Your goal in this project is to containerize and deploy an RoR application on Kubernetes. The application is located in the /usercode/elearning directory and is ready to use.

This application provides search functionality and enables users to search for courses. These courses are stored and fetched from a PostgreSQL database.

If you wish to explore more, please have a look at the application code.

**Task 1: Create a Cluster**

You need a cluster to run Kubernetes. A **cluster** is a group of computers working together that can be viewed as a single system.

Kubernetes provides several distributions to create a cluster. You can use any of the following distributions to create one:

* minikube
* kind
* Docker Desktop
* kubeadm

**Note**: For this project, we will use a kind cluster.

To complete this task, do the following:

* Change the directory to /usercode.
* Create a kind cluster.
* Verify that the cluster has been created.

There’s no need to install kind since it’s already set up for this project.

**Note:** The cluster creation might take a little while, approximately one to two minutes. Please recreate the cluster if the workspace shuts down or the session times out.

If you’re unsure how to create a cluster, click the “Show Hint” button.

Hide Hint

* Use the kind create command to create the cluster.
* Use the kind get command to verify cluster creation.

If you’re feeling stuck, click the “Show Solution” button.

Hide Solution

* Use the following command to change the directory:

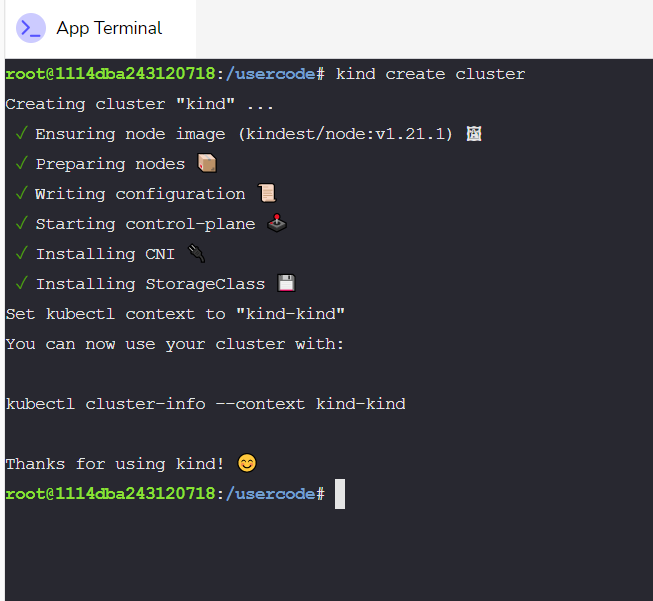
cd usercode

* Create the cluster by using the following command:

kind create cluster

* Use the following command to display the name of the cluster, i.e., kind:

kind get clusters



**Task 2: Set up Helm**

Helm is necessary to install and use Helm charts. It’s designed to bundle Kubernetes resource files into a single package that can be easily deployed.

To complete this task, do the following:

* Download the Helm 3 installation script.
* Change file permissions.
* Set up Helm.
* Verify that Helm is installed correctly.

**Note**: If the workspace resets or the session times out, use the commands.sh file to recreate the cluster and install Helm. You can use the bash commands.sh command to run this file.

If you’re unsure how to set up Helm, click the “Show Hint” button.

Hide Hint

* Get the official Helm script command from GitHub, and use the curl command to download the Helm 3 installation script.
* Use the chmod command to change file permissions.
* Run the ./get\_helm.sh file to install Helm.
* Use the Helm help command to verify the installation.

If you’re feeling stuck, click the “Show Solution” button.

Hide Solution

* Use the following command to download the Helm 3 installation script:

curl -fsSL -o get\_helm.sh https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3

* Use the following command to change access permissions:

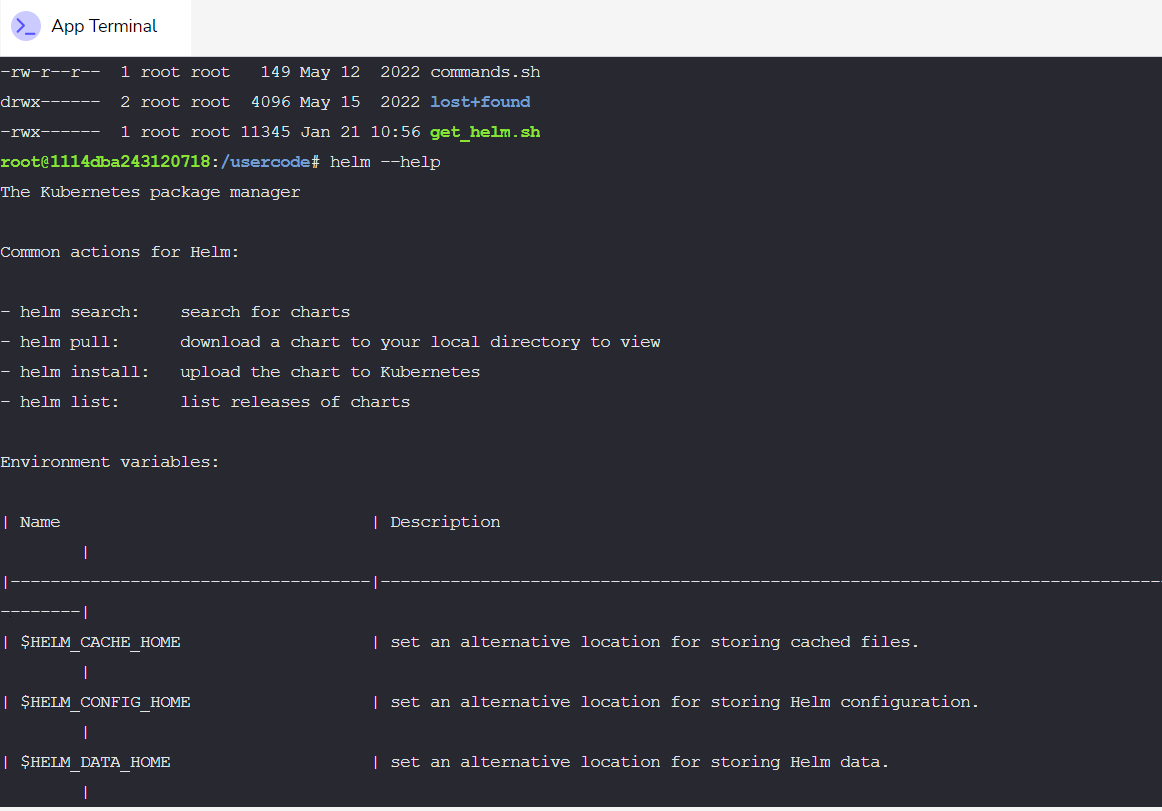
chmod 700 get\_helm.sh

* Use the following command to run the Helm installation script:

./get\_helm.sh

* Use the following command to verify the installation:

helm --help



**Task 3: Containerize and Push the Frontend**

To deploy the application’s frontend, start by containerizing it. The Dockerfile has already been created for you. It’s located in the /usercode/elearning folder.

To complete this task, do the following:

* Change the directory to /usercode/elearning.
* Create a Docker image.
* Push this image to Docker Hub.

After you build the image, verify the existence of the image using the following command:

docker images

If you wish to skip this task, you can also find a pre-built Docker image by using the following name:

educative1/multiple\_deployments:latest

If you’re unsure how to create an image and push it to Docker Hub, click the “Show Hint” button.

Show Hint

If you’re feeling stuck, click the “Show Solution” button.

Hide Solution

* Use the following command to change the current directory:

cd /usercode/elearning

* Use the following command to build the Docker image. Here my-username is the username of your Docker Hub account, and tag will be the tag of the Docker image:

docker build -t <my-username>/<my-image-name>:<tag> .

* Use the following command to log in to Docker Hub. Here my-username is your Docker Hub username, and my-password is your Docker Hub password:

docker login -u <my-username> -p <my-password>

* Use the following command to push the image to Docker Hub:

docker push <my-username>/<my-image-name>:<tag>

**Task 4: Set up the Chart**

A **chart**, also known as a package, is a set of Helm-specific files that install an application on a Kubernetes cluster. A Helm chart helps deploy applications ranging from the simplest to the most complex.

To complete this task, do the following:

* Change your directory to /usercode.
* Create a Helm chart named multiple-deployments.
* Delete the following directories and files:
  + /usercode/multiple-deployments/templates/hpa.yaml
  + /usercode/multiple-deployments/templates/ingress.yaml
  + /usercode/multiple-deployments/templates/serviceaccount.yaml
  + /usercode/multiple-deployments/templates/tests
* Delete the content from the following files:
  + /usercode/multiple-deployments/values.yaml
  + /usercode/multiple-deployments/templates/deployment.yaml
  + /usercode/multiple-deployments/templates/service.yaml
  + /usercode/multiple-deployments/templates/NOTES.txt
  + /usercode/multiple-deployments/templates/\_helpers.tpl

If you’re unsure how to create a Helm chart, click the “Show Hint” button.

Show Hint

If you’re feeling stuck, click the “Show Solution” button.

Hide Solution

* Use the following command to change the current directory:

cd /usercode

* You can create the chart using the following command:

helm create multiple-deployments

* Delete the directories and files by clicking on the triple dots next to the file name in the file tree, or use the following commands:

rm -r  /usercode/multiple-deployments/templates/tests

rm /usercode/multiple-deployments/templates/ingress.yaml /usercode/multiple-deployments/templates/hpa.yaml /usercode/multiple-deployments/templates/serviceaccount.yaml

* Open each of the files from the list below, and delete their content by pressing “Ctrl+A” and “Backspace”.
  + /usercode/multiple-deployments/values.yaml
  + /usercode/multiple-deployments/templates/deployment.yaml
  + /usercode/multiple-deployments/templates/service.yaml
  + /usercode/multiple-deployments/templates/NOTES.txt
  + /usercode/multiple-deployments/templates/\_helpers.tpl

**Task 5: Add the Database**

**Artifact Hub** enables users to find, install, and package projects. It contains a wide range of published packages with different, customized applications. To deploy the database, use a Helm chart published from Artifact Hub.

Perform the following steps to deploy the database:

* Change the directory to /usercode/multiple-deployments/.
* Search for the PostgreSQL Helm chart named “bitnami” using [Artifact Hub](https://artifacthub.io/).
* Add the database as a dependency in the /usercode/multiple-deployments/Chart.yaml file with the following specifications:
  + name: PostgreSQL
  + version: 10.14.0
  + repository: bitnami
* Download PostgreSQL as a dependency.
* Verify that the dependency has downloaded successfully.

If you need any help adding and downloading the dependency, click the “Show Hint” button.

Show Hint

If you’re feeling stuck, click the “Show Solution” button.

Hide Solution

* Use the following command to change the current directory:

cd /usercode/multiple-deployments

* Use the following command to search for the Helm chart:

helm search hub postgresql

* Add the following code at the end of the /usercode/multiple-deployments/Chart.yaml file:

dependencies:  
  - name: postgresql  
    version: 10.14.0  
    repository: "https://charts.bitnami.com/bitnami"

* Use the following command to download the required dependencies:

helm dependency update

* Refresh the editor. You’ll see that the following files have now been created:
  + /usercode/multiple-deployments/Chart.lock
  + /usercode/multiple-deployments/charts/postgresql-10.14.0.tgz

Our RoR application is dependent on the /usercode/multiple-deployments/charts/postgresql-10.14.0.tgz file. The /usercode/multiple-deployments/Chart.lock file can be used to rebuild the dependencies to the specifications of an exact version.

**Task 6: Configure the Database**

To deploy the database, configure the postgresql Helm chart. The database chart will use the values defined in the /usercode/multiple-deployments/values.yaml file.

Open the /usercode/multiple-deployments/values.yaml file. To complete the task successfully, do the following:

* Change the bitnami PostgreSQL chart version to 10.14.0.
* From “VALUES SCHEMA,” figure out the parameters needed to perform the following tasks:
  + Change the Admin username to postgres.
  + Change the password to postgrespassword.
* Configure the chart according to the following specifications:
  + postgresql.fullnameOverride: “postgres”
  + postgresql.image: [your-image-name-here]
  + postgresql.tag: 11.14.0-debian-10-r17
  + postgresql.postgresqlDatabase: elearning3\_development
  + postgresql.<your-username-parameter>: [your-username]
  + postgresql.<your-password-parameter>: [your-password]

**Note**: If you want to learn more about the chart and its parameters, you may download the chart using the helm pull chartrepo/chartname command.

If you need any help in configuring the chart, click the “Show Hint” button.

Show Hint

If you’re feeling stuck, click the “Show Solution” button.

Hide Solution

* Use this [link](https://artifacthub.io/packages/helm/bitnami/postgresql/10.14.0) to view the PostgreSQL bitnami chart.
* From the “CHART VERSIONS” field, select the “10.14.0” version.
* From “VALUES SCHEMA,” use the following fields to set your username and password:
  + postgresqlUsername
  + postgresqlPassword
* After adding the image and overriding the name, the /usercode/multiple-deployments/values.yaml will look like this:

postgresql:  
  fullnameOverride: "postgres"  
  image:  
    tag: 11.14.0-debian-10-r17

* After adding all the required specifications, the /usercode/multiple-deployments/values.yaml file will display these parameters:

postgresql:  
  fullnameOverride: "postgres"  
  image:  
    tag: 11.14.0-debian-10-r17  
  postgresqlDatabase: elearning3\_development  
  postgresqlUsername: postgres  
  postgresqlPassword: postgrespassword

**Task 7: Modify the Values File**

**ConfigMaps** are Kubernetes objects used to store key-value pairs. You can inject environment variables into the containers using these key-value pairs.

To do this, open the /usercode/multiple-deployments/values.yaml file. Configure the chart to use the following specifications:

* postgresql.databasePort: 5432
* configmap.name: postgres-configmap

If you need any help in adding these values in the file, click the “Show Hint” button.

Show Hint

If you’re feeling stuck, click the “Show Solution” button.

Hide Solution

Edit the /usercode/multiple-deployments/values.yaml file so that it looks like the code snippet below:

postgresql:  
  fullnameOverride: "postgres"  
  image:  
    tag: 11.14.0-debian-10-r17  
  postgresqlDatabase: elearning3\_development  
  postgresqlUsername: postgres  
  postgresqlPassword: postgrespassword  
  databasePort: 5432  
  
configmap:   
  name: postgres-configmap

**Task 8: Create a ConfigMap**

Your task now is to create a ConfigMap that will use the database values defined in the values.yaml file. This ConfigMap will be used to inject environment variables into the front-end container.

To do this, create a configmap.yaml file in the /usercode/multiple-deployments/templates/ folder.

The ConfigMap should contain the following specifications:

* name: configmap.name
* data.POSTGRES\_SVC: postgresql.fullnameOverride
* data.POSTGRES\_PORT: postgresql.databasePort
* data.POSTGRES\_DB: postgresql.postgresqlDatabase
* data.POSTGRES\_USER: postgresql.postgresqlUsername
* data.POSTGRES\_PASSWORD: postgresql.postgresqlPassword

Create the /usercode/multiple-deployments/templates/configmap.yaml file, and adjust the ConfigMap to the above mentioned specifications. Convert the values obtained from the /usercode/multiple-deployments/values.yaml file into strings.

If you’re unsure how to create and configure the ConfigMap, click the “Show Hint” button.

Show Hint

If you’re feeling stuck, click the “Show Solution” button.

Hide Solution

* Use the following command to create the configmap.yaml file:

touch /usercode/multiple-deployments/templates/configmap.yaml

Or, in the file tree, hover over the templates folder, and click the three dots. Then, click the “New File” option.

* Copy the following code in /usercode/multiple-deployments/templates/configmap.yaml:

apiVersion: v1  
kind: ConfigMap  
metadata:  
  name: {{ .Values.configmap.name }}  
data:  
  POSTGRES\_SVC: {{quote .Values.postgresql.fullnameOverride}}  
  POSTGRES\_PORT: {{quote .Values.postgresql.databasePort}}  
  POSTGRES\_DB: {{quote .Values.postgresql.postgresqlDatabase}}  
  POSTGRES\_USER: {{quote .Values.postgresql.postgresqlUsername}}  
  POSTGRES\_PASSWORD: {{quote .Values.postgresql.postgresqlPassword}}

**Task 9: Modify the Values File**

To deploy the application as a Service, you’ll have to create another Kubernetes Service object.

To do this, open the /usercode/multiple-deployments/values.yaml file, and configure the chart to the following specifications:

* app.project: helm-project
* app.tier: application
* service.type: NodePort
* service.port: 31111
* service.nodePort: 31111
* service.targetPort: 3000

If you need some help adding these values to the file, click the “Show Hint” button.

Show Hint

If you’re feeling stuck, click the “Show Solution” button.

Hide Solution

Edit your /usercode/multiple-deployments/values.yaml file so that it looks like the code snippet below:

postgresql:  
  fullnameOverride: "postgres"  
  image:  
    tag: 11.14.0-debian-10-r17  
  postgresqlDatabase: elearning3\_development  
  postgresqlUsername: postgres  
  postgresqlPassword: postgrespassword  
  databasePort: 5432  
  
configmap:   
  name: postgres-configmap  
  
app:  
  project: helm-project  
  tier: application  
  
service:  
  type: NodePort  
  port: 31111           
  nodePort: 31111  
  targetPort: 3000

**Task 10: Create a Service**

The next task is to create a Service for the front-end. Later on, this Service will be used to expose the RoR application. The Service will use the database values defined in the /usercode/multiple-deployments/values.yaml file.

To get started, open the /usercode/multiple-deployments/templates/service.yaml file. The Service should contain the following specifications:

* metadata.name to
* spec.type: service.type
* spec.ports.port: service.port
* spec.ports.targetPort: service.targetPort
* spec.ports.nodePort: service.nodePort
* spec.ports.protocol: TCP
* spec.ports.name: http
* spec.selector.project: app.project
* spec.selector.tier: app.tier

If you need some help to create the Service, click the “Show Hint” button.

Show Hint

If you’re feeling stuck, click the “Show Solution” button.

Hide Solution

Edit your /usercode/multiple-deployments/templates/service.yaml file so that it looks like the code snippet below:

apiVersion: v1  
kind: Service  
metadata:  
  name: {{ .Release.Name }}-service  
spec:  
  type: {{ .Values.service.type }}  
  ports:  
    - port: {{ .Values.service.port }}  
      targetPort: {{ .Values.service.targetPort }}  
      nodePort: {{ .Values.service.nodePort }}  
      protocol: TCP  
      name: http  
  selector:  
    project: {{.Values.app.project}}  
    tier: {{.Values.app.tier}}

**Task 11: Add Values to Deploy the Frontend**

To deploy the frontend, we need to create a Deployment.

First, define values for this Deployment. To do this, open the /usercode/multiple-deployments/values.yaml file.

* Add the following specifications to the /usercode/multiple-deployments/values.yaml file:
  + image.repository: [your-image-name].
  + image.tag: [your-image-tag].
  + deployments.name: app
  + deployments.containerPort: 3000
  + deployments.imagePullPolicy: Always
* Declare an initContainer that checks port 5432 every five seconds for database availability. This initContainer should contain the following specifications:
  + initContainers.name: check-db-ready
  + initContainers.image: postgres:9.6.5
  + initContainers.command: [Write-your-command-here]

**Note**: Get your image name by executing the docker images command. You can also access the pre-built docker image using the following name:

educative1/multiple\_deployments:latest

If you’re unsure how to add these values, click the “Show Hint”.

Show Hint

If you’re feeling stuck, click the “Show Solution” button.

Hide Solution

Edit your /usercode/multiple-deployments/values.yaml file so that it looks like the code snippet below:

postgresql:  
  fullnameOverride: "postgres"  
  image:  
    tag: 11.14.0-debian-10-r17  
  postgresqlDatabase: elearning3\_development  
  postgresqlUsername: postgres  
  postgresqlPassword: postgrespassword  
  databasePort: 5432  
  
configmap:   
  name: postgres-configmap  
  
app:  
  project: helm-project  
  tier: application  
  
service:  
  type: NodePort  
  port: 31111           
  nodePort: 31111  
  targetPort: 3000  
  
image:  
  repository: <your-image-name-here>  
  tag: <your-image-tag-here>  
  
deployments:   
  name: app  
  containerPort: 3000  
  imagePullPolicy: Always  
  
initContainers:  
  name: check-db-ready  
  image: postgres:9.6.5  
  command: ['sh', '-c',   
          'until pg\_isready -h postgres -p 5432;   
          do echo waiting for database; sleep 5; done;']

**Note**: Don’t forget to change the repository and tag in the image section.

**Task 12: Create a Deployment**

To access the application, you’ll have to create a Deployment object. This Deployment object will use the values defined in the /usercode/multiple-deployments/values.yaml file. To do this, open the /usercode/multiple-deployments/templates/deployment.yaml file.

Create a Deployment object that contains the following specifications:

* metadata.name: [[The.Release.Name](http://the.release.name/" \t "_blank)]
* spec.selector.matchLabels.project: app.project
* spec.selector.matchLabels.tier: app.tier
* spec.template.metadata.labels.project: app.project
* spec.template.metadata.labels.tier: app.tier
* spec.template.spec.initContainers.name: initContainers.name
* spec.template.spec.initContainers.image: initContainers.image
* spec.template.spec.initContainers.image: initContainers.image
* spec.template.spec.initContainers.command: initContainers.command
* spec.template.spec.containers.name: deployments.name
* spec.template.spec.containers.image: image.repository: image.tag
* spec.template.spec.containers.imagePullPolicy: deployments.imagePullPolicy
* spec.template.spec.containers.ports.containerPort: deployments.containerPort
* spec.template.spec.containers.envFrom.configMapRef.name: configmap.name

If you’re unsure how to create the Deployment object, click the “Show Hint” button.

Show Hint

If you’re feeling stuck, click the “Show Solution” button.

Hide Solution

Add the following code to the /usercode/multiple-deployments/templates/deployment.yaml file:

apiVersion: apps/v1  
kind: Deployment  
metadata:  
  name: {{ .Release.Name }}-deployment  
spec:  
  selector:  
    matchLabels:  
      project: {{.Values.app.project}}  
      tier: {{.Values.app.tier}}  
  template:  
    metadata:  
      labels:  
        project: {{.Values.app.project}}  
        tier: {{.Values.app.tier}}  
    spec:  
      initContainers:  
      - name: check-db-ready  
        image: postgres:9.6.5  
        command: ['sh', '-c',   
          'until pg\_isready -h postgres -p 5432;   
          do echo waiting for database; sleep 10; done;']  
      containers:  
      - name: {{.Values.deployments.name}}  
        image: {{.Values.image.repository}}:{{.Values.image.tag}}  
        imagePullPolicy: {{.Values.deployments.imagePullPolicy}}  
        ports:  
        - containerPort: {{.Values.deployments.containerPort}}  
        envFrom:  
          - configMapRef:  
                name: {{ .Values.configmap.name }}

**Task 13: Deploy the Chart**

Congratulations on creating the Deployment! Once you’ve edited all the essential fields, it’s time to deploy the application.

You can deploy the chart successfully by doing the following:

* Change the directory to /usercode.
* In the /usercode/multiple-deployments/templates/NOTES.txt file, add some text that will be printed once the application is deployed.
* Verify that there is no issue with the chart’s formatting.
* Install the Helm chart.
* Verify the Pod’s status.

Since Helm uses the name as the key, you *cannot* install two applications with the same name in the same namespace. This means that if the installation fails, you need to upgrade the chart by using the helm upgrade your-release-name ./your-chart-name command or reinstall the chart with a different name.

**Note**: It will take some time for the Pods to get ready. Please verify that the Pods are running before you proceed to the next task.

If you’re unsure how to deploy the chart, click the “Show Hint” button.

Show Hint

If you’re feeling stuck, click the “Show Solution” button.

Hide Solution

* Use the following command to change the current directory:

cd /usercode

* In the /usercode/multiple-deployments/templates/NOTES.txt file, add the following text:
  + Your application has been successfully deployed.
* Use the following command to verify that there are no syntax errors:

helm lint ./multiple-deployments

You should get 1 chart(s) linted, 0 chart(s) failed in the output.

* Use the following command to deploy the chart:

helm install <your-release-name> ./multiple-deployments

Along with some details of the Deployment, the message that you added in the /usercode/multiple-deployments/templates/NOTES.txt file will be printed to the screen in the ”Notes” section.

* Use the following command to check the Pod’s status:

watch kubectl get pods

**Note**: Press “Ctrl + C” to disconnect the session, or open a new terminal to perform the next task.

**Task 14: Access the Application**

In the project’s final task, access the already deployed application on the internet.

Follow these two steps to successfully access the application:

* Check the status of the Service.
* Access the internal Kubernetes application using the internet. To do this, map the Service port to 31111.

Verify that the application is running by refreshing your web browser.

If you’re unsure how to access your application, click the “Show Hint” button.

Show Hint

If you’re feeling stuck, click the “Show Solution” button.

Hide Solution

* Use the following command to check the status of the Service:

kubectl get service

* Use the following command to map the Service to the localhost:

kubectl port-forward svc/<your-service-name> --address 0.0.0.0 31111:31111

**Note**: The service name will look like this following pattern: <release-name>-service.

**Congratulations!**

Congratulations on completing your Helm project! You’ve done an excellent job.

Applying your skills with hands-on projects like this one is a great way to get comfortable with new techniques and technologies.

Now that you’ve finished this project, you can create and containerize your applications. You can also configure your Kubernetes resources, bundle them, and deploy them using Helm. The skills you’ve learned in this course will allow you to configure published charts and use them in your application.

Now that you’ve created multiple Deployments using Helm, you can visit the “Projects Homepage” for more exciting guided projects and courses that take your expertise to the next level.