Red Hat
Training and Certification

Student Workbook

Kubernetes 1.22 DO100B

Managing Cloud-native Applications with Kubernetes

Edition 2

DO100B-K1.22-en-2-7067502

Red Hat
Learning Community

Join a community dedicated to learning open source

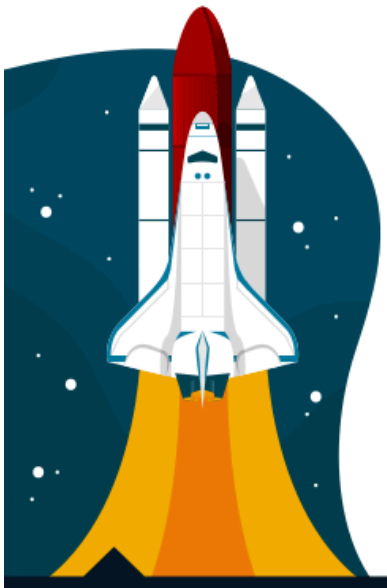
The Red Hat® Learning Community is a collaborative platform for users to accelerate open source skill adoption while working with Red Hat products and experts.

Network with tens of thousands of comunity 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

Red Hat

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 Gutterman, Fernando Lozano
DevOps Engineer:	Richard Allred
Editor:	Sam Ffrench

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*
CPU	1.6 GHz or faster processor	Multi-core i7 or equivalient
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

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
```

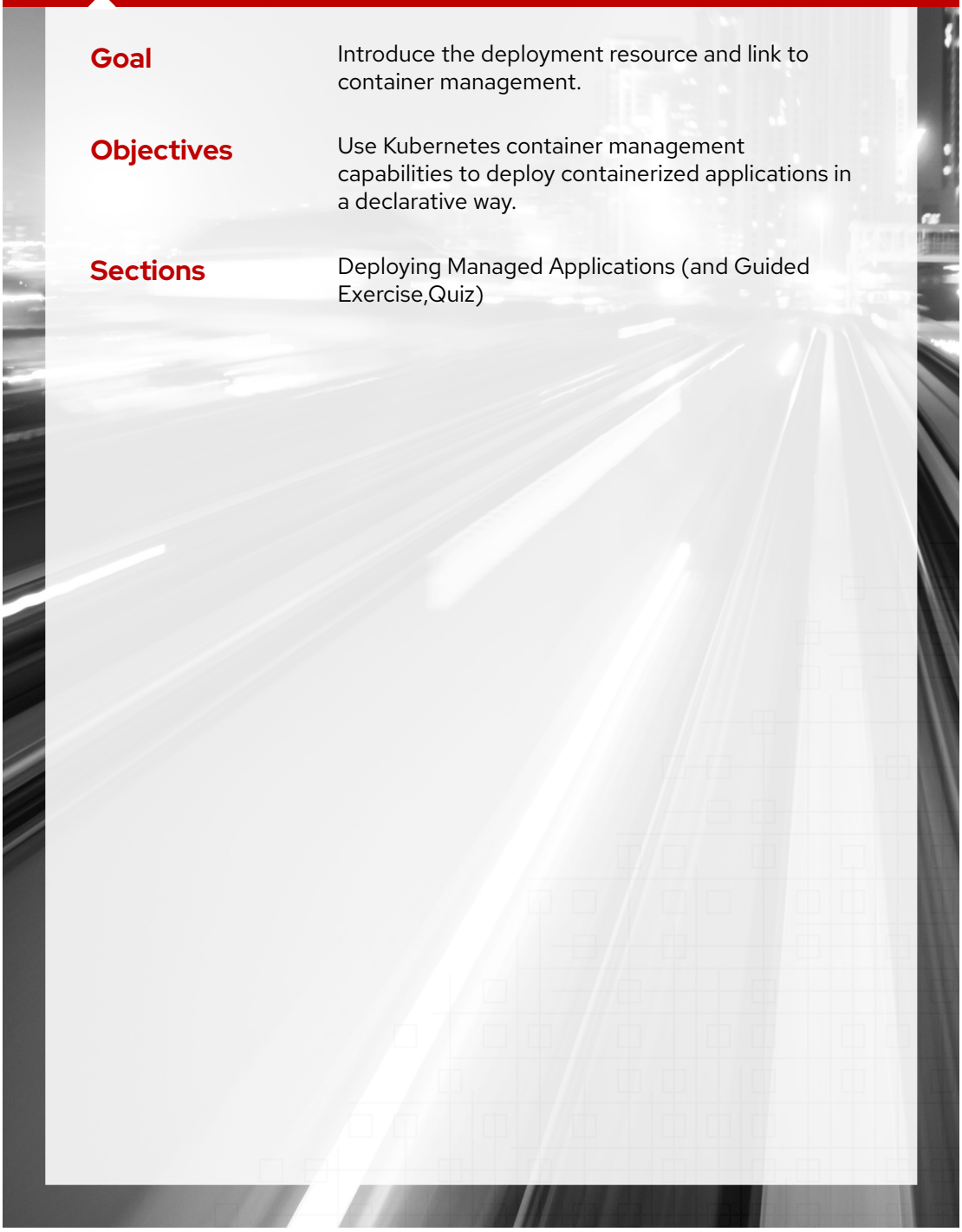
Alternatively, you can type commands in one line on all systems, such as:

```
[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 at 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-qtfs9`. 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-qtfs9  0/1     Pending             0          0s
do100-versioned-hello-76c4494b5d-qtfs9  0/1     Pending             0          0s
do100-versioned-hello-76c4494b5d-qtfs9  0/1     ContainerCreating   0          0s
do100-versioned-hello-76c4494b5d-qtfs9  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-qtfs9`. 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-qtfs9  0/1     Pending             0        0s
do100-versioned-hello-76c4494b5d-qtfs9  0/1     Pending             0        0s
do100-versioned-hello-76c4494b5d-qtfs9  0/1     ContainerCreating   0        0s
do100-versioned-hello-76c4494b5d-qtfs9  1/1     Running             0        2s
do100-versioned-hello-76c4494b5d-qtfs9  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-qtfs9 0/1   Pending           0        0s
do100-versioned-hello-76c4494b5d-qtfs9 0/1   Pending           0        0s
do100-versioned-hello-76c4494b5d-qtfs9 0/1   ContainerCreating 0        0s
do100-versioned-hello-76c4494b5d-qtfs9 1/1   Running           0        2s
do100-versioned-hello-76c4494b5d-qtfs9 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-qtfs9 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.

```
[user@host ~]$ kubectl get pods -w
NAME                                   READY STATUS            RESTARTS AGE
...output omitted...
do100-versioned-hello-76c4494b5d-qtfs9  1/1   Running           0        2s
...output omitted...
do100-versioned-hello-76c4494b5d-8qmk8  1/1   Running           0        2s      (1)
...output omitted...
do100-versioned-hello-76c4494b5d-7729k  0/1   Pending           0        0s      (2)
do100-versioned-hello-76c4494b5d-7729k  0/1   Pending           0        0s
do100-versioned-hello-76c4494b5d-7729k  0/1   ContainerCreating 0        0s
do100-versioned-hello-76c4494b5d-7729k  1/1   Running           0        1s      (3)
do100-versioned-hello-77fc5857fc-qtfs9  1/1   Terminating       0        4m46s   (4)
do100-versioned-hello-76c4494b5d-vrlkw  0/1   Pending           0        0s      (5)
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
do100-versioned-hello-77fc5857fc-qtfs9  0/1   Terminating       0        4m48s
do100-versioned-hello-76c4494b5d-vrlkw  1/1   Running           0        2s      (6)
do100-versioned-hello-77fc5857fc-8qmk8  1/1   Terminating       0        4m49s   (7)
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.

Sections

- Exposing Applications for External Access (and Guided Exercise,Quiz)
- Exposing Applications for External Access: Ingress Resources (and Guided Exercise,Quiz)

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.

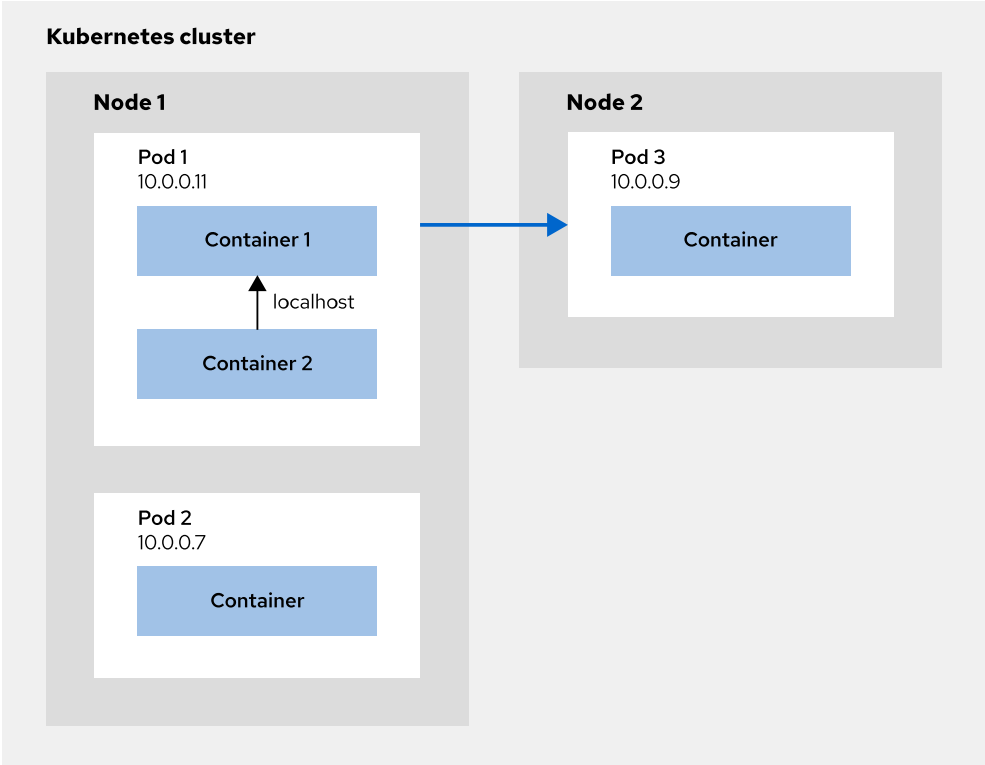


Figure 2.1: Networking in a Kubernetes cluster

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.

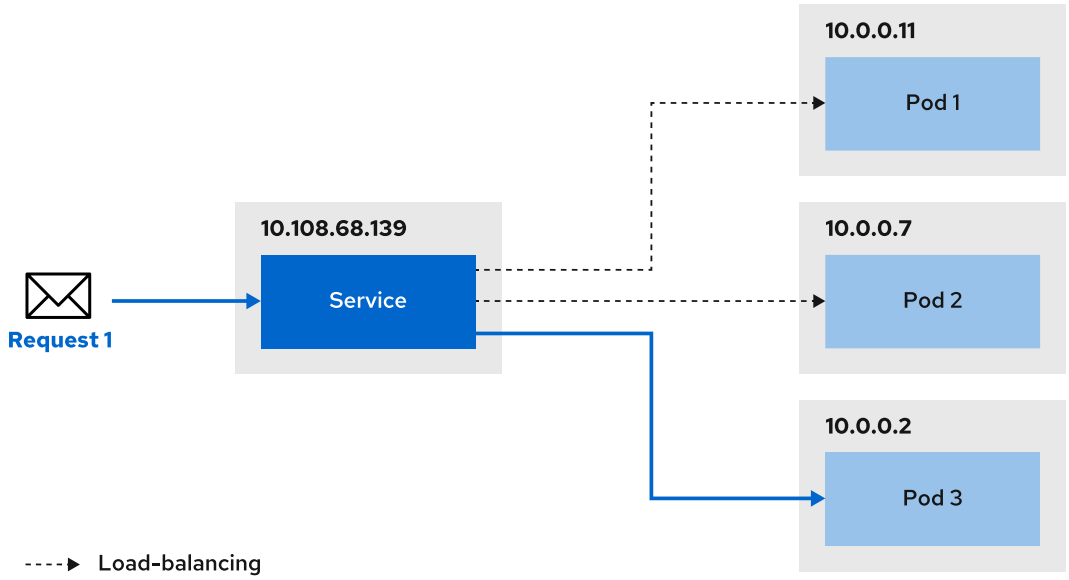


Figure 2.2: A service redirecting traffic to pod replicas

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.

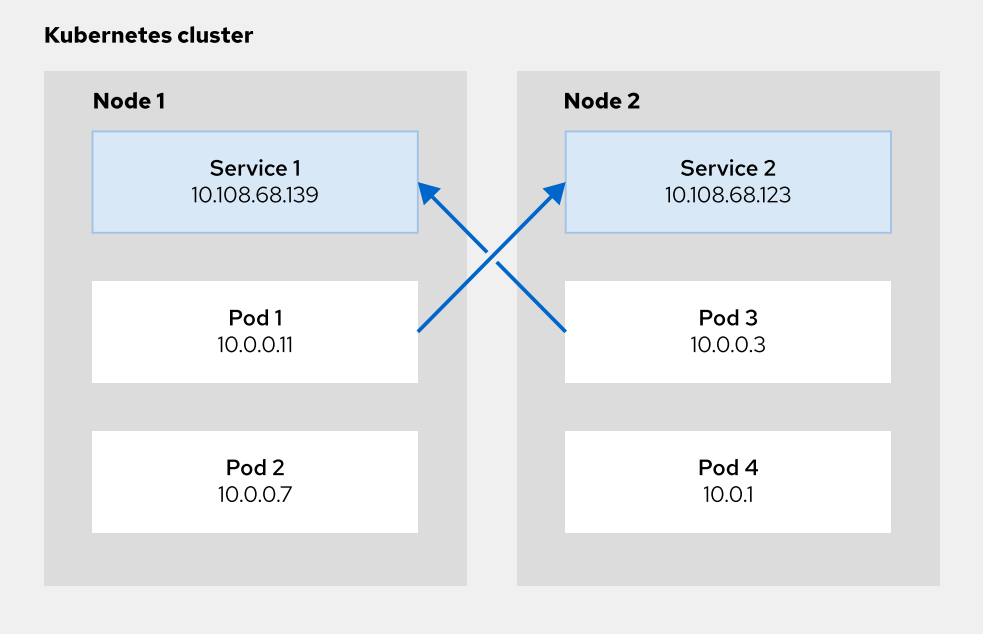


Figure 2.3: Pod communication using services

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 in `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 `DO100x-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 DO100x-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 `DO100x-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 `DO100x-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-external` image.

```
[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 you 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 reponsible 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 reponsible 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
...output omitted...
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
...output omitted...
  Resource                    Requests     Limits
  --------                    --------     ------
  cpu                         600m (17%)   0 (0%)
  memory                      1506Mi (9%)  512Mi (3%)
...output omitted...
```

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"
```

1. 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.

2. 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.

3. 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.

4. 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.

5. 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.

6. 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.

7. 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-ctsjp   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
`initialDelaySeconds`	Yes	Determines how long to wait after the container starts before beginning the probe.	0
`timeoutSeconds`	Yes	Determines how long to wait for the probe to finish. If this time is exceeded, then Kubernetes assumes that the probe failed.	1
`periodSeconds`	No	Specifies the frequency of the checks.	10
`successThreshold`	No	Specifies the minimum consecutive successes for the probe to be considered successful after it has failed.	1
`failureThreshold`	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: /health (1)
    port: 8080
  initialDelaySeconds: 15 (2)
  timeoutSeconds: 1 (3)
...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: 8080 (1)
  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:
    ...output omitted...
    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:

```
[user@host ~]$ kubectl get pods
NAME                           READY   STATUS    RESTARTS      AGE
...output omitted...
do100-probes-7794c5cb4f-vwl4x   0/1     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.

```
[user@host ~]$ kubectl get pods
NAME                           READY   STATUS    RESTARTS      AGE
...output omitted...
do100-probes-7794c5cb4f-vwl4x   1/1     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 sucesfully.
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 Kubernenetes terminates the container.
b. If the readiness probe fails once, then Kubernenetes stops sending traffic to the container.
c. If the readiness probe fails five consecutive times, then Kubernenetes terminates the container.
d. If the readiness probe fails five consecutive times, then Kubernenetes 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 sucesfully.
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 Kubernenetes terminates the container.
b. If the readiness probe fails once, then Kubernenetes stops sending traffic to the container.
c. If the readiness probe fails five consecutive times, then Kubernenetes terminates the container.
d. If the readiness probe fails five consecutive times, then Kubernenetes 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 (1)(2)
    key2: value2 (3)(4)
kind: ConfigMap (5)
metadata:
    name: myconf (6)
```

(1) The name of the first key. By default, an environment variable or a file the with 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
bmV3cGFzc3dvcmQK
```

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":"bmV3cGFzc3dvcmQK"}}'
```

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 `DO100x-apps` repository
- You have executed the setup script located at `DO100x-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 `DO100x-apps` Git repository.

```
[user@host ~]$ cd DO100x-apps
```

   1.2. Inspect the `DO100x-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...
```

1.3. Create a new deployment called `app-config` using the `DO100x-apps/app-config/kubernetes/deployment.yml` file.

```
[user@host DO100x-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 DO100x-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 DO100x-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 DO100x-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 DO100x-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 DO100x-apps]$ kubectl describe cm/appconfmap
Name:  appconfmap
...output omitted...
Data
====
APP_MSG:
---
Test Message
...output omitted...
```

3.3. Review the contents of the `DO100x-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 DO100x-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 DO100x-apps]$ kubectl get secret/appconffilesec -o json
{
    "apiVersion": "v1",
    "data": {
        "myapp.sec": "dXNlcm5hbWU9dXNlcjEKcGFzc3dvcmQ9cGFzczEKc2...
    },
    "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 DO100x-apps]$ kubectl set env deployment/app-config \
--from configmap/appconfmap
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 `DO100x-apps/app-config/kubernetes/secret.yml` file.

```
[user@host DO100x-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 DO100x-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 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`

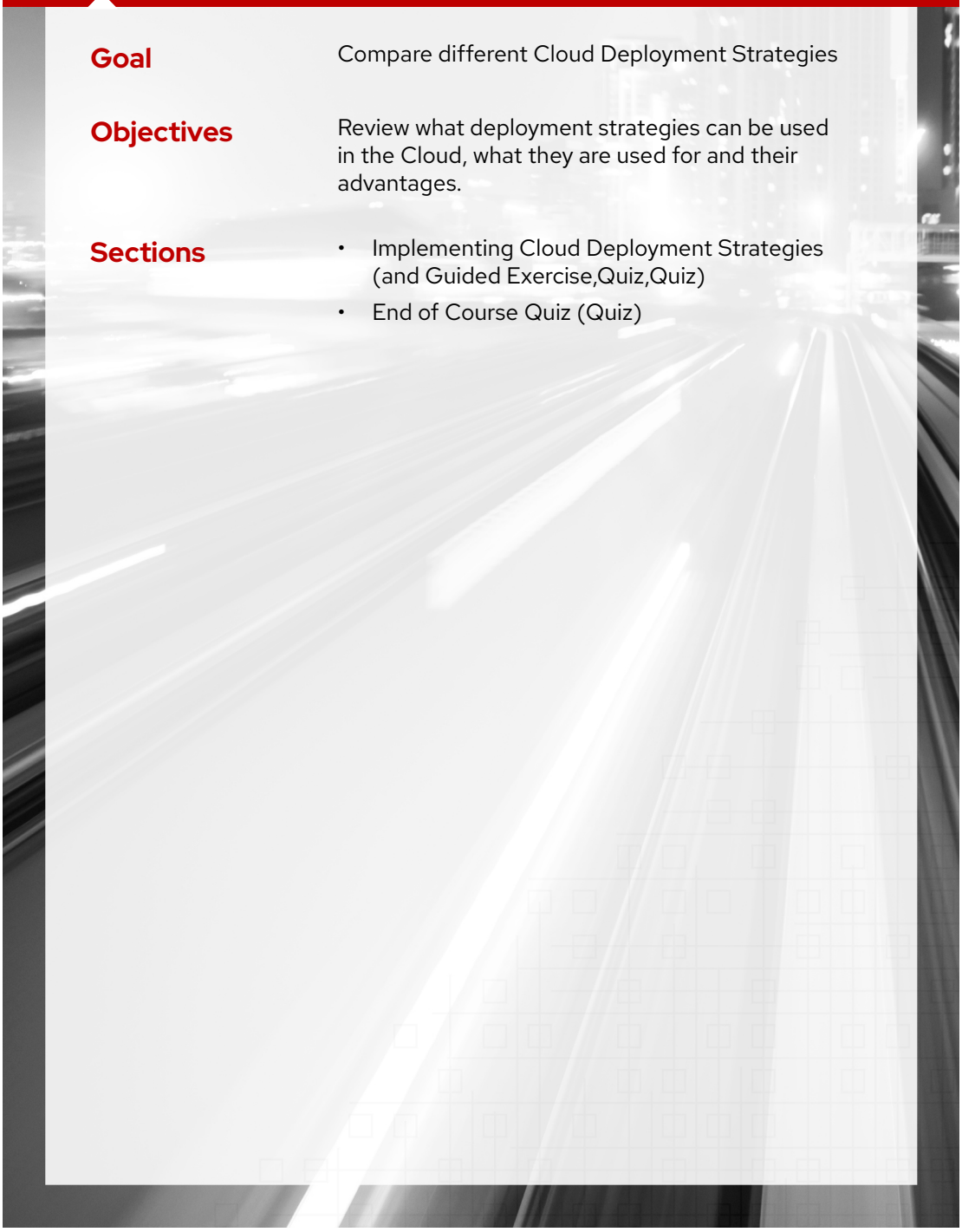
Summary

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: RollingUpdate (1)
    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-snx4f   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-lnlcz   0/1     ContainerCreating   0          3s
...output omitted...
do100-multi-version-8477f6f4bb-lnlcz   0/1     Running             0          5s
do100-multi-version-8477f6f4bb-lnlcz   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.

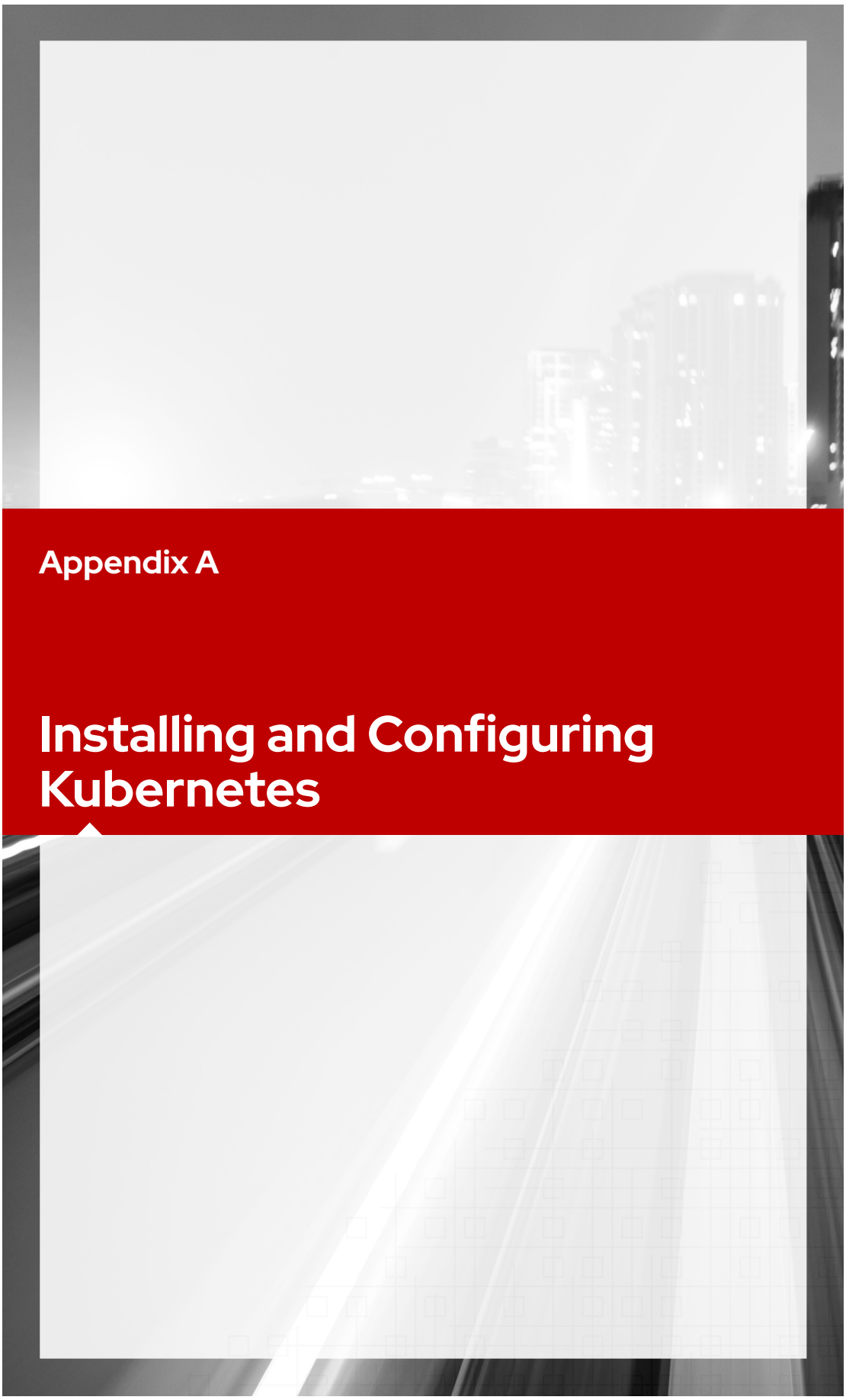
19. **Which two of the following statements about deployment strategies are correct? (Choose two.)**
    a. Developers configure all deployment strategies, such as the `Recreate` and A/B `Deployment` strategies, in the `Deployment` resource manifest.
    b. Developers configure some deployment strategies, 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
```

Verity 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/jettech/kube-webhook-certgen:v1.5.1
    ▪ Using image docker.io/jettech/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.

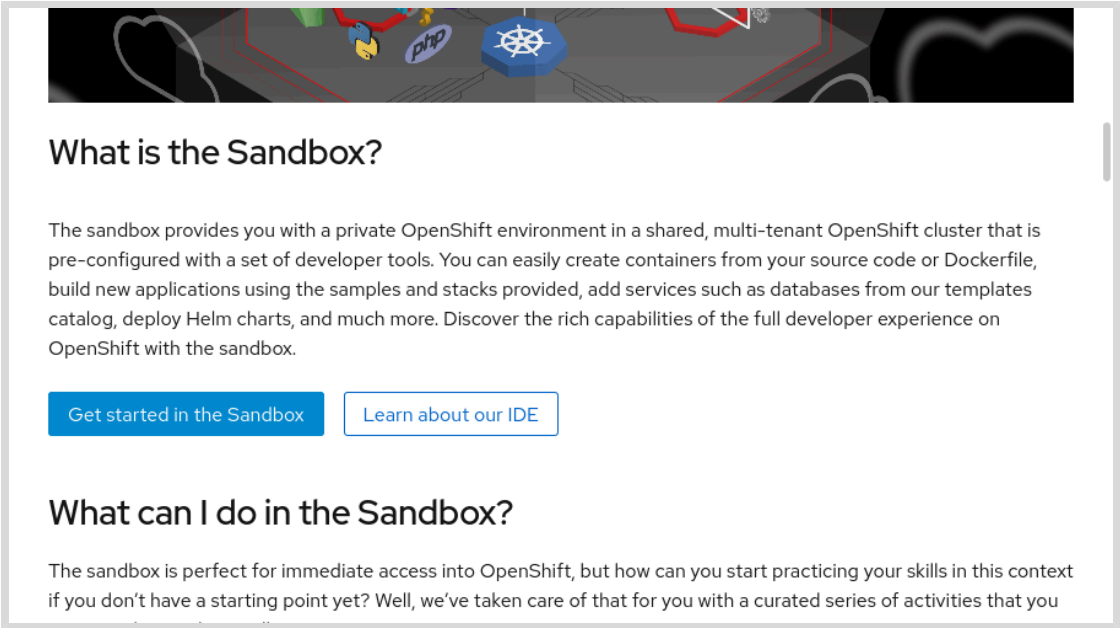
6. Using a Developer Sandbox for Red Hat OpenShift as a Remote Kubernetes cluster

`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.

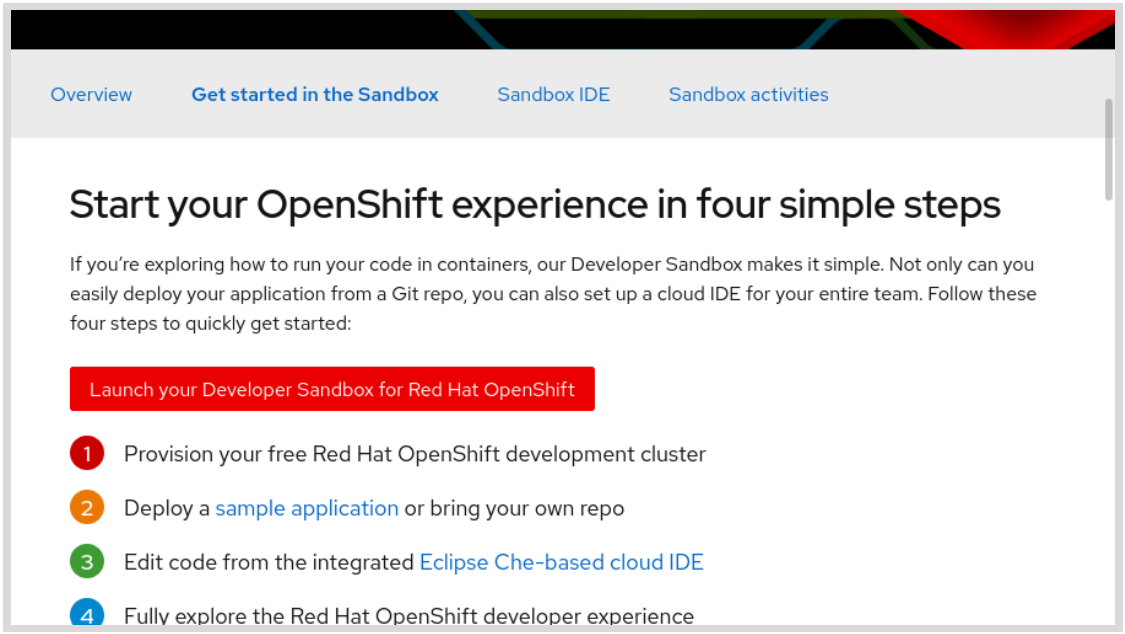
6.1. Create a Developer Sandbox account

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



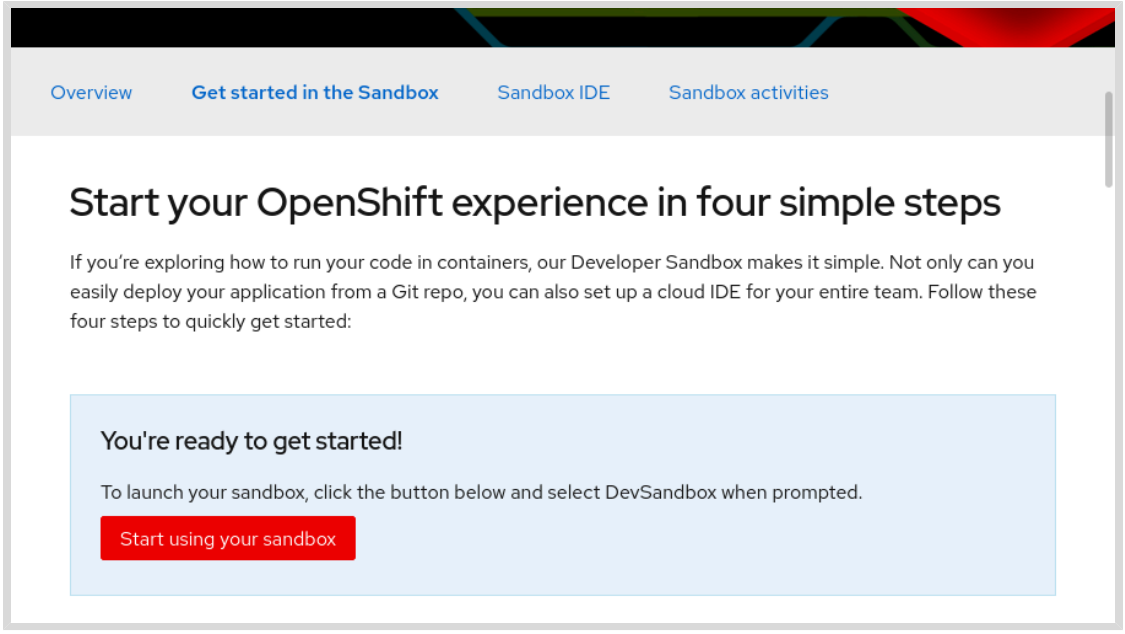
You need a free Red Hat account to use Developer Sandbox. Log in to your Red Hat account, or if you do not have one, then click **Create one now**. Fill in the form choosing a `Personal` account type, and then click **CREATE MY ACCOUNT**. You might need to accept Red Hat terms and conditions to use the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

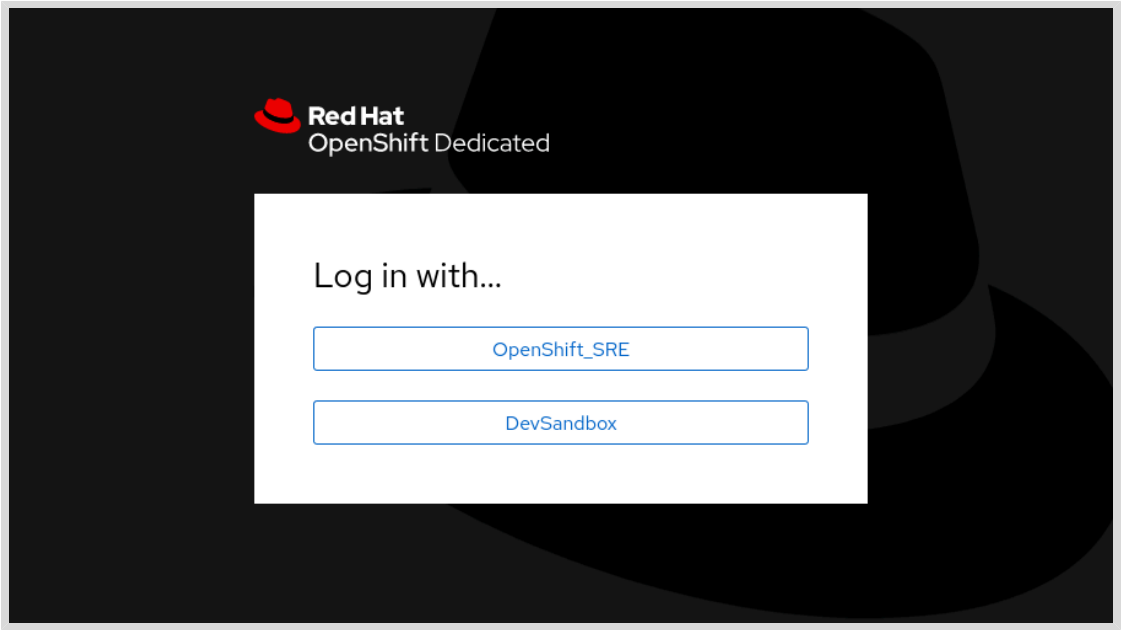


If you just created your account, then you might need to wait some seconds for account approval. You might need to verify your account via 2-factor authentication.

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.



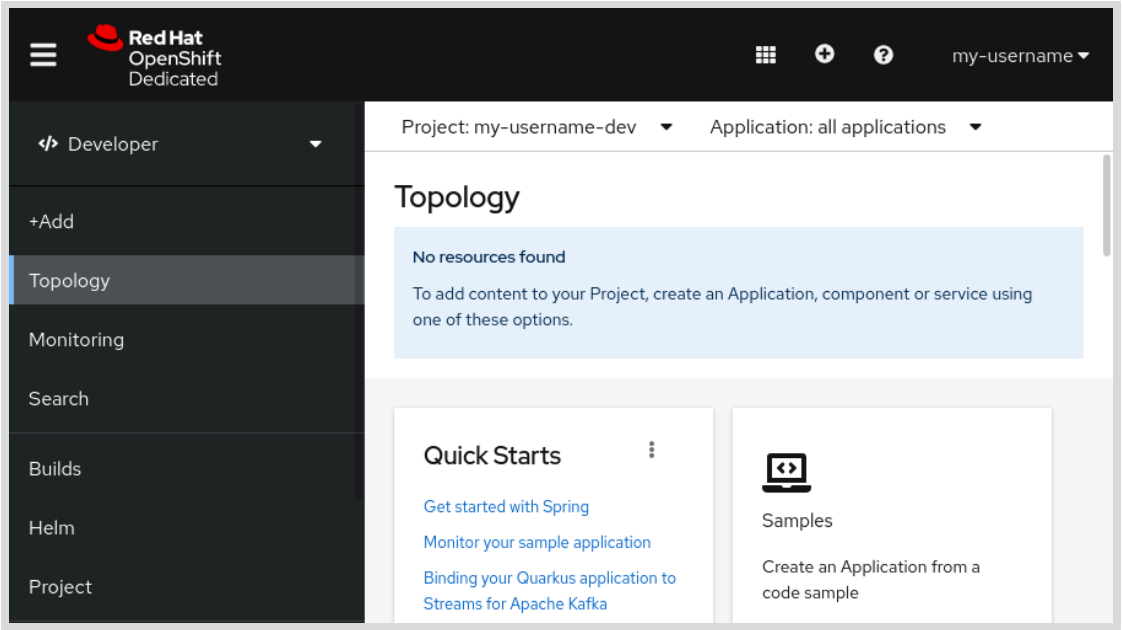
In the OpenShift log in form, click **DevSandbox** to select the authentication method.



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.



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 you 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 Sillicon 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 `DO100x-apps` repository

The `DO100x-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/DO100x-apps.git
...output omitted...
```

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

- Open a web browser and navigate to `https://github.com/RedHatTraining/DO100x-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 `DO100x-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 `DO100x-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 DO100x-apps]$ chmod +x ./setup/operating-system/setup.sh
[user@host DO100x-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 `DO100x-apps` directory and execute the following command. This command allows you to run unsigned PowerShell scripts in your current terminal session.

```
[user@host DO100x-apps]$ Set-ExecutionPolicy -Scope Process -ExecutionPolicy
 Bypass
```

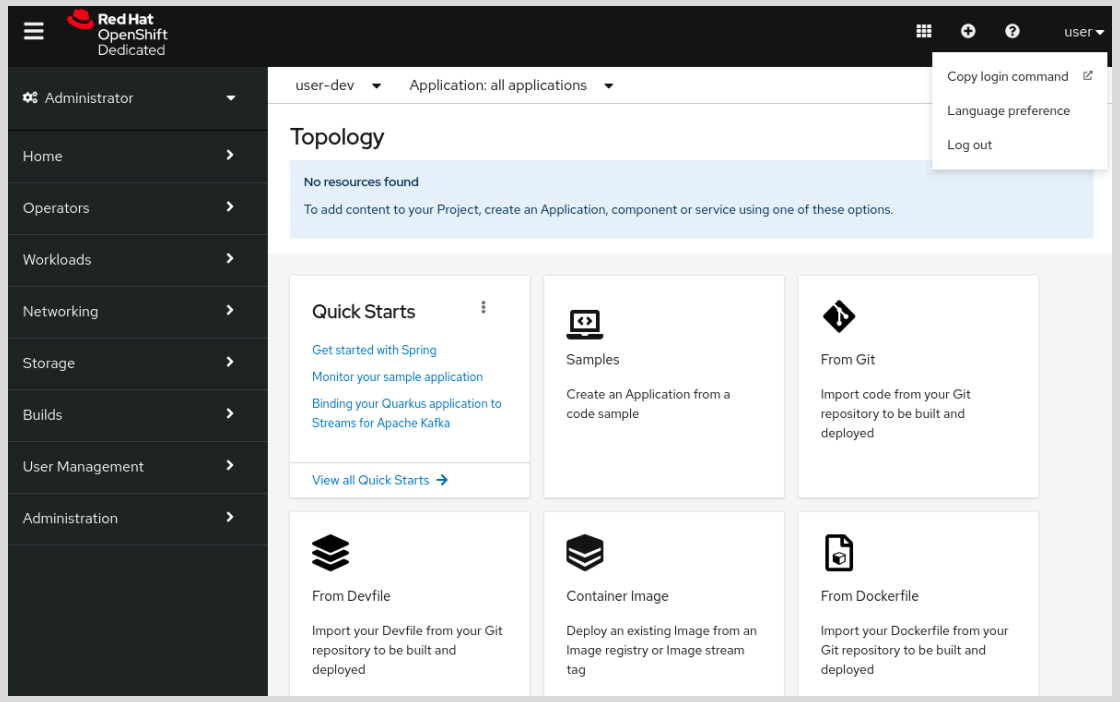
Then, run the script located at `.\setup\windows\setup.ps1`.

```
[user@host DO100x-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-fo866bbddv8wbQObpmy321eSiqj1g --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 appropiate script. The following instructions will depend on your operating system.

- **Linux and macOS**

In your command-line terminal, move to the `DO100x-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 DO100x-apps]$ chmod +x ./setup/operating-system/setup-sandbox.sh
[user@host DO100x-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 `DO100x-apps` directory and execute the following command. This command allows you to run unsigned PowerShell scripts in your current terminal session.

```
[user@host DO100x-apps]$ Set-ExecutionPolicy -Scope Process -ExecutionPolicy
 Bypass
```

Then, run the script located at `.\setup\windows\setup-sandbox.ps1`.

```
[user@host DO100x-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.