate the deployment and management of application containers.



## Chapter 2

# Running Containerized Applications

### Goal

Spin-up your first application in Kubernetes.

### Objectives

- See the differences between several Kubernetes implementations, and understand how to prepare different Kubernetes flavours for this course.
- Review the basic usage of the kubectl command and understand how to connect to your Kubernetes cluster by using the CLI.
- Execute a pre-built application in your Kubernetes cluster and review the resources related to the process.

### Sections

- Contrasting Kubernetes Distributions (and Guided Exercise)
- Introducing Kubectl (and Guided Exercise)
- Running and Interacting with Your First Application (and Guided Exercise)

# Contrasting Kubernetes Distributions

---

## Objectives

After completing this section, you should be able to see the differences between several Kubernetes implementations, and understand how to prepare different Kubernetes flavours for this course.

## Kubernetes Distributions

Kubernetes has historically been a general solution for container management and orchestration. With this versatility, Kubernetes can solve the same problems in different ways depending on needs and opinions. Because of this, Kubernetes has evolved into different opinionated distributions based on:

- The target size of the cluster: From small single-node clusters to large-scale clusters of hundreds of thousands of nodes.
- The location of the nodes: Either locally on the developer workstation, on premises (such as a private data center), on the cloud, or a hybrid solution of those two.
- The ownership of the management: Self-managed clusters versus Kubernetes-as-a-Service.

The following table shows a classification for some of the most popular Kubernetes distributions:

	<b>Big Scale</b>	<b>Small Scale</b>
<b>Self-Managed - Local</b>		<b>minikube,</b> <b>CodeReady Containers,</b> <b>Microk8s,</b> <b>Docker Kubernetes</b>
<b>Self-Managed - On Premises / Hybrid</b>	Red Hat OpenShift, VMWare Tanzu, Rancher	
<b>Kubernetes-as-a-Service - On Cloud</b>	OpenShift Dedicated, Google Container Engine, Amazon EKS	<b>Developer Sandbox</b>



### Note

This course supports **minikube** (version 1.20.0) for local development and **Developer Sandbox** for remote development. Instructions and exercises have been tested in the following operating systems:

- Fedora Linux 33 and 34
- Red Hat Enterprise Linux 8
- Windows 10 Pro and Enterprise
- MacOS Big Sur (11.3.1)

Visit the links in the *References* section for a comprehensive list of Kubernetes certified distributions.

## Kubernetes Extensions

Kubernetes is highly extendable for adding more services to the platform. Each distribution provides different approaches (or none) for adding capabilities to Kubernetes:

### DNS

DNS allows internal name resolution inside the cluster, so pods and services can refer to others by using a fixed name.

Both `minikube` and OpenShift include a `CoreDNS` controller that provides this feature.

### Dashboard

The dashboard provides a graphical user interface to Kubernetes.

`minikube` provides an add-on and utility commands for using the general-purpose `Dashboard` open source application. OpenShift includes the `Console`, a dedicated application that integrates most of the Kubernetes extensions provided by OpenShift.

### Ingress

The ingress extension allows traffic to get into the cluster network, redirecting requests from managed domains to services and pods. Ingress enables services and applications inside the cluster to expose ports and features to the public.

`minikube` uses an ingress add-on based on the `ingress nginx` controller.



#### Note

You must install the ingress add-on for `minikube` for some exercises. Refer to *Guided Exercise: Contrasting Kubernetes Distributions* for instructions.

OpenShift deploys an ingress controller based on `HAProxy` and controlled by a `Ingress Operator`. OpenShift also introduces the `route` resource. A `route` extends the `ingress` manifest to ease controlling ingress traffic.

### Storage

The storage extension allows pods to use persistent storage and nodes to distribute and share the storage contents.

OpenShift bases its storage strategy on `Red Hat OpenShift Data Foundation`, a storage provider supporting multiple storage strategies across nodes and hybrid clouds. `minikube` provides out-of-the-box storage by using the underlying storage infrastructure (either local the file system or the virtual machine's file-system). This feature is provided by the `storage-provisioner` add-on. `minikube` also provides a `storage-provisioner-gluster` add-on that allows Kubernetes to use `Gluster` as shared persistent storage.

### Authentication and authorization.

Kubernetes embeds a certificate authority (CA) and considers anyone that presents a certificate issued by that CA as a valid user.

`minikube` provides the user with an administrator `minikube` account, so users have total control over the cluster.

Different OpenShift implementations differ on authentication features, but all of them agree on avoiding the use of administration accounts. Developer Sandbox provides limited access to the user, restricting them to the `username-dev` and `username-stage` namespaces.

Authorization in Kubernetes is role based. Authorized users or administrators can assign predefined roles to users on each resource. For example, administrators can grant read-only access to auditor users to application namespaces.

## Operators

Operators are a core feature of most Kubernetes distributions. Operators allow automated management of applications and Kubernetes services, by using a declarative approach.

`minikube` requires the `olm` add-on to be installed to enable operators in the cluster.

OpenShift distributions enable operators by default, despite the fact that Kubernetes-as-a-Service platforms usually restrict user-deployed operators. Developer Sandbox does not allow users to install operators, but comes with the `RHOAS-Operator` and the `Service Binding Operator` by default.

### Comparison summary of Kubernetes features

	<code>minikube</code>	<code>Developer Sandbox</code>
<b>DNS</b>		
<b>Dashboard</b>	Dashboard add-on	OpenShift Console
<b>Ingress</b>	NGINX Ingress add-on	Operator-controlled HAProxy
<b>Storage</b>	Local or Gluster add-ons	Red Hat OpenShift Data Foundation,
<b>Authentication</b>	Administrator <code>minikube</code> user	Developer used restricted to 2 namespaces
<b>Operators</b>	OLM add-on. No restrictions	Limited to RHOAS and Service Binding



## References

### minikube documentation

<https://minikube.sigs.k8s.io/docs/>

### Developer Sandbox

<https://developers.redhat.com/developer-sandbox>

### CNCF Cloud Native Interactive Landscape - Certified Kubernetes Platforms

<https://landscape.cncf.io/card-mode?category=certified-kubernetes-distribution,certified-kubernetes-hosted,certified-kubernetes-installer&grouping=category>

### Certificate Signing Requests

<https://kubernetes.io/docs/reference/access-authn-authz/certificate-signing-requests/#normal-user>

## ► Guided Exercise

# Contrasting Kubernetes Distributions

In this exercise you will prepare your development environment to use a local or remote Kubernetes instance.

### Outcomes

You should be able to:

- Install a local Kubernetes instance by using `minikube` on Linux, macOS or Windows.
- Register for using a remote Kubernetes instance by using Developer Sandbox for Red Hat OpenShift.

## Instructions



### Note

Installing a local Kubernetes cluster requires administrative privileges in your development workstation. If you do not have administrative privileges then jump directly to *Guided Exercise: Contrasting Kubernetes Distributions* to use a remote Kubernetes cluster.

Deploying a fully developed, multi-node Kubernetes cluster typically requires significant time and compute resources. With `minikube`, you can quickly deploy a local Kubernetes cluster, allowing you to focus on learning Kubernetes operations and application development.

`minikube` is an open source utility that allows you to quickly deploy a local Kubernetes cluster on your personal computer. By using virtualization technologies, `minikube` creates a *virtual machine* (VM) that contains a single-node Kubernetes cluster. VMs are virtual computers and each VM is allocated its own system resources and operating system.

The latest `minikube` releases also allow you to create your cluster by using containers instead of virtual machines. Nevertheless, this solution is still not mature, and it is not supported for this course.

`minikube` is compatible with Linux, macOS, and Windows.

To install `minikube` on your system, you will need:

- An Internet connection
- At least 2 GB of free memory
- 2 CPUs or more
- At least 20 GB of free disk space
- A locally installed hypervisor (using a container runtime is not supported in this course)

Before installing `minikube`, a hypervisor technology must be installed or enabled on your local system. A **hypervisor** is software that creates and manages virtual machines (VMs) on a shared physical hardware system. The hypervisor pools and isolates hardware resources for VMs, allowing many VMs to run on a shared physical hardware system, such as a server.

## ► 1. Using minikube in Linux-based systems

In this course we support Fedora Linux 33 and 34, as well as Red Hat Enterprise Linux 8.

### 1.1. Installing a Hypervisor on Linux

The preferred hypervisor for Linux systems is `kvm2` [<https://minikube.sigs.k8s.io/docs/drivers/kvm2/>]. `minikube` communicates with the hypervisor using the `libvirt` virtualization API libraries.



#### Note

Prefix the following commands with `sudo` if you are running a user without administrative privileges.

Use your system package manager to install the complete set of virtualization libraries:

```
[root@host ~]# dnf install @virtualization
```

or select the minimum set of required libraries

```
[root@host ~]# dnf install qemu-kvm libvirt libvirt-python libguestfs-tools
virt-install
```

Start the `libvirtd` service:

```
[root@host ~]# systemctl start libvirtd
...output omitted...
[root@host ~]# systemctl enable libvirtd
...output omitted...
```

### 1.2. Installing minikube on Linux

There are three alternatives to install `minikube` in a Linux system:

- If your system contains a package manager or software manager including `minikube`, then use it and verify the version matches the minimum requirements.

```
[root@host ~]# dnf install minikube
```

- If the repositories for your package manager do not include an appropriate version for `minikube`, then go to <https://github.com/kubernetes/minikube/releases> and download the latest release matching your operating system.

For Debian-based systems, download the file `minikube_VERSION_[amd64|arm64|armhf|ppc64el|s390x].deb` and install it using the `dpkg` command:

```
[root@host ~]# dpkg -i FILE
```

For RPM-based distributions, download the file `minikube-VERSION.[aarch64|armv7hl|ppc64le|s390x|x86_64].rpm` and install it using the `rpm` command:

```
[root@host ~]# rpm -Uvh FILE
```

- Alternatively, download the binary `minikube-linux-[amd64|arm|arm64]` file and install using the `install` command:

```
[root@host ~]# install FILE /usr/local/bin/minikube
```

### 1.3. Starting Your minikube Cluster on Linux

To initialize your `minikube` cluster, use the `minikube start` command.

```
[root@host ~]# minikube start --driver=kvm2
# minikube v1.20.0 on Fedora 33
# Using the kvm2 driver based on user configuration
# Starting control plane node minikube in cluster minikube
# Creating kvm2 VM (CPUs=4, Memory=16384MB, Disk=20000MB) ...
# Preparing Kubernetes v1.20.2 on Docker 20.10.6 ...
  • Generating certificates and keys ...
  • Booting up control plane ...
  • Configuring RBAC rules ...
# Verifying Kubernetes components...
  • Using image gcr.io/k8s-minikube/storage-provisioner:v5
# Enabled addons: storage-provisioner, default-storageclass
# Done! kubectl is now configured to use "minikube" cluster and "default"
namespace by default
```



#### Note

To set the default driver, run the command `minikube config set driver DRIVER`.

## ► 2. Using minikube on macOS systems

This course supports macOS Big Sur (version 11.3.1) and later minor updates.

### 2.1. Installing a Hypervisor on macOS

Despite Docker being the preferred installation method in macOS, current `minikube` versions do not support all the features required for this course when running the `docker` driver (to be precise, `ingress` add-on is not yet supported). For this reason, we are using the `Oracle VM VirtualBox` hypervisor for running the virtual machines needed to support `minikube` on macOS.

Oracle VM VirtualBox is a free and open source hypervisor available for macOS systems. To install Oracle VM VirtualBox:

1. Download the latest version of VirtualBox for OS X hosts from <https://virtualbox.org/wiki/Downloads>
2. Open the downloaded `dmg` file and follow the onscreen instructions to complete the installation.



#### Note

Network connectivity might be temporarily lost while VirtualBox installs virtual network adapters. A system reboot can also be required after a successful installation.

Alternatively, if the `brew` command is available in your system, then you can install `VirtualBox` using the `brew install` command.

```
host:~ root# brew install virtualbox
```

## 2.2. Installing minikube on macOS

To install `minikube` on macOS, download from <https://github.com/kubernetes/minikube/releases> the appropriate file for your architecture: `minikube-darwin-amd64` or `minikube-darwin-arm64`.

Then open a terminal window, change to the directory where you downloaded the installer and use the `install` command with administrator privileges to run the installer. Make sure you install `minikube` in a folder in your system path, such as `/usr/local/bin/minikube`

```
host:~ root# cd Downloads
host:~ root# sudo install _FILE /usr/local/bin/minikube
```

This places the `minikube` executable in `/usr/local/bin/minikube`.

Apple notarizing features forbid running files downloaded from the internet unless authorized. If you try to run the `minikube` command then you will get a "minikube" cannot be opened because the developer cannot be verified message, and the application will be terminated.

To authorize the `minikube` application, use the `xattr` command with administrative privileges with the following options:

```
host:~ root# sudo xattr -r -d com.apple.quarantine /usr/local/bin/minikube
```

Verify that now you can execute the `minikube` command:

```
host:~ root# minikube version
minikube version: v1.20.0
commit: c61663e942ec43b20e8e70839dcca52e44cd85ae
```

Your output can differ, but must show the available version and the commit it is based on.

## 2.3. Starting Your minikube Cluster on macOS

To initialize your `minikube` cluster, use the `minikube start` command.

```
host:~ root# minikube start --driver=virtualbox
# minikube v1.20.0 on Darwin 11.3.1
# Using the virtualbox driver based on user configuration
# Starting control plane node minikube in cluster minikube
# Creating virtualbox VM (CPUs=4, Memory=16384MB, Disk=20000MB) ...
# Preparing Kubernetes v1.20.2 on Docker 20.10.6 ...
  • Generating certificates and keys ...
  • Booting up control plane ...
  • Configuring RBAC rules ...
# Verifying Kubernetes components...
```

```
▪ Using image gcr.io/k8s-minikube/storage-provisioner:v5
# Enabled addons: storage-provisioner, default-storageclass
# Done! kubectl is now configured to use "minikube" cluster and "default"
namespace by default
```



### Note

To set the default driver, run the command `minikube config set driver DRIVER`.

## ► 3. Using minikube on Windows

In this course we support Windows 10 Pro and Enterprise versions.

### 3.1. Installing a Hypervisor on Windows

There are several hypervisors available for Windows systems, including **Oracle VM VirtualBox** and **Microsoft Hyper-V**.



### Warning

System driver conflicts might occur if more than one hypervisor is installed or enabled. Do not install or enable more than one hypervisor on your system.

#### Oracle VM VirtualBox installation

Oracle VM VirtualBox is a free, open source hypervisor. As the original driver for `minikube`, Oracle VM VirtualBox provides the most stability for Microsoft Windows 10 users.

1. Download the latest version of VirtualBox for Windows Hosts from <https://virtualbox.org/wiki/Downloads>
2. Open the downloaded VirtualBox executable and follow the onscreen instructions to complete the installation.



### Note

Network connectivity might be temporarily lost while VirtualBox installs virtual network adapters. A system reboot can also be required after a successful installation.

#### Enabling Microsoft Hyper-V

Microsoft Hyper-V is a built-in hypervisor available on modern 64-bit versions of Microsoft Windows 10 Pro, Enterprise, and Education editions. Enabling Microsoft Hyper-V can be accomplished through PowerShell or by adjusting system Settings.

Refer to Microsoft Hyper-V documentation for system and hardware requirements.

- Via PowerShell
  - Launch a PowerShell console as Administrator.
  - In the console window, run the following command:

```
PS C:\> Enable-WindowsOptionalFeature -Online -FeatureName Microsoft-Hyper-V -All
```

- Restart your system when prompted to finish the installation process.
- Via Settings
  - In the search box on the taskbar, type **Programs and Features**, and select it from the search results.
  - Select **Turn Windows features on or off** from the list of options under Control Panel Home.
  - Select **Hyper-V** and click **OK** to begin the installation.
  - Restart your system when prompted to finish the installation process.

With Microsoft Hyper-V successfully enabled, create an external virtual switch to grant `minikube` access to your physical network.

1. Open a PowerShell console as Administrator.
2. Determine the name of the network adapter, such as Wi-Fi or Ethernet, to use by running `Get-NetAdapter`.
3. Create an external virtual switch named **minikube** that uses the selected network adapter and allows the management operating system to share the adapter:

```
PS C:\> New-VMSwitch -name minikube -NetAdapterName <AdapterName>
-AllowManagementOS $true
```

4. Restart your system to clear out potential routing problems.

### 3.2. Installing minikube on Windows

With a hypervisor installed, your system is now ready to begin the `minikube` installation process.

1. Download the stand-alone `minikube` Windows installer from <https://github.com/kubernetes/minikube/releases>
2. Open the downloaded `minikube-installer.exe` to begin the guided installation process.



#### Note

If you executed the `minikube-installer.exe` installer from a terminal window, close the terminal and open a new one before you start using `minikube`.

### 3.3. Starting Your minikube Cluster on Windows

To initialize your `minikube` cluster, use the `minikube start` command.

To use the Microsoft Hyper-V driver, provide the `--driver=hyperv` option to `minikube`:

```
PS C:\> minikube start --driver=hyperv
# minikube v1.20.0 on Microsoft Windows 10 Enterprise 10.0.18363 Build 18363
# Using the hyperv driver based on user configuration
# Starting control plane node minikube in cluster minikube
# Creating hyperv VM (CPUs=4, Memory=16384MB, Disk=20000MB) ...
# Preparing Kubernetes v1.20.2 on Docker 20.10.6 ...
  • Generating certificates and keys ...
  • Booting up control plane ...
  • Configuring RBAC rules ...
# Verifying Kubernetes components...
  • Using image gcr.io/k8s-minikube/storage-provisioner:v5
# Enabled addons: storage-provisioner, default-storageclass
# Done! kubectl is now configured to use "minikube" cluster and "default"
namespace by default
```

To use the Oracle VM VirtualBox driver, provide the `--driver=virtualbox` option to `minikube`:

```
PS C:\> minikube start --driver=virtualbox
# minikube v1.20.0 on Microsoft Windows 10 Enterprise 10.0.18363 Build 18363
# Using the virtualbox driver based on user configuration
# Starting control plane node minikube in cluster minikube
# Creating virtualbox VM (CPUs=4, Memory=16384MB, Disk=20000MB) ...
# Preparing Kubernetes v1.20.2 on Docker 20.10.6 ...
  • Generating certificates and keys ...
  • Booting up control plane ...
  • Configuring RBAC rules ...
# Verifying Kubernetes components...
  • Using image gcr.io/k8s-minikube/storage-provisioner:v5
# Enabled addons: storage-provisioner, default-storageclass
# Done! kubectl is now configured to use "minikube" cluster and "default"
namespace by default
```



### Note

To set the default driver, run the command `minikube config set driver DRIVER`.

## ► 4. Verifying your minikube installation

Use the `minikube status` command to validate that the `minikube` installation is running successfully:

```
[root@host ~]# minikube status
minikube
type: Control Plane
host: Running
kublet: Running
apiserver: Running
kubeconfig: Configured
```

In case of errors, make sure you are using the appropriate driver during the installation, or refer to minikube Get Started documentation [<https://minikube.sigs.k8s.io/docs/start/>] for troubleshooting.

## ► 5. Adding extensions

`minikube` comes with the bare minimum set of features. To add more features, `minikube` provides an add-on based extension system. Developers can add more features by installing the needed add-ons.

Use the `minikube addons list` command for a comprehensive list of the add-ons available and the installation status.

- Installing the Ingress Add-on. For this course you must install the `ingress` add-on.

With your cluster up and ready, use the following command to enable the add-on:

```
[root@host ~]# minikube addons enable ingress
  • Using image k8s.gcr.io/ingress-nginx/controller:v0.44.0
  • Using image docker.io/jetttech/kube-webhook-certgen:v1.5.1
  • Using image docker.io/jetttech/kube-webhook-certgen:v1.5.1
# Verifying ingress addon...
# The 'ingress' addon is enabled
```

Versions and docker images can vary in your deployment, but make sure the final validation is successful.

- Installing the Dashboard add-on. The `dashboard` add-on is not required for this course but serves as a visual graphical interface if you are not comfortable with CLI commands.

Enable the `dashboard` add-on with the following command:

```
[root@host ~]# minikube addons enable dashboard
  • Using image kubernetesui/dashboard:v2.1.0
...output omitted...
# The 'dashboard' addon is enabled
```

Once the dashboard is enabled you can reach it by using the `minikube dashboard` command. This command will open the dashboard web application in your default browser.

Press `Ctrl+C` in the terminal to finish the connection to the dashboard.

## ► 6. Using a Developer Sandbox for Red Hat OpenShift as a Remote Kubernetes cluster

Developer Sandbox for Red Hat OpenShift is a free Kubernetes-as-a-Service platform offered by Red Hat Developers, based on Red Hat OpenShift.

Developer Sandbox allows users access to a pre-created Kubernetes cluster. Access is restricted to two namespaces (or projects if using OpenShift nomenclature). Developer Sandbox deletes pods after eight consecutive hours of running, and limits resources to 7 GB of RAM and 15 GB of persistent storage.

### 6.1. Create a Developer Sandbox account

Go to <https://developers.redhat.com/developer-sandbox> and click **Get started in the Sandbox**.



### What is the Sandbox?

The sandbox provides you with a private OpenShift environment in a shared, multi-tenant OpenShift cluster that is pre-configured with a set of developer tools. You can easily create containers from your source code or Dockerfile, build new applications using the samples and stacks provided, add services such as databases from our templates catalog, deploy Helm charts, and much more. Discover the rich capabilities of the full developer experience on OpenShift with the sandbox.

[Get started in the Sandbox](#)

[Learn about our IDE](#)

### What can I do in the Sandbox?

The sandbox is perfect for immediate access into OpenShift, but how can you start practicing your skills in this context if you don't have a starting point yet? Well, we've taken care of that for you with a curated series of activities that you

You need a free Red Hat account to use Developer Sandbox. Log in to your Red Hat account, or if you do not have one, then click **Create one now**. Fill in the form choosing a **Personal** account type, and then click **CREATE MY ACCOUNT**. You might need to accept Red Hat terms and conditions to use the Developer Program services.

When the account is ready you will be redirected back to the Developer Sandbox page. Click **Launch your Developer Sandbox for Red Hat OpenShift** to log in to Developer Sandbox.

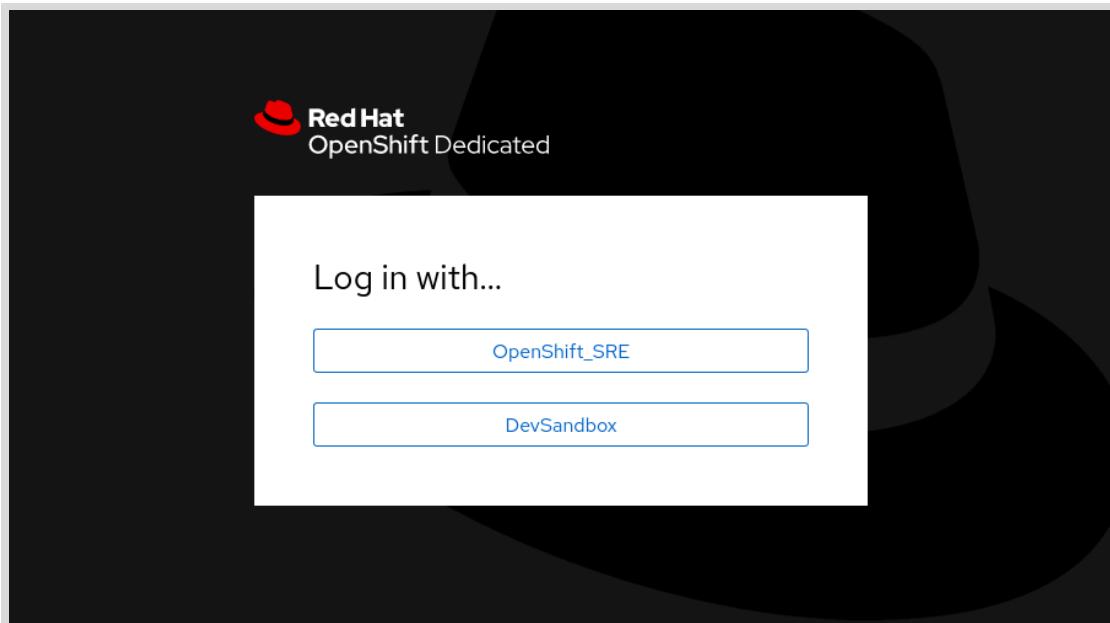
The screenshot shows a web-based developer sandbox interface. At the top, there's a navigation bar with four items: "Overview", "Get started in the Sandbox" (which is highlighted in blue), "Sandbox IDE", and "Sandbox activities". Below the navigation bar, the main content area has a title "Start your OpenShift experience in four simple steps". A descriptive paragraph follows, stating: "If you're exploring how to run your code in containers, our Developer Sandbox makes it simple. Not only can you easily deploy your application from a Git repo, you can also set up a cloud IDE for your entire team. Follow these four steps to quickly get started:". Below this text is a red rectangular button with white text that says "Launch your Developer Sandbox for Red Hat OpenShift". To the right of the button is a numbered list of four steps: 1. Provision your free Red Hat OpenShift development cluster, 2. Deploy a [sample application](#) or bring your own repo, 3. Edit code from the integrated [Eclipse Che-based cloud IDE](#), and 4. Fully explore the Red Hat OpenShift developer experience.

If you just created your account, then you might need to wait some seconds for account approval. You might need to verify your account via 2-factor authentication.

Once the account is approved and verified, Click **Start using your sandbox**. You might need to accept Red Hat terms and conditions to use the Developer Sandbox.

The screenshot shows the same web-based developer sandbox interface as the previous one. The "Get started in the Sandbox" tab is still selected. The main content area has the same "Start your OpenShift experience in four simple steps" title and descriptive paragraph. Below the steps, there is a light blue callout box containing the text "You're ready to get started!". Inside the callout box, it says "To launch your sandbox, click the button below and select DevSandbox when prompted." Below this text is a red rectangular button with white text that says "Start using your sandbox".

In the OpenShift log in form, click **DevSandbox** to select the authentication method.



If requested, then log in using your Red Hat account credentials.

## 6.2. Start Using Developer Sandbox

The Kubernetes cluster and two namespaces are created for you. Only the *username-dev* and *username-stage* namespaces are available, and you can not create more.

A screenshot of the Red Hat OpenShift Dedicated dashboard. The top navigation bar includes the Red Hat logo, "OpenShift Dedicated", and a user dropdown set to "my-username". The left sidebar has a "Developer" section with options: "+Add", "Topology" (which is selected and highlighted in blue), "Monitoring", "Search", "Builds", "Helm", and "Project". The main content area shows a "Topology" view with a message: "No resources found. To add content to your Project, create an Application, component or service using one of these options." Below this are two "Quick Starts" cards: "Get started with Spring" and "Monitor your sample application" under the "Quick Starts" heading, and a "Samples" card with the sub-instruction "Create an Application from a code sample".

## ► 7. Enabling external access to Ingress.

Some exercises require you to identify the external IP and hostname associated to your Ingress so you can access your application from outside the cluster.

### 7.1. External access to Minikube.

Routing traffic from your local machine to your Minikube Kubernetes cluster requires two steps.

First you must find the local IP assigned to your Ingress add on. The `minikube ip` command is the easiest way to find the ingress IP:

```
[user@host ~]$ minikube ip  
192.168.99.103
```

Your IP will probably be different as it depends on your virtual environment configuration.

Second, you must declare a hostname for your ingress, and associate the hostname to the ingress IP. For this course, unless the hostname is already in use, you will declare `hello.example.com` as the hostname.

The association between an IP and a hostname is usually performed by a DNS server, but to avoid registering a domain name to use with Minikube, we can use the system's local name resolution service.

In **Linux and macOS** systems, edit the `/etc/hosts` file with elevated privileges. You might use the `sudo` command or the `su` command to obtain elevated privileges. In **Windows** systems, edit the `C:\Windows\System32\drivers\etc\hosts` file with administrative privileges.

Add the following line to the bottom of the file and replace `<IP-ADDRESS>` with the IP address listed in the previous step.

```
<IP-ADDRESS> hello.example.com
```



### Note

If for any reason you need to delete and recreate your Minikube cluster, then review the IP address assigned to the cluster and update the hosts file accordingly.

For accessing services in the cluster you will use the declared hostname and potentially any path associated to the ingress. So, if using the `hello.example.com` hostname and assuming the application is mapped to the path `/myapp`, then your application will be available in the URL `http://hello.example.com/myapp`.

#### 7.2. External access to Developer Sandbox.

External access to Developer Sandbox is already prepared. The ingress controller in Developer Sandbox is pre-configured to listen to requests sent to any sub-domain for a hostname known as the **wildcard domain**. The wildcard domain is a configuration value set by administrators when creating the cluster.

To enable external access to a service you must provide a sub-domain for the wildcard domain, so first you must know the wildcard domain.

There are several approaches for finding the wildcard domain associated to your Developer Sandbox instance:

#### **Infer the wildcard domain from the Console URL**

When you log into your Developer Sandbox account, you will be redirected to the OpenShift Console installed in the cluster. The URL for the OpenShift Console is of the form `https://console-openshift-console.apps._WILDCARD-DOMAIN_`.

To get the wildcard domain, remove from the API URL the `https://console-openshift-console`, and replace `api` by `apps`. For example, the wildcard domain for the Console URL `https://console-openshift-console.apps.sandbox.x8i5.p1.openshiftapps.com` is `apps.sandbox.x8i5.p1.openshiftapps.com`.

### Infer the wildcard domain from the API URL

In *Guided Exercise: Connecting Kubectl To Your Cluster* you will find instructions to identify the API URL for your Developer Sandbox cluster.

To get the wildcard domain, remove from the API URL the `https://`, the `:6443`, and change `api` to `apps`. For example, the wildcard domain for the API URL `https://api.sandbox.x8i5.p1.openshiftapps.com:6443` is `apps.sandbox.x8i5.p1.openshiftapps.com`.

### Infer from a dry-run route creation

If you are using a Linux or macOS system, you can use the `kubectl` command to create a `route` resource that contains an example URL and extract the hostname from it:

```
[user@host ~]$ echo '{"apiVersion": "route.openshift.io/v1", \  
"kind": "Route", "metadata": {"name": "example"}, \  
"spec": {"to": {"kind": "Service", "name": "hello"}}}' | kubectl apply \  
--dry-run=server -o jsonpath='{.spec.host}' -f - \  
example-username-dev.apps.sandbox.x8i5.p1.openshiftapps.com
```

To get the wildcard domain, remove the first part of the hostname, that is everything before the first period. For example, the wildcard domain for the hostname `example-username-dev.apps.sandbox.x8i5.p1.openshiftapps.com` is `apps.sandbox.x8i5.p1.openshiftapps.com`.

Once you know the wildcard domain for your Developer Sandbox cluster, use it to generate a sub-domain to be used by your services. Remember that sub-domains must be unique for the shared Developer Sandbox cluster. One method for creating a unique sub-domain is to compose it in the format of `<DEPLOYMENT-NAME>-<NAMESPACE-NAME>. <WILDCARD-DOMAIN>`.

So, if using the `apps.sandbox.x8i5.p1.openshiftapps.com` wildcard domain and assuming a deployment named `hello` in a namespace named `username-dev` then you can compose your application hostname as `hello-username-dev.apps.sandbox.x8i5.p1.openshiftapps.com`.

Assuming the application is mapped to the path `/myapp`, then your application will be available in the URL `http://hello-username-dev.apps.sandbox.x8i5.p1.openshiftapps.com/myapp`.

## Finish

This concludes the guided exercise.

# Introducing Kubectl

---

## Objectives

After completing this section, you should be able to review the basic usage of the kubectl command and understand how to connect to your Kubernetes cluster by using the CLI.

## Introducing kubectl

The kubectl tool is a Kubernetes command-line tool that allows you to interact with your Kubernetes cluster. It provides an easy way to perform tasks such as creating resources or redirecting cluster traffic. The kubectl tool is available for the three main operating systems (Linux, Windows and macOS).

For example, the following command displays the kubectl and Kubernetes version.

```
[user@host ~]$ kubectl version
Client Version: version.Info{Major:"1", Minor:"21", GitVersion:"v1.21.0",
GitCommit:"cb303e613a121a29364f75cc67d3d580833a7479", GitTreeState:"clean",
BuildDate:"2021-04-08T16:31:21Z", GoVersion:"go1.16.1", Compiler:"gc",
Platform:"linux/amd64"}
Server Version: version.Info{Major:"1", Minor:"20",
Goos=linux, Goarch=amd64, GitCommit:"75370d3fb99594d6f0263f3de0bd08237381b77d", GitTreeState:"clean",
BuildDate:"2021-05-09T17:55:51Z", GoVersion:"go1.15.7", Compiler:"gc",
Platform:"linux/amd64"}
```

## Introducing kubectl configuration

Kubectl reads all the information necessary to connect to the Kubernetes cluster from a config file within your system. By default, this file is located at \$HOME/.kube/config. You can change this path by setting the environment variable KUBECONFIG to a custom file.

For example, the following sample sets the KUBECONFIG to the file /tmp/config.

```
[user@host ~]$ export KUBECONFIG=/tmp/config
```

All the commands related to the kubectl configuration are of the form:

```
[user@host ~]$ kubectl config option
```

If you want to see what the configuration file contains, then you can use the following command.

```
[user@host ~]$ kubectl config view
```

The kubectl configuration file comprehends three topics:

- Cluster: the URL for the API of a Kubernetes cluster. This URL identifies the cluster itself.

- User: credentials that identify a user connecting to the Kubernetes cluster.
- Context: puts together a cluster (the API URL) and a user (who is connecting to that cluster).

For example, you might have two contexts that are using different clusters but the same user.

## Defining Clusters

It is often necessary to work with multiple clusters, so `kubectl` can hold the information of several Kubernetes clusters. In relation to the configuration for `kubectl`, a cluster is just the URL of the API of the Kubernetes cluster. The `kubectl config set-cluster` command allows you to create a new cluster connection by using the API URL.

For example, the following command creates a new cluster connection named `my-cluster` with server `127.0.0.1:8087`.

```
[user@host ~]$ kubectl config set-cluster my-cluster --server=127.0.0.1:8087
```

Use the `get-clusters` option to list all available clusters.

```
[user@host ~]$ kubectl config get-clusters
my-cluster
my-cluster-2
```

## Defining Users

The cluster configuration tells `kubectl` where the Kubernetes cluster is. The user configuration identifies who connects to the cluster. To connect to the cluster, it is necessary to provide an authentication method. There are several options to authenticate with the cluster:

- Using a token

The following command creates a new user named `my-user` with the token `Py93bt12mT`.

```
[user@host ~]$ kubectl config set-credentials my-user --token=Py93bt12mT
```

- Using basic authentication

The following command creates a new user named `my-user` with username `kubernetes-username` and password `kubernetes-password`.

```
[user@host ~]$ kubectl config set-credentials my-user \
--username=redhat-username --password=redhat-password
```

- Using certificates

The following command creates a new user named `my-user` with a certificate `redhat-certificate.crt` and a key `redhat-key.key`.

```
[user@host ~]$ kubectl config set-credentials my-user \
--client-certificate=redhat-certificate.crt \
--client-key=redhat-key.key
```

Use the `get-users` option to list all available users.

```
[user@host ~]$ kubectl config get-users
my-user
my-user-2
```

## Defining Contexts

A context puts together a cluster and a user. Kubectl uses both to connect and authenticate against a Kubernetes cluster.

For example, the following command creates a new context by using a cluster named `my-cluster` and a user named `my-user`.

```
[user@host ~]$ kubectl config set-context \
--cluster=my-cluster --user=my-user
```

In a kubectl context, it is possible to set a namespace. If provided, then any command would be executed in that namespace. The following command creates a context that points to the `redhat-dev` namespace.

```
[user@host ~]$ kubectl config set-context my-context \
--cluster=my-cluster --user=my-user --namespace=redhat-dev
```

Once a context has been created, you can select it by using the `use-context` command.

```
[user@host ~]$ kubectl config use-context my-context
```

After executing the previous command, further kubectl commands will use the `my-cluster` context and, therefore, the cluster and user associated to that context.

You can also list the contexts available in the configuration by using the `get-contexts` option. The \* in the CURRENT column indicates the context that you are currently using.

CURRENT	NAME	CLUSTER	AUTHINFO	NAMESPACE
*	my-context	172.0.7.2:6443	my-user	redhat-dev
	my-context-2	172.1.8.0:6443	my-user-2	

Another way of checking the current context is by using the `current-context` option.

```
[user@host ~]$ kubectl config current-context
my-context
```

## Working with resources

Once you are connected to a Kubernetes cluster, kubectl allows you to list, create, update and delete Kubernetes resources. Most of these commands will be introduced in later chapters in the course, but there are some of them that can be mentioned at this point.

### The `get` command

This command is used to display one or more resources.

For example, `kubectl get pods` will display all pods in the current namespace.

```
[user@host ~]$ kubectl get pods
NAME      READY   STATUS    RESTARTS   AGE
example1  1/1     Running   0          67s
example2  1/1     Running   0          67s
```

If you want to display just the information for one pod, then add the pod's name to the previous command.

```
[user@host ~]$ kubectl get pods example1
NAME      READY   STATUS    RESTARTS   AGE
example1  1/1     Running   0          67s
```

You can use this command to display other resources (services, jobs, ingresses...).



### Note

Use the command `kubectl api-resources` to display all resource types that you can create.

## The delete command

This command allows you to delete a resource.

For example, `kubectl delete pod example1` deletes the pod named `example1`.

```
[user@host ~]$ kubectl delete pod example1
pod "example1" deleted
```

You can use this command to delete other resources (services, jobs, ingresses...).

## The apply command

A common way to manage Kubernetes resources is by using a `manifest`. A `manifest` is a YML or JSON file containing one or many Kubernetes resources.

The `apply` command allows you to create, update or delete resources from a manifest.

For example, the following snippet creates a `Deployment` resource.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
```

```
metadata:  
  labels:  
    app: nginx  
spec:  
  containers:  
  - name: nginx  
    image: nginx:1.7.9  
    ports:  
    - containerPort: 80
```

If the snippet was in a file named `deployment.yml`, then you could use `apply` to create the deployment. Note that the `-f` option is used to indicate the file.

```
[user@host ~]$ kubectl apply -f deployment.yml  
deployment.apps/nginx-deployment created
```



## References

### Overview of Kubectl

<https://kubernetes.io/docs/reference/kubectl/overview/>

### Kubectl Command Reference

<https://kubernetes.io/docs/reference/generated/kubectl/kubectl-commands>

## ► Guided Exercise

# Connecting Kubectl To Your Cluster

In this exercise you will install the kubectl command-line tool on your computer, and connect to the Kubernetes cluster that you will be using throughout the course.

## Outcomes

You should be able to:

- Install kubectl
- Connect to Minikube (if you are using Minikube)
- Connect to the OpenShift Developer Sandbox (if you are using the OpenShift Developer Sandbox)

## Before You Begin

Ensure you have either installed Minikube or created an OpenShift Developer Sandbox account.

## Instructions

The installation procedure of kubectl depends on your operating system.

### ► 1. Installing kubectl in Linux-based systems.

The Linux distributions supported in this course are Fedora Linux 34 and Red Hat Enterprise Linux 8.

You can install kubectl by downloading the binary and moving it to your PATH. At the same time, it is possible to use a package-manager.

#### 1.1. Using curl and the binary file

- Open a command-line terminal to download the kubectl binary.

```
[user@host ~]$ curl -LO https://dl.k8s.io/release/v1.21.0/bin/linux/amd64/kubectl
```

- Copy the binary to your PATH and make sure it has executable permissions.

```
[user@host ~]$ sudo install -m 0755 kubectl /usr/local/bin/kubectl
```

- Verify that kubectl has been installed successfully.

```
[user@host ~]$ kubectl version --client
Client Version: version.Info{Major:"1", Minor:"21", GitVersion:"v1.21.0",
GitCommit:"cb303e613a121a29364f75cc67d3d580833a7479", GitTreeState:"clean",
BuildDate:"2021-04-08T16:31:21Z", GoVersion:"go1.16.1", Compiler:"gc",
Platform:"linux/amd64"}
```

## 1.2. Using the dnf package manager.

- Open a command-line terminal. Install the package by using `dnf`.

```
[user@host ~]$ dnf install kubectl
...output omitted...
=====
Package           Architecture   Version      Repository      Size
=====
Installing:
  kubectl          x86_64        1.21.1-0    kubernetes     9.8 M

Transaction Summary
=====
Install 1 Package
...output omitted...
```

- Verify that `kubectl` has been installed successfully.

```
[user@host ~]$ kubectl version --client
Client Version: version.Info{Major:"1", Minor:"21", GitVersion:"v1.21.0",
GitCommit:"cb303e613a121a29364f75cc67d3d580833a7479", GitTreeState:"clean",
BuildDate:"2021-04-08T16:31:21Z", GoVersion:"go1.16.1", Compiler:"gc",
Platform:"linux/amd64"}
```

## ► 2. Installing kubectl in macOS.

### 2.1. Using curl and the binary file

- Open a command-line terminal. Download the `kubectl` binary by using `curl`. Replace the `os-architecture` parameter depending on your Mac processor. If your Mac comes with an Intel processor, use `amd64`. If your Mac comes with an Apple Silicon processor, use `arm64`.

```
[user@host ~]$ curl -LO "https://dl.k8s.io/release/v1.21.0/bin/darwin/os-
architecture/kubectl"
```

- Give the binary file executable permissions. Move the binary file executable to your PATH.

```
[user@host ~]$ chmod +x ./kubectl
[user@host ~]$ sudo mv ./kubectl /usr/local/bin/kubectl
```

- Verify that `kubectl` has been installed successfully.

```
[user@host ~]$ kubectl version --client
Client Version: version.Info{Major:"1", Minor:"21", GitVersion:"v1.21.0",
GitCommit:"cb303e613a121a29364f75cc67d3d580833a7479", GitTreeState:"clean",
BuildDate:"2021-04-08T16:31:21Z", GoVersion:"go1.16.1", Compiler:"gc",
Platform:"linux/amd64"}
```

## 2.2. Using the homebrew package manager



### Note

If you have previously installed Minikube with homebrew, kubectl should already be installed in your computer. You can skip the installation step and directly verify that kubectl has been installed correctly.

- Install kubectl by using brew

```
[user@host ~]$ brew install kubectl
...output omitted...
```

- Verify that kubectl has been installed successfully.

```
[user@host ~]$ kubectl version --client
Client Version: version.Info{Major:"1", Minor:"21", GitVersion:"v1.21.0",
GitCommit:"cb303e613a121a29364f75cc67d3d580833a7479", GitTreeState:"clean",
BuildDate:"2021-04-08T16:31:21Z", GoVersion:"go1.16.1", Compiler:"gc",
Platform:"linux/amd64"}
```

## ► 3. Installing kubectl in Windows

### 3.1. Using the binary

- Create a new folder, such as C:\kube, to use as the destination directory of the kubectl binary download.
- Download the latest release of the kubectl binary from <https://dl.k8s.io/release/v1.21.0/bin/windows/amd64/kubectl.exe> and save it to the previously created folder.
- Add the binary to your PATH.
  - In the search box on the taskbar, type env, and select **Edit the system environment variables** from the search results.
  - Click **Environment Variables** on the **System Properties** screen.
  - Under the **System variables** section, select the row containing Path and click **Edit**. This will open the **Edit environment variable** screen.
  - Click **New** and type the full path of the folder containing the kubectl.exe (for example, C:\kube).
  - Click **OK** to save the change and close the editor.
  - Click **OK → OK** to close out of the remaining screens.
- Verify that kubectl has been installed successfully.

```
[user@host ~]$ kubectl version --client
Client Version: version.Info{Major:"1", Minor:"21", GitVersion:"v1.21.0",
GitCommit:"cb303e613a121a29364f75cc67d3d580833a7479", GitTreeState:"clean",
BuildDate:"2021-04-08T16:31:21Z", GoVersion:"go1.16.1", Compiler:"gc",
Platform:"linux/amd64"}
```

#### ► 4. Download the D0100x-apps repository

The D0100x-apps GitHub repository contains files used throughout the course. The best way to work with it is by using Git. However, if you are not used to Git, you can simply download it as a regular folder and place it anywhere in your computer.

##### 4.1. Using Git

If you have Git installed in your system, you can simply clone the repository.

```
[user@host ~]$ git clone https://github.com/RedHatTraining/D0100x-apps.git
...output omitted...
```

##### 4.2. Downloading it from the GitHub web page

- Open a web browser and navigate to <https://github.com/RedHatTraining/D0100x-apps>.
- Click **Code** and then click **Download ZIP**. A ZIP file with the repository content is downloaded.

#### ► 5. Connecting kubectl to the Kubernetes cluster

The D0100x-apps contains a script that handles all configurations for you under the setup directory. The script you should run depends on your operating system (Linux, macOS or Windows) and the Kubernetes distribution you use (Minikube or the OpenShift Developer Sandbox).

If you run the Minikube script, it will configure Minikube to work as a Kubernetes cluster with restricted access. You will only have access to two namespaces. This way, we simulate a real Kubernetes cluster, where usually developers do not have full access.



##### Note

If you want to recover full access over your cluster, then you can change the kubectl context to the default Minikube context, `minikube`. Use the command `kubectl config use-context minikube`.

If you run the OpenShift Developer Sandbox script, it will configure kubectl to run commands against the Openshift Developer Sandbox cluster. The script will ask you to provide some information such as cluster url, username or token.

##### 5.1. Using Minikube

###### • Linux and macOS

In your command-line terminal, move to the D0100x-apps directory and run the script located at `./setup/operating-system/setup.sh`. Replace `operating-`

system for linux if you are using Linux. Use macos if you are using macOS. Make sure the script has executable permissions.

```
[user@host D0100x-apps]$ chmod +x ./setup/operating-system/setup.sh
[user@host D0100x-apps]$ ./setup/operating-system/setup.sh
Creating namespace 'user-dev' ...
Creating namespace 'user-stage' ...
Creating certificates ...
Creating Kubectl credentials for 'user' ...
Creating Kubectl context 'user-context' for user 'user' ...
Creating role resources for user 'user' in namespace 'user-dev' ...
Creating role resources for user 'user' in namespace 'user-stage' ...
Switched to context "user-context".
Switching to namespace 'user-dev' ...
OK!
```

- **Windows**

In your command-line terminal, move to the D0100x-apps directory. Run the command `Set-ExecutionPolicy -Scope Process -ExecutionPolicy Bypass`. This command allows you to run unsigned PowerShell scripts in your current terminal session.

```
[user@host D0100x-apps]$ Set-ExecutionPolicy -Scope Process -ExecutionPolicy
Bypass
```

Run the script located at `.\setup\windows\setup.ps1`.

```
[user@host D0100x-apps]$ .\setup\windows\setup.ps1
Creating namespace 'user-dev' ...
Creating namespace 'user-stage' ...
Creating certificates ...
Creating Kubectl credentials for 'user' ...
Creating Kubectl context 'user-context' for user 'user' ...
Creating role resources for user 'user' in namespace 'user-dev' ...
Creating role resources for user 'user' in namespace 'user-stage' ...
Switched to context "user-context".
Switching to namespace 'user-dev' ...
OK!
```

## 5.2. Using the OpenShift Developer Sandbox

The setup-sandbox script will ask you to provide a server, a token and your username. To find the server and token follow these steps:

- Open a web browser and navigate to the OpenShift Developer Sandbox website. Log in with your username and password.
- Click on your username in the upper right pane of the screen. A dropdown menu opens.
- In the dropdown menu, click **Copy login command**. A new tab opens, log in again with your account if necessary by clicking **DevSanbox**.

- Click **Display Token**.
- The token you must provide in the script shows in your web browser.
- The server you must provide is a parameter of the `oc login` command displayed. For example, in the command `oc login --token=sha256~Gs54-tjq1Bo-fo866bbddv8wbQ0bpmy321eS1qj1g --server=https://api.sandbox.x8i5.p1.openshiftapps.com:6443`, the server is `https://api.sandbox.x8i5.p1.openshiftapps.com:6443`.

- Keep these values. You will be asked for them in the script.

Run the appropriate script. The following instructions will depend on your operating system.

- **Linux and macOS**

In your command-line terminal, move to the `D0100x-apps` directory and run the script located at `./setup/operating-system/setup-sandbox.sh`. Replace

operating-system for linux if you are using Linux. Use macos if you are using macOS. Make sure the script has executable permissions.

```
[user@host D0100x-apps]$ chmod +x ./setup/operating-system/setup-sandbox.sh
[user@host D0100x-apps]$ ./setup/operating-system/setup-sandbox.sh
What is the OpenShift cluster URL?
https://api.sandbox.x8i5.p1.openshiftapps.com:6443
What is the OpenShift token?
sha256-wVSG...DDn0
What is your OpenShift username?
user
Creating Kubectl context...
Context created successfully
```

- **Windows**

In your command-line terminal, move to the D0100x-apps directory. Run the command `Set-ExecutionPolicy -Scope Process -ExecutionPolicy Bypass`. This command allows you to run unsigned PowerShell scripts in your current terminal session.

```
[user@host D0100x-apps]$ Set-ExecutionPolicy -Scope Process -ExecutionPolicy
Bypass
```

Run the script located at `.\setup\windows\setup-sandbox.ps1`.

```
[user@host D0100x-apps]$ .\setup\windows\setup-sandbox.ps1
What is the OpenShift cluster URL?
https://api.sandbox.x8i5.p1.openshiftapps.com:6443
What is the OpenShift token?
sha256-wVSG...DDn0
What is your OpenShift username?
user
Creating Kubectl context...
Context created successfully
```

## Finish

This concludes the guided exercise.

# Running and Interacting with Your First Application

## Objectives

After completing this section, you should be able to execute a pre-built application in your Kubernetes cluster and review the resources related to the process.

## Running Pods From Container Images

The simplest way to run a container in your Kubernetes cluster is with the `kubectl run` command. At a minimum, you must specify a name and container image. This container image must be accessible by the Kubernetes cluster.

The following example command creates a new pod named `myname` that uses the container image referenced by `myimage`.

```
[user@host ~] kubectl run myname --image=myimage
```

Recent versions of `kubectl run` can only create new pods. For example, older example uses of this command might include a `--replicas` option, which has been removed.



### Important

Use `kubectl run` to create pods for quick tests and experimentation.

## Creating Resources

The `kubectl create` command creates new resources within the Kubernetes cluster. You must specify the name and type of the resource, along with any information required for that resource type.

You can specify `--dry-run=client` to prevent the creation of the object within the cluster. By combining this with the output type option, you can generate resource definitions.

For example, the following command outputs the YAML definition of a new deployment resource named `webserver`, by using the `nginx` image.

```
[user@host ~] kubectl create deployment webserver \
--image=nginx --dry-run=client -o yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  creationTimestamp: null
  labels:
    app: webserver
    name: webserver
spec:
  replicas: 1
  selector:
```

```
matchLabels:  
  app: webserver  
strategy: {}  
template:  
  metadata:  
    creationTimestamp: null  
    labels:  
      app: webserver  
  spec:  
    containers:  
    - image: nginx  
      name: nginx  
      resources: {}  
status: {}
```

You can save this output to a file to actually create the object later. Reference the file by specifying it with the `-f` option.

For example, the following command creates a new resource using the definition found in `mydef.yaml`:

```
[user@host ~] kubectl create -f mydef.yaml  
...output omitted...
```

## Comparing Create and Apply

The `kubectl create` command can only create *new* resources. To modify an existing resource or create it if it does not exist, use the `kubectl apply` command. Like `kubectl create`, this command also accepts YAML or JSON definitions.

If you are familiar with certain variants of SQL syntax, then `kubectl create` is comparable to `INSERT` whereas `kubectl apply` is akin to `UPsert`.

## Executing Commands Within an Existing Pod

With the `kubectl exec` command, you can execute commands inside *existing* pods. The `kubectl exec` command is useful for troubleshooting problematic containers, but the changes are not persistent. To make persistent changes, you must change the container image.

At a minimum, this command requires the name of the pod and the command to execute. For example, the following command will execute `ls` within the running pod named `mypod`.

```
[user@host ~] kubectl exec mypod -- /bin/ls  
bin  
boot  
dev  
...output omitted...
```

The `--` separates the parts of the command intended for Kubernetes itself from the command that should be passed to and executed within the container.

## Connecting a Shell to an Existing Pod

A common use of `kubectl exec` is to open a new shell within a running pod. For example, the following command creates and attaches a new shell session to the pod named `mypod`:

```
[user@host ~] kubectl exec --stdin --tty mypod -- /bin/bash
```

Notice the addition of the `--stdin` and `--tty` options. These are necessary to ensure input and output are forwarded correctly to the interactive shell within the container.



### References

#### **Remove kubectl run generators PR**

<https://github.com/kubernetes/kubernetes/pull/87077>

#### **Get a Shell to a Running Container|Kubernetes**

<https://kubernetes.io/docs/tasks/debug-application-cluster/get-shell-running-container/>

#### **kubectl Cheat Sheet**

<https://kubernetes.io/docs/reference/kubectl/cheatsheet/>

## ► Guided Exercise

# Running and Interacting with Your First Application

In this exercise, you will create a pod and connect to it. You will also create and manage a new namespace resource by using a resource definition file that you create.

### Outcomes

You should be able to:

- Connect a shell session to an existing pod.
- Create a resource definition file.
- Use a resource definition to create and update a namespace resource.



#### Note

You do not need to understand Kubernetes namespaces to do this exercise, as they are solely used as an example resource.

### Before You Begin

You need a working Kubernetes cluster, and your `kubectl` command must be configured to communicate with the cluster.

Ensure your `kubectl` context refers to the `user-dev` namespace. Use the `kubectl config set-context --current --namespace=user-dev` command to switch to the appropriate namespace.

### Instructions

- 1. Use `kubectl run` and `kubectl exec` to create a new pod and attach a shell session to it.
- 1.1. Create a new pod named `webserver` that uses the `httpd` container image.



#### Note

This course uses the backslash character (\) to break long commands. On Linux and macOS, you can use the line breaks.

On Windows, use the backtick character (`) to break long commands. Alternatively, do not break long commands.

Refer to *Orientation to the Classroom Environment* for more information about long commands.

```
[user@host ~] kubectl run webserver \
--image=registry.access.redhat.com/ubi8/httpd-24:1-161
pod/webserver created
```

1.2. Confirm the `webserver` pod is running.

```
[user@host ~] kubectl get pods
NAME      READY   STATUS    RESTARTS   AGE
webserver  1/1     Running   0          5s
```

1.3. Connect to the pod by using `kubectl exec`.

```
[user@host ~] kubectl exec --stdin --tty \
webserver -- /bin/bash
root@webserver:/#
```

1.4. View the contents of the `httpd` configuration file within the pod.

```
[root@webserver:/#] cat /etc/httpd/conf/httpd.conf
...output omitted...
ServerAdmin root@localhost
...output omitted...
```

1.5. Disconnect from the pod by using the `exit` command.

```
[root@webserver:/#] exit
```

► 2. Create a pod resource definition file and use that file to create another pod in your cluster.

2.1. Create a new file named `probes-pod.yaml`. Add the following resource manifest to the file.

```
apiVersion: v1
kind: Pod
metadata:
  name: probes
  labels:
    app: probes
  namespace: user-dev
spec:
  containers:
    - name: probes
      image: 'quay.io/redhattraining/do100-probes:external'
      ports:
        - containerPort: 8080
```

Replace `user` by your OpenShift Developer Sandbox username.

You can generate the basic YAML file with the `kubectl run` command

```
[user@host ~] kubectl run probes  
--image=quay.io/redhattraining/do100-probes:external --dry-run=client -o yaml
```

- 2.2. Use the `kubectl create` command to create a new pod from the resource manifest file.

```
[user@host ~] kubectl create -f probes-pod.yml  
pod/probes created
```

- 2.3. Verify that the pod is running.

```
[user@host ~] kubectl get pods  
NAME      READY   STATUS    RESTARTS   AGE  
probes    1/1     Running   0          15s  
webserver 1/1     Running   0          10m
```

► **3.** Modify the `metadata.labels.app` field of the pod manifest and apply the changes.

- 3.1. Open the `probes-pod.yaml` file. Replace `probes` by `do100-probes` in the `metadata.labels.app` field.

```
apiVersion: v1  
kind: Pod  
metadata:  
  name: probes  
  labels:  
    app: do100-probes  
...output omitted...
```

- 3.2. Attempt to update the pod by using the `kubectl create` command.

```
[user@host ~] kubectl create -f probes-pod.yml  
Error from server (AlreadyExists): error when creating "probes-pod.yml": pods  
"probes" already exists
```

Notice the error. Because you have previously created the pod, you can not use `kubectl create`.

- 3.3. Update the pod by using the `kubectl apply` command.

```
[user@host ~] kubectl apply -f probes-pod.yml  
pod/probes configured
```



### Note

The preceding usage of `kubectl apply` produces a warning that the `kubectl.kubernetes.io/last-applied-configuration` annotation is missing. In most scenarios, this can be safely ignored.

Ideally, to use `kubectl apply` in this precise manner, you should use the `--save-config` option with `kubectl create`.

3.4. Verify that the label has been updated by using the `kubectl describe pod`.

```
[user@host ~] kubectl describe pod probes
Name:           probes
Namespace:      user-dev
...output omitted...
Labels:         app=do100-probes
...output omitted...
```

## Finish

Delete the pod and namespace to clean your cluster.

```
[user@host ~]$ kubectl delete pod/webserver
pod "webserver" deleted
[user@host ~]$ kubectl delete pod/probes
pod "probes" deleted
```

This concludes the guided exercise.

# Summary

---

In this chapter, you learned:

- Different application and organizational needs determine how you should run and configure your Kubernetes cluster.
- Kubernetes includes extensions to provide additional cluster functionality, such as storage and ingress.
- The command-line tool `kubectl` is the main way to interact and configure a Kubernetes cluster.



# Red Hat

## Training and Certification

### Student Workbook

Kubernetes 1.22 DO100B

### Managing Cloud-native Applications with Kubernetes

Edition 2







## Join a community dedicated to learning open source

The Red Hat® Learning Community is a collaborative platform for users to accelerate open source skill adoption while working with Red Hat products and experts.



**Network** with tens of thousands of community members



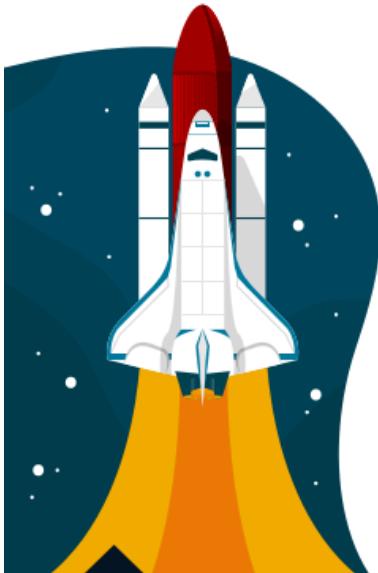
**Engage** in thousands of active conversations and posts



**Join and interact** with hundreds of certified training instructors



**Unlock** badges as you participate and accomplish new goals



This knowledge-sharing platform creates a space where learners can connect, ask questions, and collaborate with other open source practitioners.

**Access** free Red Hat training videos

**Discover** the latest Red Hat Training and Certification news

**Connect** with your instructor - and your classmates - before, after, and during your training course.

**Join** peers as you explore Red Hat products

Join the conversation [learn.redhat.com](https://learn.redhat.com)



Copyright © 2020 Red Hat, Inc. Red Hat, Red Hat Enterprise Linux, the Red Hat logo, and Ansible are trademarks or registered trademarks of Red Hat, Inc. or its subsidiaries in the United States and other countries.



# Managing Cloud-native Applications with Kubernetes



**Kubernetes 1.22 DO100B**  
**Managing Cloud-native Applications with Kubernetes**  
**Edition 2 7067502**  
**Publication date 20220000**

Authors: Enol Alvarez de Prado, Manuel Aude Morales,  
Guy Bianco IV, Marek Czernek, Natalie Lind, Michael Phillips,  
Eduardo Ramirez Ronco, Alejandro Serna Borja, Jordi Sola Alaball  
Course Architects: Zachary Guterman, Fernando Lozano  
DevOps Engineer: Richard Allred  
Editor: Sam Ffrench

Copyright © 2022 Red Hat, Inc.

The contents of this course and all its modules and related materials, including handouts to audience members, are  
Copyright © 2022 Red Hat, Inc.

No part of this publication may be stored in a retrieval system, transmitted or reproduced in any way, including, but not limited to, photocopy, photograph, magnetic, electronic or other record, without the prior written permission of Red Hat, Inc.

This instructional program, including all material provided herein, is supplied without any guarantees from Red Hat, Inc. Red Hat, Inc. assumes no liability for damages or legal action arising from the use or misuse of contents or details contained herein.

If you believe Red Hat training materials are being used, copied, or otherwise improperly distributed, please send email to [training@redhat.com](mailto:training@redhat.com) or phone toll-free (USA) +1 (866) 626-2994 or +1 (919) 754-3700.

Red Hat, Red Hat Enterprise Linux, the Red Hat logo, JBoss, OpenShift, Fedora, Hibernate, Ansible, CloudForms, RHCA, RHCE, RHCSA, Ceph, and Gluster are trademarks or registered trademarks of Red Hat, Inc. or its subsidiaries in the United States and other countries.

Linux® is the registered trademark of Linus Torvalds in the United States and other countries.

Java® is a registered trademark of Oracle American, Inc. and/or its affiliates.

XFS® is a registered trademark of Hewlett Packard Enterprise Development LP or its subsidiaries in the United States and/or other countries.

MySQL® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js® is a trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack word mark and the Square O Design, together or apart, are trademarks or registered trademarks of OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. Red Hat, Inc. is not affiliated with, endorsed by, or sponsored by the OpenStack Foundation or the OpenStack community.

All other trademarks are the property of their respective owners.

<b>Document Conventions</b>	<b>ix</b>
	ix
<b>Introduction</b>	<b>xi</b>
Managing Cloud-native Applications with Kubernetes .....	xi
Orientation to the Classroom Environment .....	xii
<b>1. Deploying Managed Applications</b>	<b>1</b>
Deploying Managed Applications .....	2
Guided Exercise: Deploying Managed Applications .....	8
Quiz: Deploying Managed Applications .....	14
Summary .....	18
<b>2. Configuring Networking in Kubernetes</b>	<b>19</b>
Exposing Applications for External Access .....	20
Guided Exercise: Exposing Applications for External Access .....	25
Quiz: Exposing Applications for External Access .....	30
Exposing Applications for External Access: Ingress Resources .....	34
Guided Exercise: Exposing Applications for External Access: Ingress Resources .....	37
Quiz: Exposing Applications for External Access: Ingress Resources .....	41
Summary .....	47
<b>3. Customize Deployments for Application Requirements</b>	<b>49</b>
Limiting Resource Usage .....	50
Guided Exercise: Limiting Resource Usage .....	58
Quiz: Limiting Resource Usage .....	61
Proving Liveness, Readiness and Startup .....	65
Guided Exercise: Proving Liveness, Readiness and Startup .....	69
Quiz: Proving Liveness, Readiness and Startup .....	76
Configuring Cloud Applications .....	80
Guided Exercise: Configuring Cloud Applications .....	87
Quiz: Configuring Cloud Applications .....	92
Summary .....	96
<b>4. Implementing Cloud Deployment Strategies</b>	<b>97</b>
Implementing Cloud Deployment Strategies .....	98
Guided Exercise: Implementing Cloud Deployment Strategies .....	101
Quiz: Implementing Cloud Deployment Strategies .....	104
Quiz: End of Course Quiz .....	108
Summary .....	120
<b>A. Installing and Configuring Kubernetes</b>	<b>121</b>
Guided Exercise: Installing and Configuring Kubernetes .....	122
<b>B. Connecting to your Kubernetes Cluster</b>	<b>135</b>
Guided Exercise: Connecting to your Kubernetes Cluster .....	136



# Document Conventions

---

This section describes various conventions and practices used throughout all Red Hat Training courses.

## Admonitions

Red Hat Training courses use the following admonitions:



### References

These describe where to find external documentation relevant to a subject.



### Note

These are tips, shortcuts, or alternative approaches to the task at hand. Ignoring a note should have no negative consequences, but you might miss out on something that makes your life easier.



### Important

These provide details of information that is easily missed: configuration changes that only apply to the current session, or services that need restarting before an update will apply. Ignoring these admonitions will not cause data loss, but may cause irritation and frustration.



### Warning

These should not be ignored. Ignoring these admonitions will most likely cause data loss.

## Inclusive Language

Red Hat Training is currently reviewing its use of language in various areas to help remove any potentially offensive terms. This is an ongoing process and requires alignment with the products and services covered in Red Hat Training courses. Red Hat appreciates your patience during this process.



# Introduction

## Managing Cloud-native Applications with Kubernetes

Managing Cloud-native Applications with Kubernetes (DO100b) is designed for IT professionals without previous cloud application deployment experience to learn basic Kubernetes skills. This course is the second course of a three-course specialization. In this specialization, you will run, deploy, and test containerized applications with zero-downtime releases.

### Course Objectives

- Introduction to deploying Cloud Native Applications on a Kubernetes cluster.

### Audience

- Developers who wish to deploy cloud applications.
- Administrators who are new to Kubernetes.
- Architects who are considering using container technologies in software architectures.
- Site Reliability Engineers who are considering using Kubernetes.

### Prerequisites

- Experience running commands and navigating the file system on Linux, MacOS, and Windows.
- Experience with web application development and corresponding technologies.
- Finishing the DO100a Course is strongly recommended.

# Orientation to the Classroom Environment

---

DO100b is a **Bring Your Developer Workstation (BYDW)** class, where you use your own internet-enabled system to access the shared OpenShift cluster. The following operating systems are supported:

- Red Hat Enterprise Linux 8 or Fedora Workstation 34 or later
- Ubuntu 20.04 LTS or later
- Microsoft Windows 10
- macOS 10.15 or later

## BYDW System Requirements

Attribute	Minimum Requirements	Recommended
<b>CPU</b>	1.6 GHz or faster processor	Multi-core i7 or equivalent
<b>Memory</b>	8 GB	16 GB or more
<b>Disk</b>	10 GB free space HD	10 GB or more free space SSD
<b>Display Resolution</b>	1024x768	1920x1080 or greater

This course follows the DO100a course, which instructed you to install the following programs:

- Git 2.18 or later (Git Bash for Windows systems)
- The Kubernetes CLI (`kubectl`) v1.21 or later
- Minikube (Optional)



### Important

If you did not finish the prerequisite course, ensure you have installed and correctly configured the required programs.

Visit the DO100a course for more information about the prerequisite workstation configuration that is necessary for this course.

## BYDW Systems Support Considerations

Depending on your system, you might see differences between your command-line shell and the examples given in this course.

### **Red Hat Enterprise Linux or Fedora Workstation**

- If you use Bash as the default shell, then your prompt might match the [user@host ~]\$ prompt used in the course examples, although different Bash configurations can produce different results.
- If you use another shell, such as zsh, then your prompt format will differ from the prompt used in the course examples.
- When performing the exercises, interpret the [user@host ~]\$ prompt used in the course as a representation of your system prompt.
- All the commands from the exercises should be functional.

### **Ubuntu**

- You might find differences in the prompt format.
- In Ubuntu, your prompt might be similar to user@host:~\$.
- When performing the exercises, interpret the [user@host ~]\$ prompt used in the course as a representation of your Ubuntu prompt.
- All the commands from the exercises should be functional.

### **macOS**

- You might find differences in the prompt format.
- In macOS, your prompt might be similar to host:~ user\$.
- When performing the exercises, interpret the [user@host ~]\$ prompt used in the course as a representation of your macOS prompt.
- All the commands from the exercises should be functional.
- You might need to grant execution permissions to the installed runtimes.

### **Microsoft Windows**

- Windows does not support Bash natively. Instead, you must use PowerShell.
- In Windows PowerShell, your prompt should be similar to PS C:\Users\user>.
- When performing the exercises, interpret the [user@host ~]\$ Bash prompt as a representation of your Windows PowerShell prompt.
- For some commands, Bash syntax and PowerShell syntax are similar, such as cd or ls. You can also use the slash character (/) in file system paths.
- For other commands, the course provides help to transform Bash commands into equivalent PowerShell commands.
- This course only provides support for Windows PowerShell.
- The Windows firewall might ask for additional permissions in certain exercises.

## **Executing Long Commands**

This course breaks long commands into multiple lines by using the backslash character (\), for example:

```
[user@host ~]$ long command \
argument_1 \
argument_2
```

The preceding example works on the Linux and macOS systems.

On Windows, use the backtick character (`), for example:

```
[user@host ~]$ long command ` \
argument_1 ` \
argument_2
```

Alternatively, you can type commands in one line on all systems, such as:

```
[user@host ~]$ long command argument_1 argument_2
```

## Chapter 1

# Deploying Managed Applications

### Goal

Introduce the deployment resource and link to container management.

### Objectives

Use Kubernetes container management capabilities to deploy containerized applications in a declarative way.

### Sections

Deploying Managed Applications (and Guided Exercise, Quiz)

# Deploying Managed Applications

---

## Objectives

After completing this section, you should be able to use Kubernetes container management capabilities to deploy containerized applications in a declarative way.

## Managing Containers

One of the most significant features of Kubernetes is that it enables developers to use a declarative approach for automatic container life cycle management. *Declarative* approach means developers declare **what** should be the status of the application, and Kubernetes will update the containers to reach that state.

Some basic values a developer must declare are:

- The container images used by the application.
- The number of instances (replicas) of the application that Kubernetes must run simultaneously.
- The strategy for updating the replicas when a new version of the application is available.

With this information, Kubernetes deploys the application, keeps the number of replicas, and terminates or redeploys application containers when the state of the application does not match the declared configuration. Kubernetes continuously revisits this information and updates the state of the application accordingly.

This behavior enables important features of Kubernetes as a container management platform:

### Automatic deployment

Kubernetes deploys the configured application without manual intervention.

### Automatic scaling

Kubernetes creates as many replicas of the application as requested. If the number of replicas requested increases or decreases, then Kubernetes automatically creates new containers (scale-up) or terminates exceeding containers (scale-down) to match the requested number.

### Automatic restart

If a replica terminates unexpectedly or becomes unresponsive, then Kubernetes deletes the associated container and automatically spins up a new one to match the expected replica count.

### Automatic rollout

When a new version of the application is detected, or a new configuration applies, Kubernetes automatically updates the existing replicas. Kubernetes monitors this rollout process to make sure the application retains the declared number of active replicas.

## Creating a Deployment

A Deployment resource contains all the information Kubernetes needs to manage the life cycle of the application's containers.

The simplest way to create a Deployment resource is by using the `kubectl create deployment` command.

```
[user@host ~]$ kubectl create deployment deployment-name \
--image image --replicas=REPLICAS_NUMBER
deployment.apps/deployment-name created
```

This command creates a Deployment resource named *deployment-name*. This Deployment instructs Kubernetes to deploy *three* replicas of the application pod, and to use the *image* container image.

Use the `kubectl get deployment deployment-name` command to retrieve the Deployment resource from Kubernetes.

Use the `--output yaml` parameter to get detailed information about the resource in the YAML format. Alternatively, you can use the short `-o yaml` version.

```
[user@host ~]$ kubectl get deployment deployment-name -o yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: deployment-name
    name: deployment-name
  ...output omitted...
spec:
  ...output omitted...
  replicas: 3
  ...output omitted...
  selector:
    matchLabels:
      app: deployment-name
    template:
      metadata:
        ...output omitted...
        labels:
          app: deployment-name
      spec:
        containers:
        - image: image
  ...output omitted...
```



### Note

Review `kubectl get` options and adapt the output to your needs. For example, use the `--show-managed-fields=false` to skip the `metadata.managedFields` section of the Deployment. Use different values for the `'-o'` option for formatting or filtering the output. Find more details in the links in the References section.

The Kubernetes declarative deployment approach enables you to use the *GitOps* principles. GitOps focuses on a versioned repository, such as git, which stores your deployment configuration.

Following GitOps principles, you can store the Deployment manifest in YAML or JSON format in your application repository. Then, after the appropriate changes, you can create the Deployment manually or programmatically by using the `kubectl apply -f deployment-file` command.

You can also edit Deployment resource manifests directly from the command line. The `kubectl edit deployment deployment-name` command retrieves the Deployment resource and opens it in a local text editor (the exact editor depends on your system and local configuration). When the editor closes, the `kubectl edit` command applies any changes to the manifest.



### Note

The `kubectl get deployment deployment-name -o yaml` command contains run time information about the deployment.

For example, the output contains current deployment status, creation timestamps, and similar information. Deployment YAML files with run time information might not be reusable across namespaces and projects.

Deployment YAML files that you want to check-in to your version control system, such as git, should not contain any run time information. Kubernetes generates this information as needed.

## Understanding the Schema of a Deployment Resource

Before updating a Deployment resource, it is important to know the schema of the resource and the meaning of the most significant parts.

The following depicts the main entries in a Deployment manifest:

```
apiVersion: apps/v1
kind: Deployment 1
metadata: 2
...output omitted...
labels:
  app: versioned-hello
  name: versioned-hello
...output omitted...
spec: 3
...output omitted...
  replicas: 3 4
...output omitted...
  selector:
    matchLabels:
      app: versioned-hello
  strategy:
    type: RollingUpdate 5
...output omitted...
  template:
    metadata:
      labels:
        app: versioned-hello
...output omitted...
    spec: 6
      containers:
        - image: quay.io/redhattraining/versioned-hello:v1.1 7
```

```

name: versioned-hello
...output omitted...
status: ⑧
...output omitted...
replicas: 3 ⑨
...output omitted...

```

- ① Manifest kind identifies the resource type.
- ② Manifest metadata. Include deployment name and labels.
- ③ Deployment specification contains deployment configuration.
- ④ Number of desired replicas of the container.
- ⑤ Deployment strategy to use when updating pods.
- ⑥ Includes a list of pod definitions for each new container created by the deployment as well as other fields to control container management.
- ⑦ Container image used to create new containers.
- ⑧ Current status of the deployment. This section is automatically generated and updated by Kubernetes.
- ⑨ The current number of replicas currently deployed.

## Replicas

The `replicas` section under the `spec` section (also denoted as the `spec.replicas` section) declares the number of *expected* replicas that Kubernetes should keep running. Kubernetes will continuously review the number of replicas that are running and responsive, and scale accordingly.

## Deployment Strategy

When the application changes due to an image change or a configuration change, Kubernetes replaces the old running containers with updated ones. However, just redeploying all replicas at once can lead to problems with the application, such as:

- Leaving the application with too few running replicas.
- Creating too many replicas and leading to an overcommitment of resources.
- Rendering the application unavailable if the new version is faulty.

To avoid these issues, Kubernetes defines two strategies:

### `RollingUpdate`

Kubernetes terminates and deploys pods progressively. This strategy defines a maximum amount of pods unavailable anytime. It defines the difference between the available pods and the desired available replicas. The `RollingUpdate` strategy also defines an amount of pods deployed at any time over the number of desired replicas. Both values default to 25% of the desired replicas.

### `Recreate`

This strategy means that no issues are expected to impact the application, so Kubernetes terminates all replicas and recreates them on a best effort basis.

**Note**

Different distributions of Kubernetes include other deployment strategies. Refer to the documentation of the distribution for details.

## Template

When Kubernetes deploys new pods, it needs the exact manifest to create the pod. The `spec.template.spec` section holds exactly the same structure as a Pod manifest. Kubernetes uses this section to create new pods as needed.

The following entries in the template deserve special attention:

- The `spec.template.spec.containers.image` entry declares the image (or images) Kubernetes will deploy in the pods managed by this Deployment.
- Kubernetes uses the `spec.template.spec.containers.name` entry as a *prefix* for the names of the pods it creates.

## Labels

Labels are key-value pairs assigned in resource manifests. Both developers and Kubernetes use labels to identify sets of grouped resources, such as all resources belonging to the same application or environment. Depending on the position inside the Deployment, labels have a different meaning:

### `metadata.labels`

Labels applied directly to the manifest, in this case the Deployment resource. You can find objects matching these labels with the `kubectl get kind --selector="key=value"`. For example, `kubectl get deployment --selector="app=myapp"` returns all deployments with a label `app=myapp` in the `metadata.labels` section.

### `spec.selector.matchLabels.selector`

Determine what pods are under the control of the Deployment resource. Even if some pods in the cluster are not deployed via this Deployment, if they match the labels in this section then they will count as replicas and follow the rules defined in this Deployment manifest.

### `spec.template.metadata.labels`

Like the rest of the template, it defines how Kubernetes creates new pods using this Deployment. Kubernetes will label all the pods created by this Deployment resource with these values.



## References

### Deployments

<https://kubernetes.io/docs/concepts/workloads/controllers/deployment/>

### DeploymentSpec v1

<https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.20/#deploymentspec-v1-apps>

### Labels and Selectors

<https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/>

### Kubernetes Reference Docs

<https://kubernetes.io/docs/reference/generated/kubectl/kubectl-commands>

## ► Guided Exercise

# Deploying Managed Applications

In this exercise you will deploy a managed containerized application in your Kubernetes cluster. You will observe how automatic deployment works and some of the *High Availability* features of Kubernetes.

## Outcomes

You should be able to:

- Deploy an application container with several replicas.
- Review the structure of the `Deployment` resource manifest.
- Update the application to a new version without losing availability.

## Before You Begin

You need a working Kubernetes cluster, and your `kubectl` command must be configured to communicate with the cluster.

Make sure your `kubectl` context refers to a namespace where you have enough permissions, usually `username-dev` or `username-stage`. Use the `kubectl config set-context --current --namespace=namespace` command to switch to the appropriate namespace.

## Instructions

In this exercise, you will deploy an existing application by using the container image `quay.io/redhattraining/do100-versioned-hello:v1.0-external`.

- 1. Create a default deployment that starts one replica of the `do100-versioned-hello` application and review the deployment was successful.
- 1.1. Use the `kubectl create deployment` command to create the deployment. Name the deployment `do100-versioned-hello`.



### Note

This course uses the backslash character (\) to break long commands. On Linux and macOS, you can use the line breaks.

On Windows, use the backtick character (`) to break long commands. Alternatively, do not break long commands.

Refer to *Orientation to the Classroom Environment* for more information about long commands.

```
[user@host ~]$ kubectl create deployment do100-versioned-hello \
--image quay.io/redhattraining/do100-versioned-hello:v1.0-external
deployment.apps/do100-versioned-hello created
```

- 1.2. Validate that the deployment created the expected application pod. Use the `kubectl get pods -w` command in a new terminal. Keep the command running for observing further updates.

```
[user@host ~]$ kubectl get pods -w
NAME                               READY   STATUS    RESTARTS   AGE
do100-versioned-hello-76c4494b5d-4ldff   1/1     Running   0          51m
```

- 1.3. Use the `kubectl describe deployment` command to get relevant information about the deployment:

```
[user@host ~]$ kubectl describe deployment do100-versioned-hello
Name:           do100-versioned-hello
...output omitted...
Labels:         app=do100-versioned-hello
...output omitted...
Selector:       app=do100-versioned-hello
Replicas:       1 desired | 1 updated | 1 total | 1 available | 0 unavailable
...output omitted...
Pod Template:
  Labels:  app=do100-versioned-hello
  Containers:
    do100-versioned-hello:
      Image:   quay.io/redhattraining/do100-versioned-hello:v1.0-external
...output omitted...
```

Optionally, use the `kubectl get deployment` command to get the full manifest for the deployment:

```
[user@host ~]$ kubectl get deployment do100-versioned-hello -o yaml
apiVersion: apps/v1
kind: Deployment
...output omitted...
labels:
  app: do100-versioned-hello
  name: do100-versioned-hello
...output omitted...
spec:
...output omitted...
  replicas: 1
...output omitted...
  selector:
    matchLabels:
      app: do100-versioned-hello
  strategy:
    rollingUpdate:
      maxSurge: 25%
```

```
maxUnavailable: 25%
type: RollingUpdate
template:
  metadata:
    ...output omitted...
    labels:
      app: do100-versioned-hello
  spec:
    containers:
      - image: quay.io/redhattraining/do100-versioned-hello:v1.0-external
    ...output omitted...
      name: do100-versioned-hello
    ...output omitted...
```

► 2. Enable high availability by increasing the number of replicas to two.

- 2.1. Edit the number of replicas in the Deployment resource using the `kubectl scale deployment` command:

```
[user@host ~]$ kubectl scale deployment do100-versioned-hello --replicas=2
deployment.apps/do100-versioned-hello scaled
```

- 2.2. Validate that Kubernetes deployed a new replica pod. Go back to the terminal where the `kubectl get pods -w` command is running. Observe how the output displays a new pod named `do100-versioned-hello-76c4494b5d-qtf9`. The pod updates from the Pending status to ContainerCreating and finally to Running.

```
[user@host ~]$ kubectl get pods -w
NAME                               READY   STATUS            RESTARTS   AGE
do100-versioned-hello-76c4494b5d-4ldff  1/1    Running          0          51m
do100-versioned-hello-76c4494b5d-qtf9  0/1    Pending           0          0s
do100-versioned-hello-76c4494b5d-qtf9  0/1    Pending           0          0s
do100-versioned-hello-76c4494b5d-qtf9  0/1    ContainerCreating 0          0s
do100-versioned-hello-76c4494b5d-qtf9  1/1    Running          0          2s
```

► 3. Verify high availability features in Kubernetes. Kubernetes must ensure that two replicas are available at all times. Terminate one pod and observe how Kubernetes creates a new pod to ensure the desired number of replicas.

- 3.1. Terminate one of the pods by using the `kubectl delete pod` command. This action emulates a failing application or unexpected pod unavailability.

```
[user@host ~]$ kubectl delete pod \
do100-versioned-hello-POD_ID
```

This example terminates the pod named `do100-versioned-hello-76c4494b5d-qtf9`. Use a pod name from your list of pods, which are displayed in the terminal that is running the `kubectl get pods -w` command.

- 3.2. In the terminal running the `kubectl get pods -w` command, observe that the deleted pod changed to the Terminating status.

```
[user@host ~]$ kubectl get pods -w
NAME                               READY   STATUS            RESTARTS   AGE
do100-versioned-hello-76c4494b5d-4ldff 1/1    Running           0          51m
do100-versioned-hello-76c4494b5d-qtf9  0/1    Pending           0          0s
do100-versioned-hello-76c4494b5d-qtf9  0/1    Pending           0          0s
do100-versioned-hello-76c4494b5d-qtf9  0/1    ContainerCreating 0          0s
do100-versioned-hello-76c4494b5d-qtf9  1/1    Running           0          2s
do100-versioned-hello-76c4494b5d-qtf9  1/1    Terminating       0          97m
...output omitted...
```

Immediately after the pod becomes unavailable, Kubernetes creates a new replica:

```
[user@host ~]$ kubectl get pods -w
NAME                               READY   STATUS            RESTARTS   AGE
do100-versioned-hello-76c4494b5d-4ldff 1/1    Running           0          51m
do100-versioned-hello-76c4494b5d-qtf9  0/1    Pending           0          0s
do100-versioned-hello-76c4494b5d-qtf9  0/1    Pending           0          0s
do100-versioned-hello-76c4494b5d-qtf9  0/1    ContainerCreating 0          0s
do100-versioned-hello-76c4494b5d-qtf9  1/1    Running           0          2s
do100-versioned-hello-76c4494b5d-qtf9  1/1    Terminating       0          97m
do100-versioned-hello-76c4494b5d-8qmk8  0/1    Pending           0          0s
do100-versioned-hello-76c4494b5d-8qmk8  0/1    Pending           0          0s
do100-versioned-hello-76c4494b5d-8qmk8  0/1    ContainerCreating 0          1s
do100-versioned-hello-76c4494b5d-qtf9  0/1    Terminating       0          97m
do100-versioned-hello-76c4494b5d-8qmk8  1/1    Running           0          2s
...output omitted...
```



### Note

Note that the `Terminating` and `Pending` status might appear many times in the output. This repetition reflects the fact that those statuses are aggregating status for another fine-grained status for the deployment.

- ▶ 4. Deploy a new version of the application and observe the default deployment `rollingUpdate` strategy.
  - 4.1. Edit the deployment and update the container image version from `v1.0-external` to `v1.1-external`. Use the `kubectl edit deployment` command to edit the Deployment manifest:

```
[user@host ~]$ kubectl edit deployment do100-versioned-hello
```

Look for the `image: quay.io/redhattraining/do100-versioned-hello:v1.0-external` entry and change the image tag from `v1.0-external` to `v1.1-external`. Save the changes and exit the editor.

- 4.2. Analyze the status timeline for the pods. Observe how Kubernetes orchestrates the termination and deployment of the pods. The maximum unavailability is zero pods (25% of 2 pods, rounding down), so there must always be at least two available replicas. The maximum surge is 1 (20% of 2 pods, rounding up), so Kubernetes will create new replicas one by one.

NAME	READY	STATUS	RESTARTS	AGE
<i>...output omitted...</i>				
do100-versioned-hello-76c4494b5d-qtfs9	1/1	Running	0	2s
<i>...output omitted...</i>				
<b>do100-versioned-hello-76c4494b5d-8qmk8</b>	<b>1/1</b>	<b>Running</b>	<b>0</b>	<b>2s</b> <b>①</b>
<i>...output omitted...</i>				
<b>do100-versioned-hello-76c4494b5d-7729k</b>	<b>0/1</b>	<b>Pending</b>	<b>0</b>	<b>0s</b> <b>②</b>
do100-versioned-hello-76c4494b5d-7729k	0/1	Pending	0	0s
do100-versioned-hello-76c4494b5d-7729k	0/1	ContainerCreating	0	0s
<b>do100-versioned-hello-76c4494b5d-7729k</b>	<b>1/1</b>	<b>Running</b>	<b>0</b>	<b>1s</b> <b>③</b>
<b>do100-versioned-hello-77fc5857fc-qtfs9</b>	<b>1/1</b>	<b>Terminating</b>	<b>0</b>	<b>4m46s</b> <b>④</b>
<b>do100-versioned-hello-76c4494b5d-vrlkw</b>	<b>0/1</b>	<b>Pending</b>	<b>0</b>	<b>0s</b> <b>⑤</b>
do100-versioned-hello-76c4494b5d-vrlkw	0/1	Pending	0	0s
do100-versioned-hello-76c4494b5d-vrlkw	0/1	ContainerCreating	0	0s
do100-versioned-hello-77fc5857fc-qtfs9	0/1	Terminating	0	4m47s
do100-versioned-hello-77fc5857fc-qtfs9	0/1	Terminating	0	4m48s
do100-versioned-hello-77fc5857fc-qtfs9	0/1	Terminating	0	4m48s
<b>do100-versioned-hello-76c4494b5d-vrlkw</b>	<b>1/1</b>	<b>Running</b>	<b>0</b>	<b>2s</b> <b>⑥</b>
<b>do100-versioned-hello-77fc5857fc-8qmk8</b>	<b>1/1</b>	<b>Terminating</b>	<b>0</b>	<b>4m49s</b> <b>⑦</b>
do100-versioned-hello-77fc5857fc-8qmk8	0/1	Terminating	0	4m50s

- ① Rollout starts. At this point, two v1.0 replicas are available.
- ② Kubernetes creates a single new v1.1 replica, as `maxSurge` limits the over-deployment to one.
- ③ The new v1.1 replica is running and becomes available.
- ④ There are three pods available, so Kubernetes terminates the `do100-versioned-hello-77fc5857fc-qtfs9` pod (v1.0 replica).
- ⑤ After starting the termination, the number of available pods is two, so Kubernetes starts a new v1.1 replica.
- ⑥ The new v1.1 replica is running and becomes available.
- ⑦ Again, three pods are available, so Kubernetes terminates the last v1.0 replica.

## Finish

Delete the deployment to clean your cluster. Kubernetes automatically deletes the associated pods.

```
[user@host ~]$ kubectl delete deployment do100-versioned-hello
deployment.apps "do100-versioned-hello" deleted
```

Observe in the other terminal that Kubernetes automatically terminates all pods associated with the deployment:

```
[user@host ~]$ kubectl get pods -w
NAME                               READY   STATUS        RESTARTS   AGE
...output omitted...
do100-versioned-hello-77fc5857fc-8qmk8   0/1    Terminating   0          4m50s
do100-versioned-hello-76c4494b5d-7729k   1/1    Terminating   0          5m35s
do100-versioned-hello-76c4494b5d-vrlkw   1/1    Terminating   0          5m35s
do100-versioned-hello-76c4494b5d-7729k   0/1    Terminating   0          5m35s
do100-versioned-hello-76c4494b5d-vrlkw   0/1    Terminating   0          5m35s
...output omitted...
```

Press **Ctrl+C** to exit the `kubectl get pods -w` command.

This concludes the guided exercise.

## ► Quiz

# Deploying Managed Applications

Choose the correct answers to the following questions:

► 1. **Which of the following statements about the Deployment resource is correct?**

(Choose one.)

- a. The Deployment and the Pod resources are identical because a Deployment resource creates one or more Pod resources.
- b. The Deployment resource can only spawn more than one replica of the application.
- c. If a Deployment resource manages an application that crashes, the Deployment resource restarts the application.
- d. If an unmanaged application pod crashes, Kubernetes restarts the application.

► 2. **Which two of the following statements about the provided YAML manifest are correct? (Choose two.)**

```
apiVersion: apps/v1
kind: Pod
metadata:
  app: do100b
  name: do100b
spec:
  replicas: 1
  selector:
    matchLabels:
      app: do100b
  template:
    metadata:
      labels:
        app: do100b
    spec:
      containers:
        - name: do100b
```

- a. The YAML manifest is missing the image of the application.
- b. The YAML manifest is correct.
- c. The YAML manifest contains an incorrect Kind
- d. The YAML manifest contains an incorrect label. The label on the Deployment resource does not match the label on the Pod resource.

► 3. **Which of the following statements about the deployment strategy of the Deployment resource is correct? (Choose one.)**

- a. The deployment strategy configuration enables developers to implement routing strategies, such as blue-green deployment or A/B deployment.
- b. The Deployment resource does not enable configuring a deployment strategy. Each Pod resource contains a deployment strategy configuration.
- c. The RollingUpdate deployment strategy terminates all application pods, and recreates the new application pods.
- d. The Deployment resource enables you to configure the RollingUpdate or Recreate deployment strategies.

► 4. **Based on the following information, which deployment strategy is most suitable for updating the build system? (Choose one.)**

You are a DevOps engineer responsible for updating of the internal build system application to a new version.

The build system can experience downtime.

However, it is important for the build system to be updated as quickly as possible.

The tool system has more than 100 replicas.

The update is thoroughly tested.

- a. The Recreate strategy, because that is the fastest update strategy and the application can experience downtime. Additionally, the new application is well tested.
- b. The RollingUpdate strategy, because that is the fastest update strategy without causing downtime.
- c. The RollingUpdate strategy, because that is the only strategy for large applications. Each Pod resource contains a deployment strategy configuration.
- d. The Canary strategy, because we need to verify the new version before deploying it in a production environment.

## ► Solution

# Deploying Managed Applications

Choose the correct answers to the following questions:

► 1. **Which of the following statements about the Deployment resource is correct?**

(Choose one.)

- a. The Deployment and the Pod resources are identical because a Deployment resource creates one or more Pod resources.
- b. The Deployment resource can only spawn more than one replica of the application.
- c. If a Deployment resource manages an application that crashes, the Deployment resource restarts the application.
- d. If an unmanaged application pod crashes, Kubernetes restarts the application.

► 2. **Which two of the following statements about the provided YAML manifest are correct? (Choose two.)**

```
apiVersion: apps/v1
kind: Pod
metadata:
  app: do100b
  name: do100b
spec:
  replicas: 1
  selector:
    matchLabels:
      app: do100b
  template:
    metadata:
      labels:
        app: do100b
    spec:
      containers:
        - name: do100b
```

- a. The YAML manifest is missing the image of the application.
- b. The YAML manifest is correct.
- c. The YAML manifest contains an incorrect Kind
- d. The YAML manifest contains an incorrect label. The label on the Deployment resource does not match the label on the Pod resource.

► 3. **Which of the following statements about the deployment strategy of the Deployment resource is correct? (Choose one.)**

- a. The deployment strategy configuration enables developers to implement routing strategies, such as blue-green deployment or A/B deployment.
- b. The Deployment resource does not enable configuring a deployment strategy. Each Pod resource contains a deployment strategy configuration.
- c. The RollingUpdate deployment strategy terminates all application pods, and recreates the new application pods.
- d. The Deployment resource enables you to configure the RollingUpdate or Recreate deployment strategies.

► 4. **Based on the following information, which deployment strategy is most suitable for updating the build system? (Choose one.)**

You are a DevOps engineer responsible for updating of the internal build system application to a new version.

The build system can experience downtime.

However, it is important for the build system to be updated as quickly as possible.

The tool system has more than 100 replicas.

The update is thoroughly tested.

- a. The Recreate strategy, because that is the fastest update strategy and the application can experience downtime. Additionally, the new application is well tested.
- b. The RollingUpdate strategy, because that is the fastest update strategy without causing downtime.
- c. The RollingUpdate strategy, because that is the only strategy for large applications. Each Pod resource contains a deployment strategy configuration.
- d. The Canary strategy, because we need to verify the new version before deploying it in a production environment.

# Summary

---

In this chapter, you learned:

- Kubernetes, as a container manager platform, controls the deployment of containers.
- Deployment resources declare the images, replicas and other desired deployment information. Kubernetes updates the state of the application to match the desired status.
- Changes on the application state (such as a container stopping unexpectedly) or in the desired state (such as the desired number of replicas) trigger new container deployments.
- Deployment resources also define the strategy for updating pods, either RollingUpdate or Recreate.

## Chapter 2

# Configuring Networking in Kubernetes

### Goal

Introduce communication between Kubernetes applications and the rest of the world.

### Objectives

- Enable intra-pod network communications for applications deployed in Kubernetes, and learn how to keep communication up even with automatic deployments.
  - Expose service-backed applications to clients outside the Kubernetes cluster.
- 
- Exposing Applications for External Access (and Guided Exercise,Quiz)
  - Exposing Applications for External Access: Ingress Resources (and Guided Exercise,Quiz)

### Sections

# Exposing Applications for External Access

## Objectives

After completing this section, you should be able to enable intra-pod network communications for applications deployed in Kubernetes, and learn how to keep communication up even with automatic deployments.

## Kubernetes Networking

When pods are created, they are assigned an IP address. You use this IP to access the pod from anywhere within the Kubernetes cluster. Containers inside a pod share the same network space, which means that, within the pod, containers can communicate with each other by using the `localhost` address.

A Kubernetes cluster might be split across different nodes. A node is a physical machine where resources run. A cluster is a logical view of a set of nodes. These nodes are different machines, but they work together as a logical unit. This makes it easier to work with different machines at the same time because you can simply deploy resources to the cluster and not to individual nodes.

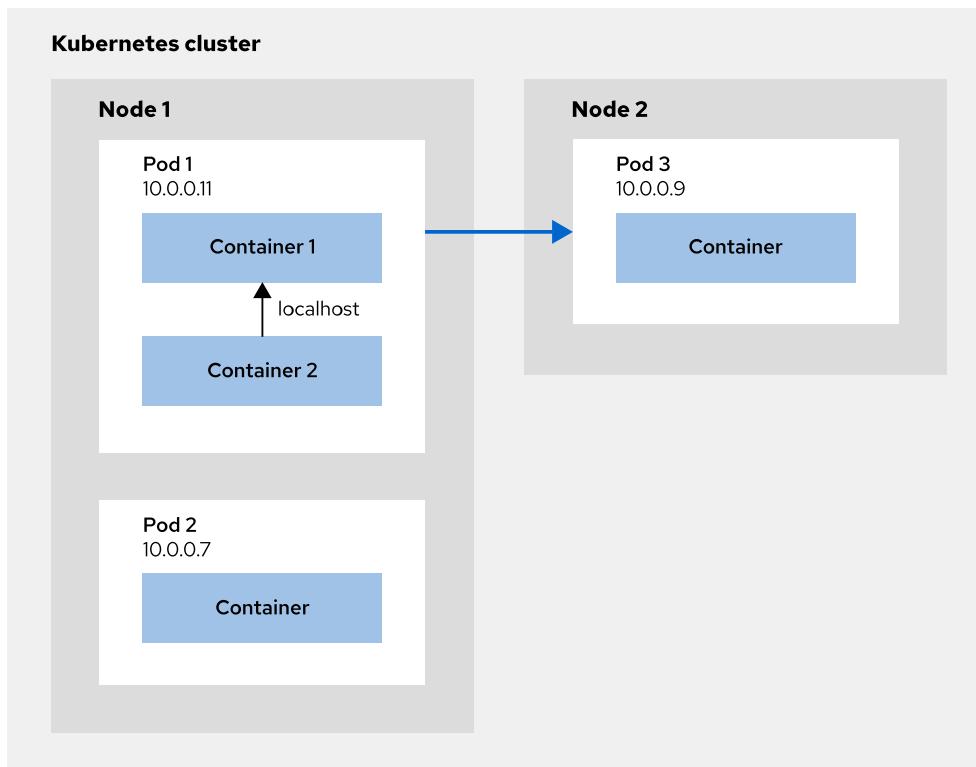


Figure 2.1: Networking in a Kubernetes cluster

## Introducing Kubernetes Services

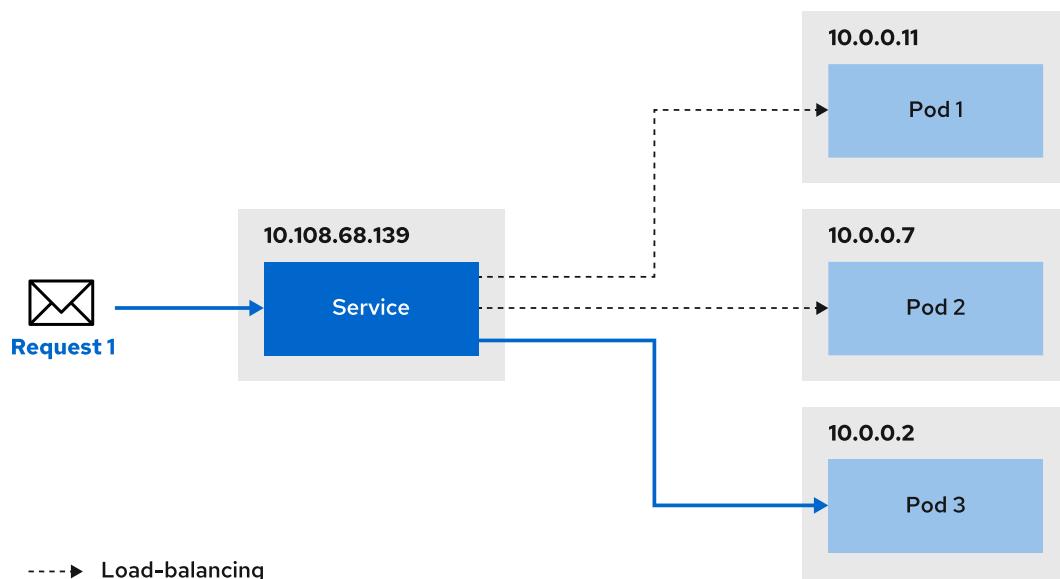
In a real world environment, deployments are performed on a daily basis. When bugs are fixed or new features are added to an application, a new image version is created and deployed. This means that pods are constantly created and destroyed (the pods of the older version are removed and new pods are allocated for the newer version).

At the same time, applications usually have several replicas and traffic is split across the replicas. This ensures that no single replica is overworked. This is called **load-balancing**.

In both use cases, the problem is the same: you need a way to reach the pods regardless of the machine where they are located. To solve this, Kubernetes introduces the concept of **Service**.

A service is an abstraction that defines the access to a set of pods. By using a service, you don't access pods directly through their private IP addresses. Instead, a service targets several pods based on certain criteria (for example, a label) and forwards any requests to **one** of the pods matching that criteria.

In other words, a service allows you to group pods with a logical relationship and it allows you to reach them in a reliable way. At the same time, it implements a load-balancing mechanism among the pods that it targets.

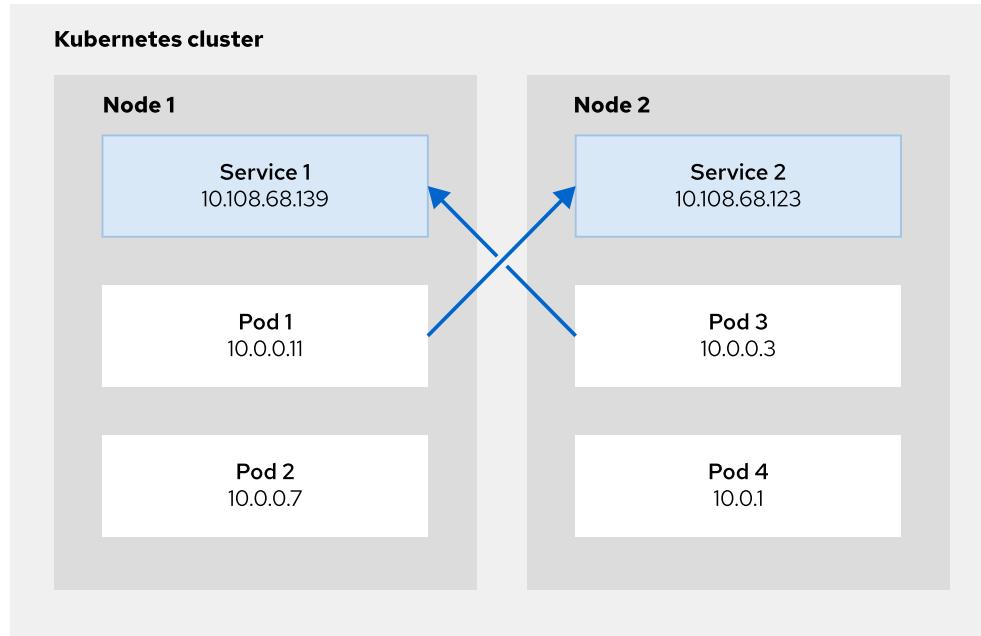


**Figure 2.2: A service redirecting traffic to pod replicas**

For example, if you want to have three replicas of your application then three pods will be created. If you create a service that targets these pods, then the service receives any incoming requests and it routes it to one of them.

By default, a service is given a cluster-internal IP address, which is only valid within the cluster. This type of service is called **ClusterIP**. This means that pods deployed in the cluster can make requests to the service by using the ClusterIP.

The following diagram illustrates the communication between pods and services. For example, Pod 1 uses the ClusterIP of Service 2 to make requests to the service.



**Figure 2.3: Pod communication using services**

If you want to expose the service externally, then you can use other types of services such as `NodePort` or `LoadBalancer`. However, the most common way to expose a service outside of your cluster is by using another Kubernetes resource called `Ingress`. `Ingress` is covered in upcoming sections of this course.

## Creating Kubernetes Services

When creating a service, it is necessary to define the port that the service will serve on. This port is mapped to a `target port` inside the pod that the service targets. Incoming requests to the service `port` are forwarded to the `target port` in the pod. If no `target port` is provided, then the `port` value is used.

There are two ways to create a service in Kubernetes:

- **Using `kubectl expose`**

The easiest way to create a service is by using the `kubectl expose` command.

```
[user@host ~]$ kubectl expose deployment deployment-name \
--port=8081 --name=service-name --target-port=3000
```

The previous command creates a service named `service-name`, which targets deployment `deployment-name`. It listens on port 8081 and it points to port 3000 inside the pod.

Use the command `kubectl get service` to list the services available. The output will provide you with information such as the ClusterIP (IP only valid within the Kubernetes cluster) and the port used to access the service. A sample output might look like this:

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
service-name	ClusterIP	10.108.68.139	<none>	8081/TCP	3s

- **Applying a manifest**

An approach in line with the DevOps principles is creating services through a manifest. The following sample creates a service named `nginx-service` and targets any pod with the label `app: nginx`. The service listens for requests in port 8081 and forwards them to port 3000 inside the pod. Because the manifest does not include the `type` field, it creates a service with type `ClusterIP`.

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-service ①
spec:
  selector: ②
    app: nginx
  ports: ③
    - protocol: TCP
      port: 8081 ④
      targetPort: 3000 ⑤
```

- ① Name of the service
- ② Labels used to select the target pods
- ③ Port mapping
- ④ The port that the service will serve on
- ⑤ The port inside the pod where requests are forwarded

## Discovering Kubernetes Services

A service abstracts your application from knowing the exact location of the pods you are using, but you still need to know the IP of the service to use it from your application. Directly using the IP is a bad idea because if the IP changes in the future, then you would need to manually update it in your application. To avoid this, Kubernetes provides two ways to discover services:

- **Environment variables**

By default, when a service is created, Kubernetes injects some environment variables in pods within the same namespace. These variables follow the pattern:

```
SERVICE-NAME_VARIABLE-NAME
```

If you have a service named `nginx-provider`, that generates the following variables (non-exhaustive) then you can simply inject these environment variables into your application:

- `NGINX_PROVIDER_SERVICE_HOST`, which contains the IP address of the Service. For example, `10.0.0.11`
- `NGINX_PROVIDER_SERVICE_PORT`, which contains the port where Service listens on. For example, `6379`

However, your application tries to fetch the environment variables only on start-up. This means that if the value of the variable changes (for example, a service gets a different IP) after your application has started, then your application is not notified and it references an invalid value (the previous IP address of the service). The same happens if the service is created after your application boots-up.

- **DNS**

Given the limitations of the Kubernetes built-in environment variables, the preferred way of accessing services from your application is using DNS.

Every service in the cluster is assigned a DNS name, which matches with the service's lower cased name. This allows applications to access services using always the same reference. The default FQDN follows the pattern:

```
service.namespace.svc.cluster.local
```

However, it is possible to avoid this long form. The DNS server also resolves the following hosts:

- `service.namespace.cluster.local`
- `service.namespace`
- `service` (in this case, Kubernetes expects the service to be in the same namespace)

For example, if you have a service named `nginx-service` that exposes an HTTP endpoint in the default HTTP port (80), then you can use `http://nginx-service` **if your application is in the same namespace as the service**. If the service was in a namespace named `nginx-apps`, then you use `http://nginx-service.nginx-apps`.



## References

### Services

<https://kubernetes.io/docs/concepts/services-networking/service/>

#### Discovering services

<https://kubernetes.io/docs/concepts/services-networking/service/#discovering-services>

## ► Guided Exercise

# Exposing Applications for External Access

In this exercise you will deploy two apps in different namespaces. They communicate by using the built-in Kubernetes DNS resolution system.

### Outcomes

You should be able to:

- Create a service using `kubectl expose`
- Create a service using a manifest
- Use DNS resolution for service communication

### Before You Begin

Ensure that:

- Minikube and `kubectl` are running on your machine
- You have cloned the `D0100x-apps` repository
- You have executed the `setup` script

## Instructions

To illustrate how communication is handled in Kubernetes, you use two applications.

The `name-generator` app produces random names that can be consumed in the `/random-name` endpoint.

The `email-generator` app produces random emails that can be consumed in the `/random-email` endpoint.

The `email-generator` app consumes `name-generator` to include a random name in the emails that it generates.

Make sure your `kubectl` context uses the namespace `username-dev`. This allows you to execute `kubectl` commands directly into that namespace.

**Note**

This course uses the backslash character (\) to break long commands. On Linux and macOS, you can use the line breaks.

On Windows, use the backtick character (`) to break long commands. Alternatively, do not break long commands.

Refer to *Orientation to the Classroom Environment* for more information about long commands.

```
[user@host D0100x-apps]$ kubectl config set-context --current \
--namespace=username-dev
```

- 1. Deploy the name-generator app in the *username-dev* namespace.

- 1.1. Open a command-line terminal. In the *D0100x-apps* repository, navigate to the name-generator folder.
- 1.2. Use the `kubectl apply` command to create a Deployment from the manifest located in the `kubernetes` directory. It creates three replicas of the name-generator app by using the `quay.io/redhattraining/do100-name-generator:v1.0-external` image.

```
[user@host name-generator]$ kubectl apply -f kubernetes/deployment.yml
deployment.apps/name-generator created
```

- 1.3. List the deployments to verify it has been created successfully. Use the command `kubectl get deployment`.

```
[user@host name-generator]$ kubectl get deployment
NAME          READY   UP-TO-DATE   AVAILABLE   AGE
name-generator 3/3      3           3           90m
```

- 2. Create a Service for the deployment of the name-generator app by using the `kubectl expose` command.

- 2.1. Using the deployment name, expose the service at port number 80. The following command creates a service that forwards requests on port 80 for the DNS name `name-generator.namespace.local-domain` to containers created by the name-generator deployment on port 8080.

```
[user@host name-generator]$ kubectl expose deployment name-generator \
--port 80 --target-port=8080
service/name-generator exposed
```

- 2.2. List the services to verify that the name-generator service has been created successfully. Use the command `kubectl get service`.

```
[user@host name-generator]$ kubectl get service
NAME           TYPE      CLUSTER-IP   EXTERNAL-IP   PORT(S)    AGE
name-generator  ClusterIP  10.98.55.248 <none>        80/TCP     31s
```

- 3. Review the code of the `email-generator` to see how the request to the `name-generator` is made. Deploy the app in the `username-dev` namespace.
- 3.1. In the `D0100x-apps` repository, navigate to the `email-generator` folder.
  - 3.2. In the `app` directory, open the `server.js` file. The `server.js` file is a NodeJS application, which exposes the endpoint `/random-email` on the `8081` port.
  - 3.3. In the same folder, open the `generate-email.js` file. The `generateEmail` method generates a random email by making an HTTP request to the `name-generator` service.
  - 3.4. The `getNameFromExternalService` method performs the actual HTTP request. The host, which is the `name-generator` service name, is defined in the `NAME_GENERATOR_URL` variable.
  - 3.5. On the command line, return to the `email-generator` folder.
  - 3.6. Apply the Deployment manifest in the `username-dev` namespace. It is located in the `kubernetes` folder and creates three replicas of the `email-generator` app by using the `quay.io/redhattraining/do100-email-generator:v1.0-externalimage`.

```
[user@host email-generator]$ kubectl apply -f kubernetes/deployment.yml
deployment.apps/email-generator created
```

- 3.7. List the deployments in the namespace to verify it has been created successfully. Use the command `kubectl get deployment`.

```
[user@host email-generator]$ kubectl get deployment
NAME          READY   UP-TO-DATE   AVAILABLE   AGE
email-generator  3/3      3           3           5s
name-generator  3/3      3           3           66m
```

- 4. Create a service for the deployment of the `email-generator` app by using a manifest.
- 4.1. Apply the Service manifest in the `username-dev` namespace. It is located in the `kubernetes` folder. Use the `kubectl apply` command.  
This command exposes the service in the `80` port and targets port `8081`, which is where the `email-generator` app serves.

```
[user@host email-generator]$ kubectl apply -f kubernetes/service.yml
service/email-generator created
```

- 4.2. List the services to verify that the `email-generator` service has been created successfully. Use the command `kubectl get service`.

```
[user@host email-generator]$ kubectl get service
NAME           TYPE      CLUSTER-IP   EXTERNAL-IP   PORT(S)    AGE
email-generator  ClusterIP  10.108.68.139 <none>        80/TCP     3s
name-generator   ClusterIP  10.109.14.167 <none>        80/TCP     68m
```

- 5. Verify that everything works properly by making an HTTP request to the `email-generator` app from the `username-stage` namespace. The result should contain a name plus some numbers at the end.
- 5.1. To make a request to the `email-generator` app from another namespace, you use the Kubernetes DNS resolution pattern `service-name.namespace`. In this case, the host is `email-generator.username-dev`.
  - 5.2. Create a temporary pod that enables you to make a request to the `email-generator` application. Run the following command, which provides you with a terminal to execute `curl`.

```
[user@host email-generator]$ kubectl run -n username-stage \
curl -it --rm \
--image=registry.access.redhat.com/ubi8/ubi-minimal -- sh
```

Note that:

- The command creates a pod named `curl` in the `username-stage` namespace.
- The pod contains one container that uses the `registry.access.redhat.com/ubi8/ubi-minimal` container image.
- After Kubernetes creates the pod, you create an interactive remote shell session into the pod.
- When you exit out of the interactive session, Kubernetes terminates the pod.

The command might take some time to execute. If you see the message `If you don't see a command prompt, try pressing Enter.`, then press `Enter` on your keyboard and the terminal opens.

- 5.3. In the terminal, make an HTTP request to the `email-generator` service by using `curl`. Because the service runs on the default HTTP port (80), you do not need to specify the port. You can also omit the local DNS domain.

```
[user@host email-generator]$ curl \
http://email-generator.username-dev/random-email
```

You should see a response in JSON format similar to this:

```
{"email":"Susan487@host"}
```

- 5.4. Type `exit` to exit the terminal. The pod used to make the request is automatically deleted.

## Finish

Remove all resources used in this exercise.

You can delete all resources in the namespace with the following command:

```
[user@host email-generator]$ kubectl delete all --all
pod "email-generator-ff5fdf658-bz8v2" deleted
pod "email-generator-ff5fdf658-k6ln6" deleted
pod "email-generator-ff5fdf658-pn466" deleted
pod "name-generator-9744675d-4kmp9" deleted
pod "name-generator-9744675d-grw9g" deleted
pod "name-generator-9744675d-tlpz9" deleted
service "email-generator" deleted
service "name-generator" deleted
deployment.apps "email-generator" deleted
deployment.apps "name-generator" deleted
```

Alternatively, you can delete the resources individually. Delete both the `email-generator` and `name-generator` services:

```
[user@host email-generator]$ kubectl delete service email-generator
service "email-generator" deleted
[user@host email-generator]$ kubectl delete service name-generator
service "name-generator" deleted
```

Delete both the `email-generator` and `name-generator` deployments:

```
[user@host email-generator]$ kubectl delete deployment email-generator
deployment.apps "email-generator" deleted
[user@host email-generator]$ kubectl delete deployment name-generator
deployment.apps "name-generator" deleted
```

This concludes the guided exercise.

## ► Quiz

---

# Exposing Applications for External Access

- 1. Which two of the following statements about pod-to-pod networking are correct? (Choose two.)
- a. Multiple containers in a pod share network space. For example, such containers can communicate by using `localhost`.
  - b. A pod can reach other pods on the same Kubernetes node. Pods on different Kubernetes nodes remain unreachable.
  - c. A pod can reach other pods on the same Kubernetes node. Pods on different Kubernetes nodes require the `Service` resource to be reachable.
  - d. Kubernetes assigns each pod an IP address. Any pod can reach any other pod within a Kubernetes cluster by using the assigned IP address.
- 2. Which two of the following statements about the Kubernetes Service resource are correct? (Choose two.)
- a. Kubernetes uses only the `LoadBalancer` service type.
  - b. The `ClusterIP` service type routes requests to a set of pods in a round-robin way to load balance the requests.
  - c. A service cannot target less than three replicas. This is known as the N-1 rule.
  - d. You can use three Kubernetes service types: `ClusterIP`, `LoadBalancer`, and `NodePort`. `ClusterIP` is the default service type which exposes traffic within the Kubernetes cluster.

- 3. Based on the provided Service resource manifest, which of the following statements are correct? (Choose two.)

```
apiVersion: v1
kind: Service
metadata:
  name: camel-api
spec:
  selector:
    app: camel-api
  ports:
    - protocol: TCP
      port: 443
      targetPort: 8443
```

- a. The service listens for requests on port 8443.
  - b. The service listens for requests on port 443.
  - c. The service routes requests to a pod with the label app=camel-api to port 8443.
  - d. The service routes requests to a pod with the label app=camel-api to port 443.
- 4. Your application uses the following FQDN to discover a Service resource: `frontend-v1.build-stage.svc.cluster.local`. Which of the following statements are correct? (Choose two.)
- a. The service name is `frontend`.
  - b. The service is in the `build-stage.svc` namespace.
  - c. The service name is `frontend-v1`.
  - d. The service is in the `build-stage` namespace.

## ► Solution

---

# Exposing Applications for External Access

- 1. Which two of the following statements about pod-to-pod networking are correct? (Choose two.)
- a. Multiple containers in a pod share network space. For example, such containers can communicate by using `localhost`.
  - b. A pod can reach other pods on the same Kubernetes node. Pods on different Kubernetes nodes remain unreachable.
  - c. A pod can reach other pods on the same Kubernetes node. Pods on different Kubernetes nodes require the `Service` resource to be reachable.
  - d. Kubernetes assigns each pod an IP address. Any pod can reach any other pod within a Kubernetes cluster by using the assigned IP address.
- 2. Which two of the following statements about the Kubernetes Service resource are correct? (Choose two.)
- a. Kubernetes uses only the `LoadBalancer` service type.
  - b. The `ClusterIP` service type routes requests to a set of pods in a round-robin way to load balance the requests.
  - c. A service cannot target less than three replicas. This is known as the N-1 rule.
  - d. You can use three Kubernetes service types: `ClusterIP`, `LoadBalancer`, and `NodePort`. `ClusterIP` is the default service type which exposes traffic within the Kubernetes cluster.

- 3. Based on the provided Service resource manifest, which of the following statements are correct? (Choose two.)

```
apiVersion: v1
kind: Service
metadata:
  name: camel-api
spec:
  selector:
    app: camel-api
  ports:
    - protocol: TCP
      port: 443
      targetPort: 8443
```

- a. The service listens for requests on port 8443.
  - b. The service listens for requests on port 443.
  - c. The service routes requests to a pod with the label `app=camel-api` to port 8443.
  - d. The service routes requests to a pod with the label `app=camel-api` to port 443.
- 4. Your application uses the following FQDN to discover a Service resource: `frontend-v1.build-stage.svc.cluster.local`. Which of the following statements are correct? (Choose two.)
- a. The service name is `frontend`.
  - b. The service is in the `build-stage.svc` namespace.
  - c. The service name is `frontend-v1`.
  - d. The service is in the `build-stage` namespace.

# Exposing Applications for External Access: Ingress Resources

---

## Objectives

After completing this section, you should be able to expose service-backed applications to clients outside the Kubernetes cluster.

## Kubernetes Ingress

Kubernetes assigns IP addresses to pods and services. Pod and service IP addresses are not usually accessible outside of the cluster. Unless prevented by network policies, the Kubernetes cluster typically allows internal communication between pods and services. This internal communication allows application pods to interact with services that are not externally accessible, such as database services.

For a web application that should be accessible to external users, you must create a Kubernetes ingress resource. An ingress maps a domain name, or potentially a URL, to an existing service. On its own, the ingress resource does not provide access to the specified host or path. The ingress resource interacts with a Kubernetes ingress controller to provide external access to a service over HTTP or HTTPS.

## Kubernetes Ingress Controller

Kubernetes ingress controllers act as load balancers for HTTP and HTTPS requests coming in to your cluster. Because ingress controllers can be specific to an environment, Kubernetes clusters are not configured to use an ingress controller by default.

As a developer, you cannot choose the ingress controller used by your environment you also cannot configure it.

Operations teams will install and configure an ingress controller appropriate to their environment. This includes configuring the ingress controller based on the networking characteristics of your environment. Most cloud providers and Kubernetes distributions implement their own ingress controllers, tailored for their products and network environments.

Local and self-managed Kubernetes distributions tend to use ingress controllers offered by open source projects, network vendors or sample ingress controllers provided by Kubernetes. Find a list of ingress controllers provided by upstream Kubernetes in the References section.

## Ingress Resource Configuration

One of the main things that you must specify in your ingress resource configuration is the host name used to access your application. This host name is part of the cluster configuration as it comes defined by factors external to the cluster, such as network characteristics and DNS services. The operations team must provide the cluster host name to developers.

Production deployments must have DNS records pointing to the Kubernetes cluster. Some Kubernetes distributions use wildcard DNS records to link a family of host names to the same Kubernetes cluster. A wildcard DNS record is a DNS record that, given a parent wildcard domain, maps all its subdomains to a single IP. For example, a wildcard DNS record might map the

wildcard domain \*.example.com to the IP 10.0.0.8. DNS requests for Subdomains such as hello.example.com or myapp.example.com will obtain 10.0.0.8 as a response.

The following is an example of an ingress resource.

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: hello ①
spec:
  rules:
    - host: hello.example.com ②
      http:
        paths:
          - path: / ③
            pathType: Prefix ④
            backend:
              service:
                name: hello ⑤
                port:
                  number: 8080 ⑥
```

- ① The name of the ingress resource.
- ② The host name used by external users to access your application.
- ③ This value is used in combination with pathType to determine if the URL request matches any of the accepted paths. A path value of / with the pathType value of Prefix is the equivalent of a wildcard that matches any path.
- ④ This value is used in combination with path to determine if the URL matches any of the accepted paths. A pathType of Prefix offers a bit more flexibility allowing for matches where the path and the requested URL can contain either a trailing / or not. A pathType of Exact requires the requested URL to exactly match the path value.
- ⑤ The name of the service to which requests are redirected.
- ⑥ The port number on which the service listens.

## Testing Your Ingress

If you have an application already running in your Kubernetes cluster, then you can test the ingress resource by verifying the external access to the application. The test checks that the ingress resource successfully configures with the ingress controller to redirect the HTTP or HTTPS request.

After using either the `kubectl create` command or the `kubectl apply` command to create an ingress resource, use a web browser to access your application URL. Browse to the host name and the path defined in the ingress resource and verify that the request is forwarded to the application and the browser get the response:



Optionally you can use the `curl` command to perform simple tests.

```
[user@host ~]$ curl hello.example.com
<html>
<body>
    <h1>Hello, world from nginx!</h1>
</body>
</html>
```

If the browser does not obtain the expected response then verify that:

- The host name and paths are the ones used in the ingress resource.
- Your system can translate the host name to the IP address for the ingress controller (via your hosts file or a DNS entry).
- The ingress resource is available and its information is correct.
- If applicable, verify that the ingress controller is installed and running in your cluster.



## References

### Ingress

<https://kubernetes.io/docs/concepts/services-networking/ingress/>

### Ingress Controllers

<https://kubernetes.io/docs/concepts/services-networking/ingress-controllers/>

## ► Guided Exercise

# Exposing Applications for External Access: Ingress Resources

In this exercise you will provide external access to a service running inside your Kubernetes cluster.

### Outcomes

You should be able to:

- Verify that the service IP address and the associated pod IP addresses for an application are not accessible outside of the cluster.
- Create an ingress resource to provide external access to an application service.
- Confirm that the ingress redirects traffic to the service.

### Before You Begin

You need a working Kubernetes cluster with the following:

- Your `kubectl` command configured to communicate with the cluster.
- An ingress controller enabled in your cluster and the associated domain name mapping.
- Your Kubernetes context referring to your cluster and using the `username-dev` namespace.

## Instructions

- 1. Deploy a sample `hello` application. The `hello` app displays a greeting and its local IP address. When running under Kubernetes, this is the IP address assigned to its pod.
- Create a new deployment named `hello` that uses the container image located at `quay.io/redhattraining/do100-hello-ip:v1.0-external` in the `username-dev` namespace. Configure the deployment to use three pods.
- 1.1. Create the `hello` deployment with three replicas. Use the container image located at `quay.io/redhattraining/do100-hello-ip:v1.0-external`. This container image simply displays the IP address of its associated pod.



#### Note

This course uses the backslash character (\) to break long commands. On Linux and macOS, you can use the line breaks.

On Windows, use the backtick character (`) to break long commands. Alternatively, do not break long commands.

Refer to *Orientation to the Classroom Environment* for more information about long commands.

```
[user@host ~]$ kubectl create deployment hello \
--image quay.io/redhattraining/do100-hello-ip:v1.0-external \
--replicas 3
deployment.apps/hello created
```

Run the `kubectl get pods -w` command to verify that three pods are running. Press `Ctrl+C` to exit the `kubectl` command after all three `hello` pods display the `Running` status.

```
[user@host ~]$ kubectl get pods -w
NAME           READY   STATUS            RESTARTS   ...
hello-5f87ddc987-76hxn  0/1    ContainerCreating  0          ...
hello-5f87ddc987-j8bbv   0/1    ContainerCreating  0          ...
hello-5f87ddc987-ndbk5  0/1    ContainerCreating  0          ...
hello-5f87ddc987-j8bbv  1/1    Running          0          ...
hello-5f87ddc987-ndbk5  1/1    Running          0          ...
hello-5f87ddc987-76hxn  1/1    Running          0          ...
```

- 1.2. Create a service for the `hello` deployment that redirects to pod port 8080.

Run the `kubectl expose` command to create a service that redirects to the `hello` deployment. Configure the service to listen on port 8080 and redirect to port 8080 within the pod.

```
[user@host ~]$ kubectl expose deployment/hello --port 8080
service/hello exposed
```

Verify that Kubernetes created the service:

```
[user@host ~]$ kubectl get service/hello
NAME      TYPE      CLUSTER-IP      EXTERNAL-IP      PORT(S)      ...
hello     ClusterIP  10.103.208.46  <none>          8080/TCP      ...
```

Note that the IP associated to the service is private to the Kubernetes cluster. You can not access that IP directly.

- 2. Create an ingress resource that directs external traffic to the `hello` service.

- 2.1. Use a text editor to create a file in your current directory named `ingress-hello.yml`.

Create the `ingress-hello.yml` file with the following content. Ensure correct indentation (using spaces rather than tabs) and then save the file.

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: hello
  labels:
    app: hello
spec:
  rules:
    - host: INGRESS-HOST
```

```

http:
  paths:
    - path: /
      pathType: Prefix
      backend:
        service:
          name: hello
          port:
            number: 8080

```

Replace *INGRESS-HOST* by the host name associated to your Kubernetes cluster, such as `hello.example.com` or `hello-username-dev.apps.sandbox.x8i5.p1.openshiftapps.com`.

The file at <https://github.com/RedHatTraining/DO100x-apps/blob/main/network/ingress-hello.yml> contains the correct content for the `ingress-hello.yml` file. You can download the file and use it for comparison.

- 2.2. Use the `kubectl create` command to create the ingress resource.

```
[user@host ~]$ kubectl create -f ingress-hello.yml
ingress.networking.k8s.io/hello created
```

- 2.3. Display information about the `hello` ingress. If the command does not display an IP address, then wait up to a minute and try running the command again.

```
[user@host ~]$ kubectl get ingress/hello
NAME      CLASS      HOSTS          ADDRESS      PORTS      ...
hello     <none>    hello.example.com  192.168.64.7  80        ...
```

The value in the `HOST` column matches the `host` line specified in your `ingress-hello.yml` file. Your IP address is likely different from the one displayed here.

- 3. Verify that the ingress resource successfully provides access to the `hello` service and the pods associated with the service.

- 3.1. Access the domain name specified by your ingress resource.

Use the `curl` command to access the domain name associated to your ingress controller. Repeat the same command multiple times and note the IP of the responding pod replica varies:

```
[user@host ~]$ curl hello.example.com
Hello from IP: 172.17.0.5
[user@host ~]$ curl hello.example.com
Hello from IP: 172.17.0.5
[user@host ~]$ curl hello.example.com
Hello from IP: 172.17.0.6
[user@host ~]$ curl hello.example.com
Hello from IP: 172.17.0.6
[user@host ~]$ curl hello.example.com
Hello from IP: 172.17.0.4
[user@host ~]$ curl hello.example.com
Hello from IP: 172.17.0.4
```

The `hello` ingress queries the `hello` service to identify the IP addresses of the pod endpoints. The `hello` ingress then uses round robin load balancing to spread the requests among the available pods, and each pod responds to the `curl` command with the pod IP address.

Optionally, open a web browser and navigate to the wildcard domain name. The web browser displays a message similar to the following.

```
Hello from IP: 172.17.0.6
```

Refresh your browser window to repeat the request and see different responses.



### Note

Because load balancers frequently create an association between a web client and a server (one of the `hello` pods in this case), reloading the web page is unlikely to display a different IP address. This association, sometimes referred to as a *sticky session*, does not apply to the `curl` command.

## Finish

Delete the created resources that have the `app=hello` label.

```
[user@host ~]$ kubectl delete deployment,service,ingress -l app=hello
deployment.apps "hello" deleted
service "hello" deleted
ingress.networking.k8s.io "hello" deleted
```

Verify that no resources with the `app=hello` label exist in the current namespace.

```
[user@host ~]$ kubectl get deployment,service,ingress -l app=hello
No resources found in username-dev namespace.
```

This concludes the guided exercise.

## ► Quiz

# Exposing Applications for External Access: Ingress Resources

Choose the correct answers to the following questions:

- ▶ 1. **Which of the following statements best describes an ingress controller? (Choose one.)**
  - a. The ingress controller acts as a load balancer and routes requests to services based on the Ingress resources.
  - b. Each Ingress resource creates a new ingress controller that is responsible for requests to that Ingress resource.
  - c. Ingress controller is a load balancer external to your Kubernetes cluster, typically in your disaster recovery data center.
  - d. Ingress controller is an optional add-on for better request tracing and latency.
  
- ▶ 2. **Which of the following commands creates an Ingress resource? Assume the names of resources are correct. (Choose one.)**
  - a. kubectl expose pod mypod
  - b. kubectl create -f ingress.yaml
  - c. kubectl expose deployment mydeployment
  - d. kubectl edit service myservice

► 3. Which of the following URLs will be matched by the provided Ingress resource manifest configuration? (Choose one.)

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: frontend-example
spec:
  rules:
    - host: frontend.com
      http:
        paths:
          - path: /
            pathType: Exact
            backend:
              service:
                name: frontend
                port:
                  number: 3000
```

- a. frontend.com
- b. frontend.com/example
- c. frontend-example.com
- d. None, because the resource is misconfigured

► 4. Which two of the following URLs will be matched by the provided Ingress resource manifest configuration? (Choose two.)

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: frontend-example
spec:
  rules:
    - host: frontend.com
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: frontend
                port:
                  number: 3000
```

- a. `frontend.com`
- b. `frontend.com/example`
- c. `frontend-example.com`
- d. None, because the resource is misconfigured

## ► Solution

# Exposing Applications for External Access: Ingress Resources

Choose the correct answers to the following questions:

- ▶ 1. **Which of the following statements best describes an ingress controller? (Choose one.)**
  - a. The ingress controller acts as a load balancer and routes requests to services based on the **Ingress** resources.
  - b. Each **Ingress** resource creates a new ingress controller that is responsible for requests to that **Ingress** resource.
  - c. Ingress controller is a load balancer external to your Kubernetes cluster, typically in your disaster recovery data center.
  - d. Ingress controller is an optional add-on for better request tracing and latency.
  
- ▶ 2. **Which of the following commands creates an Ingress resource? Assume the names of resources are correct. (Choose one.)**
  - a. kubectl expose pod mypod
  - b. kubectl create -f ingress.yaml
  - c. kubectl expose deployment mydeployment
  - d. kubectl edit service myservice

► 3. Which of the following URLs will be matched by the provided Ingress resource manifest configuration? (Choose one.)

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: frontend-example
spec:
  rules:
    - host: frontend.com
      http:
        paths:
          - path: /
            pathType: Exact
            backend:
              service:
                name: frontend
                port:
                  number: 3000
```

- a. **frontend.com**
- b. **frontend.com/example**
- c. **frontend-example.com**
- d. None, because the resource is misconfigured

► 4. Which two of the following URLs will be matched by the provided Ingress resource manifest configuration? (Choose two.)

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: frontend-example
spec:
  rules:
    - host: frontend.com
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: frontend
                port:
                  number: 3000
```

- a. frontend.com
- b. frontend.com/example
- c. frontend-example.com
- d. None, because the resource is misconfigured

# Summary

---

In this chapter, you learned:

- Kubernetes offers several virtual networks to enable communication between cluster nodes and between pods.
- The **service** resource abstracts internal pod communication so applications are not dependent of dynamic IPs and ports.
- Services discover other services by using environment variables injected by Kubernetes or, preferably, by using Kubernetes internal DNS service.
- Kubernetes allow exposing services to external networks using **ingress** resources and controllers.
- Ingress implementations vary on different Kubernetes distributions, but all of them link a subdomain name to a service.



## Chapter 3

# Customize Deployments for Application Requirements

### Goal

Advanced container management features.

### Objectives

- Leverage how to avoid applications overusing system resources.
- Review how Kubernetes evaluates application health status via probes and automatic application restart.
- Create Kubernetes resources holding application configuration and secrets, and how to make that configuration available to running applications.

### Sections

- Limiting Resource Usage (and Guided Exercise,Quiz)
- Proving Liveness, Readiness and Startup (and Guided Exercise,Quiz)
- Configuring Cloud Applications (and Guided Exercise,Quiz)

# Limiting Resource Usage

## Objectives

After completing this section, you should be able to leverage how to avoid applications overusing system resources.

## Defining Resource Requests and Limits for Pods

A pod definition can include both resource requests and resource limits:

### Resource requests

Used for scheduling and indicating that a pod cannot run with less than the specified amount of compute resources. The scheduler tries to find a node with sufficient compute resources to satisfy the requests.

### Resource limits

Used to prevent a pod from using up all compute resources from a node. The node that runs a pod configures the Linux kernel cgroups feature to enforce the pod's resource limits.

You should define resource requests and resource limits for each container in a deployment. If not, then the deployment definition will include a `resources: {}` line for each container.

Modify the `resources: {}` line to specify the desired requests and or limits. For example:

```
...output omitted...
spec:
  containers:
    - image: quay.io/redhattraining/hello-world-nginx:v1.0
      name: hello-world-nginx
      resources:
        requests:
          cpu: "10m"
          memory: 20Mi
        limits:
          cpu: "80m"
          memory: 100Mi
  status: {}
```



### Note

If you use the `kubectl edit` command to modify a deployment, then ensure you use the correct indentation. Indentation mistakes can result in the editor refusing to save changes. To avoid indentation issues, you can use the `kubectl set resources` command to specify resource requests and limits.

The following command sets the same requests and limits as the preceding example:

```
[user@host ~]$ kubectl set resources deployment hello-world-nginx \
--requests cpu=10m, memory=20Mi --limits cpu=80m, memory=100Mi
```

If a resource quota applies to a resource request, then the pod should define a resource request. If a resource quota applies to a resource limit, then the pod should also define a resource limit. Even without quotas, you should define resource requests and limits.

## Viewing Requests, Limits, and Actual Usage

Using the Kubernetes command-line interface, cluster administrators can view compute usage information for individual nodes. The `kubectl describe node` command displays detailed information about a node, including information about the pods running on the node. For each pod, it shows CPU and memory requests, as well as limits for each. If a request or limit is not specified, then the pod shows a 0 for that column. A summary of all resource requests and limits is also displayed.

```
[user@host ~]$ kubectl describe node node1.us-west-1.compute.internal
Name:           node1.us-west-1.compute.internal
Roles:          worker
Labels:         beta.kubernetes.io/arch=amd64
                beta.kubernetes.io/instance-type=m4.xlarge
                beta.kubernetes.io/os=linux
...
Non-terminated Pods:            (20 in total)
...  Name                 CPU Requests  ...  Memory Requests  Memory Limits  AGE
...  ----
...  tuned-vdwt4        10m (0%)    ...  50Mi (0%)      0 (0%)       8d
...  dns-default-2rpwf  110m (3%)   ...  70Mi (0%)      512Mi (3%)   8d
...  node-ca-6xwmn     10m (0%)    ...  10Mi (0%)      0 (0%)       8d
...
Resource          Requests     Limits
-----          -----
cpu              600m (17%)  0 (0%)
memory          1506Mi (9%) 512Mi (3%)
...

```



### Note

The summary columns for **Requests** and **Limits** display the sum totals of defined requests and limits. In the preceding output, only one of the 20 pods running on the node defined a memory limit, and that limit was 512Mi.

The `kubectl describe node` command displays requests and limits, and the `kubectl top` command shows actual usage. For example, if a pod requests 10m of CPU, then the scheduler will ensure that it places the pod on a node with available capacity. Although the pod requested 10m of CPU, it might use more or less than this value, unless it is also constrained by a CPU limit. The `kubectl top nodes` command shows actual usage for one or more nodes in the cluster, and the `kubectl top pods` command shows actual usage for each pod in a namespace.

```
[user@host ~]$ kubectl top nodes -l node-role.kubernetes.io/worker
NAME                      CPU(cores)   CPU%   MEMORY(bytes)   MEMORY%
node1.us-west-1.compute.internal  519m        14%    3126Mi        20%
node2.us-west-1.compute.internal  167m        4%     1178Mi        7%
```

## Applying Quotas

Kubernetes can enforce quotas that track and limit the use of two kinds of resources:

### Object counts

The number of Kubernetes resources, such as pods, services, and routes.

### Compute resources

The number of physical or virtual hardware resources, such as CPU, memory, and storage capacity.

Imposing a quota on the number of Kubernetes resources avoids exhausting other limited software resources, such as IP addresses for services.

Similarly, imposing a quota on the amount of compute resources avoids exhausting the capacity of a single node in a Kubernetes cluster. It also prevents one application from starving other applications of resources.

Kubernetes manages resource quotas by using a `ResourceQuota` or `quota` resource. A quota specifies hard resource usage limits for a namespace. All attributes of a quota are optional, meaning that any resource that is not restricted by a quota can be consumed without bounds.



### Note

Although a single quota resource can define all of the quotas for a namespace, a namespace can also contain multiple quotas. For example, one quota resource might limit compute resources, such as total CPU allowed or total memory allowed. Another quota resource might limit object counts, such as the number of pods allowed or the number of services allowed. The effect of multiple quotas is cumulative, but it is expected that two different `ResourceQuota` resources for the same namespace do not limit the use of the same type of Kubernetes or compute resource. For example, two different quotas in a namespace should not both attempt to limit the maximum number of pods allowed.

The following table describes some resources that a quota can restrict by their count or number:

Resource Name	Quota Description
pods	Total number of pods
replicationcontrollers	Total number of replication controllers
services	Total number of services
secrets	Total number of secrets
persistentvolumeclaims	Total number of persistent volume claims

The following table describes some compute resources that can be restricted by a quota:

Compute Resource Name	Quota Description
cpu (requests.cpu)	Total CPU use across all containers
memory (requests.memory)	Total memory use across all containers
storage (requests.storage)	Total storage requests by containers across all persistent volume claims

Quota attributes can track either resource requests or resource limits for all pods in the namespace. By default, quota attributes track resource requests. Instead, to track resource limits, prefix the compute resource name with `limits`, for example, `limits.cpu`.

The following listing shows a `ResourceQuota` resource defined using YAML syntax. This example specifies quotas for both the number of resources and the use of compute resources:

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: dev-quota
spec:
  hard:
    services: "10"
    cpu: "1300m"
    memory: "1.5Gi"
```

Resource units are the same for pod resource requests and resource limits. For example, `Gi` means `GiB`, and `m` means millicores. One millicore is the equivalent to 1/1000 of a single CPU core.

Resource quotas can be created in the same way as any other Kubernetes resource; that is, by passing a YAML or JSON resource definition file to the `kubectl create` command:

```
[user@host ~]$ kubectl create --save-config -f dev-quota.yml
```

Another way to create a resource quota is by using the `kubectl create quota` command, for example:

```
[user@host ~]$ kubectl create quota dev-quota --hard
services=10,cpu=1300,memory=1.5Gi
```

Use the `kubectl get resourcequota` command to list available quotas, and use the `kubectl describe resourcequota` command to view usage statistics related to any hard limits defined in the quota, for example:

```
[user@host ~]$ kubectl get resourcequota
NAME      AGE   REQUEST
compute-quota  51s   cpu: 500m/10, memory: 300Mi/1Gi ...
count-quota   28s   pods: 1/3, replicationcontrollers: 1/5, services: 1/2 ...
```

Without arguments, the `kubectl describe quota` command displays the cumulative limits set for all `ResourceQuota` resources in the namespace:

```
[user@host ~]$ kubectl describe quota
Name:          compute-quota
Namespace:    schedule-demo
Resource      Used     Hard
-----
cpu           500m    10
memory        300Mi   1Gi

Name:          count-quota
Namespace:    schedule-demo
Resource      Used     Hard
-----
pods          1       3
replicationcontrollers 1       5
services       1       2
```

An active quota can be deleted by name using the `kubectl delete` command:

```
[user@host ~]$ kubectl delete resourcequota QUOTA
```

When a quota is first created in a namespace, the namespace restricts the ability to create any new resources that might violate a quota constraint until it has calculated updated usage statistics. After a quota is created and usage statistics are up-to-date, the namespace accepts the creation of new resources. When creating a new resource, the quota usage immediately increments. When deleting a resource, the quota use decrements during the next full recalculation of quota statistics for the namespace.

Quotas are applied to new resources, but they do not affect existing resources. For example, if you create a quota to limit a namespace to 15 pods, but 20 pods are already running, then the quota will not remove the additional 5 pods that exceed the quota.



### Important

`ResourceQuota` constraints are applied for the namespace as a whole, but many Kubernetes processes, such as builds and deployments, create pods inside the namespace and might fail because starting them would exceed the namespace quota.

If a modification to a namespace exceeds the quota for a resource count, then Kubernetes denies the action and returns an appropriate error message to the user. However, if the modification exceeds the quota for a compute resource, then the operation does not fail immediately; Kubernetes retries the operation several times, giving the administrator an opportunity to increase the quota or to perform another corrective action, such as bringing a new node online.



### Important

If a quota that restricts usage of compute resources for a namespace is set, then Kubernetes refuses to create pods that do not specify resource requests or resource limits for that compute resource. To use most templates and builders with a namespace restricted by quotas, the namespace must also contain a limit range resource that specifies default values for container resource requests.

## Applying Limit Ranges

A `LimitRange` resource, also called a `limit`, defines the default, minimum, and maximum values for compute resource requests, and the limits for a single pod or container defined inside the namespace. A resource request or limit for a pod is the sum of its containers.

To understand the difference between a limit range and a resource quota, consider that a limit range defines valid ranges and default values for a single pod, and a resource quota defines only top values for the sum of all pods in a namespace. A cluster administrator concerned about resource usage in a Kubernetes cluster usually defines both limits and quotas for a namespace.

A limit range resource can also define default, minimum, and maximum values for the storage capacity requested by an image, image stream, or persistent volume claim. If a resource that is added to a namespace does not provide a compute resource request, then it takes the default value provided by the limit ranges for the namespace. If a new resource provides compute resource requests or limits that are smaller than the minimum specified by the namespace limit ranges, then the resource is not created. Similarly, if a new resource provides compute resource requests or limits that are higher than the maximum specified by the namespace limit ranges, then the resource is not created.

The following listing shows a limit range defined using YAML syntax:

```

apiVersion: "v1"
kind: "LimitRange"
metadata:
  name: "dev-limits"
spec:
  limits:
    - type: "Pod"
      max: ①
        cpu: "500m"
        memory: "750Mi"
      min: ②
        cpu: "10m"
        memory: "5Mi"
    - type: "Container"
      max: ③
        cpu: "500m"
        memory: "750Mi"
      min: ④
        cpu: "10m"
        memory: "5Mi"
      default: ⑤
        cpu: "100m"
        memory: "100Mi"
      defaultRequest: ⑥
        cpu: "20m"
        memory: "20Mi"
    - type: "PersistentVolumeClaim" ⑦
      min:
        storage: "1Gi"
      max:
        storage: "50Gi"

```

- ➊ The maximum amount of CPU and memory that all containers within a pod can consume. A new pod that exceeds the maximum limits is not created. An existing pod that exceeds the maximum limits is restarted.
- ➋ The minimum amount of CPU and memory consumed across all containers within a pod. A pod that does not satisfy the minimum requirements is not created. Because many pods only have one container, you might set the minimum pod values to the same values as the minimum container values.
- ➌ The maximum amount of CPU and memory that an individual container within a pod can consume. A new container that exceeds the maximum limits does not create the associated pod. An existing container that exceeds the maximum limits restarts the entire pod.
- ➍ The minimum amount of CPU and memory that an individual container within a pod can consume. A container that does not satisfy the minimum requirements prevents the associated pod from being created.
- ➎ The default maximum amount of CPU and memory that an individual container can consume. This is used when a CPU resource limit or a memory limit is not defined for the container.
- ➏ The default CPU and memory an individual container requests. This default is used when a CPU resource request or a memory request is not defined for the container. If CPU and memory quotas are enabled for a namespace, then configuring the `defaultRequest` section allows pods to start, even if the containers do not specify resource requests.
- ➐ The minimum and maximum sizes allowed for a persistent volume claim.

Users can create a limit range resource in the same way as any other Kubernetes resource; that is, by passing a YAML or JSON resource definition file to the `kubectl create` command:

```
[user@host ~]$ kubectl create --save-config -f dev-limits.yml
```

Use the `kubectl describe limitrange` command to view the limit constraints enforced in a namespace:

```
[user@host ~]$ kubectl describe limitrange dev-limits
Name:          dev-limits
Namespace:    schedule-demo
Type          Resource        Min   Max   Default Request ...
----          -----        ---   ---   -----      ...
Pod           cpu            10m   500m  -          ...
Pod           memory         5Mi   750Mi -          ...
Container     memory         5Mi   750Mi 20Mi     ...
Container     cpu            10m   500m  20m     ...
PersistentVolumeClaim storage       1Gi   50Gi  -          ...
```

An active limit range can be deleted by name with the `kubectl delete` command:

```
[user@host ~]$ kubectl delete limitrange dev-limits
```

After a limit range is created in a namespace, all requests to create new resources are evaluated against each limit range resource in the namespace. If the new resource violates the minimum or maximum constraint enumerated by any limit range, then the resource is rejected. If the new

resource does not set an explicit value, and the constraint supports a default value, then the default value is applied to the new resource as its usage value.

All resource update requests are also evaluated against each limit range resource in the namespace. If the updated resource violates any constraint, then the update is rejected.



### Important

Avoid setting `LimitRange` constraints that are too high, or `ResourceQuota` constraints that are too low. A violation of `LimitRange` constraints prevents pod creation, resulting in error messages. A violation of `ResourceQuota` constraints prevents a pod from being scheduled to any node. The pod might be created but remain in the pending state.



### References

#### Resource Quotas

<https://kubernetes.io/docs/concepts/policy/resource-quotas/>

#### Limit Ranges

<https://kubernetes.io/docs/concepts/policy/limit-range/>

## ► Guided Exercise

# Limiting Resource Usage

In this exercise, you will configure an application so that it does not consume all computing resources from the cluster and its compute nodes.

## Outcomes

You should be able to use the Kubernetes command-line interface to:

- Configure an application to specify resource requests for CPU and memory usage.
- Modify an application to work within existing cluster restrictions.

## Before You Begin

You need a working Kubernetes cluster, and your `kubectl` command must be configured to communicate with the cluster.

Make sure your `kubectl` context refers to a namespace where you have enough permissions, usually `username-dev` or `username-stage`. Use the `kubectl config set-context --current --namespace=namespace` command to switch to the appropriate namespace.

## Instructions

- 1. Deploy a test application for this exercise that explicitly requests container resources for CPU and memory.
- 1.1. Create a deployment resource file and save it to a file named `hello-limit.yaml`. Name the application `hello-limit` and use the container image located at `quay.io/redhattraining/hello-world-nginx:v1.0`.



### Note

This course uses the backslash character (\) to break long commands. On Linux and macOS, you can use the line breaks.

On Windows, use the backtick character (`) to break long commands. Alternatively, do not break long commands.

Refer to *Orientation to the Classroom Environment* for more information about long commands.

```
[user@host ~]$ kubectl create deployment hello-limit \
--image quay.io/redhattraining/hello-world-nginx:v1.0 \
--dry-run=client -o yaml > hello-limit.yaml
```

- 1.2. Edit the file `hello-limit.yaml` to replace the resources: `{}` line with the highlighted lines below. Ensure that you have proper indentation before saving the file.

```
...output omitted...
spec:
  containers:
    - image: quay.io/redhattraining/hello-world-nginx:v1.0
      name: hello-world-nginx
      resources:
        requests:
          cpu: "8"
          memory: 20Mi
status: {}
```

- 1.3. Create the new application using your resource file.

```
[user@host ~]$ kubectl create --save-config -f hello-limit.yaml
deployment.apps/hello-limit created
```

- 1.4. Although a new deployment was created for the application, the application pod should have a status of Pending.

```
[user@host ~]$ kubectl get pods
NAME                      READY   STATUS    RESTARTS   AGE
hello-limit-d86874d86b-fpmrt   0/1     Pending   0          10s
```

- 1.5. The pod cannot be customized because none of the compute nodes have sufficient CPU resources. This can be verified by viewing warning events.

```
[user@host ~]$ kubectl get events --field-selector type=Warning
LAST SEEN    TYPE      REASON           OBJECT           MESSAGE
88s         Warning   FailedScheduling   pod/hello-limit-d86874d86b-fpmrt  0/3
nodes are available: 8 Insufficient cpu.
```

► 2. Redeploy your application so that it requests fewer CPU resources.

- 2.1. Edit the `hello-limit.yaml` file to request 1.2 CPUs for the container. Change the `cpu: "8"` line to match the highlighted line below.

```
...output omitted...
resources:
  requests:
    cpu: "1200m"
    memory: 20Mi
```

- 2.2. Apply the changes to your application.

```
[user@host ~]$ kubectl apply -f hello-limit.yaml
deployment.apps/hello-limit configured
```

- 2.3. Verify that your application deploys successfully. You might need to run `kubectl get pods` multiple times until you see a running pod. The previous pod with a pending status will terminate and eventually disappear.

```
[user@host ~]$ kubectl get pods
NAME                  READY   STATUS        RESTARTS   AGE
hello-limit-d86874d86b-fpmrt   0/1     Terminating   0          2m19s
hello-limit-7c7998ff6b-ctsjsp   1/1     Running      0          6s
```



### Note

If your application pod does not get customized, modify the `hello-limit.yaml` file to reduce the CPU request to `1000m`. Apply the changes again and verify the pod status is `Running`.

## Finish

Delete the created resources to clean your cluster.

```
[user@host ~] kubectl delete -f hello-limit.yaml
deployment.apps "hello-limit" deleted

[user@host ~] rm hello-limit.yaml
```

This concludes the guided exercise.

## ► Quiz

# Limiting Resource Usage

Choose the correct answers to the following questions:

- 1. **The CPU quota of your Kubernetes cluster namespace allows pods to use up to five CPU cores. The following warning shows the reason why Kubernetes cannot run a pod. Based on the warning, which of the following solutions would solve this situation? (Choose one.)**

```
[user@host ~]$ kubectl get events --field-selector type=Warning
Warning FailedScheduling pod/webapp-d8486446b-fpmrt 0/4 nodes are available:
10 Insufficient cpu.
```

- a. Add Kubernetes nodes with more than five CPU cores to the cluster.
  - b. Decrease the `resources.limits.cpu` property of the container to be less than five.
  - c. Decrease the `resources.requests.cpu` property of the container to be less than five.
  - d. Decrease the `resources.limits.memory` property of the container to be less than five.
- 2. **Which three of the following resources can you restrict with quotas? (Choose three.)**
- a. `cpu`
  - b. `memory`
  - c. `network.requests`
  - d. `ingress.speed`
  - e. `pods`

► **3. The following is a description of the resource requirements for an application.**

**Which two of the following resource limits would you recommend for the described application? (Choose two.)**

- \* The application does not have minimum CPU requirements to start.
- \* The application occasionally performs very CPU-intensive tasks.
- \* The application requires 10MiB of memory to run.
- \* The application memory consumption is constant.
- \* The application deployment must have at least 2 replicas.

- a. `requests.cpu`, because the application will require many CPU cores occasionally.
- b. `requests.memory`, because the application needs a minimum amount of memory to run.
- c. `requests.pods`, because there must be two pods running.
- d. `limits.cpu`, because the application might overuse the cluster CPUs occasionally.
- e. `limits.memory`, because memory consumption might increase.
- f. `limits.pods`, because there must be no more than two pods running.

► **4. Which two of the following statements about limit ranges are true? (Choose two.)**

- a. Limit range presents no difference when compared with resource quotas.
- b. Limit ranges define default, minimum, and maximum values for resource requests.
- c. A resource request or limit for a pod is the sum of the pod containers.
- d. You can define `LimitRange` resources at the cluster level. Defining `LimitRange` resources for a single namespace is not supported.

## ► Solution

# Limiting Resource Usage

Choose the correct answers to the following questions:

- 1. **The CPU quota of your Kubernetes cluster namespace allows pods to use up to five CPU cores. The following warning shows the reason why Kubernetes cannot run a pod. Based on the warning, which of the following solutions would solve this situation? (Choose one.)**

```
[user@host ~]$ kubectl get events --field-selector type=Warning
Warning FailedScheduling pod/webapp-d8486446b-fpmrt 0/4 nodes are available:
10 Insufficient cpu.
```

- a. Add Kubernetes nodes with more than five CPU cores to the cluster.
  - b. Decrease the resources.limits.cpu property of the container to be less than five.
  - c. Decrease the resources.requests.cpu property of the container to be less than five.
  - d. Decrease the resources.limits.memory property of the container to be less than five.
- 2. **Which three of the following resources can you restrict with quotas? (Choose three.)**
- a. cpu
  - b. memory
  - c. network.requests
  - d. ingress.speed
  - e. pods

► **3. The following is a description of the resource requirements for an application.**

**Which two of the following resource limits would you recommend for the described application? (Choose two.)**

- \* The application does not have minimum CPU requirements to start.
- \* The application occasionally performs very CPU-intensive tasks.
- \* The application requires 10MiB of memory to run.
- \* The application memory consumption is constant.
- \* The application deployment must have at least 2 replicas.

- a. requests.cpu, because the application will require many CPU cores occasionally.
- b. requests.memory, because the application needs a minimum amount of memory to run.
- c. requests.pods, because there must be two pods running.
- d. limits.cpu, because the application might overuse the cluster CPUs occasionally.
- e. limits.memory, because memory consumption might increase.
- f. limits.pods, because there must be no more than two pods running.

► **4. Which two of the following statements about limit ranges are true? (Choose two.)**

- a. Limit range presents no difference when compared with resource quotas.
- b. Limit ranges define default, minimum, and maximum values for resource requests.
- c. A resource request or limit for a pod is the sum of the pod containers.
- d. You can define LimitRange resources at the cluster level. Defining LimitRange resources for a single namespace is not supported.

# Proving Liveness, Readiness and Startup

---

## Objectives

After completing this section, you should be able to review how Kubernetes evaluates application health status via probes and automatic application restart.

## Kubernetes Readiness and Liveness Probes

Applications can become unreliable for a variety of reasons, for example:

- Temporary connection loss
- Configuration errors
- Application errors

Developers can use *probes* to monitor their applications. Probes make developers aware of events such as application status, resource usage, and errors.

Monitoring of such events is useful for fixing problems, but can also help with resource planning and managing.

A probe is a periodic check that monitors the health of an application. Developers can configure probes by using either the `kubectl` command-line client or a YAML deployment template.

There are currently three types of probes in Kubernetes:

### Startup Probe

A startup probe verifies whether the application within a container is started. Startup probes run before any other probe, and, unless it finishes successfully, disables other probes. If a container fails its startup probe, then the container is killed and follows the pod's `restartPolicy`.

This type of probe is only executed at startup, unlike readiness probes, which are run periodically.

The startup probe is configured in the `spec.containers.startupprobe` attribute of the pod configuration.

### Readiness Probe

Readiness probes determine whether or not a container is ready to serve requests. If the readiness probe returns a failed state, then Kubernetes removes the IP address for the container from the endpoints of all Services.

Developers use readiness probes to instruct Kubernetes that a running container should not receive any traffic. This is useful when waiting for an application to perform time-consuming initial tasks, such as establishing network connections, loading files, and warming caches.

The readiness probe is configured in the `spec.containers.readinessprobe` attribute of the pod configuration.

### Liveness Probe

Liveness probes determine whether or not an application running in a container is in a healthy state. If the liveness probe detects an unhealthy state, then Kubernetes kills the container and tries to redeploy it.

The liveness probe is configured in the `spec.containers.livenessprobe` attribute of the pod configuration.

Kubernetes provides five options that control these probes:

Name	Mandatory	Description	Default Value
<code>initialDelaySeconds</code>	Yes	Determines how long to wait after the container starts before beginning the probe.	0
<code>timeoutSeconds</code>	Yes	Determines how long to wait for the probe to finish. If this time is exceeded, then Kubernetes assumes that the probe failed.	1
<code>periodSeconds</code>	No	Specifies the frequency of the checks.	10
<code>successThreshold</code>	No	Specifies the minimum consecutive successes for the probe to be considered successful after it has failed.	1
<code>failureThreshold</code>	No	Specifies the minimum consecutive failures for the probe to be considered failed after it has succeeded.	3

## Methods of Checking Application Health

Startup, readiness, and liveness probes can check the health of applications in three ways: HTTP checks, container execution checks, and TCP socket checks.

### HTTP Checks

An HTTP check is ideal for applications that return HTTP status codes, such as REST APIs.

HTTP probe uses GET requests to check the health of an application. The check is successful if the HTTP response code is in the range 200-399.

The following example demonstrates how to implement a readiness probe with the HTTP check method:

```
...contents omitted...
readinessProbe:
  httpGet:
    path: /health①
    port: 8080
  initialDelaySeconds: 15②
  timeoutSeconds: 1③
...contents omitted...
```

- ① The readiness probe endpoint.
- ② How long to wait after the container starts before checking its health.
- ③ How long to wait for the probe to finish.

## Container Execution Checks

Container execution checks are ideal in scenarios where you must determine the status of the container based on the exit code of a process or shell script running in the container.

When using container execution checks Kubernetes executes a command inside the container. Exiting the check with a status of `0` is considered a success. All other status codes are considered a failure.

The following example demonstrates how to implement a container execution check:

```
...contents omitted...
livenessProbe:
  exec:
    command: ①
      - cat
      - /tmp/health
  initialDelaySeconds: 15
  timeoutSeconds: 1
...contents omitted...
```

- ① The command to run and its arguments, as a YAML array.

## TCP Socket Checks

A TCP socket check is ideal for applications that run as daemons, and open TCP ports, such as database servers, file servers, web servers, and application servers.

When using TCP socket checks Kubernetes attempts to open a socket to the container. The container is considered healthy if the check can establish a successful connection.

The following example demonstrates how to implement a liveness probe by using the TCP socket check method:

```
...contents omitted...
livenessProbe:
  tcpSocket:
    port: 8080①
  initialDelaySeconds: 15
  timeoutSeconds: 1
...contents omitted...
```

- ① The TCP port to check.

## Creating Probes

To configure probes on a deployment, edit the deployment's resource definition. To do this, you can use the `kubectl edit` or `kubectl patch` commands.

The following example includes adding a probe into a deployment resource definition by using the `kubectl edit` command.

```
[user@host ~]$ kubectl edit deployment mydeployment
apiVersion: apps/v1
kind: Deployment
  ...contents omitted...
spec:
  ...contents omitted...
  template:
    ...contents omitted...
    spec:
      containers:
        - image: quay.io/redhattraining/do100-versioned-hello:v1.0
          name: do100-versioned-hello
          readinessProbe:
            failureThreshold: 3
            httpGet:
              path: /healthz
              port: 8080
              scheme: HTTP
            periodSeconds: 10
            successThreshold: 1
            timeoutSeconds: 1
      ...contents omitted...
```



### References

#### **The official Kubernetes documentation on probes**

<https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-startup-probes/>

## ► Guided Exercise

# Proving Liveness, Readiness and Startup

In this exercise, you will configure liveness and readiness probes to monitor the health of an application deployed to your Kubernetes cluster.

The application you deploy in this exercise exposes two HTTP GET endpoints:

- The `/healthz` endpoint responds with a 200 HTTP status code when the application pod can receive requests.

The endpoint indicates that the application pod is healthy and reachable. It does not indicate that the application is ready to serve requests.

- The `/ready` endpoint responds with a 200 HTTP status code if the overall application works.

The endpoint indicates that the application is ready to serve requests.

In this exercise, the `/ready` endpoint responds with the 200 HTTP status code when the application pod starts. The `/ready` endpoint responds with the 503 HTTP status code for the first 30 seconds after deployment to simulate slow application startup.

You will configure the `/healthz` endpoint for the liveness probe, and the `/ready` endpoint for the readiness probe.

You will simulate network failures in your Kubernetes cluster and observe behavior in the following scenarios:

- The application is not available.
- The application is available but cannot reach the database. Consequently, it cannot serve requests.

## Outcomes

You should be able to:

- Configure readiness and liveness probes for an application from the command line.
- Locate probe failure messages in the event log.

## Before You Begin

You need a working Kubernetes cluster, and your `kubectl` command must be configured to communicate with the cluster.

Make sure your `kubectl` context refers to a namespace where you have enough permissions, usually `username-dev` or `username-stage`. Use the `kubectl config set-context --current --namespace=namespace` command to switch to the appropriate namespace.

## Instructions

- 1. Deploy the do100-probes sample application to the Kubernetes cluster and expose the application.

- 1.1. Create a new deployment by using kubectl.



### Note

This course uses the backslash character (\) to break long commands. On Linux and macOS, you can use the line breaks.

On Windows, use the backtick character (`) to break long commands. Alternatively, do not break long commands.

Refer to *Orientation to the Classroom Environment* for more information about long commands.

```
[user@host ~]$ kubectl create deployment do100-probes \
--image quay.io/redhattraining/do100-probes:external
deployment.apps/do100-probes created
```

- 1.2. Expose the deployment on port 8080.

```
[user@host ~]$ kubectl expose deployment/do100-probes --port 8080
service/do100-probes exposed
```

- 1.3. Use a text editor to create a file in your current directory called probes-ingress.yml.

Create the probes-ingress.yml file with the following content. Ensure correct indentation (using spaces rather than tabs) and then save the file.

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: do100-probes
  labels:
    app: do100-probes
spec:
  rules:
    - host: INGRESS-HOST
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: do100-probes
                port:
                  number: 8080
```

Replace *INGRESS-HOST* with the hostname associated with your Kubernetes cluster, such as `hello.example.com` or `do100-probes-USER-dev.apps.sandbox.x8i5.p1.openshiftapps.com`.

The file at <https://github.com/RedHatTraining/DO100x-apps/blob/main/probes/probes-ingress.yml> contains the correct content for the `probes-ingress.yml` file. You can download the file and use it for comparison.

- 1.4. Use the `kubectl create` command to create the ingress resource.

```
[user@host ~]$ kubectl create -f probes-ingress.yml
ingress.networking.k8s.io/do100-probes created
```

- 2. Manually test the application's `/ready` and `/healthz` endpoints.

- 2.1. Display information about the `do100-probes` ingress. If the command does not display an IP address, then wait up to a minute and try running the command again.

```
[user@host ~]$ kubectl get ingress/do100-probes
NAME      CLASS      HOSTS          ADDRESS      PORTS      ...
do100-probes  nginx    hello.example.com  192.168.49.2  80        ...
```

The value in the `HOST` column matches the `host` line specified in your `probes-ingress.yml` file. Your IP address is likely different from the one displayed here.

- 2.2. Test the `/ready` endpoint:

```
[user@host ~]$ curl -i hello.example.com/ready
```

On Windows, remove `-i` flag:

```
[user@host ~]$ curl hello.example.com/ready
```

The `/ready` endpoint simulates a slow startup of the application, and so for the first 30 seconds after the application starts, it returns an HTTP status code of 503, and the following response:

```
HTTP/1.1 503 Service Unavailable
...output omitted...
Error! Service not ready for requests...
```

After the application has been running for 30 seconds, it returns:

```
HTTP/1.1 200 OK
...output omitted...
Ready for service requests...
```

- 2.3. Test the `/healthz` endpoint of the application:

```
[user@host ~]$ curl -i hello.example.com/healthz
HTTP/1.1 200 OK
...output omitted...
OK
```

On Windows, remove the `-i` flag:

```
[user@host ~]$ curl hello.example.com/healthz
```

2.4. Test the application response:

```
[user@host ~]$ curl hello.example.com
Hello! This is the index page for the app.
```

► 3. Activate readiness and liveness probes for the application.

3.1. Use the `kubectl edit` command to edit the deployment definition and add readiness and liveness probes.

- For the liveness probe, use the `/healthz` endpoint on the port 8080.
- For the readiness probe, use the `/ready` endpoint on the port 8080.
- For both probes:
  - Configure an initial delay of 15 seconds.
  - Configure the timeout as 2 seconds.

```
[user@host ~] kubectl edit deployment/do100-probes
...output omitted...
```

This command opens your default system editor. Make changes to the definition so that it displays as follows.

```
...output omitted...
spec:
  ...output omitted...
  template:
    spec:
      containers:
        - image: quay.io/redhattraining/do100-probes:external
          ...output omitted...
        readinessProbe:
          httpGet:
            path: /ready
            port: 8080
            initialDelaySeconds: 15
            timeoutSeconds: 2
        livenessProbe:
          httpGet:
            path: /healthz
            port: 8080
            initialDelaySeconds: 15
            timeoutSeconds: 2
```

**Warning**

The YAML resource is space sensitive. Use spaces to preserve the spacing.

Do not use the tab character to edit the preceding deployment.

Save and exit the editor to apply your changes.

3.2. Verify the value in the `livenessProbe` and `readinessProbe` entries:

```
[user@host ~]$ kubectl describe deployment do100-probes
...output omitted...
  Liveness:      http-get http://:8080/healthz delay=15s timeout=2s period=10s
#success=1 #failure=3
  Readiness:     http-get http://:8080/ready   delay=15s timeout=2s period=10s
#success=1 #failure=3
...output omitted...
```

3.3. Wait for the application pod to redeploy and change into the READY state:

NAME	READY	STATUS	RESTARTS	AGE
do100-probes-7794c5cb4f-vwl4x	<b>0/1</b>	Running	0	6s

The READY status shows `0/1` if the AGE value is less than approximately 30 seconds. After that, the READY status is `1/1`. Note the pod name for the following steps.

NAME	READY	STATUS	RESTARTS	AGE
do100-probes-7794c5cb4f-vwl4x	<b>1/1</b>	Running	0	62s

3.4. Use the `kubectl logs` command to see the results of the liveness and readiness probes. Use the pod name from the previous step.

```
[user@host ~]$ kubectl logs -f do100-probes-7794c5cb4f-vwl4x
...output omitted...
nodejs server running on  http://0.0.0.0:8080
ping /healthz => pong [healthy]
ping /ready => pong [notready]
ping /healthz => pong [healthy]
ping /ready => pong [notready]
ping /healthz => pong [healthy]
ping /ready => pong [ready]
...output omitted...
```

Observe that the readiness probe fails for about 30 seconds after redeployment, and then succeeds. Recall that the application simulates a slow initialization of the application by forcibly setting a 30-second delay before it responds with a status of ready.

Do not terminate this command. You will continue to monitor the output of this command in the next step.

► **4.** Simulate a network failure.

In case of a network failure, a service becomes unresponsive. This means both the liveness and readiness probes fail.

Kubernetes can resolve the issue by recreating the container on a different node.

- 4.1. In a different terminal window or tab, run the following commands to simulate a liveness probe failure:

```
[user@host ~]$ curl http://hello.example.com/flip?op=kill-health  
Switched app state to unhealthy...  
[user@host ~]$ curl http://hello.example.com/flip?op=kill-ready  
Switched app state to not ready...
```

- 4.2. Return to the terminal where you are monitoring the application deployment:

```
[user@host ~]$ kubectl logs -f do100-probes-7794c5cb4f-vwl4x  
...output omitted...  
Received kill request for health probe.  
Received kill request for readiness probe.  
...output omitted...  
ping /ready => pong [notready]  
ping /healthz => pong [unhealthy]  
...output omitted...  
Received kill request for health probe.  
...output omitted...  
Received kill request for readiness probe.  
...output omitted...
```

Kubernetes restarts the pod when the liveness probe fails repeatedly (three consecutive failures by default). This means Kubernetes restarts the application on an available node not affected by the network failure.

You see this log output only when you immediately check the application logs after you issue the kill request. If you check the logs after Kubernetes restarts the pod, then the logs are cleared and you only see the output shown in the next step.

- 4.3. Verify that Kubernetes restarts the unhealthy pod. Keep checking the output of the `kubectl get pods` command. Observe the RESTARTS column and verify that the count is greater than zero. Note the name of the new pod.

```
[user@host ~]$ kubectl get pods  
NAME                  READY   STATUS    RESTARTS      AGE  
do100-probes-95758759b-4cm2j   1/1     Running   1 (11s ago)   62s
```

- 4.4. Review the application logs. The liveness probe succeeds and the application reports a healthy state.

```
[user@host ~]$ kubectl logs -f do100-probes-95758759b-4cm2j
...output omitted...
ping /ready => pong [ready]
ping /healthz => pong [healthy]
...output omitted...
```

## Finish

Delete the deployment, ingress, and service resources to clean your cluster. Kubernetes automatically deletes the associated pods.

```
[user@host ~]$ kubectl delete deployment do100-probes
deployment.apps "do100-versioned-hello" deleted
[user@host ~]$ kubectl delete service do100-probes
service "do100-probes" deleted
[user@host ~]$ kubectl delete ingress do100-probes
ingress.networking.k8s.io "do100-probes" deleted
```

This concludes the guided exercise.

## ► Quiz

# Proving Liveness, Readiness and Startup

Choose the correct answers to the following questions:

► 1. **Which three of the following methods are valid Kubernetes probes? (Choose three.)**

- a. TCP socket check
- b. Startup check
- c. Container Execution check
- d. End-to-end check
- e. HTTP check

► 2. **Given the following startup probe configuration, which of the following statements is correct? (Choose one.)**

```
startupProbe:  
  httpGet:  
    path: /health  
    port: 8080  
  failureThreshold: 10  
  periodSeconds: 5
```

- a. The startup probe retries the HTTP check at most five times.
- b. After a failure, the probe waits 10 seconds until the next verification.
- c. The startup probe disables the liveness and readiness probes until it finishes successfully.
- d. The probe check must succeed 10 times for Kubernetes to consider the probe successful.

► 3. Given the following liveness probe configuration, which of the following statements is correct? (Choose one.)

```
livenessProbe:  
  exec:  
    command:  
      - check-status  
  failureThreshold: 1  
  initialDelaySeconds: 5  
  periodSeconds: 10  
  successThreshold: 1  
  timeoutSeconds: 1
```

- a. The liveness probe waits five seconds after the container has started, then runs the `check-status` command inside the container.
- b. The liveness probe runs the `check-status` command immediately after the container starts.
- c. The liveness probe runs the `check-status` command every five seconds.
- d. The liveness probe fails if the `check-status` command exists with a non-zero code five consecutive times.

► 4. Given the following readiness probe configuration, which of the following statements is correct?

```
readinessProbe:  
  httpGet:  
    path: /ready  
    port: 8080  
  failureThreshold: 5  
  periodSeconds: 10  
  successThreshold: 1  
  timeoutSeconds: 1
```

- a. If the readiness probe fails once, then Kubernetes terminates the container.
- b. If the readiness probe fails once, then Kubernetes stops sending traffic to the container.
- c. If the readiness probe fails five consecutive times, then Kubernetes terminates the container.
- d. If the readiness probe fails five consecutive times, then Kubernetes stops sending traffic to the container.

## ► Solution

# Proving Liveness, Readiness and Startup

Choose the correct answers to the following questions:

► 1. **Which three of the following methods are valid Kubernetes probes? (Choose three.)**

- a. TCP socket check
- b. Startup check
- c. Container Execution check
- d. End-to-end check
- e. HTTP check

► 2. **Given the following startup probe configuration, which of the following statements is correct? (Choose one.)**

```
startupProbe:  
  httpGet:  
    path: /health  
    port: 8080  
  failureThreshold: 10  
  periodSeconds: 5
```

- a. The startup probe retries the HTTP check at most five times.
- b. After a failure, the probe waits 10 seconds until the next verification.
- c. The startup probe disables the liveness and readiness probes until it finishes successfully.
- d. The probe check must succeed 10 times for Kubernetes to consider the probe successful.

► 3. Given the following liveness probe configuration, which of the following statements is correct? (Choose one.)

```
livenessProbe:  
  exec:  
    command:  
      - check-status  
  failureThreshold: 1  
  initialDelaySeconds: 5  
  periodSeconds: 10  
  successThreshold: 1  
  timeoutSeconds: 1
```

- a. The liveness probe waits five seconds after the container has started, then runs the `check-status` command inside the container.
- b. The liveness probe runs the `check-status` command immediately after the container starts.
- c. The liveness probe runs the `check-status` command every five seconds.
- d. The liveness probe fails if the `check-status` command exists with a non-zero code five consecutive times.

► 4. Given the following readiness probe configuration, which of the following statements is correct?

```
readinessProbe:  
  httpGet:  
    path: /ready  
    port: 8080  
  failureThreshold: 5  
  periodSeconds: 10  
  successThreshold: 1  
  timeoutSeconds: 1
```

- a. If the readiness probe fails once, then Kubernetes terminates the container.
- b. If the readiness probe fails once, then Kubernetes stops sending traffic to the container.
- c. If the readiness probe fails five consecutive times, then Kubernetes terminates the container.
- d. If the readiness probe fails five consecutive times, then Kubernetes stops sending traffic to the container.

# Configuring Cloud Applications

---

## Objectives

After completing this section, you should be able to create Kubernetes resources holding application configuration and secrets, and how to make that configuration available to running applications.

## Externalizing Application Configuration in Kubernetes

Developers configure their applications through a combination of environment variables, command-line arguments, and configuration files. When deploying applications to Kubernetes, configuration management presents a challenge due to the immutable nature of containers. When running containerized applications, decoupling application and configuration code is of a higher priority than in traditional deployments.

The recommended approach for containerized applications is to decouple the static application binaries from the dynamic configuration data and to externalize the configuration. This separation ensures the portability of applications across many environments.

For example, you want to promote an application that is deployed to a Kubernetes cluster from a development environment to a production environment, with intermediate stages such as testing and user acceptance. You must use the same application container image in all stages and have the configuration details specific to each environment outside the container image.

## Using Secret and Configuration Map Resources

Kubernetes provides the secret and configuration map resource types to externalize and manage configuration for applications.

Secret resources are used to store sensitive information, such as passwords, keys, and tokens. As a developer, it is important to create secrets to avoid compromising credentials and other sensitive information in your application. There are different secret types that enforce usernames and keys in the secret object. Some of them are `service-account-token`, `basic-auth`, `ssh-auth`, `tls`, and `opaque`. The default type is `opaque`, which allows unstructured and non-validated key:value pairs that can contain arbitrary values.

Configuration map resources are similar to secret resources, but they store nonsensitive data. A configuration map resource can be used to store fine-grained information, such as individual properties, or coarse-grained information, such as entire configuration files and JSON data.

You can create configuration map and secret resources using the `kubectl` command. You can then reference them in your pod specification and Kubernetes automatically injects the resource data into the container as environment variables, or as files mounted through volumes inside the application container.

You can also configure the deployment to reference configuration map and secret resources. Kubernetes then automatically redeploys the application and makes the data available to the container.

Data is stored inside a secret resource by using base64 encoding. When data from a secret is injected into a container, the data is decoded and either mounted as a file, or injected as environment variables inside the container.



### Note

Encoding any text in base64 does not add any layer of security except against casual snoop.

## Features of Secrets and Configuration Maps

Notice the following with respect to secrets and configuration maps:

- They can be referenced independently of their definition.
- For security reasons, mounted volumes for these resources are backed by temporary file storage facilities (tmpfs) and never stored on a node.
- They are scoped to a namespace.

## Creating and Managing Secrets and Configuration Maps

Secrets and configuration maps must be created before creating the pods that depend on them. Use the `kubectl create` command to create secrets and configuration map resources.

To create a new configuration map that stores string literals:

```
[user@host ~]$ kubectl create configmap config_map_name \
--from-literal key1=value1 \
--from-literal key2=value2
```

To create a new secret that stores string literals:

```
[user@host ~]$ kubectl create secret generic secret_name \
--from-literal username=user1 \
--from-literal password=mypa55w0rd
```

To create a new configuration map that stores the contents of a file or a directory containing a set of files:

```
[user@host ~]$ kubectl create configmap config_map_name \
--from-file /home/demo/conf.txt
```

When you create a configuration map from a file, the key name will be the name of the file by default and the value will be the contents of the file.

When you create a configuration map resource based on a directory, each file with a valid name key in the directory is stored in the configuration map. Subdirectories, symbolic links, device files, and pipes are ignored.

Run the `kubectl create configmap --help` command for more information.

**Note**

You can also abbreviate the configmap resource type argument as cm in the kubectl command-line interface. For example:

```
[user@host ~]$ kubectl create cm myconf --from-literal key1=value1
[user@host ~]$ kubectl get cm myconf
```

To create a new secret that stores the contents of a file or a directory containing a set of files:

```
[user@host ~]$ kubectl create secret generic secret_name \
--from-file /home/demo/mysecret.txt
```

When you create a secret from either a file or a directory, the key names are set the same way as configuration maps.

For more details, including storing TLS certificates and keys in secrets, run the `kubectl create secret --help` and the `kubectl secret` commands.

## Configuration Map and Secret Resource Definitions

Because configuration maps and secrets are regular Kubernetes resources, you can use either the `kubectl create` command to import these resource definition files in YAML or JSON format.

A sample configuration map resource definition in YAML format is shown below:

```
apiVersion: v1
data:
  key1: value1 ①②
  key2: value2 ③④
kind: ConfigMap ⑤
metadata:
  name: myconf ⑥
```

- ① The name of the first key. By default, an environment variable or a file with the same name as the key is injected into the container depending on whether the configuration map resource is injected as an environment variable or a file.
- ② The value stored for the first key of the configuration map.
- ③ The name of the second key.
- ④ The value stored for the second key of the configuration map.
- ⑤ The Kubernetes resource type; in this case, a configuration map.
- ⑥ A unique name for this configuration map inside a project.

A sample secret resource in YAML format is shown below:

```

apiVersion: v1
data:
  username: cm9vdAo= 1 2
  password: c2VjcmV0Cg== 3 4
kind: Secret 5
metadata:
  name: mysecret 6
  type: Opaque

```

- 1** The name of the first key. This provides the default name for either an environment variable or a file in a pod, just like the key names from a configuration map.
- 2** The value stored for the first key, in base64-encoded format.
- 3** The name of the second key.
- 4** The value stored for the second key, in base64-encoded format.
- 5** The Kubernetes resource type; in this case, a secret.
- 6** A unique name for this secret resource inside a project.

## Commands to Manipulate Configuration Maps

To view the details of a configuration map in JSON format, or to export a configuration map resource definition to a JSON file for offline creation:

```
[user@host ~]$ kubectl get configmap/myconf -o json
```

To delete a configuration map:

```
[user@host ~]$ kubectl delete configmap/myconf
```

To edit a configuration map, use the `kubectl edit` command. This command opens an inline editor, with the configuration map resource definition in YAML format:

```
[user@host ~]$ kubectl edit configmap/myconf
```

Use the `kubectl patch` command to edit a configuration map resource. This approach is non-interactive and is useful when you need to script the changes to a resource:

```
[user@host ~]$ kubectl patch configmap/myconf --patch '{"data": {"key1": "newvalue1"}}'
```

## Commands to Manipulate Secrets

The commands to manipulate secret resources are similar to those used for configuration map resources.

To view or export the details of a secret:

```
[user@host ~]$ kubectl get secret/mysecret -o json
```

To delete a secret:

```
[user@host ~]$ kubectl delete secret/mysecret
```

To edit a secret, first encode your data in base64 format, for example:

```
[user@host ~]$ echo 'newpassword' | base64  
bmV3cGFzc3dvcnQK
```

Use the encoded value to update the secret resource using the `kubectl edit` command:

```
[user@host ~]$ kubectl edit secret/mysecret
```

You can also edit a secret resource using the `kubectl patch` command:

```
[user@host ~]$ kubectl patch secret/mysecret --patch \  
'{"data":{"password":"bmV3cGFzc3dvcnQK"}}'
```

## Injecting Data from Secrets and Configuration Maps into Applications

Configuration maps and secrets can be mounted as data volumes, or exposed as environment variables, inside an application container.

To inject all values stored in a configuration map into environment variables for pods created from a deployment use the `kubectl set env` command:

```
[user@host ~]$ kubectl set env deployment/mydcname \  
--from configmap/myconf
```

To mount all keys from a configuration map as files from a volume inside pods created from a deployment, use the `kubectl set volume` command:

```
[user@host ~]$ kubectl set volume deployment/mydcname --add \  
-t configmap -m /path/to/mount/volume \  
--name myvol --configmap-name myconf
```

To inject data inside a secret into pods created from a deployment, use the `kubectl set env` command:

```
[user@host ~]$ kubectl set env deployment/mydcname \  
--from secret/mysecret
```

To mount data from a secret resource as a volume inside pods created from a deployment, use the `kubectl set volume` command:

```
[user@host ~]$ kubectl set volume deployment/mydcname --add \  
-t secret -m /path/to/mount/volume \  
--name myvol --secret-name mysecret
```

## Application Configuration Options

Use configuration maps to store configuration data in plain text and if the information is not sensitive. Use secrets if the information you are storing is sensitive.

If your application only has a few simple configuration variables that can be read from environment variables or passed on the command line, then use environment variables to inject data from configuration maps and secrets. Environment variables are the preferred approach over mounting volumes inside the container.

However, if your application has a large number of configuration variables, or if you are migrating a legacy application that makes extensive use of configuration files, then use the volume mount approach instead of creating an environment variable for each of the configuration variables. For example, if your application expects one or more configuration files from a specific location on your file system, then you should create secrets or configuration maps from the configuration files and mount them inside the container ephemeral file system at the location that the application expects.

For example, to create a secret pointing to the `/home/student/configuration.properties` file, use the following command:

```
[user@host ~]$ kubectl create secret generic security \
--from-file /home/student/configuration.properties
```

To inject the secret into the application, configure a volume that refers to the secret created in the previous command. The volume must point to an actual directory inside the application where the secret's file is stored.

In the following example, the `configuration.properties` file is stored in the `/opt/app-root/secure` directory. To bind the file to the application, configure the deployment configuration from the application:

```
spec:
  template:
    spec:
      containers:
        - name: container
          image: repo.my/image-name
          volumeMounts:
            - mountPath: "/opt/app-root/secure"
              name: secure-volumen
              readOnly: true
      volumes:
        - name: secure-volumen
          secret:
            secretName: secret-name
```

To add the volume mount on a running application, you can use the `kubectl patch` command.

To create a configuration map, use the following command:

```
[user@host ~]$ kubectl create configmap properties \
--from-file /home/student/configuration.properties
```

To bind the application to the configuration map, update the deployment configuration from that application to use the configuration map:

```
[user@host ~]$ kubectl set env deployment/application \
--from configmap/properties
```



## References

### ConfigMaps

<https://kubernetes.io/docs/concepts/configuration/configmap/>

### Secrets

<https://kubernetes.io/docs/concepts/configuration/secret/>

## ► Guided Exercise

# Configuring Cloud Applications

In this exercise, you will use configuration maps and secrets to externalize the configuration for a containerized application.

## Outcomes

You should be able to:

- Deploy a simple Node.js-based application that prints configuration details from environment variables and files.
- Inject configuration data into the container using configuration maps and secrets.
- Change the data in the configuration map and verify that the application picks up the changed values.

## Before You Begin

Ensure that:

- Minikube and `kubectl` are running on your machine
- You have cloned the `D0100x-apps` repository
- You have executed the setup script located at `D0100x-apps/setup/operating-system/setup.sh`

Make sure your `kubectl` context uses the namespace `username-dev`. This allows you to execute `kubectl` commands directly into that namespace.

```
[user@host ~]$ kubectl config set-context --current --namespace=username-dev
```

## Instructions

### ► 1. Review the application source code and deploy the application.

- 1.1. Enter your local clone of the `D0100x-apps` Git repository.

```
[user@host ~]$ cd D0100x-apps
```

- 1.2. Inspect the `D0100x-apps/app-config/app/app.js` file.

The application reads the value of the `APP_MSG` environment variable and prints the contents of the `/opt/app-root/secure/myapp.sec` file:

```
const response = Value in the APP_MSG env var is => ${process.env.APP_MSG}\n;
...output omitted...
// Read in the secret file
fs.readFile('/opt/app-root/secure/myapp.sec', 'utf8', function (secerr, secdata) {
...output omitted...
```

13. Create a new deployment called app-config using the D0100x-apps/app-config/kubernetes/deployment.yml file.

```
[user@host D0100x-apps]$ kubectl apply -f app-config/kubernetes/deployment.yml
deployment.apps/app-config created
```

► 2. Test the application.

- 2.1. Expose the deployment on port 8080:

```
[user@host D0100x-apps]$ kubectl expose deployment/app-config --port 8080
service/app-config exposed
```

- 2.2. Modify the app-config/kubernetes/ingress.yml file to contain correct host value for your Kubernetes environment:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: app-config
  labels:
    app: app-config
spec:
  rules:
    - host: _INGRESS-HOST_
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: app-config
                port:
                  number: 8080
```

Replace `_INGRESS-HOST_` with the hostname associated with your Kubernetes cluster, such as `hello.example.com` or `app-config-USER-dev.apps.sandbox.x8i5.p1.openshiftapps.com`.

- 2.3. Create the ingress resource to be able to invoke the service just exposed:

```
[user@host D0100x-apps]$ kubectl create -f app-config/kubernetes/ingress.yml
ingress.networking.k8s.io/app-config created
```

- 2.4. Invoke the host URL by using the curl command:

```
[user@host D0100x-apps]$ curl hello.example.com
Value in the APP_MSG env var is => undefined
Error: ENOENT: no such file or directory, open '/opt/app-root/secure/myapp.sec'
```

The undefined value for the environment variable and the ENOENT: no such file or directory error are shown because neither the environment variable nor the file exists in the container.

- 3. Create the configuration map and secret resources.
- 3.1. Create a configuration map resource to hold configuration variables that store plain text data.  
Create a new configuration map resource called appconfmap. Store a key called APP\_MSG with the value Test Message in this configuration map:



**Note**

This course uses the backslash character (\) to break long commands. On Linux and macOS, you can use the line breaks.

On Windows, use the backtick character (`) to break long commands. Alternatively, do not break long commands.

Refer to *Orientation to the Classroom Environment* for more information about long commands.

```
[user@host D0100x-apps]$ kubectl create configmap appconfmap \
--from-literal APP_MSG="Test Message"
configmap/appconfmap created
```

- 3.2. Verify that the configuration map contains the configuration data:

```
[user@host D0100x-apps]$ kubectl describe cm/appconfmap
Name:  appconfmap
...output omitted...
Data
=====
APP_MSG:
...
Test Message
...output omitted...
```

- 3.3. Review the contents of the D0100x-apps/app-config/app/myapp.sec file:

```
username=user1
password=pass1
salt=xyz123
```

- 3.4. Create a new secret to store the contents of the myapp.sec file.

```
[user@host D0100x-apps]$ kubectl create secret generic appconffilesec \
--from-file app-config/app/myapp.sec
secret/appconffilesec created
```

- 3.5. Verify the contents of the secret. Note that the contents are stored in base64-encoded format:

```
[user@host D0100x-apps]$ kubectl get secret/appconffilesec -o json
{
    "apiVersion": "v1",
    "data": {
        "myapp.sec": "dXNlcj5hbWU9dXNlcjEKcGFzc3dvcmQ9cGFzcxEKc2..."
    },
    "kind": "Secret",
    "metadata": {
        ...output omitted...
        "name": "appconffilesec",
        ...output omitted...
    },
    "type": "Opaque"
}
```

- ▶ 4. Inject the configuration map and the secret into the application container.
- 4.1. Use the `kubectl set env` command to add the configuration map to the deployment configuration:
- ```
[user@host D0100x-apps]$ kubectl set env deployment/app-config \
--from configmap/appconffmap
deployment.apps/app-config env updated
```
- 4.2. Use the `kubectl patch` command to add the secret to the deployment configuration:  
Patch the app-config deployment using the following patch code. You can find this content in the D0100x-apps/app-config/kubernetes/secret.yml file.
- ```
[user@host D0100x-apps]$ kubectl patch deployment app-config \
--patch-file app-config/kubernetes/secret.yml
deployment.apps/app-config patched
```
- ▶ 5. Verify that the application is redeployed and uses the data from the configuration map and the secret.
- 5.1. Verify that the configuration map and secret were injected into the container. Retest the application using the route URL:

```
[user@host D0100x-apps]$ curl hello.example.com
Value in the APP_MSG env var is => Test Message
The secret is => username=user1
password=pass1
salt=xyz123
```

Kubernetes injects the configuration map as an environment variable and mounts the secret as a file into the container. The application reads the environment variable and file and then displays its data.

## Finish

Delete the created resources to clean your cluster. Kubernetes automatically deletes the associated pods.

```
[user@host ~]$ kubectl delete all,ingress -l app=app-config
pod "app-config-5cb9674bc5-wktrj" deleted
service "app-config" deleted
deployment.apps "app-config" deleted
ingress.networking.k8s.io "app-config" deleted
[user@host ~]$ kubectl delete cm appconfmap
configmap "appconfmap" deleted
[user@host ~]$ kubectl delete secret appconffilesec
secret "appconffilesec" deleted
```

This concludes the guided exercise.

## ► Quiz

# Configuring Cloud Applications

Choose the correct answers to the following questions:

- ▶ 1. **How can you manage sensitive configuration properties in Kubernetes? (Choose one.)**
  - a. By obfuscating sensitive information in the application code.
  - b. By using the `Secret` resource in Kubernetes.
  - c. By using the `ConfigMap` resource in Kubernetes.
  - d. By creating a Kubernetes volume and storing sensitive information as files in this volume.
- ▶ 2. **Which two of the following statements about configuration management in Kubernetes are correct? (Choose two.)**
  - a. Externalizing configuration allows you to decouple configuration from the application code.
  - b. Injecting configuration values into application containers as environment variables is generally recommended.
  - c. `ConfigMap` resources store information by using base64 encoding.
  - d. Storing sensitive information in a `Secret` resource adds multiple layers of security.
- ▶ 3. **Given the following command, which of the following statements is correct? (Choose one.)**

```
kubectl create configmap my_weather_app_config \
--from-literal WEATHER_HOST=value1 \
--from-literal WEATHER_STATION=value2 \
--from-literal WEATHER_UNITS=value3
```

- a. The command injects the configuration values as environment variables in the `my_weather_app_config` deployment pods.
- b. The command stores the configuration values in base64.
- c. The command stores the provided literal values in a new `configmap` resource called `my_weather_app_config`.
- d. The command injects the provided literal values as a volume in the `my_weather_app_config` deployment pods.

- 4. Assume you have an access token that is used to consume the GitHub API. The token is stored in a Kubernetes resource called `api_token`. Your team has recently regenerated the token in GitHub. Consequently, you must update the value of the `api_token` resource. Which two of the following commands are valid options? (Choose two.)
- a. `kubectl edit configmap/api_token`
  - b. `kubectl edit secret/api_token`
  - c. `kubectl set env deployment/my_app --from secret/api_token`
  - d. `kubectl patch configmap/api_token`
  - e. `kubectl patch secret/api_token`

## ► Solution

# Configuring Cloud Applications

Choose the correct answers to the following questions:

- ▶ 1. **How can you manage sensitive configuration properties in Kubernetes? (Choose one.)**
  - a. By obfuscating sensitive information in the application code.
  - b. By using the **Secret** resource in Kubernetes.
  - c. By using the **ConfigMap** resource in Kubernetes.
  - d. By creating a Kubernetes volume and storing sensitive information as files in this volume.
  
- ▶ 2. **Which two of the following statements about configuration management in Kubernetes are correct? (Choose two.)**
  - a. Externalizing configuration allows you to decouple configuration from the application code.
  - b. Injecting configuration values into application containers as environment variables is generally recommended.
  - c. **ConfigMap** resources store information by using base64 encoding.
  - d. Storing sensitive information in a **Secret** resource adds multiple layers of security.
  
- ▶ 3. **Given the the following command, which of the following statements is correct? (Choose one.)**

```
kubectl create configmap my_weather_app_config \
--from-literal WEATHER_HOST=value1 \
--from-literal WEATHER_STATION=value2 \
--from-literal WEATHER_UNITS=value3
```

- a. The command injects the configuration values as environment variables in the **my\_weather\_app\_config** deployment pods.
- b. The command stores the configuration values in base64.
- c. The command stores the provided literal values in a new **configmap** resource called **my\_weather\_app\_config**.
- d. The command injects the provided literal values as a volume in the **my\_weather\_app\_config** deployment pods.

- 4. Assume you have an access token that is used to consume the GitHub API. The token is stored in a Kubernetes resource called `api_token`. Your team has recently regenerated the token in GitHub. Consequently, you must update the value of the `api_token` resource. Which two of the following commands are valid options? (Choose two.)
- a. `kubectl edit configmap/api_token`
  - b. `kubectl edit secret/api_token`
  - c. `kubectl set env deployment/my_app --from secret/api_token`
  - d. `kubectl patch configmap/api_token`
  - e. `kubectl patch secret/api_token`

# Summary

---

In this chapter, you learned:

- Deployments in Kubernetes can include resource limits to ensure the application gets enough resources or is restricted from using too many.
- Application probes can be configured to facilitate monitoring application health and readiness.

## Chapter 4

# Implementing Cloud Deployment Strategies

### Goal

Compare different Cloud Deployment Strategies

### Objectives

Review what deployment strategies can be used in the Cloud, what they are used for and their advantages.

### Sections

- Implementing Cloud Deployment Strategies (and Guided Exercise,Quiz,Quiz)
- End of Course Quiz (Quiz)

# Implementing Cloud Deployment Strategies

---

## Objectives

After completing this section, you should be able to review what deployment strategies can be used in the Cloud, what they are used for and their advantages.

## Deployment Strategies in Kubernetes

A deployment strategy is a method of changing or upgrading an application. The objective is to make changes or upgrades with minimal downtime and with reduced impact on end users.

Kubernetes provides several deployment strategies. These strategies are organized into two primary categories:

- By using the deployment strategy defined in the application deployment.
- By using the Kubernetes router to route traffic to specific application pods.

Strategies defined within the deployment impact all routes that use the application. Strategies that use router features affect individual routes.

The following are strategies that are defined in the application deployment:

### **Rolling**

The rolling strategy is the default strategy.

This strategy progressively replaces instances of the previous version of an application with instances of the new version of the application. It uses the configured readiness probe to determine when the new pod is ready. After the probe for the new pod succeeds, the deployment controller terminates an old pod.

If a significant issue occurs, the deployment controller aborts the rolling deployment.

Rolling deployments are a type of canary deployment. By using the readiness probe, Kubernetes tests a new version before replacing all of the old instances. If the readiness probe never succeeds, then Kubernetes removes the canary instance and rolls back the deployment.

Use a rolling deployment strategy when:

- You require no downtime during an application update.
- Your application supports running an older version and a newer version at the same time.

### **Recreate**

With this strategy, Kubernetes first stops all pods running the application and then creates pods with the new version. This strategy creates down time because there is a time period with no running instances of your application.

Use a recreate deployment strategy when:

- Your application does not support running multiple different versions simultaneously.
- Your application uses a persistent volume with **ReadWriteOnce** (RWO) access mode, which does not allow writes from multiple pods.

You can configure the strategy in the `Deployment` object, for example by using the YAML manifest file:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: hello
  name: hello
spec:
  replicas: 4
  selector:
    matchLabels:
      app: hello
  strategy:
    type: RollingUpdate①
    rollingUpdate:②
      maxSurge: 50%③
      maxUnavailable: 10%④
  template:
    metadata:
      labels:
        app: hello
    spec:
      containers:
        - image: quay.io/redhattraining/versioned-hello:v1.1
          name: versioned-hello
```

- ① Defining the `RollingUpdate` strategy for the `hello` deployment.
- ② The `RollingUpdate` strategy accepts the `rollingUpdate` object to configure further strategy parameters.
- ③ The `maxSurge` parameter sets the maximum number of pods that can be scheduled above the desired number of pods. This deployment configures 4 pods. Consequently, 2 new pods can be created at a time.
- ④ The `maxUnavailable` parameter sets the maximum number of pods that can be unavailable during the update. Kubernetes calculates the absolute number from the configured percentage by rounding down. Consequently, `maxUnavailable` is set to 0 with the current deployment parameters.

Use the `kubectl describe` command to view the details of a deployment strategy:

```
[user@host ~] kubectl describe deploy/hello
...output omitted...
StrategyType:           RollingUpdate
MinReadySeconds:         0
RollingUpdateStrategy:   10% max unavailable, 50% max surge
...output omitted...
```

## Implementing Advanced Deployment Strategies Using the Kubernetes Router

The following are advanced deployment strategies that use the Kubernetes router:

### Blue-green Deployment

With blue-green deployments, two identical environments run concurrently. Each environment is labeled either blue or green and runs a different version of the application.

For example, the Kubernetes router is used to direct traffic from the current version labeled green to the newer version labeled blue. During the next update, the current version is labeled blue and the new version is labeled green.

At any given point, the exposed route points to one of the services and can be swapped to point to a different service. This allows you to test the new version of your application service before routing traffic to it. When your new application version is ready, simply swap the router to point to the updated service.

### A/B Deployment

The A/B deployment strategy allows you to deploy a new version of the application for a limited set of users. You can configure Kubernetes to route a percentage of requests between two different deployed versions of an application.

By controlling the portion of requests sent to each version, you can gradually increase the traffic sent to the new version. Once the new version receives all traffic, the old version is removed.



### References

#### Kubernetes documentation on deployment strategies

<https://kubernetes.io/docs/concepts/workloads/controllers/deployment/#strategy>

#### What is blue green deployment?

<https://www.redhat.com/en/topics/devops/what-is-blue-green-deployment>

## ► Guided Exercise

# Implementing Cloud Deployment Strategies

In this exercise you will deploy a managed containerized application in your Kubernetes cluster. You will observe how some of Kubernetes' automatic deployments and high availability features work.

## Outcomes

You should be able to:

- Deploy an application container with several replicas.
- Review the structure of the `Deployment` resource manifest.
- Update the application to a new version without losing availability.

## Before You Begin

You need a working Kubernetes cluster, and your `kubectl` command must be configured to communicate with the cluster.

Make sure your `kubectl` context refers to a namespace where you have enough permissions, usually `username-dev` or `username-stage`. Use the `kubectl config set-context --current --namespace=namespace` command to switch to the appropriate namespace.

## Instructions

- 1. Deploy a Node.js application container to your Kubernetes cluster.
- 1.1. Use the `kubectl create` command to create a new application with the following parameters:



### Note

This course uses the backslash character (\) to break long commands. On Linux and macOS, you can use the line breaks.

On Windows, use the backtick character (`) to break long commands. Alternatively, do not break long commands.

Refer to *Orientation to the Classroom Environment* for more information about long commands.

```
[user@host ~]$ kubectl create deployment do100-multi-version \
--replicas=5 \
--image quay.io/redhattraining/do100-multi-version:v1-external
deployment.apps/do100-multi-version created
```

- 1.2. Wait until the pod is deployed. The pod should be in the READY state.

```
[user@host ~]$ kubectl get pods
NAME                               READY   STATUS    RESTARTS   AGE
do100-multi-version-788cb59f94-54lcz 1/1     Running   0          4s
do100-multi-version-788cb59f94-cv4dd 1/1     Running   0          4s
do100-multi-version-788cb59f94-nt2lh 1/1     Running   0          4s
do100-multi-version-788cb59f94-snxf4 1/1     Running   0          4s
do100-multi-version-788cb59f94-x7k7n 1/1     Running   0          4s
```

Note that the exact names of your pods will likely differ from the previous example.

- 1.3. Review the application logs to see the running version.

```
[user@host ~] kubectl logs deploy/do100-multi-version
Found 5 pods, using pod/do100-multi-version-788cb59f94-54lcz

> multi-version@1.0.0 start /opt/app-root/src
> node app.js

do100-multi-version server running version 1.0 on  http://0.0.0.0:8080
```

Note the version number that the application logs.

## ▶ 2. Edit the deployment to change the application version and add a readiness probe.

- 2.1. Verify that the deployment strategy for the application is `RollingUpdate`:

```
[user@host ~]$ kubectl describe deploy/do100-multi-version
...output omitted...
StrategyType: RollingUpdate
...output omitted...
```

- 2.2. Use `kubectl edit` to modify the deployment resource.

```
[user@host ~] kubectl edit deployment/do100-multi-version
...output omitted...
```

- 2.3. Update the version of the image to `v2-external`. Additionally, configure a readiness probe so that you can watch the new deployment as it happens.

Your deployment resource should look like the following:

```
...output omitted...
spec:
  ...output omitted...
  template:
    ...output omitted...
    spec:
      containers:
        - image: quay.io/redhattraining/do100-multi-version: v2-external
          ...output omitted...
      readinessProbe:
        httpGet:
```

```

    path: /ready
    port: 8080
    initialDelaySeconds: 2
    timeoutSeconds: 2
  
```

When you are done, save your changes and close the editor.

- 3. Verify that the new version of the application is deployed via the rolling deployment strategy.

- 3.1. Watch the pods as Kubernetes redeploys the application.

```
[user@host ~]$ kubectl get pods -w
NAME                               READY   STATUS        RESTARTS   AGE
do100-multi-version-788cb59f94-54lcz   1/1    Running      0          5m28s
...output omitted...
do100-multi-version-8477f6f4bb-1nlcz   0/1    ContainerCreating   0          3s
...output omitted...
do100-multi-version-8477f6f4bb-1nlcz   0/1    Running      0          5s
do100-multi-version-8477f6f4bb-1nlcz   1/1    Running      0          40s
...output omitted...
do100-multi-version-788cb59f94-54lcz   0/1    Terminating   0          6m8s
...output omitted...
```

As the new application pods start and become ready, pods running the older version are terminated. Note that the application takes about thirty seconds to enter the ready state.

Press **Ctrl+C** to stop the command.

- 3.2. View the logs of the new version of the application.

```
[user@host ~] kubectl logs deploy/do100-multi-version
Found 5 pods, using pod/do100-multi-version-8477f6f4bb-9h5xl

> multi-version@1.0.0 start /opt/app-root/src
> node app.js

do100-multi-version server running version 2.0 on  http://0.0.0.0:8080
```

## Finish

Delete the deployment to clean your cluster. Kubernetes automatically deletes the associated pods.

```
[user@host ~]$ kubectl delete deploy/do100-multi-version
deployment.apps "do100-multi-version" deleted
```

This concludes the guided exercise.

## ► Quiz

# Implementing Cloud Deployment Strategies

Choose the correct answers to the following questions:

- 1. Assume you maintain an application that does not support running multiple different versions at the same time. Which of the following deployment strategies is the most suitable for updating the application? (Choose one.)
- a. Rolling
  - b. Blue-green deployment
  - c. Recreate
  - d. A/B deployment
- 2. Consider the following Deployment resource configuration. Which two of the following statements are correct? (Choose two.)

```
spec:  
  replicas: 10  
  strategy:  
    type: RollingUpdate  
    rollingUpdate:  
      maxSurge: 40%  
      maxUnavailable: 20%
```

- a. During a deployment update, a maximum of four new pods can be scheduled.
- b. During a deployment update, a maximum of two pods can be unavailable.
- c. During a deployment update, a maximum of two new pods can be scheduled.
- d. During a deployment update, a maximum of four pods can be unavailable.

► 3. Based on the following scenario, which two of the following statements are correct?  
(Choose two.)

Your team uses Kubernetes to run a web application: `your-app:v1`.

The deployment strategy for this application is `RollingUpdate`.

Now, your team wants to deploy `your-app:v2`.

Therefore, you update the `Deployment` resource to use the `v2` version.

- a. During the update, the server responds with a `503 Service Unavailable` error code, because Kubernetes has to stop old pods before creating the new ones.
- b. During the update, you might see `v1` pods until Kubernetes creates all the new `v2` pods.
- c. Pods that use both the `v1` and `v2` versions stay running after the deployment update succeeds.
- d. Kubernetes determines when new `v2` pods are ready before terminating `v1` pods.

► 4. Your team just finished the development of a new feature. Instead of delivering the new feature to all users, you decide to test the feature first with a limited set of users. Which of the following deployment strategies should you use? (Choose one.)

- a. Recreate
- b. Rolling
- c. Blue-green deployment
- d. A/B deployment

## ► Solution

# Implementing Cloud Deployment Strategies

Choose the correct answers to the following questions:

- 1. Assume you maintain an application that does not support running multiple different versions at the same time. Which of the following deployment strategies is the most suitable for updating the application? (Choose one.)
- a. Rolling
  - b. Blue-green deployment
  - c. Recreate
  - d. A/B deployment
- 2. Consider the following Deployment resource configuration. Which two of the following statements are correct? (Choose two.)

```
spec:  
  replicas: 10  
  strategy:  
    type: RollingUpdate  
    rollingUpdate:  
      maxSurge: 40%  
      maxUnavailable: 20%
```

- a. During a deployment update, a maximum of four new pods can be scheduled.
- b. During a deployment update, a maximum of two pods can be unavailable.
- c. During a deployment update, a maximum of two new pods can be scheduled.
- d. During a deployment update, a maximum of four pods can be unavailable.

► 3. Based on the following scenario, which two of the following statements are correct?  
(Choose two.)

Your team uses Kubernetes to run a web application: `your-app:v1`.

The deployment strategy for this application is `RollingUpdate`.

Now, your team wants to deploy `your-app:v2`.

Therefore, you update the `Deployment` resource to use the `v2` version.

- a. During the update, the server responds with a `503 Service Unavailable` error code, because Kubernetes has to stop old pods before creating the new ones.
- b. During the update, you might see `v1` pods until Kubernetes creates all the new `v2` pods.
- c. Pods that use both the `v1` and `v2` versions stay running after the deployment update succeeds.
- d. Kubernetes determines when new `v2` pods are ready before terminating `v1` pods.

► 4. Your team just finished the development of a new feature. Instead of delivering the new feature to all users, you decide to test the feature first with a limited set of users. Which of the following deployment strategies should you use? (Choose one.)

- a. Recreate
- b. Rolling
- c. Blue-green deployment
- d. A/B deployment

## ► Quiz

# End of Course Quiz

Choose the correct answers to the following questions:

- 1. **Which two of the following Kubernetes features simplify the lifecycle management of containerized applications? (Choose two.)**
- a. Restarting of unresponsive containers
  - b. Container registry integration
  - c. Application rollout strategies
  - d. Decreased resource consumption
- 2. **Which of the following statements about application lifecycle management in Kubernetes is correct? (Choose one.)**
- a. If the state of a Kubernetes deployment changes, for example due to a node failure, developers must adjust the deployment state manually.
  - b. Developers declare the application state in a resource manifest. Kubernetes ensures the application state is identical to the declared state. This is the declarative approach.
  - c. Developers declare the application state in a resource manifest. Additionally, developers must declare how to achieve that state. This is the imperative approach.
  - d. Kubernetes uses the declarative approach for some resources, such as the Deployment resource, and the imperative approach for other resources, such as the Pod resource.
- 3. **Your task is to create an application that uses the provided parameters. Which of the following commands is correct? (Choose one.)**

Application name is lb-v1.

Application is based on the lb:v1 image.

Application should restart automatically if it crashes.

- a. kubectl run lb-v1 -- lb:v1
- b. kubectl run deployment lb-v1 --image lb:v1
- c. kubectl create deployment lb-v1 --image lb:v1
- d. kubectl run lb-v1 --image lb:v1

- 4. Consider the provided resource manifest. Which two of the following statements about the manifest are correct? (Choose two.)

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: sso
  name: sso
spec:
  replicas: 2
  selector:
    matchLabels:
      app: sso
  template:
    metadata:
      labels:
        app: sso
    spec:
      containers:
        - image: redhat/sso:latest
          name: redhat-sso
```

- a. The provided manifest declares a Pod resource.
  - b. The provided manifest declares a Deployment resource that manages two Pod replicas.
  - c. The resource uses the redhat-sso container image.
  - d. The resource uses the redhat/sso:latest container image.
- 5. Consider the previously provided resource manifest. Which two of the following statements about the manifest are correct? (Choose two.)
- a. The spec.replicas property configures the number of replicas.
  - b. The spec.template.replicas property configures the number of replicas.
  - c. The metadata.labels property configures the labels applied to pods.
  - d. The spec.template.metadata.labels configures the label applied to pods.

- 6. You tried to create a Deployment resource by using the kubectl utility. However, kubectl returns the provided error message. Which of the following statements is the likely cause of the error message? (Choose one.)

```
[user@host ~] $ kubectl create -f deployment.yaml
The Deployment "sso" is invalid: spec.template.metadata.labels: Invalid value:
map[string]string{"app":"keycloak"}: `selector` does not match template `labels`
```

- a. Kubectl is likely not authorized to create resources in the Kubernetes cluster.
- b. The resource manifest is misconfigured. The `spec.template.metadata.labels` property likely does not match the `spec.selector.matchLabels` property.
- c. The command is incorrect. The correct command is `kubectl apply`.
- d. The Kubernetes cluster likely experienced a transient fault state. To fix the issue, issue the command again.

- 7. Which of the following statements about Kubernetes networking and containers is correct? (Choose one.)

- a. Each container in a Kubernetes cluster has a unique IP address.
- b. Each Kubernetes pod has a unique IP address.
- c. Pods can reach other pods only on the same Kubernetes node.
- d. Multiple pods cannot use the same port on one Kubernetes node.

- 8. Your task is to connect the sso and api application pods. Based on the provided requirements, which of the following solutions is the most suitable? (Choose one.)

The sso pods must reach api pods.  
The api pods scale up at 4pm due to peak traffic.  
The api pods scale down at 9pm due to reduced traffic.  
All traffic is internal to the Kubernetes cluster.

- a. Both application pods must rely on unique pod IP addresses.
- b. Create a Service resource of type LoadBalancer with a cloud provider and route the traffic externally to one of the pods.
- c. Create a Service resource of type NodePort and route the traffic externally to one of the pods.
- d. Create a Service resource of type ClusterIP that routes requests to the api pods.  
This ensures that the sso pods always reach the api pods.

► **9. Based on the provided description, which of the following commands creates a Service resource? (Choose one.)**

Route requests to a deployment named portal.  
Listen on port 443.  
Route request to port 8443.

- a. kubectl create service portal --port 443 --target-port 8443
- b. kubectl expose deployment/portal --port 443 --target-port 8443
- c. kubectl expose deployment/portal --port 8443 --target-port 443
- d. kubectl expose portal --listen-port 443 --route-port 8443.

► **10. Which two of the following statements about the Kubernetes Service resource are correct? (Choose two.)**

- a. ClusterIP type services simplify discovering IP addresses of new pods by providing a stable IP address that routes requests to a set of specified pods.
- b. Pods can reach services by using DNS names.
- c. Multiple services in one namespace cannot use the same port.
- d. The port and targetPort service configurations must use the same port number.

► **11. Which of the following statements about the Kubernetes Ingress resource is correct? (Choose one.)**

- a. The Ingress and Service resources are identical.
- b. The Ingress resource replaces the deprecated Service resource.
- c. The Ingress resource routes external traffic into the Kubernetes cluster.
- d. The Ingress resource cannot use the Service resource to route requests.

► **12. Which two of the following statements about resource limits and resource requests in Kubernetes are correct? (Choose two.)**

- a. You cannot limit resources that your applications consume in a Kubernetes cluster.
- b. Resource requests configure the global minimum resources for any application in a Kubernetes cluster.
- c. Resource requests configure the minimum resources for an application container.
- d. Resource limits prevent an application container from consuming more resources than configured.

- 13. You are in charge of operating an application that is running in a Kubernetes cluster. You discover that the application becomes unresponsive after around 3000 served clients, probably due to a memory leak. Which of the following statements is a suitable temporary solution for the issue until the core development team fixes the issue? (Choose one.)
- a. You must manually monitor the application and restart it when it becomes unresponsive.
  - b. You can configure a startup probe and restart the application if it fails. This is useful because the issue happens before the application starts.
  - c. You can configure a readiness probe and stop routing traffic to the pod if the pod becomes unresponsive. This is useful because you can examine the issue when it happens.
  - d. You can configure a liveness probe and restart the application when it becomes unresponsive. This is useful because it minimizes the downtime of the application without the need for manual intervention.
- 14. Which two of the following statements about the ConfigMap and Secret resources are correct? (Choose two.)
- a. The ConfigMap resource stores data by using the base64 encoding.
  - b. The Secret resource stores data in an encrypted format.
  - c. The Secret resource stores data by using the base64 encoding.
  - d. The ConfigMap resource is suitable for storing non-sensitive data.
- 15. Which two of the following statements are correct ways of exposing the ConfigMap and Secret resources to your application? (Choose two.)
- a. You can inject the values as environment variables.
  - b. You can expose the values by using the etcd database.
  - c. You can expose the values by using the kube-api service.
  - d. You can mount all keys as files.
- 16. Which two of the following statements about externalizing application configuration are correct? (Choose two.)
- a. Externalizing values like passwords is not always beneficial because it makes such values harder to find.
  - b. Externalizing application configuration means removing the values from application source code and reading the values at runtime, for example from environment variables.
  - c. Developers can use the ConfigMap and Secret Kubernetes resources to externalize application configuration.
  - d. Applications that externalize values like database credentials are difficult to deploy in varying environments, such as dev, stage, and prod.

- 17. Your team just finished the development of a new feature. You decide to test the feature by using a production environment. However, you do not want to expose the feature to users. Which of the following deployment strategies should you use? (Choose one.)

- a. Recreate
- b. Rolling
- c. Blue-green deployment
- d. A/B deployment

- 18. Consider the following Deployment resource configuration. Which of the following statements is correct? (Choose one.)

```
spec:  
  replicas: 10  
  strategy: {}
```

- a. You cannot update the deployment because the resource does not specify an update strategy.
- b. The deployment configuration is invalid because the manifest does not specify an update strategy.
- c. The update strategy defaults to the Recreate strategy.
- d. The update strategy defaults to the RollingUpdate strategy.

- 19. Which two of the following statements about deployment strategies are correct? (Choose two.)

- a. Developers configure all deployment strategies, such as the Recreate and A/B Deployment strategies, in the Deployment resource manifest.
- b. Developers configure some deployment strategies, such as the Recreate and RollingUpdate strategies, in the Deployment resource manifest.
- c. Developers must configure advanced deployment strategies by using the Kubernetes ingress router.
- d. All applications can always use all deployment strategies.

- 20. Which of the following commands shows you the deployment strategy of a Deployment resource in a Kubernetes cluster? (Choose one.)

- a. kubectl get deployment example
- b. kubectl describe deployment
- c. kubectl describe deployment example
- d. kubectl logs deployment example

## ► Solution

# End of Course Quiz

Choose the correct answers to the following questions:

- 1. **Which two of the following Kubernetes features simplify the lifecycle management of containerized applications? (Choose two.)**
- a. Restarting of unresponsive containers
  - b. Container registry integration
  - c. Application rollout strategies
  - d. Decreased resource consumption
- 2. **Which of the following statements about application lifecycle management in Kubernetes is correct? (Choose one.)**
- a. If the state of a Kubernetes deployment changes, for example due to a node failure, developers must adjust the deployment state manually.
  - b. Developers declare the application state in a resource manifest. Kubernetes ensures the application state is identical to the declared state. This is the declarative approach.
  - c. Developers declare the application state in a resource manifest. Additionally, developers must declare how to achieve that state. This is the imperative approach.
  - d. Kubernetes uses the declarative approach for some resources, such as the Deployment resource, and the imperative approach for other resources, such as the Pod resource.
- 3. **Your task is to create an application that uses the provided parameters. Which of the following commands is correct? (Choose one.)**

Application name is lb-v1.

Application is based on the lb:v1 image.

Application should restart automatically if it crashes.

- a. kubectl run lb-v1 -- lb:v1
- b. kubectl run deployment lb-v1 --image lb:v1
- c. kubectl create deployment lb-v1 --image lb:v1
- d. kubectl run lb-v1 --image lb:v1

► 4. Consider the provided resource manifest. Which two of the following statements about the manifest are correct? (Choose two.)

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: sso
  name: sso
spec:
  replicas: 2
  selector:
    matchLabels:
      app: sso
  template:
    metadata:
      labels:
        app: sso
    spec:
      containers:
        - image: redhat/sso:latest
          name: redhat-sso
```

- a. The provided manifest declares a Pod resource.
- b. The provided manifest declares a **Deployment** resource that manages two Pod replicas.
- c. The resource uses the **redhat-sso** container image.
- d. The resource uses the **redhat/sso:latest** container image.

► 5. Consider the previously provided resource manifest. Which two of the following statements about the manifest are correct? (Choose two.)

- a. The **spec.replicas** property configures the number of replicas.
- b. The **spec.template.replicas** property configures the number of replicas.
- c. The **metadata.labels** property configures the labels applied to pods.
- d. The **spec.template.metadata.labels** configures the label applied to pods.

- 6. You tried to create a Deployment resource by using the kubectl utility. However, kubectl returns the provided error message. Which of the following statements is the likely cause of the error message? (Choose one.)

```
[user@host ~] $ kubectl create -f deployment.yaml
The Deployment "sso" is invalid: spec.template.metadata.labels: Invalid value:
map[string]string{"app":"keycloak"}: `selector` does not match template `labels`
```

- a. Kubectl is likely not authorized to create resources in the Kubernetes cluster.
- b. The resource manifest is misconfigured. The `spec.template.metadata.labels` property likely does not match the `spec.selector.matchLabels` property.
- c. The command is incorrect. The correct command is `kubectl apply`.
- d. The Kubernetes cluster likely experienced a transient fault state. To fix the issue, issue the command again.

- 7. Which of the following statements about Kubernetes networking and containers is correct? (Choose one.)

- a. Each container in a Kubernetes cluster has a unique IP address.
- b. Each Kubernetes pod has a unique IP address.
- c. Pods can reach other pods only on the same Kubernetes node.
- d. Multiple pods cannot use the same port on one Kubernetes node.

- 8. Your task is to connect the sso and api application pods. Based on the provided requirements, which of the following solutions is the most suitable? (Choose one.)

The sso pods must reach api pods.  
The api pods scale up at 4pm due to peak traffic.  
The api pods scale down at 9pm due to reduced traffic.  
All traffic is internal to the Kubernetes cluster.

- a. Both application pods must rely on unique pod IP addresses.
- b. Create a Service resource of type LoadBalancer with a cloud provider and route the traffic externally to one of the pods.
- c. Create a Service resource of type NodePort and route the traffic externally to one of the pods.
- d. Create a Service resource of type ClusterIP that routes requests to the api pods.  
This ensures that the sso pods always reach the api pods.

► **9. Based on the provided description, which of the following commands creates a Service resource? (Choose one.)**

Route requests to a deployment named portal.  
Listen on port 443.  
Route request to port 8443.

- a. kubectl create service portal --port 443 --target-port 8443
- b. kubectl expose deployment/portal --port 443 --target-port 8443
- c. kubectl expose deployment/portal --port 8443 --target-port 443
- d. kubectl expose portal --listen-port 443 --route-port 8443.

► **10. Which two of the following statements about the Kubernetes Service resource are correct? (Choose two.)**

- a. ClusterIP type services simplify discovering IP addresses of new pods by providing a stable IP address that routes requests to a set of specified pods.
- b. Pods can reach services by using DNS names.
- c. Multiple services in one namespace cannot use the same port.
- d. The port and targetPort service configurations must use the same port number.

► **11. Which of the following statements about the Kubernetes Ingress resource is correct? (Choose one.)**

- a. The Ingress and Service resources are identical.
- b. The Ingress resource replaces the deprecated Service resource.
- c. The Ingress resource routes external traffic into the Kubernetes cluster.
- d. The Ingress resource cannot use the Service resource to route requests.

► **12. Which two of the following statements about resource limits and resource requests in Kubernetes are correct? (Choose two.)**

- a. You cannot limit resources that your applications consume in a Kubernetes cluster.
- b. Resource requests configure the global minimum resources for any application in a Kubernetes cluster.
- c. Resource requests configure the minimum resources for an application container.
- d. Resource limits prevent an application container from consuming more resources than configured.

- 13. You are in charge of operating an application that is running in a Kubernetes cluster. You discover that the application becomes unresponsive after around 3000 served clients, probably due to a memory leak. Which of the following statements is a suitable temporary solution for the issue until the core development team fixes the issue? (Choose one.)
- a. You must manually monitor the application and restart it when it becomes unresponsive.
  - b. You can configure a startup probe and restart the application if it fails. This is useful because the issue happens before the application starts.
  - c. You can configure a readiness probe and stop routing traffic to the pod if the pod becomes unresponsive. This is useful because you can examine the issue when it happens.
  - d. You can configure a liveness probe and restart the application when it becomes unresponsive. This is useful because it minimizes the downtime of the application without the need for manual intervention.
- 14. Which two of the following statements about the ConfigMap and Secret resources are correct? (Choose two.)
- a. The ConfigMap resource stores data by using the base64 encoding.
  - b. The Secret resource stores data in an encrypted format.
  - c. The Secret resource stores data by using the base64 encoding.
  - d. The ConfigMap resource is suitable for storing non-sensitive data.
- 15. Which two of the following statements are correct ways of exposing the ConfigMap and Secret resources to your application? (Choose two.)
- a. You can inject the values as environment variables.
  - b. You can expose the values by using the etcd database.
  - c. You can expose the values by using the kube-api service.
  - d. You can mount all keys as files.
- 16. Which two of the following statements about externalizing application configuration are correct? (Choose two.)
- a. Externalizing values like passwords is not always beneficial because it makes such values harder to find.
  - b. Externalizing application configuration means removing the values from application source code and reading the values at runtime, for example from environment variables.
  - c. Developers can use the ConfigMap and Secret Kubernetes resources to externalize application configuration.
  - d. Applications that externalize values like database credentials are difficult to deploy in varying environments, such as dev, stage, and prod.

- 17. Your team just finished the development of a new feature. You decide to test the feature by using a production environment. However, you do not want to expose the feature to users. Which of the following deployment strategies should you use? (Choose one.)

- a. Recreate
- b. Rolling
- c. Blue-green deployment
- d. A/B deployment

- 18. Consider the following Deployment resource configuration. Which of the following statements is correct? (Choose one.)

```
spec:  
  replicas: 10  
  strategy: {}
```

- a. You cannot update the deployment because the resource does not specify an update strategy.
- b. The deployment configuration is invalid because the manifest does not specify an update strategy.
- c. The update strategy defaults to the Recreate strategy.
- d. The update strategy defaults to the RollingUpdate strategy.

- 19. Which two of the following statements about deployment strategies are correct? (Choose two.)

- a. Developers configure all deployment strategies, such as the Recreate and A/B Deployment strategies, in the Deployment resource manifest.
- b. Developers configure some deployment strategies, such as the Recreate and RollingUpdate strategies, in the Deployment resource manifest.
- c. Developers must configure advanced deployment strategies by using the Kubernetes ingress router.
- d. All applications can always use all deployment strategies.

- 20. Which of the following commands shows you the deployment strategy of a Deployment resource in a Kubernetes cluster? (Choose one.)

- a. kubectl get deployment example
- b. kubectl describe deployment
- c. kubectl describe deployment example
- d. kubectl logs deployment example

# Summary

---

In this chapter, you learned:

- Deployments can use one of several types of deployment strategies.
- These strategies dictate how new versions of an application are rolled out.

## **Appendix A**

# **Installing and Configuring Kubernetes**

## ► Guided Exercise

# Installing and Configuring Kubernetes

In this exercise you will prepare your development environment to use a local or remote Kubernetes instance.

### Outcomes

You should be able to:

- Install a local Kubernetes instance by using `minikube` on Linux, macOS or Windows.
- Register for using a remote Kubernetes instance by using `Developer Sandbox` for Red Hat OpenShift.

## Instructions

Deploying a fully developed, multi-node Kubernetes cluster typically requires significant time and compute resources. With `minikube`, you can quickly deploy a local Kubernetes cluster, allowing you to focus on learning Kubernetes operations and application development.

`minikube` is an open source utility that allows you to quickly deploy a local Kubernetes cluster on your personal computer. By using virtualization technologies, `minikube` creates a *virtual machine* (VM) that contains a single-node Kubernetes cluster. VMs are virtual computers and each VM is allocated its own system resources and operating system.

The latest `minikube` releases also allow you to create your cluster by using containers instead of virtual machines. Nevertheless, this solution is still not mature, and it is not supported for this course.

`minikube` is compatible with Linux, macOS, and Windows.

To install `minikube` on your system, you will need:

- An Internet connection
- At least 2 GB of free memory
- 2 CPUs or more
- At least 20 GB of free disk space
- A locally installed hypervisor (using a container runtime is not supported in this course)

Before installing `minikube`, a hypervisor technology must be installed or enabled on your local system. A **hypervisor** is software that creates and manages virtual machines (VMs) on a shared physical hardware system. The hypervisor pools and isolates hardware resources for VMs, allowing many VMs to run on a shared physical hardware system, such as a server.

### ► 1. Using minikube in Linux-based systems

In this course we support Fedora Linux 33 and 34, as well as Red Hat Enterprise Linux 8.

#### 1.1. Installing a Hypervisor on Linux

The preferred hypervisor for Linux systems is `kvm2` [<https://minikube.sigs.k8s.io/docs/drivers/kvm2/>]. `minikube` communicates with the hypervisor using the `libvirt` virtualization API libraries.



**Note**

Prefix the following commands with `sudo` if you are running a user without administrative privileges.

Use your system package manager to install the complete set of virtualization libraries:

```
[root@host ~]# dnf install @virtualization
```

or select the minimum set of required libraries

```
[root@host ~]# dnf install qemu-kvm libvirt libvirt-python libguestfs-tools  
virt-install
```

Start the `libvirtd` service:

```
[root@host ~]# systemctl start libvirtd  
...output omitted...  
[root@host ~]# systemctl enable libvirtd  
...output omitted...
```

## 1.2. Installing minikube on Linux

There are three alternatives to install `minikube` in a Linux system:

- If your system contains a package manager or software manager including `minikube`, then use it and verify the version matches the minimum requirements.

```
[root@host ~]# dnf install minikube
```

- If the repositories for your package manager do not include an appropriate version for `minikube`, then go to <https://github.com/kubernetes/minikube/releases> and download the latest release matching your operating system.

For Debian-based systems, download the file `minikube_VERSION_[amd64|arm64|armhf|ppc64el|s390x].deb` and install it using the `dpkg` command:

```
[root@host ~]# dpkg -i FILE
```

For RPM-based distributions, download the file `minikube-VERSION.[aarch64|armv7hl|ppc64le|s390x|x86_64].rpm` and install it using the `rpm` command:

```
[root@host ~]# rpm -Uvh FILE
```

- Alternatively, download the binary `minikube-linux-[amd64|arm|arm64]` file and install using the `install` command:

```
[root@host ~]# install FILE /usr/local/bin/minikube
```

### 1.3. Starting Your minikube Cluster on Linux

To initialize your `minikube` cluster, use the `minikube start` command.

```
[root@host ~]# minikube start --driver=kvm2
# minikube v1.20.0 on Fedora 33
# Using the kvm2 driver based on user configuration
# Starting control plane node minikube in cluster minikube
# Creating kvm2 VM (CPUs=4, Memory=16384MB, Disk=20000MB) ...
# Preparing Kubernetes v1.20.2 on Docker 20.10.6 ...
  • Generating certificates and keys ...
  • Booting up control plane ...
  • Configuring RBAC rules ...
# Verifying Kubernetes components...
  • Using image gcr.io/k8s-minikube/storage-provisioner:v5
# Enabled addons: storage-provisioner, default-storageclass
# Done! kubectl is now configured to use "minikube" cluster and "default"
namespace by default
```



#### Note

To set the default driver, run the command `minikube config set driver DRIVER`.

## ► 2. Using minikube on macOS systems

This course supports macOS Big Sur (version 11.3.1) and later minor updates.

### 2.1. Installing a Hypervisor on macOS

Despite Docker being the preferred installation method in macOS, current `minikube` versions do not support all the features required for this course when running the `docker` driver (to be precise, `ingress` add-on is not yet supported). For this reason, we are using the Oracle VM VirtualBox hypervisor for running the virtual machines needed to support `minikube` on macOS.

Oracle VM VirtualBox is a free and open source hypervisor available for macOS systems. To install Oracle VM VirtualBox:

1. Download the latest version of VirtualBox for OS X hosts from <https://virtualbox.org/wiki/Downloads>
2. Open the downloaded `dmg` file and follow the onscreen instructions to complete the installation.



#### Note

Network connectivity might be temporarily lost while VirtualBox installs virtual network adapters. A system reboot can also be required after a successful installation.

Alternatively, if the `brew` command is available in your system, then you can install `VirtualBox` using the `brew install` command.

```
host:~ root# brew install virtualbox
```

## 2.2. Installing minikube on macOS

To install `minikube` on macOS, download from <https://github.com/kubernetes/minikube/releases> the appropriate file for your architecture: `minikube-darwin-amd64` or `minikube-darwin-arm64`.

Then open a terminal window, change to the directory where you downloaded the installer and use the `install` command with administrator privileges to run the installer. Make sure you install `minikube` in a folder in your system path, such as `/usr/local/bin/minikube`

```
host:~ root# cd Downloads
host:~ root# sudo install _FILE /usr/local/bin/minikube
```

This places the `minikube` executable in `/usr/local/bin/minikube`.

Apple notarizing features forbid running files downloaded from the internet unless authorized. If you try to run the `minikube` command then you will get a "minikube" cannot be opened because the developer cannot be verified message, and the application will be terminated.

To authorize the `minikube` application, use the `xattr` command with administrative privileges with the following options:

```
host:~ root# sudo xattr -r -d com.apple.quarantine /usr/local/bin/minikube
```

Verify that now you can execute the `minikube` command:

```
host:~ root# minikube version
minikube version: v1.20.0
commit: c61663e942ec43b20e8e70839dcca52e44cd85ae
```

Your output can differ, but must show the available version and the commit it is based on.

## 2.3. Starting Your minikube Cluster on macOS

To initialize your `minikube` cluster, use the `minikube start` command.

```
host:~ root# minikube start --driver=virtualbox
# minikube v1.20.0 on Darwin 11.3.1
# Using the virtualbox driver based on user configuration
# Starting control plane node minikube in cluster minikube
# Creating virtualbox VM (CPUs=4, Memory=16384MB, Disk=20000MB) ...
# Preparing Kubernetes v1.20.2 on Docker 20.10.6 ...
  • Generating certificates and keys ...
  • Booting up control plane ...
  • Configuring RBAC rules ...
# Verifying Kubernetes components...
```

```
▪ Using image gcr.io/k8s-minikube/storage-provisioner:v5
# Enabled addons: storage-provisioner, default-storageclass
# Done! kubectl is now configured to use "minikube" cluster and "default"
namespace by default
```



**Note**

To set the default driver, run the command `minikube config set driver DRIVER`.

### ► 3. Using minikube on Windows

In this course we support Windows 10 Pro and Enterprise versions.

#### 3.1. Installing a Hypervisor on Windows

There are several hypervisors available for Windows systems, including **Oracle VM VirtualBox** and **Microsoft Hyper-V**.



**Warning**

System driver conflicts might occur if more than one hypervisor is installed or enabled. Do not install or enable more than one hypervisor on your system.

#### Oracle VM VirtualBox installation

Oracle VM VirtualBox is a free, open source hypervisor. As the original driver for `minikube`, Oracle VM VirtualBox provides the most stability for Microsoft Windows 10 users.

1. Download the latest version of VirtualBox for Windows Hosts from <https://virtualbox.org/wiki/Downloads>
2. Open the downloaded VirtualBox executable and follow the onscreen instructions to complete the installation.



**Note**

Network connectivity might be temporarily lost while VirtualBox installs virtual network adapters. A system reboot can also be required after a successful installation.

#### Enabling Microsoft Hyper-V

Microsoft Hyper-V is a built-in hypervisor available on modern 64-bit versions of Microsoft Windows 10 Pro, Enterprise, and Education editions. Enabling Microsoft Hyper-V can be accomplished through PowerShell or by adjusting system Settings.

Refer to Microsoft Hyper-V documentation for system and hardware requirements.

- Via PowerShell
  - Launch a PowerShell console as Administrator.
  - In the console window, run the following command:

```
PS C:\> Enable-WindowsOptionalFeature -Online -FeatureName Microsoft-Hyper-V -All
```

- Restart your system when prompted to finish the installation process.
- Via Settings
  - In the search box on the taskbar, type **Programs and Features**, and select it from the search results.
  - Select **Turn Windows features on or off** from the list of options under Control Panel Home.
  - Select **Hyper-V** and click **OK** to begin the installation.
  - Restart your system when prompted to finish the installation process.

With Microsoft Hyper-V successfully enabled, create an external virtual switch to grant **minikube** access to your physical network.

1. Open a PowerShell console as Administrator.
2. Determine the name of the network adapter, such as Wi-Fi or Ethernet, to use by running **Get-NetAdapter**.
3. Create an external virtual switch named **minikube** that uses the selected network adapter and allows the management operating system to share the adapter:

```
PS C:\> New-VMSwitch -name minikube -NetAdapterName <AdapterName>
-AllowManagementOS $true
```

4. Restart your system to clear out potential routing problems.

### 3.2. Installing minikube on Windows

With a hypervisor installed, your system is now ready to begin the **minikube** installation process.

1. Download the stand-alone **minikube** Windows installer from <https://github.com/kubernetes/minikube/releases>
2. Open the downloaded **minikube-installer.exe** to begin the guided installation process.



#### Note

If you executed the **minikube-installer.exe** installer from a terminal window, close the terminal and open a new one before you start using **minikube**.

### 3.3. Starting Your minikube Cluster on Windows

To initialize your **minikube** cluster, use the **minikube start** command.

To use the Microsoft Hyper-V driver, provide the **--driver=hyperv** option to **minikube**:

```
PS C:\> minikube start --driver=hyperv
# minikube v1.20.0 on Microsoft Windows 10 Enterprise 10.0.18363 Build 18363
# Using the hyperv driver based on user configuration
# Starting control plane node minikube in cluster minikube
# Creating hyperv VM (CPUs=4, Memory=16384MB, Disk=20000MB) ...
# Preparing Kubernetes v1.20.2 on Docker 20.10.6 ...
  • Generating certificates and keys ...
  • Booting up control plane ...
  • Configuring RBAC rules ...
# Verifying Kubernetes components...
  • Using image gcr.io/k8s-minikube/storage-provisioner:v5
# Enabled addons: storage-provisioner, default-storageclass
# Done! kubectl is now configured to use "minikube" cluster and "default"
namespace by default
```

To use the Oracle VM VirtualBox driver, provide the `--driver=virtualbox` option to `minikube`:

```
PS C:\> minikube start --driver=virtualbox
# minikube v1.20.0 on Microsoft Windows 10 Enterprise 10.0.18363 Build 18363
# Using the virtualbox driver based on user configuration
# Starting control plane node minikube in cluster minikube
# Creating virtualbox VM (CPUs=4, Memory=16384MB, Disk=20000MB) ...
# Preparing Kubernetes v1.20.2 on Docker 20.10.6 ...
  • Generating certificates and keys ...
  • Booting up control plane ...
  • Configuring RBAC rules ...
# Verifying Kubernetes components...
  • Using image gcr.io/k8s-minikube/storage-provisioner:v5
# Enabled addons: storage-provisioner, default-storageclass
# Done! kubectl is now configured to use "minikube" cluster and "default"
namespace by default
```



### Note

To set the default driver, run the command `minikube config set driver DRIVER`.

## ► 4. Verifying your minikube installation

Use the `minikube status` command to validate that the `minikube` installation is running successfully:

```
[root@host ~]# minikube status
minikube
type: Control Plane
host: Running
kubelet: Running
apiserver: Running
kubeconfig: Configured
```

In case of errors, make sure you are using the appropriate driver during the installation, or refer to minikube Get Started documentation [<https://minikube.sigs.k8s.io/docs/start/>] for troubleshooting.

## ► 5. Adding extensions

`minikube` comes with the bare minimum set of features. To add more features, `minikube` provides an add-on based extension system. Developers can add more features by installing the needed add-ons.

Use the `minikube addons list` command for a comprehensive list of the add-ons available and the installation status.

- Installing the Ingress Add-on. For this course you must install the `ingress` add-on.

With your cluster up and ready, use the following command to enable the add-on:

```
[root@host ~]# minikube addons enable ingress
  • Using image k8s.gcr.io/ingress-nginx/controller:v0.44.0
  • Using image docker.io/jetttech/kube-webhook-certgen:v1.5.1
  • Using image docker.io/jetttech/kube-webhook-certgen:v1.5.1
# Verifying ingress addon...
# The 'ingress' addon is enabled
```

Versions and docker images can vary in your deployment, but make sure the final validation is successful.

- Installing the Dashboard add-on. The `dashboard` add-on is not required for this course but serves as a visual graphical interface if you are not comfortable with CLI commands.

Enable the `dashboard` add-on with the following command:

```
[root@host ~]# minikube addons enable dashboard
  • Using image kubernetesui/dashboard:v2.1.0
...output omitted...
# The 'dashboard' addon is enabled
```

Once the dashboard is enabled you can reach it by using the `minikube dashboard` command. This command will open the dashboard web application in your default browser.

Press `Ctrl+C` in the terminal to finish the connection to the dashboard.

## ► 6. Using a Developer Sandbox for Red Hat OpenShift as a Remote Kubernetes cluster

Developer Sandbox for Red Hat OpenShift is a free Kubernetes-as-a-Service platform offered by Red Hat Developers, based on Red Hat OpenShift.

Developer Sandbox allows users access to a pre-created Kubernetes cluster. Access is restricted to two namespaces (or projects if using OpenShift nomenclature). Developer Sandbox deletes pods after eight consecutive hours of running, and limits resources to 7 GB of RAM and 15 GB of persistent storage.

### 6.1. Create a Developer Sandbox account

Go to <https://developers.redhat.com/developer-sandbox> and click **Get started in the Sandbox**.



### What is the Sandbox?

The sandbox provides you with a private OpenShift environment in a shared, multi-tenant OpenShift cluster that is pre-configured with a set of developer tools. You can easily create containers from your source code or Dockerfile, build new applications using the samples and stacks provided, add services such as databases from our templates catalog, deploy Helm charts, and much more. Discover the rich capabilities of the full developer experience on OpenShift with the sandbox.

[Get started in the Sandbox](#)

[Learn about our IDE](#)

### What can I do in the Sandbox?

The sandbox is perfect for immediate access into OpenShift, but how can you start practicing your skills in this context if you don't have a starting point yet? Well, we've taken care of that for you with a curated series of activities that you

You need a free Red Hat account to use Developer Sandbox. Log in to your Red Hat account, or if you do not have one, then click **Create one now**. Fill in the form choosing a **Personal** account type, and then click **CREATE MY ACCOUNT**. You might need to accept Red Hat terms and conditions to use the Developer Program services.

When the account is ready you will be redirected back to the Developer Sandbox page. Click **Launch your Developer Sandbox for Red Hat OpenShift** to log in to Developer Sandbox.

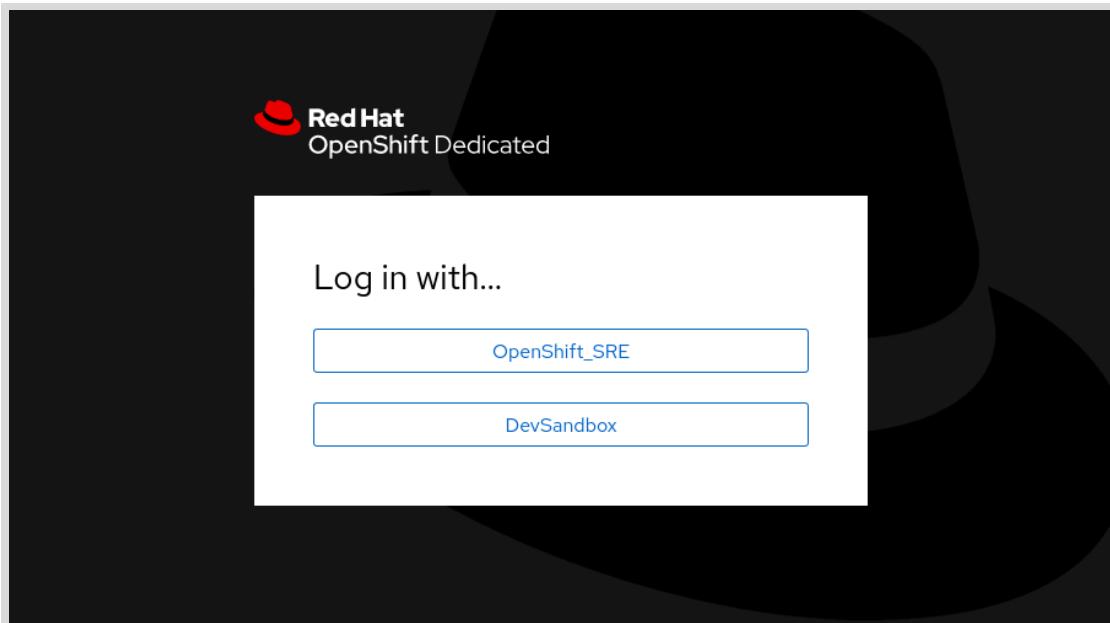
The screenshot shows a web-based developer sandbox interface. At the top, there's a navigation bar with four items: "Overview", "Get started in the Sandbox" (which is highlighted in blue), "Sandbox IDE", and "Sandbox activities". Below the navigation bar, the main content area has a title "Start your OpenShift experience in four simple steps". Underneath the title, there's a brief description: "If you're exploring how to run your code in containers, our Developer Sandbox makes it simple. Not only can you easily deploy your application from a Git repo, you can also set up a cloud IDE for your entire team. Follow these four steps to quickly get started:". Below this description is a red rectangular button with white text that says "Launch your Developer Sandbox for Red Hat OpenShift". To the right of the button is a numbered list of four steps: 1. Provision your free Red Hat OpenShift development cluster, 2. Deploy a [sample application](#) or bring your own repo, 3. Edit code from the integrated [Eclipse Che-based cloud IDE](#), and 4. Fully explore the Red Hat OpenShift developer experience.

If you just created your account, then you might need to wait some seconds for account approval. You might need to verify your account via 2-factor authentication.

Once the account is approved and verified, Click **Start using your sandbox**. You might need to accept Red Hat terms and conditions to use the Developer Sandbox.

The screenshot shows the same web-based developer sandbox interface as the previous one. The "Get started in the Sandbox" tab is still selected. The main content area has the same "Start your OpenShift experience in four simple steps" title and description. Below the description is a light blue callout box containing the text "You're ready to get started!". Inside the callout box, it says "To launch your sandbox, click the button below and select DevSandbox when prompted." Below this text is a red rectangular button with white text that says "Start using your sandbox".

In the OpenShift log in form, click **DevSandbox** to select the authentication method.



If requested, then log in using your Red Hat account credentials.

## 6.2. Start Using Developer Sandbox

The Kubernetes cluster and two namespaces are created for you. Only the *username-dev* and *username-stage* namespaces are available, and you can not create more.

A screenshot of the Red Hat OpenShift Dedicated dashboard. The left sidebar has a dark theme with white text and icons. It includes sections for "Developer" (selected), "+Add", "Topology" (highlighted with a blue bar), "Monitoring", "Search", "Builds", "Helm", and "Project". The main content area has a light background. At the top, it shows "Project: my-username-dev" and "Application: all applications". Below that, the "Topology" section displays the message "No resources found" and "To add content to your Project, create an Application, component or service using one of these options." It features two "Quick Starts" cards: "Get started with Spring" and "Monitor your sample application", and a "Samples" card with the sub-instruction "Binding your Quarkus application to Streams for Apache Kafka".

## ► 7. Enabling external access to Ingress.

Some exercises require you to identify the external IP and hostname associated to your Ingress so you can access your application from outside the cluster.

### 7.1. External access to Minikube.

Routing traffic from your local machine to your Minikube Kubernetes cluster requires two steps.

First you must find the local IP assigned to your Ingress add on. The `minikube ip` command is the easiest way to find the ingress IP:

```
[user@host ~]$ minikube ip  
192.168.99.103
```

Your IP will probably be different as it depends on your virtual environment configuration.

Second, you must declare a hostname for your ingress, and associate the hostname to the ingress IP. For this course, unless the hostname is already in use, you will declare `hello.example.com` as the hostname.

The association between an IP and a hostname is usually performed by a DNS server, but to avoid registering a domain name to use with Minikube, we can use the system's local name resolution service.

In **Linux and macOS** systems, edit the `/etc/hosts` file with elevated privileges. You might use the `sudo` command or the `su` command to obtain elevated privileges. In **Windows** systems, edit the `C:\Windows\System32\drivers\etc\hosts` file with administrative privileges.

Add the following line to the bottom of the file and replace `<IP-ADDRESS>` with the IP address listed in the previous step.

```
<IP-ADDRESS> hello.example.com
```



### Note

If for any reason you need to delete and recreate your Minikube cluster, then review the IP address assigned to the cluster and update the hosts file accordingly.

For accessing services in the cluster you will use the declared hostname and potentially any path associated to the ingress. So, if using the `hello.example.com` hostname and assuming the application is mapped to the path `/myapp`, then your application will be available in the URL `http://hello.example.com/myapp`.

#### 7.2. External access to Developer Sandbox.

External access to Developer Sandbox is already prepared. The ingress controller in Developer Sandbox is pre-configured to listen to requests sent to any sub-domain for a hostname known as the **wildcard domain**. The wildcard domain is a configuration value set by administrators when creating the cluster.

To enable external access to a service you must provide a sub-domain for the wildcard domain, so first you must know the wildcard domain.

There are several approaches for finding the wildcard domain associated to your Developer Sandbox instance:

#### **Infer the wildcard domain from the Console URL**

When you log into your Developer Sandbox account, you will be redirected to the OpenShift Console installed in the cluster. The URL for the OpenShift Console is of the form `https://console-openshift-console.apps._WILDCARD-DOMAIN_`.

To get the wildcard domain, remove from the API URL the `https://console-openshift-console`, and replace `api` by `apps`. For example, the wildcard domain for the Console URL `https://console-openshift-console.apps.sandbox.x8i5.p1.openshiftapps.com` is `apps.sandbox.x8i5.p1.openshiftapps.com`.

### Infer the wildcard domain from the API URL

To get the wildcard domain, remove from the API URL the `https://`, the `:6443`, and change `api` to `apps`. For example, the wildcard domain for the API URL `https://api.sandbox.x8i5.p1.openshiftapps.com:6443` is `apps.sandbox.x8i5.p1.openshiftapps.com`.

### Infer from a dry-run route creation

If you are using a Linux or macOS system, you can use the `kubectl` command to create a `route` resource that contains an example URL and extract the hostname from it:

```
[user@host ~]$ echo '{"apiVersion": "route.openshift.io/v1", \  
"kind": "Route", "metadata": {"name": "example"}, \  
"spec": {"to": {"kind": "Service", "name": "hello"}}}' | kubectl apply \  
--dry-run=server -o jsonpath='{.spec.host}' -f - \  
example-username-dev.apps.sandbox.x8i5.p1.openshiftapps.com
```

To get the wildcard domain, remove the first part of the hostname, that is everything before the first period. For example, the wildcard domain for the hostname `example-username-dev.apps.sandbox.x8i5.p1.openshiftapps.com` is `apps.sandbox.x8i5.p1.openshiftapps.com`.

Once you know the wildcard domain for your Developer Sandbox cluster, use it to generate a sub-domain to be used by your services. Remember that sub-domains must be unique for the shared Developer Sandbox cluster. One method for creating a unique sub-domain is to compose it in the format of `<DEPLOYMENT-NAME>-<NAMESPACE-NAME>. <WILDCARD-DOMAIN>`.

So, if using the `apps.sandbox.x8i5.p1.openshiftapps.com` wildcard domain and assuming a deployment named `hello` in a namespace named `username-dev` then you can compose your application hostname as `hello-username-dev.apps.sandbox.x8i5.p1.openshiftapps.com`.

Assuming the application is mapped to the path `/myapp`, then your application will be available in the URL `http://hello-username-dev.apps.sandbox.x8i5.p1.openshiftapps.com/myapp`.

## Finish

This concludes the guided exercise.

## **Appendix B**

# **Connecting to your Kubernetes Cluster**

## ► Guided Exercise

# Connecting to your Kubernetes Cluster

In this exercise you will install the kubectl command-line tool on your computer, and connect to the Kubernetes cluster that you will be using throughout the course.

## Outcomes

You should be able to:

- Install kubectl
- Connect to Minikube (if you are using Minikube)
- Connect to the OpenShift Developer Sandbox (if you are using the OpenShift Developer Sandbox)

## Before You Begin

Ensure you have either installed Minikube or created an OpenShift Developer Sandbox account.

## Instructions

The installation procedure of kubectl depends on your operating system.

### ► 1. Installing kubectl in Linux-based systems.

The Linux distributions supported in this course are Fedora Linux 34 and Red Hat Enterprise Linux 8.

You can install kubectl by downloading the binary and moving it to your PATH. At the same time, it is possible to use a package-manager.

#### 1.1. Using curl and the binary file

- Open a command-line terminal to download the kubectl binary.

```
[user@host ~]$ curl -LO https://dl.k8s.io/release/v1.21.0/bin/linux/amd64/kubectl
```

- Copy the binary to your PATH and make sure it has executable permissions.

```
[user@host ~]$ sudo install -m 0755 kubectl /usr/local/bin/kubectl
```

- Verify that kubectl has been installed successfully.

```
[user@host ~]$ kubectl version --client
Client Version: version.Info{Major:"1", Minor:"21", GitVersion:"v1.21.0",
GitCommit:"cb303e613a121a29364f75cc67d3d580833a7479", GitTreeState:"clean",
BuildDate:"2021-04-08T16:31:21Z", GoVersion:"go1.16.1", Compiler:"gc",
Platform:"linux/amd64"}
```

## 1.2. Using the dnf package manager.

- Open a command-line terminal. Install the package by using `dnf`.

```
[user@host ~]$ dnf install kubectl
...output omitted...
=====
Package           Architecture   Version      Repository      Size
=====
Installing:
  kubectl          x86_64        1.21.1-0    kubernetes     9.8 M

Transaction Summary
=====
Install 1 Package
...output omitted...
```

- Verify that `kubectl` has been installed successfully.

```
[user@host ~]$ kubectl version --client
Client Version: version.Info{Major:"1", Minor:"21", GitVersion:"v1.21.0",
GitCommit:"cb303e613a121a29364f75cc67d3d580833a7479", GitTreeState:"clean",
BuildDate:"2021-04-08T16:31:21Z", GoVersion:"go1.16.1", Compiler:"gc",
Platform:"linux/amd64"}
```

## ► 2. Installing kubectl in macOS.

### 2.1. Using curl and the binary file

- Open a command-line terminal. Download the `kubectl` binary by using `curl`. Replace the `os-architecture` parameter depending on your Mac processor. If your Mac comes with an Intel processor, use `amd64`. If your Mac comes with an Apple Silicon processor, use `arm64`.

```
[user@host ~]$ curl -LO "https://dl.k8s.io/release/v1.21.0/bin/darwin/os-
architecture/kubectl"
```

- Give the binary file executable permissions. Move the binary file executable to your PATH.

```
[user@host ~]$ chmod +x ./kubectl
[user@host ~]$ sudo mv ./kubectl /usr/local/bin/kubectl
```

- Verify that `kubectl` has been installed successfully.

```
[user@host ~]$ kubectl version --client
Client Version: version.Info{Major:"1", Minor:"21", GitVersion:"v1.21.0",
GitCommit:"cb303e613a121a29364f75cc67d3d580833a7479", GitTreeState:"clean",
BuildDate:"2021-04-08T16:31:21Z", GoVersion:"go1.16.1", Compiler:"gc",
Platform:"linux/amd64"}
```

## 2.2. Using the homebrew package manager



### Note

If you have previously installed Minikube with homebrew, kubectl should already be installed in your computer. You can skip the installation step and directly verify that kubectl has been installed correctly.

- Install kubectl by using brew

```
[user@host ~]$ brew install kubectl  
...output omitted...
```

- Verify that kubectl has been installed successfully.

```
[user@host ~]$ kubectl version --client  
Client Version: version.Info{Major:"1", Minor:"21", GitVersion:"v1.21.0",  
GitCommit:"cb303e613a121a29364f75cc67d3d580833a7479", GitTreeState:"clean",  
BuildDate:"2021-04-08T16:31:21Z", GoVersion:"go1.16.1", Compiler:"gc",  
Platform:"linux/amd64"}
```

## ► 3. Installing kubectl in Windows

### 3.1. Using the binary

- Create a new folder, such as C:\kube, to use as the destination directory of the kubectl binary download.
- Download the latest release of the kubectl binary from <https://dl.k8s.io/release/v1.21.0/bin/windows/amd64/kubectl.exe> and save it to the previously created folder.
- Add the binary to your PATH.
  - In the search box on the taskbar, type env, and select **Edit the system environment variables** from the search results.
  - Click **Environment Variables** on the **System Properties** screen.
  - Under the **System variables** section, select the row containing Path and click **Edit**. This will open the **Edit environment variable** screen.
  - Click **New** and type the full path of the folder containing the kubectl.exe (for example, C:\kube).
  - Click **OK** to save the change and close the editor.
  - Click **OK → OK** to close out of the remaining screens.
- Verify that kubectl has been installed successfully.

```
[user@host ~]$ kubectl version --client
Client Version: version.Info{Major:"1", Minor:"21", GitVersion:"v1.21.0",
GitCommit:"cb303e613a121a29364f75cc67d3d580833a7479", GitTreeState:"clean",
BuildDate:"2021-04-08T16:31:21Z", GoVersion:"go1.16.1", Compiler:"gc",
Platform:"linux/amd64"}
```

#### ► 4. Download the D0100x-apps repository

The D0100x-apps GitHub repository contains files used throughout the course. The best way to work with it is by using Git. However, if you are not used to Git, you can simply download it as a regular folder and place it anywhere in your computer.

##### 4.1. **Using Git**

If you have Git installed in your system, you can simply `clone` the repository.

```
[user@host ~]$ git clone https://github.com/RedHatTraining/D0100x-apps.git
...output omitted...
```

##### 4.2. **Downloading it from the GitHub web page**

- Open a web browser and navigate to <https://github.com/RedHatTraining/D0100x-apps>.
- Click **Code** and then click **Download ZIP**. A ZIP file with the repository content is downloaded.

#### ► 5. Connecting kubectl to the Kubernetes cluster

The D0100x-apps contains a script that handles all configurations for you under the `setup` directory. The script you should run depends on your operating system (Linux, macOS or Windows) and the Kubernetes distribution you use (Minikube or the OpenShift Developer Sandbox).

If you run the Minikube script, it will configure Minikube to work as a Kubernetes cluster with restricted access. You will only have access to two namespaces. This way, we simulate a real Kubernetes cluster, where usually developers do not have full access.



##### Note

If you want to recover full access over your cluster, then you can change the `kubectl` context to the default Minikube context, `minikube`. Use the command `kubectl config use-context minikube`.

If you run the OpenShift Developer Sandbox script, it will configure `kubectl` to run commands against the Openshift Developer Sandbox cluster. The script will ask you to provide some information such as cluster url, username or token.

##### 5.1. **Using Minikube**

###### • **Linux and macOS**

In your command-line terminal, move to the D0100x-apps directory and run the script located at `./setup/operating-system/setup.sh`. Replace `operating-`

system for linux if you are using Linux. Use macos if you are using macOS. Make sure the script has executable permissions.

```
[user@host D0100x-apps]$ chmod +x ./setup/operating-system/setup.sh
[user@host D0100x-apps]$ ./setup/operating-system/setup.sh
Creating namespace 'user-dev' ...
Creating namespace 'user-stage' ...
Creating certificates ...
Creating Kubectl credentials for 'user' ...
Creating Kubectl context 'user-context' for user 'user' ...
Creating role resources for user 'user' in namespace 'user-dev' ...
Creating role resources for user 'user' in namespace 'user-stage' ...
Switched to context "user-context".
Switching to namespace 'user-dev' ...
OK!
```

- **Windows**

In your PowerShell terminal, move to the D0100x-apps directory and execute the following command. This command allows you to run unsigned PowerShell scripts in your current terminal session.

```
[user@host D0100x-apps]$ Set-ExecutionPolicy -Scope Process -ExecutionPolicy
Bypass
```

Then, run the script located at .\setup\windows\setup.ps1.

```
[user@host D0100x-apps]$ .\setup\windows\setup.ps1
Creating namespace 'user-dev' ...
Creating namespace 'user-stage' ...
Creating certificates ...
Creating Kubectl credentials for 'user' ...
Creating Kubectl context 'user-context' for user 'user' ...
Creating role resources for user 'user' in namespace 'user-dev' ...
Creating role resources for user 'user' in namespace 'user-stage' ...
Switched to context "user-context".
Switching to namespace 'user-dev' ...
OK!
```

## 5.2. Using the OpenShift Developer Sandbox

The setup-sandbox script will ask you to provide a server, a token and your username. To find the server and token follow these steps:

- Open a web browser and navigate to the OpenShift Developer Sandbox website. Log in with your username and password.
- Click on your username in the upper right pane of the screen. A dropdown menu opens.
- In the dropdown menu, click **Copy login command**. A new tab opens, log in again with your account if necessary by clicking **DevSanbox**.

- Click **Display Token**.
- The token you must provide in the script shows in your web browser.
- The server you must provide is a parameter of the `oc login` command displayed. For example, in the command `oc login --token=sha256~Gs54-tjq1Bo-fo866bbddv8wbQ0bpmy321eS1qj1g --server=https://api.sandbox.x8i5.p1.openshiftapps.com:6443`, the server is `https://api.sandbox.x8i5.p1.openshiftapps.com:6443`.

- Keep these values. You will be asked for them in the script.

Run the appropriate script. The following instructions will depend on your operating system.

- **Linux and macOS**

In your command-line terminal, move to the `D0100x-apps` directory and run the script located at `./setup/operating-system/setup-sandbox.sh`. Replace

operating-system for linux if you are using Linux. Use macos if you are using macOS. Make sure the script has executable permissions.

```
[user@host D0100x-apps]$ chmod +x ./setup/operating-system/setup-sandbox.sh
[user@host D0100x-apps]$ ./setup/operating-system/setup-sandbox.sh
What is the OpenShift cluster URL?
https://api.sandbox.x8i5.p1.openshiftapps.com:6443
What is the OpenShift token?
sha256-wVSG...DDn0
What is your OpenShift username?
user
Creating Kubectl context...
Context created successfully
```

- **Windows**

In your PowerShell terminal, move to the D0100x-apps directory and execute the following command. This command allows you to run unsigned PowerShell scripts in your current terminal session.

```
[user@host D0100x-apps]$ Set-ExecutionPolicy -Scope Process -ExecutionPolicy
Bypass
```

Then, run the script located at .\setup\windows\setup-sandbox.ps1.

```
[user@host D0100x-apps]$ .\setup\windows\setup-sandbox.ps1
What is the OpenShift cluster URL?
https://api.sandbox.x8i5.p1.openshiftapps.com:6443
What is the OpenShift token?
sha256-wVSG...DDn0
What is your OpenShift username?
user
Creating Kubectl context...
Context created successfully
```

## Finish

This concludes the guided exercise.



# Red Hat

## Training and Certification

### Student Workbook

Red Hat OpenShift Container Platform 4.6 DO101X  
**Advanced Application Management with  
Red Hat OpenShift**

Edition 2







## Join a community dedicated to learning open source

The Red Hat® Learning Community is a collaborative platform for users to accelerate open source skill adoption while working with Red Hat products and experts.



**Network** with tens of thousands of community members



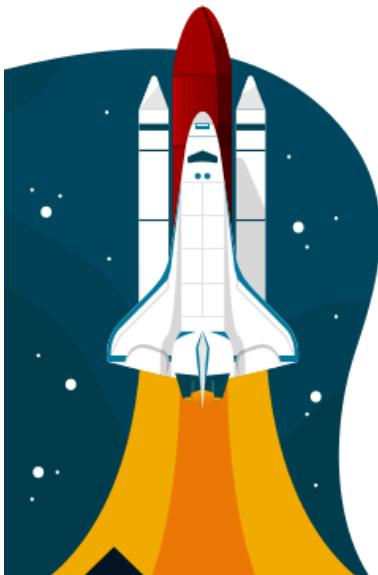
**Engage** in thousands of active conversations and posts



**Join and interact** with hundreds of certified training instructors



**Unlock** badges as you participate and accomplish new goals



This knowledge-sharing platform creates a space where learners can connect, ask questions, and collaborate with other open source practitioners.

**Access** free Red Hat training videos

**Discover** the latest Red Hat Training and Certification news

**Connect** with your instructor - and your classmates - before, after, and during your training course.

**Join** peers as you explore Red Hat products

Join the conversation [learn.redhat.com](https://learn.redhat.com)



Copyright © 2020 Red Hat, Inc. Red Hat, Red Hat Enterprise Linux, the Red Hat logo, and Ansible are trademarks or registered trademarks of Red Hat, Inc. or its subsidiaries in the United States and other countries.



# **Advanced Application Management with Red Hat OpenShift**



**Red Hat OpenShift Container Platform 4.6 DO101X**  
**Advanced Application Management with Red Hat OpenShift**  
**Edition 2 7a6f4c2**  
**Publication date 20210000**

Authors: Enol Álvarez de Prado, Guy Bianco IV, Marek Czernek,  
Zach Guterman, Dan Kolepp, Herve Quatremain,  
Eduardo Ramírez Ronco, Randy Thomas  
Course Architect: Ravishankar Srinivasan  
Editors: Sam Ffrench, Nicole Muller

Copyright © 2021 Red Hat, Inc.

The contents of this course and all its modules and related materials, including handouts to audience members, are  
Copyright © 2021 Red Hat, Inc.

No part of this publication may be stored in a retrieval system, transmitted or reproduced in any way, including, but not limited to, photocopy, photograph, magnetic, electronic or other record, without the prior written permission of Red Hat, Inc.

This instructional program, including all material provided herein, is supplied without any guarantees from Red Hat, Inc. Red Hat, Inc. assumes no liability for damages or legal action arising from the use or misuse of contents or details contained herein.

If you believe Red Hat training materials are being used, copied, or otherwise improperly distributed, please send email to [training@redhat.com](mailto:training@redhat.com) or phone toll-free (USA) +1 (866) 626-2994 or +1 (919) 754-3700.

Red Hat, Red Hat Enterprise Linux, the Red Hat logo, JBoss, OpenShift, Fedora, Hibernate, Ansible, CloudForms, RHCA, RHCE, RHCSA, Ceph, and Gluster are trademarks or registered trademarks of Red Hat, Inc. or its subsidiaries in the United States and other countries.

Linux® is the registered trademark of Linus Torvalds in the United States and other countries.

Java® is a registered trademark of Oracle American, Inc. and/or its affiliates.

XFS® is a registered trademark of Hewlett Packard Enterprise Development LP or its subsidiaries in the United States and/or other countries.

MySQL® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js® is a trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack word mark and the Square O Design, together or apart, are trademarks or registered trademarks of OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. Red Hat, Inc. is not affiliated with, endorsed by, or sponsored by the OpenStack Foundation or the OpenStack community.

All other trademarks are the property of their respective owners.

Contributors: Richard Allred, Joel Birchler, Nicolette Lucas, David Sacco, Heider Souza, Sajith Sugathan, Chris Tusa

<b>Document Conventions</b>	<b>ix</b>
	ix
<b>Introduction</b>	<b>xi</b>
Advanced Application Management with Red Hat OpenShift .....	xi
Orientation to the Classroom Environment .....	xii
<b>1. Configuring a Cloud Application Developer Environment</b>	<b>1</b>
Developing Applications with VSCode .....	2
Guided Exercise: Developing Applications with VSCode .....	9
Initializing a Git Repository .....	13
Guided Exercise: Initializing a Git Repository .....	26
Managing Application Source Code with Git .....	39
Guided Exercise: Managing Application Source Code with Git .....	51
Summary .....	59
<b>2. Deploying Applications to Red Hat OpenShift Container Platform</b>	<b>61</b>
Deploying an Application to Red Hat OpenShift Container Platform .....	62
Guided Exercise: Deploying an Application to Red Hat OpenShift Container Platform .....	72
Summary .....	81
<b>3. Configuring Application Builds in OpenShift</b>	<b>83</b>
Updating an Application .....	84
Guided Exercise: Updating an Application .....	88
Configuring Application Secrets .....	100
Guided Exercise: Configuring Application Secrets .....	104
Connecting an Application to a Database .....	121
Guided Exercise: Connecting an Application to a Database .....	126
Summary .....	143
<b>4. Scaling Applications in OpenShift</b>	<b>145</b>
Scaling an Application .....	146
Guided Exercise: Scaling an Application .....	150
Summary .....	161
<b>5. Troubleshooting Applications in OpenShift</b>	<b>163</b>
Troubleshooting and Fixing an Application .....	164
Guided Exercise: Troubleshooting and Fixing an Application .....	171
Summary .....	182
<b>6. Deploying Containerized Applications on OpenShift</b>	<b>183</b>
Describing Kubernetes and OpenShift Architecture .....	184
Creating Kubernetes Resources .....	191
Guided Exercise: Deploying a Database Server on OpenShift .....	203
Creating Routes .....	207
Guided Exercise: Exposing a Service as a Route .....	211
Creating Applications with Source-to-Image .....	216
Guided Exercise: Creating a Containerized Application with Source-to-Image .....	226
Summary .....	233



# Document Conventions

---

This section describes various conventions and practices used throughout all Red Hat Training courses.

## Admonitions

Red Hat Training courses use the following admonitions:



### References

These describe where to find external documentation relevant to a subject.



### Note

These are tips, shortcuts, or alternative approaches to the task at hand. Ignoring a note should have no negative consequences, but you might miss out on something that makes your life easier.



### Important

These provide details of information that is easily missed: configuration changes that only apply to the current session, or services that need restarting before an update will apply. Ignoring these admonitions will not cause data loss, but may cause irritation and frustration.



### Warning

These should not be ignored. Ignoring these admonitions will most likely cause data loss.

## Inclusive Language

Red Hat Training is currently reviewing its use of language in various areas to help remove any potentially offensive terms. This is an ongoing process and requires alignment with the products and services covered in Red Hat Training courses. Red Hat appreciates your patience during this process.



# Introduction

## Advanced Application Management with Red Hat OpenShift

Red Hat OpenShift Container Platform is a containerized application platform that allows enterprises to accelerate and streamline application development, delivery, and deployment on-premise or in the cloud. As OpenShift and Kubernetes continue to be widely adopted by enterprises, developers are increasingly required to understand how to develop, build, and deploy applications with a containerized application platform. While some developers are interested in managing the underlying OpenShift infrastructure, most developers want to focus their time and energy on developing applications, and leveraging OpenShift for its simple building, deployment, and scaling capabilities.

Advanced Application Management with Red Hat OpenShift (DO101x) introduces developers to Red Hat OpenShift Container Platform.

# Orientation to the Classroom Environment

---

DO101x is a *Bring Your Developer Workstation (BYDW)* class, where you use your own internet-enabled system to access the shared OpenShift cluster. The following operating systems are supported:

- Red Hat Enterprise Linux 8 or Fedora Workstation 32 or later
- Ubuntu 20.04 LTS or later
- Microsoft Windows 10
- macOS 10.15 or later

## BYDW System Requirements

<b>Attribute</b>	<b>Minimum Requirements</b>	<b>Recommended</b>
<b>CPU</b>	1.6 GHz or faster processor	Multi-core i7 or equivalent
<b>Memory</b>	8 GB	16 GB or more
<b>Disk</b>	10 GB free space HD	10 GB or more free space SSD
<b>Display Resolution</b>	1024x768	1920x1080 or greater

You must have permissions to install additional software on your system. Some hands-on learning activities in DO101x provide instructions to install the following programs:

- Node.js
- Git 2.18 or later (Git Bash for Windows systems)
- The OpenShift CLI (oc) 4.6.0 or later

You might already have these tools installed. If you do not, then wait until the day you start this course to ensure a consistent course experience.

## BYDW Systems Support Considerations

Depending on your system, you might see differences between your command-line shell and the examples given in this course.

### Red Hat Enterprise Linux or Fedora Workstation

- If you use Bash as the default shell, then your prompt might match the [user@host ~]\$ prompt used in the course examples, although different Bash configurations can produce different results.
- If you use another shell, such as zsh, then your prompt format will differ from the prompt used in the course examples.

- When performing the exercises, interpret the [user@host ~]\$ prompt used in the course as a representation of your system prompt.

### **Ubuntu**

- You might find differences in the prompt format.
- In Ubuntu, your prompt might be similar to user@host:~\$.
- When performing the exercises, interpret the [user@host ~]\$ prompt used in the course as a representation of your Ubuntu prompt.

### **macOS**

- You might find differences in the prompt format.
- In macOS, your prompt might be similar to host:~ user\$.
- When performing the exercises, interpret the [user@host ~]\$ prompt used in the course as a representation of your macOS prompt.
- You might need to grant execution permissions to the installed runtimes.

### **Microsoft Windows**

- Windows does not support Bash natively. Instead, you must use PowerShell.
- In Windows PowerShell, your prompt should be similar to PS C:\Users\user>.
- When performing the exercises, interpret the [user@host ~]\$ Bash prompt as a representation of your Windows PowerShell prompt.
- For some commands, Bash syntax and PowerShell syntax are similar, such as cd or ls. You can also use the slash character (/) in file system paths.
- This course only provides support for Windows PowerShell.
- The Windows firewall might ask for additional permissions in certain exercises.

## **Executing Long Commands**

This course breaks long commands into multiple lines by using the backslash character (\), for example:

```
[user@host ~]$ long command \
    argument_1 \
    argument_2
```

The preceding example works on the Linux and macOS systems.

On Windows, use the backtick character (`), for example:

```
[user@host ~]$ long command ` \
    argument_1 ` \
    argument_2
```

Alternatively, you can type commands in one line on all systems, such as:

```
[user@host ~]$ long command argument_1 argument_2
```

## Additional Accounts and Services

### Additional Accounts and Services

This course further requires the following accounts:

- A personal GitHub account.
- A Red Hat account.

This course does not require paid versions of any of the preceding accounts.

## Chapter 1

# Configuring a Cloud Application Developer Environment

### Goal

Configure a developer environment with a modern integrated developer environment and version control.

### Objectives

- Edit application source code with VSCode.
- Create a Git repository.
- Use version control to collaborate and manage application source code.

### Sections

- Developing Applications with VSCode (and Guided Exercise)
- Initializing a Git Repository (and Guided Exercise)
- Managing Application Source Code with Git (and Guided Exercise)

# Developing Applications with VSCode

---

## Objectives

After completing this section, you should be able to edit application source code with VSCode.

## Integrated Development Environments

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

- 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 the following:

- 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:

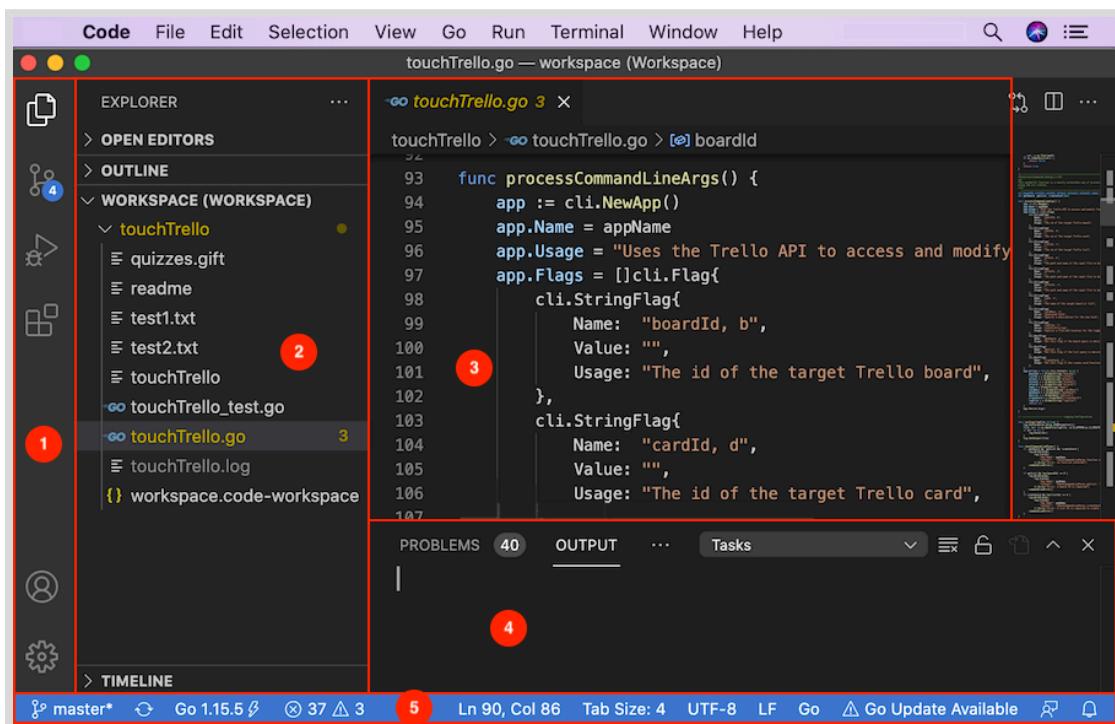


Figure 1.1: The VS Code user interface.

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 shortcuts for the Explorer, Search, Source Control, Debug, and Extensions views. You can also access these activity views from the View 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 files side by side.
4. **Panels**. Located below the editors, individual panels provide different output or debugging information for activities in VS Code. By default, four panels are provided:
  - 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

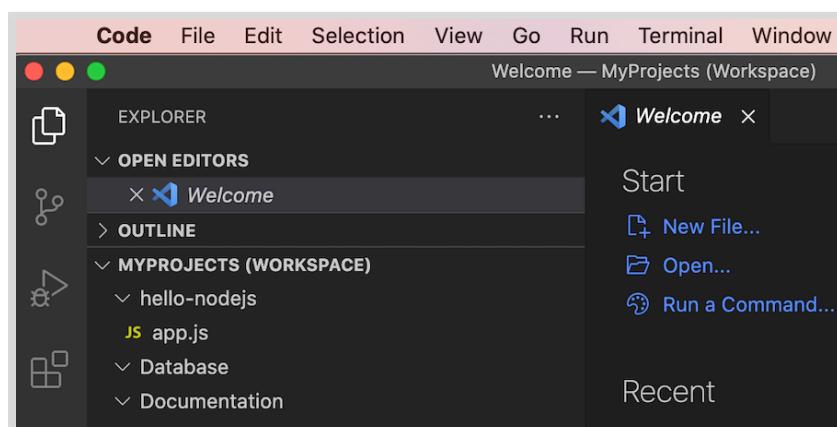
VS Code organizes a collection of related software projects and configuration settings in a **workspace**. Configuration data for each workspace is stored in a file with a `.code-workspace` extension.

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 1.2: 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**.

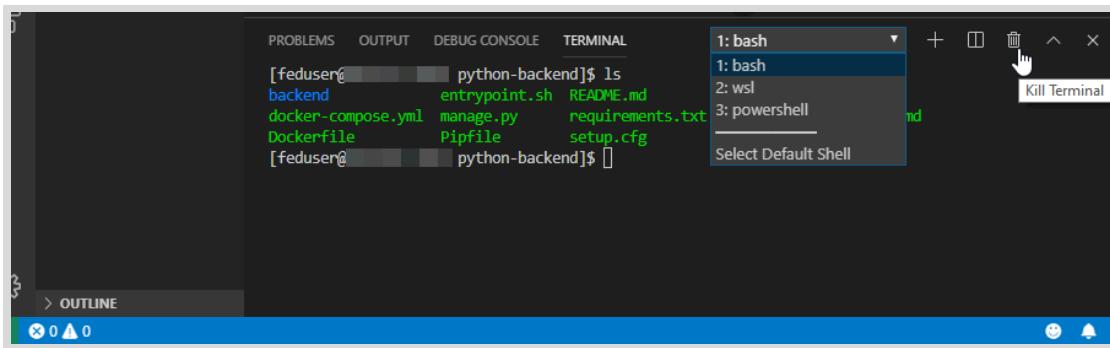


Figure 1.3: The VS Code integrated terminal

## 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. Any user or organization can contribute extensions. After an extension is developed, it is advertised and published on the VS Code marketplace.

You can search, download, and install extensions from the marketplace within 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.

In Node.js, **modules** are the smallest unit of reusable code. Node.js provides several built-in modules.

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

Use the built in `require` function to load a module. Consider the following example:

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

The `http` variable is a reference to the built-in `http` module, while the `mymodule` variable is 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 a collection of one or more Node.js modules. Packages can be downloaded from a Node.js package repository.

Application dependencies are defined in the `package.json` file, located at the root of the Node.js project folder. An example `package.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 `package.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 is included 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. This creates a `package.json` file for a new Node.js application.

You can define additional scripts within the `scripts` section of `package.json`. For example, it is customary to define a script named `start` that starts the application. To run a script, pass

the name of the script to the `npm run` sub-command. For example, if you define a script named `build`, run that script with the following:

```
[user@host myapp]$ npm run build
```



**Note**

Some script names do not require the `run` sub-command. For example, `npm start` is the same as `npm run start`.

## 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 and save it as a dependency:

```
[user@host ~]$ npm install -S express
```

After installing the `express` package, Express is available to your application.

For example, the `express` package is a dependency for an application named `myapp`. The application has the following package.json:

```
{
  "name": "myapp",
  "version": "0.0.1",
  "private": true,
  "scripts": {
    "start": "node app.js"
  },
  "dependencies": {
    "express": "~4.16.1"
  }
}
```

Start the `myapp` application with the `npm start` command, which runs `node app.js`. The `app.js` file contains the primary application logic and contains the following:

```
const express = require('express');
const 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 application sends a response of `Hello, World!`.



## References

### Integrated Development Environment - Wikipedia

[https://en.wikipedia.org/wiki/Integrated\\_development\\_environment](https://en.wikipedia.org/wiki/Integrated_development_environment)

For more information about Visual Studio Code workspaces and project folders, refer to the Visual Studio Code documentation at  
<https://code.visualstudio.com/docs/editor/multi-root-workspaces>

For more information about Visual Studio Code integrated terminal, refer to the Visual Studio Code documentation at  
<https://code.visualstudio.com/docs/editor/integrated-terminal>

For more information about Node.js modules, refer to the Node.js documentation at  
[https://nodejs.org/api/modules.html#modules\\_modules](https://nodejs.org/api/modules.html#modules_modules)

## ► Guided Exercise

# Developing Applications with VSCode

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.

## Before You Begin

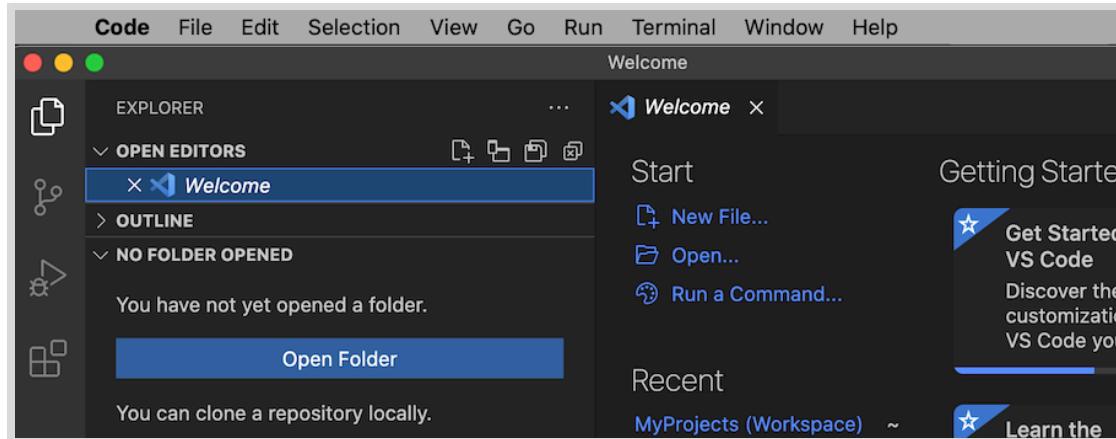
To perform this exercise, ensure that:

- You have access to a Linux (Debian or Fedora-based), macOS, or Windows system, including the required permissions to install software on that system.

## Instructions

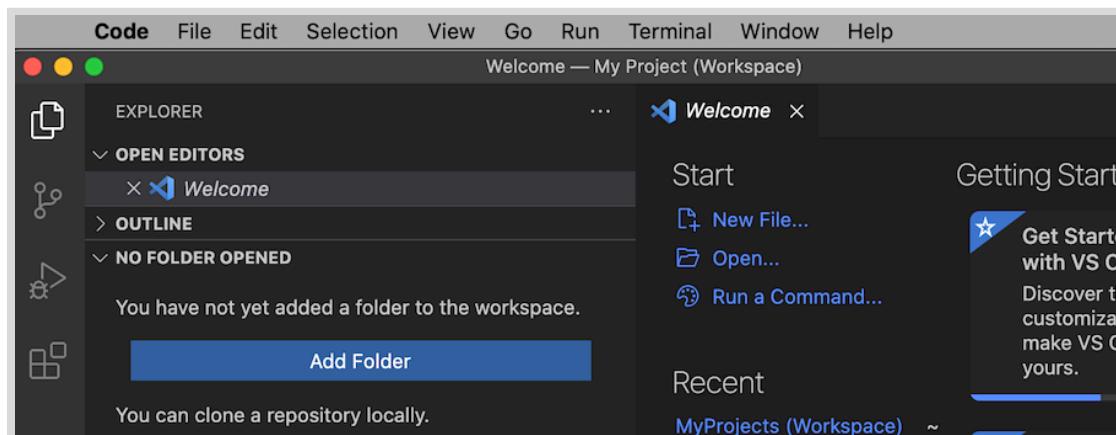
- 1. Download and install the long term support (LTS) version of Node.js.
  - On Windows, download and install the latest LTS version [<https://nodejs.org/dist/v16.13.0/node-v16.13.0-x64.msi>].
  - On macOS, download and install the latest LTS version [<https://nodejs.org/dist/v16.13.0/node-v16.13.0.pkg>].
  - On Linux, use the documentation [<https://nodejs.org/en/download/package-manager/>] to download and install the latest package that contains Node.js.
- 2. Download and install VS Code.

Use the appropriate installation for your device from the VS Code Download site [<https://code.visualstudio.com/download>].
- 3. Open VS Code and create a workspace to host your projects.
  - 3.1. Open the VS Code application according to your operating system. Click **View > Explorer** to display the Explorer view.
  - 3.2. 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 1.4: The VS Code application**

- 3.3. 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 1.5: The VS Code workspace.**

- 4. Create a **hello-nodejs** project folder, and then add it to your workspace.

- 4.1. 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.

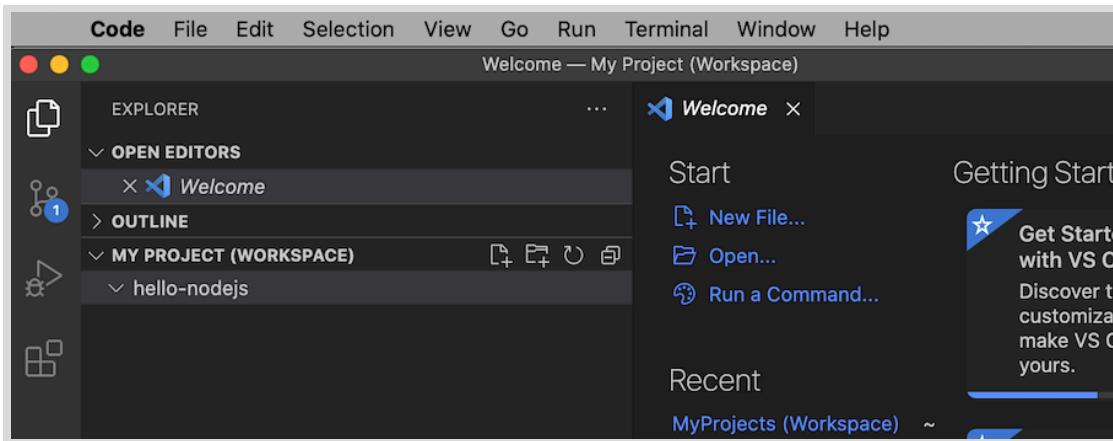


Figure 1.6: Add a folder to the VS Code workspace.

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

```
console.log("Hello World!\n");
```

- 5.1. Right-click `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.
- 5.2. Add the text `console.log("Hello World!\n");` to the `app.js` editor tab, and then save the file (`File > Save`).
- ▶ 6. Right-click `hello-nodejs` in the workspace, and then select `Open in Integrated Terminal` to access the `hello-nodejs` project from the VS Code integrated terminal.

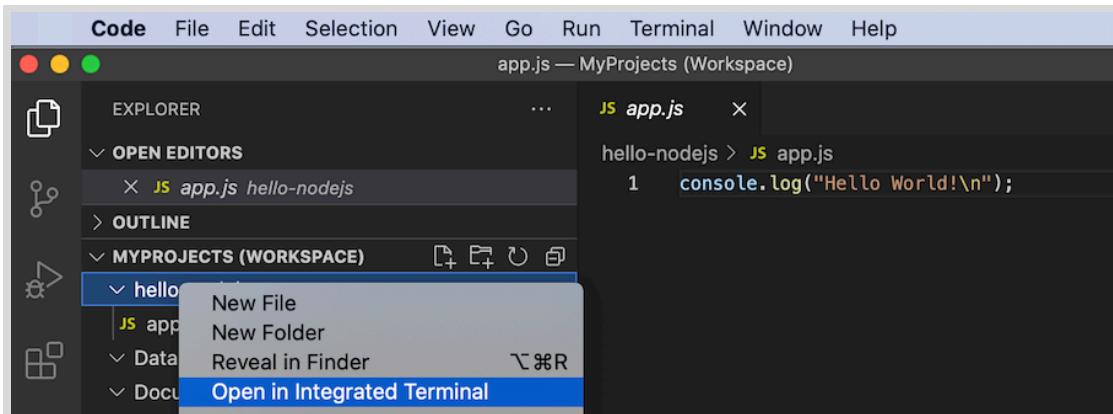
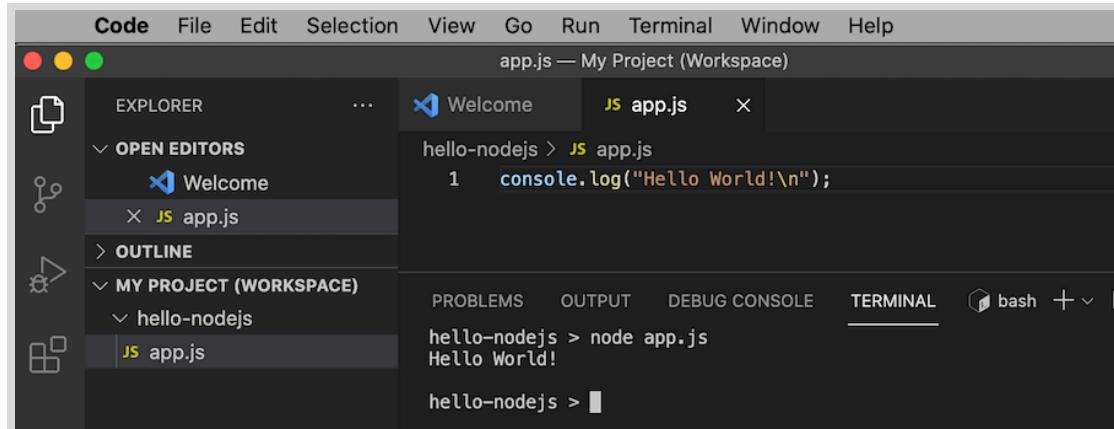


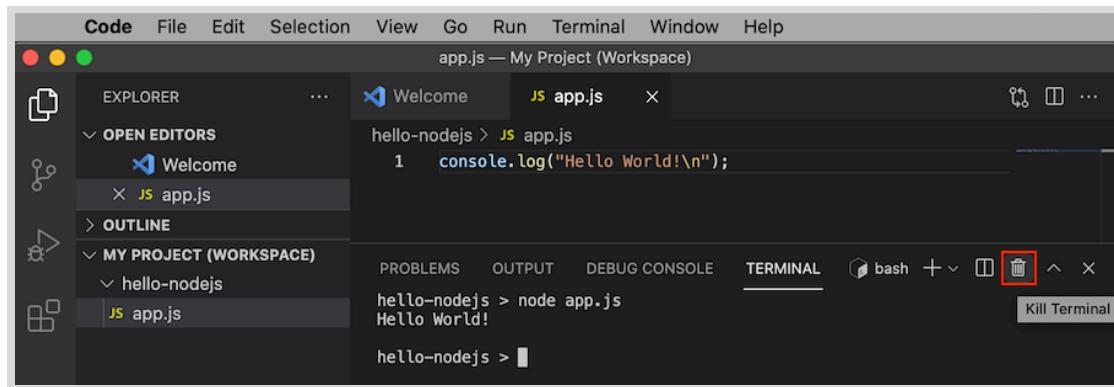
Figure 1.7: Open a project folder in the VS Code integrated terminal.

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



**Figure 1.8: Execute a Node.js application in the VS Code integrated terminal.**

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



**Figure 1.9: Closing the integrated terminal window in VS Code.**

## Finish

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 one of the most popular version control systems. 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 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 files, 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 changes from the remote repository, and when local changes are uploaded to the remote repository.

To track file changes in Git, you create a series of project snapshots. 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** - The creator or creators of the commit.
- **Commit hash** - A unique identifier for the commit. A commit hash consists of 40 hexadecimal characters. If a Git command requires a commit hash to perform an operation, then you can abbreviate the commit to enough characters to be unique within the repository.

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.

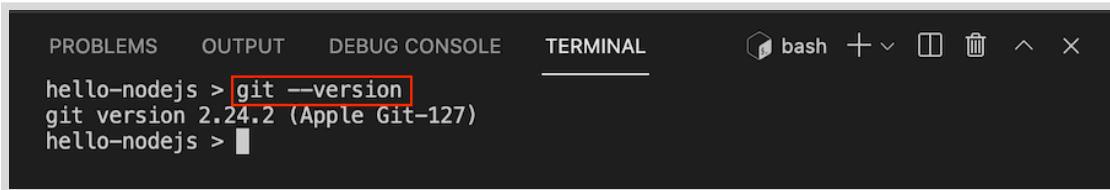
When other contributors push commits to a remote repository, those commits are not present in your local repository. To download new commits from other contributors, you **pull** changes from the remote Git repository.

## 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:



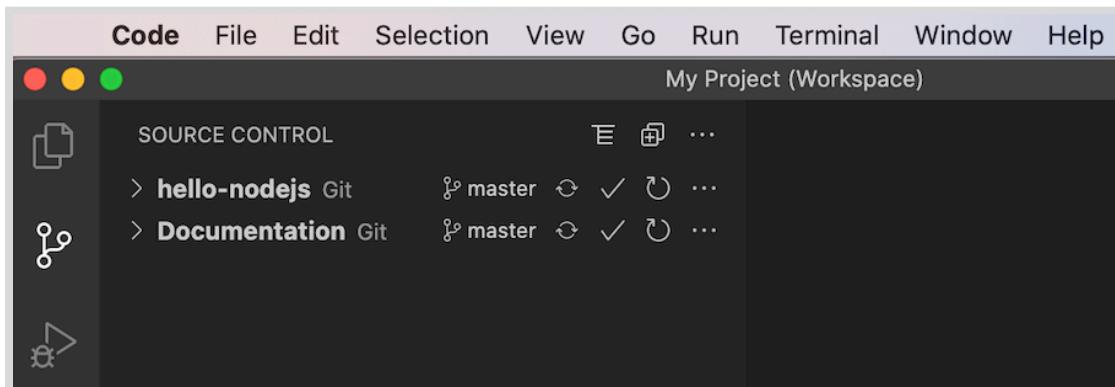
```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
bash + □ └ ┌ ×
hello-nodejs > git --version
git version 2.24.2 (Apple Git-127)
hello-nodejs >
```

**Figure 1.10: Testing the installation of Git in VS Code**

## The Source Control View in VS Code

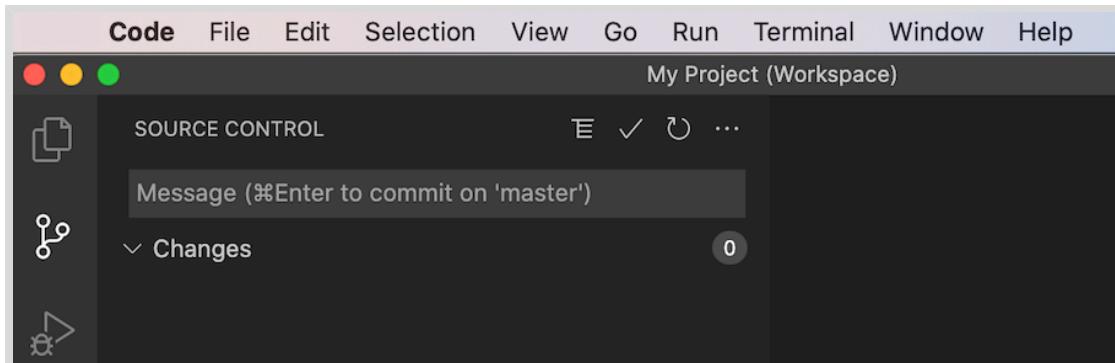
Use the VS Code Command Palette ([View > Command Palette...](#)) and the Source Control view ([View > SCM](#)) to execute Git operations, such as cloning a repository or committing code changes.

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 list:



**Figure 1.11: The source control list in the source control view of VS Code.**

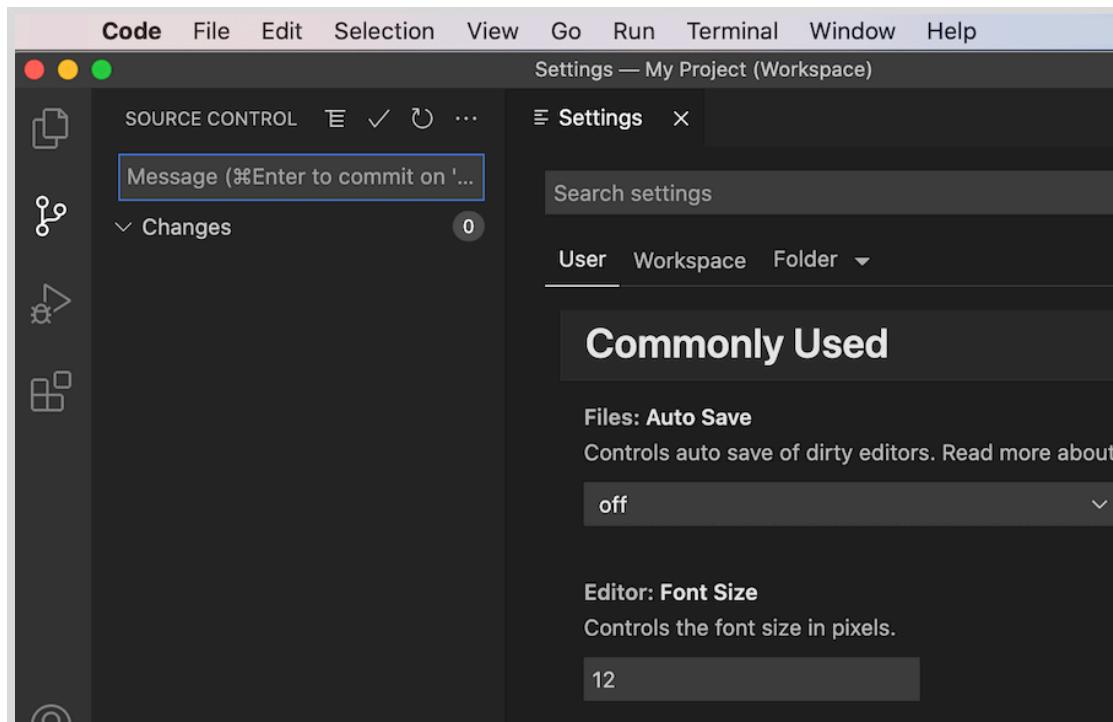
By default, when there is only a single Git repository in your workspace, then the Source Control view does not display the SOURCE CONTROL list.



**Figure 1.12: Source control view for a single Git repository.**

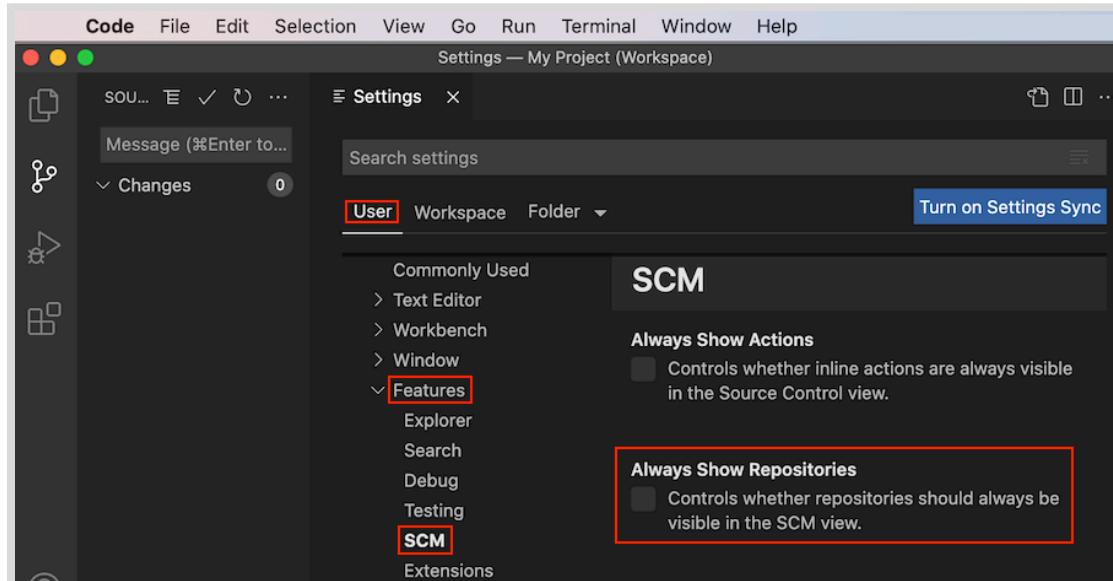
For a consistent user interface, independent of the number of Git repositories in your workspace, you must enable the **Always Show Repositories** source control management option.

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 1.13:** 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 Repositories**.



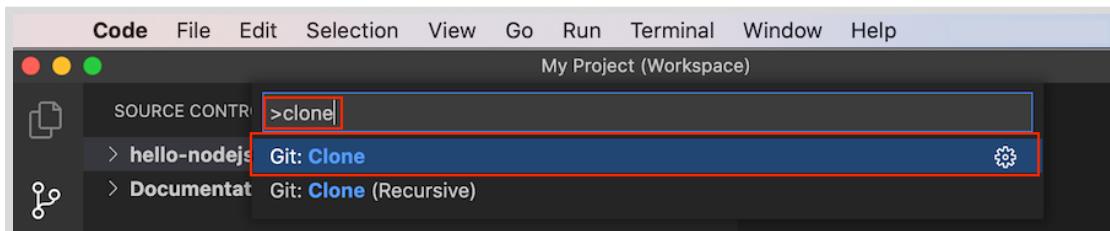
**Figure 1.14:** Option to always show the source control repositories list in the source control view of VS Code.

When you enable this option, then VS Code displays the **SOURCE CONTROL** 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 1.15:** 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

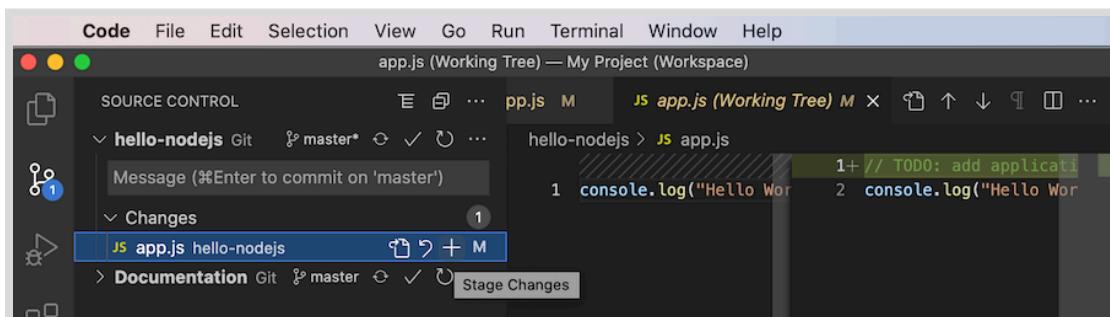
After adding a cloned repository to your VS Code workspace, you can edit project files like any other workspace files. As you edit project files, Git assigns a status to each file:

- **Modified** - The file differs 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 1.16:** Review file changes before staging a file.

If the file changes are accurate and complete, click the plus button next to the file to stage changes, which adds 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:

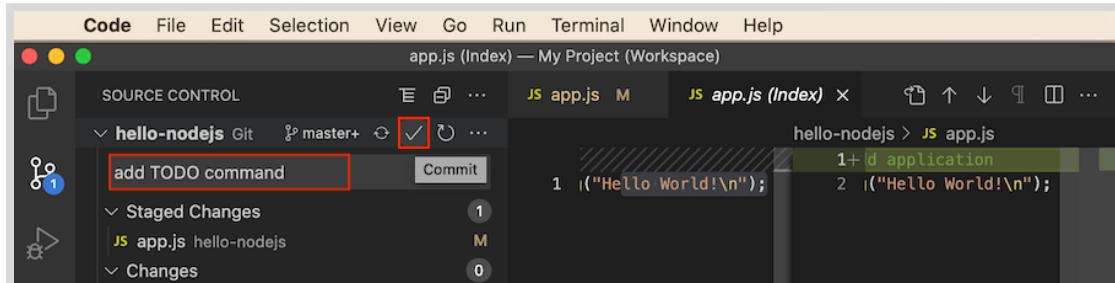


Figure 1.17: Commit staged files in VS Code.

## 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 publish local commits to the remote repository, Git performs a push operation. To retrieve commits from the remote repository, Git performs a pull 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. This is the synchronize changes icon, highlighted in the following figure.

The icon shows there are zero commits to download and one commit to upload to the remote repository:

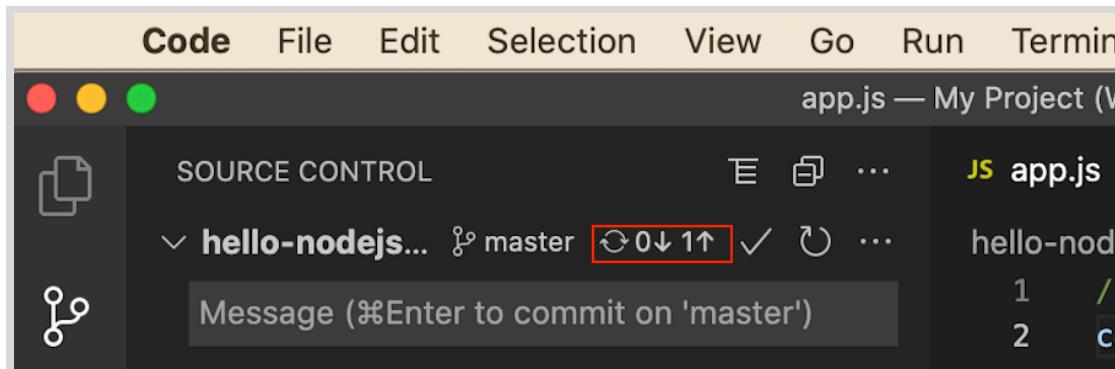


Figure 1.18: Git repository status in the source control view and status bar.



### Note

Because Git does not run in the background, VS Code and Git will not be aware of available commits to download until some synchronize action is performed. In general, do not assume that these numbers are always accurate.

Click the synchronize changes icon 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 REPOSITORIES heading, then you may not see a the synchronize changes icon.

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

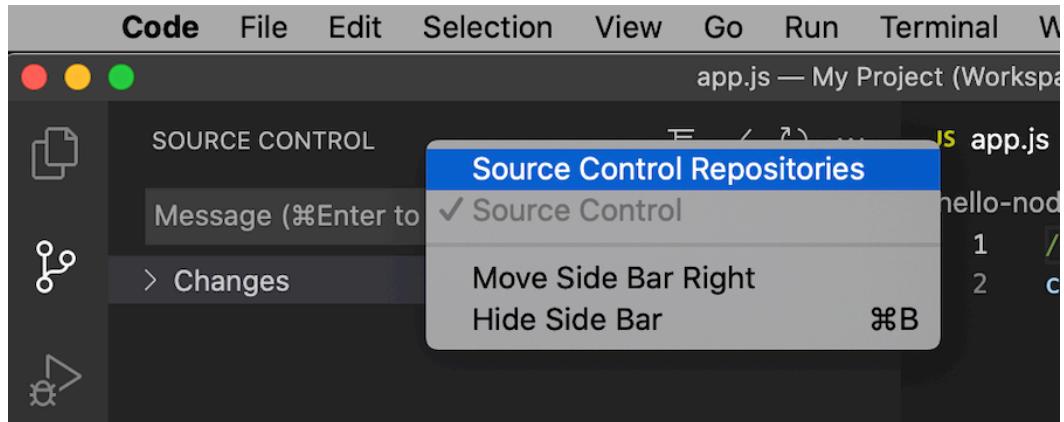


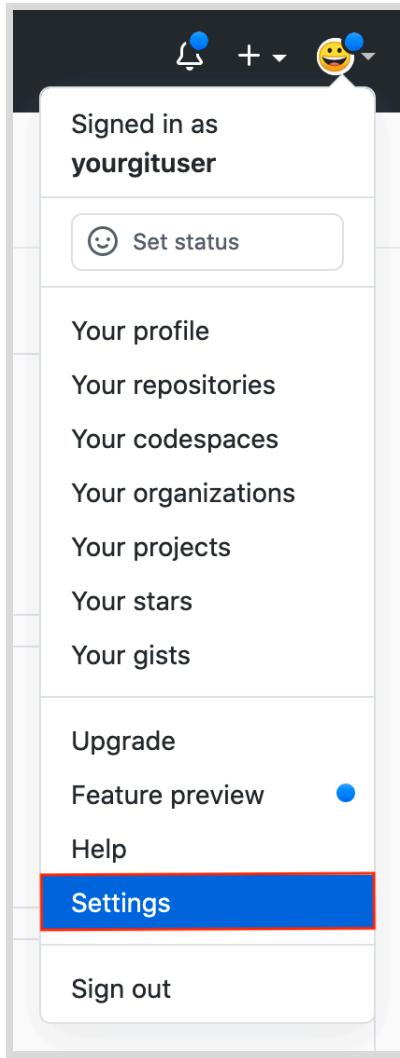
Figure 1.19: Enable the source control repositories listing.

## Adding a Github Access Token

If you attempt to push a commit to a remote repository for the first time, VS Code redirects you to a browser window to login into Github to provide access. After logging into Github, generate a personal access token for VSCode to use for your repository. A personal access token is required to be used in place of a password with the command line or with the Github API.

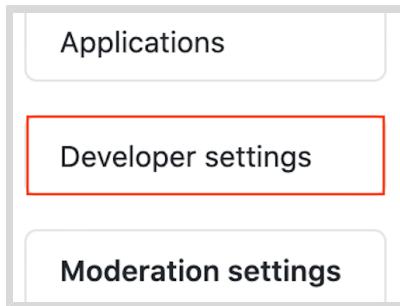
To create a personal access token take the following steps:

1. After logging into Github, select your profile icon in the upper right hand corner and click (Your Profile Icon > Settings).



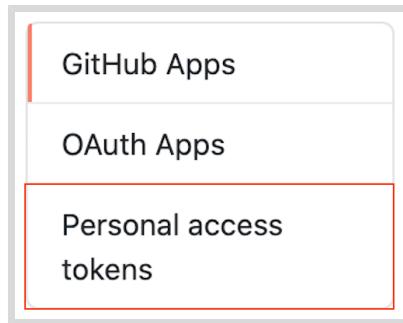
**Figure 1.20:** Github profile settings.

2. On the left sidebar, click Developer settings.



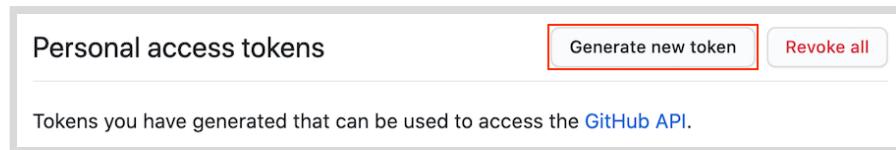
**Figure 1.21:** Github developer settings.

3. In the left sidebar, click Personal access tokens.



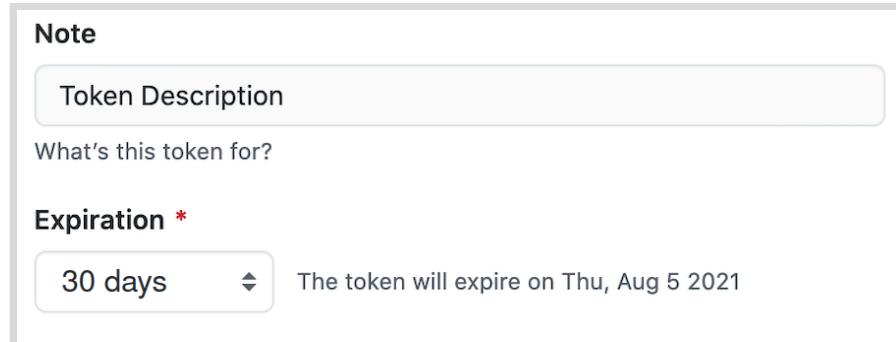
**Figure 1.22: Github personal access token.**

4. Click Generate new token.



**Figure 1.23: Generate new GitHub token.**

5. In the Note field, give your token a descriptive name. Select an appropriate Expiration date for your token.



**Figure 1.24: Token note and expiration**

6. Select the scopes, or permissions, you would like to grant this token. To use your token to access repositories from the command line, select repo.

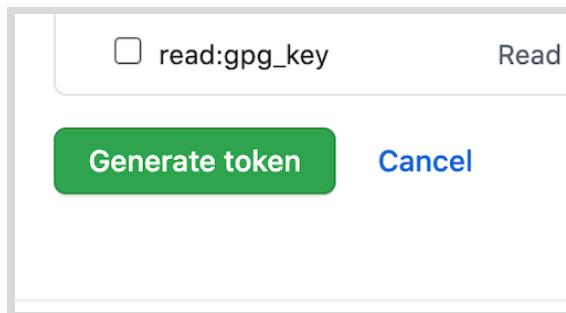
**Select scopes**

Scopes define the access for personal tokens. [Read more about OAuth scopes.](#)

<input checked="" type="checkbox"/> <b>repo</b>	Full control of private repositories
<input checked="" type="checkbox"/> repo:status	Access commit status
<input checked="" type="checkbox"/> repo_deployment	Access deployment status
<input checked="" type="checkbox"/> public_repo	Access public repositories
<input checked="" type="checkbox"/> repo:invite	Access repository invitations
<input checked="" type="checkbox"/> security_events	Read and write security events
<input type="checkbox"/> <b>workflow</b>	Update GitHub Action workflows
<input type="checkbox"/> <b>write:packages</b>	Upload packages to GitHub Package Registry
<input type="checkbox"/> read:packages	Download packages from GitHub Package Registry
<input type="checkbox"/> <b>delete:packages</b>	Delete packages from GitHub Package Registry
<input type="checkbox"/> <b>admin:org</b>	Full control of orgs and teams, read and write org projects
<input type="checkbox"/> write:org	

**Figure 1.25: Token scope and permissions.**

7. Towards the bottom of the page, click Generate token.



**Figure 1.26: Generate the GitHub token.**

8. Copy the generated token and provided it when prompted by VS Code. Alternatively, if you are asked for username and password, use the token in the username field and leave the password field empty.

**Personal access tokens**

[Generate new token](#) [Revoke all](#)

Tokens you have generated that can be used to access the [GitHub API](#).

Make sure to copy your personal access token now. You won't be able to see it again!

<input checked="" type="checkbox"/> ghp_PtBDf3I9DC0TvhbqvRkjGYJ80HBhqy1UAc92		<a href="#">Delete</a>
--	--	------------------------

**Figure 1.27: Copy the generated token.**

9. Add the personal access token to the VS Code prompt.

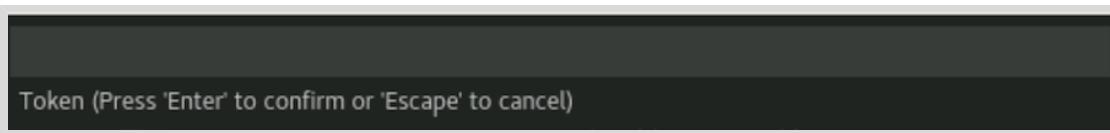


Figure 1.28: Use the Generated token.

## 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 **initialize** 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.

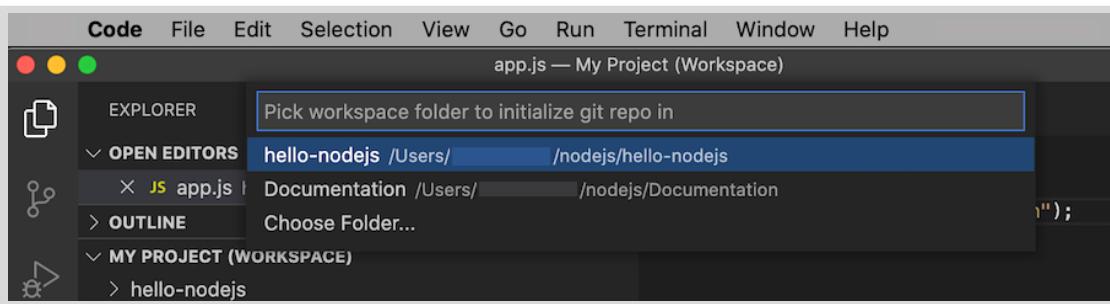


Figure 1.29: Using VS Code to initialize a Git repository.

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.

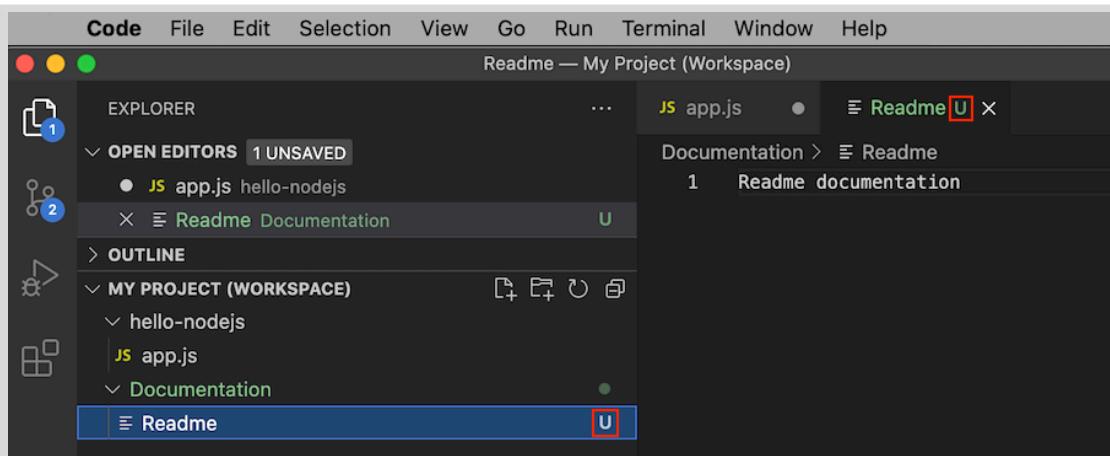


Figure 1.30: Untracked files in a Git repository.

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.

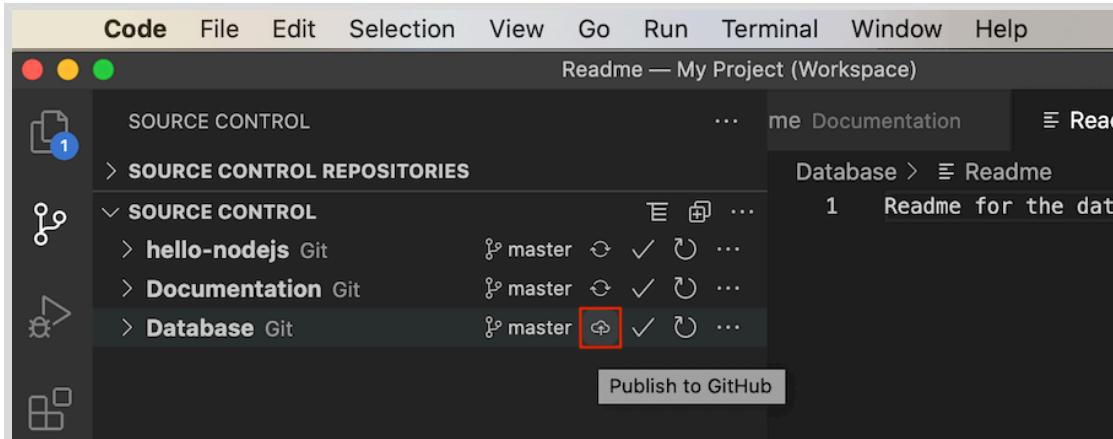
SSH access to Git repository requires the configuration of your SSH keys with the code hosting platform.

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` list.



**Figure 1.31: Publish a local Git repository to a remote repository.**

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



## References

For more information about installing Git, refer to the Git documentation at  
<https://git-scm.com/book/en/v2/Getting-Started-Installing-Git>

### Git Basics

<https://git-scm.com/book/en/v2/Git-Basics-Getting-a-Git-Repository>

For more information about using Git and version control in VS Code, refer to the  
VS Code documentation at  
<https://code.visualstudio.com/docs/editor/versioncontrol>

For more information about using a personal access token, refer to the Github  
documentation at

### Creating a personal access token

<https://docs.github.com/en/github/authenticating-to-github/keeping-your-account-and-data-secure/creating-a-personal-access-token>

## ► 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.

### Before You Begin

To perform this exercise, ensure that:

- 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.

## Instructions

### ► 1. Download and install Git.

#### 1.1. 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.

#### 1.2. 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 2.24.2 (Apple Git-127)
```



**Note**

The Git version on your system might differ. Any recent version of Git will work with this course.

1.3. 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.

2.1. Open VS Code.

2.2. Click **Terminal > New Terminal** to open an integrated terminal.

- 2.3. 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.

- 2.4. In a browser, navigate to <https://github.com>. If you do not have a GitHub account, then create one. Log in to GitHub.
- 2.5. In the VS Code integrated terminal, execute `git config --global user.name yourgituser`, replacing `yourgituser` with your GitHub user name.
- 2.6. 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.

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\dk141\Documents\Dan\vscode_workspace> git --version
git version 2.23.0.windows.1
PS C:\Users\dk141\Documents\Dan\vscode_workspace> git config --global user.name yourgituser
PS C:\Users\dk141\Documents\Dan\vscode_workspace> git config --global user.email user@example.com
PS C:\Users\dk141\Documents\Dan\vscode_workspace> 
```

**Figure 1.32: 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.

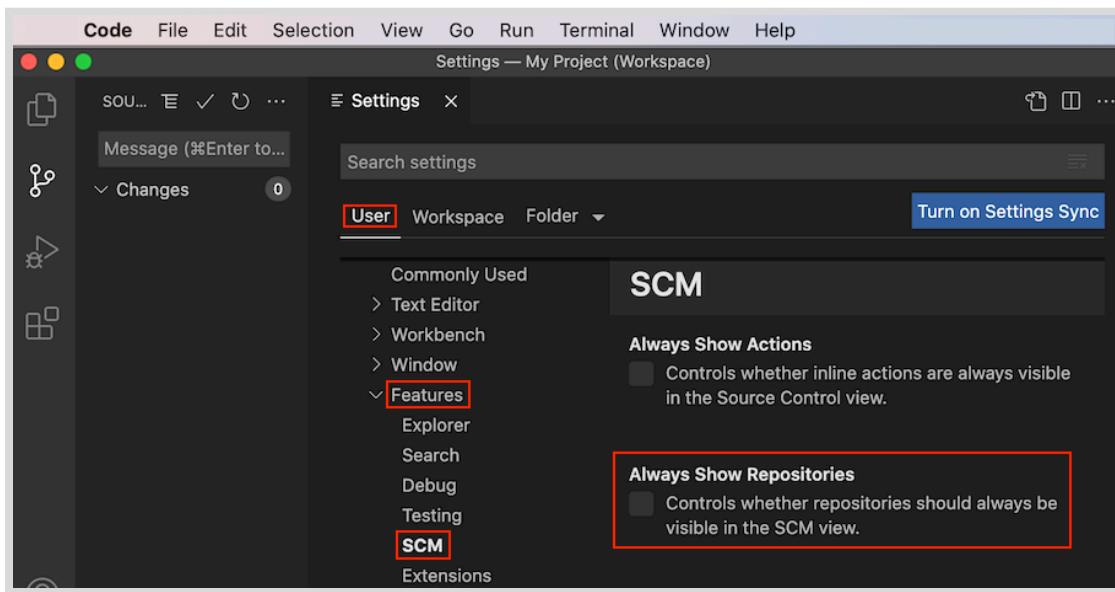
To cache your credentials on Linux or macOS systems, execute the following command from a system terminal:

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

- 3. Enable the **Always Show Repositories** source control management option in VS Code.

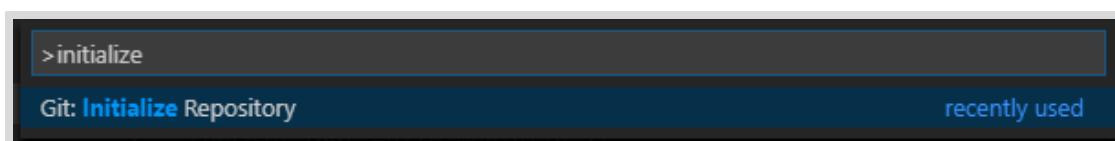
- 3.1. Access the Command Palette (**View > Command Palette...**) and type **settings**. Select Preferences: Open Settings (UI) from the list of options.
- 3.2. When the **Settings** window displays, click **User > Features > SCM**.

- 3.3. VS Code displays Source Control Management (SCM) options for VS Code. Select **Always Show Repositories**.



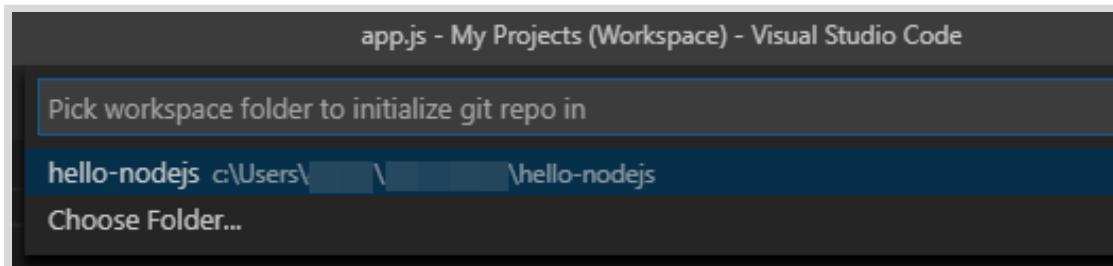
**Figure 1.33: Option to always show the source control repositories list in the source control view of VS Code.**

- 3.4. 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.
  - 4.1. Download the following zip file to your system:  
<https://github.com/RedHatTraining/DO101x-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.
  - 4.2. Click **File > Add Folder to Workspace...**
  - 4.3. 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.
  - 5.1. Access the VS Code Command Palette (**View > Command Palette...**).
  - 5.2. 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 1.34: Git repository initialization using the command palette.**

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

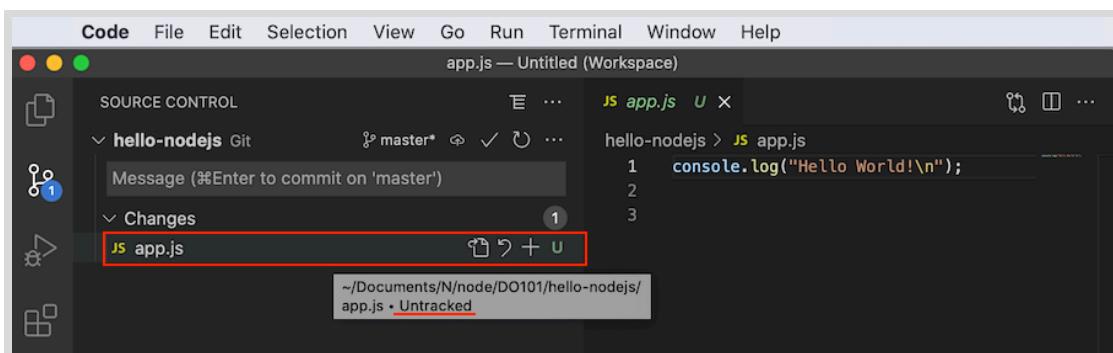


**Figure 1.35: Selection prompt to initialize a local Git repository.**

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

- 6.1. Click **View > SCM** to access the **Source Control** view in the Activity Bar.

- 6.2. 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.

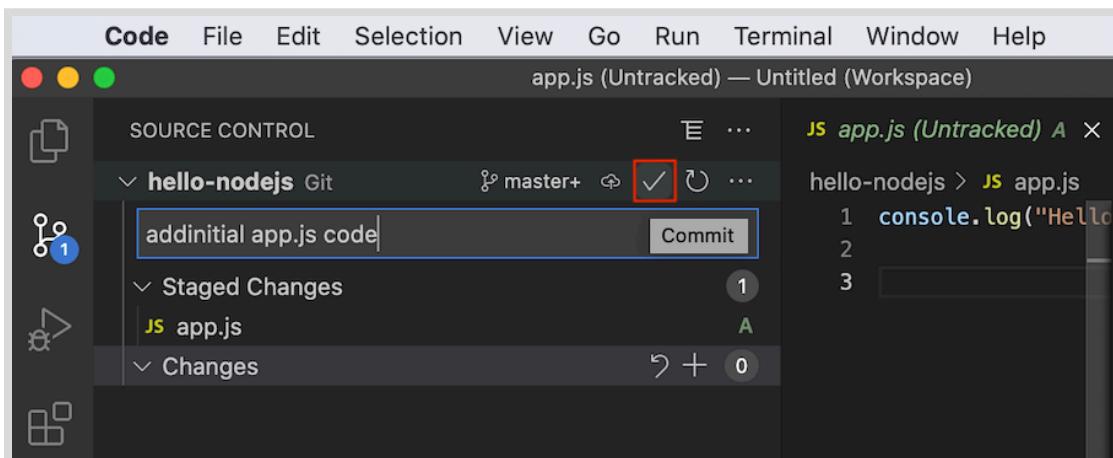


**Figure 1.36: List of changed files.**

This stages the `app.js` file for the next commit.

The file now appears under the **STAGED CHANGES** heading.

- 6.3. 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 1.37: VS Code commit message box.**

- ▶ 7. Create a new GitHub repository to host your project files. Add an access token for access to your git repository.
  - 7.1. In a browser, navigate to <https://github.com>. If you are not logged in to Github, then log in.
  - 7.2. Click the + on the upper-right, and then select New repository from the list displayed.

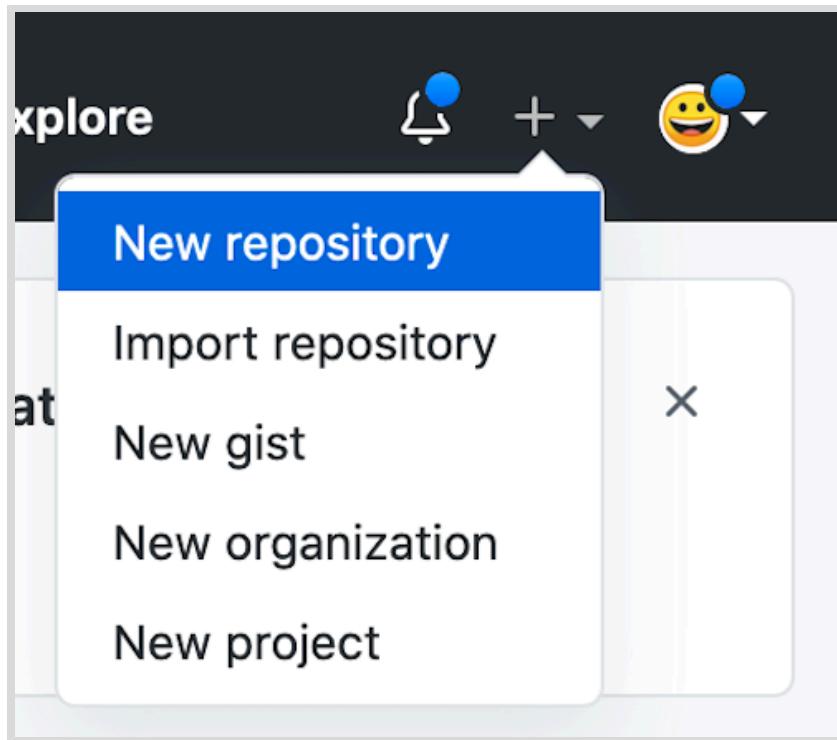


Figure 1.38: Create a new Git repository on GitHub.

- 7.3. 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:

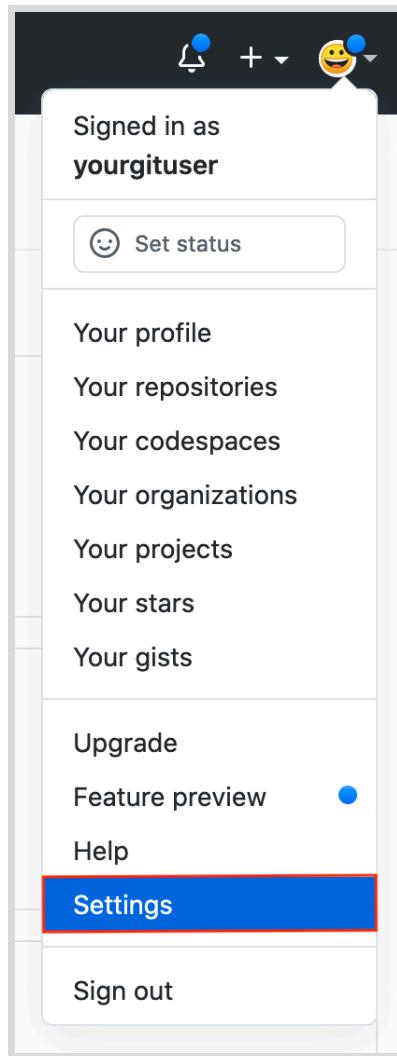
The screenshot shows the GitHub repository summary page for 'yourgituser / hello-nodejs'. At the top, there are navigation links for Code, Issues (0), Pull requests (0), Projects (0), Wiki, Security, Insights, and Settings. Below these, a 'Quick setup — if you've done this kind of thing before' section provides instructions for cloning the repository via Set up in Desktop, HTTPS, or SSH, with the URL <https://github.com/yourgituser/hello-nodejs.git>. It also suggests creating a new file or uploading an existing one, and recommends including a README, LICENSE, and .gitignore. A code block shows the initial setup commands:

```
echo "# hello-nodejs" >> README.md  
git init  
git add README.md  
git commit -m "first commit"  
git remote add origin https://github.com/yourgituser/hello-nodejs.git  
git push -u origin master
```

Below this, sections for '...or create a new repository on the command line' and '...or push an existing repository from the command line' provide their respective command-line scripts. The final section, '...or import code from another repository', includes a 'Import code' button and a note about initializing the repository from Subversion, Mercurial, or TFS.

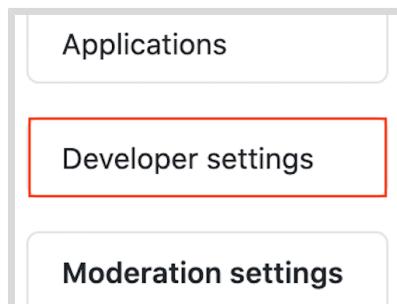
**Figure 1.39: The summary page for a new GitHub repository.**

- 7.4. Create a personal access token to allow access to your remote repository. Select your profile icon in the upper right hand corner and click (Your Profile Icon > Settings).



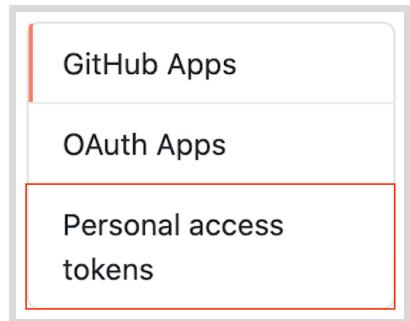
**Figure 1.40: GitHub profile settings menu item.**

7.5. On the left sidebar, click Developer settings.



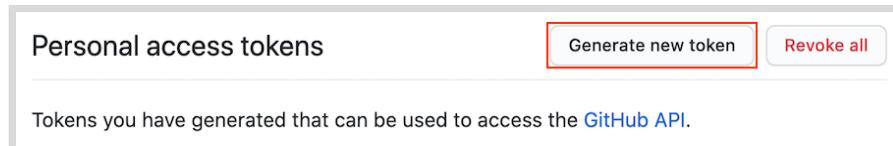
**Figure 1.41: GitHub developer settings menu item.**

7.6. In the left sidebar, click Personal access tokens.



**Figure 1.42: GitHub personal access token menu item.**

- 7.7. Click Generate new token.



**Figure 1.43: Generate new GitHub token button.**

- 7.8. In the Note field, give your token a descriptive name. Select 30 days as the Expiration date for your token.

A screenshot of a form for creating a new GitHub token. It includes a "Note" section with a "Token Description" input field and a "What's this token for?" placeholder. Below that is an "Expiration \*" section with a dropdown menu showing "30 days" and a note stating "The token will expire on Thu, Aug 5 2021".

**Figure 1.44: Token note and expiration fields.**

- 7.9. Select the scopes, or permissions, you would like to grant this token. To use your token to access repositories from the command line, select repo.

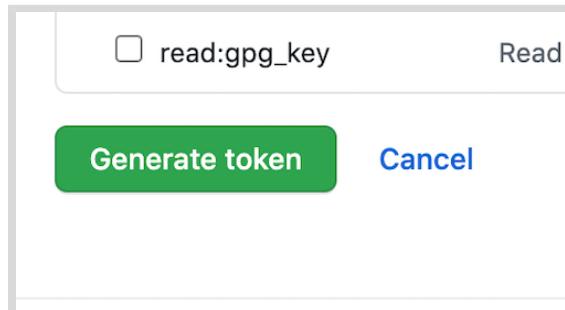
**Select scopes**

Scopes define the access for personal tokens. [Read more about OAuth scopes.](#)

<input checked="" type="checkbox"/> <b>repo</b>	Full control of private repositories
<input checked="" type="checkbox"/> repo:status	Access commit status
<input checked="" type="checkbox"/> repo_deployment	Access deployment status
<input checked="" type="checkbox"/> public_repo	Access public repositories
<input checked="" type="checkbox"/> repo:invite	Access repository invitations
<input checked="" type="checkbox"/> security_events	Read and write security events
<input type="checkbox"/> <b>workflow</b>	Update GitHub Action workflows
<input type="checkbox"/> <b>write:packages</b>	Upload packages to GitHub Package Registry
<input type="checkbox"/> read:packages	Download packages from GitHub Package Registry
<input type="checkbox"/> <b>delete:packages</b>	Delete packages from GitHub Package Registry
<input type="checkbox"/> <b>admin:org</b>	Full control of orgs and teams, read and write org projects
<input type="checkbox"/> write:org	

**Figure 1.45: Token scope and permissions settings.**

7.10. Click **Generate token**.



**Figure 1.46: Generate the GitHub token button.**

7.11. Copy the generated token.

**Personal access tokens**

[Generate new token](#) [Revoke all](#)

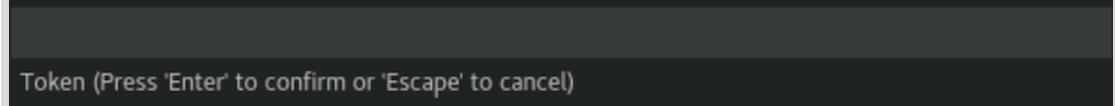
Tokens you have generated that can be used to access the [GitHub API](#).

Make sure to copy your personal access token now. You won't be able to see it again!

✓ ghp\_PtBDf3I9DC0TvhbqvRkjGYJ80HBhqy1UAc92  [Delete](#)

**Figure 1.47: Copy the generated token to the clipboard.**

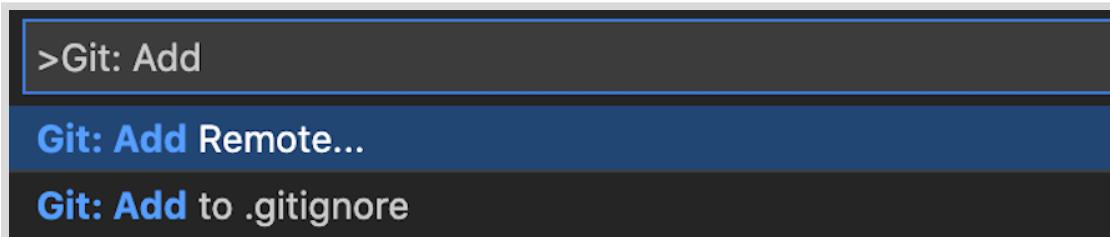
7.12. Save the personal access token to be used later.



Token (Press 'Enter' to confirm or 'Escape' to cancel)

**Figure 1.48: Use the generated token.**

- ▶ **8.** Add your new GitHub repository as a remote repository for the `hello-nodejs` project.
  - 8.1. In VS Code, type `Git: Add` in the Command Palette (`View > Command Palette...`). Then, select `Git: Add Remote` from the list of options.



**Figure 1.49: The remote URL prompt to add a remote Git repository.**

- 8.2. If you have more than one local Git repository in VS Code, then select `hello-nodejs` from the list of options.
- 8.3. At the next prompt, enter the HTTPS URL of your `hello-nodejs` GitHub repository with the personal access token. The URL form is:  
`https://TOKEN@github.com/yourgituser/hello-nodejs`.  
In the URL, replace `TOKEN` with the generated personal access token.



### Warning

If you do not add the personal access token to your remote URL, you will not be authorized to push commits to your repository.

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.

- ▶ **9.** Publish your local repository commits to the GitHub repository.
  - 9.1. Locate the `hello-nodejs` entry in the `SOURCE CONTROL` section, and then click the "Publish Changes" icon.

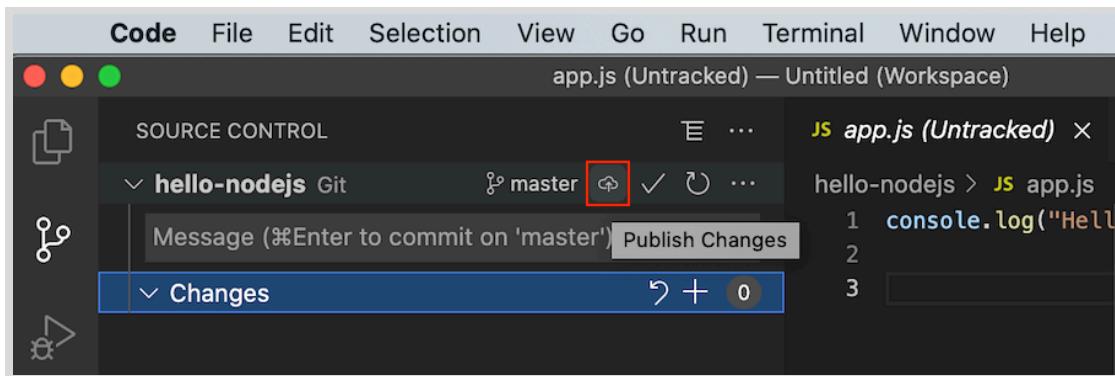


Figure 1.50: Source control view with publish changes highlighted.

The first time VS Code connects to GitHub, a prompt for your GitHub credentials displays. When prompted, provide your GitHub personal access token.

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

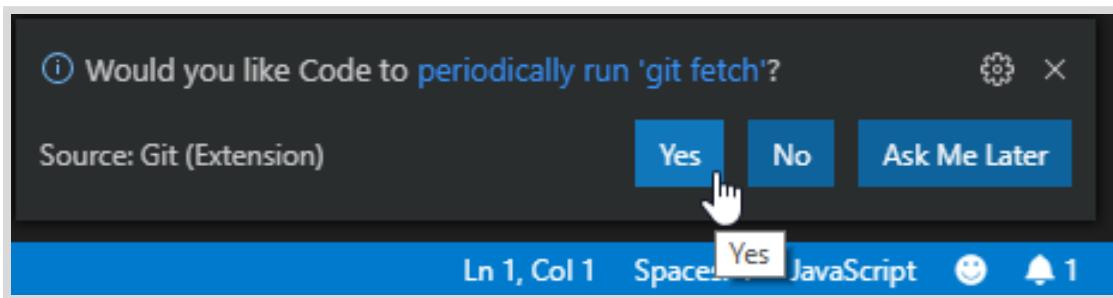
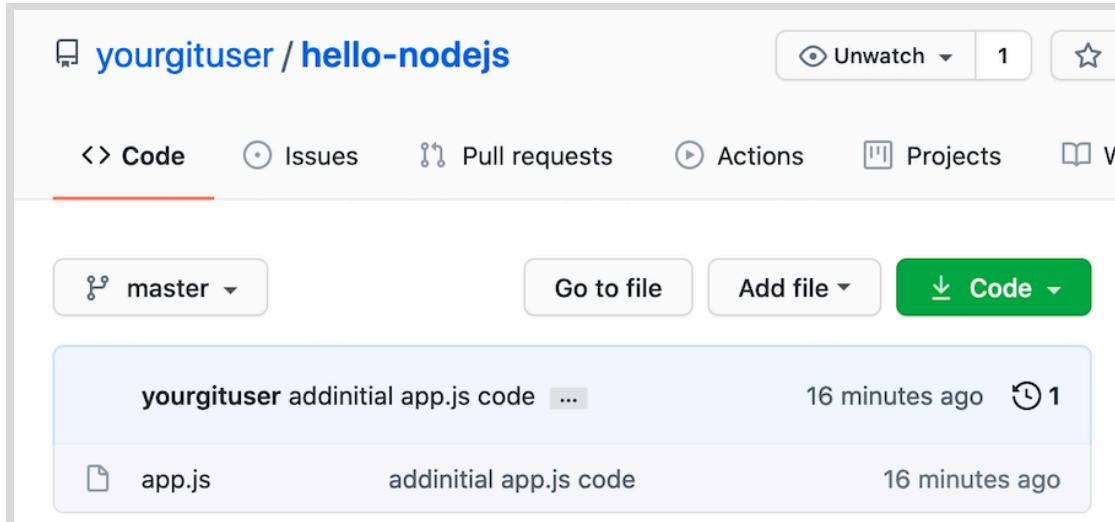


Figure 1.51: VS Code prompt to periodically fetch new commits.

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 1.52:** Local project files are present on GitHub.

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

## Finish

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. A base branch named **main** is preferred by many projects. Some platforms use **master** by default. When you create a commit in your repository, the base branch is updated with the new commit.

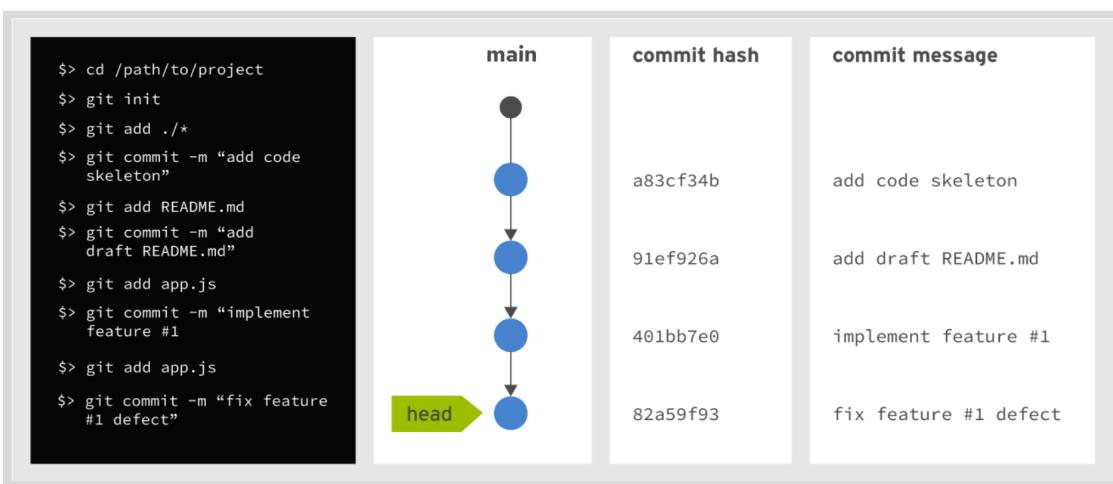
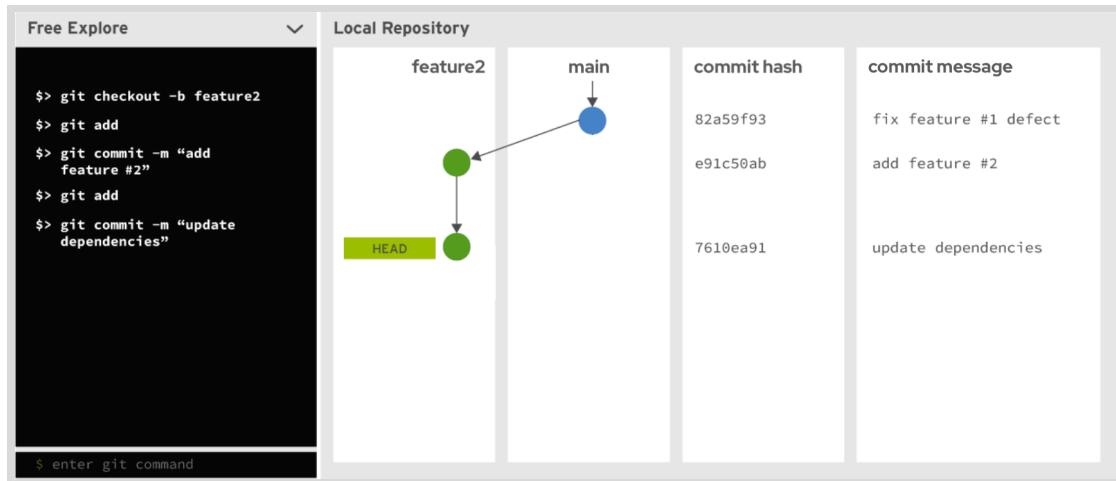


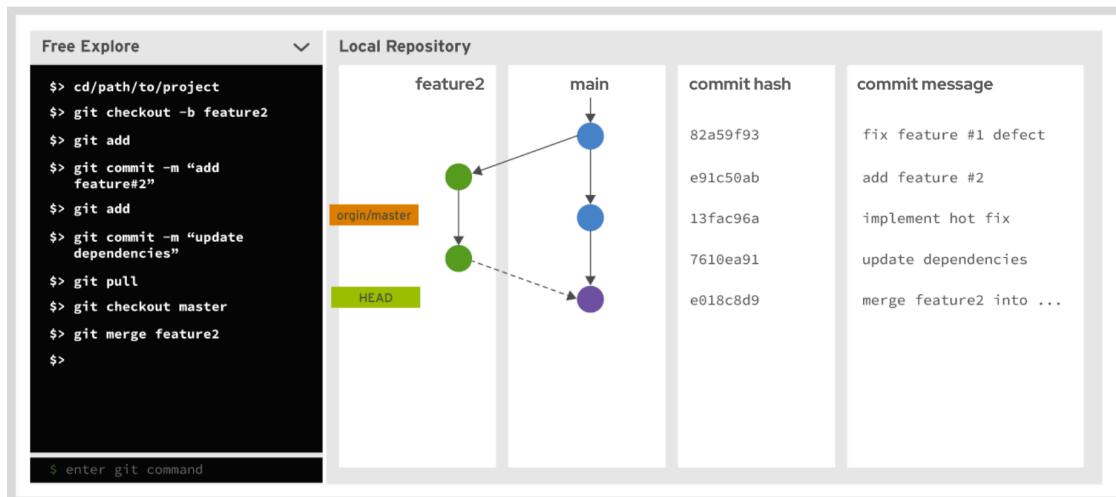
Figure 1.53: Commits to the main branch of a Git repository.

By convention, the **main** 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 **main** branch. This new branch, called a **feature branch**, contains commits corresponding to code changes for the new feature. The **main** branch is not affected by commits to the feature branch.



**Figure 1.54: 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 **main** 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 **main** branch. **Merging** is the process of combining the commit histories from two separate branches into a single branch.



**Figure 1.55: Merging a feature branch into the main branch.**

## 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 change 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.

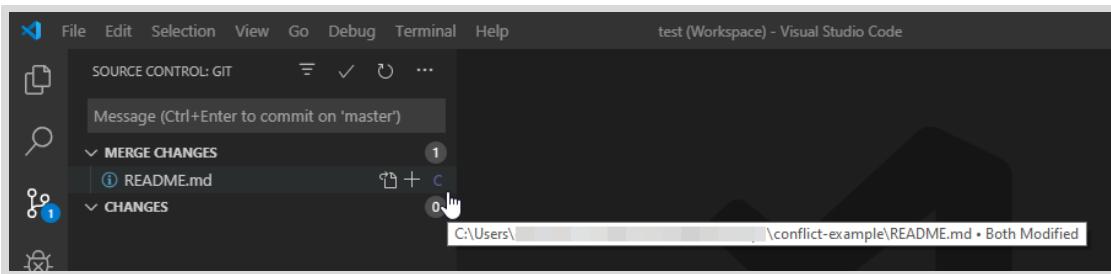


Figure 1.56: A merge conflict in the Source Control view of VS Code.

Git also inserts markers in each affected file to indicate the sections that contain content conflicts from both branches. If you click 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.

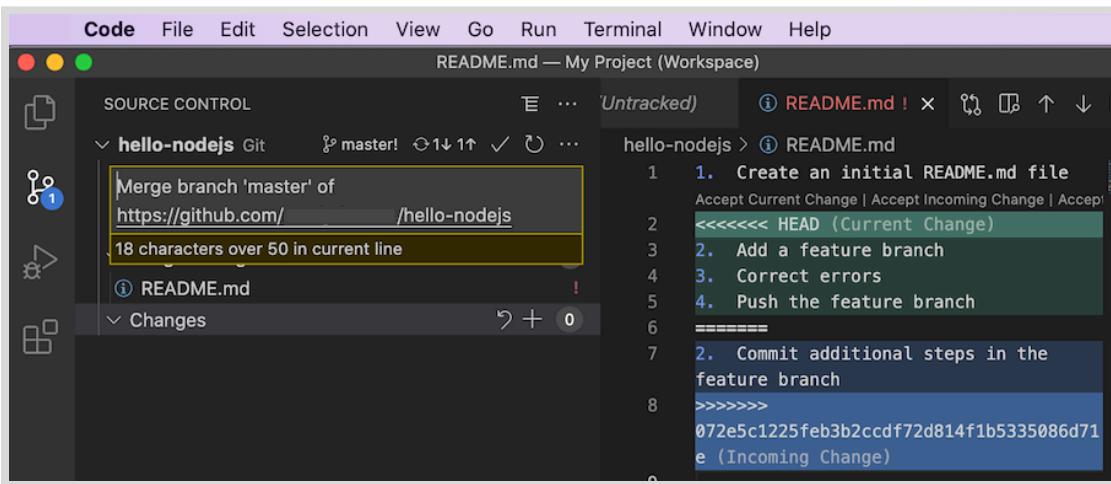


Figure 1.57: VS Code highlights the conflicts in a merge conflict file.

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 incoming changes pulled in from the remote repository, 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.

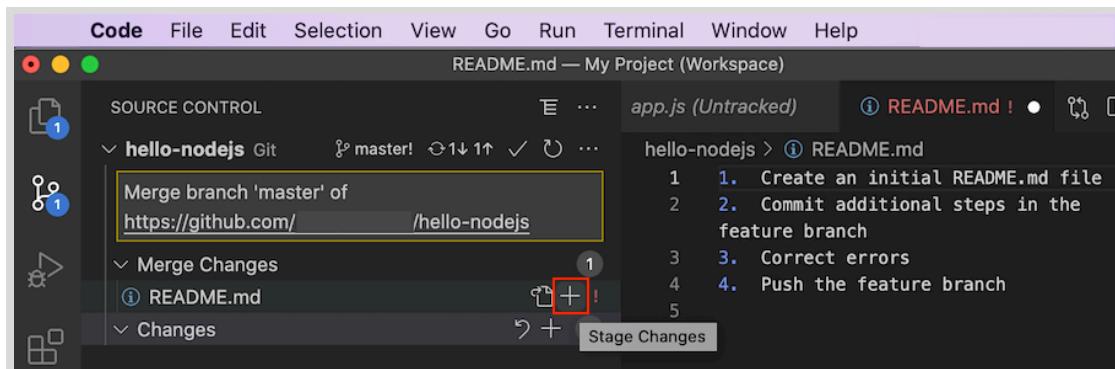


Figure 1.58: Resolving a merge conflict in VS Code.

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, `main`. Developers push code changes directly to the `main` 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 `main` 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, conflicts occur often. 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 `main` branch. Incomplete code compromises the stability of the `main` branch. Teams that use a centralized workflow must adopt additional processes and communication strategies to mitigate these risks.

One way teams mitigate code stability issues with a centralized workflow 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 **main** branch. The aim of this workflow is to always have deployable and stable code for every commit on the **main** 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. One or more 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 **main** 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.

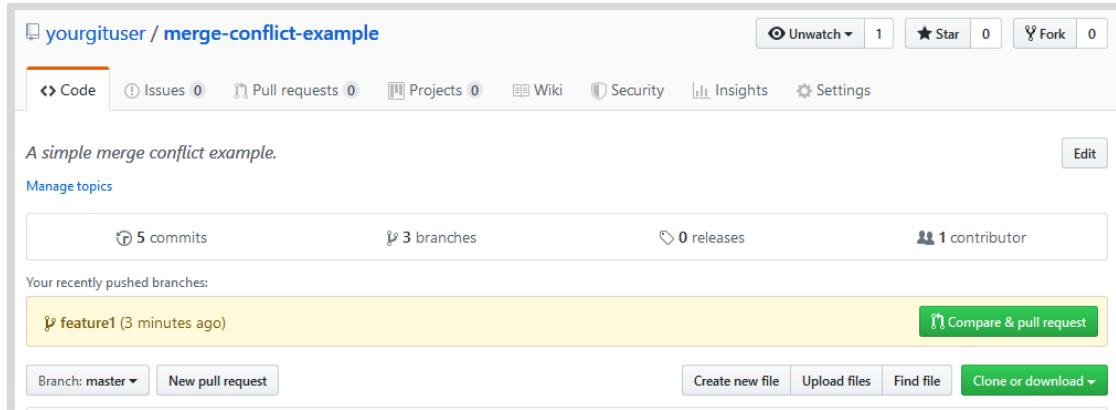
When a developer is ready to merge a feature branch into the **main** branch, the developer pushes the local feature branch to the remote repository. Then, the developer submits a request to the team, such as a pull request or merge request, to have the code changes in the feature branch reviewed by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

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 **main**.

**Note**

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 1.59: GitHub user interface after you push a feature branch.**

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

By default, you request that the **feature1** branch be merged (or pulled) into the **main** 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.

**Figure 1.60: The GitHub form to submit a pull request.**

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

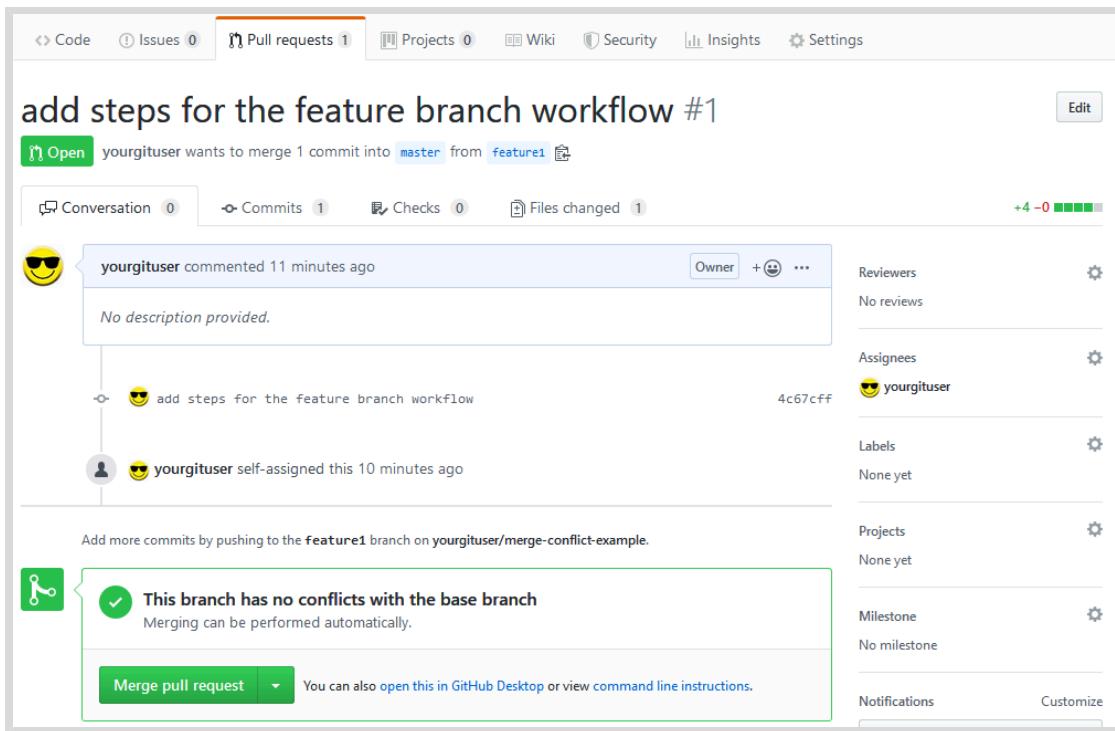


Figure 1.61: A GitHub pull request.

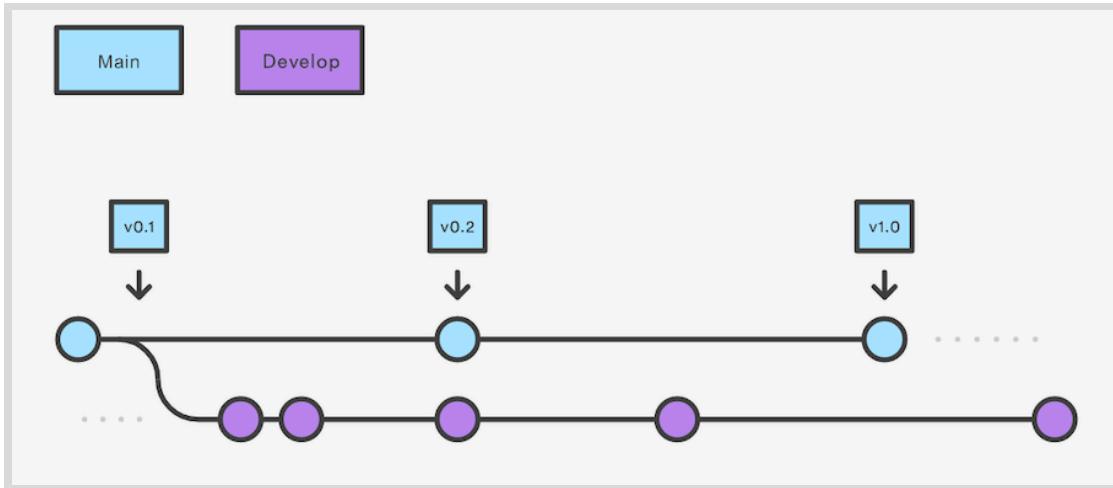
After a reviewer grants approval, you can merge the pull request. When you are ready to merge the changes to the **main** branch, click **Merge pull request**. GitHub displays a message for the merge operation. Click **Confirm merge** to merge the **feature1** branch into the **main** branch.

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

## Gitflow Workflow

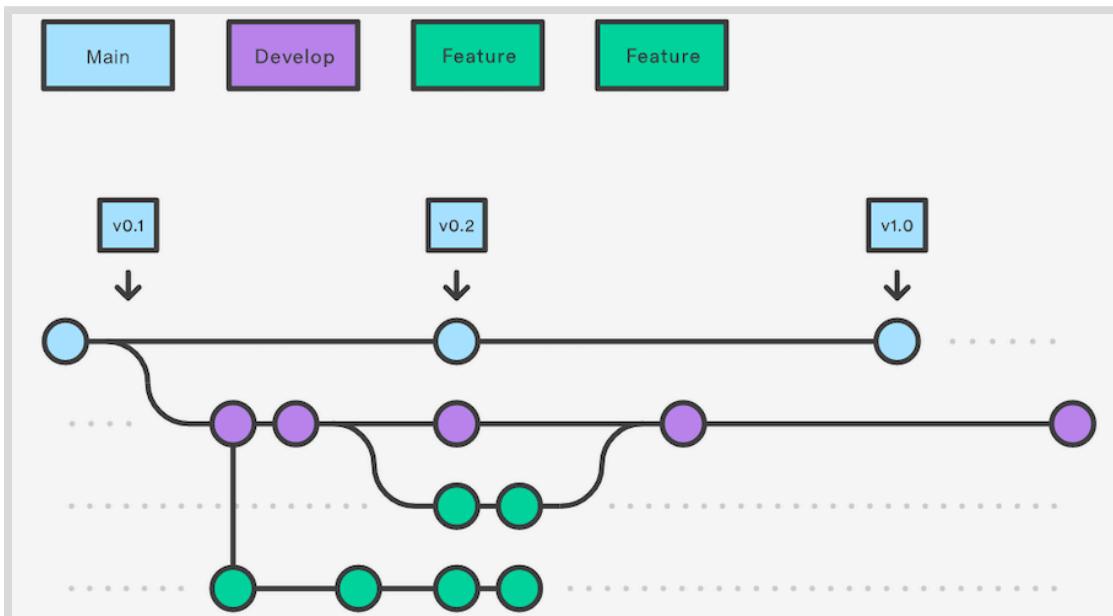
The **Gitflow Workflow** uses all the same constructs discussed in the **Feature Branch Workflow** and also defines a strict branching model around project releases. Gitflow is ideally suited for projects that have a scheduled release cycle and for the DevOps best practice of continuous delivery. In addition to feature branches, it uses individual branches for preparing, maintaining, and recording releases.

Instead of a single **main** branch, this workflow uses two branches to record the history of the project. The **main** branch stores the official release history, and the **develop** branch serves as an integration branch for features. It's also convenient to tag all commits in the **main** branch with a version number.



**Figure 1.62: Gitflow develop and main branches.**

Instead of branching off of the `main`, feature branches use the `develop` branch as their parent. Completed features are merged back into `develop` and never interact with `main`.



**Figure 1.63: Gitflow feature branches from develop.**

Once the `develop` branch has acquired enough features, a release branch is created with `develop` as the parent. Creating this branch starts the next release cycle, so no new features can be added after this point. Only bug fixes, documentation, or other release-oriented assets should be added. Once it is ready to ship, the release branch gets merged into `main` and tagged with a version number.

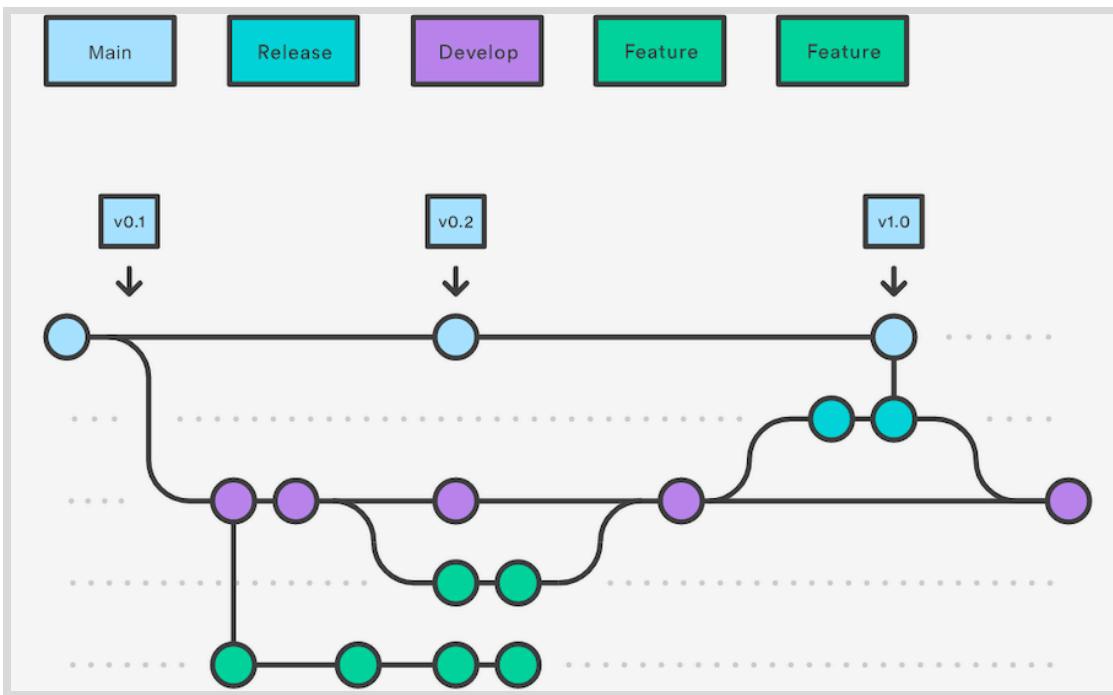


Figure 1.64: Gitflow release branches.

Using a dedicated branch to prepare releases makes it possible for one team to polish the current release while another team continues working on features for the next release. It also creates well-defined phases of development that are seen in the structure of the repository.

## 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.

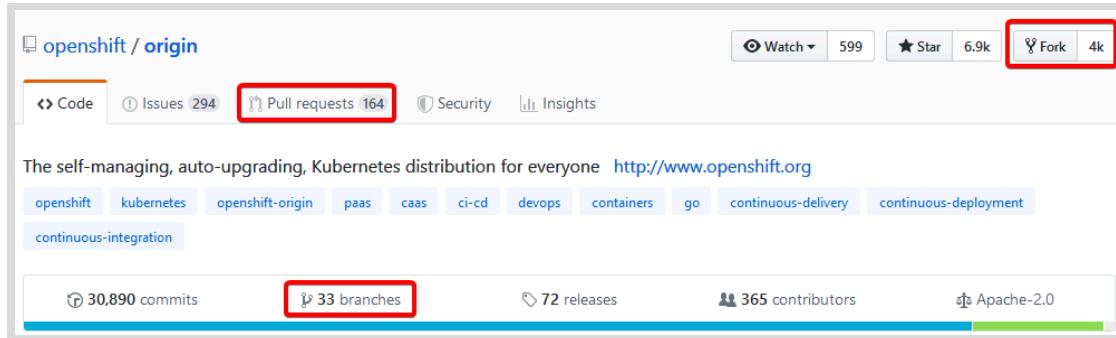


Figure 1.65: The OpenShift Origin GitHub repository.

Notice that there are more pull requests than branches in the OpenShift Origin GitHub repository. Most 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 REPOSITORIES** section contains an entry for each Git repository in your VS Code workspace. Each entry displays the current branch for the repository.

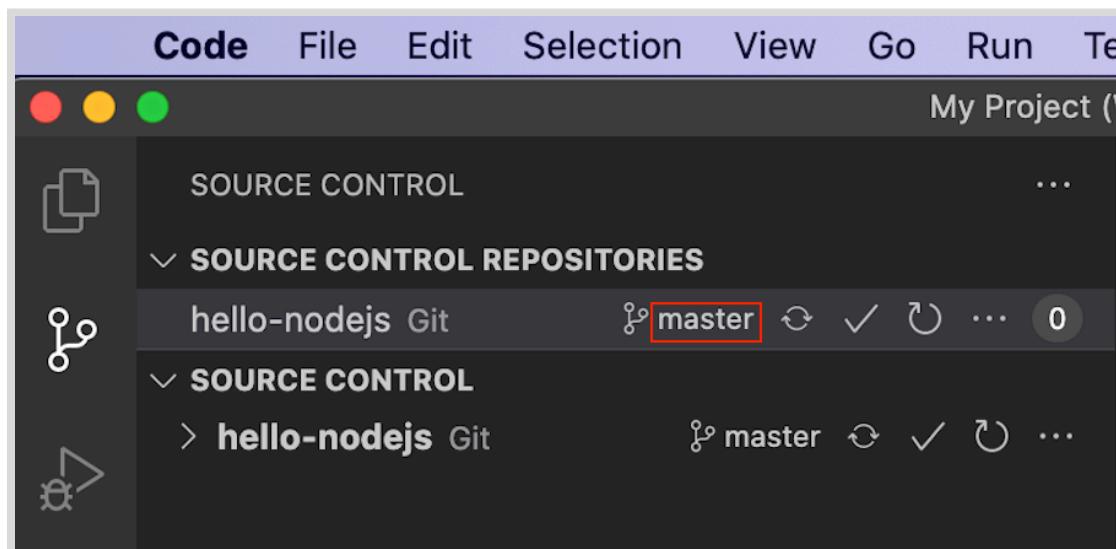
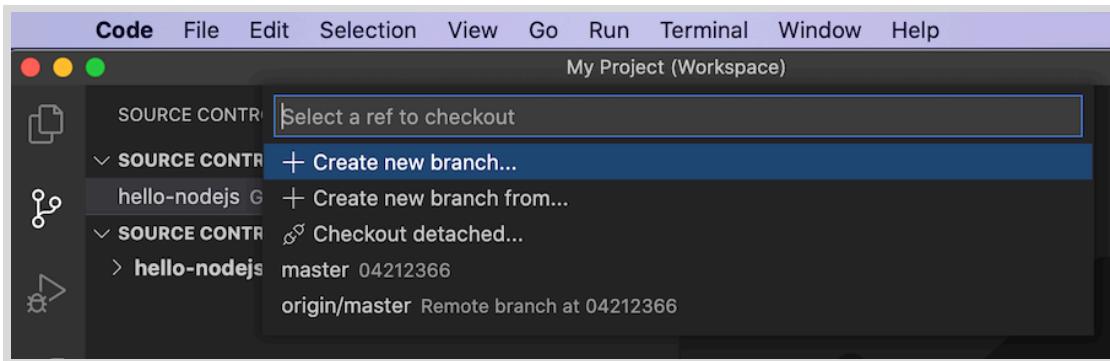


Figure 1.66: The Source Control view shows the current branch for each workspace Git repository.

To switch to a different branch in a repository, click the current branch name for the repository entry under the **SOURCE CONTROL REPOSITORIES** 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 1.67: 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`

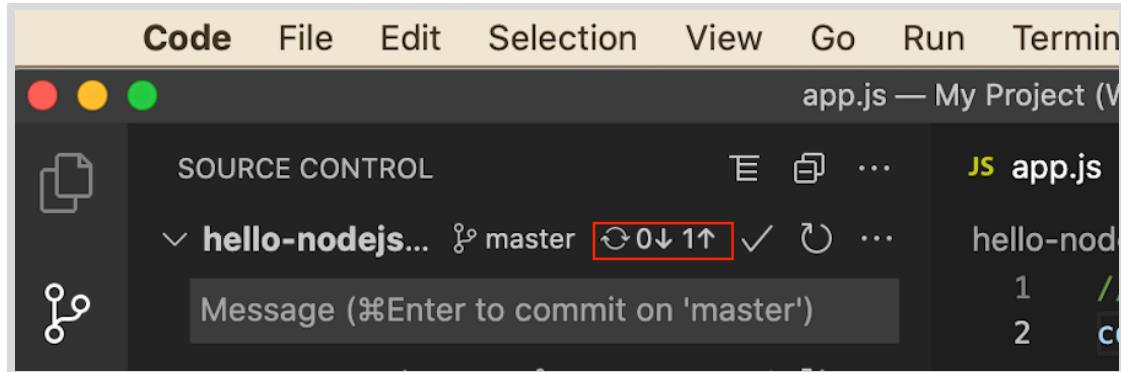
A branch naming standard also defines the set of allowable characters. Branch names are often limited to alphanumeric characters and field separators (such as `/`, `_`, or `-` characters).

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 1.68: Push a local commit to the remote repository.**



### References

For more information about Git branches, refer to the Git documentation at  
<https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell>

For more information about merging, refer to the Git documentation at  
<https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging>

For more information about using Git and version control in VS Code, refer to the  
VS Code documentation at  
<https://code.visualstudio.com/docs/editor/versioncontrol>

For more information about using the Gitflow Workflow, refer to the Atlassian  
documentation at  
<https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow>

## ► 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.

## Before You Begin

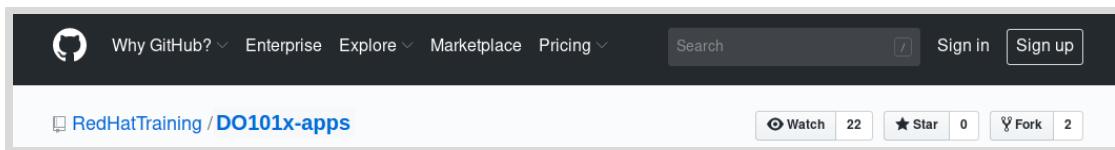
To perform this exercise, ensure that:

- 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.

## Instructions

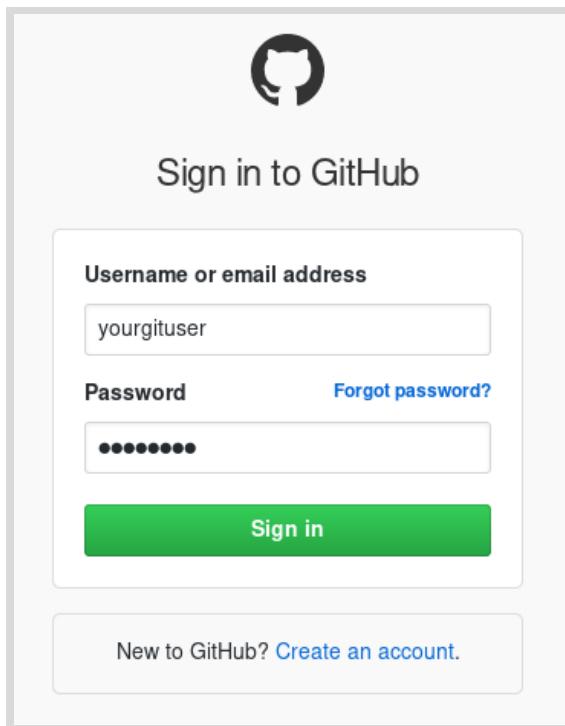
### ► 1. Fork the sample applications for this course into your personal GitHub account.

- 1.1. Open a web browser, and navigate to <https://github.com/RedHatTraining/DO101x-apps>. If you are not logged in to GitHub, then click **Sign in** in the upper-right corner.



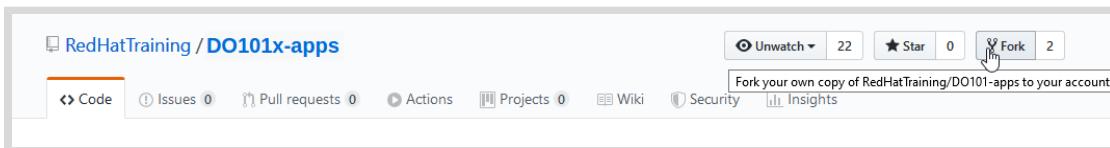
**Figure 1.69: Sign-in page for the course Git repository.**

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



**Figure 1.70: GitHub Sign-in page.**

13. Return to <https://github.com/RedHatTraining/DO101x-apps> and click **Fork** in the upper-right corner.



**Figure 1.71: Fork the Red Hat Training DO101x-apps repository.**

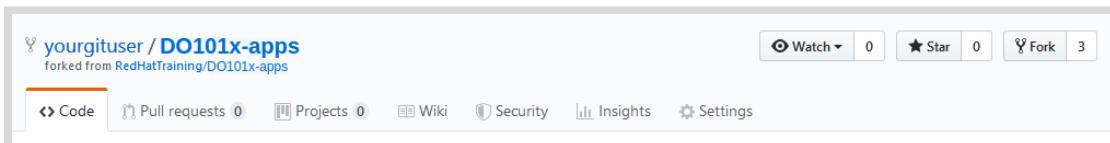
14. In the Fork DO101x-apps window, click *yourgituser* to select your personal GitHub project.



### Important

While it is possible to rename your personal fork of the <https://github.com/RedHatTraining/DO101x-apps> repository, the example output in this course assumes that you retain the name DO101x-apps when you fork the repository.

15. After a few minutes, the GitHub web interface displays your new *yourgituser/DO101x-apps* repository.



**Figure 1.72: A personal fork of the Red Hat Training DO101x-apps repository.**

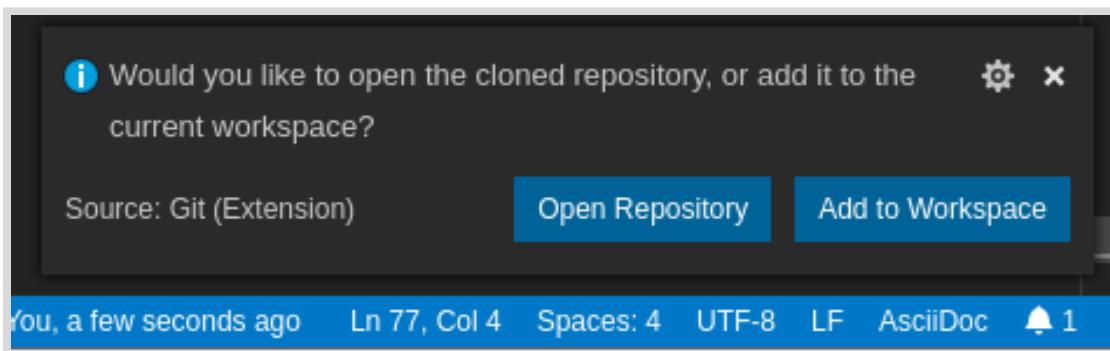
- 2. Clone the sample applications for this course from your personal GitHub account using VS Code.

- 2.1. If you do not have VS Code open from a previous exercise, then open it.
- 2.2. In the Command Palette (**View > Command Palette...**), type **clone**. Select **Git: Clone** from the list of options.



**Figure 1.73: Using the command palette to clone a repository.**

- 2.3. In the prompt that displays, enter the HTTPS URL for your repository, <https://TOKEN@github.com/yourgituser/D0101x-apps>.  
In the URL, replace *TOKEN* with the generated personal access token.
- 2.4. In the file window that opens, choose a local folder to store the repository clone. VS Code creates a D0101x-apps subfolder in the folder you select.  
Click **Select Repository Location** to select your location.
- 2.5. 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.



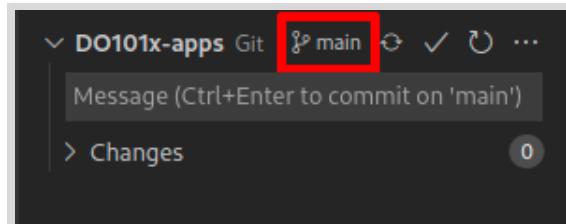
**Figure 1.74: Add a cloned repository to the current workspace.**



### 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 D0101x-apps folder. Select the D0101x-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.
- 3.1. From the Source Control view (**View > SCM**), click `main` in the D0101x-apps entry under **SOURCE CONTROL REPOSITORIES**.



**Figure 1.75: Checkout a new branch for a repository.**



**Note**

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

- 3.2. Select **Create new branch...** from the list of options.
- 3.3. When prompted, enter **devenv-versioning** for the branch name. The D0101x-apps entry in the Source Control view updates to the devenv-versioning branch.
- 3.4. In the Explorer view (**View > Explorer**), click the D0101x-express-helloworld/app.js file. VS Code opens an editor tab for the file.
- 3.5. To implement the new feature, add the following lines to the app.js file:

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

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

A screenshot of the VS Code code editor showing the file 'app.js'. Lines 8 through 10 have been added to handle the '/mars' endpoint. The code editor shows syntax highlighting for JavaScript.

**Figure 1.76: 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** endpoint.

**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.

- 4.1. Right-click `express-helloworld` in the Explorer view (**View > Explorer**) and then select **Open in Integrated Terminal**.
- 4.2. Execute the `npm install` command in the integrated terminal to install the application dependencies.
- 4.3. Execute the `npm start` command in the integrated terminal to start the application.

The screenshot shows the Visual Studio Code interface. At the top, there's a tab bar with 'JS app.js M X'. Below the tabs, the main workspace shows code for an Express.js application. The code defines two routes: one for the root ('/') returning 'Hello World!' and another for '/mars' returning 'Hello Mars!'. Lines 1 through 11 are visible. Below the code editor is a tab bar with 'PROBLEMS', 'OUTPUT', 'TERMINAL', and '...', with 'TERMINAL' being the active tab. The integrated terminal window displays the command `npm start` and its output: the application starts and listens on port 8080. The status bar at the bottom shows the path `/Documents/N/node/ex2/D0101-apps/express-helloworld`.

```

JS app.js M X
DO101-apps > express-helloworld > JS app.js > ⚡ app.get('/') callback
1  var express = require('express');
2  app = express();
3
4  app.get('/', function (req, res) {
5    res.send('Hello World!\n');
6  });
7
8  app.get('/mars', function(req, res) {
9    res.send('Hello Mars!\n');
10   });
11

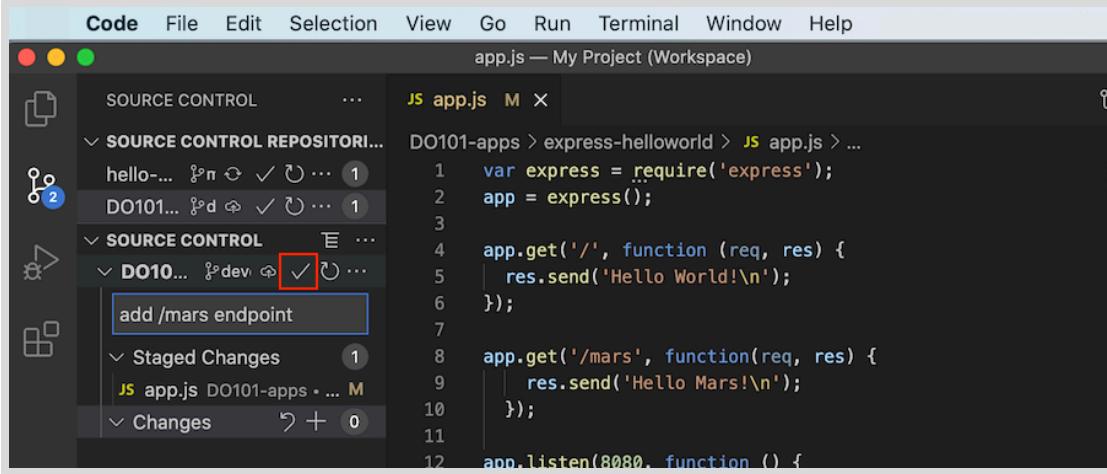
PROBLEMS OUTPUT TERMINAL ...
express-helloworld > npm start
> express-helloworld@1.0.0 start /Users/          /Documents/N/node/ex2/D0
101-apps/express-helloworld
> node app.js
Example app listening on port 8080!

```

**Figure 1.77: Executing a Node.js application from the integrated terminal.**

- 4.4. In a browser, navigate to `http://localhost:8080/`. Verify that the application responds with a `Hello World!` message.
- 4.5. In a browser, navigate to `http://localhost:8080/mars`. Verify that the application responds with a `Hello Mars!` message.
- 4.6. 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.

- 4.7. 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.
- 5.1. Access the Source Control view (**View > SCM**) and then click D0101x-apps in the SOURCE CONTROL REPOSITORIES list.
  - 5.2. Hover over the entry for the app.js file under the CHANGES heading for the D0101x-apps repository. Click the Stage Changes icon to add the changes to the app.js file to your next commit.
  - 5.3. Add a commit message of add /mars endpoint in the message prompt, and then click the check mark button to commit the staged changes:



The screenshot shows the Red Hat OpenShift IDE interface. The menu bar includes Code, File, Edit, Selection, View, Go, Run, Terminal, Window, and Help. The title bar says "app.js — My Project (Workspace)". The left sidebar has a "SOURCE CONTROL" section with "SOURCE CONTROL REPOSITORIES" expanded, showing "hello-..." and "DO101...". Under "DO101...", "Changes" is expanded, showing a staged change for "app.js" with the message "add /mars endpoint". A red box highlights the checkmark icon next to the message. The main editor area shows the code for "app.js":

```
DO101-apps > express-helloworld > JS app.js > ...
1 var express = require('express');
2 app = express();
3
4 app.get('/', function (req, res) {
5   res.send('Hello World!\n');
6 });
7
8 app.get('/mars', function(req, res) {
9   res.send('Hello Mars!\n');
10 });
11
12 app.listen(8080, function () {
```

Figure 1.78: 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.
- 6.1. In the Source Control view (**View > SCM**), locate the D0101x-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.

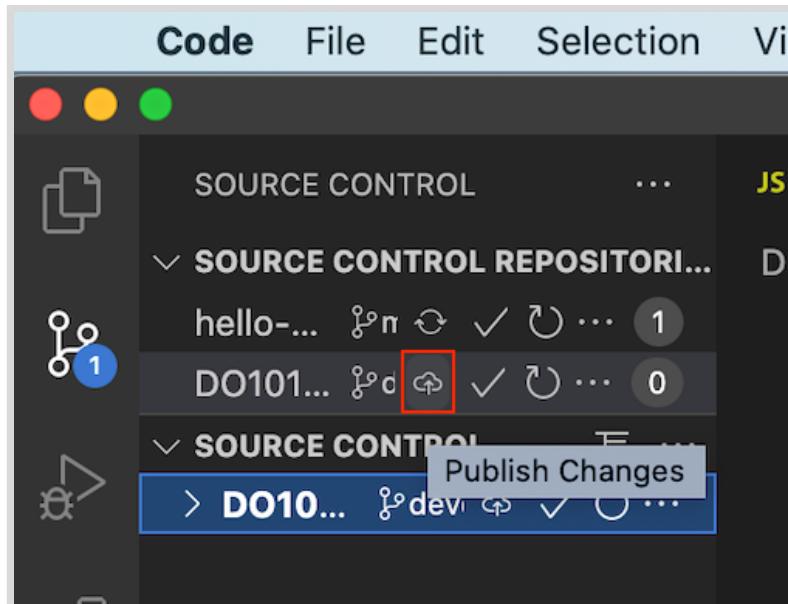


Figure 1.79: Publish a local branch to a remote repository.

- 6.2. In a browser, navigate to your personal DO101x-apps repository at <https://github.com/yourgituser/DO101x-apps>. The GitHub interface indicates that you recently pushed code changes to the devenv-versioning branch.

A screenshot of a GitHub repository page for 'yourgituser / DO101x-apps'. The page shows a notification message: 'devenv-versioning had recent pushes 1 minute ago'. Below the notification are buttons for 'Compare &amp; pull request', 'master', 'Go to file', 'Add file', and 'Code'.

Figure 1.80: A GitHub repository indicates a recent push for the new branch.

- 6.3. Click commits below the Code button to display a list of recent commits.
- 6.4. From the branch list, select devenv-versioning. An entry for the add /mars endpoint commit displays at the top of the commit history.



**Figure 1.81: The commit history for a GitHub repository branch.**

- 6.5. 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.

A screenshot of the GitHub code diff interface for the file 'express-helloworld/app.js'. The diff shows the addition of a new endpoint '/mars'. The code changes are as follows:

**Figure 1.82: GitHub displays commit code changes.**

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

## Finish

This concludes the guided exercise.

# Summary

---

In this chapter, you learned:

- Integrated Development Environments (IDEs) combine separate software development tools (compilers, test suites, linters) into a single application to improve your productivity.
- Git is a common software configuration management tool used to implement version control of application code changes.
- Many source code hosting services provide access to the Git repository for a project, which enables collaboration on source code development.



## Chapter 2

# Deploying Applications to Red Hat OpenShift Container Platform

### Goal

Deploy an application to OpenShift.

### Objectives

Deploy an application to Red Hat OpenShift Container Platform.

### Sections

Deploying an Application to Red Hat OpenShift Container Platform (and Guided Exercise)

# Deploying an Application to Red Hat OpenShift Container Platform

---

## Objectives

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

## 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.

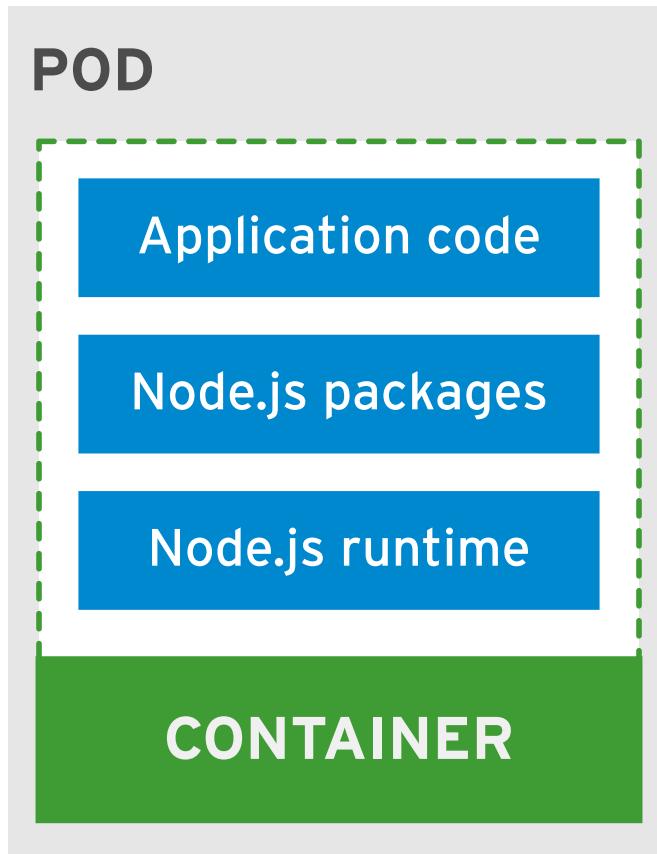
The Developer Sandbox for Red Hat OpenShift, at <https://developers.redhat.com/developer-sandbox>, provides an environment for developers to use OpenShift and relevant tools without needing to configure a cluster themselves. The sandbox provides a private OpenShift environment in a shared, multi-tenant OpenShift cluster that is pre-configured with a set of developer tools.

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 can 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.



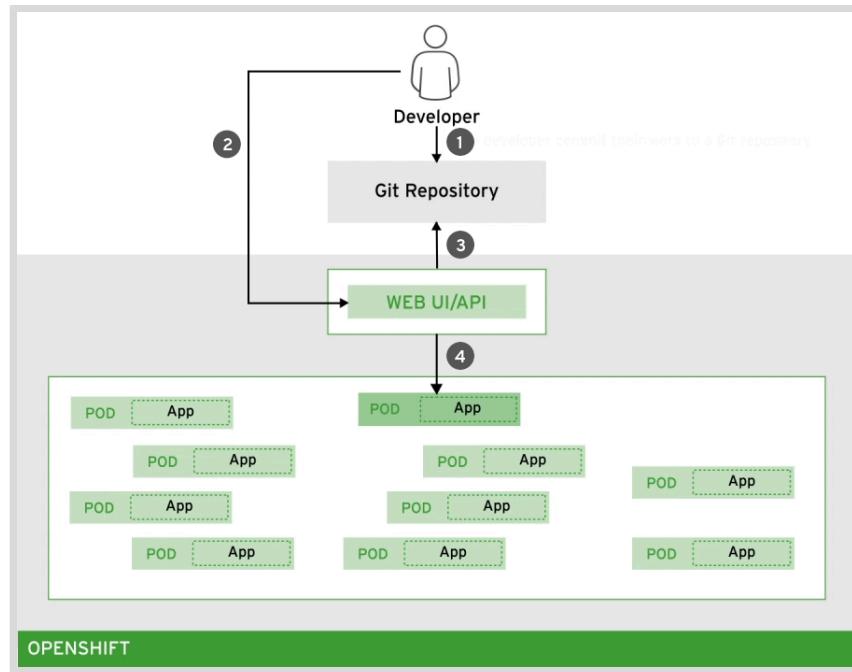
**Figure 2.1: A container in a pod**

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.

The following diagram shows an OpenShift platform, hosting several applications 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.

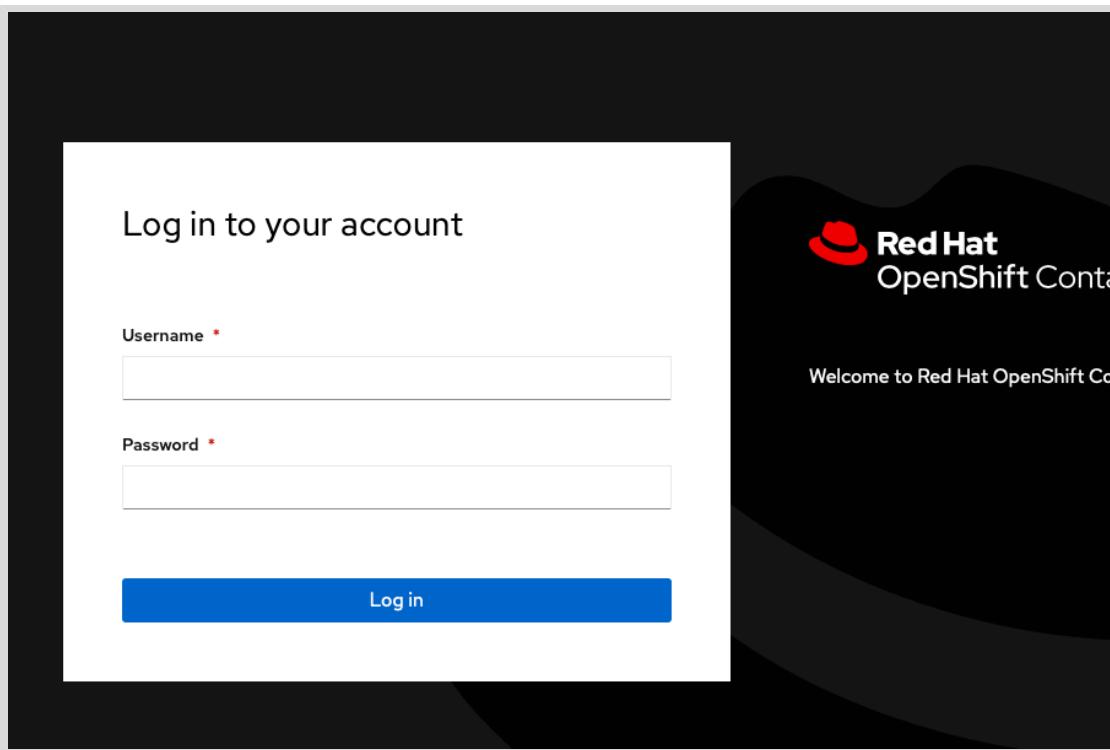
For example, the Pod resource type represents a container running on the platform. A Route resource associates a public URL to the application, so your customers can reach it from outside OpenShift.

## 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.

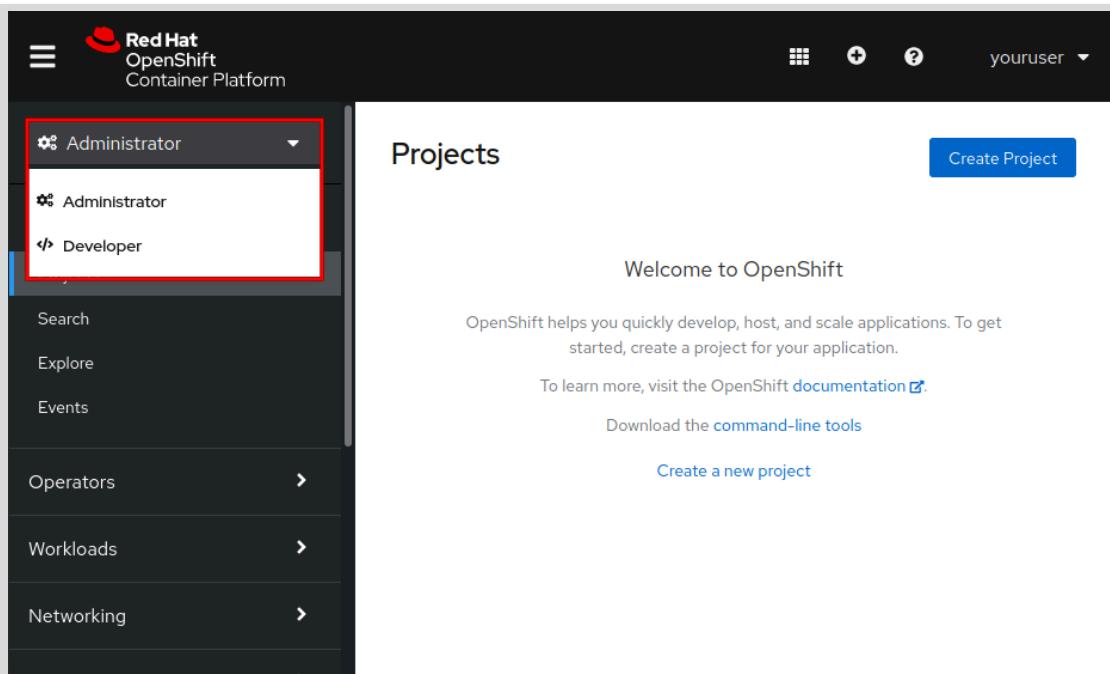
## Logging in and Accessing the Developer Perspective

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 2.2: 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.



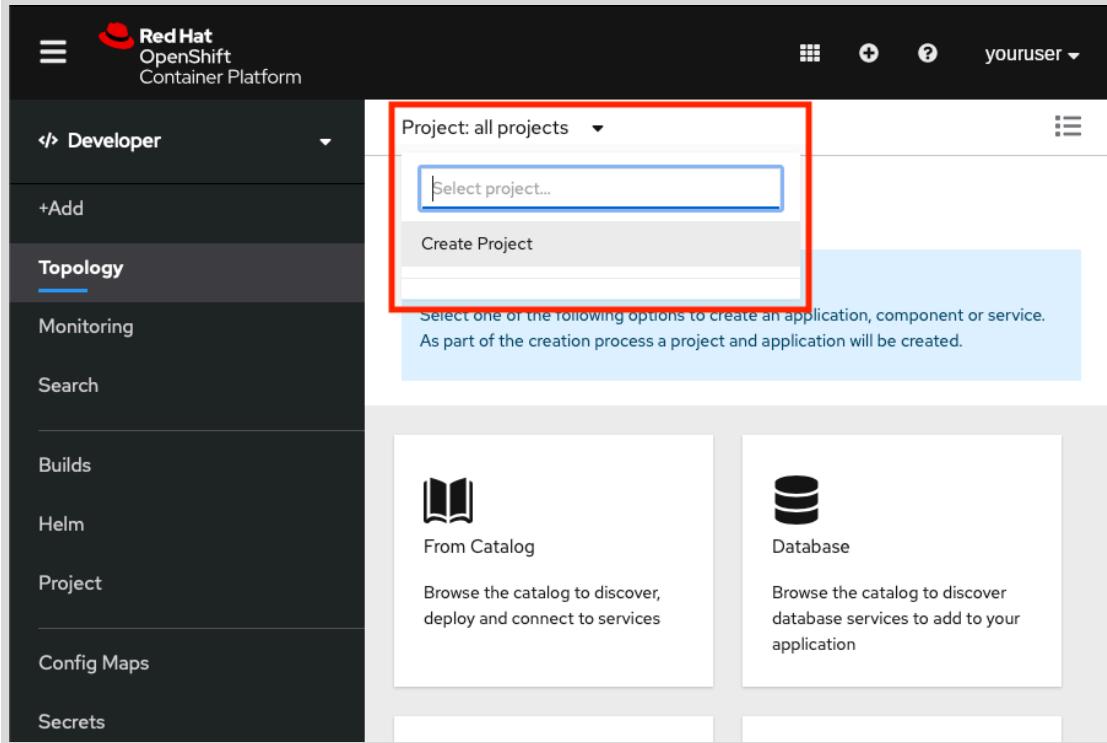
**Figure 2.3: The administrator and developer perspectives**

## 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, select **Create Project** from the Project list.

**Note**  
If you do not see the Project list, switch to the **Developer** perspective and select Topology from the menu on the left.



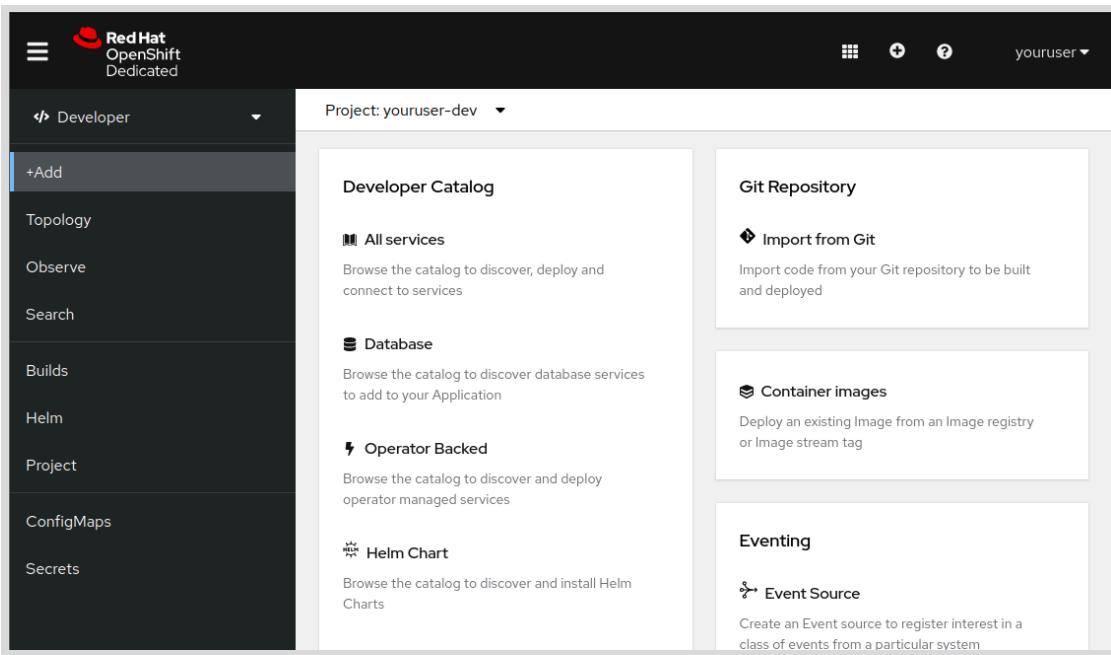
The screenshot shows the Red Hat OpenShift Container Platform web interface. The top navigation bar includes the Red Hat logo, 'Red Hat OpenShift Container Platform', and a user dropdown. On the left, a sidebar menu lists 'Developer', '+Add', 'Topology' (which is underlined in blue), 'Monitoring', 'Search', 'Builds', 'Helm', 'Project', 'Config Maps', and 'Secrets'. The main content area has a title 'Project: all projects' with a dropdown arrow. Below it is a search bar containing 'Select project...' and a 'Create Project' button. A tooltip below the search bar reads: 'Select one of the following options to create an application, component or service. As part of the creation process a project and application will be created.' At the bottom, there are two cards: 'From Catalog' (with a book icon) and 'Database' (with a database icon). The 'From Catalog' card describes: 'Browse the catalog to discover, deploy and connect to services'. The 'Database' card describes: 'Browse the catalog to discover database services to add to your application'.

**Figure 2.4: Creating a project**

**Note**  
If you use the Red Hat OpenShift Developer Sandbox, you cannot create new projects.

## 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 2.5: Selecting a method to deploy a new application in OpenShift**

With the **Developer 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.

 **Note**

You may need to select the **Template** check box under the **Type** filter to view the language templates.

The screenshot shows the Red Hat OpenShift Container Platform interface. On the left, a sidebar menu includes options like Developer, +Add, Topology, Monitoring, Search, Builds, Helm, Project, Config Maps, and Secrets. The Project section is currently selected. The main content area is titled "Project: bookstore". It displays a list of languages: Java, JavaScript, .NET, Perl, Ruby, PHP, Python, and Go. Under the PHP section, there is a card for "CakePHP + MySQL" which is described as a "Template" provided by Red Hat, Inc. It states: "An example CakePHP application with a MySQL database. For more information about using this...". There are also other cards for PHP templates.

**Figure 2.6: Available languages in the catalog**

After selecting the application language from the catalog and clicking **Instantiate Template**, OpenShift provides a form to collect the application details.

The screenshot shows the deployment form for an application. The sidebar menu is identical to Figure 2.6. The main form has several fields: "Memory Limit (MySQL)" set to "512Mi", "Volume Capacity" set to "1Gi", "Git Repository URL" set to "https://github.com/sclorg/cakephp-ex.git", and "Git Reference" and "Context Directory" both empty. Below these, there is a note: "Set this to a branch name, tag or other ref of your repository if you are not using the default branch." At the bottom, there is a field for "Application Hostname".

**Figure 2.7: Deploying an application – Git details**

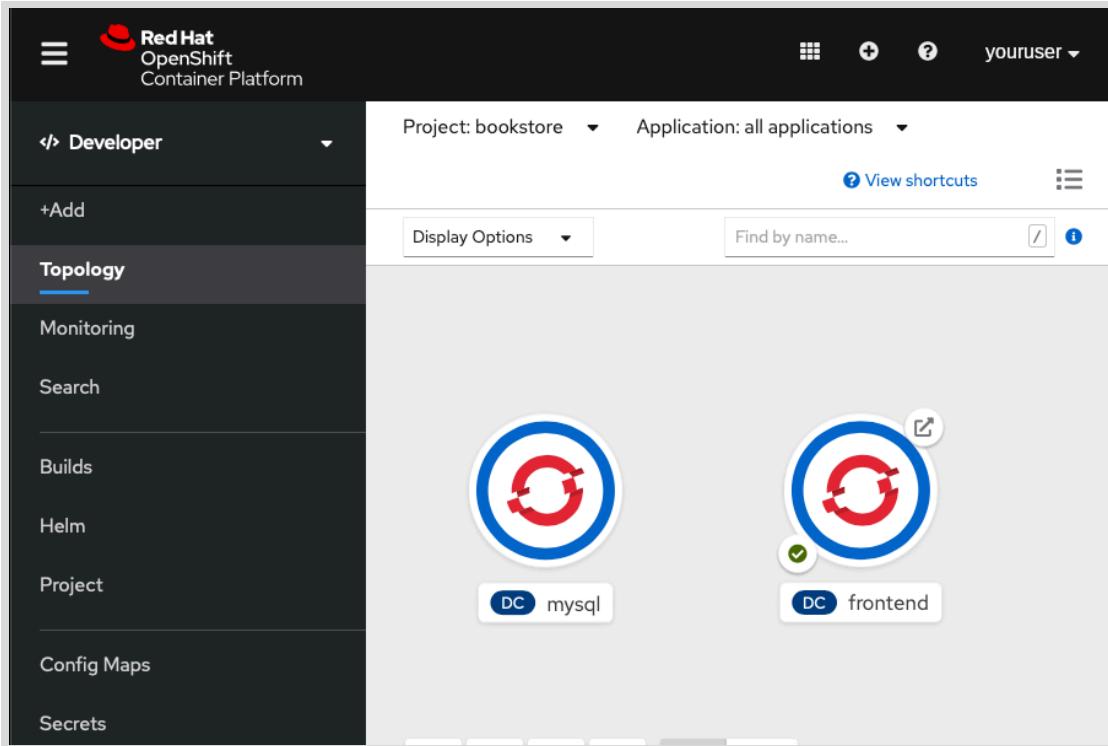
Use the fields in the first section of the form to provide Git repository details for the application.

If the source code to deploy is not in the Git **main** 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.

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.

## 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 2.8: 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.

The screenshot shows the Red Hat OpenShift Container Platform web interface. The left sidebar has a 'Developer' section with options like Topology, Monitoring, Search, Builds, Helm, Project, Config Maps, and Secrets. The main content area is titled 'Project: bookstore' and 'Application: all applications'. It displays a resource named 'DC frontend'. The 'Resources' tab is selected, showing details such as a circular icon with a red and blue swirl, a green checkmark, and a 'DC frontend' label. Below this, there are sections for 'Pods' (listing 'frontend-1-hhl9v' as running) and 'Builds' (listing a completed build). A 'View logs' button is also present.

Figure 2.9: 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.

The screenshot shows the Red Hat OpenShift Container Platform web interface. The left sidebar has a 'Developer' section with options like Topology, Monitoring, Search, Builds, Helm, Project, Config Maps, and Secrets. The main content area is titled 'Project: bookstore' and 'Routes > Route Details'. It displays a route named 'RT frontend' with status 'Accepted'. The 'Details' tab is selected, showing route details such as Name: frontend, Location: http://frontend-bookstore.apps.na3.dev.nextcle.com, Namespace: bookstore, Status: Accepted, Labels: app=cakephp-mysql-persistent, template=cakephp-mysql-persistent, template.openshift...=a47f4b51-4bb..., and Host: frontend-bookstore.apps.na3.dev.nextcle.com.

Figure 2.10: Application's route details

## Editing OpenShift Resources

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

The screenshot shows the Red Hat OpenShift Container Platform web console. The left sidebar is titled 'Developer' and includes options like '+Add', 'Topology', 'Monitoring', 'Search', 'Builds', 'Helm', 'Project', 'Config Maps', and 'Secrets'. The main content area shows a 'Routes' section with a 'Route Details' card for a route named 'frontend'. The card displays information such as 'Name: frontend', 'Namespace: bookstore', 'Labels: app=cakephp-mysql-persistent, template=cakephp-mysql-persistent, template.openshift...=a47f4b51-4bbb...', 'Location: http://frontend-bookstore.apps.na3.dev.nextcle.com', 'Status: Accepted', and 'Host: frontend-bookstore.apps.na3.dev.nextcle.com'. A red box highlights the 'Actions' dropdown menu on the right, which contains four items: 'Edit Labels', 'Edit Annotations', 'Edit Route', and 'Delete Route'.

**Figure 2.11: Available actions for a route resource**

This menu provides options to edit and delete the resource.

 **References**

**What is OpenShift**  
<https://cloud.redhat.com/learn/what-is-openshift>

For more information, refer to the Product Documentation for Red Hat OpenShift Container Platform at  
[https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform](https://access.redhat.com/documentation/en-us/openshift_container_platform)

## ► Guided Exercise

# Deploying 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:

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

## Before You Begin

To perform this exercise, ensure that:

- The `express-helloworld` Node.js application is in your GitHub D0101x-app repository from the previous activity. Your changes should be in the `devenv-versioning` branch.

This course uses the developer sandbox for Red Hat OpenShift Container Platform (RHOCUP) as a remote RHOCUP cluster. Developer Sandbox for Red Hat OpenShift is a free Kubernetes-as-a-Service platform offered by Red Hat Developers, based on Red Hat OpenShift. Developer Sandbox enables users access to a pre-created RHOCUP cluster.

Access is restricted to two namespaces. Developer Sandbox deletes pods after eight consecutive hours of running, and limits resources to 7 GB of RAM and 15 GB of persistent storage.

## Instructions

### ► 1. Create a Developer Sandbox account.

Navigate to <https://developers.redhat.com/developer-sandbox> and click **Get started in the Sandbox**.



## What is the Sandbox?

The sandbox provides you with a private OpenShift environment in a shared, multi-tenant OpenShift cluster that is pre-configured with a set of developer tools. You can easily create containers from your source code or Dockerfile, build new applications using the samples and stacks provided, add services such as databases from our templates catalog, deploy Helm charts, and much more. Discover the rich capabilities of the full developer experience on OpenShift with the sandbox.

[Get started in the Sandbox](#)

[Learn about our IDE](#)

## What can I do in the Sandbox?

The sandbox is perfect for immediate access into OpenShift, but how can you start practicing your skills in this context if you don't have a starting point yet? Well, we've taken care of that for you with a curated series of activities that you

You need a free Red Hat account to use Developer Sandbox. Log in to your Red Hat account, or if you do not have one, then click **Create one now**. Fill in the form choosing a

Personal account type, and then click **CREATE MY ACCOUNT**. You might need to accept Red Hat terms and conditions to use the Developer Program services.

When the account is ready you will be redirected back to the Developer Sandbox page. Click **Launch your Developer Sandbox for Red Hat OpenShift** to log in to Developer Sandbox.

The screenshot shows the 'Get started in the Sandbox' tab selected in the navigation bar. Below it, a large heading reads 'Start your OpenShift experience in four simple steps'. A call-to-action button labeled 'Launch your Developer Sandbox for Red Hat OpenShift' is visible. A numbered list of four steps follows:

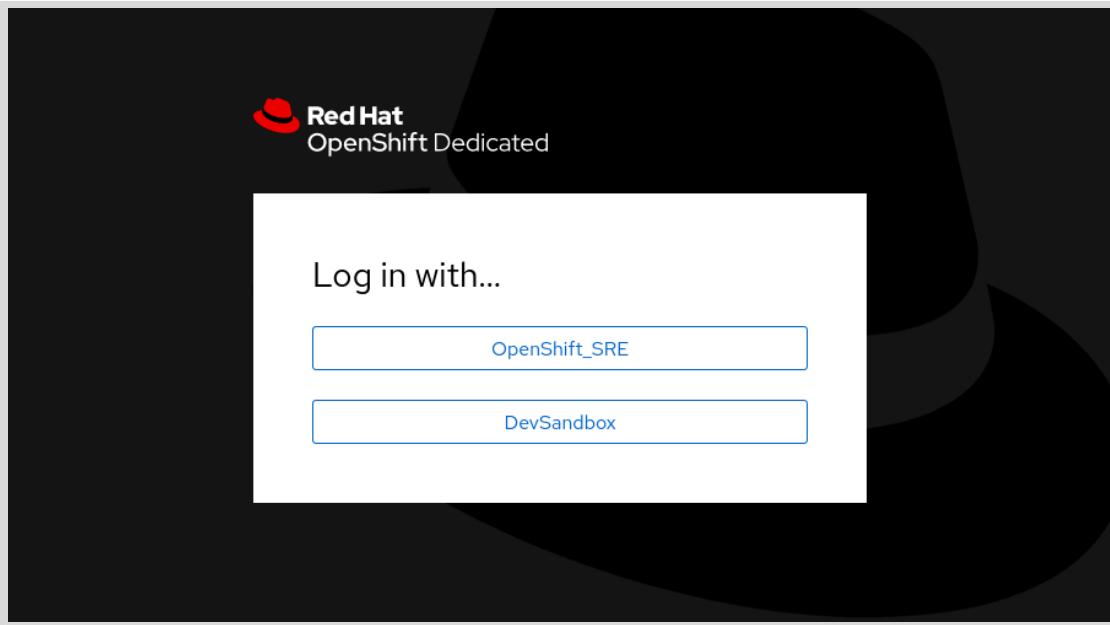
- 1 Provision your free Red Hat OpenShift development cluster
- 2 Deploy a [sample application](#) or bring your own repo
- 3 Edit code from the integrated [Eclipse Che-based cloud IDE](#)
- 4 Fully explore the Red Hat OpenShift developer experience

If you just created your account, then you might need to wait some seconds for account approval. You might need to verify your account via 2-factor authentication.

Once the account is approved and verified, Click **Start using your sandbox**. You might need to accept Red Hat terms and conditions to use the Developer Sandbox.

The screenshot shows the 'Get started in the Sandbox' tab selected in the navigation bar. Below it, a large heading reads 'Start your OpenShift experience in four simple steps'. A call-to-action button labeled 'Start using your sandbox' is visible. A message box contains the text: 'You're ready to get started! To launch your sandbox, click the button below and select DevSandbox when prompted.'

In the OpenShift log in form, click **DevSandbox** to select the authentication method.



If requested, then log in using your Red Hat account credentials.

The Red Hat OpenShift cluster and two namespaces are created for you. Only the *username-dev* and *username-stage* namespaces are available, and you can not create more.

A screenshot of the Red Hat OpenShift Dedicated interface. The top navigation bar includes the Red Hat logo, "OpenShift Dedicated", and a user dropdown "my-username". The left sidebar has tabs for "Developer" (selected), "+Add", "Topology" (highlighted with a blue bar), "Monitoring", "Search", "Builds", "Helm", and "Project". The main content area is titled "Topology" and shows a message "No resources found" with a sub-instruction: "To add content to your Project, create an Application, component or service using one of these options." Below this are two sections: "Quick Starts" (with links to "Get started with Spring", "Monitor your sample application", and "Binding your Quarkus application to Streams for Apache Kafka") and "Samples" (with a link to "Create an Application from a code sample").

- ▶ 2. 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`.
  - 2.1. Click the **Add** tab on the left side of the page and then click **All Services** in the **Developer Catalog** section.

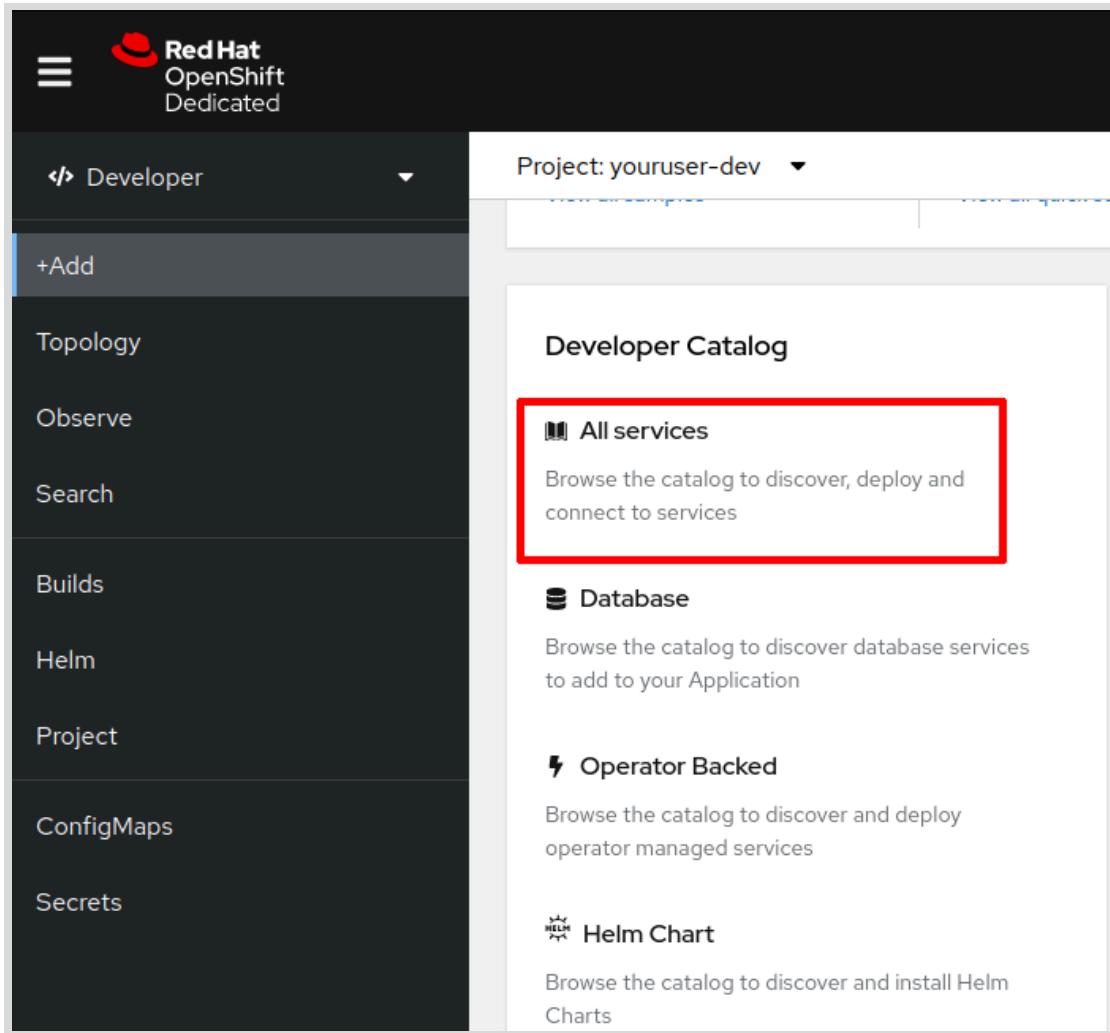


Figure 2.17: Adding a new application

- 2.2. Click Languages > JavaScript and then click the Builder Image option named Node.js. Click Create Application to enter the details of the application.



**Note**

You may need to select the Builder Image check box under the Type filter to view the builder image templates.

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

## New Application Parameters

Parameter	Value
Git Repo URL	<a href="https://github.com/yourgituser/D0101X-apps">https://github.com/yourgituser/D0101X-apps</a>
Git Reference	devenv-versioning
Context Dir	/express-helloworld
Application Name	helloworld
Name	helloworld
Resources	Deployment Config
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.

- ▶ 3. Wait for OpenShift to build and deploy your application.
  - 3.1. The web console should automatically show the Topology page. If necessary, click the **Topology** tab on the left side of the page.
  - 3.2. Wait for the application icon to change to the build complete status.

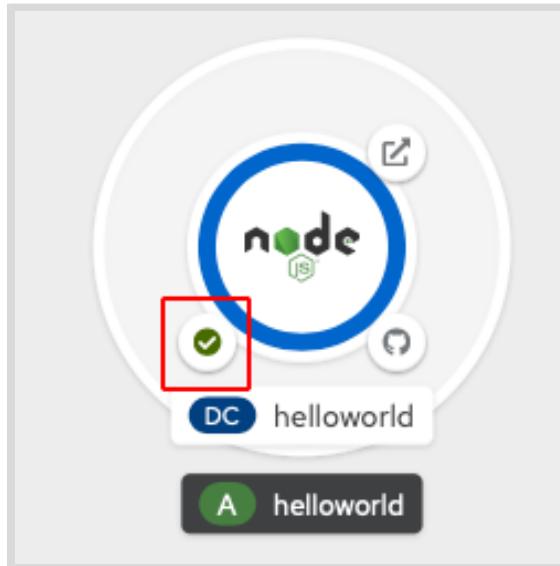


Figure 2.18: Application deployment complete

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

- ▶ 4. Review the application resources and confirm that you can access the application from the internet.
  - 4.1. Click the Node.js icon to access the application details.

- 4.2. Click the **Resources** tab to list the resources that OpenShift created during the application deployment.

The screenshot shows the Red Hat OpenShift Container Platform web interface. The top navigation bar includes the Red Hat logo, 'Red Hat OpenShift Container Platform', a user dropdown for 'youruser', and various navigation icons. The left sidebar has a 'Developer' section with a '+Add' button and links for 'Topology', 'Monitoring', 'Search', 'Builds', 'Helm', 'Project', 'Config Maps', and 'Secrets'. The main content area displays the 'Topology' view for the project 'youruser-deploy-app'. A central circular icon represents the application 'helloworld', which is built on 'node'. Below the topology are sections for 'Pods', 'Builds', 'Services', and 'Routes'. The 'Pods' section shows one pod named 'helloworld-1-ndx7h' in a 'Running' state. The 'Builds' section shows a build named 'helloworld' that is complete. The 'Services' section shows a service named 'helloworld' with a service port of 8080-tcp mapped to a pod port of 8080. The 'Routes' section shows a route named 'helloworld' with a location of <http://helloworld-youruser-deploy-app.apps.na3.dev.nextcle.com>.

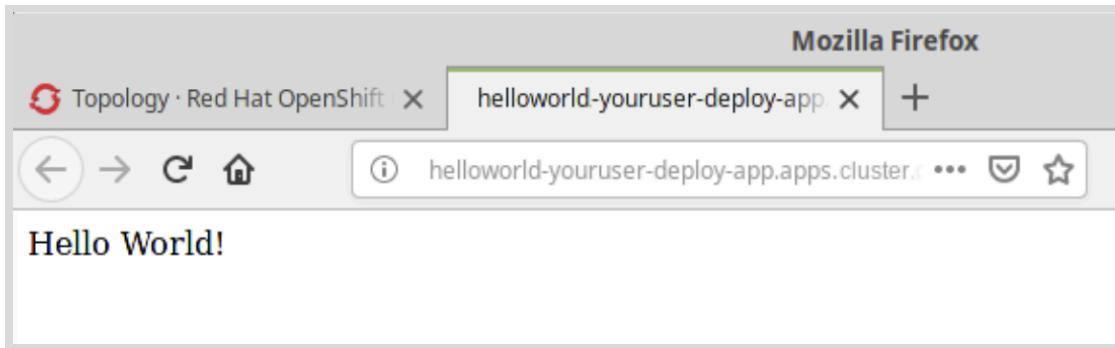
**Figure 2.19: Reviewing the application resources**

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.

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



**Figure 2.20: Accessing the helloworld application**

When done, close the browser tab.

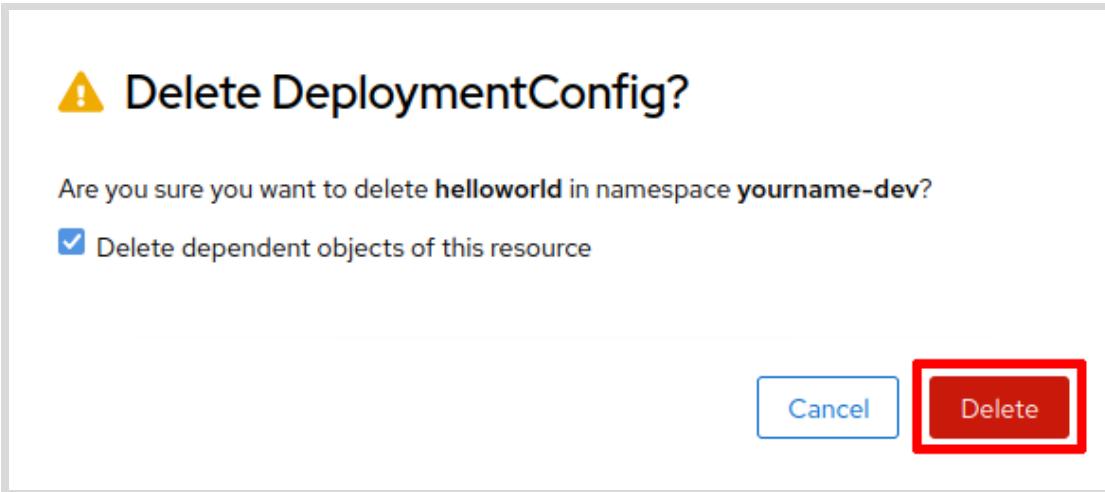
► **5.** Delete the helloworld RHOCP objects.

- 5.1. Click **Project** in the RHOCP web console. Then, click **Route**.
- 5.2. Click the context menu and select **Delete Route**.

Name	Status	Location	Service
RT helloworld	Accepted	http://helloworld-yourusername-dev.apps.sandbox-m2.ll9k.p1.openshiftapps.com	S helloworld

**Figure 2.21: Deleting the helloworld route**

- 5.3. Click **Delete** in the **Delete Route?** confirmation window.
- 5.4. Click **Project** in the RHOCP web console. Then, click **Service**. Repeat the previous steps to delete the helloworld service.
- 5.5. Click **Project** in the RHOCP web console. Then, click **DeploymentConfig**. Repeat the previous steps to delete the helloworld deployment config object.  
Ensure the **Delete dependent objects of this resource** checkbox is selected when you confirm the deletion of the object.



**Figure 2.22: Deleting the helloworld deploymentconfig resource**

- 5.6. Click **Builds** in the RHOCP web console. Then, repeat the previous steps to delete the `helloworld` buildconfig object.
- 5.7. Click **Secrets** in the RHOCP web console. Then, repeat the previous steps to delete the `helloworld-generic-webhook-secret` and `helloworld-github-webhook-secret` secrets.

## Finish

This concludes the guided exercise.

# Summary

---

In this chapter, you learned:

- OpenShift builds and runs applications in isolated pods.
- OpenShift supports building and deploying an application from source code in a Git repository.
- The web console provides a developer perspective for developers to create and manage their applications.
- A route resource associates a DNS host name to an application running in OpenShift.



## Chapter 3

# Configuring Application Builds in OpenShift

### Goal

Manage application builds in Red Hat OpenShift Container Platform.

### Objectives

- Update an application deployed on Red Hat OpenShift Container Platform.
- Add and adjust application configuration and secrets for applications deployed on Red Hat OpenShift Container Platform.
- Deploy an application that connects to a database on the Red Hat OpenShift Container Platform.

### Sections

- Configuring Application Secrets (and Guided Exercise)
- Connecting an Application to a Database (and Guided Exercise)

# 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.

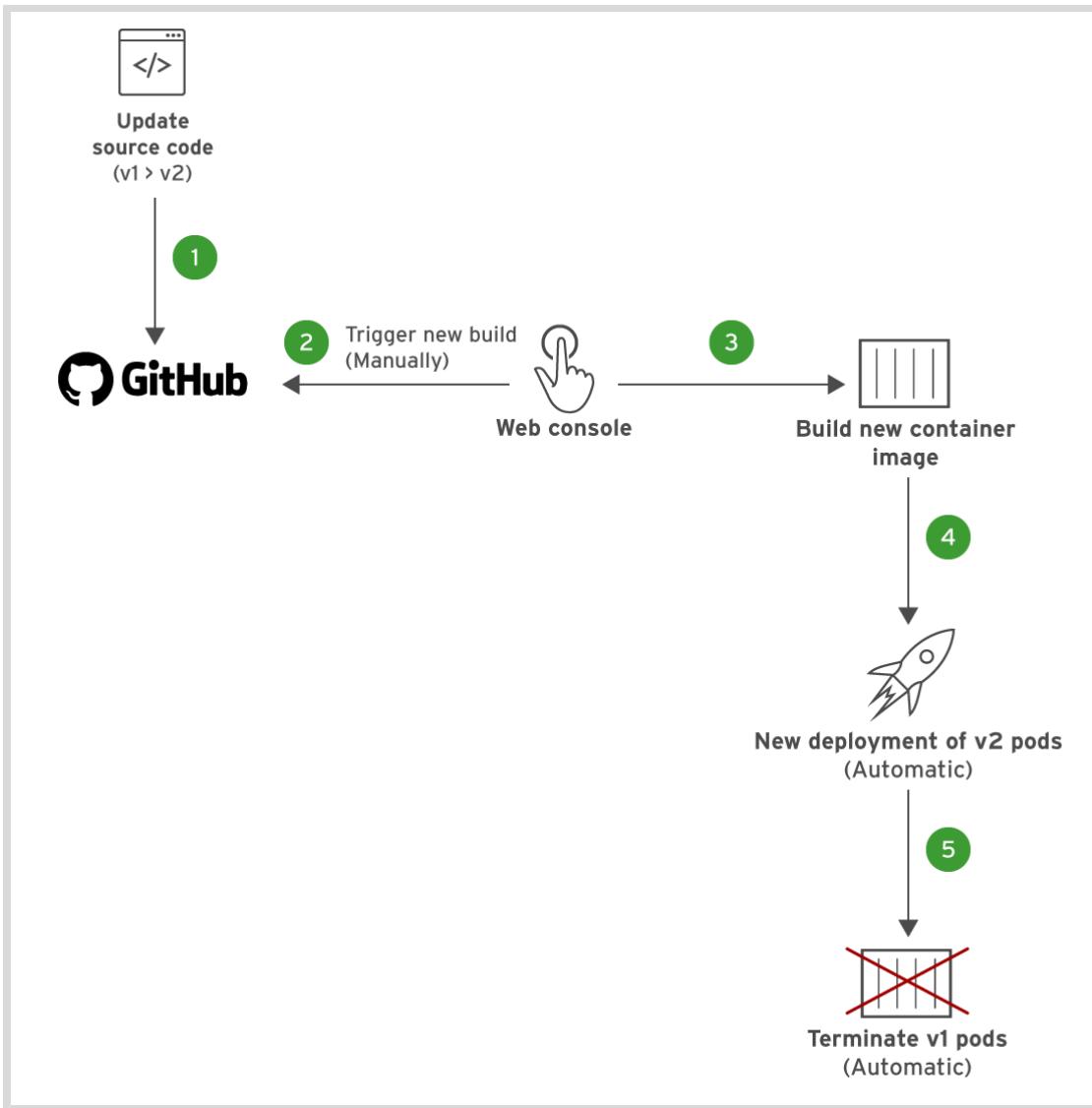
OpenShift can detect the type of application, and choose an appropriate S2I builder image to build the final application container image.

For example, if the root of the application source code tree contains a package.json file, OpenShift will select the latest Node.js S2I builder image for the build. 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:



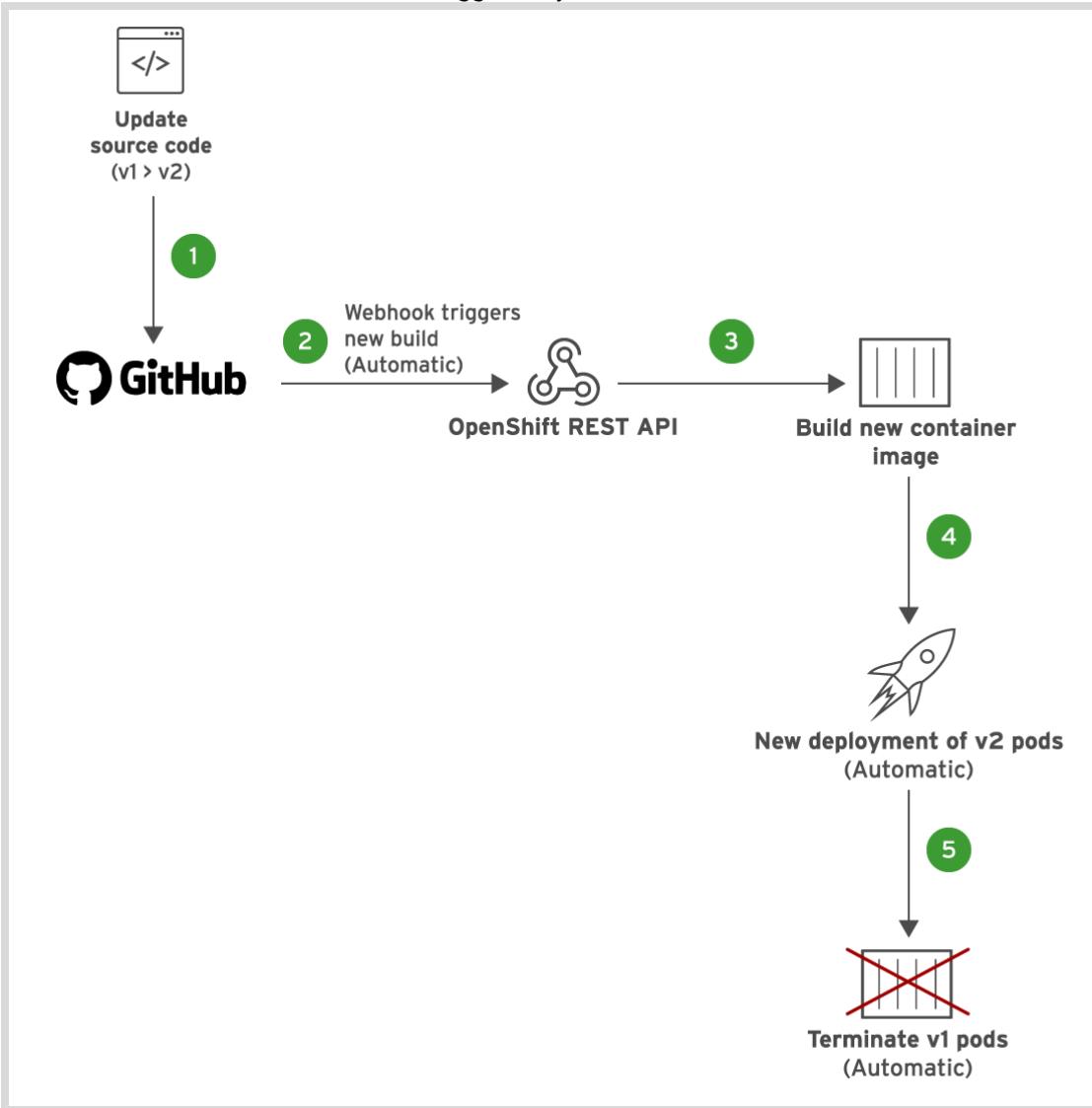
- ① Update source code for an existing application, such as from v1 to v2, and then commit the changes to GitHub.
- ② Manually trigger a new build using the OpenShift web console, or the OpenShift client command line interface (CLI).
- ③ OpenShift builds a new container image with updated code.
- ④ OpenShift 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:



- 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.



## References

For more information on OpenShift builds, refer to the **Builds** chapter in the Product Documentation for Red Hat OpenShift Container Platform at <https://docs.openshift.com/container-platform/4.6/welcome/index.html>

### Build Triggers

<https://docs.openshift.com/container-platform/4.6/builds/triggering-builds-build-hooks.html>

### Creating GitHub Webhooks

<https://developer.github.com/webhooks/creating/>

## ► 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 Red Hat OpenShift Container Platform (RHOCP) 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.

## Before You Begin

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

- A running Red Hat OpenShift Container Platform (RHOCP) cluster. See *Guided Exercise: Deploying an Application to Red Hat OpenShift Container Platform* for more information about using the developer sandbox RHOCP cluster.
- The source code for the `version` application in the `D0101X-apps` Git repository on your local system.

## Instructions

- 1. Install the Red Hat OpenShift Container Platform (RHOCP) command line interface (CLI).
- 1.1. In the RHOCP web console, click the question mark icon next to your username in the top right. Then click **Command Line Tools**.

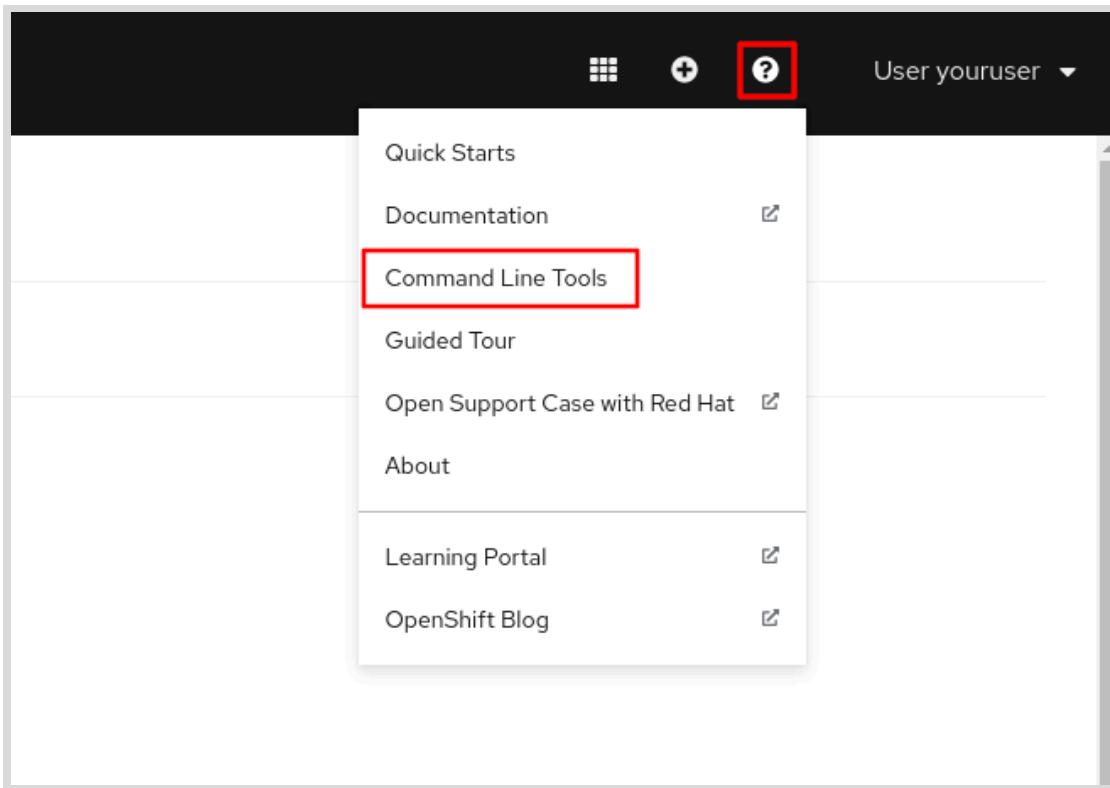


Figure 3.1: Navigate to the command line tools page

- 1.2. Download the relevant oc archive for your platform.
- 1.3. 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 and make it executable:

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

- For Windows 10 systems, decompress the downloaded archive into a directory of your choice. Then, add the full path of the directory with the oc binary to your PATH environment variable. Follow the instructions at [https://docs.microsoft.com/en-us/previous-versions/office/developer/sharepoint-2010/ee537574\(v=office.14\)#to-add-a-path-to-the-path-environment-variable](https://docs.microsoft.com/en-us/previous-versions/office/developer/sharepoint-2010/ee537574(v=office.14)#to-add-a-path-to-the-path-environment-variable) to edit the PATH environment variable.

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

```
$ oc version --client  
Client Version: 4.9.0-202110182323 ...output omitted...
```

**Note**

If you edit the PATH variable, restart VS Code in order to use the oc command from within the VS Code embedded terminal.

► 2. Deploy the `version` application to RHOCP by using the `oc` tool.

- 2.1. In the RHOCP web console, click your username at the top right, and then click **Copy Login Command**. This will open a new tab and prompt you for your username and password.
- 2.2. Once authenticated, click **Display Token** and copy the provided `oc login` command.

**Important**

Because this token is used for authentication, treat it as your RHOCP password.

- 2.3. Log in to your RHOCP account by using the copied command and notice the default project:

```
$ oc login --token=yourtoken --server=https://api.region.prod.nextcle.com:6443
Logged into "https://api.region.prod.nextcle.com:6443" as "youruser" using the
token provided.
...output omitted...
Using project "youruser-dev".
```

- 2.4. Launch the Visual Studio Code (VS Code) editor. In the Explorer view (**View > Explorer**), open the `D0101x-apps` folder in the **My Projects** workspace. The source code for the `version` application is in the `version` directory.
- 2.5. Ensure that the `D0101x-apps` repository uses the `main` branch. If you were working with another branch for a different exercise, click the current branch and then select `main` in the **Select a ref to checkout** window to switch to the `main` branch.

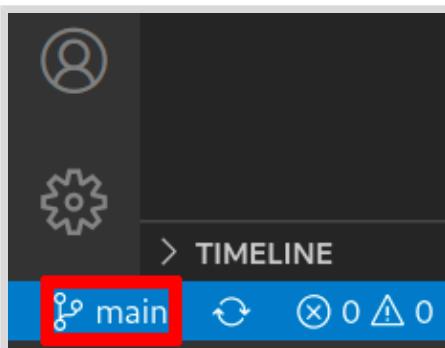


Figure 3.2: Select the main branch

**Warning**

Each exercise uses a unique branch. Always change to the `main` branch before you create a new branch. The new branch must use the `main` branch as the base.

- 2.6. Click the `main` branch in VS Code.

Select **Create new branch...** from the list of options.

- 2.7. At the prompt, enter `update-app` for the branch name.
- 2.8. Push the `update-app` branch to your `D0101x-apps` GitHub repository.  
Click the **Publish Changes** cloud icon next to the `update-app` branch name to push your local branch to your remote Git repository.  
If prompted to authorize VS Code, then click **Allow** and continue on the GitHub website.  
Alternatively, you can choose not to authorize GitHub. In such case, you will be prompted for username and password. Enter your GitHub developer token as your username, and leave the password field empty.
- 2.9. Deploy the `version` application from your `D0101x-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 split into multiple lines in this example for clarity and formatting purposes only:

```
$ oc new-app --name version \
https://github.com/yourgituser/D0101x-apps#update-app \ ❶
--context-dir version ❷
```

- ❶ The GitHub URL. Change `yourgituser` to your GitHub user. The `#update-app` indicates that RHOCP should use the `update-app` Git branch, which was created in the previous step.
- ❷ The `--context-dir` parameter indicates the directory under the `D0101x-apps` repository where the application source code is stored.

The preceding `oc` command creates and deploys the `Node.js` application. The output includes information about the latest image available for the `nodejs` image tag.



### Note

This course uses the backslash character (\) to break long commands. On Linux and macOS, you can use the line breaks.

On Windows, use the backtick character (`) to break long commands.

Alternatively, do not break long commands.

### ► 3. Test the application.

- 3.1. Open a web browser and access the RHOCP web console. Log in to the RHOCP web console by using your developer account.
- 3.2. Switch to the **Developer** perspective.
- 3.3. Click **Topology** in the navigation pane. From the **Project : list**, select the `yourusername-dev` project.
- 3.4. Click the `version` icon, and then click the **Details** tab. Verify that a single pod is running. You might have to wait for a minute or two for the application to be fully deployed.

The screenshot shows the Red Hat OpenShift Container Platform web interface. The left sidebar has a 'Developer' perspective selected, with options like '+Add', 'Topology', 'Monitoring', 'Search', 'Builds', 'Pipelines', 'Helm', 'Project', and 'Config Maps'. The main area shows 'Project: youruser-version' and 'Application: all applications'. A search bar and 'Display Options' dropdown are at the top. On the right, a 'version' deployment is detailed: it has 1 pod, is named 'version', is in the 'version' namespace, uses a RollingUpdate update strategy, and has 0 max unavailable pods. There are tabs for 'Details', 'Resources', and 'Monitoring'.

**Figure 3.3: Application deployment complete**

- 3.5. Applications created by 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
```

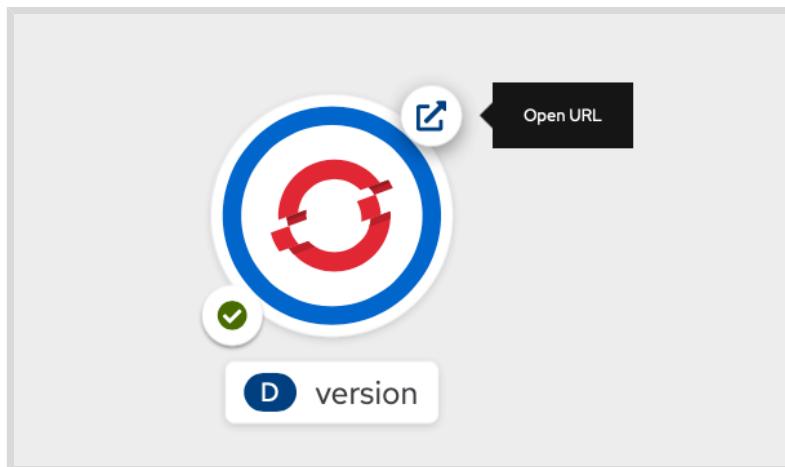


**Note**

You can also create a route by using the RHOCP web console; select the Administrator perspective, and then click **Networking > Routes > Create Route**.

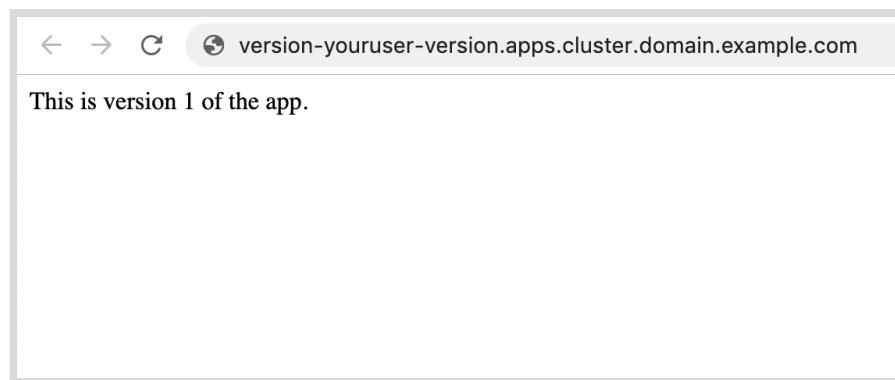
- 3.6. View the Topology page again, and observe that the `version` deployment now displays an icon to open a URL.

Click **Open URL** to view the application.



**Figure 3.4: Route URL**

- 3.7. 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.

- 4.1. In the VS Code explorer view (**View > Explorer**), click the `app.js` file in the `version` application folder of the `D0101x-apps` folder. VS Code opens an editor tab for the `app.js` file.

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

```
...output omitted...
var response;

app.get('/', function (req, res) {
    response = 'This is version 2 of the app.' + '\n';

    //send the response to the client
    res.send(response);

});
...output omitted...
```

4.3. Save the changes to the file.

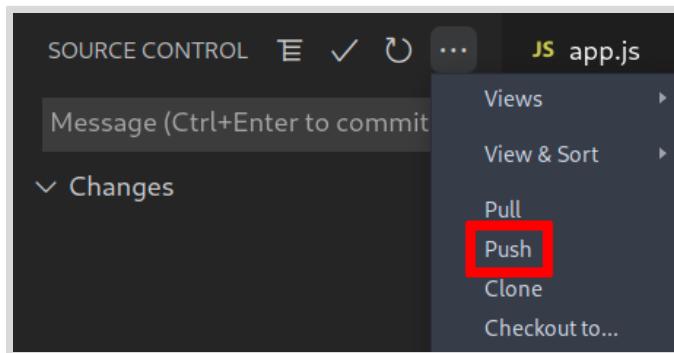
4.4. Commit your changes locally, and then push the new commit to your GitHub repository.

Switch to the **Source Control** view (**View > SCM**). Locate the entry for the `app.js` file under the **CHANGES** heading for the `D0101x-apps` directory.

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

4.5. 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.

4.6. Click the **Views and more actions > push** to publish the changes to the remote repository.



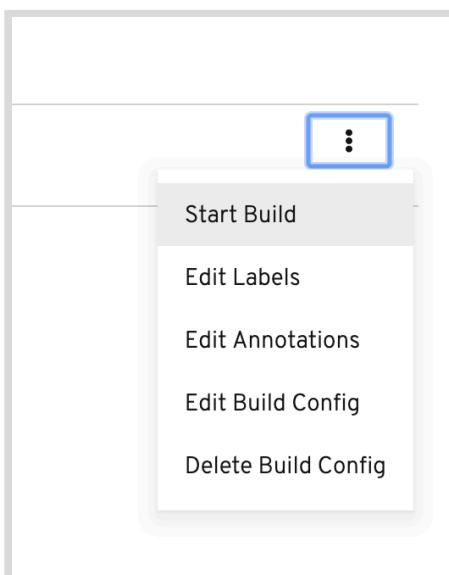
**Figure 3.6: Push changes to remote repository**

If prompted that this action will push and pull commits to and from the origin, then click **OK** to continue.

► 5. Trigger a new build manually by using the RHOCP web console.

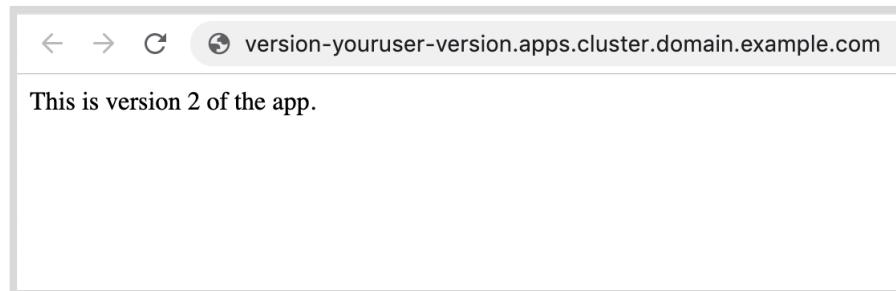
5.1. Click **Builds** in the navigation pane to view the **Build Configs** page.

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



**Figure 3.7: Start build**

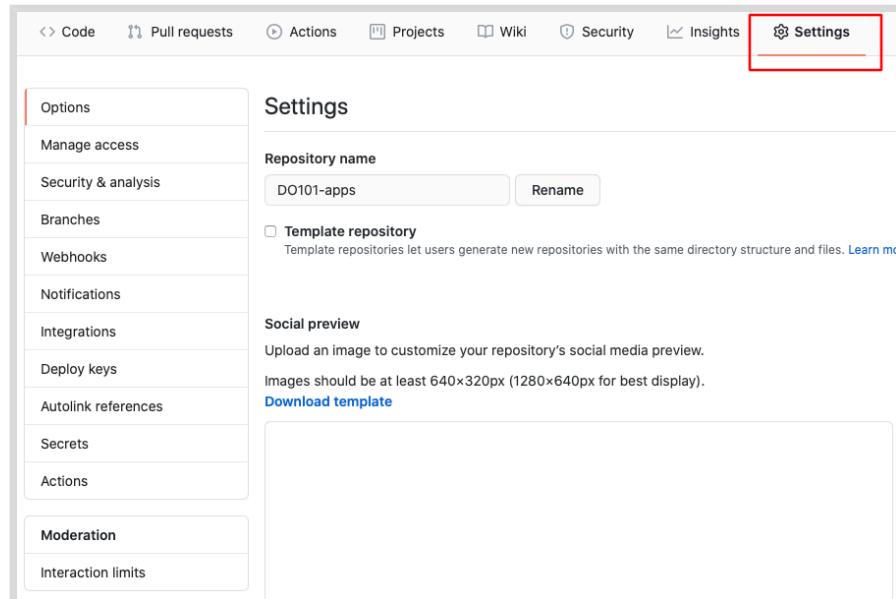
- 5.3. You should now see the details page for the build **version-2**.  
Click the **Logs** tab to see the progress of the build. A new container image with the updated source code is built and pushed to the RHOCP image registry.  
Wait until the application container image is built and deployed. You should see a **Push successful** message in the logs before proceeding to the next step.
- 5.4. 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.

- ▶ 6. Set up webhooks in GitHub to automatically trigger a new build when you push changes to the repository.
  - 6.1. Navigate to your **D0101x-apps** repository (<https://github.com/yourgituser/D0101x-apps>) on GitHub using a web browser. Log in to your GitHub account if prompted.
  - 6.2. Click the **Settings** tab to open the settings page for the repository.



**Figure 3.9: GitHub repository settings**

- 6.3. Click **Webhooks** in the left menu to open the webhooks page.

The screenshot shows the GitHub Settings interface. The top navigation bar includes links for Code, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. The 'Settings' link is underlined, indicating it is active. On the left, a sidebar menu lists various settings categories: Options, Manage access, Security & analysis, Branches (which is expanded and has 'Webhooks' highlighted with a red box), Notifications, Integrations, Deploy keys, Autolink references, Secrets, Actions, Moderation, and Interaction limits. The main content area is titled 'Webhooks' and contains the following text: 'Webhooks allow external services to be notified when certain events happen. When the specified events happen, we'll send a POST request to each of the URLs you provide. Learn more in our [Webhooks Guide](#)'. A 'Add webhook' button is located at the bottom right of this section.

Figure 3.10: Webhooks page

- 6.4. 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.
- 6.5. Get the webhook payload URL.  
Switch to the RHOCP web console in your browser, and then click **Builds** in the navigation pane.
- 6.6. Click the **version** build config to bring up the **Build Config Details** page.  
Scroll to the bottom of this page and click the **Copy URL with Secret** link next to the **GitHub** type.

Webhooks			
Type	Webhook URL	Secret	
GitHub	<a href="https://api.cluster.domain.example.com:6443/apis/build.openshift.io/v1/namespaces/youruser-version/buildconfigs/version/webhooks/&lt;secret&gt;/github">https://api.cluster.domain.example.com:6443/apis/build.openshift.io/v1/namespaces/youruser-version/buildconfigs/version/webhooks/&lt;secret&gt;/github</a>	No secret	<a href="#">Copy URL with Secret</a>
Generic	<a href="https://api.cluster.domain.example.com:6443/apis/build.openshift.io/v1/namespaces/youruser-version/buildconfigs/version/webhooks/&lt;secret&gt;/generic">https://api.cluster.domain.example.com:6443/apis/build.openshift.io/v1/namespaces/youruser-version/buildconfigs/version/webhooks/&lt;secret&gt;/generic</a>	No secret	<a href="#">Copy URL with Secret</a>

Figure 3.11: Payload URL

- 6.7. 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 request to the payload URL to verify availability, and displays a green check mark for the webhook, if it was successful.

Refresh the page to see the checkmark.

The screenshot shows the 'Webhooks' section of the OpenShift application settings. On the left, there's a sidebar with 'Options', 'Collaborators', 'Branches', 'Webhooks' (which is selected and highlighted in orange), and 'Notifications'. The main area is titled 'Webhooks' and contains a sub-section titled 'Webhooks allow external services to be notified when certain events happen. When the specified events happen, we'll send a POST request to each of the URLs you provide. Learn more in our [Webhooks Guide](#)'. Below this, there's a list of active webhooks. One webhook is listed with the URL <https://api.cluster.domain.example.com:6443/apis/build.openshift.io/v1/namespaces/youruser-...> and the event type '(push)'. There are 'Edit' and 'Delete' buttons next to the URL. At the top right of the main area, there's a button labeled 'Add webhook'.

**Figure 3.12: Webhook active**

- ▶ 7. Push changes to the Git repository to automatically trigger a new build.
  - 7.1. Edit the `D0101X-apps/version/app.js` file in VS Code to change the `response` variable to version 3, as follows:

```
...output omitted...
var response;

app.get('/', function (req, res) {
  response = 'This is version 3 of the app.' + '\n';

  //send the response to the client
  res.send(response);

});
...output omitted...
```

- 7.2. Save the changes to your file.
- 7.3. Stage the changes, commit, and then push the changes to your Git repository using steps similar those used for version 2 of the application.
- 7.4. After you commit and push the changes to the remote Git repository, RHOCP starts a new build.  
In the RHOCP web console, click **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.

Name	Namespace	Status	Created
B version-1	NS youruser-version	✓ Complete	Aug 10, 2:43 pm
B version-2	NS youruser-version	✓ Complete	Aug 10, 3:13 pm
B version-3	NS youruser-version	↻ Running	a minute ago

Figure 3.13: Build list

75. 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.

► 8. Clean up.

Delete the `youruser-version` project and remove the webhooks from GitHub.

- 8.1. Navigate to the **Settings** page of the `D0101X-apps` repository in GitHub.

Click **Webhooks** to view the webhooks page.

- 8.2. 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.

- 8.3. List all RHOCP objects you created:

```
$ oc get all --selector app=version
NAME          TYPE        CLUSTER-IP      EXTERNAL-IP    PORT(S)      AGE
service/version   ClusterIP  172.30.198.42  <none>        8080/TCP    8m44s

NAME           READY   UP-TO-DATE  AVAILABLE  AGE
deployment.apps/version  1/1     1           1          8m45s
...output omitted...
```

8.4. Delete RHOC objects with the app=version label:

```
$ oc delete all --selector app=version
service "version" deleted
deployment.apps "version" deleted
buildconfig.build.openshift.io "version" deleted
imagestream.image.openshift.io "version" deleted
route.route.openshift.io "version" deleted
```

## Finish

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.



### Warning

Data encoded in the base64 format is not cryptographically safe. A third party can decode the base64 format. Additionally, the values in the `Secret` objects can be stored in the plain-text form in the `etcd` data store.

Do not use `Secret` objects to store sensitive data like production database passwords.

The following is an example configuration map definition in YAML:

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

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

The following is an example secret definition in YAML:

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

- ① OpenShift resource type (secret)
- ② Data stored in the secret in base64 encoded format
- ③ 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.



### Note

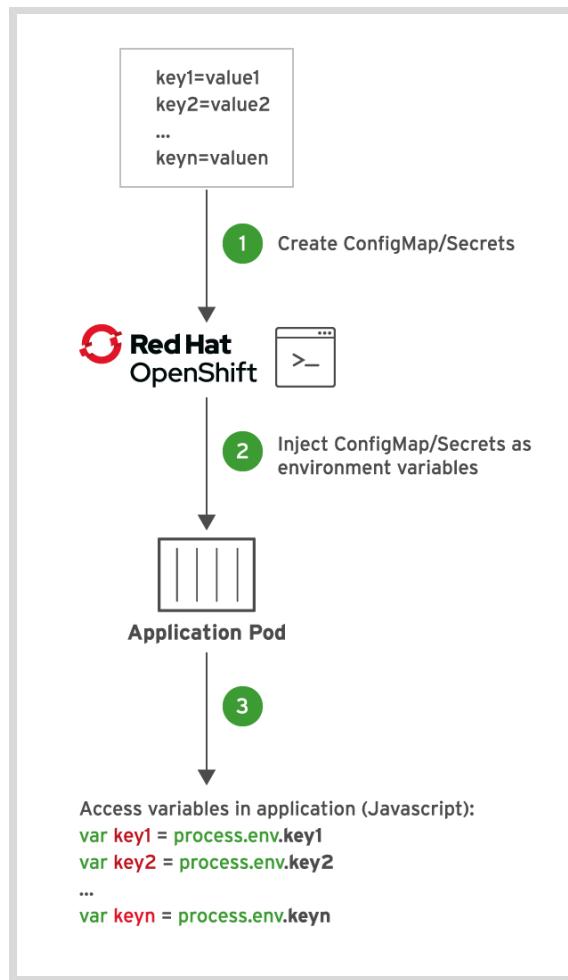
You can decode the values into plain text.

For example, on a Linux system, decode the `c2VjcmV0Cg==` value:

```
$ echo "c2VjcmV0Cg==" | base64 --decode
secret
```

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 3.15: 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 by 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, by using key based look-ups. OpenShift converts the base64 encoded data into the 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.

When the value of a secret or a configuration map changes, you must restart all pods that use the values for the new value to propagate into the pod.



## References

### OpenShift Secrets

<https://docs.openshift.com/container-platform/4.6/nodes/pods/nodes-pods-secrets.html>

### Configuring your application in OpenShift

<https://blog.openshift.com/configuring-your-application-part-1/>

## ► 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 by 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.

## Before You Begin

To perform this exercise, ensure you have access to:

- A running Red Hat OpenShift (RHOCOP) cluster. See *Guided Exercise: Deploying an Application to Red Hat OpenShift Container Platform* for more information about using the developer sandbox RHOCOP cluster.
- The source code for the `weather` application in the `D0101x-apps` Git repository on your local system.
- The `oc` command.

## Instructions

- 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.
- 1.1. Create a new account for the OpenWeatherMap API. Navigate to the website <https://openweathermap.org/> in a web browser.
  - 1.2. To create a new account, click `Sign in` and then `Create an Account`.
  - 1.3. 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`.
  - 1.4. When asked to provide details about the usage of the API, enter your name and select `Other` in the `Purpose` field.
  - 1.5. 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.

Name	Description	Price plan	Limits	Details
Weather	Current weather and forecast	Free plan	Hourly forecast: unavailable Daily forecast: unavailable Calls per minute: 60 3 hour forecast: 5 days	<a href="#">view</a>

**Figure 3.16: API services**

- 1.6. Click **API keys** to view the API keys for your account.

Key	Name	
yourapikey	Default	<input checked="" type="checkbox"/> <input type="button" value="x"/>

Create key

\* Name

**Figure 3.17: API keys**

- 1.7. 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**.

To verify that your API key is active, invoke the URL `http://api.openweathermap.org/data/2.5/weather?q=London&appid=api_key` in a browser.



### Important

It may take a few minutes for your API key to be activated before you can invoke the OpenWeatherMap API.

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:

```
{"coord":{"lon":-0.13,"lat":51.51}, "weather": ...output omitted...
{"temp": ...output omitted...
{"type":1, "id":1414, "country":"GB", "sunrise": ...output omitted...
```

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.

- 2.1. Launch the Visual Studio Code (VS Code) editor, and then open the `D0101x-apps` folder in the `My Projects` workspace. The source code for the `weather` application is in the `weather` directory.
- 2.2. Inspect the `D0101x-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": "1.4.5",
  "debug": "4.3.2",
  "dotenv": "10.0.0",
  "express": "4.17.1",
  "http-errors": "1.8.0",
  "morgan": "1.10.0",
  "node-fetch": "^2.6.1",
  "package-json": "^7.0.0",
  "pug": "3.0.2"
}
```

- 2.3. Inspect the `D0101x-apps/weather/app.js` file, which is the main entry point for the application. There is a single `Express.js` route definition called `indexRouter`:

```
...output omitted...
var cookieParser = require('cookie-parser');
var logger = require('morgan');

app.use('/', indexRouter);
var app = express();
...output omitted...
```

- 2.4. The code for the `indexRouter` route is defined in the `D0101x-apps/weather/routes/index.js` file. Open this file in VS Code. This file contains the main business logic in the application.
- 2.5. The first method handles HTTP GET requests to the '/' URL:

```
...output omitted...
router.get('/', function(req, res) {
  res.render('index', { weather: null, err: null });
});
...output omitted...
```

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



### Note

The code for the HTML form is in the D0101x-apps/weather/views/index.pug file.

- 2.6. 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:

```
...output omitted...
router.post('/get_weather', async function (req,res) {
  let city = req.body.city;
  let url = http://api.openweathermap.org/data/2.5/weather?q=${city}&units=
${UNITS}&appid=${OWM_API_KEY};

  try {
    let data = await fetch(url);
    let weather = await data.json();
  ...output omitted...
```

- 2.7. 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:

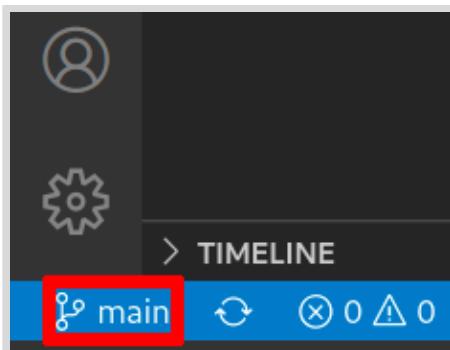
```
...output omitted...
const OWM_API_KEY = process.env.OWM_API_KEY || 'invalid_key';
...output omitted...
```

- 2.8. 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):

```
...output omitted...
const UNITS = process.env.UNITS || 'metric';
...output omitted...
```

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

- 3.1. Ensure that the D0101x-apps repository uses the `main` branch. If you were working with another branch for a different exercise, click the current branch and then select `main` in the `Select a ref to checkout` window to switch to the `main` branch.



**Figure 3.18: Select the main branch**



### Warning

Each exercise uses a unique branch. Always change to the **main** branch before you create a new branch. The new branch must use the **main** branch as the base.

- 3.2. Click the **main** branch in VS Code.

Select **Create new branch...** from the list of options.

- 3.3. At the prompt, enter **weather** for the branch name.

The Source Control view updates the **D0101x-apps** entry with the new branch name.

- 3.4. Push the **weather** branch to your **D0101x-apps** GitHub repository.

Click the **Publish Changes** cloud icon next to the **weather** branch to push your local branch to your remote Git repository.

If prompted to authorize VS Code to push commits, then click **Allow** and continue on the GitHub website.

Alternatively, you can choose not to authorize GitHub. In such case, you will be prompted for username and password. Enter your GitHub developer token as your username, and leave the password field empty.

► **4.** Create a secret to store the API key for the OpenWeatherMap service.

- 4.1. Log in to the RHOCP web console. Confirm the **Developer** perspective is active.

- 4.2. Click **Secrets** in the left menu.

- 4.3. Click **Create > Key/Value Secret** to create a new secret.

The screenshot shows the Red Hat OpenShift Container Platform web interface. The left sidebar has a 'Developer' tab selected, with 'Secrets' highlighted. The main area is titled 'Project: youruser-weather'. Below it, the 'Secrets' list is displayed with several entries. A red box highlights the 'Create' button and the 'Key/Value Secret' option in the dropdown menu.

Name	Namespace	Type
builder-dockercfg-z56zx	youruser-weather	kubernetes
builder-token-gqc8b	youruser-weather	kubernetes.io/service-account-token
builder-token-qq7c8	youruser-weather	kubernetes.io/service-account-token
default-dockercfg-wl5sh	youruser-weather	kubernetes.io/dockercfg
default-token-5lhqn	youruser-weather	kubernetes.io/service-account-token

**Figure 3.19: Create secret**

- 4.4. 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.

## Create Key/Value Secret

Key/value secrets let you inject sensitive data into your application as files or environment variables.

**Secret Name \***

Unique name of the new secret.

**Key \***

**Value**

Drag and drop file with your value here or browse to upload it.

yourapikey

[+ Add Key/Value](#)

**Figure 3.20: Secret details**

Click **Create** to create the secret.

► **5.** Create a configuration map to store the configuration related to the **weather** application.

- 5.1. 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).

Select **Config Maps** from the left menu.

- 5.2. 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.

- 5.3. Change the name to **weather-config** and replace the data field with the **UNITS: metric** key-value pair:

```

apiVersion: v1
kind: ConfigMap
metadata:
  name: weather-config
  namespace: youruser-dev
data:
  UNITS: metric

```

**Important**

Ensure that you maintain correct indentation as shown in the snippet. YAML files are indentation sensitive. Two spaces precede the `UNITS` key.

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.

- 6.1. Select **Add** in the left menu, and then click **All services** in the **Developer Catalog** section.
- 6.2. Click **Languages > JavaScript** and then click the **Builder Image** option named `Node.js`.

**Warning**

If you do not see the `Node.js` builder image, make sure the **Builder Image** filter is selected in the **Type** section.

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

- 6.3. Configure the application build according to the following table. To access the Git parameters, click **Show Advanced Git Options**.

**New Application Parameters**

Form Field	Value
Git Repo URL	<a href="https://github.com/yourgituser/D0101X-apps">https://github.com/yourgituser/D0101X-apps</a>
Git Reference	<code>weather</code>
Context Dir	<code>/weather</code>
Application Name	<code>weather</code>
Name	<code>weather</code>

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

- 6.4. Click Deployment to customize the deployment options. Reference the secret and configuration map that you created earlier.

The screenshot shows the Red Hat OpenShift Container Platform interface. The left sidebar is titled 'Developer' and includes options like '+Add', 'Topology', 'Monitoring', 'Search', 'Builds', 'Helm', 'Project', 'Config Maps', and 'Secrets'. The main content area is titled 'Select the resource type to generate' and shows two options: 'Deployment' (selected) and 'Deployment Config'. Under 'Deployment', it says 'apps/Deployment' and describes it as enabling declarative updates for Pods and ReplicaSets. Under 'Deployment Config', it says 'apps.openshift.io/DeploymentConfig' and describes it as defining the template for a pod and managing deployments. Below these is a section titled 'Advanced Options' with a checked checkbox for 'Create a route to the application', which exposes the application at a public URL. At the bottom, there is a note: 'Click on the names to access advanced options for Routing, Health Checks, Build Configuration, Deployment, Scaling, Resource Limits and Labels.' A red box highlights this note.

**Figure 3.21: Deployment for weather application**

Click Add from Config Map or Secret

The screenshot shows the same Red Hat OpenShift interface as Figure 3.21, but with a focus on environment variables. The 'Advanced Options' section has a checked checkbox for 'Create a route to the application'. Below it is a section titled 'Deployment' with a checked checkbox for 'Auto deploy when new image is available'. Under 'Environment Variables (Runtime only)', there is a table with two columns: 'NAME' and 'VALUE'. A row shows 'name' in the NAME column and 'value' in the VALUE column. Below this table are two buttons: '+ Add Value' and '+ Add from Config Map or Secret'. A red box highlights these two buttons. At the bottom, there is a note: 'Click on the names to access advanced options for Routing, Health Checks, Build Configuration, Deployment, Scaling, Resource Limits and Labels.'

**Figure 3.22: Add environment variables from config map or secret**

- 6.5. A new row is added to the **Environment Variables** table. Add a key called **OWM\_API\_KEY**. Click the **Select a resource** list and select **owm-api-secret**. Click the **Select a key** list and select **OWM\_API\_KEY**.

NAME	VALUE
name	value
OWM_API_KEY	<input type="button" value="owm-api-secret"/> <input type="button" value="OWM_API_KEY"/>

**Figure 3.23: Select secret**

- 6.6. Click **Add from Config Map or Secret** again. A third row is added to the **Environment Variables** table.  
Add a key called **UNITS**. Click the **Select a resource** list and select **weather-config**. Click the **Select a key** list and select **UNITS**.
- 6.7. Remove the first row in the **Environment Variables** table by clicking the minus (-) icon next to the empty **value** field.  
Your final **Environment Variables** table should display as follows:

The screenshot shows the Red Hat OpenShift Container Platform web interface. On the left, a sidebar menu includes options like Developer, +Add, Topology, Monitoring, Search, Builds, Helm, Project, Config Maps, and Secrets. The main panel displays deployment settings for a project named 'youruser-weather'. It shows an environment variable table with two entries:

NAME	VALUE
OWM_API_KEY	S owm-api-secret OWM_API_KEY
UNITS	CM weather-config UNITS

Below the table are buttons for '+ Add Value' and '+ Add from Config Map or Secret'. A note at the bottom says: 'Click on the names to access advanced options for Routing, Health Checks, Build Configuration, Scaling, Resource Limits and Labels.'

Figure 3.24: Final environment variables table

- 6.8. 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.
- 7.1. 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.

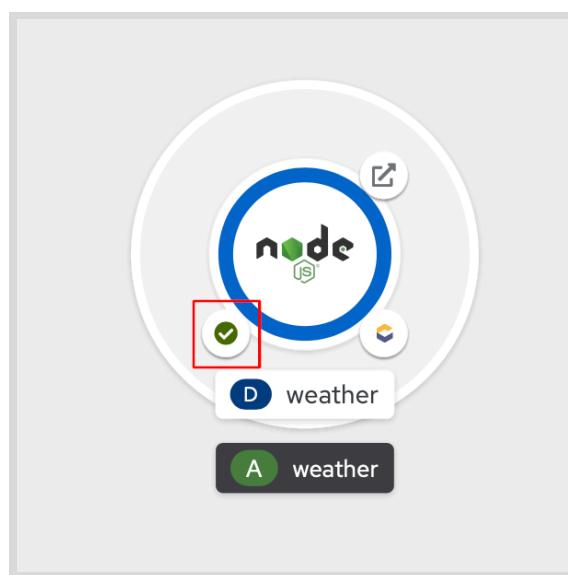
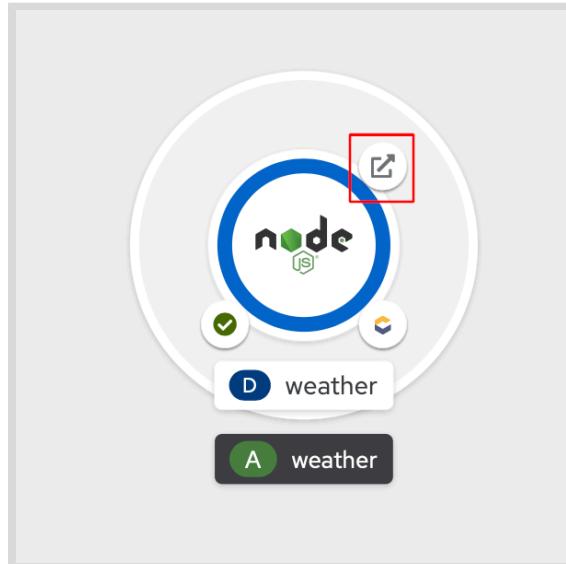


Figure 3.25: Weather app built successfully

- 7.2. Click the **Open URL** link to open the route URL for the **weather** application.



**Figure 3.26: Weather route URL**

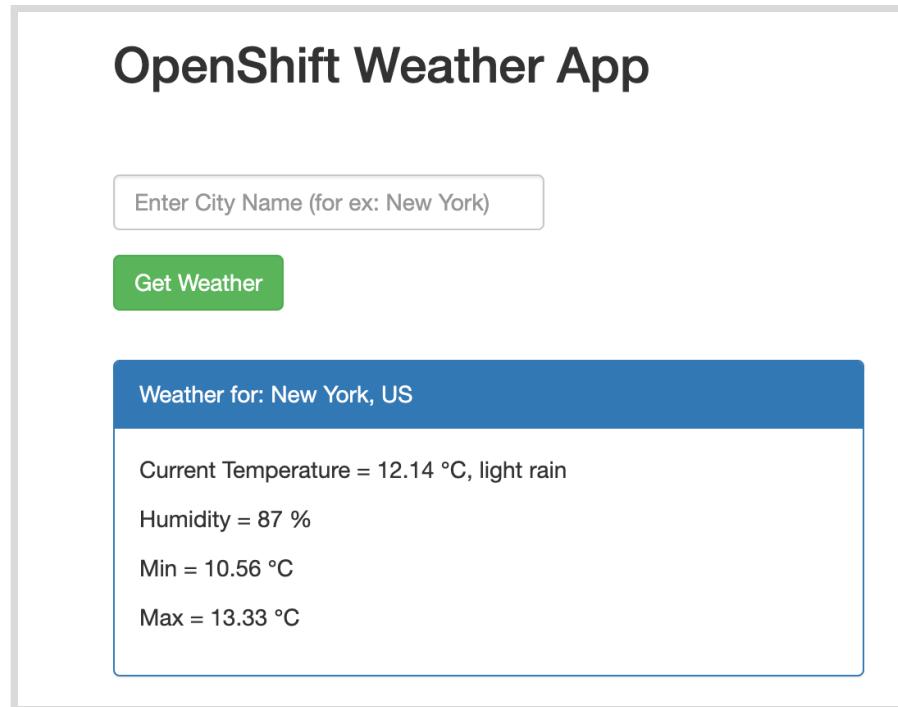
- 7.3. The home page for the **weather** application displays.



**Figure 3.27: Weather app home page**

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

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



**Figure 3.28: Weather forecast**

Close the browser tab.

- ▶ **8.** Switch back to the OpenShift web console to view the logs for the `weather` application.
  - 8.1. Click **Topology**, and then click the `weather` deployment.
  - 8.2. Click **Resources**, and then click **View Logs** for the `weather` pod to view the logs for the `weather` application.

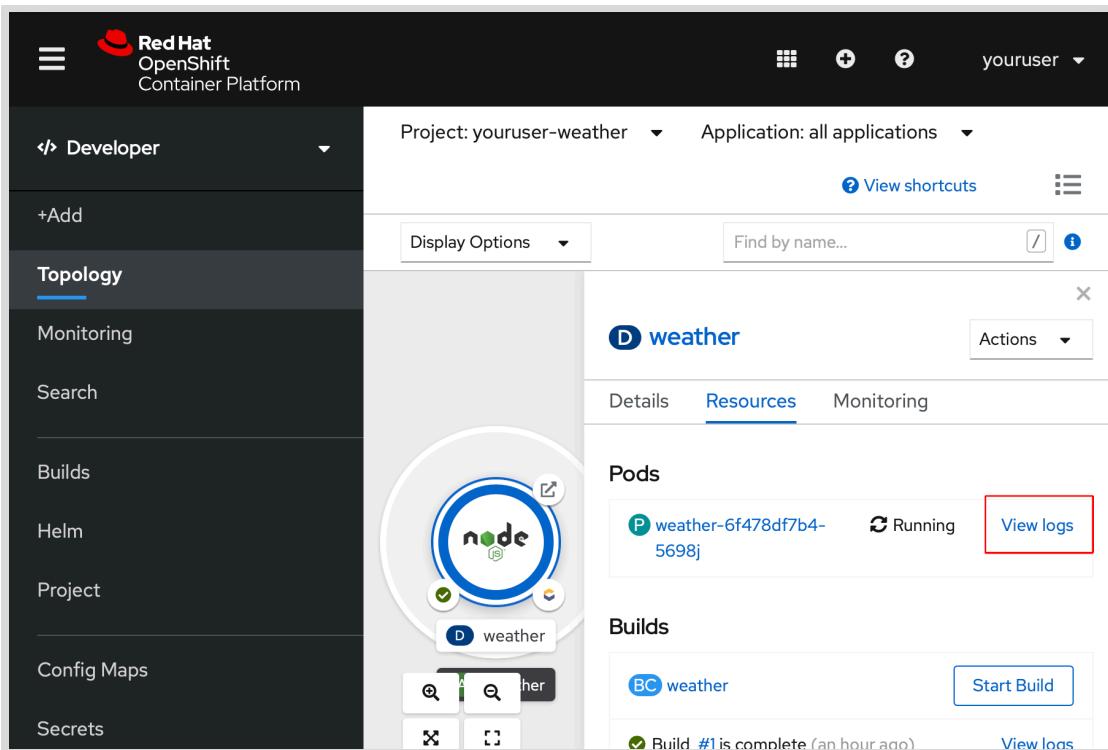


Figure 3.29: View logs for the weather application

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.

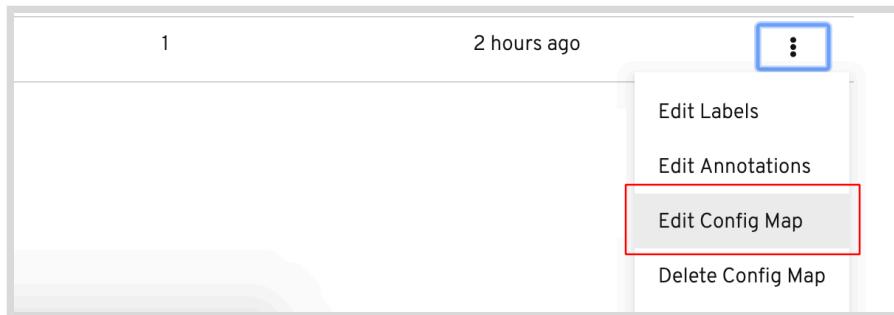
```
{
  coord: { lon: -73.99, lat: 40.73 },
  weather:
    [ { id: 500, main: 'Rain', description: 'light rain', icon: '10d' },
      { id: 701, main: 'Mist', description: 'mist', icon: '50d' },
      { id: 300,
          main: 'Drizzle',
          description: 'light intensity drizzle',
          icon: '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 },
  clouds: { all: 80 },
  sys: { type: 1, id: 5104, country: 'US', sunrise: 845, sunset: 1905 }
}
```

Figure 3.30: 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.

9.1. Select `Config Maps` from the left menu.

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



**Figure 3.31: Edit config map**

- 9.3. 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:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: weather-config
  namespace: youruser-weather
  ...output omitted...
data:
  UNITS: imperial
```

- 9.4. Click `Save` to save your changes.

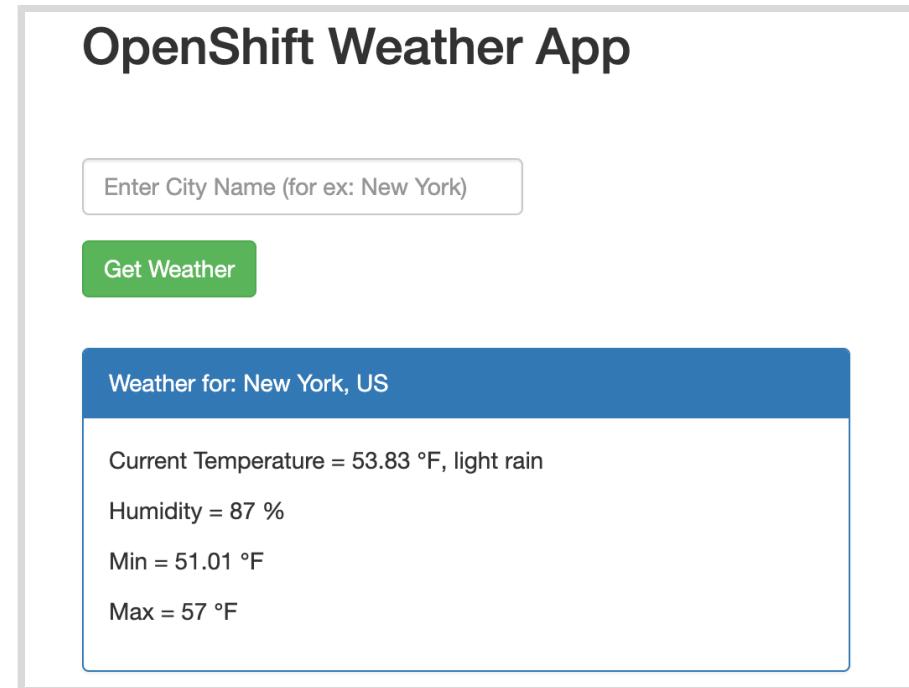
- 10. Restart the application pod and verify that the weather forecast is displayed in imperial units.

- 10.1. Click `Topology`, and then click the `weather` deployment.

Click the pod name. Then, click `Actions > Delete Pod` and confirm the pod deletion. Wait for a new pod to start.

- 10.2. Click `Topology`. Then, click the `Open URL` link to open the route URL for the `weather` application.

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



**Figure 3.32: Weather forecast in imperial units**

Close the browser tab.

► **11.** Clean up.

- 11.1. In the RHOCP web console, click your username at the top right, and then click **Copy Login Command**. This will open a new tab and prompt you for your username and password.
- 11.2. Once authenticated, click **Display Token** and copy the provided `oc login` command.
- 11.3. Log in to your RHOCP account by using the copied command and notice the default project:

```
$ oc login --token=yourtoken --server=https://api.region.prod.nextcle.com:6443
Logged into "https://api.region.prod.nextcle.com:6443" as "youruser" using the
token provided.
...output omitted...
Using project "youruser-dev".
```

- 11.4. Delete RHOCP objects with the `app=weather` label:

```
$ oc delete all --selector app=weather
pod "weather-c7867c8d9-gt6nl" deleted
service "weather" deleted
deployment.apps "weather" deleted
buildconfig.build.openshift.io "weather" deleted
imagestream.image.openshift.io "weather" deleted
route.route.openshift.io "weather" deleted
```

- 11.5. Delete the `weather-config` configmap object:

```
$ oc delete configmap weather-config
configmap "weather-config" deleted
```

- 11.6. Delete the `owm-api-secret`, `weather-generic-webhook-secret`, and `weather-github-webhook-secret` secret objects:

```
$ oc delete secret owm-api-secret weather-generic-webhook-secret
weather-github-webhook-secret
secret "weather-generic-webhook-secret" deleted
secret "weather-github-webhook-secret" deleted
secret "owm-api-secret" deleted
```

## Finish

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, by 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 by using environment variables. OpenShift injects the secret data, as environment variables, into application pods at run time.



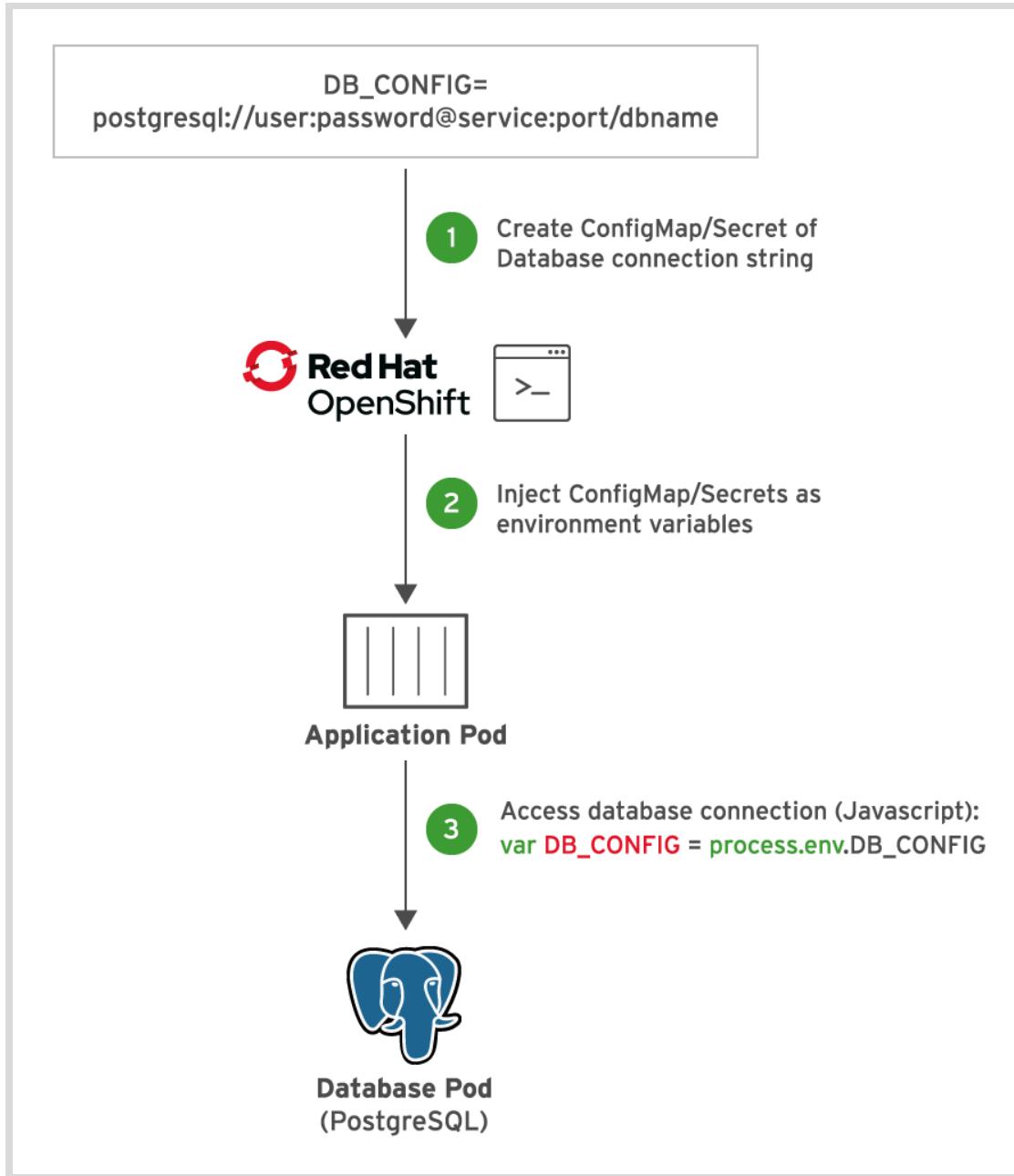
### Note

When deploying a database by 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 3.33: 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 by 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, by 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
```

The connection string has the following parts:

- **username** - The database user name for accessing the database
- **password** - The password for accessing the database
- **dbservice** - The service name for the database deployed on OpenShift
- **port** - The TCP port where the database server listens for incoming connections
- **dbname** - The database name

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

#### 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. Select **Secrets** in the left menu.
2. Click **Create > Key/Value Secret** to create a new secret.
3. Create a new key-value secret by using the database connection string as the value.

Create Key/Value Secret

Key/value secrets let you inject sensitive data into your application as files or environment variables.

**Secret Name \***

Unique name of the new secret.

**Key \***

**Value**

Drag and drop file with your value here or browse to upload it.

[+ Add Key/Value](#)

**Create** **Cancel**

**Figure 3.34: Secret details**

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

**Deployment Configuration**

Auto deploy when new image is available

Auto deploy when deployment configuration changes

**Environment Variables (Runtime only)**

NAME	VALUE
DB_CONFIG	mysecret

[+ Add Value](#) [+ Add from Config Map or Secret](#)

**Figure 3.35: Map secret to environment variable**

- 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...
```



## References

For more information, refer to the Creating an application with a database section in the OpenShift Container Platform CLI Tools guide at  
[https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.6/html-single/cli\\_tools/index#creating-an-application-with-a-database](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.6/html-single/cli_tools/index#creating-an-application-with-a-database)

### OpenShift and Databases

<https://blog.openshift.com/openshift-connecting-database-using-port-forwarding/>

### OpenShift Port Forwarding

[https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.6/html-single/nodes/index#nodes-containers-port-forwarding](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.6/html-single/nodes/index#nodes-containers-port-forwarding)

## ► Guided Exercise

# Connecting an Application 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.

## Before You Begin

To perform this exercise, ensure you have access to:

- A running Red Hat OpenShift Container Platform (RHOCOP) cluster. See *Guided Exercise: Deploying an Application to Red Hat OpenShift Container Platform* for more information about using the developer sandbox RHOCOP cluster.
- The source code for the `contacts` application in the `D0101x-apps` Git repository on your local system.
- The `oc` command.

## Instructions

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

1. Launch the Visual Studio Code (VS Code) editor, and then open the `D0101x-apps` folder in the `My Projects` workspace. The source code for the `contacts` application is in the `contacts` directory.
2. Inspect the `D0101x-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": "1.4.5",
  "debug": "4.3.2",
  "dotenv": "10.0.0",
  "express": "4.17.1",
  "http-errors": "1.8.0",
  "morgan": "1.10.0",
  "pg": "8.6.0",
```

```

    "pug": "3.0.2"
},
...output omitted...

```

- 1.3. Inspect the D0101x-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);
```

- 1.4. The code for the indexRouter route is defined in the D0101x-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.
- 1.5. 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.

```

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

```



### Note

The code for the HTML front-end is in the D0101x-apps/contacts/views/index.pug file.

- 1.6. 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...

```

- 1.7. The database connection information is configured in the D0101x-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...
```

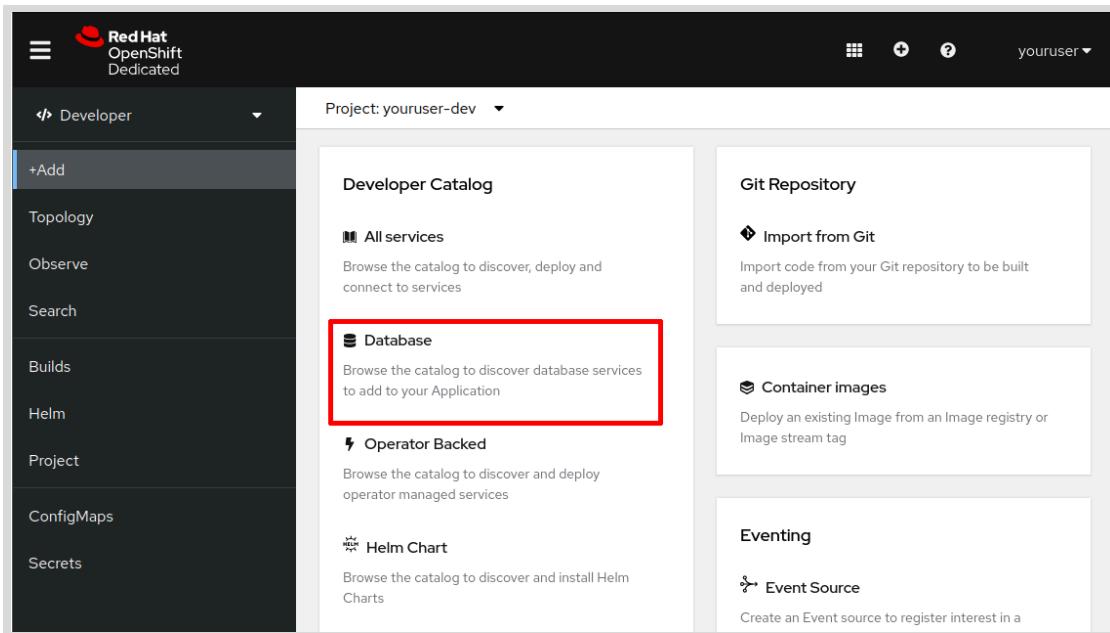
- ▶ 2. Create a new branch in your Git repository for the `contacts` application.
- 2.1. In the source control view in VS Code (`View > SCM`), ensure that the `D0101x-apps` entry under `SOURCE CONTROL REPOSITORIES` shows the `main` branch. If you are working with another branch for a different exercise, click the current branch and then select `main` in the `Select a ref to checkout` window to switch to the `main` branch.



**Warning**

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

- 2.2. Click `main` in the `D0101x-apps` entry, for `SOURCE CONTROL REPOSITORIES`.  
Select `Create new branch...` from the list of options.
  - 2.3. At the prompt, enter `contacts` for the branch name. The Source Control view updates the `D0101x-apps` entry with the new branch name.
  - 2.4. Push the `contacts` branch to your `D0101x-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. Deploy a PostgreSQL database in the `youruser-dev` project.
- 3.1. Log in to the OpenShift web console by using your developer account. Confirm the `Developer` perspective is active.
  - 3.2. Click `Add`, and then ensure that the current project is set to `youruser-dev`.
  - 3.3. Click `Database` in the `Developer Catalog` section.



**Figure 3.36: Add database**

3.4. On the **Developer Catalog** page, click **Databases > Postgres**.

3.5. Click the **PostgreSQL (Ephemeral)** option.



**Warning**

If you do not see the **PostgreSQL (Ephemeral)** template, make sure the **Template** filter is selected in the **Type** section.

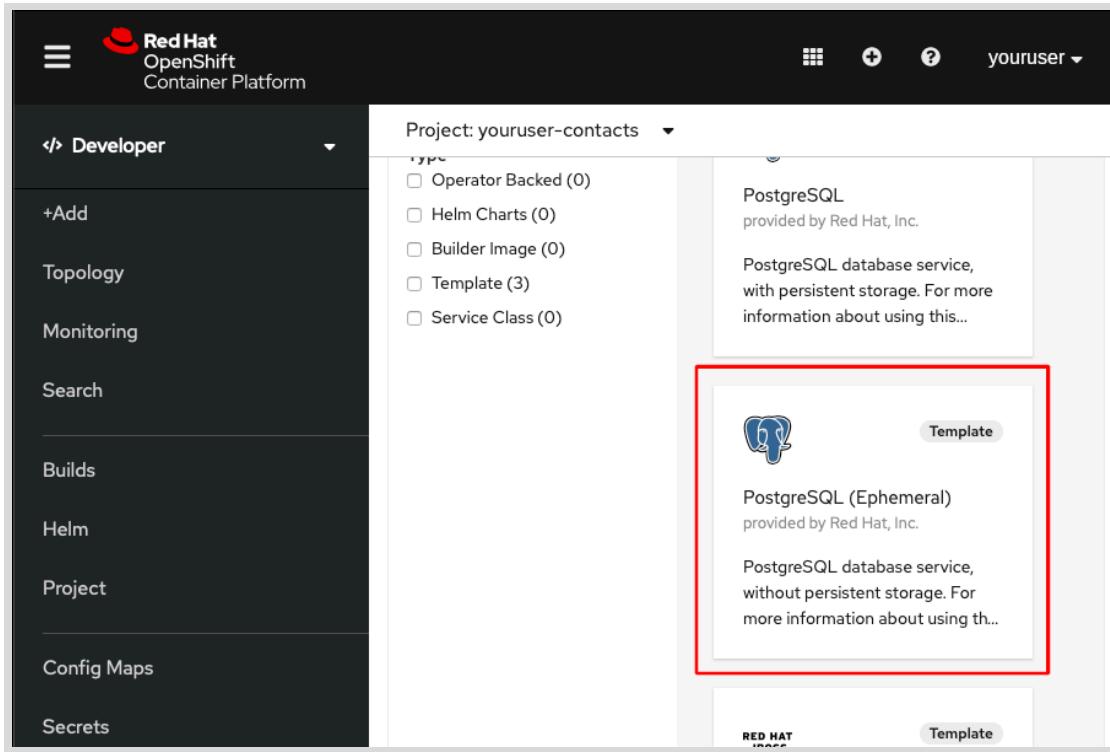


Figure 3.37: Add PostgreSQL database

- 3.6. Click **Instantiate Template**, and then complete the form according to the following table:

#### New Database Form

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

Leave all other fields at their default values.

**Namespace \***

**Memory Limit \***

Maximum amount of memory the container can use.

**Namespace**

The OpenShift Namespace where the ImageStream resides.

**Database Service Name \***

The name of the OpenShift Service exposed for the database.

**PostgreSQL Connection Username**

Username for PostgreSQL user that will be used for accessing the database.

**PostgreSQL Connection Password**

Password for the PostgreSQL connection user.

**PostgreSQL Database Name \***

Name of the PostgreSQL database accessed.

**Version of PostgreSQL Image \***

Version of PostgreSQL image to be used (10-el7, 10-el8, or latest).

**Figure 3.38: New PostgreSQL database form**

Click **Create** to start the deployment of the database.

- 3.7. Click **Topology** in the left side menu, and then click the `contactsdbs` deployment. Click **Resources**, and then verify that a single PostgreSQL database pod is running.

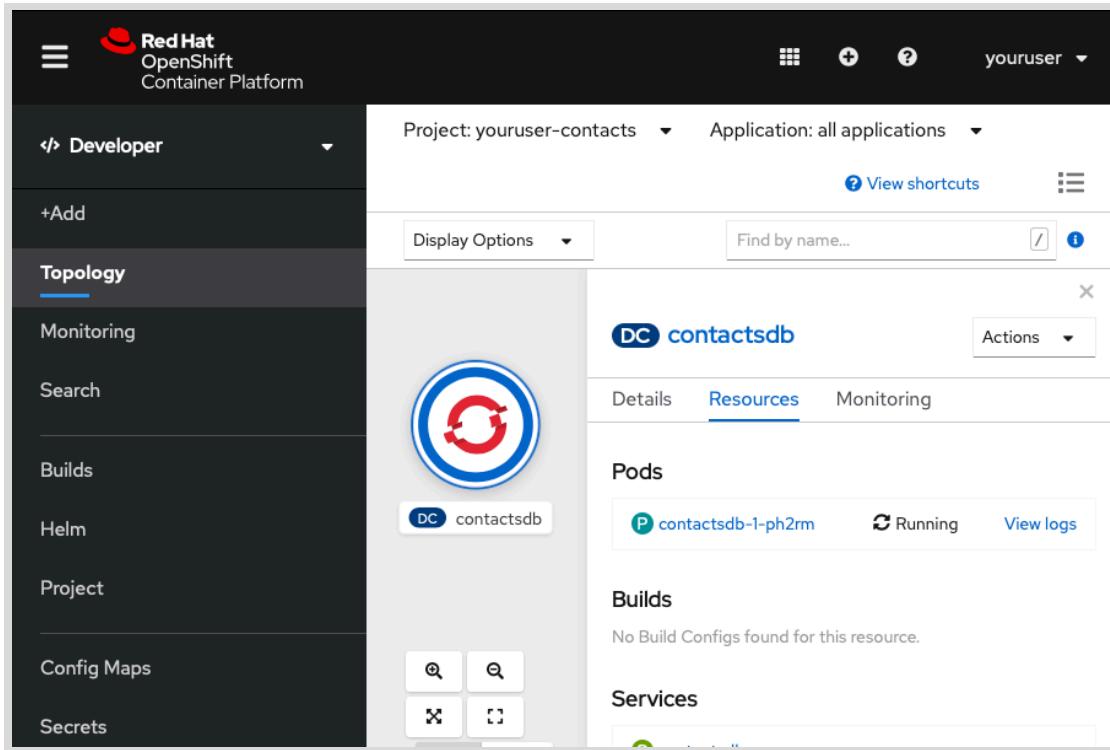


Figure 3.39: PostgreSQL database deployment

► 4. Create a secret to store the database connection information.

- 4.1. Select Secrets in the left menu.
- 4.2. Click Create > Key/Value Secret to create a new secret.

The screenshot shows the Red Hat OpenShift Container Platform web interface. The left sidebar has a 'Developer' tab selected, with 'Secrets' highlighted. The main area is titled 'Secrets' and shows a list of existing secrets. A 'Create' button with a dropdown menu is visible. The 'Key/Value Secret' option in the dropdown is highlighted with a red box.

Name	Namespace	Type
builder-dockercfg-z56zx	youruser-weather	kubernetes
builder-token-gqc8b	youruser-weather	kubernetes.io/service-account-token
builder-token-qq7c8	youruser-weather	kubernetes.io/service-account-token
default-dockercfg-wl5sh	youruser-weather	kubernetes.io/dockercfg
default-token-5lhqn	youruser-weather	kubernetes.io/service-account-token

**Figure 3.40: Create secret**

- 4.3. 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.

## Create Key/Value Secret

Key/value secrets let you inject sensitive data into your application as files or environment variables.

**Secret Name \***

Unique name of the new secret.

**Key \***

**Value**

Drag and drop file with your value here or browse to upload it.

`postgresql://contacts:contacts@contactsdb:5432/contactsdb`

[+ Add Key/Value](#)

**Create** **Cancel**

The screenshot shows the 'Create Key/Value Secret' interface. It has fields for 'Secret Name' (containing 'contactsdb-secret'), 'Key' (containing 'DB\_CONFIG'), and 'Value' (containing a PostgreSQL connection string). There are 'Create' and 'Cancel' buttons at the bottom.

**Figure 3.41: Secret details**

Recall that the DB\_CONFIG environment variable is mapped to the connectionString attribute in the D0101x-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>`

- 4.4. Click **Create** to create the secret.
- 5. Deploy the contacts application to OpenShift.
  - 5.1. Select Add, and then click All services in the Developer Catalog section.
  - 5.2. Select Languages > JavaScript, and then click the Node . js builder image option.



**Warning**

If you do not see the Node . js builder image, make sure the Builder Image filter is selected in the Type section.

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

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

### New Application Parameters

Form Field	Value
Git Repo URL	<a href="https://github.com/yourgituser/D0101x-apps">https://github.com/yourgituser/D0101x-apps</a>
Git Reference	contacts
Context Dir	/contacts
Application Name	contacts
Name	contacts

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

- 5.4. Click **Deployment** to customize the deployment options. Reference the secret that you created earlier.

The screenshot shows the Red Hat OpenShift Container Platform web interface. The top navigation bar includes the Red Hat logo, 'Red Hat OpenShift Container Platform', and a user dropdown for 'youruser'. The left sidebar has a 'Developer' tab selected, showing options like '+Add', 'Topology', 'Monitoring', 'Search', 'Builds', 'Helm', 'Project', 'Config Maps', and 'Secrets'. The main content area is titled 'Select the resource type to generate' and shows two options: 'Deployment' (selected) and 'Deployment Config'. The 'Deployment' section describes it as enabling declarative updates for Pods and ReplicaSets. The 'Deployment Config' section describes it as defining a template for a pod and managing deployments. Below this is an 'Advanced Options' section with a checked checkbox for 'Create a route to the application', which exposes the application at a public URL. At the bottom, there is a note: 'Click on the name to access advanced options for Routing, Health Checks, Build Configuration, Deployment, Scaling, Resource Limits and Labels.' A red box highlights the 'Labels' link in this note.

Figure 3.42: Deployment for contacts application

Click **Add from Config Map or Secret**

The screenshot shows the Red Hat OpenShift Container Platform web interface. The left sidebar has a 'Developer' tab selected, with options like '+Add', 'Topology', 'Monitoring', 'Search', 'Builds', 'Helm', 'Project', 'Config Maps', and 'Secrets'. The main area shows 'Project: youruser-weather' and 'Application: all applications'. Under 'Advanced Options', there's a checkbox for 'Create a route to the application' which is checked. Below it is a 'Deployment' section with a checkbox for 'Auto deploy when new image is available' which is also checked. The 'Environment Variables (Runtime only)' table has two columns: 'NAME' and 'VALUE'. A new row is being added, with 'name' in the NAME field and 'value' in the VALUE field. At the bottom of the table, there are buttons for '+ Add Value' and '+ Add from Config Map or Secret', with the latter being highlighted by a red box. A note at the bottom says: 'Click on the names to access advanced options for Routing, Health Checks, Build Configuration, Scaling, Resource Limits and Labels.'

**Figure 3.43: Add environment variables from config map or secret**

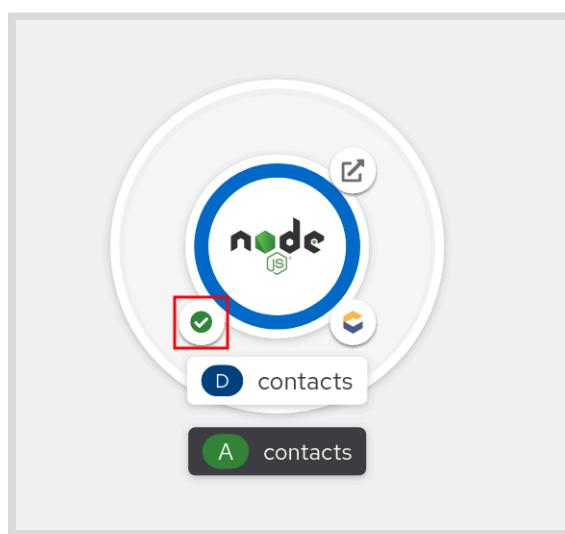
- 5.5. A new row is added to the Environment Variables table. Enter DB\_CONFIG in the NAME field in the new row. Click the Select a resource list and select contactsdb-secret.
- 5.6. Click the Select a key list and select DB\_CONFIG.
- 5.7. Remove the first row in the Environment Variables table by clicking the minus (-) icon next to the empty value field.

Your final Environment Variables table should display as follows:

The screenshot shows the Red Hat OpenShift Container Platform web interface. On the left, there's a sidebar with various developer tools: +Add, Topology, Monitoring, Search, Builds, Helm, Project, Config Maps, and Secrets. The main panel is titled 'Advanced Options' and contains a checkbox for 'Create a route to the application' which is checked. Below it, under 'Deployment', is another checked checkbox for 'Auto deploy when new image is available'. Under 'Environment Variables (Runtime only)', there's a table with one row. The 'NAME' column has 'DB\_CONFIG' and the 'VALUE' column has a dropdown menu set to 'S contactsdb-secret'. There are also '+' buttons for 'Add Value' and 'Add from Config Map or Secret'. At the bottom right of the main panel, a note says: 'Click on the names to access advanced options for Routing, Health Checks, Build Configuration, Scaling, Resource Limits and Labels.'

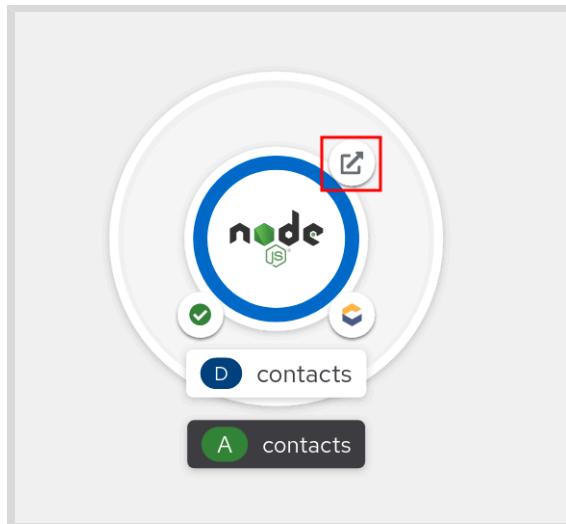
**Figure 3.44: Final environment variables table**

- 5.8. 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.
  
- ▶ **6.** Test the application.
  - 6.1. 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 3.45: Contacts app built successfully**

- 6.2. Click **Open URL** to open the route URL for the **contacts** application.



**Figure 3.46: Contacts app route URL**

6.3. The home page for the contacts application displays.

A screenshot of a web browser displaying the "Contact List" page for the "contacts" application. The URL in the address bar is "contacts-youruser-contacts.apps.cluster.domain.example.com". The page has a large title "Contact List" at the top. Below it is a "Seed Data" button. A message box states "No contacts found".

**Figure 3.47: Contacts app home page**

6.4. Click **Seed Data** to populate the database with sample contacts.

6.5. A table with five sample contacts that are fetched from the database displays.

ID	First Name	Last Name	EMail
1	Bilbo	Baggins	bilbo@theshire.com
2	Frodo	Baggins	frodo@theshire.com
3	Samwise	Gamgee	sam@theshire.com
4	Peregrin	Took	pippin@theshire.com
5	Meriadoc	Brandybuck	merry@theshire.com

Figure 3.48: Sample contacts

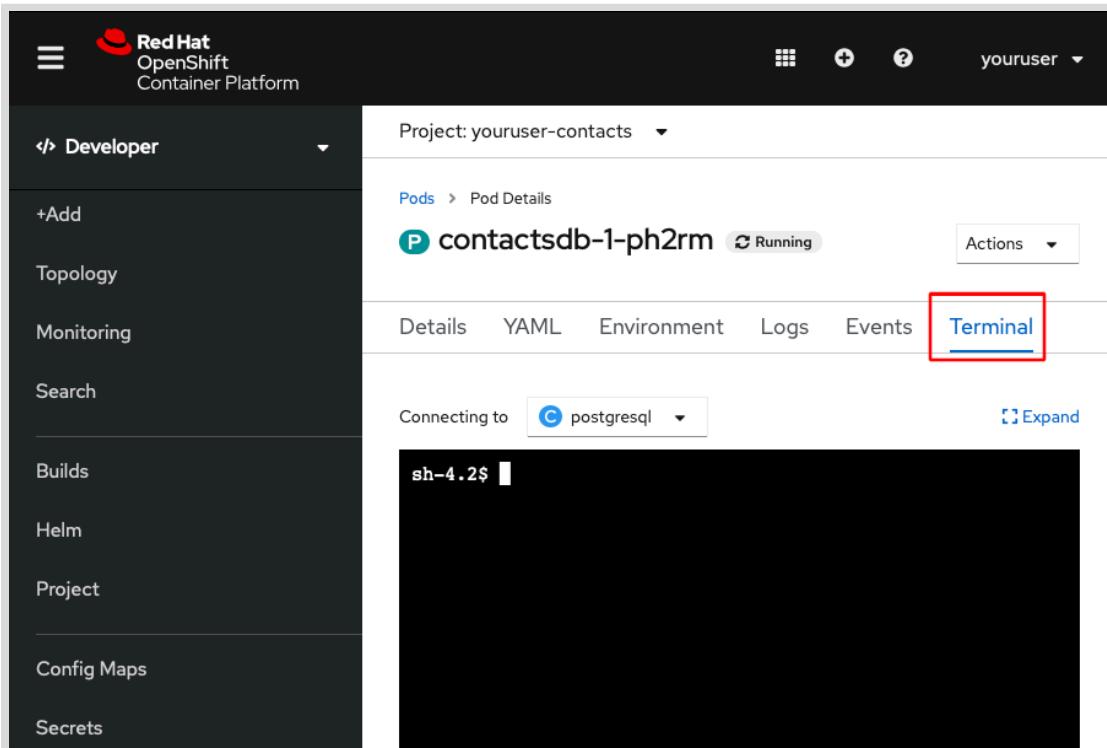
Close the browser tab.

- ▶ 7. Verify contacts data in the PostgreSQL database using the OpenShift web console.
  - 7.1. Click **Topology** in the left side menu.
  - 7.2. Click on the **contactsdb** deployment, and then click **Resources**.
  - 7.3. Click the **contactsdb** pod, listed in the **Pods** section. The pod name will differ from this example.

The screenshot shows the Red Hat OpenShift web console interface. The left sidebar has a 'Developer' section with options like '+Add', 'Topology' (which is selected), 'Monitoring', 'Search', 'Builds', 'Helm', 'Project', 'Config Maps', and 'Secrets'. The main content area shows project and application details ('Project: youruser-contacts' and 'Application: all applications'). Below that are 'Display Options' and a search bar. The 'Topology' view for the 'contactsdb' deployment is displayed, showing the 'Details', 'Resources' (selected), and 'Monitoring' tabs. Under the 'Resources' tab, the 'Pods' section lists a single pod named 'P contactsdb-1-ph2rm'. This pod is highlighted with a red box. To the right of the pod are 'Actions' (dropdown), 'Running' status, and a 'View logs' button. Below the pod list, there are sections for 'Builds' (no results) and 'Services'.

Figure 3.49: Click contactsdb pod

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



**Figure 3.50: Click terminal tab**

- 7.5. 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 `contactsdbs` database, and has five rows in it.

First, connect to the `contactsdbs` database.

```
sh-4.2$ psql -U contacts contactsdbs
```

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

```
contactsdbs=> \dt
```

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

```
contactsdbs=> select * from contacts;
```

The following output displays.

Connecting to C postgresql ▾

```

sh-4.2$ psql -U contacts contactsdb
psql (10.6)
Type "help" for help.

contactsdb=> \dt
           List of relations
 Schema |    Name    | Type  | Owner
-----+-----+-----+-----
 public | contacts | table | contacts
(1 row)

contactsdb=> select * from contacts;
 id | firstname | lastname |          email
----+-----+-----+-----+
 1 | Bilbo     | Baggins  | bilbo@theshire.com
 2 | Frodo     | Baggins  | frodo@theshire.com
 3 | Samwise   | Gamgee   | sam@theshire.com
 4 | Peregrin  | Took     | pippin@theshire.com
 5 | Meriadoc  | Brandybuck | merry@theshire.com
(5 rows)

contactsdb=>
  
```

Figure 3.51: Database output

## ▶ 8. Clean up.

- 8.1. In the RHOCP web console, click your username at the top right, and then click **Copy Login Command**. This will open a new tab and prompt you for your username and password.
- 8.2. Once authenticated, click **Display Token** and copy the provided `oc login` command.
- 8.3. Log in to your RHOCP account by using the copied command and notice the default project:

```
$ oc login --token=yourtoken --server=https://api.region.prod.nextcle.com:6443
Logged into "https://api.region.prod.nextcle.com:6443" as "youruser" using the
token provided.
...output omitted...
Using project "youruser-dev".
```

- 8.4. Delete RHOCP objects with the `app=contacts` label:

```
$ oc delete all --selector app=contacts
pod "contacts-5bdc7dbfb4-hdq48" deleted
service "contacts" deleted
deployment.apps "contacts" deleted
buildconfig.build.openshift.io "contacts" deleted
imagestream.image.openshift.io "contacts" deleted
route.route.openshift.io "contacts" deleted
```

- 8.5. Delete RHOCP objects with the template=postgresql-ephemeral-template label:

```
$ oc delete all,secret --selector template=postgresql-ephemeral-template
replicationcontroller "contactsdbs-1" deleted
service "contactsdbs" deleted
deploymentconfig.apps.openshift.io "contactsdbs" deleted
secret "contactsdbs" deleted
```

- 8.6. Delete the contactsdbs-secret, contacts-github-webhook-secret, and contacts-generic-webhook-secret secrets:

```
$ oc delete secret contactsdbs-secret contacts-generic-webhook-secret
  contacts-github-webhook-secret
secret "contactsdbs-secret" deleted
secret "contacts-generic-webhook-secret" deleted
secret "contacts-github-webhook-secret" deleted
```

## Finish

This concludes the guided exercise.

# Summary

---

- You can manually trigger new builds using the OpenShift web console or the OpenShift CLI.
- You can automatically trigger new builds using Webhooks.
- You can externalize the application configuration using secrets and configuration maps.
- Secrets are used to store sensitive information like passwords.
- Configuration maps are used to store non-sensitive plain text configuration parameters.
- Applications can store database connection credentials and other information in a secret, and integrate with databases using OpenShift services.



## Chapter 4

# Scaling Applications in OpenShift

### Goal

Scale and test an application with Red Hat OpenShift Container Platform.

### Objectives

Scale an application deployed on Red Hat OpenShift Container Platform (RHOCUP) to meet load demand.

### Sections

Scaling an Application (and Guided Exercise)

# Scaling an Application

---

## Objectives

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

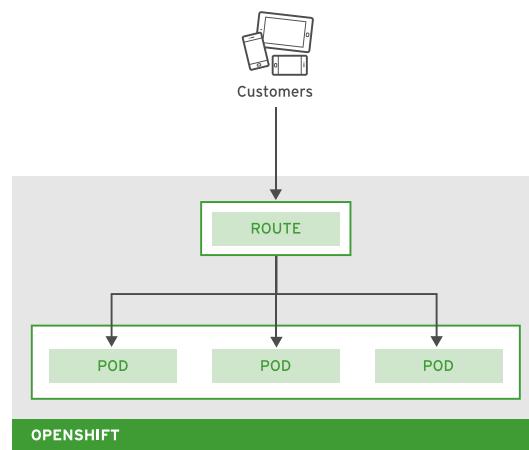
## Describing Pod Scaling

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, RHOCP automatically directs the service request to an available pod. 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. This makes it difficult to determine the number of needed pods before running the application. However, there are extra costs associated with running more pods than required when traffic is not at its peak.

RHOCP refers to the action of changing the number of pods for an application as **scaling**. **Scaling up** increases the number of pods for an application. **Scaling down** decreases the number of pods. Scaling up enables the application to handle more client requests, and scaling down provides cost savings as the load drops.

For users to reach an application from outside RHOCP, you must create a route resource that associates a public URL to the application. When scaling your application, RHOCP 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, RHOCP first deploys a new pod and then waits for the pod to be ready. Only after the new pod becomes available does RHOCP configure the route to send traffic to the new pod. When scaling down, RHOCP reconfigures the route to stop sending traffic to the pod, and then deletes the pod.

## Preparing your Application for Scaling

RHOCP 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 RHOCP. The application must be able to work correctly with multiple instances of itself.

For example, some web applications maintain user state, such as a cookie-based HTTP session. Because a user can be directed to any of the running pods, that session data must be available to all of the pods and not just the first one the user reaches. These applications might keep the session data in a central store, such as a database or in-memory shared store. They cannot only store the data in the pod local file system.

Additionally, database servers, such as MariaDB and PostgreSQL, often do not support running in multiple pods by default. One way to scale database servers is to use Operators.

## Configuring the Route Resource

By default, RHOCP tries to direct all requests from a client to the same pod. You can configure the route resource to modify this behavior.

If the round-robin algorithm is selected, RHOCP distributes the requests in a round-robin fashion between the pods. In the case of two pods, RHOCP sends the first request to the first pod, the second request to the second pod, the third request to the first pod, and so on.



### Note

Distributing requests between multiple servers is also called *load balancing*.

To edit the route details:

1. Navigate to the **Topology** page.
2. Click the application icon.
3. Click the **Overview** tab.
4. Select the route resource.
5. Modify the annotations section in the YAML file to edit the route parameters.

## Scaling an Application Manually

With the RHOCP 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.

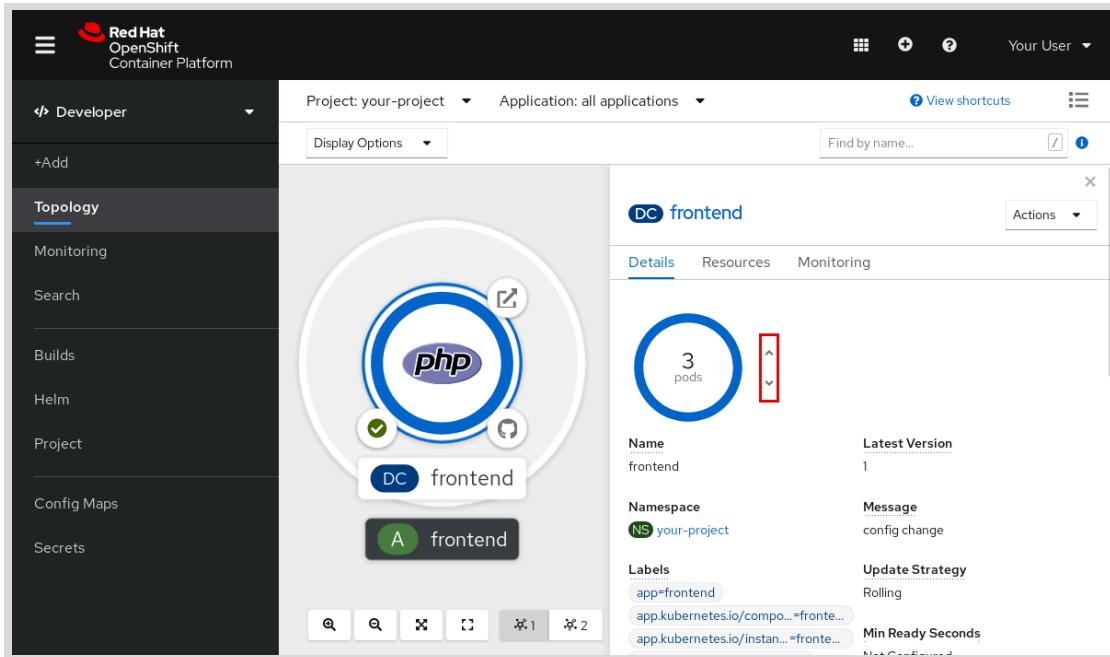


Figure 4.2: Scaling an application with the web console

## Configuring the Horizontal Pod Autoscaler

In addition to manual scaling, RHOCUP provides the **Horizontal Pod Autoscaler (HPA)** feature. An HPA automatically increases or decreases the number of pods depending on average CPU utilization. Developers can also configure an 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.

In the following example, the command enables and configures an HPA for the `frontend` deployment:

```
[student@workstation ~]$ oc autoscale deployment/frontend --min=1 --max=5
--cpu-percent=80
```

The options are as follows:

### **deployment/frontend**

Name of the application deployment.

### **--min=1**

Minimum number of pods.

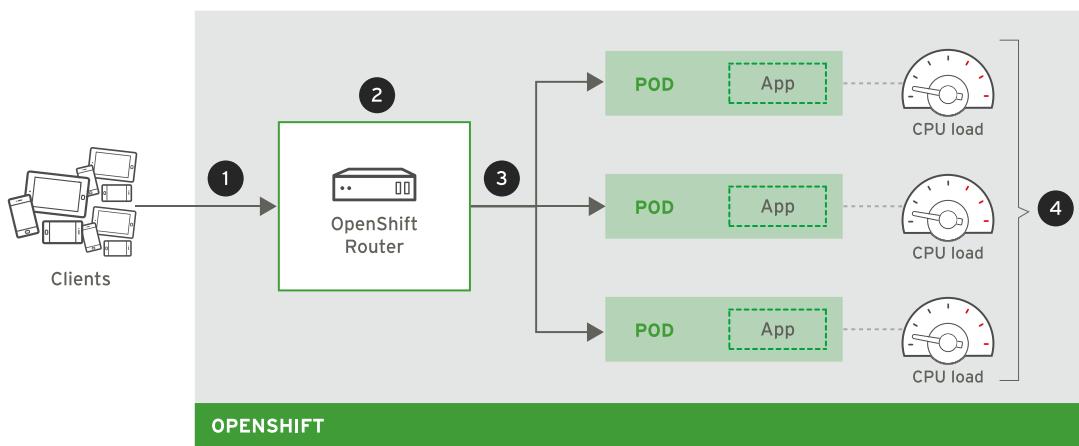
### **--max=5**

Maximum number of pods.

### **--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.



- ➊ Clients access the application.
- ➋ The RHOCP router distributes the requests to the pods.
- ➌ The application in the pods processes the request.
- ➍ When the number of clients increases, the CPU utilization increases in each pod. When the CPU utilization is above the configured value, HPA scales up by deploying a new pod.



## References

For more information, refer to the **Scaling application pods and checking builds and routes** section in the **Creating and managing applications on RHOCP** guide at  
[https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.6/html-single/applications/index#odc-scaling-application-pods-and-checking-builds-and-routes\\_viewing-application-composition-using-topology-view](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.6/html-single/applications/index#odc-scaling-application-pods-and-checking-builds-and-routes_viewing-application-composition-using-topology-view)

For more information on scaling pods, refer to the **Automatically scaling pods** section in the **Configuring and managing nodes in RHOCP** guide at  
[https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.6/html-single/nodes/index#nodes-pods-autoscaling](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.6/html-single/nodes/index#nodes-pods-autoscaling)

For more information on using custom metrics with HPA, refer to the **Exposing custom application metrics for autoscaling** section in the **Configuring and using the monitoring stack in RHOCP** guide at  
[https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.6/html-single/monitoring/index#exposing-custom-application-metrics-for-autoscaling](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.6/html-single/monitoring/index#exposing-custom-application-metrics-for-autoscaling)

## ► Guided Exercise

# Scaling an Application

In this exercise, you will configure Red Hat OpenShift Container Platform (RHOCP) to automatically scale up an application to meet load demand.

## Outcomes

You should be able to use RHOCP to:

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

## Before You Begin

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

- An RHOCP cluster. See *Guided Exercise: Deploying an Application to Red Hat OpenShift Container Platform* for more information about using the developer sandbox RHOCP cluster.
- Visual Studio Code (VS Code).
- Git.
- The `oc` command.

Additionally, you should have your fork of the `D0101x-apps` repository cloned and opened in VS Code.



### Note

Install the `oc` command-line utility by following the instructions in *Guided Exercise: Updating an Application*.

## Instructions

- 1. Using VS Code, create a new branch named `scale` in the `D0101x-apps` repository. Push the branch to your GitHub repository.

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

- 1.1. Open VS Code. Click **View > SCM** to access the source control view. Ensure that the `D0101x-apps` entry under **SOURCE CONTROL PROVIDERS** shows the `main` branch.

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

**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.

- 1.2. Click **main** in the D0101x-apps entry under SOURCE CONTROL PROVIDERS, and then select **Create new branch**. Type **scale** to name the branch.

- 1.3. Click the **Views and more actions > push** 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, to authorize VS Code to push commits, then click **Allow** and continue on the GitHub website.

Alternatively, you can choose not to authorize GitHub. In such case, you will be prompted for username and password. Enter your GitHub developer token as your username, and leave the password field empty.

- 2. 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 RHOCP.

- 2.1. Access the RHOCP web console and log in. Confirm the **Developer** perspective is active.

- 2.2. Click **Add**, and then ensure that the current project is set to **youruser-dev**.

- 2.3. Click **All services** in the **Developer Catalog** section.

- 2.4. Click **Languages > PHP**, and then **Type > Builder Image**. Make sure no other filtering options are selected. Click the PHP builder image to select it.

The screenshot shows the 'Developer Catalog' interface. On the left, there's a sidebar with 'All Items' and a 'Languages' section containing Java, JavaScript, .NET, Perl, Ruby, and PHP. 'PHP' is highlighted with a red box. Below this is a 'Type' section with checkboxes for Operator Backed (0), Helm Charts (0), Builder Image (1), Template (2), and Service Class (0). The 'Builder Image (1)' checkbox is also highlighted with a red box. On the right, there's a main panel titled 'PHP' with a 'Builder Image' badge. It says 'provided by Red Hat, Inc.' and describes building and running PHP 7.3 applications on UBI 7. A red box highlights this entire card.

Figure 4.3: Creating a PHP application

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

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

#### New Application Parameters

Parameter	Value
Git Repo URL	<a href="https://github.com/yourgituser/D0101x-apps">https://github.com/yourgituser/D0101x-apps</a>
Git Reference	scale
Context Dir	/php-scale
Application Name	scale
Name	scale

Select **Deployment** as the resource type to generate.

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

- ▶ 3. Wait for RHOCP to build and deploy your application and verify that you can access it from the internet.

- 3.1. The web console should automatically show the Topology page. Otherwise, manually click the **Topology** tab.
- 3.2. Wait for the application icon to show that the build and deployment are finished.



**Figure 4.4: The scale application finished deploying**

Once they have finished, click the **Open URL** icon to access the application.



**Figure 4.5: Accessing the scale application**

A new browser tab displays the application.



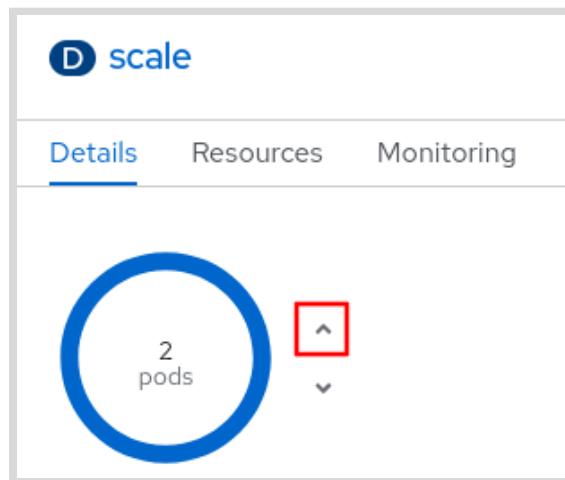
**Figure 4.6: Displaying the scale application**



**Note**

The host name and IP address displayed in the application will likely differ on your system.

- 3.3. Close the application browser tab.
- ▶ **4.** Scale the application to two pods and confirm that you can still access it.
  - 4.1. From the **Topology** page, click the PHP icon, and then click the **Details** tab.
  - 4.2. Click the up arrow to scale the application to two pods.
- ▶ **5.** By default, the load balancer redirects all requests from a particular client to the same pod. Configure the route resource to disable the affinity between clients and pods. Then, use your web browser to test that RHOCp distributes requests between the two pods.
  - 5.1. From the **Topology** page, click the PHP icon and then the **Resources** tab.
  - 5.2. Click the **scale** route resource.



**Figure 4.7: Scaling up the application**

- 4.3. 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.
- 4.4. Close the application browser tab.
- ▶ **5.** By default, the load balancer redirects all requests from a particular client to the same pod. Configure the route resource to disable the affinity between clients and pods. Then, use your web browser to test that RHOCp distributes requests between the two pods.
  - 5.1. From the **Topology** page, click the PHP icon and then the **Resources** tab.
  - 5.2. Click the **scale** route resource.

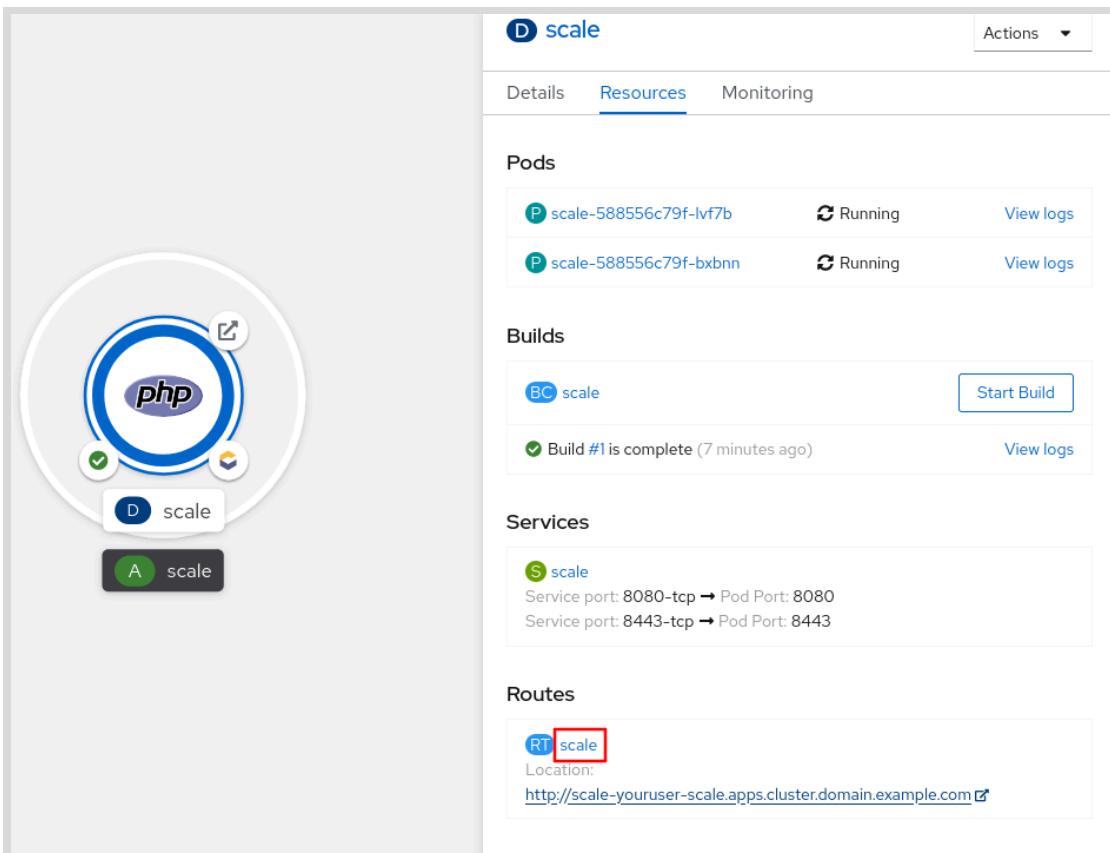


Figure 4.8: Accessing the route resource

- 5.3. Click the **YAML** tab, and then use the editor to add the following two lines in the annotations section:

```
annotations:
  openshift.io/host.generated: 'true'
  haproxy.router.openshift.io/balance: roundrobin
  haproxy.router.openshift.io/disable_cookies: 'true'
```



### Important

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

Save the changes to reconfigure the load balancer.

- 5.4. Use your web browser to confirm that RHOCP distributes requests between the two pods.  
Open the application again and refresh the page several times. Notice that the pod name and IP address alternate with every request.
- 5.5. Close the application browser tab.
6. Scale down the application to one pod.

- 6.1. From the **Topology** page, click the PHP icon, and then click the **Details** tab.
- 6.2. Click the down arrow to scale the application to one pod.
- 6.3. 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. RHOCP directs all requests to the remaining pod.

- 6.4. Close the application browser tab.

- 7. Configure RHOCP to automatically scale up your application when the CPU load is above 80%.
- 7.1. In the RHOCP web console, click your username at the top right, and then click **Copy Login Command**. This will open a new tab and prompt you for your username and password.
  - 7.2. Once authenticated, click **Display Token** and copy the provided `oc login` command.



### Important

Because this token is used for authentication, treat it as your RHOCP password.

- 7.3. Log in to your RHOCP account by using the copied command:

```
$ oc login --token=yourtoken --server=https://api.region.prod.nextcle.com:6443
Logged into "https://api.region.prod.nextcle.com:6443" as "youruser" using the
token provided.
...output omitted...
Now using project "youruser-dev" on server ...output omitted...
```

- 7.4. Configure the autoscaler for the application. Set the maximum number of pods to three and the CPU load to 80%.

```
$ oc autoscale deployment/scale --min=1 --max=3 --cpu-percent=80
horizontalpodautoscaler.autoscaling/scale autoscaled
```

- 8. To test your configuration, deploy the **stress** application to trigger the autoscaler.

ApacheBench (ab) is an Apache HTTP server benchmarking tool. The **stress** example application uses this tool to send many concurrent requests to your PHP application.

The ab command is called from the `D0101x-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. RHOCP automatically sets those variables in all the pods in the project.

- 8.1. In the RHOCP web console, click the **Add** tab, and then click **Import from Git**.
- 8.2. Complete the form according to the following table. To access the Git parameters, click **Show Advanced Git Options**.

## Stress Application Parameters

Parameter	Value
Git Repo URL	<a href="https://github.com/yourgituser/D0101X-apps">https://github.com/yourgituser/D0101X-apps</a>
Git Reference	scale
Context Dir	/stress
Application	Create application
Application Name	stress
Name	stress

Make sure Create a route to the application is **not** checked.

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

- 8.3. 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.

The screenshot shows the OpenShift web interface for the 'stress' application. On the left, there's a circular icon with a red and blue circular arrow and a green checkmark, labeled 'stress'. Below it is a button labeled 'A stress'. The main panel has a header 'D stress' with an 'Actions' dropdown. A 'Health Checks' section notes that container stress does not have health checks. Below are tabs for 'Details', 'Resources' (which is selected), and 'Monitoring'. The 'Pods' section shows one pod named 'stress-8855f578-k82rr' which is 'Running'. A 'View logs' button is highlighted with a red box. The 'Builds' section shows one build named 'BC stress' that is complete, with a 'Start Build' button and a 'View logs' button. The 'Services' section shows a service named 'stress' with port 8080 mapped to pod port 8080. The 'Routes' section indicates no routes are found.

Figure 4.9: Accessing the logs of the stress pod

Notice that ApacheBench is running:

```
This is ApacheBench, Version 2.3 <$Revision: 1826891 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 172.30.199.15 (be patient)
```

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

- ▶ 9. Inspect the scale application and confirm that the number of pods automatically increases to three.

9.1. From the Topology page, click the PHP icon, and then click the Details tab.

9.2. Notice that the number of pods application increases to three.



**Note**

It might take a few minutes for the pods to increase.

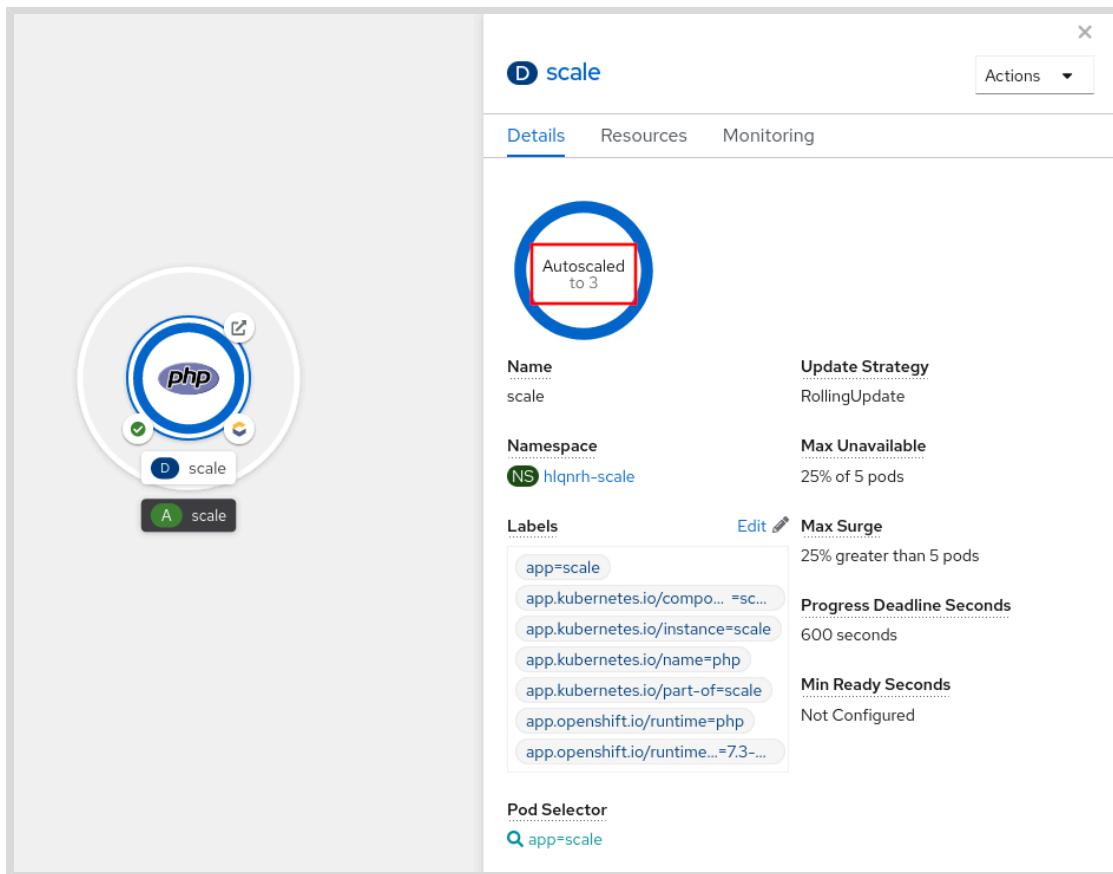


Figure 4.10: Autoscaling the application

- 10. Stop the **stress** application by manually scaling it down to zero pods. Notice that the **scale** application automatically scales down to one pod.
- 10.1. From the Topology page, click the **stress** icon, and then click the **Details** tab.
  - 10.2. Click the down arrow to scale the application to zero pods.
  - 10.3. Click the PHP icon, and then click the **Details** tab. It may take up to 10 minutes for the number of pods in the **scale** application to scale down to one.
- 11. Clean up.

- 11.1. RHOCP objects with the **app=stress** label:

```
$ oc delete all --selector app=stress
service "stress" deleted
deployment.apps "stress" deleted
buildconfig.build.openshift.io "stress" deleted
imagestream.image.openshift.io "stress" deleted
```

- 11.2. RHOCP objects with the **app=scale** label:

```
$ oc delete all --selector app=scale
pod "scale-9fd9d895c-h8wdm" deleted
service "scale" deleted
deployment.apps "scale" deleted
buildconfig.build.openshift.io "scale" deleted
imagestream.image.openshift.io "scale" deleted
route.route.openshift.io "scale" deleted
```

11.3. Delete the horizontal pod autoscaler object:

```
$ oc delete hpa scale
horizontalpodautoscaler.autoscaling "scale" deleted
```

11.4. Delete the `scale-generic-webhook-secret` and `scale-github-webhook-secret` secrets:

```
$ oc delete secret scale-generic-webhook-secret scale-github-webhook-secret
secret "scale-generic-webhook-secret" deleted
secret "scale-github-webhook-secret" deleted
```

11.5. Delete the `stress-generic-webhook-secret` and `stress-github-webhook-secret` secrets:

```
$ oc delete secret stress-generic-webhook-secret stress-github-webhook-secret
secret "stress-generic-webhook-secret" deleted
secret "stress-github-webhook-secret" deleted
```

## Finish

This concludes the guided exercise.

# Summary

---

In this chapter, you learned:

- Red Hat OpenShift Container Platform adapts your application to user demand by increasing or decreasing the number of pods.
- OpenShift automatically configures the route resource to load balance the user requests between pods.
- Use the OpenShift web console to scale applications manually.
- When configured, the Horizontal Pod Autoscaler (HPA) automatically scales applications based on CPU utilization.



## Chapter 5

# Troubleshooting Applications in OpenShift

### Goal

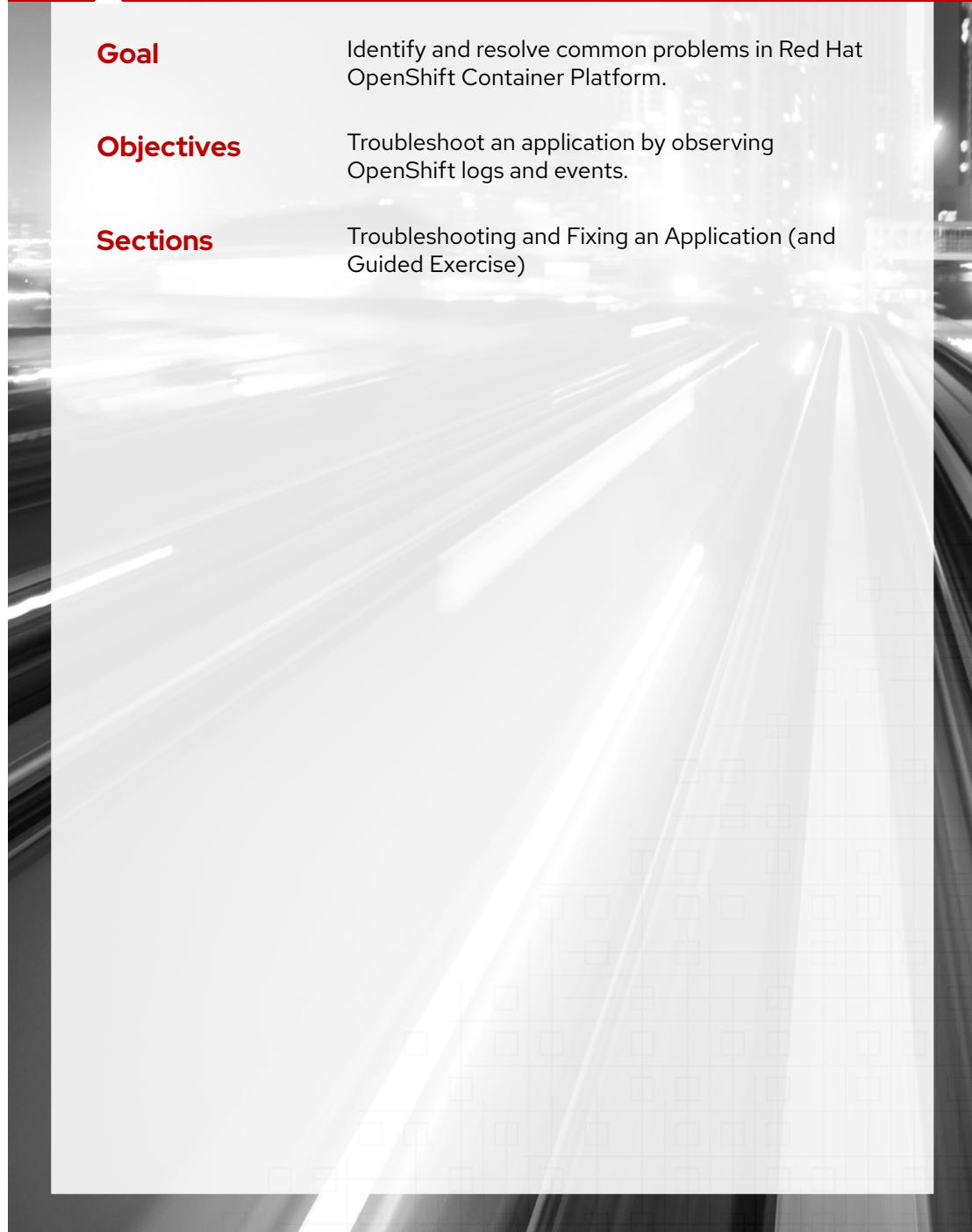
Identify and resolve common problems in Red Hat OpenShift Container Platform.

### Objectives

Troubleshoot an application by observing OpenShift logs and events.

### Sections

Troubleshooting and Fixing an Application (and Guided Exercise)



# 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 `Deployment` or `DeploymentConfig` resources 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.



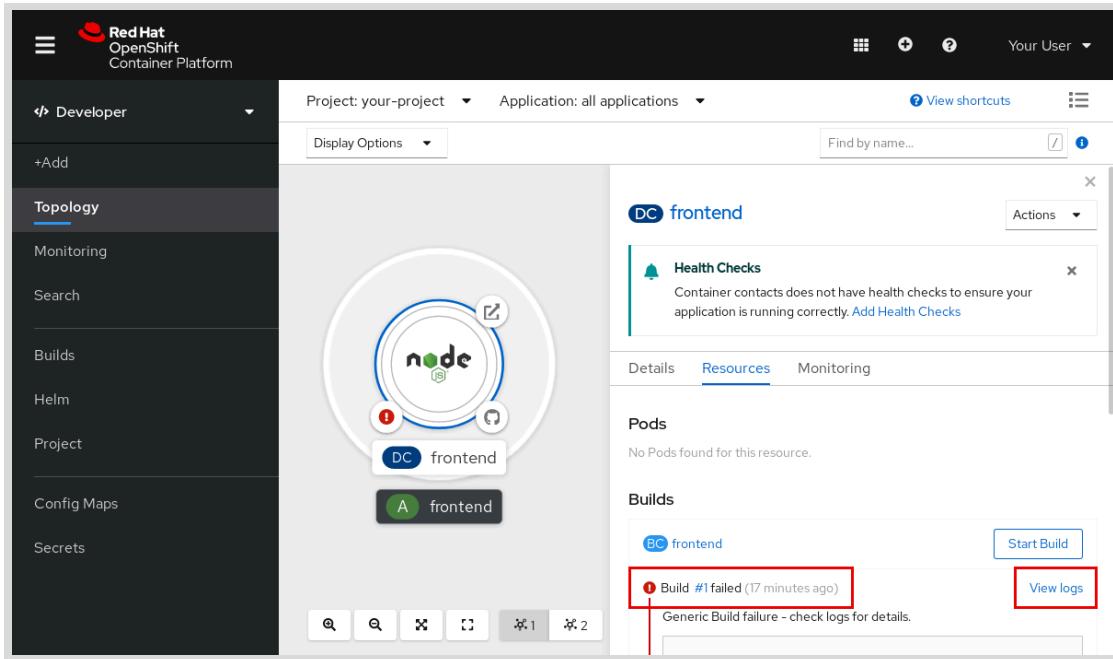
Figure 5.1: Successful and failed builds

If the build is successful, then OpenShift deploys the application. However, the deployment might also fail regardless of the build status. In the following screen capture, the icon on the left shows a successful build and a successful deployment. The icon on the right shows a successful build and a failed deployment.



Figure 5.2: Successful and failed deployments

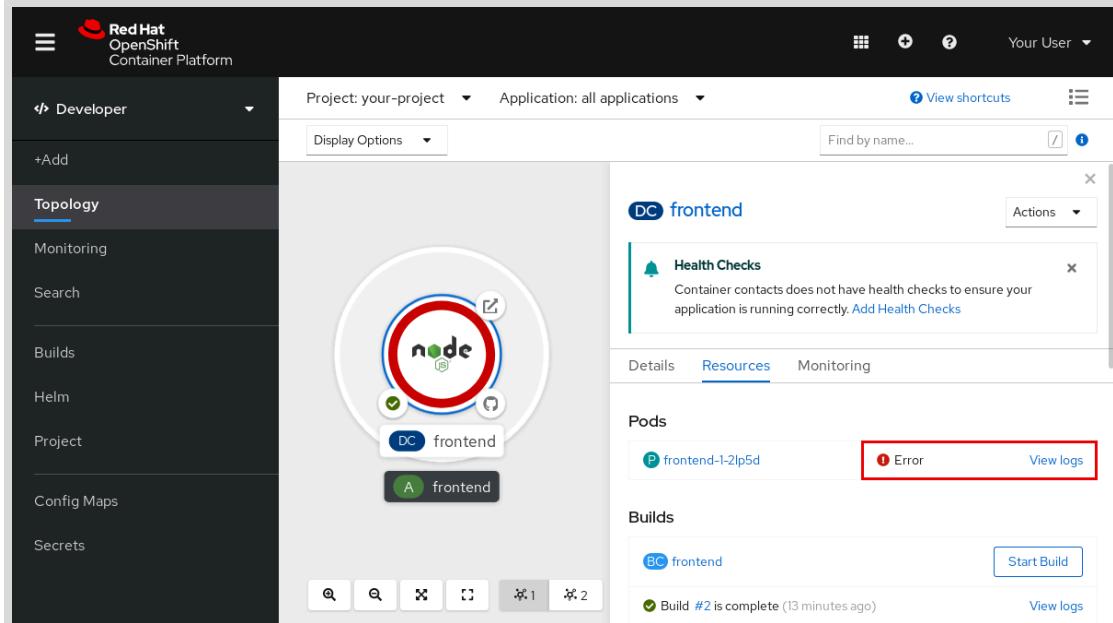
To get more details and access the logs, click the application's icon and navigate to the Resources tab.



**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.

The screenshot shows the Red Hat OpenShift Container Platform interface. On the left, a sidebar menu includes options like Developer, Topology (which is selected), Monitoring, Search, Builds, Helm, Project, Config Maps, and Secrets. The main content area displays a circular topology diagram with a central node labeled 'node' and two surrounding nodes labeled 'DC frontend' and 'A frontend'. Below the diagram, there are search and filter buttons. To the right, a detailed view for the 'dc frontend' pod is shown. It includes a 'Health Checks' section with a warning message about missing health checks, and tabs for 'Details', 'Resources' (selected), and 'Monitoring'. Under the 'Pods' section, the pod 'P frontend-4-wjkzw' is listed as 'Running'. In the 'Builds' section, it shows a completed build #6. A red box highlights the pod name 'P frontend-4-wjkzw' in the 'Pods' list.

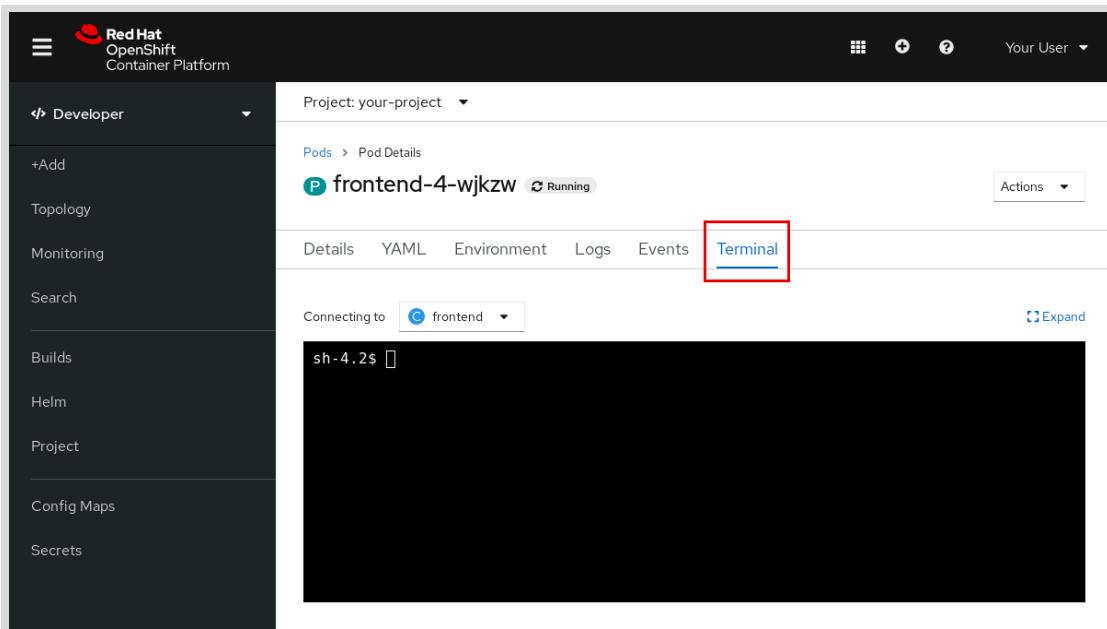
**Figure 5.5: Accessing the pod details**

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.

The screenshot shows the Red Hat OpenShift Container Platform interface, specifically the 'Pod Details' view for the 'P frontend-4-wjkzw' pod. The sidebar menu is identical to Figure 5.5. The main content shows the pod details with tabs for 'Details', 'YAML', 'Environment', 'Logs' (which is selected and highlighted with a red box), 'Events', and 'Terminal'. Below the tabs, there's a log streaming interface with a dropdown for selecting the container ('frontend'). The log output shows 14 lines of npm and node.js startup logs. A red box highlights the 'Logs' tab. At the bottom right, there are download and expand options.

**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 <https://developers.redhat.com/blog/2018/05/15/debug-your-node-js-application-on-openshift-with-chrome-devtools/>
- Remote Debugging of Java Applications on OpenShift at <https://servicesblog.redhat.com/2019/03/06/remote-debugging-of-java-applications-on-openshift/>

## 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 redeloys 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 **Home > Events** menu in the Administrator perspective.

**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.



## References

For more information on events, refer to the Viewing system event information in an OpenShift Container Platform cluster chapter in the Configuring and managing nodes in OpenShift Container Platform guide at  
[https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.6/html-single/nodes/index#nodes-containers-events](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.6/html-single/nodes/index#nodes-containers-events)

For more information, refer to the Performing and interacting with builds in OpenShift Container Platform at  
[https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.6/html-single/builds/index](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.6/html-single/builds/index)

## ► Guided Exercise

# Troubleshooting and Fixing 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.

### Before You Begin

To perform this exercise, ensure you have:

- Access to a running Red Hat OpenShift Container Platform (RHOCP) cluster. See *Guided Exercise: Deploying an Application to Red Hat OpenShift Container Platform* for more information about using the developer sandbox RHOCP cluster.
- Visual Studio Code (VS Code) and Git are installed on your system
- Cloned your GitHub D0101x-app repository in VS Code.
- Installed the oc command.

## Instructions

- 1. In VS Code, in the D0101x-apps repository, create a new branch named `troubleshoot` for recording your work in this exercise. Push that branch to your GitHub repository.
- 1.1. If you do not have VS Code open from a previous exercise, then open it.
  - 1.2. In the source control view in VS Code (*View > SCM*), ensure that the D0101x-apps entry under SOURCE CONTROL REPOSITORIES shows the main branch.



#### Note

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

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

**Warning**

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

13. Click `main` in the `D0101x-apps` entry under **SOURCE CONTROL REPOSITORIES**, and then select **Create new branch**. Type `troubleshoot` to name the branch.
  14. To push the new branch to GitHub, click the **Publish Changes** icon for the `D0101x-apps` entry. If prompted to authorize VS Code to push commits, then click **Allow** and continue on the GitHub website.  
Alternatively, you can choose not to authorize GitHub. In such case, you will be prompted for username and password. Enter your GitHub developer token as your username, and leave the password field empty.
- 2. The `contacts-troubleshoot` application you deploy in this exercise requires a PostgreSQL database. Deploy that database from the catalog.
- 2.1. Log in to the RHOCP web console by using your developer account. Confirm the **Developer perspective** is active.
  - 2.2. Click **Add**, and then click **Database**. Make sure that there are no filtering options selected.
  - 2.3. Click **Postgres**, and then click **PostgreSQL (Ephemeral)**.

**Warning**

If you do not see the `PostgreSQL (Ephemeral)` template, make sure the **Template** filter is selected in the **Type** section.

Click **Instantiate Template**, and then complete the form according to the following table:

**PostgreSQL Parameters**

Parameter	Value
Database Service Name	<code>postgresql</code>
PostgreSQL Database Name	<code>contactsdbs</code>

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.

- 2.4. To inspect the secret resource that OpenShift creates to store the database authentication parameters, click the **Secrets** menu.
- 2.5. From the resulting list of secrets, click the `postgresql` secret.

Secret	Type	Age	Actions
youruser-troubleshoot-app-kubernetes-service-account-token	Opaque	7 minutes ago	⋮
deployer-dockercfg-mzlg	kubernetes.io/dockercfg	1	⋮ 7 minutes ago
deployer-token-4tv18	kubernetes.io/service-account-token	4	⋮ 7 minutes ago
deployer-token-d6hx2	kubernetes.io/service-account-token	4	⋮ 7 minutes ago
pipeline-dockercfg-kf4cl	kubernetes.io/dockercfg	1	⋮ 7 minutes ago
pipeline-token-fcj7w	kubernetes.io/service-account-token	4	⋮ 7 minutes ago
pipeline-token-tq9c8	kubernetes.io/service-account-token	4	⋮ 7 minutes ago
<b>postgreSQL</b>	Opaque	3	⋮ 5 minutes ago
postgreSQL-ephemeral-parameters-m5whn	Opaque	5	⋮ 5 minutes ago

**Figure 5.9: 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.

Data	Reveal Values
database-name	.....
database-password	.....
database-user	.....

**Figure 5.10: Data stored in the secret resource**

- ▶ 3. 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.
  - 3.1. Click **Add**, and then click **All services** in the **Developer Catalog** section.
  - 3.2. Click **Languages > JavaScript**, and then click the **Node . js** builder image. Click **Create Application** to enter the details of the application.
  - 3.3. Complete the form according to the following table. To access the Git parameters, click **Show Advanced Git Options**.

## New Application Parameters

Parameter	Value
Git Repo URL	<a href="https://github.com/yourgituser/D0101X-apps">https://github.com/yourgituser/D0101X-apps</a>
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.

- ▶ 4. OpenShift fails to build your application. Use the OpenShift web console to locate the error. Access the log to get more details.
  - 4.1. The web console should automatically show the **Topology** page. If this page is not displayed, then click the **Topology** tab.
  - 4.2. After a few minutes, the application displays an error state.

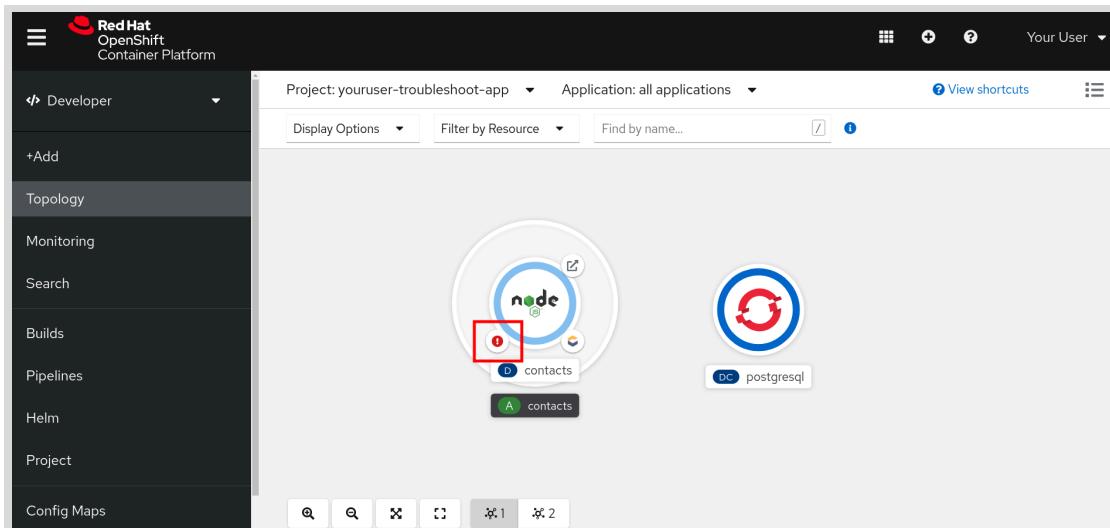
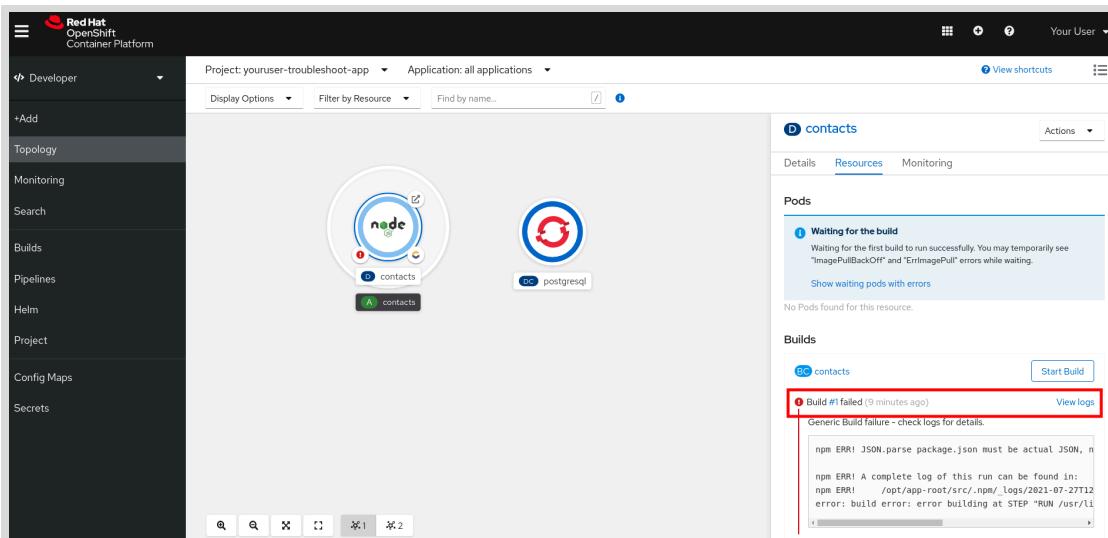


Figure 5.11: Error in the build step

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 5.12: 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.

- 4.3. 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.

```
...output omitted...
---> Installing dependencies
npm ERR! code EJSONPARSE
npm ERR! file /opt/app-root/src/package.json
npm ERR! JSON.parse Failed to parse json
npm ERR! JSON.parse Unexpected token d in JSON at position 118 while parsing near
  '... ./bin/www.js'
npm ERR! JSON.parse  },
npm ERR! JSON.parse dependencies": {
npm ERR! JSON.parse   ...
npm ERR! JSON.parse Failed to parse package.json data.
npm ERR! JSON.parse package.json must be actual JSON, not just JavaScript.

npm ERR! A complete log of this run can be found in:
npm ERR!     /opt/app-root/src/.npm/_logs/2020-08-11T10_12_06_526Z-debug.log
subprocess exited with status 1
subprocess exited with status 1
error: build error: error building at STEP "RUN /usr/libexec/s2i/assemble": exit
status 1
```

5. Open the `D0101x-apps/contacts-troubleshoot/package.json` file to identify and fix the issue. When finished, commit and push your change to GitHub.
  - 5.1. In the VS Code Explorer view (**View > Explorer**), open the `D0101x-apps/contacts-troubleshoot/package.json` file.
  - 5.2. The `dependencies` keyword must be enclosed in double quotes. Notice that the opening double quote is missing.

- 5.3. Add the missing double quote and then save the file.

```
"dependencies": {
    "cookie-parser": "1.4.5",
    "debug": "4.3.2",
    "dotenv": "10.0.0",
    "express": "4.17.1",
    "http-errors": "1.8.0",
    "morgan": "1.10.0",
    "pg": "8.6.0",
    "pug": "3.0.2"
},
```

- 5.4. 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.
- 5.5. To push your change to the GitHub repository, click the **Synchronize Changes** icon for the DO101x-apps entry, under **SOURCE CONTROL REPOSITORIES**. If VS Code displays a message saying that this action will push and pull commits to and from **origin/troubleshoot**, then click **OK**.

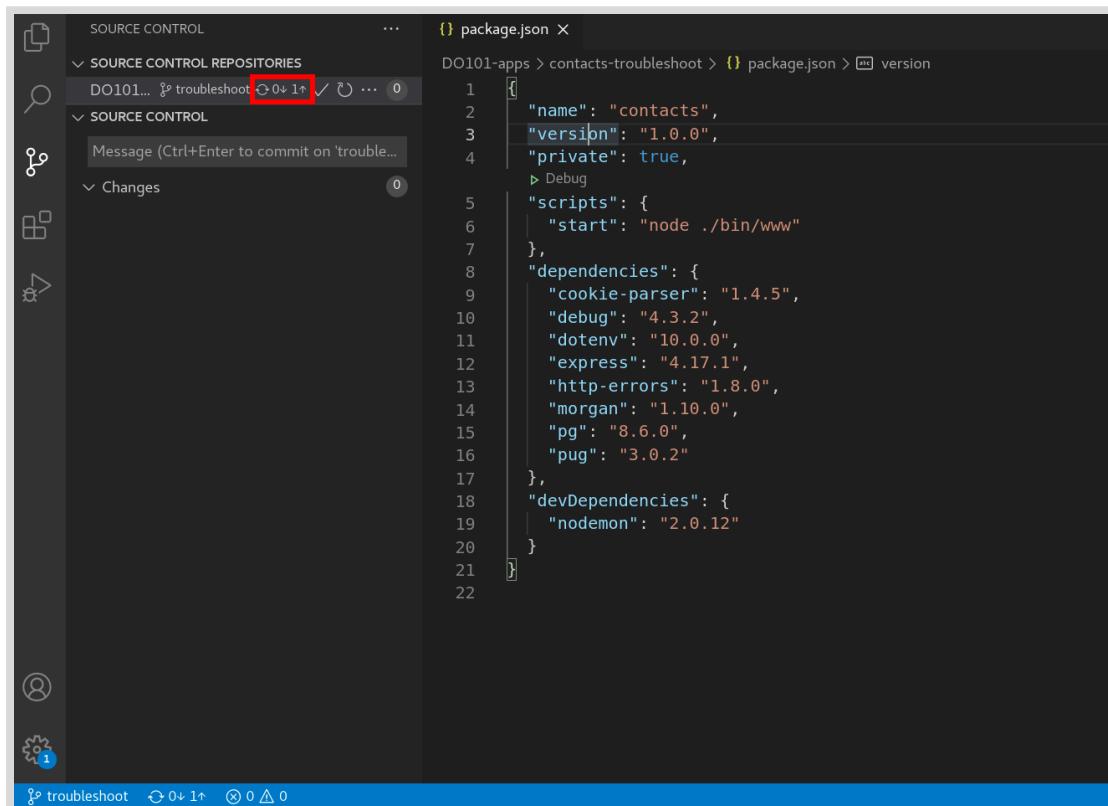
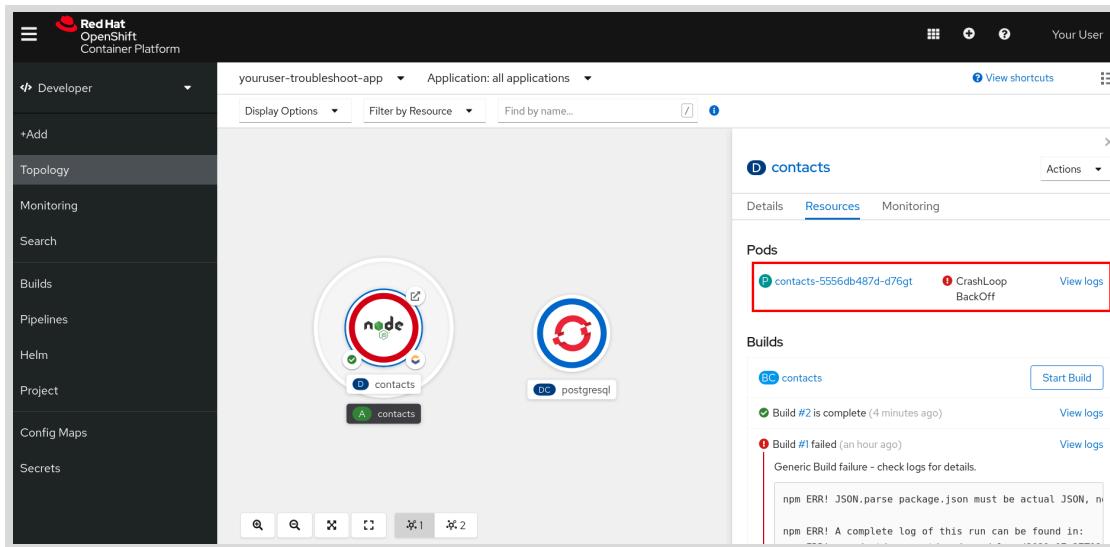


Figure 5.13: Pushing changes to GitHub

- 6. Use the OpenShift web console to initiate a new build and deployment. Click the **Topology** tab. Click the **Node.js** icon, and then click the **Resources** tab. Click **Start Build** to initiate a build. For this new build, OpenShift retrieves your fixed version of the application from GitHub.

- ▶ **7.** 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.
  - 7.1. Wait for the build to finish and notice that the build is successful. The pod, however, is in the error state.



**Figure 5.14: Deployment in error**

The error status may alternate between CrashLoopBackOff and Error.

- 7.2. Click View Logs near the pod in error to access its log.

```
...output omitted...
> node ./bin/www.js

internal/modules/cjs/loader.js:638
    throw err;
    ^

Error: Cannot find module '/opt/app-root/src/bin/www.js'
...output omitted...
```

During startup, Node.js cannot find the program to run.



### Note

If you have difficulties displaying the pod logs in the RHOCP web console, you can use the oc utility to get the logs.

Execute `oc get pods` to get the pod name. Execute `oc logs POD_NAME` to get the logs of a pod.

For example, `oc logs contacts-f45d565c9-8558b` prints the logs of the `contacts-f45d565c9-8558b` pod to the console.

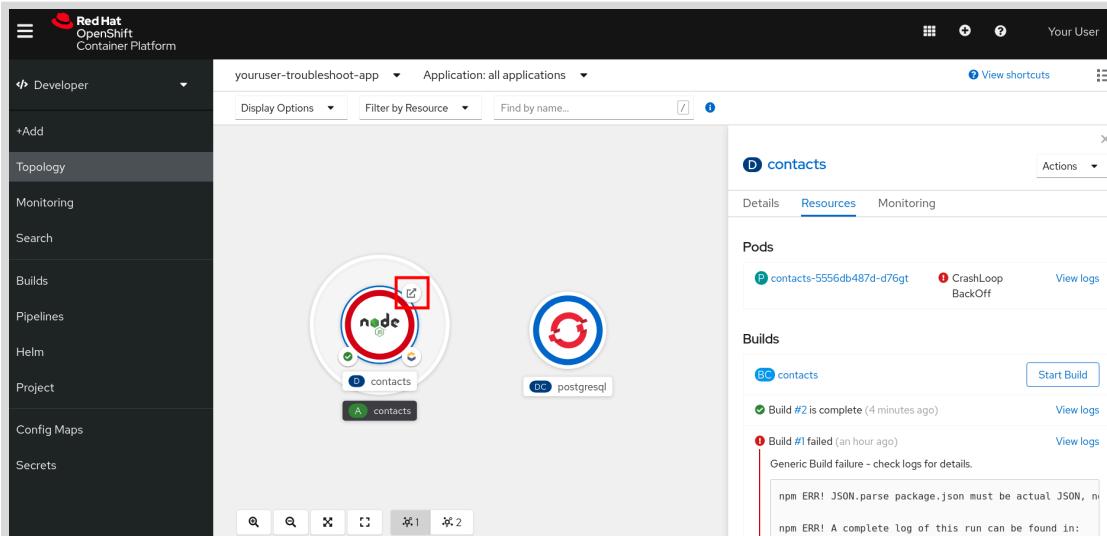
- ▶ **8.** The Node.js application declares the program to start in the package.json file. Identify and fix the issue in the `D0101X-apps/contacts-troubleshoot/package.json` file. When finished, commit and push your change to GitHub.

- 8.1. Open the D0101x-apps/contacts-troubleshoot/package.json file in VS Code.
- 8.2. The file defines the program to run in the `scripts` section. The program is `./bin/www.js`.
- 8.3. In VS Code, navigate to the `./bin/` directory (D0101x-apps/contacts-troubleshoot/bin/). In this directory, the program file is `www`, not `www.js`.
- 8.4. In `package.json`, replace `www.js` by `www`, and then save the file.

```
...output omitted...
"scripts": {
  "start": "node ./bin/www"
},
...output omitted...
```

- 8.5. 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.
  - 8.6. To push your change to the GitHub repository, click the **Synchronize Changes** icon for the `D0101x-apps` entry, under **SOURCE CONTROL REPOSITORIES**.
- 9. 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.
- 10. 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.

- 10.1. Click the **Open URL** button to access the application.



**Figure 5.15: Accessing the contacts application**

- 10.2. The application displays an error message regarding the database connection.

## Contact List

```
Database connection failure! error: password authentication failed for user "undefined" at Parser.parseErrorMessage (/opt/app-root/src/node_modules/pg-protocol/dist/parser.js:287:98) at Parser.handlePacket (/opt/app-root/src/node_modules/pg-protocol/dist/parser.js:126:29) at Parser.parse (/opt/app-root/src/node_modules/pg-protocol/dist/index.js:11:42) at Socket.emit (events.js:314:20) at addChunk (_stream_readable.js:298:12) at readableAddChunk (_stream_readable.js:273:9) at Socket.Readable.push (_stream_readable.js:214:10) at TCP.onStreamRead (internal/stream_base_commons.js:188:23)
```

**Figure 5.16: Database error**

This message indicates that the provided user name for database authentication is not defined. When finished, close the browser tab.

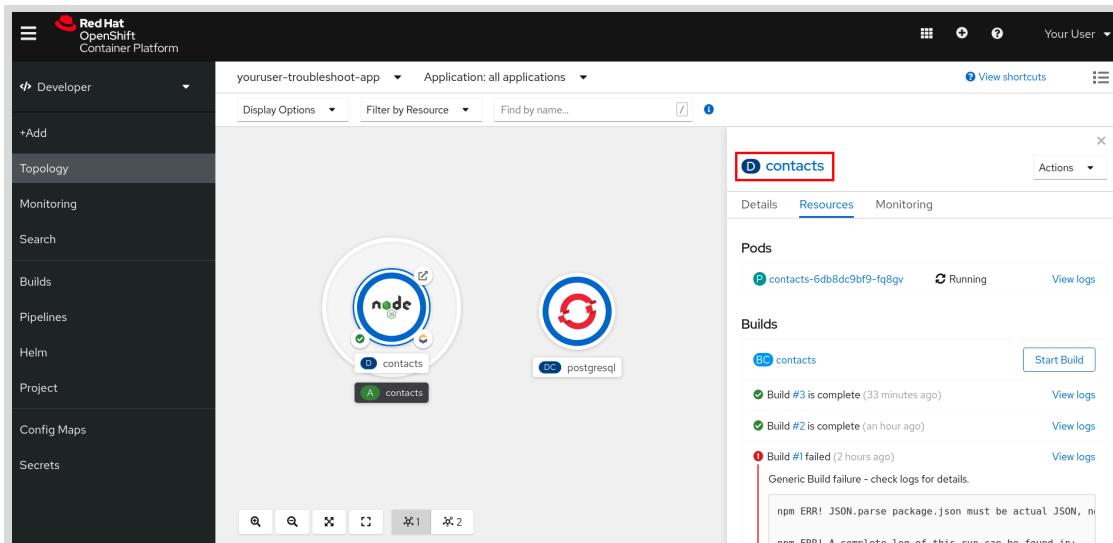
- 11. The Node.js application defines the database parameters in the D0101x-apps/contacts-troubleshoot/db/config.js file. Open this file with VS Code.

```
...output omitted...
// The following variables should be defined in the
// secret resource associated with the database.
var db_user = process.env["database-user"];
var db_pass = process.env["database-password"];
var db_name = process.env["database-name"];
...output omitted...
```

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.

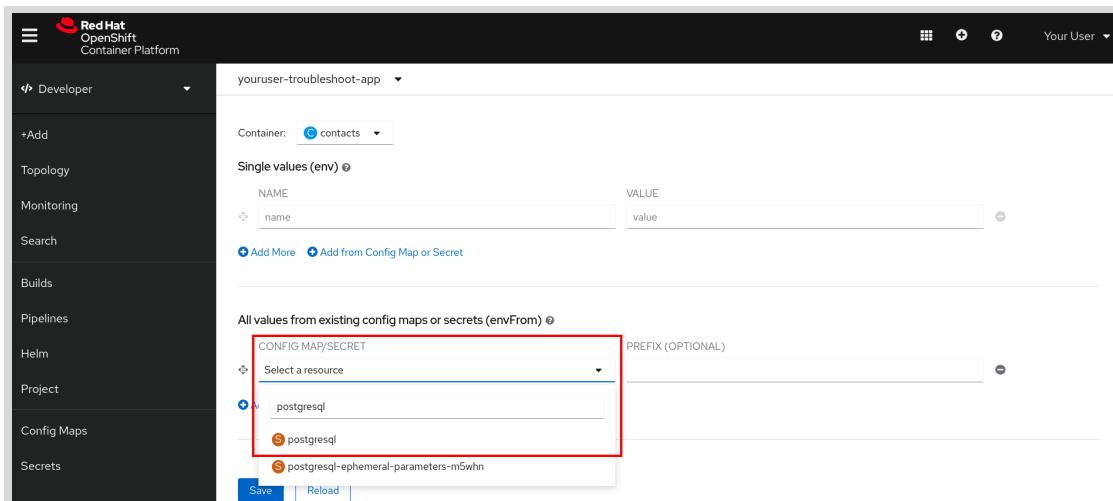
- 12. Use the OpenShift web console to associate the postgresql secret to the contacts deployment configuration resource. With this association, OpenShift defines each entry in the secret as environment variables in the pod. When done, confirm that the application is finally working as expected.

- 12.1. Click the contacts deployment link.



**Figure 5.17: Accessing the deployment configuration details**

- 12.2. 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 5.18: Associating the secret with the deployment configuration**

When you update the deployment configuration resource, OpenShift automatically redeploys the application.

- 12.3. 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.

ID	First Name	Last Name	EMail
1	Bilbo	Baggins	bilbo@theshire.com
2	Frodo	Baggins	frodo@theshire.com
3	Samwise	Gamgee	sam@theshire.com
4	Peregrin	Took	pippin@theshire.com
5	Meriadoc	Brandybuck	merry@theshire.com

Figure 5.19: The contacts application

When done, close the browser tab.

► 13. Clean up.

13.1. Delete RHOCP objects with the `app=contacts` label:

```
$ oc delete all --selector app=contacts
pod "contacts-847b766bc8-f9bf8" deleted
service "contacts" deleted
deployment.apps "contacts" deleted
buildconfig.build.openshift.io "contacts" deleted
imagestream.image.openshift.io "contacts" deleted
route.route.openshift.io "contacts" deleted
```

13.2. Delete RHOCP objects with the `template=postgresql-ephemeral-template` label:

```
$ oc delete all,secret --selector template=postgresql-ephemeral-template
replicationcontroller "postgresql-1" deleted
service "postgresql" deleted
deploymentconfig.apps.openshift.io "postgresql" deleted
secret "postgresql" deleted
```

13.3. Delete the `contacts-github-webhook-secret`, and `contacts-generic-webhook-secret` secrets:

```
$ oc delete secret contacts-generic-webhook-secret contacts-github-webhook-secret
secret "contacts-generic-webhook-secret" deleted
secret "contacts-github-webhook-secret" deleted
```

## Finish

This concludes the guided exercise.

# Summary

---

In this chapter, you learned:

- The **Topology** page of the OpenShift web console is used to identify the deployment step in error.
- OpenShift maintains logs for the build, deployment, and running processes.
- For advanced troubleshooting, use the console to run commands inside a running pod.
- OpenShift events record significant actions, such as the start of a pod or the failure of a build process.

## Chapter 6

# Deploying Containerized Applications on OpenShift

### Goal

Deploy single container applications on OpenShift Container Platform.

### Objectives

- Describe the architecture of Kubernetes and Red Hat OpenShift Container Platform.
- Create standard Kubernetes resources.
- Expose services using OpenShift routes.
- Deploy an application using the Source-to-Image (S2I) facility of OpenShift Container Platform.

### Sections

- Describing Kubernetes and OpenShift Architecture (and Quiz)
- Creating Kubernetes Resources (and Guided Exercise)
- Creating Routes (and Guided Exercise)
- Creating Applications with Source-to-Image (and Guided Exercise)

# Describing Kubernetes and OpenShift Architecture

---

## Objectives

After completing this section, students should be able to

- Describe the architecture of a Kubernetes cluster running on the Red Hat OpenShift Container Platform (RHOCP).
- List the main resource types provided by Kubernetes and RHOCP.
- Identify the network characteristics of containers, Kubernetes, and RHOCP.
- List mechanisms to make a pod externally available.

## Kubernetes and OpenShift

In previous chapters we saw that Kubernetes is an orchestration service that simplifies the deployment, management, and scaling of containerized applications. One of the main advantages of using Kubernetes is that it uses several nodes to ensure the resiliency and scalability of its managed applications. Kubernetes forms a cluster of node servers that run containers and are centrally managed by a set of control plane servers. A server can act as both a control plane node and a compute node, but those roles are usually separated for increased stability.

### Kubernetes Terminology

Term	Definition
Node	A server that hosts applications in a Kubernetes cluster.
Control Plane	Provides basic cluster services such as APIs or controllers.
Compute Node	This node executes workloads for the cluster. Application pods are scheduled onto compute nodes.
Resource	Resources are any kind of component definition managed by Kubernetes. Resources contain the configuration of the managed component (for example, the role assigned to a node), and the current state of the component (for example, if the node is available).
Controller	A controller is a Kubernetes process that watches resources and makes changes attempting to move the current state towards the desired state.
Label	A key-value pair that can be assigned to any Kubernetes resource. Selectors use labels to filter eligible resources for scheduling and other operations.
Namespace	A scope for Kubernetes resources and processes, so that resources with the same name can be used in different boundaries.

**Note**

The latest Kubernetes versions implement many controllers as Operators. Operators are Kubernetes plug-in components that can react to cluster events and control the state of resources. Operators and CoreOS Operator Framework are outside the scope of this document.

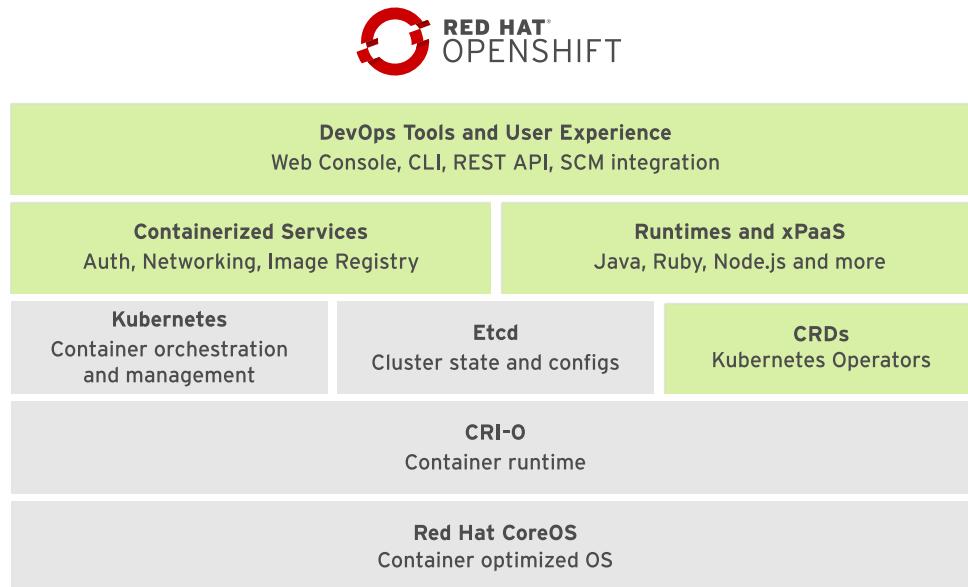
Red Hat OpenShift Container Platform is a set of modular components and services built on top of Red Hat CoreOS and Kubernetes. RHOCP adds PaaS capabilities such as remote management, increased security, monitoring and auditing, application lifecycle management, and self-service interfaces for developers.

An OpenShift cluster is a Kubernetes cluster that can be managed the same way, but using the management tools provided by OpenShift, such as the command-line interface or the web console. This allows for more productive workflows and makes common tasks much easier.

### OpenShift Terminology

Term	Definition
Infra Node	A node server containing infrastructure services like monitoring, logging, or external routing.
Console	A web UI provided by the RHOCP cluster that allows developers and administrators to interact with cluster resources.
Project	OpenShift extension of Kubernetes' namespaces. Allows the definition of user access control (UAC) to resources.

The following schema illustrates the OpenShift Container Platform stack:

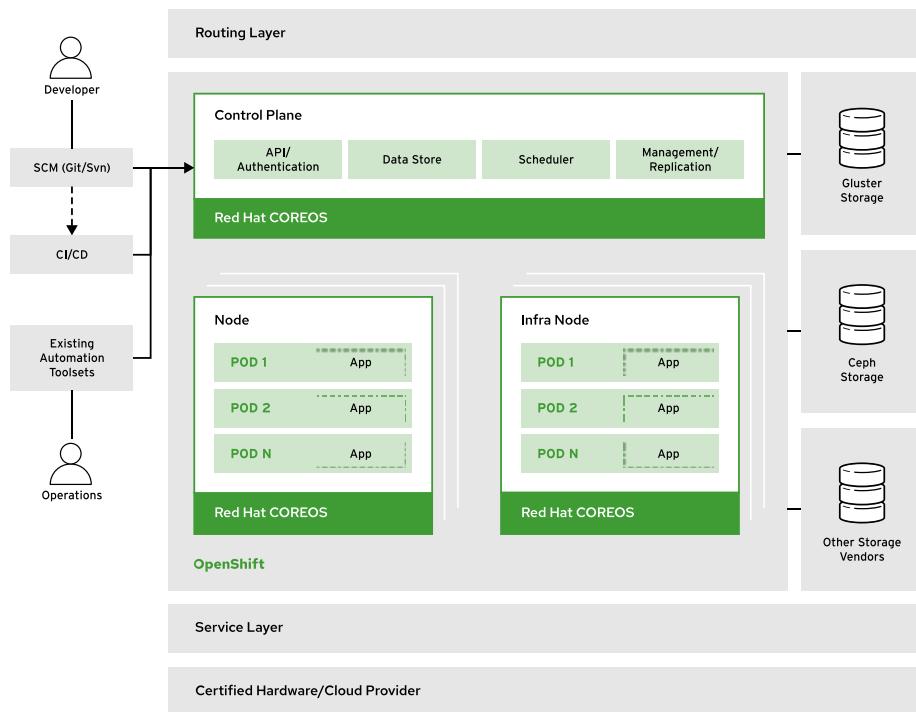


**Figure 6.1: OpenShift component stack**

From bottom to top, and from left to right, this shows the basic container infrastructure, integrated and enhanced by Red Hat:

- The base OS is Red Hat CoreOS. Red Hat CoreOS is a Linux distribution focused on providing an immutable operating system for container execution.
- CRI-O is an implementation of the Kubernetes Container Runtime Interface (CRI) to enable using Open Container Initiative (OCI) compatible runtimes. CRI-O can use any container runtime that satisfies CRI: `runc` (used by the Docker service), `libpod` (used by Podman) or `rkt` (from CoreOS).
- Kubernetes manages a cluster of hosts, physical or virtual, that run containers. It uses resources that describe multicontainer applications composed of multiple resources, and how they interconnect.
- Etcd is a distributed key-value store, used by Kubernetes to store configuration and state information about the containers and other resources inside the Kubernetes cluster.
- Custom Resource Definitions (CRDs) are resource types stored in Etcd and managed by Kubernetes. These resource types form the state and configuration of all resources managed by OpenShift.
- Containerized services fulfill many PaaS infrastructure functions, such as networking and authorization. RHOCP uses the basic container infrastructure from Kubernetes and the underlying container runtime for most internal functions. That is, most RHOCP internal services run as containers orchestrated by Kubernetes.
- Runtimes and xPaaS are base container images ready for use by developers, each preconfigured with a particular runtime language or database. The xPaaS offering is a set of base images for Red Hat middleware products such as JBoss EAP and ActiveMQ. Red Hat OpenShift Application Runtimes (RHOAR) are a set runtimes optimized for cloud native applications in OpenShift. The application runtimes available are Red Hat JBoss EAP, OpenJDK, Thorntail, Eclipse Vert.x, Spring Boot, and Node.js.
- RHOCP provides web UI and CLI management tools for managing user applications and RHOCP services. The OpenShift web UI and CLI tools are built from REST APIs which can be used by external tools such as IDEs and CI platforms.

This OpenShift and Kubernetes architecture illustration gives further insight into how the infrastructure components work together.



**Figure 6.2: OpenShift and Kubernetes architecture**

## New Features in RHOC 4

RHOC 4 is a massive change from previous versions. As well as keeping backwards compatibility with previous releases, it includes new features, such as:

- CoreOS as the mandatory operating system for all nodes, offering an immutable infrastructure optimized for containers.
- A brand new cluster installer which guides the process of installation and update.
- A self-managing platform, able to automatically apply cluster updates and recoveries without disruption.
- A redesigned application lifecycle management.
- An Operator SDK to build, test, and package Operators.

## Describing Kubernetes Resource Types

Kubernetes has six main resource types that can be created and configured using a YAML or a JSON file, or using OpenShift management tools:

### **Pods (po)**

Represent a collection of containers that share resources, such as IP addresses and persistent storage volumes. It is the basic unit of work for Kubernetes.

### **Services (svc)**

Define a single IP/port combination that provides access to a pool of pods. By default, services connect clients to pods in a round-robin fashion.

### **Replication Controllers (rc)**

A Kubernetes resource that defines how pods are replicated (horizontally scaled) into different nodes. Replication controllers are a basic Kubernetes service to provide high availability for pods and containers.

### **Persistent Volumes (pv)**

Define storage areas to be used by Kubernetes pods.

### **Persistent Volume Claims (pvc)**

Represent a request for storage by a pod. PVCs links a PV to a pod so its containers can make use of it, usually by mounting the storage into the container's file system.

### **ConfigMaps (cm) and Secrets**

Contains a set of keys and values that can be used by other resources. ConfigMaps and Secrets are usually used to centralize configuration values used by several resources. Secrets differ from ConfigMaps maps in that Secrets' values are always encoded (not encrypted) and their access is restricted to fewer authorized users.



#### **Note**

Although Kubernetes pods can be created standalone, they are usually created by high-level resources such as replication controllers.

## **OpenShift Resource Types**

The main resource types added by OpenShift Container Platform to Kubernetes are as follows:

### **Deployment and Deployment config (dc)**

OpenShift 4.5 introduced the Deployment resource concept to replace the DeploymentConfig as the default configuration for pods. Both are the representation of a set of containers included in a pod, and the deployment strategies to be used. It contains the configuration to be applied to all containers of each pod replica, such as the base image, tags, storage definitions and the commands to be executed when the containers start.

The Deployment object serves as the improved version of the DeploymentConfig object. Some replacements of functionalities between both objects are the following:

- Automatic rollback is no longer supported by deployment objects.
- Every change in the pod template used by deployment objects triggers a new rollout automatically.
- Lifecycle hooks are no longer supported by deployment objects.
- The deployment process of a deployment object can be paused at any time without affecting the deployer process.
- A deployment object can have as many active replica sets as the user wants, scaling down old replicas after a while. In contrast, the deploymentconfig object can only have two replication sets active at the same time.

Even if Deployment objects are meant to act as the default replacement of DeploymentConfig objects, in OpenShift 4.6 users can still make use of them if they need a specific feature provided by these objects. In this case, it is necessary to specify the type of object when creating a new application by specifying the `--as-deployment-config` flag.

### **Build config (bc)**

Defines a process to be executed in the OpenShift project. Used by the OpenShift Source-to-Image (S2I) feature to build a container image from application source code stored in a Git repository. A bc works together with a dc to provide a basic but extensible continuous integration and continuous delivery workflows.

### **Routes**

Represent a DNS host name recognized by the OpenShift router as an ingress point for applications and microservices.



#### **Note**

To obtain a list of all the resources available in a RHOCP cluster and their abbreviations, use the `oc api-resources` or `kubectl api-resources` commands.

Although Kubernetes replication controllers can be created standalone in OpenShift, they are usually created by higher-level resources such as deployment controllers.

## **Describing Networking in Red Hat OpenShift**

Each container deployed in a Kubernetes cluster has an IP address assigned from an internal network that is accessible only from the node running the container. Because of the container's ephemeral nature, IP addresses are constantly assigned and released.

Kubernetes provides a software-defined network (SDN) that spawns the internal container networks from multiple nodes and allows containers from any pod, inside any host, to access pods from other hosts. Access to the SDN only works from inside the same Kubernetes cluster.

Containers inside Kubernetes pods should not connect to each other's dynamic IP address directly. Services resolves this problem by linking more stable IP addresses from the SDN to the pods. If pods are restarted, replicated, or rescheduled to different nodes, services are updated, providing scalability and fault tolerance.

External access to containers is more complicated. Kubernetes services can be set as a `NodePort` service type. With this type of service, Kubernetes allocates a network port from a predefined range in each of the cluster nodes and proxies it into your service. Unfortunately, this approach does not scale well.

OpenShift makes external access to containers both scalable and simpler by defining route resources. A route defines external-facing DNS names and ports for a service. A router (ingress controller) forwards HTTP and TLS requests to the service addresses inside the Kubernetes SDN. The only requirement is that the desired DNS names are mapped to the IP addresses of the RHOCP router nodes.



## References

### Kubernetes documentation website

<https://kubernetes.io/docs/>

### OpenShift documentation website

<https://docs.openshift.com/>

### Understanding Operators

<https://docs.openshift.com/container-platform/4.6/operators/olm-what-operators-are.html>

# Creating Kubernetes Resources

---

## Objectives

After completing this section, students should be able to create standard Kubernetes resources.

## The Red Hat OpenShift Container Platform (RHOCP) Command-line Tool

The main method of interacting with an RHOCP cluster is using the `oc` command. The basic usage of the command is through its subcommands in the following syntax:

```
$ oc <command>
```

Before interacting with a cluster, most operations require the user to log in. The syntax to log in is shown below:

```
$ oc login <clusterUrl>
```

## Describing Pod Resource Definition Syntax

RHOCP runs containers inside Kubernetes pods, and to create a pod from a container image, OpenShift needs a *pod resource definition*. This can be provided either as a JSON or YAML text file, or can be generated from defaults by the `oc new-app` command or the OpenShift web console.

A pod is a collection of containers and other resources. An example of a WildFly application server pod definition in YAML format is shown below:

```
apiVersion: v1
kind: Pod①
metadata:
  name: wildfly②
  labels:
    name: wildfly③
spec:
  containers:
    - resources:
        limits:
          cpu: 0.5
      image: do276/todojee④
      name: wildfly
      ports:
        - containerPort: 8080⑤
          name: wildfly
    env:⑥
      - name: MYSQL_ENV_MYSQL_DATABASE
        value: items
```

```

- name: MYSQL_ENV_MYSQL_USER
  value: user1
- name: MYSQL_ENV_MYSQL_PASSWORD
  value: mypa55

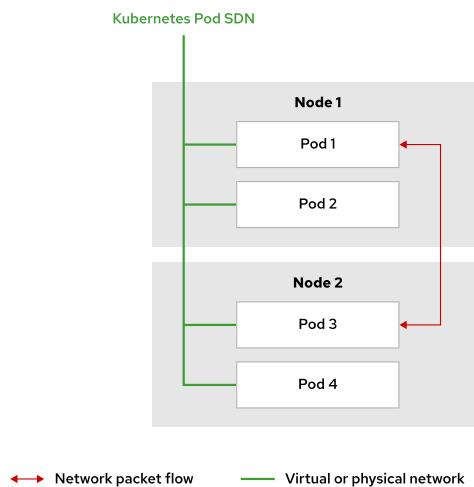
```

- ➊ Declares a Kubernetes pod resource type.
- ➋ A unique name for a pod in Kubernetes that allows administrators to run commands on it.
- ➌ Creates a label with a key named name that other resources in Kubernetes, usually as service, can use to find it.
- ➍ Defines the container image name.
- ➎ A container-dependent attribute identifying which port on the container is exposed.
- ➏ Defines a collection of environment variables.

Some pods may require environment variables that can be read by a container. Kubernetes transforms all the name and value pairs to environment variables. For instance, the MYSQL\_ENV\_MYSQL\_USER variable is declared internally by the Kubernetes runtime with a value of user1, and is forwarded to the container image definition. Because the container uses the same variable name to get the user's login, the value is used by the WildFly container instance to set the username that accesses a MySQL database instance.

## Describing Service Resource Definition Syntax

Kubernetes provides a virtual network that allows pods from different compute nodes to connect. But, Kubernetes provides no easy way for a pod to discover the IP addresses of other pods:

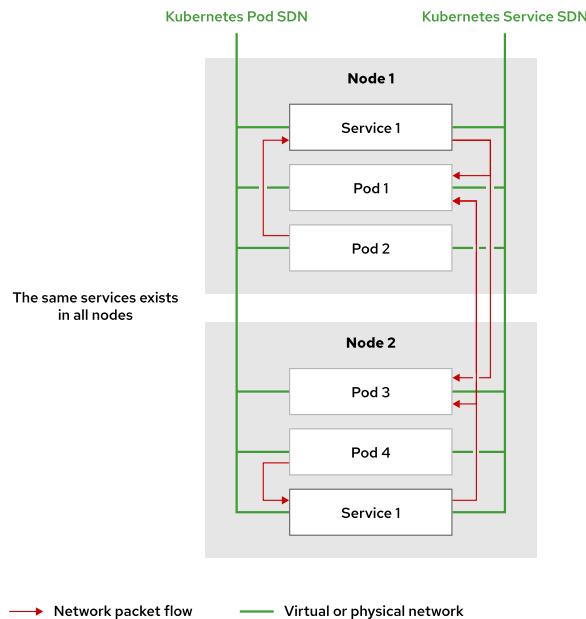


**Figure 6.3: Basic Kubernetes networking**

If Pod 3 fails and is restarted, it could return with a different IP address. This would cause Pod 1 to fail when attempting to communicate with Pod 3. A service layer provides the abstraction required to solve this problem.

Services are essential resources to any OpenShift application. They allow containers in one pod to open network connections to containers in another pod. A pod can be restarted for many reasons, and it gets a different internal IP address each time. Instead of a pod having to discover the IP

address of another pod after each restart, a service provides a stable IP address for other pods to use, no matter what compute node runs the pod after each restart:



**Figure 6.4: Kubernetes services networking**

Most real-world applications do not run as a single pod. They need to scale horizontally, so many pods run the same containers from the same pod resource definition to meet growing user demand. A service is tied to a set of pods, providing a single IP address for the whole set, and a load-balancing client request among member pods.

The set of pods running behind a service is managed by a Deployment resource. A Deployment resource embeds a ReplicationController that manages how many pod copies (replicas) have to be created, and creates new ones if any of them fail. Deployment and ReplicationController resources are explained later in this chapter.

The following example shows a minimal service definition in JSON syntax:

```
{
  "kind": "Service", ①
  "apiVersion": "v1",
  "metadata": {
    "name": "quotedb" ②
  },
  "spec": {
    "ports": [ ③
      {
        "port": 3306,
        "targetPort": 3306
      }
    ],
    "selector": {
      "name": "mysql ldb" ④
    }
  }
}
```

```

        }
    }
}

```

- ➊ The kind of Kubernetes resource. In this case, a Service.
- ➋ A unique name for the service.
- ➌ `ports` is an array of objects that describes network ports exposed by the service. The `targetPort` attribute has to match a `containerPort` from a pod container definition, and the `port` attribute is the port that is exposed by the service. Clients connect to the service port and the service forwards packets to the pod `targetPort`.
- ➍ `selector` is how the service finds pods to forward packets to. The target pods need to have matching labels in their metadata attributes. If the service finds multiple pods with matching labels, it load balances network connections between them.

Each service is assigned a unique IP address for clients to connect to. This IP address comes from another internal OpenShift SDN, distinct from the pods' internal network, but visible only to pods. Each pod matching the `selector` is added to the service resource as an endpoint.

## Discovering Services

An application typically finds a service IP address and port by using environment variables. For each service inside an OpenShift project, the following environment variables are automatically defined and injected into containers for all pods inside the same project:

- `SVC_NAME_SERVICE_HOST` is the service IP address.
- `SVC_NAME_SERVICE_PORT` is the service TCP port.



### Note

The `SVC_NAME` part of the variable is changed to comply with DNS naming restrictions: letters are capitalized and underscores (`_`) are replaced by dashes (`-`).

Another way to discover a service from a pod is by using the OpenShift internal DNS server, which is visible only to pods. Each service is dynamically assigned an SRV record with an FQDN of the form:

```
SVC_NAME.PROJECT_NAME.svc.cluster.local
```

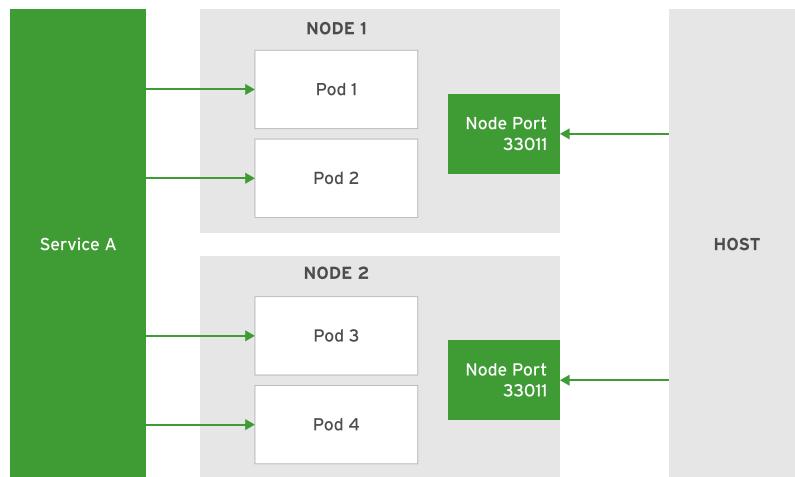
When discovering services using environment variables, a pod has to be created and started only after the service is created. If the application was written to discover services using DNS queries, however, it can find services created after the pod was started.

There are two ways for an application to access the service from outside an OpenShift cluster:

1. `NodePort` type: This is an older Kubernetes-based approach, where the service is exposed to external clients by binding to available ports on the compute node host, which then proxies connections to the service IP address. Use the `oc edit svc` command to edit service attributes and specify `NodePort` as the value for `type`, and provide a port value for the `nodePort` attribute. OpenShift then proxies connections to the service via the public IP address of the compute node host and the port value set in `NodePort`.

2. OpenShift Routes: This is the preferred approach in OpenShift to expose services using a unique URL. Use the `oc expose` command to expose a service for external access or expose a service from the OpenShift web console.

Figure 6.5 illustrates how NodePort services allow external access to Kubernetes services. OpenShift routes are covered in more detail later in this course.



**Figure 6.5: Alternative method for external access to a Kubernetes service**

OpenShift provides the `oc port-forward` command for forwarding a local port to a pod port. This is different from having access to a pod through a service resource:

- The port-forwarding mapping exists only on the workstation where the `oc` client runs, while a service maps a port for all network users.
- A service load balances connections to potentially multiple pods, whereas a port-forwarding mapping forwards connections to a single pod.



**Note**

Red Hat discourages the use of the NodePort approach to avoid exposing the service to direct connections. Mapping via port-forwarding in OpenShift is considered a more secure alternative.

The following example demonstrates the use of the `oc port-forward` command:

```
[user@host ~]$ oc port-forward mysql-openshift-1-glqrp 3306:3306
```

The command forwards port 3306 from the developer machine to port 3306 on the db pod, where a MySQL server (inside a container) accepts network connections.



**Note**

When running this command, make sure you leave the terminal window running. Closing the window or canceling the process stops the port mapping.

## Creating Applications

Simple applications, complex multitier applications, and microservice applications can be described by a single resource definition file. This single file would contain many pod definitions, service definitions to connect the pods, replication controllers or Deployment to horizontally scale the application pods, PersistentVolumeClaims to persist application data, and anything else needed that can be managed by OpenShift.

The `oc new-app` command can be used with the `-o json` or `-o yaml` option to create a skeleton resource definition file in JSON or YAML format, respectively. This file can be customized and used to create an application using the `oc create -f <filename>` command, or merged with other resource definition files to create a composite application.

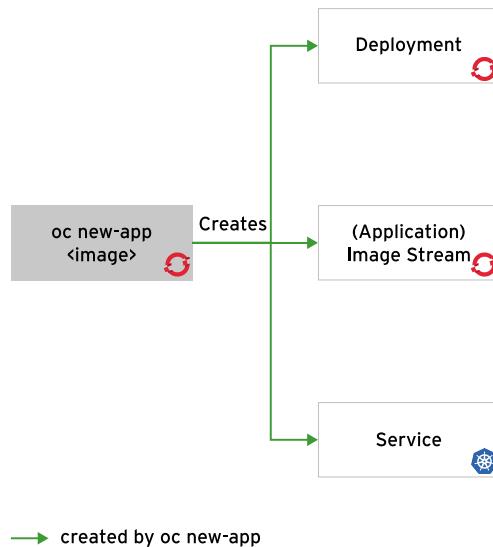
The `oc new-app` command can create application pods to run on OpenShift in many different ways. It can create pods from existing docker images, from Dockerfiles, and from raw source code using the Source-to-Image (S2I) process.

Run the `oc new-app -h` command to understand all the different options available for creating new applications on OpenShift.

The following command creates an application based on an image, `mysql`, from Docker Hub, with the label set to `db=mysql`:

```
[user@host ~]$ oc new-app mysql MYSQL_USER=user MYSQL_PASSWORD=pass
  MYSQL_DATABASE=testdb -l db=mysql
```

The following figure shows the Kubernetes and OpenShift resources created by the `oc new-app` command when the argument is a container image:



**Figure 6.6: Resources created for a new application**

The following command creates an application based on an image from a private Docker image registry:

```
oc new-app --docker-image=myregistry.com/mycompany/myapp --name=myapp
```

The following command creates an application based on source code stored in a Git repository:

```
oc new-app https://github.com/openshift/ruby-hello-world --name=ruby-hello
```

You will learn more about the Source-to-Image (S2I) process, its associated concepts, and more advanced ways to use `oc new-app` to build applications for OpenShift in the next section.

The following command creates an application based on an existing template:

```
$ oc new-app \
--template=mysql-persistent \
-p MYSQL_USER=user1 -p MYSQL_PASSWORD=mypa55 -p MYSQL_DATABASE=testdb \
-p MYSQL_ROOT_PASSWORD=r00tpa55 -p VOLUME_CAPACITY=10Gi
...output omitted...
```



### Note

You will learn more about templates on the next chapter.

## Managing Persistent Storage

In addition to the specification of custom images, you can create persistent storage and attach it to your application. In this way you can make sure your data is not lost when deleting your pods. To list the `PersistentVolume` objects in a cluster, use the `oc get pv` command:

```
[admin@host ~]$ oc get pv
NAME      CAPACITY   ACCESS MODES  RECLAIM POLICY  STATUS     CLAIM      ...
pv0001    1Mi        RWO          Retain        Available   ...
pv0002    10Mi       RWX          Recycle       Available   ...
...output omitted...
```

To see the YAML definition for a given `PersistentVolume`, use the `oc get` command with the `-o yaml` option:

```
[admin@host ~]$ oc get pv pv0001 -o yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  creationTimestamp: ...value omitted...
  finalizers:
  - kubernetes.io/pv-protection
  labels:
    type: local
    name: pv0001
  resourceVersion: ...value omitted...
  selfLink: /api/v1/persistentvolumes/pv0001
  uid: ...value omitted...
spec:
```

```
accessModes:
- ReadWriteOnce
capacity:
  storage: 1Mi
hostPath:
  path: /data/pv0001
  type: ""
persistentVolumeReclaimPolicy: Retain
status:
  phase: Available
```

To add more `PersistentVolume` objects to a cluster, use the `oc create` command:

```
[admin@host ~]$ oc create -f pv1001.yaml
```



### Note

The above `pv1001.yaml` file must contain a persistent volume definition, similar in structure to the output of the `oc get pv pv-name -o yaml` command.

## Requesting Persistent Volumes

When an application requires storage, you create a `PersistentVolumeClaim` (PVC) object to request a dedicated storage resource from the cluster pool. The following content from a file named `pvc.yaml` is an example definition for a PVC:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: myapp
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

The PVC defines storage requirements for the application, such as capacity or throughput. To create the PVC, use the `oc create` command:

```
[admin@host ~]$ oc create -f pvc.yaml
```

After you create a PVC, OpenShift attempts to find an available `PersistentVolume` resource that satisfies the PVC's requirements. If OpenShift finds a match, it binds the `PersistentVolume` object to the `PersistentVolumeClaim` object. To list the PVCs in a project, use the `oc get pvc` command:

```
[admin@host ~]$ oc get pvc
NAME      STATUS    VOLUME   CAPACITY  ACCESS MODES  STORAGECLASS  AGE
myapp    Bound     pv0001   1Gi       RWO          6s
```

The output indicates whether a persistent volume is bound to the PVC, along with attributes of the PVC (such as capacity).

To use the persistent volume in an application pod, define a volume mount for a container that references the `PersistentVolumeClaim` object. The application pod definition below references a `PersistentVolumeClaim` object to define a volume mount for the application:

```
apiVersion: "v1"
kind: "Pod"
metadata:
  name: "myapp"
  labels:
    name: "myapp"
spec:
  containers:
    - name: "myapp"
      image: openshift/myapp
      ports:
        - containerPort: 80
          name: "http-server"
      volumeMounts:
        - mountPath: "/var/www/html"
          name: "pvol" ①
  volumes:
    - name: "pvol" ②
      persistentVolumeClaim:
        claimName: "myapp" ③
```

- ① This section declares that the `pvol` volume mounts at `/var/www/html` in the container file system.
- ② This section defines the `pvol` volume.
- ③ The `pvol` volume references the `myapp` PVC. If OpenShift associates an available persistent volume to the `myapp` PVC, then the `pvol` volume refers to this associated volume.

## Managing OpenShift Resources at the Command Line

There are several essential commands used to manage OpenShift resources as described below.

Use the `oc get` command to retrieve information about resources in the cluster. Generally, this command outputs only the most important characteristics of the resources and omits more detailed information.

The `oc get RESOURCE_TYPE` command displays a summary of all resources of the specified type. The following illustrates example output of the `oc get pods` command.

NAME	READY	STATUS	RESTARTS	AGE
nginx-1-5r583	1/1	Running	0	1h
myapp-1-l44m7	1/1	Running	0	1h

## oc get all

Use the `oc get all` command to retrieve a summary of the most important components of a cluster. This command iterates through the major resource types for the current project and prints out a summary of their information:

NAME	DOCKER REPO	TAGS	UPDATED	
is/nginx	172.30.1.1:5000/basic-kubernetes/nginx	latest	About an hour ago	
NAME	REVISION	DESIRED	CURRENT	TRIGGERED BY
dc/nginx	1	1	1	config,image(nginx:latest)
NAME	DESIRED	CURRENT	READY	AGE
rc/nginx-1	1	1	1	1h
NAME	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
svc/nginx	172.30.72.75	<none>	80/TCP, 443/TCP	1h
NAME	READY	STATUS	RESTARTS	AGE
po/nginx-1-ypp8t	1/1	Running	0	1h

## oc describe

If the summaries provided by `oc get` are insufficient, use the `oc describe RESOURCE_TYPE RESOURCE_NAME` command to retrieve additional information. Unlike the `oc get` command, there is no way to iterate through all the different resources by type. Although most major resources can be described, this functionality is not available across all resources. The following is an example output from describing a pod resource:

Name:	mysql-openshift-1-glqrp
Namespace:	mysql-openshift
Priority:	0
Node:	cluster-worker-1/172.25.250.52
Start Time:	Fri, 15 Feb 2019 02:14:34 +0000
Labels:	app=mysql-openshift deployment=mysql-openshift-1
<i>...output omitted...</i>	
Status:	Running
IP:	10.129.0.85

## oc get

The `oc get RESOURCE_TYPE RESOURCE_NAME` command can be used to export a resource definition. Typical use cases include creating a backup, or to aid in the modification of a definition. The `-o yaml` option prints out the object representation in YAML format, but this can be changed to JSON format by providing a `-o json` option.

## oc create

This command creates resources from a resource definition. Typically, this is paired with the `oc get RESOURCE_TYPE RESOURCE_NAME -o yaml` command for editing definitions.

## oc edit

This command allows the user to edit resources of a resource definition. By default, this command opens a `vi` buffer for editing the resource definition.

## oc delete

The `oc delete RESOURCE_TYPE name` command removes a resource from an OpenShift cluster. Note that a fundamental understanding of the OpenShift architecture is needed here, because deleting managed resources such as pods results in new instances of those resources being automatically created. When a project is deleted, it deletes all of the resources and applications contained within it.

## oc exec

The `oc exec CONTAINER_ID options` command executes commands inside a container. You can use this command to run interactive and noninteractive batch commands as part of a script.

## Labelling Resources

When working with many resources in the same project, it is often useful to group those resources by application, environment, or some other criteria. To establish these groups, you define labels for the resources in your project. Labels are part of the `metadata` section of a resource, and are defined as key/value pairs, as shown in the following example:

```
apiVersion: v1
kind: Service
metadata:
  ...contents omitted...
  labels: app: nexus template: nexus-persistent-template
  name: nexus
  ...contents omitted...
```

Many `oc` subcommands support a `-l` option to process resources from a label specification. For the `oc get` command, the `-l` option acts as a selector to only retrieve objects that have a matching label:

```
$ oc get svc,deployments -l app=nexus
NAME          TYPE      CLUSTER-IP      EXTERNAL-IP      PORT(S)      AGE
service/nexus  ClusterIP  172.30.29.218  <none>           8081/TCP    4h

NAME                           REVISION      DESIRED      CURRENT      ...
deployment.apps.openshift.io/nexus  1            1            1            ...
```



### Note

Although any label can appear in resources, both the `app` and `template` keys are common for labels. By convention, the `app` key indicates the application related to this resource. The `template` key labels all resources generated by the same template with the template's name.

When using templates to generate resources, labels are especially useful. A template resource has a `labels` section separated from the `metadata.labels` section. Labels defined in the `labels`

section do not apply to the template itself, but are added to every resource generated by the template:

```
apiVersion: template.openshift.io/v1
kind: Template
labels: app: nexus template: nexus-persistent-template
metadata:
...contents omitted...
labels: maintainer: redhat
  name: nexus-persistent
...contents omitted...
objects:
- apiVersion: v1
  kind: Service
  metadata:
    name: nexus
labels: version: 1
...contents omitted...
```

The previous example defines a template resource with a single label: `maintainer: redhat`. The template generates a service resource with three labels: `app: nexus`, `template: nexus-persistent-template`, and `version: 1`.



## References

Additional information about pods and services is available in the *Pods and Services* section of the OpenShift Container Platform documentation:

### Architecture

[https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.6/html-single/architecture/index](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.6/html-single/architecture/index)

Additional information about creating images is available in the OpenShift Container Platform documentation:

### Creating Images

[https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.6/html-single/images/index](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.6/html-single/images/index)

Labels and label selectors details are available in *Working with Kubernetes Objects* section for the Kubernetes documentation:

### Labels and Selectors

<https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/>

## ► Guided Exercise

# Deploying a Database Server on OpenShift

In this exercise, you will create and deploy a MySQL database pod on OpenShift using the `oc new-app` command.

### Outcomes

You should be able to create and deploy a MySQL database pod on OpenShift.

### Before You Begin

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

- A running Red Hat OpenShift (RHOCP) cluster. See *Guided Exercise: Deploying an Application to Red Hat OpenShift Container Platform* for more information about using the developer sandbox RHOCP cluster.
- The `oc` command.

## Instructions

- 1. Log in with the `oc` utility.
- 1.1. Log in to the RHOCP web console.
  - 1.2. In the RHOCP web console, click your username at the top right, and then click **Copy Login Command**. This will open a new tab and prompt you for your username and password.
  - 1.3. Once authenticated, click **Display Token** and copy the provided `oc login` command.
  - 1.4. Log in to your RHOCP account by using the copied command and notice the default project:

```
$ oc login --token=yourtoken --server=https://api.region.prod.nextcle.com:6443
Logged into "https://api.region.prod.nextcle.com:6443" as "youruser" using the
token provided.
...output omitted...
Using project "youruser-dev".
```

- 2. Create a new application from the `mysql-persistent` template using the `oc new-app` command.

This image requires that you use the `-p` option to set the `MYSQL_USER`, `MYSQL_PASSWORD`, `MYSQL_DATABASE`, `MYSQL_ROOT_PASSWORD` and `VOLUME_CAPACITY` environment variables.

Use the `--template` option with the `oc new-app` command to specify a template with persistent storage so that OpenShift does not try and pull the image from the internet:

```
[student@workstation ~]$ oc new-app \
--template=mysql-persistent \
-p MYSQL_USER=user1 -p MYSQL_PASSWORD=mypa55 -p MYSQL_DATABASE=testdb \
-p MYSQL_ROOT_PASSWORD=r00tpa55 -p VOLUME_CAPACITY=10Gi
--> Deploying template "openshift/mysql-persistent" to project
${RHT_OCP4_DEV_USER}-mysql-openshift
...output omitted...
--> Creating resources ...
  secret "mysql" created
  service "mysql" created
  persistentvolumeclaim "mysql" created
  deploymentconfig.apps.openshift.io "mysql" created
--> Success
  Application is not exposed. You can expose services to the outside world by
executing one or more of the commands below:
  'oc expose service/mysql'
Run 'oc status' to view your app.
```

- 3. Verify that the MySQL pod was created successfully and view the details about the pod and its service.
- 3.1. Run the `oc status` command to view the status of the new application and verify that the deployment of the MySQL image was successful:

```
[student@workstation ~]$ oc status
In project youruser-dev on server ...

svc/mysql - 172.30.151.91:3306
...output omitted...
deployment #1 deployed 6 minutes ago - 1 pod
```

- 3.2. List the pods in this project to verify that the MySQL pod is ready and running:

```
[student@workstation ~]$ oc get pods
NAME          READY   STATUS    RESTARTS   AGE
mysql-1-5vfn4 1/1     Running   0          109s
```



#### Note

Notice the name of the running pod. You need this information to be able to log in to the MySQL database server later.

- 3.3. Use the `oc describe` command to view more details about the pod:

```
[student@workstation ~]$ oc describe pod mysql-1-5vfn4
Name:           mysql-1-5vfn4
Namespace:      youruser-dev
Priority:      0
Node:          master01/192.168.50.10
Start Time:    Mon, 29 Mar 2021 16:42:13 -0400
Labels:        deployment=mysql-1
```

```
...output omitted...
Status:      Running
IP:          10.10.0.34
...output omitted...
```

- 3.4. List the services in this project and verify that the service to access the MySQL pod was created:

```
[student@workstation ~]$ oc get svc
NAME            TYPE      CLUSTER-IP      EXTERNAL-IP      PORT(S)      AGE
mysql           ClusterIP   172.30.151.91   <none>        3306/TCP    10m
```

- 3.5. Retrieve the details of the mysql service using the `oc describe` command and note that the service type is `ClusterIP` by default:

```
[student@workstation ~]$ oc describe service mysql
Name:            mysql
Namespace:       ${RHT_OCP4_DEV_USER}-mysql-openshift
Labels:          app=mysql-persistent
                 app.kubernetes.io/component=mysql-persistent
                 app.kubernetes.io/instance=mysql-persistent
                 template=mysql-persistent-template
Annotations:     openshift.io/generated-by: OpenShiftNewApp
...output omitted...
Selector:        name=mysql
Type:            ClusterIP
IP:              172.30.151.91
Port:            3306-tcp  3306/TCP
TargetPort:      3306/TCP
Endpoints:       10.10.0.34:3306
Session Affinity: None
Events:          <none>
```

- 3.6. List the persistent storage claims in this project:

```
[student@workstation ~]$ oc get pvc
NAME      STATUS      VOLUME                                     CAPACITY  ...
STORAGECLASS
mysql     Bound       pvc-e9bf0b1f-47df-4500-afb6-77e826f76c15  10Gi     ...
standard
```

- 3.7. Retrieve the details of the mysql pvc using the `oc describe` command:

```
[student@workstation ~]$ oc describe pvc/mysql
Name:            mysql
Namespace:       ${RHT_OCP4_DEV_USER}-mysql-openshift
StorageClass:    standard
Status:          Bound
Volume:          pvc-e9bf0b1f-47df-4500-afb6-77e826f76c15
Labels:          app=mysql-persistent
                 app.kubernetes.io/component=mysql-persistent
                 app.kubernetes.io/instance=mysql-persistent
```

```
template=mysql-persistent-template
Annotations:      openshift.io/generated-by: OpenShiftNewApp
...output omitted...
Capacity:        10Gi
Access Modes:    RWO
VolumeMode:     Filesystem
Mounted By:     mysql-1-5vfn4
```

► **4.** Verify that the MySQL database was created successfully.

- 4.1. Get the MySQL pod name:

```
[student@workstation ~]$ oc get pods
NAME          READY   STATUS    RESTARTS   AGE
mysql-1-5vfn4 1/1     Running   0          18m
mysql-1-deploy 0/1     Completed  0          18m
```

- 4.2. Issue the `show databases;` SQL query to the MySQL pod:

```
[student@workstation ~]$ oc exec mysql-1-5vfn4 -- mysql -uuser1 -pmypa55 \
--protocol tcp -h localhost -e 'show databases;'
Database
information_schema
testdb
```

► **5.** Clean up.

- 5.1. Remove RHOCP resources with the `app=mysql-persistent` label.

```
[student@workstation ~]$ oc delete all,secrets,pvc -l app=mysql-persistent
replicationcontroller "mysql-1" deleted
service "mysql" deleted
deploymentconfig.apps.openshift.io "mysql" deleted
secret "mysql" deleted
persistentvolumeclaim "mysql" deleted
```

## Finish

This concludes the exercise.

# Creating Routes

---

## Objectives

After completing this section, students should be able to expose services using OpenShift routes.

## Working with Routes

Services allow for network access between pods inside an OpenShift instance, and routes allow for network access to pods from users and applications outside the OpenShift instance.

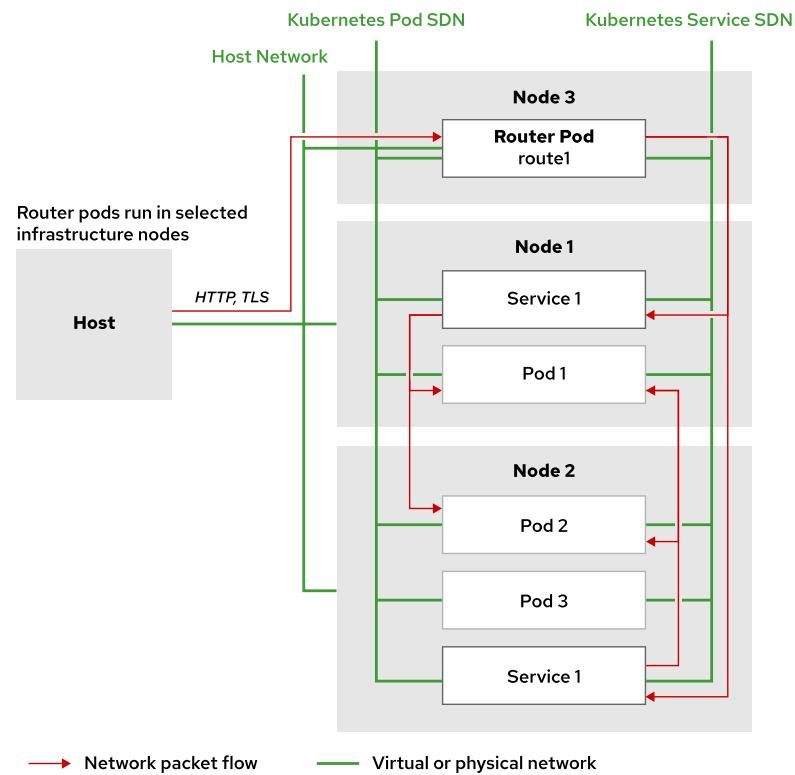


Figure 6.7: OpenShift routes and Kubernetes services

A route connects a public-facing IP address and DNS host name to an internal-facing service IP. It uses the service resource to find the endpoints; that is, the ports exposed by the service.

OpenShift routes are implemented by a cluster-wide router service, which runs as a containerized application in the OpenShift cluster. OpenShift scales and replicates router pods like any other OpenShift application.

**Note**

In practice, to improve performance and reduce latency, the OpenShift router connects directly to the pods using the internal pod software-defined network (SDN).

The router service uses *HAProxy* as the default implementation.

An important consideration for OpenShift administrators is that the public DNS host names configured for routes need to point to the public-facing IP addresses of the nodes running the router. Router pods, unlike regular application pods, bind to their nodes' public IP addresses instead of to the internal pod SDN.

The following example shows a minimal route defined using JSON syntax:

```
{
  "apiVersion": "v1",
  "kind": "Route",
  "metadata": {
    "name": "quoteapp"
  },
  "spec": {
    "host": "quoteapp.apps.example.com",
    "to": {
      "kind": "Service",
      "name": "quoteapp"
    }
  }
}
```

The `apiVersion`, `kind`, and `metadata` attributes follow standard Kubernetes resource definition rules. The `Route` value for `kind` shows that this is a route resource, and the `metadata.name` attribute gives this particular route the identifier `quoteapp`.

As with pods and services, the main part is the `spec` attribute, which is an object containing the following attributes:

- `host` is a string containing the FQDN associated with the route. DNS must resolve this FQDN to the IP address of the OpenShift router. The details to modify DNS configuration are outside the scope of this course.
- `to` is an object stating the resource this route points to. In this case, the route points to an OpenShift Service with the name set to `quoteapp`.

**Note**

Names of different resource types do not collide. It is perfectly legal to have a route named `quoteapp` that points to a service also named `quoteapp`.

**Important**

Unlike services, which use selectors to link to pod resources containing specific labels, a route links directly to the service resource name.

## Creating Routes

Use the `oc create` command to create route resources, just like any other OpenShift resource. You must provide a JSON or YAML resource definition file, which defines the route, to the `oc create` command.

The `oc new-app` command does not create a route resource when building a pod from container images, Dockerfiles, or application source code. After all, `oc new-app` does not know if the pod is intended to be accessible from outside the OpenShift instance or not.

Another way to create a route is to use the `oc expose service` command, passing a service resource name as the input. The `--name` option can be used to control the name of the route resource. For example:

```
$ oc expose service quotedb --name quote
```

By default, routes created by `oc expose` generate DNS names of the form: `route-name-project-name.default-domain`

Where:

- `route-name` is the name assigned to the route. If no explicit name is set, OpenShift assigns the route the same name as the originating resource (for example, the service name).
- `project-name` is the name of the project containing the resource.
- `default-domain` is configured on the OpenShift Control Plane and corresponds to the wildcard DNS domain listed as a prerequisite for installing OpenShift.

For example, creating a route named `quote` in project named `test` from an OpenShift instance where the wildcard domain is `cloudapps.example.com` results in the FQDN `quote-test.cloudapps.example.com`.



### Note

The DNS server that hosts the wildcard domain knows nothing about route host names. It merely resolves any name to the configured IP addresses. Only the OpenShift router knows about route host names, treating each one as an HTTP virtual host. The OpenShift router blocks invalid wildcard domain host names that do not correspond to any route and returns an HTTP 404 error.

## Leveraging the Default Routing Service

The default routing service is implemented as an HAProxy pod. Router pods, containers, and their configuration can be inspected just like any other resource in an OpenShift cluster:

```
$ oc get pod --all-namespaces | grep router
openshift-ingress   router-default-746b5cfb65-f6sdm 1/1     Running   1          4d
```

Note that you can query information on the default router using the associated label as shown here.

By default, router is deployed in `openshift-ingress` project. Use `oc describe pod` command to get the routing configuration details:

```
$ oc describe pod router-default-746b5cfb65-f6sdm
Name:           router-default-746b5cfb65-f6sdm
Namespace:      openshift-ingress
...output omitted...
Containers:
  router:
  ...output omitted...
    Environment:
      STATS_PORT:          1936
      ROUTER_SERVICE_NAMESPACE:  openshift-ingress
      DEFAULT_CERTIFICATE_DIR: /etc/pki/tls/private
      ROUTER_SERVICE_NAME:    default
      ROUTER_CANONICAL_HOSTNAME: apps.cluster.lab.example.com
...output omitted...
```

The subdomain, or default domain to be used in all default routes, takes its value from the ROUTER\_CANONICAL\_HOSTNAME entry.



## References

Additional information about the architecture of routes in OpenShift is available in the *Architecture* and *Developer Guide* sections of the

### **OpenShift Container Platform documentation.**

[https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.6/](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.6/)

## ► Guided Exercise

# Exposing a Service as a Route

In this exercise, you will create, build, and deploy an application on an OpenShift cluster and expose its service as a route.

### Outcomes

You should be able to expose a service as a route for a deployed OpenShift application.

### Before You Begin

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

- A running Red Hat OpenShift (RHOCP) cluster. See *Guided Exercise: Deploying an Application to Red Hat OpenShift Container Platform* for more information about using the developer sandbox RHOCP cluster.
- The oc command.

## Instructions

- 1. Log in with the oc utility.
- 1.1. Log in to the RHOCP web console.
  - 1.2. In the RHOCP web console, click your username at the top right, and then click **Copy Login Command**. This will open a new tab and prompt you for your username and password.
  - 1.3. Once authenticated, click **Display Token** and copy the provided oc login command.
  - 1.4. Log in to your RHOCP account by using the copied command and notice the default project:

```
$ oc login --token=yourtoken --server=https://api.region.prod.nextcle.com:6443
Logged into "https://api.region.prod.nextcle.com:6443" as "youruser" using the
token provided.
...output omitted...
Using project "youruser-dev".
```

- 2. Create a new PHP application using the quay.io/redhattraining/php-hello-dockerfile image.
- 2.1. Use the oc new-app command to create the PHP application.



### Important

The following example uses a backslash (\) to indicate that the second line is a continuation of the first line. If you wish to ignore the backslash, you can type the entire command in one line.

```
[student@workstation ~]$ oc new-app \
--docker-image=quay.io/redhattraining/php-hello-dockerfile \
--name php-helloworld
--> Found container image 4b696cc (2 years old) from quay.io for "quay.io/redhattraining/php-hello-dockerfile"
...output omitted...
--> Creating resources ...
...output omitted...
--> Success
Application is not exposed. You can expose services to the outside world by
executing one or more of the commands below:
'oc expose service/php-helloworld'
Run 'oc status' to view your app.
```

- 2.2. Wait until the application finishes deploying by monitoring the progress with the `oc get pods -w` command:

```
[student@workstation ~]$ oc get pods -w
NAME                  READY   STATUS    RESTARTS   AGE
php-helloworld-74bb86f6cb-zt6wl   1/1     Running   0          5s
^C
```

Your exact output may differ in names, status, timing, and order. The container in `Running` status with a random suffix (`74bb86f6cb-zt6wl` in the example) contains the application and shows it is up and running. Alternatively, monitor the deployment logs with the `oc logs -f php-helloworld-74bb86f6cb-zt6wl` command. Press `Ctrl + C` to exit the command if necessary.

```
[student@workstation ~]$ oc logs -f php-helloworld-74bb86f6cb-zt6wl
AH00558: httpd: Could not reliably determine the server's fully qualified domain
           name, using 10.129.5.124. Set the 'ServerName' directive globally to suppress
           this message
[09-Aug-2021 22:09:45] NOTICE: [pool www] 'user' directive is ignored when FPM is
           not running as root
[09-Aug-2021 22:09:45] NOTICE: [pool www] 'group' directive is ignored when FPM is
           not running as root
^C
```

Your exact output may differ.

- 2.3. Review the service for this application using the `oc describe` command:

```
[student@workstation ~]$ oc describe svc/php-helloworld
Name:            php-helloworld
Namespace:       youruser-dev
Labels:          app=php-helloworld
                 app.kubernetes.io/component=php-helloworld
                 app.kubernetes.io/instance=php-helloworld
Annotations:    openshift.io/generated-by: OpenShiftNewApp
Selector:        deployment=php-helloworld
Type:            ClusterIP
IP:              172.30.100.236
```

```
Port:          8080-tcp  8080/TCP
TargetPort:    8080/TCP
Endpoints:    10.129.5.124:8080
Session Affinity: None
Events:        <none>
```

The IP address and namespace displayed in the output of the command may differ.

- 3. Expose the service, which creates a route. Use the default name and fully qualified domain name (FQDN) for the route:

```
[student@workstation ~]$ oc expose svc/php-helloworld
route.route.openshift.io/php-helloworld exposed
```

- 3.1. Verify the host name of the route:

```
[student@workstation ~]$ oc describe route
Name:          php-helloworld
Namespace:     youruser-dev
Created:       16 seconds ago
Labels:        app=php-helloworld
               app.kubernetes.io/component=php-helloworld
               app.kubernetes.io/instance=php-helloworld
Annotations:   openshift.io/host.generated=true
Requested Host: php-helloworld-youruser-dev.apps.sandbox-
m2.ll9k.p1.openshiftapps.com
               exposed on router default (host router-default.apps.sandbox-
m2.ll9k.p1.openshiftapps.com) 16 seconds ago
Path:          <none>
TLS Termination: <none>
Insecure Policy: <none>
Endpoint Port:  8080-tcp

Service:      php-helloworld
Weight:       100 (100%)
Endpoints:    10.129.5.124:8080
```

- 4. Access the service in a web browser to verify that the service and route are working.

For example, to use Firefox, execute:

```
[student@workstation ~]$ firefox \
php-helloworld-youruser-dev-route.apps.na46-stage2.dev.nextcle.com
```

Notice the FQDN is comprised of the application name and project name by default. The remainder of the FQDN, the subdomain, is defined when OpenShift is installed.

- 5. Replace this route with a route named xyz.

- 5.1. Delete the current route:

```
[student@workstation ~]$ oc delete route/php-helloworld
route.route.openshift.io "php-helloworld" deleted
```

**Note**

Deleting the route is optional. You can have multiple routes for the same service, provided they have different names.

- 5.2. Create a route for the service with a name of `youruser-xyz`. Replace `youruser` with your user.

```
[student@workstation ~]$ oc expose svc/php-helloworld \
--name=youruser-xyz
route.route.openshift.io/youruser-xyz exposed
```

- 5.3. Verify the host name of the route:

```
[student@workstation ~]$ oc describe route
Name:           youruser-xyz
Namespace:      youruser-dev
Created:        23 seconds ago
Labels:         app=php-helloworld
                app.kubernetes.io/component=php-helloworld
                app.kubernetes.io/instance=php-helloworld
Annotations:   openshift.io/host.generated=true
Requested Host: youruser-xyz-youruser-dev.apps.sandbox-
m2.ll9k.p1.openshiftapps.com
                  exposed on router default (host router-default.apps.sandbox-
m2.ll9k.p1.openshiftapps.com) 22 seconds ago
Path:           <none>
TLS Termination: <none>
Insecure Policy: <none>
Endpoint Port:  8080-tcp

Service:       php-helloworld
Weight:        100 (100%)
Endpoints:    10.129.5.124:8080
```

Your exact output may differ. Note the new FQDN that was generated based on the new route name. Both the route name and the project name contain your user name, hence it appears twice in the route FQDN.

- 5.4. Access the route host in a web browser.

For example:

```
[student@workstation ~]$ firefox \
youruser-xyz-youruser-dev.apps.sandbox-m2.ll9k.p1.openshiftapps.com
```

► 6. Clean up.

- 6.1. Remove RHOCP resources with the `app=php-helloworld` label.

```
[student@workstation ~]$ oc delete all -l app=php-helloworld
service "php-helloworld" deleted
deployment.apps "php-helloworld" deleted
imagestream.image.openshift.io "php-helloworld" deleted
route.route.openshift.io "youruser-xyz" deleted
```

## Finish

This concludes the guided exercise.

# Creating Applications with Source-to-Image

---

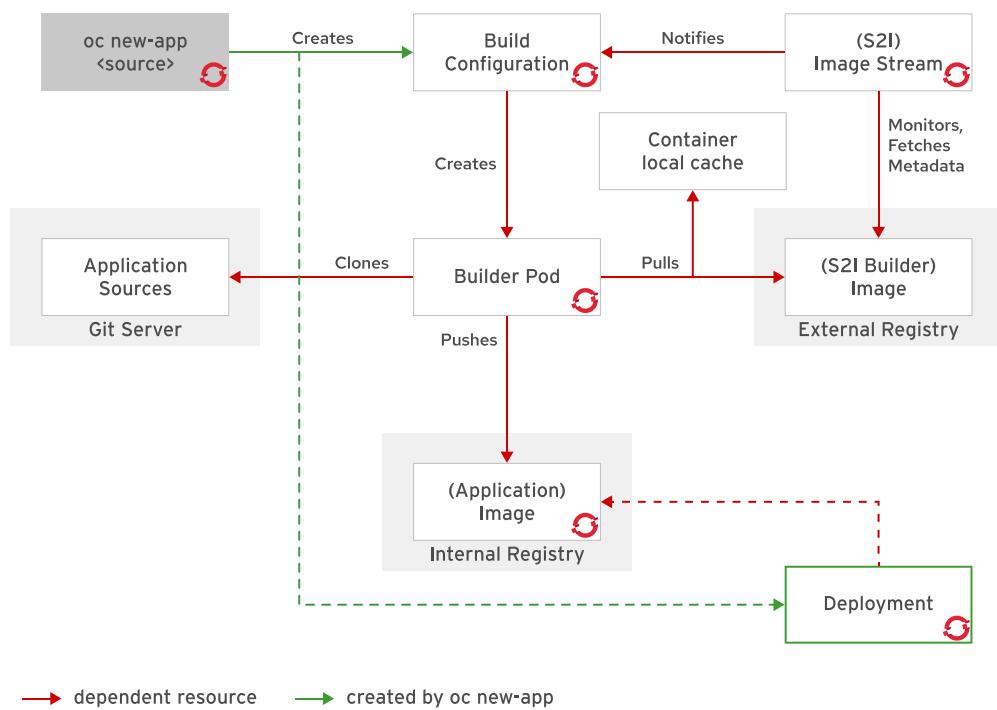
## Objectives

After completing this section, students should be able to deploy an application using the Source-to-Image (S2I) facility of OpenShift Container Platform.

## The Source-to-Image (S2I) Process

Source-to-Image (S2I) is a tool that makes it easy to build container images from application source code. This tool takes an application's source code from a Git repository, injects the source code into a base container based on the language and framework desired, and produces a new container image that runs the assembled application.

This figure shows the resources created by the `oc new-app <source>` command when the argument is an application source code repository. Notice that S2I also creates a Deployment and all its dependent resources:



**Figure 6.8: Deployment and dependent resources**

S2I is the primary strategy used for building applications in OpenShift Container Platform. The main reasons for using source builds are:

- User efficiency: Developers do not need to understand Dockerfiles and operating system commands such as `yum install`. They work using their standard programming language tools.

- Patching: S2I allows for rebuilding all the applications consistently if a base image needs a patch due to a security issue. For example, if a security issue is found in a PHP base image, then updating this image with security patches updates all applications that use this image as a base.
- Speed: With S2I, the assembly process can perform a large number of complex operations without creating a new layer at each step, resulting in faster builds.
- Ecosystem: S2I encourages a shared ecosystem of images where base images and scripts can be customized and reused across multiple types of applications.

## Describing Image Streams

OpenShift deploys new versions of user applications into pods quickly. To create a new application, in addition to the application source code, a base image (the S2I builder image) is required. If either of these two components gets updated, OpenShift creates a new container image. Pods created using the older container image are replaced by pods using the new image.

Even though it is evident that the container image needs to be updated when application code changes, it may not be evident that the deployed pods also need to be updated should the builder image change.

The image stream resource is a configuration that names specific container images associated with image stream tags, an alias for these container images. OpenShift builds applications against an image stream. The OpenShift installer populates several image streams by default during installation. To determine available image streams, use the `oc get` command, as follows:

```
$ oc get is -n openshift
NAME          IMAGE REPOSITORY           TAGS
cli           ...svc:5000/openshift/cli   latest
dotnet        ...svc:5000/openshift/dotnet  2.1,...,3.1-el7,latest
dotnet-runtime ...svc:5000/openshift/dotnet-runtime  2.1,...,3.1-el7,latest
httpd         ...svc:5000/openshift/httpd    2.4,2.4-el7,2.4-el8,latest
jenkins       ...svc:5000/openshift/jenkins  2,latest
mariadb       ...svc:5000/openshift/mariadb  10.3,10.3-el7,10.3-el8,latest
mongodb       ...svc:5000/openshift/mongodb  3.6,latest
mysql         ...svc:5000/openshift/mysql    8.0,8.0-el7,8.0-el8,latest
nginx         ...svc:5000/openshift/nginx    1.10,1.12,1.8,latest
nodejs        ...svc:5000/openshift/nodejs   10,...,12-ubi7,12-ubi8
perl          ...svc:5000/openshift/perl    5.26,...,5.30,5.30-el7
php           ...svc:5000/openshift/php     7.2-ubi8,...,7.3-ubi8,latest
postgresql    ...svc:5000/openshift/postgresql 10,10-el7,...,12-el7,12-el8
python        ...svc:5000/openshift/python   2.7,2.7-ubi7,...,3.6-ubi8,3.8
redis         ...svc:5000/openshift/redis   5,5-el7,5-el8,latest
ruby          ...svc:5000/openshift/ruby    2.5,2.5-ubi7,...,2.6,2.6-ubi7
...
```



### Note

Your OpenShift instance may have more or fewer image streams depending on local additions and OpenShift point releases.

OpenShift detects when an image stream changes and takes action based on that change. If a security issue arises in the `rhel8/nodejs-10` image, it can be updated in the image repository, and OpenShift can automatically trigger a new build of the application code.

It is likely that an organization chooses several supported base S2I images from Red Hat, but may also create their own base images.

## Building an Application with S2I and the CLI

Building an application with S2I can be accomplished using the OpenShift CLI.

An application can be created using the S2I process with the `oc new-app` command from the CLI:

```
$ oc new-app php~http://my.git.server.com/my-app \ ①  
--name=myapp ②
```

- ① The image stream used in the process appears to the left of the tilde (~) character.  
The URL after the tilde indicates the location of the source code's Git repository.
- ② Sets the application name.



### Note

Instead of using the tilde character, you can set the image stream by using the `-i` option or `--image-stream` for the full version.

```
$ oc new-app -i php http://services.lab.example.com/app --name=myapp
```

The `oc new-app` command allows creating applications using source code from a local or remote Git repository. If only a source repository is specified, `oc new-app` tries to identify the correct image stream to use for building the application. In addition to application code, S2I can also identify and process Dockerfiles to create a new image.

The following example creates an application using the Git repository in the current directory:

```
$ oc new-app .
```



### Important

When using a local Git repository, the repository must have a remote origin that points to a URL accessible by the OpenShift instance.

It is also possible to create an application using a remote Git repository and a context subdirectory:

```
$ oc new-app https://github.com/openshift/sti-ruby.git \  
--context-dir=2.0/test/puma-test-app
```

Finally, it is possible to create an application using a remote Git repository with a specific branch reference:

```
$ oc new-app https://github.com/openshift/ruby-hello-world.git#beta4
```

If an image stream is not specified in the command, `new-app` attempts to determine which language builder to use based on the presence of certain files in the root of the repository:

Language	Files
Ruby	<code>Rakefile Gemfile config.ru</code>
Java EE	<code>pom.xml</code>
Node.js	<code>app.json package.json</code>
PHP	<code>index.php composer.json</code>
Python	<code>requirements.txt config.py</code>
Perl	<code>index.pl cpanfile</code>

After a language is detected, the `new-app` command searches for image stream tags that have support for the detected language, or an image stream that matches the name of the detected language.

Create a JSON resource definition file by using the `-o json` parameter and output redirection:

```
$ oc -o json new-app php~http://services.lab.example.com/app \
--name=myapp > s2i.json
```

This JSON definition file creates a list of resources. The first resource is the image stream:

```
...output omitted...
{
    "kind": "ImageStream", ❶
    "apiVersion": "image.openshift.io/v1",
    "metadata": {
        "name": "myapp", ❷
        "creationTimestamp": null
        "labels": {
            "app": "myapp"
        },
        "annotations": {
            "openshift.io/generated-by": "OpenShiftNewApp"
        }
    },
    "spec": {
        "lookupPolicy": {
            "local": false
        }
    },
    "status": {
        "dockerImageRepository": ""
    }
},
...output omitted...
```

- ❶ Define a resource type of image stream.

- ❷ Name the image stream `myapp`.

The build configuration (`bc`) is responsible for defining input parameters and triggers that are executed to transform the source code into a runnable image. The `BuildConfig` (`BC`) is the second resource, and the following example provides an overview of the parameters used by OpenShift to create a runnable image.

```
...output omitted...
{
  "kind": "BuildConfig", ❶
  "apiVersion": "build.openshift.io/v1",
  "metadata": {
    "name": "myapp", ❷
    "creationTimestamp": null,
    "labels": {
      "app": "myapp"
    },
    "annotations": {
      "openshift.io/generated-by": "OpenShiftNewApp"
    }
  },
  "spec": {
    "triggers": [
      {
        "type": "GitHub",
        "github": {
          "secret": "S5_4BZpPabM6KrIuPBvI"
        }
      },
      {
        "type": "Generic",
        "generic": {
          "secret": "3q8K8JNDoRzhjoz1KgMz"
        }
      },
      {
        "type": "ConfigChange"
      },
      {
        "type": "ImageChange",
        "imageChange": {}
      }
    ],
    "source": {
      "type": "Git",
      "git": {
        "uri": "http://services.lab.example.com/app" ❸
      }
    },
    "strategy": {
      "type": "Source", ❹
      "sourceStrategy": {
        "from": {
          "kind": "ImageStreamTag",
          "namespace": "openshift",
        }
      }
    }
  }
}
```

```

        "name": "php:7.3" ⑤
    }
}
},
"output": {
    "to": {
        "kind": "ImageStreamTag",
        "name": "myapp:latest" ⑥
    }
},
"resources": {},
"postCommit": {},
"nodeSelector": null
},
"status": {
    "lastVersion": 0
}
},
...output omitted...

```

- ① Define a resource type of `BuildConfig`.
- ② Name the `BuildConfig` `myapp`.
- ③ Define the address to the source code Git repository.
- ④ Define the strategy to use S2I.
- ⑤ Define the builder image as the `php:7.3` image stream.
- ⑥ Name the output image stream `myapp:latest`.

The third resource is the deployment object that is responsible for customizing the deployment process in OpenShift. It may include parameters and triggers that are necessary to create new container instances, and are translated into a replication controller from Kubernetes. Some of the features provided by Deployment objects are:

- User customizable strategies to transition from the existing deployments to new deployments.
- Have as many active replica set as wanted and possible.
- Replication scaling depends of the sizes of old and new replica sets.

```

...output omitted...
{
    "kind": "Deployment", ①
    "apiVersion": "apps/v1",
    "metadata": {
        "name": "myapp", ②
        "creationTimestamp": null,
        "labels": {
            "app": "myapp",
            "app.kubernetes.io/component": "myapp",
            "app.kubernetes.io/instance": "myapp"
        },
        "annotations": {

```

```

        "image.openshift.io/triggers": "[{\\"from\\":{\\"kind\\":
        \"ImageStreamTag\",\\\"name\\\":\\\"myapp:1
        atest\\\"},\\\"fieldPath\\\":\\\"spec.template.spec.containers[?(@.name==\\\\\"myapp\\\
        \\\")].image\\\"}]", ❸
            "openshift.io/generated-by": "OpenShiftNewApp"
        }
    },
    "spec": {
        "replicas": 1,
        "selector": {
            "matchLabels": {
                "deployment": "myapp"
            }
        },
        "template": {
            "metadata": {
                "creationTimestamp": null,
                "labels": {
                    "deployment": "myapp" ❹
                },
                "annotations": {
                    "openshift.io/generated-by": "OpenShiftNewApp"
                }
            },
            "spec": {
                "containers": [
                    {
                        "name": "myapp", ❺
                        "image": " ", ❻
                        "ports": [
                            {
                                "containerPort": 8080,
                                "protocol": "TCP"
                            },
                            {
                                "containerPort": 8443,
                                "protocol": "TCP"
                            }
                        ],
                        "resources": {}
                    }
                ]
            },
            "strategy": {}
        },
        "status": {}
    },
    ...output omitted...

```

- ❶ Define a resource type of Deployment.
- ❷ Name the Deployment myapp.

- ③ An image change trigger causes the creation of a new deployment each time a new version of the `myapp:latest` image is available in the repository.
- ④ A configuration change trigger causes a new deployment to be created any time the replication controller template changes.
- ⑤ Defines the container image to deploy: `myapp:latest`.
- ⑥ Specifies the container ports. A configuration change trigger causes a new deployment to be created any time the replication controller template changes.

The last item is the service, already covered in previous chapters:

```
...output omitted...
{
  "kind": "Service",
  "apiVersion": "v1",
  "metadata": {
    "name": "myapp",
    "creationTimestamp": null,
    "labels": {
      "app": "myapp"
      "app.kubernetes.io/component": "myapp",
      "app.kubernetes.io/instance": "myapp"
    },
    "annotations": {
      "openshift.io/generated-by": "OpenShiftNewApp"
    }
  },
  "spec": {
    "ports": [
      {
        "name": "8080-tcp",
        "protocol": "TCP",
        "port": 8080,
        "targetPort": 8080
      },
      {
        "name": "8443-tcp",
        "protocol": "TCP",
        "port": 8443,
        "targetPort": 8443
      }
    ],
    "selector": {
      "deployment": "myapp"
    }
  },
  "status": {
    "loadBalancer": {}
  }
}
```

**Note**

By default, the `oc new-app` command does not create a route. You can create a route after creating the application. However, a route is automatically created when using the web console because it uses a template.

After creating a new application, the build process starts. Use the `oc get builds` command to see a list of application builds:

```
$ oc get builds
NAME          TYPE      FROM      STATUS    STARTED      DURATION
php-helloworld-1  Source   Git@9e17db8  Running  13 seconds ago
```

OpenShift allows viewing the build logs. The following command shows the last few lines of the build log:

```
$ oc logs build/myapp-1
```

**Important**

If the build is not Running yet, or OpenShift has not deployed the `s2i-build` pod yet, the above command throws an error. Just wait a few moments and retry it.

Trigger a new build with the `oc start-build build_config_name` command:

```
$ oc get buildconfig
NAME          TYPE      FROM      LATEST
myapp        Source   Git        1
```

```
$ oc start-build myapp
build "myapp-2" started
```

## Relationship Between Build and Deployment

The `BuildConfig` pod is responsible for creating the images in OpenShift and pushing them to the internal container registry. Any source code or content update typically requires a new build to guarantee the image is updated.

The `Deployment` pod is responsible for deploying pods to OpenShift. The outcome of a `Deployment` pod execution is the creation of pods with the images deployed in the internal container registry. Any existing running pod may be destroyed, depending on how the `Deployment` resource is set.

The `BuildConfig` and `Deployment` resources do not interact directly. The `BuildConfig` resource creates or updates a container image. The `Deployment` reacts to this new image or updated image event and creates pods from the container image.



## References

### Source-to-Image (S2I) Build

[https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/4.6/html-single/builds/build-strategies#builds-strategy-s2i-build\\_understanding-image-builds](https://access.redhat.com/documentation/en-us/openshift_container_platform/4.6/html-single/builds/build-strategies#builds-strategy-s2i-build_understanding-image-builds)

### S2I GitHub repository

<https://github.com/openshift/source-to-image>

## ► Guided Exercise

# Creating a Containerized Application with Source-to-Image

In this exercise, you will build an application from source code and deploy the application to an OpenShift cluster.

## Outcomes

You should be able to:

- Build an application from source code using the OpenShift command-line interface.
- Verify the successful deployment of the application using the OpenShift command-line interface.

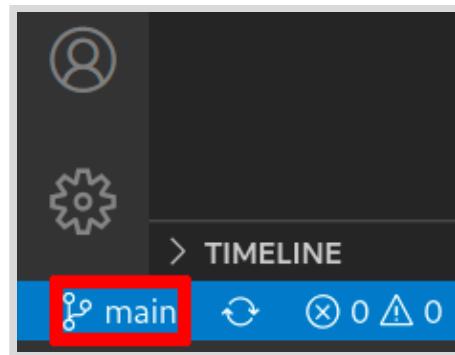
## Before You Begin

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

- A running Red Hat OpenShift (RHOCP) cluster. See *Guided Exercise: Deploying an Application to Red Hat OpenShift Container Platform* for more information about using the developer sandbox RHOCP cluster.
- The source code for the `php-helloworld` application in the `D0101x-apps` Git repository on your local system.
- The `oc` command.

## Instructions

- 1. Create and push a new branch named `s2i` to use during this exercise.
- 1.1. Launch the Visual Studio Code (VS Code) editor. In the Explorer view (**View > Explorer**), open the `D0101x-apps` folder in the `My Projects` workspace. The source code for the `version` application is in the `version` directory.
  - 1.2. Ensure that the `D0101x-apps` repository uses the `main` branch. If you were working with another branch for a different exercise, click the current branch and then select `main` in the `Select a ref to checkout` window to switch to the `main` branch.



**Figure 6.9: Select the main branch**



### Warning

Each exercise uses a unique branch. Always change to the **main** branch before you create a new branch. The new branch must use the **main** branch as the base.

- 1.3. Click the **main** branch in VS Code.  
Select **Create new branch...** from the list of options.
  - 1.4. At the prompt, enter **s2i** for the branch name.
  - 1.5. Push the **s2i** branch to your **D0101x-apps** GitHub repository.  
Click the **Publish Changes** cloud icon next to the **update-app** branch name to push your local branch to your remote Git repository.
- 2. Review the PHP source code of the application, inside the **php-helloworld** directory. Open the **index.php** file:

```
<?php  
print "Hello, World! php version is " . PHP_VERSION . "\n";  
?>
```

The application implements a simple response which returns the PHP version it is running.

- 3. Log in with the **oc** utility.
- 3.1. Log in to the RHOCP web console.
  - 3.2. In the RHOCP web console, click your username at the top right, and then click **Copy Login Command**. This will open a new tab and prompt you for your username and password.
  - 3.3. Once authenticated, click **Display Token** and copy the provided **oc login** command.
  - 3.4. Log in to your RHOCP account by using the copied command and notice the default project:

```
$ oc login --token=yourtoken --server=https://api.region.prod.nextcle.com:6443
Logged into "https://api.region.prod.nextcle.com:6443" as "youruser" using the
token provided.
...output omitted...
Using project "youruser-dev".
```

- 4. Create a new PHP application by using the Source-to-Image deployment method. Use the `php-helloworld` directory in the `s2i` branch you created in the previous step in your fork of the `D0101x-apps` Git repository.

- 4.1. Use the `oc new-app` command to create the PHP application.



### Important

The following example uses the number sign (#) to select a specific branch from the git repository, in this case the `s2i` branch created in the previous step.

```
[student@workstation ~]$ oc new-app php:latest --name=php-helloworld \
  https://github.com/YOUR_GITHUB_USERNAME/D0101x-apps#s2i \
  --context-dir php-helloworld
--> Found image 576cf8b (7 months old) in image stream "openshift/php" under tag
  "latest" for "php:latest"
...output omitted...
```

- 4.2. Wait for the build to complete and the application to deploy. Verify that the build process starts with the `oc get pods` command.

```
[student@workstation ~]$ oc get pods
NAME           READY   STATUS    RESTARTS   AGE
php-helloworld-1-build   1/1     Running   0          5s
```

- 4.3. Examine the logs for this build. Use the build pod name for this build, `php-helloworld-1-build`.

```
[student@workstation ~]$ oc logs -f php-helloworld-1-build
...output omitted...

Writing manifest to image destination
Storing signatures
Generating dockerfile with builder image image-registry.openshift-image-...
php@sha256:3206...37b4
Adding transient rw bind mount for /run/secrets/rhsm
STEP 1: FROM image-registry.openshift-image-registry.svc:5000...
...output omitted...

STEP 8: RUN /usr/libexec/s2i/assemble
...output omitted...
```

```

Pushing image .../php-helloworld:latest ...
Getting image source signatures

...output omitted...

Writing manifest to image destination
Storing signatures
Successfully pushed .../php-helloworld@sha256:3f1c...c454
Push successful
Cloning "https://github.com/YOUR_GITHUB_USER/D0101X-apps" ...
...output omitted...

```

Notice that the clone of the Git repository is the first step of the build. Next, the Source-to-Image process built a new image called s2i/php-helloworld:latest. The last step in the build process is to push this image to the OpenShift private registry.

#### 4.4. Review the Deployment for this application:

```

[student@workstation ~]$ oc describe deployment/php-helloworld
Name:                  php-helloworld
Namespace:             youruser-dev
CreationTimestamp:     Tue, 30 Mar 2021 12:54:59 -0400
Labels:                app=php-helloworld
                       app.kubernetes.io/component=php-helloworld
                       app.kubernetes.io/instance=php-helloworld
Annotations:           deployment.kubernetes.io/revision: 2
                       image.openshift.io/triggers:
                       [{"from": {"kind": "ImageStreamTag", "name": "php-
helloworld:latest"}, "fieldPath": "spec.template.spec.containers[?(@.name==\"php-
helloworld\")]..."}
                        openshift.io/generated-by: OpenShiftNewApp
Selector:              deployment=php-helloworld
Replicas:               1 desired | 1 updated | 1 total | 1 available | 0
                       unavailable
StrategyType:          RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:                deployment=php-helloworld
  Annotations:           openshift.io/generated-by: OpenShiftNewApp
  Containers:
    php-helloworld:
      Ports:                8080/TCP, 8443/TCP
      Host Ports:           0/TCP, 0/TCP
      Environment:          <none>
      Mounts:               <none>
      Volumes:              <none>
  Conditions:
    Type      Status  Reason
    ----      -----  -----
    Available  True    MinimumReplicasAvailable
    Progressing True    NewReplicaSetAvailable

```

```
OldReplicaSets: <none>
NewReplicaSet: php-helloworld-6f5d4c47ff (1/1 replicas created)
...output omitted...
```

- 4.5. Add a route to test the application:

```
[student@workstation ~]$ oc expose service php-helloworld
route.route.openshift.io/php-helloworld exposed
```

- 4.6. Find the URL associated with the new route:

```
[student@workstation ~]$ oc get route -o jsonpath='{..spec.host}{'\n'}'
php-helloworld-youruser-dev.apps.sandbox-m2.ll9k.p1.openshiftapps.com
```

- 4.7. Test the application by opening the URL you obtained in the previous step in a web browser, for example:

```
[student@workstation ~]$ firefox \
php-helloworld-youruser-dev.apps.sandbox-m2.ll9k.p1.openshiftapps.com
```

- 5. Explore starting application builds by changing the application in its Git repository and executing the proper commands to start a new Source-to-Image build.

- 5.1. Edit the `index.php` file as shown below:

```
<?php
print "Hello, World! php version is " . PHP_VERSION . "\n";
print "A change is a coming!\n";
?>
```

Save the file.

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

Switch to the `Source Control` view (`View > SCM`). Locate the entry for the `index.php` file under the `CHANGES` heading for the `D0101x-apps` directory.

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

- 5.3. 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.

- 5.4. Click the `Views and more actions > push` to publish the changes to the remote repository.

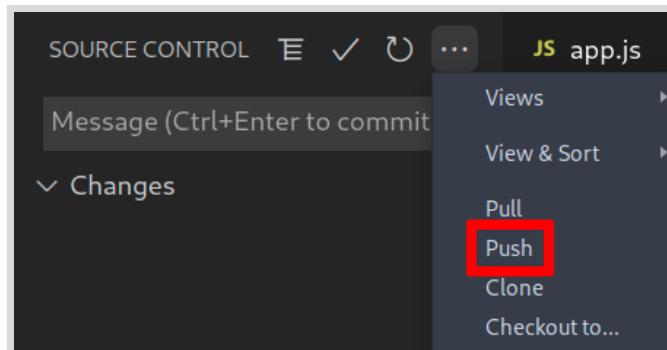


Figure 6.10: Push changes to remote repository

5.5. Start a new Source-to-Image build process.

```
[student@workstation php-helloworld]$ oc start-build php-helloworld
build.build.openshift.io/php-helloworld-2 started
```

5.6. Wait for the new build to finish:

```
[student@workstation php-helloworld]$ oc logs php-helloworld-2-build -f
...output omitted...
Successfully pushed .../php-helloworld:latest@sha256:74e757a4c0edaeda497dab7...
Push successful
```



### Note

Logs may take a while to be available after the build starts. If the previous command fails, wait and try again.

5.7. After the second build has completed use the `oc get pods` command to verify that the new version of the application is running.

```
[student@workstation php-helloworld]$ oc get pods
NAME           READY   STATUS    RESTARTS   AGE
php-helloworld-1-build  0/1     Completed   0          11m
php-helloworld-2-build  0/1     Completed   0          45s
php-helloworld-2-wq9wz  1/1     Running    0          13s
```

5.8. Test that the application serves the new content:

```
[student@workstation php-helloworld]$ firefox \
php-helloworld-youruser-dev.apps.sandbox-m2.ll9k.p1.openshiftapps.com
```

## ► 6. Clean up.

6.1. Remove RHOCP resources with the `app=php-helloworld` label.

```
[student@workstation ~]$ oc delete all -l app=php-helloworld
service "php-helloworld" deleted
deployment.apps "php-helloworld" deleted
buildconfig.build.openshift.io "php-helloworld" deleted
imagestream.image.openshift.io "php-helloworld" deleted
route.route.openshift.io "php-helloworld" deleted
```

## Finish

This concludes the guided exercise.

# Summary

---

- OpenShift Container Platform stores definitions of each OpenShift or Kubernetes resource instance as an object in the cluster's distributed database service, etcd. Common resource types are: Pod, Persistent Volume (PV), Persistent Volume Claim (PVC), Service (SVC), Route, Deployment, DeploymentConfig and Build Configuration (BC).
- Use the OpenShift command-line client oc to:
  - Create, change, and delete projects.
  - Create application resources inside a project.
  - Delete, inspect, edit, and export resources inside a project.
  - Check logs from application pods, deployments, and build operations.
- The oc new-app command can create application pods in many different ways: from an existing container image hosted on an image registry, from Dockerfiles, and from source code using the Source-to-Image (S2I) process.
- Source-to-Image (S2I) is a tool that makes it easy to build a container image from application source code. This tool retrieves source code from a Git repository, injects the source code into a selected container image based on a specific language or technology, and produces a new container image that runs the assembled application.
- A Route connects a public-facing IP address and DNS host name to an internal-facing service IP. While services allow for network access between pods inside an OpenShift instance, routes allow for network access to pods from users and applications outside the OpenShift instance.
- You can create, build, deploy and monitor applications using the OpenShift web console.

