Kubernetes Administration

-Workbook-

Course ID: KUB201v1.2

Version: 1.2.0

Date: 2021-04-20

SUSE

Proprietary Statement

SUSE LLC, has intellectual property rights relating to technology embodied in the product that is described in this document.

SUSE
Maxfeldstrasse 5
90409 Nuremberg
Germany
www.suse.com

Disclaimer

SUSE LLC, makes no representations or warranties with respect to the contents or use of this documentation, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose.

Further, SUSE LLC, reserves the right to revise this publication and to make changes to its content, at any time, without obligation to notify any person or entity of such revisions or changes. Further, SUSE LLC, makes no representations or warranties with respect to any software, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. Further, SUSE LLC, reserves the right to make changes to any and all parts of SUSE software, at any time, without any obligation to notify any person or entity of such changes.

Any products or technical information provided under this Agreement may be subject to U.S. export controls and the trade laws of other countries. You agree to comply with all export control regulations and to obtain any required licenses or classification to export, re-export or import deliverables. You agree not to export or re-export to entities on the current U.S. export exclusion lists or to any embargoed or terrorist countries as specified in the U.S. export laws. You agree to not use deliverables for prohibited nuclear, missile, or chemical biological weaponry end uses. SUSE assumes no responsibility for your failure to obtain any necessary export approvals.

This SUSE Training Manual is published solely to instruct students in the use of SUSE networking software. Although third-party application software packages may be used in SUSE training courses, this is for demonstration purposes only and shall not constitute an endorsement of any of these software applications.

Further, SUSE LLC does not represent itself as having any particular expertise in these application software packages and any use by students of the same shall be done at the student's own risk.

Table of Contents

Documentation Conventions:

The following typographical conventions are used in this manual:

Bold	Represents things you should pay attention to or buttons you click, text or options that you should click/select/type in a GUI.
Bold Gray	Represents the name of a Task or in the context of what is seen on the screen, the screen name, a tab name, column name, field name, etc.
Bold Red	Represents warnings or very important information.
Option > Option > Option	Represents a chain of items selected from a menu.
BOLD_UPPERCASE_ITALIC	Represents an "exercise variable" that you replace with another value.
`bold monospace`	Represents text displayed in a terminal or entered in a file.
`bold monospace blue`	Represents commands entered at the command line.
`bold monospace green`	Represents a file name.

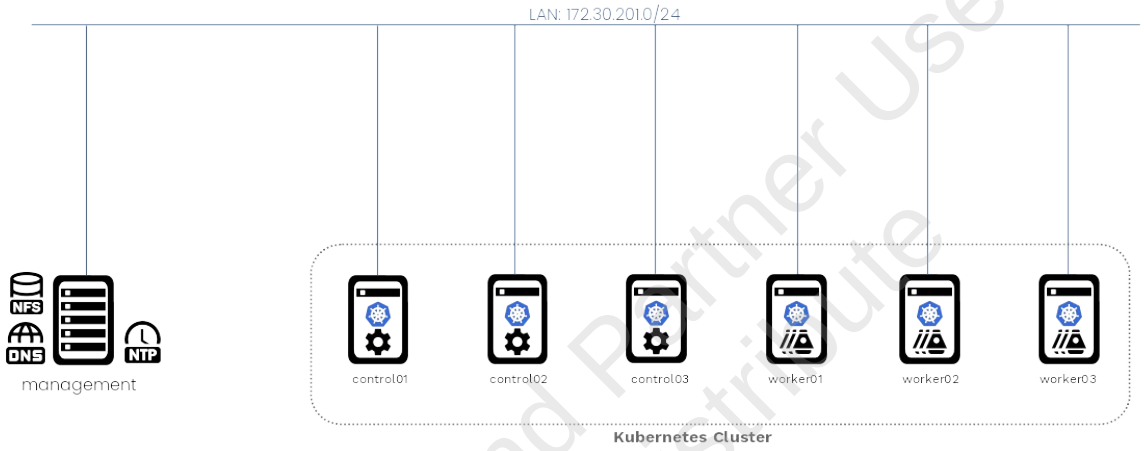
1 Course Introduction

Description:

This section introduces the course objectives and audience. It also provides and overview of the lab environment for the course.

Lab Environment Diagrams

Lab Environment Diagram



Lab Environment Information

The lab environment is comprised of 6 Kubernetes cluster nodes (three control plane nodes and three worker nodes) and a management workstation. The management workstation also functions and the DNS, NTP, NFS, etc server for the lab environment and should be powered on and remain powered on while the cluster is running.

The management workstation is where you perform the vast majority of the lab exercises.

Lab Environment Requirements

CPU: 4 Core

RAM: 50GB for VMs

Disk: 200GB

Minimum Host OS: openSUSE Leap 15.2, SLES 15 SP2

]– 1 Start the Lab Environment VMs

Description:

In this exercise, you use the Virt-Manager utility to start the lab VMs in the proper order and then log into the management workstation.

Task 1: Start the Lab Environment VMs

1. On the **lab machine**, if not already open, launch the Virt-Manager utility
   You should see the course VMs listed.
2. Right-click on the **KUB201-management** VM and select **Run**.
3. Double-click on the **KUB201-management** node to view its console.
   When the **KUB201-management** node has finished booting completely and you see the graphical login, close its console window.
   (You can use the CPU utilization graph in Virt-Manager to see when the node is finished booting and starting services.)
4. Next, right-click on the **KUB201-control01** VM and select **Run**.
   Double-click on the **KUB201-control01** node to open its console.
   When the **KUB201-control01** node has finished booting completely, close its console window.
5. Next, repeat the preceding steps for the remaining VMs in the following order:

   KUB201-control02
   KUB201-control03
   KUB201-worker01
   KUB201-worker02
   KUB201-worker03

   The required lab environment VMs should now be running.

Task 2: Log Into the Management Workstation VM

1. On the management workstation (**KUB201-management**), log in using the following credentials:

   Username: **tux**
   Password: **linux**

   You should be able to complete most if not all of the lab exercises while logged into the management workstation as the tux user.
   If for some reason you need to log into the management workstation as the root user use the following credentials:

   Username: **root**
   Password: **linux**

Summary:

In this exercise, you started the lab VMs in the required order. You then log into the management workstation as the tux user.

(End of Exercise)

2 Introduction to Containers and Container Orchestration

Description:

In this section you are introduced to container and container orchestration concepts.

(No Exercises)

3 Kubernetes Administration

Description:

In this section you are introduced to and you perform Kubernetes administration tasks.

3– 1 Use Basic `kubectl` Commands

Description:

In this exercise you take your first steps with the **`kubectl`** command. The **`kubectl`** command is primary user CLI interface into a Kubernetes cluster.

Task 1: List Commands

1. On the **management workstation**, in a terminal, enter the following command to list all nodes in your cluster:

   ```
   kubectl get nodes
   ```

   You should see a list of all of your master and worker nodes.

2. Enter the following command to list all namespaces:

   ```
   kubectl get namespaces
   ```

   You should see a list of all of the default namespaces:

   ```
   default
   kube-node-lease
   kube-public
   kube-system
   ```

   You may potentially see some additional namespaces depending on how the cluster was deployed and what has been deployed on it.

3. Enter the following command to list all of the pods in the **kube-system** namespace:

   ```
   kubectl --namespace kube-system get pods
   ```

   You should see a list of running pods. Many pods will have a standard first name and a randomized last name. Others will specify which node they are assigned to.

4. Enter the following command to list all of the services in the default namespace:

```
kubectl get services
```

You should see the standard Kubernetes service and any additional services that you might have created in this namespace.

5. Re-run the previous four steps but add the following to the end of the command:

```
-o wide
```

You should see a more verbose output for each command. This can aid greatly when troubleshooting some functions in Kubernetes. Not every command will provide more information.

6. Enter the following command to list all of the **api-resources** in Kubernetes. These are all things that can be listed and used in Kubernetes.

```
kubectl api-resources
```

All of the entries from this list can be used with the kubectl get command though not all of them will have resources on your cluster yet. Add **`-o wide`** to this command to get a list of verbs that can be used with each resource.

Task 2: Describe Commands

1. On the **management workstation**, in a terminal, enter the following command to get the status of your cluster:

```
kubectl cluster-info
```

You should get some basic information about your cluster. To get a detailed dump of cluster information, run:

```
kubectl cluster-info dump
```

2. Enter the following command to get information about the default namespace:

```
kubectl describe namespace default
```

You should see all of the available information about this namespace including **status**, **quotas**, and **resource limits**.

3. Enter the following command to display a list of the pods running in the kube-system namespace:

```
kubectl -n kube-system get pods
```

You should see the pods running in the namespace.

Choose one of the pods to use in the next step.

4. Enter the following command to get information about the pod you selected in the previous step:

   kubectl -n kube-system describe pod *POD_YOU_SELECTED*

   You should see detailed information about this pod. Any time a pod is not in a ready status, this is the first command that should be run.

> **Note**:
>
> The describe command always needs to know what kind of resource it is describing. **kubectl describe** will not work but **kubectl describe namespace** or **kubectl describe pod** will work because the resource is included in the command.

Task 3: Creation and Deletion Commands

1. On the management workstation, in a terminal, enter the following command to create a namespace:

   cd ~/course_files/KUB201/labs/manifests/simple-pod/

   kubectl create -f simple-pod.yaml

   You should see the following output to confirm that it has been installed:

   pod/simple-pod created

2. Enter the following command to display the running pods:

   kubectl get pods

   You should see the pod listed as running.

3. Enter the following command to delete a simple pod:

   kubectl delete pod simple-pod

   Enter the following command to confirm that the pod is no longer running:

4. Enter the following command to deploy the pod again

   kubectl apply -f simple-pod.yaml

The pod should be created.

5. Check that the pod is running again:

```
kubectl get pods
```

Note:

The **kubectl create** and **apply** commands can sometimes be used interchangeably. However the create commands can create new resources from the command line and the **apply** command can update existing resources with manifests files.

Task 4: Troubleshooting Commands

1. On the management workstation, in a terminal, enter the following command to get the application logs from the kube-apiserver-master01 pod:

```
kubectl logs simple-pod
```

Your output will vary depending on the cluster but you should see log entries for the api server application running in the simple-pod pod.

2. Enter the following command to enter into the pod filesystem from the previous task:

```
kubectl exec -it simple-pod -- bash
```

You should now have a new root command prompt inside the pod.

Run the following command to leave this shell:

```
exit
```

If this were your application, you could troubleshoot any issues with it directly.

3. Enter the following command to delete the pod now that you are finished with it:

```
kubectl delete pod simple-pod
```

The pod should have been deleted.

Summary:

In this exercise you learned how to use some listing, describing, creating, troubleshoot, and deleting commands. These are useful for any Kubernetes administrator but they are only the tip of what Kubernetes can do.

(End of Exercise)

3- 2 Work with Namespaces in Kubernetes

Description:

In this exercise you list, create and delete namespaces.

Task 1: List Namespaces

1. On the management workstation, in a terminal, enter the following command to list the namespaces that currently exist:

   ```
   kubectl get namespaces
   ```

   You should see, among others, namespces such as **`default`** and **`kube-system`**.
2. Enter the following command to list the pods in the current namespace:

   ```
   kubectl get pods
   ```

   You should see the message "No resources found int the default namespace.". This tells you a couple things. First, your default namespace is named "default". Second there are not pods currently running in that namespace.
3. Enter the following command to list the pods in the kube-system namespace:

   ```
   kubectl -n kube-system get pods
   ```

   This time you should see a number of pods listed.

Task 2: Create a Namespace

1. Enter the following command to create a new namespace:

   ```
   kubectl create namespace mynamespace
   ```

   The namespace should have been created.
2. Enter the following command to list that namespaces again:

```
kubectl get namespaces
```

You should see your new namespace listed.

Task 3: Delete a Namespace

1. Enter the following command to delete the new namespace:

   ```
   kubectl delete namespace mynamespace
   ```

   The namespace should have been deleted.

   Note that when deleting a namespace, all objects in the namespace also get deleted. (In this case there were no additional objects in the namespace.)

2. Enter the following command to list that namespaces again:

   ```
   kubectl get namespaces
   ```

   You should no longer see your new namespace listed.

Summary:

In this exercise you first listed the current namspaces. You then created a new namspace, verified that it existed, then deleted the namespace and verified it was deleted.

(End of Exercise)

3– 3 Deploy a Simple Pod Using a Deployment

Description:

In this exercise you deploy the Nginx web server as a simple pod on the Kubernetes cluster.

Task 1: Examine a Manifest for the Deployment

1. On the management workstation, in a terminal, enter the following commands to change to the directory containing the **nginx** manifests and display the contents of the **nginx-deployment.yaml** file:

```
cd ~/course_files/KUB201/labs/manifests/nginx

cat nginx-deployment.yaml
```

You should see the following:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
   env: "app"
   owner: nginx
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 4
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
```

```
ports:
- containerPort: 80
```

Notice the **image:** that is being used and the number of replicas that will be created. Close the file when done.

Task 2: Deploy the Pod

1. To deploy the pod, open a terminal and enter the following command:

   kubectl apply -f nginx-deployment.yaml

   You should see the deployment nginx-deployment was created.
2. Enter the following command to view the deployments:

   kubectl get deployments

   You should see four instances of the nginx-deployment pod are running. (Note that it may take a few seconds for all instances to be READY).
3. Enter the following command to view the deployed pods:

   kubectl get pods

   You should see a four instances of the nginx-deployment pod running or being created.

Summary:

In this exercise, you launched a four instances of the Nginx web server as pods in a deployment on the cluster.

(End of Exercise)

3- 4 Delete and Redeploy a Deployment

Description:

In this exercise, you delete the Nginx web server deployment that was previously deployed on the Kubernetes cluster.

Task 1: Delete the Deployment

1. To view the deployments, on the management workstation, enter the following command:

   ```
   kubectl get pods
   ```

   You should see that one or more instance of the pod in the nginx-deployment is running.

   Record the name of the first nginx pod here:

   NGINX_POD1=_____

2. Enter the following command to delete the first pod from the deployment:

   ```
   kubectl delete pod NGINX_POD1
   ```

   The pod should have been deleted.

3. Enter the following command to list the pods in the deployment again:

   ```
   kubectl get pods
   ```

   Notice that the pod you deleted is gone but a new pod was created to take its place.

4. Enter the following command to delete the deployment:

   ```
   kubectl delete deployment nginx-deployment
   ```

   You should see the nginx-deployment was deleted.

5. View the deployments again:

   ```
   kubectl get deployments
   ```

You should see that the nginx-deployment is no longer running.

6. View the pods:

   ```
   kubectl get pods
   ```

   You should see that all of the pods that were part of the nginx-deployment are no longer running or may be in the process of terminating.

Task 2: Redeploy the Deployment

1. To redeploy the deployment enter the following commands:

   ```
   cd ~/course_files/KUB201/labs/manifests/nginx/
   ```

   ```
   kubectl apply -f nginx-deployment.yaml
   ```

   You should see the deployment "nginx-deployment" was created.

2. Enter the following command to view the deployments:

   ```
   kubectl get deployments
   ```

   You should see that four instances of the nginx-deployment pod are running or are being created.

3. Enter the following command to view the deployed pods:

   ```
   kubectl get pods
   ```

   You should see four instances of the nginx-deployment pod running.

Summary:

In this exercise you deleted a single pod from the Nginx web server deployment and saw that it was replaced. You then deleted the entire deployment. Finally you redeployed the deployment.

(End of Exercise)

3- 5 Update Pods in a Deployment

Description:

In this exercise, you update a running pod in a deployment.

Task 1: Create a New Manifest for the Deployment

1. On the management workstation, enter the following commands to make a copy of the **nginx-deployment.yaml** file:

```
cd ~/course_files/KUB201/labs/manifests/nginx/

cp nginx-deployment.yaml nginx-update.yaml
```

2. In the text editor of your choice, open the **nginx-update.yaml** file
3. Edit the file to match the following (changes are in **red**):

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    env: "app"
    owner: nginx
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 4
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.9.0
        ports:
```

```
- containerPort: 80
```

4. Save the file and close the text editor

Task 2: Update the Deployment

1. To display information on the current nginx deployment enter the following command:

   ```
   kubectl describe deployment nginx-deployment
   ```

   You should see the description of the nginx deployment displayed.

   Notice the image version is: **nginx:1.7.9**

2. Open another terminal and enter the following command to watch the running pods:

   ```
   watch kubectl get pods
   ```

   You should see a list of the running pods displayed with the list updating every 2 seconds.

3. To update the deployment, open a terminal and enter the following command:

   ```
   kubectl apply -f nginx-update.yaml
   ```

   You should see the deployment nginx-deployment was configured.

4. Enter the following command to view the deployments:

   ```
   kubectl get deployments
   ```

   You should see that 4 instances of the nginx-deployment pod are DESIRED and, depending on when you ran the command, the values in the CURRENT, UP-TO-DATE and AVAILABLE columns my be more, fewer or the same number as the update happens.

5. In the terminal where you are watching the pods enter **Ctrl+c** to stop the watch command

6. Enter the following command to display information about the running deployment of nginx:

   ```
   kubectl describe deployment nginx-deployment
   ```

   Notice the image version is now: **nginx:1.9.0**

7. In the terminal where you are watching the pods enter **`Ctrl+c`** to stop the watch command

---

Summary:

In this exercise, you created a new manifest to update the running nginx deployment. You then updated the deployment and verified that it was updated.

(End of Exercise)

3- 6 Create and Edit a Service for an Application

Description:

In this exercise, you expose a service running in a pod and then edit that service.

Task 1: Examine a Manifest for the Service

1. On the management workstation, in a terminal, enter the following commands to change to the directory containing the **nginx** manifests and display the contents of the **nginx-service.yaml** file:

```
cd ~/course_files/KUB201/labs/manifests/nginx/

cat nginx-service.yaml
```

You should see the following:

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
  type: NodePort
  ports:
    - port: 80
      nodePort: 30000
  selector:
      app: nginx
```

Review the contents of the file. Notice how the type of service is **NodePort** and what port it is listening on.

Task 2: Define the Service

1. To define the service in the cluster, open a terminal and enter the following

command:

```
kubectl apply -f nginx-service.yaml
```

You should see the service nginx-service was created.

If you still have an nginx-deployment running from a previous exercise proceed to the next step.

If a previous nginx-deployment is not running, enter the following command to define an nginx web server to test the service:

```
kubectl apply -f nginx-deployment.yaml
```

2. Enter the following command to display the services:

```
kubectl get services
```

You should see that the nginx-service service is defined. Notice the 80:30000 under ports showing external port 30000 will be redirected into internal port 80. This 30000 is the NodePort which you saw in the service manifest.

Task 3: Access the Exposed Service

1. On the management workstation, in the browser of your choice, go to:

http://worker01.example.com:30000

You should see the "Welcome to nginx" web page.

Task 4: Edit the Service

1. Enter the following command to edit the nginx service:

```
kubectl edit service nginx-service
```

You should be presented with the yaml output in your default terminal editor.

2. Find the section of the file titled: spec. Change the NodePort port and then save your file:

The spec section of the file should be similar to this:

```
spec:
  type: NodePort
  ports:
    - port: 80
      nodePort: 31000
```

```
  selector:
      app: nginx
```

Change the nodePort value in red. After saving and exiting, you should see the message:

```
service/nginx-service edited
```

3. Enter the following command to display the services:

```
kubectl get services
```

You should see that the nginx-service service now has the new port.

Note:

The `kubectl edit` command is not limited to only editing services. Other Kubernetes functions can be edited on demand for troubleshooting purposes.

Task 5: Review Service

1. On the management workstation, in the browser of your choice go to:

worker01.example.com:31000

You should see the same nginx "Welcome to nginx" web page but with the new nodePort.

Task 6: Delete the Service

1. Enter the following command to delete the nginx service:

```
kubectl delete service nginx-service
```

The service should have been deleted.

2. Enter the following command to display the services:

```
kubectl get services
```

You should no longer see the service listed.

Summary:

In this exercise, you defined a NodePort service in the cluster exposing NodePort 30000 that allowed access to the nginx pod running on the cluster. You then tested the nginx pod in a web browser. You then edited the service yaml and changed the NodePort to a different port and tested it again. Finally you deleted the service.

(End of Exercise)

3- 7 Use Environment Variables in a Pod

Description:

In this exercise, you will deploy a simple container setting an environment variable in the container in the process.

Task 1: Examine the Manifest for the Pod

1. On the management workstation, in a terminal, enter the following commands to change to the directory containing the **envar** manifest and display the contents of the **envar-demo.yaml** file:

```
cd ~/course_files/KUB201/labs/manifests/envars/

cat envar-demo.yaml
```

You should see the following:

```
kind: Pod
apiVersion: v1
metadata:
  name: envar-demo
  labels:
    purpose: demonstrate-envars
spec:
  containers:
    - name: envar-demo-container
      image: gcr.io/google-samples/node-hello:1.0
      env:
      - name: DEMO_GREETING
        value: "SUSE Rocks!"
```

Notice the environment variable, DEMO_GREETING will be given the value "SUSE Rocks!" in the pod when it is deployed.

Task 2: Deploy the Pod

1. To deploy the pod, open a terminal and enter the following commands:

   ```
   kubectl apply -f envar-demo.yaml
   ```

   You should see the deployment envar-demo was created.
2. Enter the following command to view the deployed pods:

   ```
   kubectl get pods
   ```

   You should see a single instance of the envar-demo pod running.

Task 3: Display the Environment Variable in the Container

1. To enter the pod, open a terminal and enter the following command:

   ```
   kubectl exec -it envar-demo -- bash
   ```

   You should now be at a bash prompt in the container.
2. Enter the following command to display the environment variables:

   ```
   printenv
   ```

   You should see the environment variables that are set in the container. In this list you should see the DEMO_GREETING variable is set to “SUSE Rocks!”.
   If you have trouble seeing it, try this:

   ```
   printenv | grep SUSE
   ```
3. Enter the following command to exit the container:

   ```
   exit
   ```

   You should be back on the management workstation.

Task 4: Delete the Pod

1. Enter the following command to delete the pod:

   ```
   kubectl delete pod envar-demo
   ```

   You should see the pod envar-demo was deleted.

Summary:

In this exercise you launched a simple container setting an environment variable in the container in the process. You then launched a bash shell inside the container and displayed the environment variable that was set.

(End of Exercise)

3- 8 Use ConfigMaps with a Pod

Description:

In this exercise, you will deploy a simple container setting environment variables using a ConfigMap in the container in the process.

Task 1: Display the Manifests for the ConfigMap and Pod

1. On the management workstation, in a terminal, enter the following commands to change to the directory containing the ConfigMap related manifests and display the contents of the **my-configmap.yaml** file:

   ```
   cd ~/course_files/KUB201/labs/manifests/configmaps/
   ```

   ```
   cat my-configmap.yaml
   ```

   You should see the following:

   ```
   kind: ConfigMap
   apiVersion: v1
   metadata:
     name: my-configmap
   data:
     CONFIGMAP_VAR1: configmap_value_1
     CONFIGMAP_VAR2: configmap_value_2
   ```

   Notice the two variables that are being set along with their values.

2. Enter the following command to display the contents of the **configmap-demo.yaml** file:

   ```
   cat configmap-demo.yaml
   ```

   You should see the following:

   ```
   apiVersion: v1
   kind: Pod
   metadata:
     name: configmap-demo
   ```

```
  labels:
    purpose: demonstrate-configmaps
spec:
  containers:
    - name: configmap-demo-container
      image: gcr.io/google-samples/node-hello:1.0
      envFrom:
        - configMapRef:
            name: my-configmap
```

Notice the pod will read the environment variables from the configMap named **my-configmap** when the pod is deployed.

Task 2: Deploy the configMap and Pod

1. To deploy the pod, open a terminal and enter the following command:

   ```
   kubectl apply -f configmap-demo.yaml
   ```

   You should see the deployment configmap-demo was created.
2. Enter the following command to view the deployed pods:

   ```
   kubectl get pods
   ```

   You should see a single instance of the configmap-demo pod running.

Task 3: Display the Environment Variables in the Container

1. To enter the pod, open a terminal and enter the following command:

   ```
   kubectl exec -it configmap-demo -- bash
   ```

   You should now be at a bash prompt in the container.
2. Enter the following command to display the environment variables:

   ```
   printenv
   ```

   You should see the environment variables that are set in the container. In this list you should see the CONFIGMAP_VAR1 and CONFIGMAP_VAR2 variables are set to their corresponding values.

   If you have trouble seeing it, try this:

   ```
   printenv | grep CONFIGMAP_VAR
   ```

3. Enter the following command to exit the container:

   ```
   exit
   ```

   You should be back on the management workstation.

Task 4: Delete the Pod

1. Enter the following command to delete the pod:

   ```
   kubectl delete pod configmap-demo
   kubectl delete configmap my-configmap
   ```

   You should see the pod configmap-demo pod and my-configmap ConfigMap were deleted.

Summary:

In this exercise you launched a simple container setting environment variables in the container though a configMap in the process. You then launched a bash shell inside the container and displayed the environment variables that were set.

(End of Exercise)

3- 9 Define and Access Secrets as Volumes

Description:

In this exercise you define a secret using **text file** and then retrieve it using the **kubectl** command. You then access the secret as a volume in a pod.

Task 1: Define the Secret

1. On the management workstation, in a terminal, enter the following commands to change to the directory containing the secrets files manifests and display the contents of the **mysecret1.txt** file:

   ```
   cd ~/course_files/KUB201/labs/manifests/secrets/

   cat mysecret1.txt
   ```

   You should see the following:

   ```
   username: user1
   password: password1
   ```

2. Create a secret from the **mysecret1.txt** file:

   ```
   kubectl create secret generic mysecret1 \
     --from-file=mysecret1.txt
   ```

3. Confirm that the secret is available:

   ```
   kubectl get secrets
   ```

   You should see your new secret with an Opaque type.

4. Enter the following command to display information about the secret:

   ```
   kubectl describe secret mysecret1
   ```

   You should see that **Data** in the secret is a single file named **mysecret1.txt** that is 36 bytes in size.

Task 2: Deploy a Pod that Uses the Secret

1. Enter the following command to examine the **podsecret1.yaml** manifest:

   ```
   cat podsecret1.yaml
   ```

   You should see the following:

   ```
   apiVersion: v1
   kind: Pod
   metadata:
     name: podsecret1
   spec:
     containers:
     - name: opensusepod
       image: opensuse/leap
       command:
         - "bin/bash"
         - "-c"
         - "sleep 10000"
       volumeMounts:
         - name: secretmnt
           mountPoint: "/mnt"
     volumes:
     - name: secretmnt
       secret:
         secretName: mysecret1
   ```

   Notice the **volumes** and **volumeMount** sections in the manifest.

2. Enter the following command to deploy the test pod:

   ```
   kubectl apply -f podsecret1.yaml
   ```

   You should see the pod podsecret1 was created.

Task 3: Access the Secret in the Pod

1. You can access the newly created secret by entering into the pod:

   ```
   kubectl exec -it podsecret1 bash
   ```

   You should now be inside of the podsecret1 container and you should see a new command prompt.

2. Enter the following command to review the secret that you imported into the pod

   ```
   cat /mnt/mysecret1.txt
   ```

You should see the contents of **`mysecret1.txt`** displayed.

The secret will only be available in kubernetes to users who have access to it. It is also only available to processes running in this app. Other apps and users without permission would not be able to see it.

3. Exit the shell to return to your normal command line

Task 4: Delete the Pod and the Secret

1. Enter the following command to delete the pod:

   ```
   kubectl delete pod podsecret1
   ```

   The pod should be deleted.

2. Enter the following command to delete the secret:

   ```
   kubectl delete secret mysecret1
   ```

   The secret should have been deleted.

Summary:

In this exercise you defined a secret from a text file in the cluster. You then exposed that secret to a pod as a volume and then accessed the secret in the pod.

(End of Exercise)

3– 10 Define and Access Secrets as Environment Variables

Description:

In this exercise, you will define a secret using a manifest and then retrieve it using the kubectl command. You then access the secret values as environment variables in a pod.

Task 1: Define the Secret

1. On the management workstation, in a terminal, enter the following commands to change to the directory containing the secrets files manifests and display the contents of the **mysecret2.yaml** file:

   ```
   cd ~/course_files/KUB201/labs/manifests/secrets/

   cat mysecret2.yaml
   ```

   You should see the following:

   ```
   apiVersion: v1
   kind: Secret
   metadata:
     name: mysecret2
   type: Opaque
   data:
     # username=${echo "user2" | base64)
     # password=${echo "password2" | base64)
     username: dXNlcjIK
     password: cGFzc3dvcmQyCg==
   ```

   Notice the comments that describe how the username and password values where generated using the **base64** command. Data stored in **data:** fields must be base64 encoded. The values will be decoded when they are accessed.

2. Create a secret from the **mysecret2.yaml** file:

   ```
   kubectl apply -f mysecret2.yaml
   ```

3. Confirm that the secret is available:

   ```
   kubectl get secrets
   ```

   You should see your new secret with an Opaque type.

4. Enter the following command to display information about the secret:

   ```
   kubectl describe secret mysecret2
   ```

   You should see that **Data** in the secret is a **password** that is 10 bytes in size and a **username** that is 5 bytes in size.

Task 2: Deploy a Pod that Uses the Secret

1. Enter the following command to examine the **podsecret2.yaml** manifest:

   ```
   cat podsecret2.yaml
   ```

   You should see the following:

   ```
   apiVersion: v1
   kind: Pod
   metadata:
     name: podsecret2
   spec:
     containers:
     - name: opensusepod
       image: opensuse/leap
       command:
         - "bin/bash"
         - "-c"
         - "sleep 10000"
       env:
         - name: SECRET_USERNAME
           valueFrom:
             secretKeyRef:
               name: mysecret2
               key: username
         - name: SECRET_PASSWORD
           valueFrom:
             secretKeyRef:
               name: mysecret2
               key: password
   ```

   Notice the **env:** section defining 2 environment variables in the manifest.

2. Enter the following command to deploy the test pod:

```
kubectl apply -f podsecret2.yaml
```

You should see the pod podsecret2 was created.

Task 3: Access the Secret in the Pod

1. You can access the newly created secret by entering into the pod:

   ```
   kubectl exec -it podsecret2 -- bash
   ```

   You should now be inside of the podsecret2 container and you should see a new command prompt.
2. Enter the following commands to review the secret values that you imported into the pod

   ```
   echo $SECRET_USERNAME
   ```

   You should see the value of **user2** displayed.

   ```
   echo $SECRET_PASSWORD
   ```

   You should see the value of **password2** displayed.

   The secret will only be available in Kubernetes to users who have access to it. It is also only available to processes running in this app. Other apps and users without permission would not be able to see it.
3. Exit the shell to return to your normal command line

Task 4: Delete the Pod

1. Enter the following command to delete the pod:

   ```
   kubectl delete pod podsecret2
   ```

   The pod should be deleted.
2. Enter the following command to delete the secret:

   ```
   kubectl delete secret mysecret2
   ```

   The secret should have been deleted.

Summary:

In this exercise, you defined a secret from a manifest in the cluster. You then exposed that secret to a pod as environment variables and then accessed the secret values in the pod.

(End of Exercise)

3- 11 Work with Labels and Selectors

Description:

In this exercise you will deploy a set of pods with predefined labels. You will then use selectors to select specific labels and then delete those pods.

Task 1: Deploy Pods

1. On the management workstation, in a terminal, enter the following command to deploy the pods used for this exercise:

   ```
   kubectl apply -f ~/course_files/KUB201/labs/manifests/labels/
   ```

2. Confirm that the pods are available:

   ```
   kubectl get pods
   ```

   You should now see **pod1** through **pod5** created. If they do not have the status "Running", wait a few second and rerun the command. Wait until all pods are Running before continuing with the exercise.

Task 2: Select Pods by Band

1. On the management workstation, in a terminal, enter the following command to select pods with the band of **beatles**:

   ```
   kubectl get pods --selector band=beatles
   ```

   You should only see **pod3** and **pod4** listed.

   Why weren't pod 1 and pod 2 listed? Aren't they Beatles?

   Enter the following commands to display more information about pods1 and pod2:

   ```
   kubectl describe pod pod1
   ```

   Notice the typo in the labels section: **bans=beatles**

```
kubectl describe pod pod2
```

Notice the different label name: **owner=beatles**

2. Select pods with the owner of **monkees**:

```
kubectl get pods --selector band=monkees
```

You should only see **pod5**.

Task 3: Create a Label

1. On the management workstation, in a terminal, enter the following command to apply a new label to **pod5**:

```
kubectl label pods pod5 surname=jones
```

2. Select pods with the surname label with a value of **jones**:

```
kubectl get pods -l surname=jones
```

You should only see **pod5.** The option **-l** is a shorter way of using a selector than **--selector**.

Task 4: Delete Pods by Label

1. On the management workstation, in a terminal, enter the following command to view all pods and their labels:

```
kubectl get pods --show-labels
```

You should see that all of the pods have some different labels but a few have the same band.

2. Delete all of the pods with the band of **beatle** and **monkee**:

```
kubectl delete pods -l 'band in (beatle, monkee)'
```

```
kubectl get pods
```

Notice that only pods 3-5 were deleted because they are the only ones that match the labels **band=beatles** and **band=monkees**.

3. Delete the remaining pods and confirm that all pods have been deleted:

```
kubectl delete -f ./
```

```
kubectl get pods
```

Summary:

In this exercise you deployed a set of pods, reviewed their labels, added a label, and finally deleted them by their label.

(End of Exercise)

3– 12 Work with Node Selectors

Description:

In this exercise you will attach labels to nodes. You will then deploy a set of pods with predefined node selectors related to those labels and view where the pods are deployed.

Task 1: Attach Labels to Nodes

1. On the management workstation, in a terminal, enter the following commands to attach labels to some of the nodes:

   ```
   kubectl label node worker01.example.com band=beatles
   ```

   ```
   kubectl label node worker02.example.com band=monkees
   ```

2. Confirm that the labels are available:

   ```
   kubectl get nodes --show-labels
   ```

   You should see the nodes listed with their assigned labels.

3. Enter the following command to display more detailed information about the worker01 node:

   ```
   kubectl describe node worker01.example.com
   ```

   You should see more detailed information about the node displayed.
   Notice the assigned labels in the Labels: section. Also notice the pods that are currently running on the node in the Non-terminated Pods: section.
   If desired, run the command for the other worker nodes (**worker02**, **worker03**).

Task 2: Deploy Pods

1. On the management workstation, in a terminal, enter the following commands to deploy the pods used for this exercise:

```
cd ~/course_files/KUB201/labs/manifests/nodeselectors/

kubectl apply -f ./
```

2. Confirm that the pods are available:

   ```
   kubectl get pods
   ```

   You should now see pods with names such as **paul**, **davy**, **john**, etc. were created. If they do not have the status "**Running**", wait a few second and rerun the command. Wait until all pods are Running before continuing with the exercise.

Task 2: Select Pods by Band

1. On the management workstation, in a terminal, enter the following command to select pods with the band of **beatles**:

   ```
   kubectl get pods -o wide --selector band=beatles
   ```

   You should only see pods **john**, **paul**, **george** and **ringo** listed.

   Notice which node they are running on.

2. Select pods with the band of **monkees**:

   ```
   kubectl get pods -o wide --selector band=monkees
   ```

   You should only see pods **davy**, **micky**, **michael** and **peter** listed.

   Notice which node they are running on.

Task 4: Delete Pods by Label

1. On the management workstation, in a terminal, enter the following command to view all pods and their labels:

   ```
   kubectl get pods --show-labels
   ```

   You should see that all of the pods have some different label but a few have the same band.

2. Delete all of the pods with the band of **beatles** and **monkees**:

   ```
   kubectl delete pods -l 'band in (beatles, monkees)'
   ```

3. Confirm that all pods have been deleted:

   ```
   kubectl get pods
   ```

Summary:

In this exercise you attached labels to some nodes. You then deployed a set of pods, reviewed their labels, and location and finally deleted them by their label.

(End of Exercise)

3– 13 Work with Taints and Tolerations

Description:

In this exercise you will taint nodes with specific restrictions. You will then deploy a set of pods with predefined tolerations related to those taints and view where the pods are deployed.

Task 1: Add Taints to Nodes

1. On the management workstation, in a terminal, enter the following commands to taint the nodes relative to a specific band:

```
kubectl taint node worker01.example.com \
  band=beatles:NoSchedule

kubectl taint node worker02.example.com \
  band=monkees:NoSchedule

kubectl taint node worker03.example.com \
  band=beachboys:NoSchedule
```

2. Enter the following command to display more detailed information about the worker01 node:

```
kubectl describe node worker01.example.com
```

You should see more detailed information about the node displayed.

Notice the pods associated with the node in the Taints: section. Also notice the pods that are currently running on the node in the Non-terminated Pods: section.

Run the command for the other worker nodes (**worker02**, **worker03**).

Task 2: Deploy Pods

1. On the management workstation, in a terminal, enter the following

commands to deploy the pods used for this exercise:

```
cd ~/course_files/KUB201/labs/manifests/taints_tolerations/

kubectl apply -f ./
```

2. Confirm that the pods are available:

   ```
   kubectl get pods
   ```

   You should now see pods with names such as **paul**, **davey**, **john**, etc. were created. If they do not have the status "**Running**", wait a few second and rerun the command. Wait until all pods are Running before continuing with the exercise.

Task 3: Select Pods by Band

1. On the management workstation, in a terminal, enter the following command to select pods with the band of **beatles**:

   ```
   kubectl get pods -o wide --selector band=beatles
   ```

   You should only see pods **john**, **paul**, **george** and **ringo** listed.

   Notice which node they are running on.

2. Select pods with the band of **beachboys**:

   ```
   kubectl get pods -o wide --selector band=beachboys
   ```

   You should only see pods **al**, **brian, carl** , **dennis** and **mike** listed.

   Notice which node they are running on.

3. Select pods with the band of **monkees**:

   ```
   kubectl get pods -o wide --selector band=monkees
   ```

   You should only see pods **davy**, **micky**, **michael** and **peter** listed.

   Notice that they are still in pending state and not running anywhere.

4. In the text editor of your choice, open the manifests for the **monkees** pods (**davy**, **micky**, **michael**, **peter**) to be edited

   Using the other manifests ad a guide, update the **monkees** pods so that they will run on the **worker01** node.

5. Enter the following command to delete the **monkees** pods:

   ```
   kubectl delete pods -l band=monkees
   ```

The pods should be deleted.

6. Redeploy the **monkees** pods:

```
kubectl apply -f davy.yaml
kubectl apply -f micky.yaml
kubectl apply -f michael.yaml
kubectl apply -f peter.yaml
```

7. Select pods with the band of **monkees**:

```
kubectl get pods -o wide --selector band=monkees
```

You should only see pods **davy**, **micky**, **michael** and **peter** listed.

Are they running on the worker01 node? (If not, examine and correct the manifests for the pods, delete and redeploy them.)

Task 4: Delete Pods by Label

1. On the management workstation, in a terminal, enter the following command to view all pods and their labels:

```
kubectl get pods --show-labels
```

You should see that all of the pods have some different labels but a few have the same band.

2. Delete all of the pods with the band of **beatles**, **beachboys** and **monkees**:

```
kubectl delete pods -l 'band in (beatles, monkees, beachboys)'
```

3. Confirm that all pods have been deleted:

```
kubectl get pods
```

Task 5: Remove Taints from Nodes

1. On the management workstation, in a terminal, enter the following commands to taint the nodes relative to a specific band:

```
kubectl taint node worker01.example.com \
  band=beatles:NoSchedule-

kubectl taint node worker02.example.com \
  band=monkees:NoSchedule-
```

```
kubectl taint node worker03.example.com \
  band=beachboys:NoSchedule-
```

2. Enter the following command to display more detailed information about the worker01 node:

   ```
   kubectl describe node worker01.example.com
   ```

   In the Taints: section you should see **<none>** showing that there are no longer taints on the node.

   Repeat the command for the other nodes (**worker02**, **worker03**). You should see the same on those nodes.

Summary:

In this exercise you added taints to some nodes. You then deployed a set of pods, reviewed their labels, and check if and where they were running. You then updated the manifests to correct errors, redeployed some pods . Finally you deleted the pods by their labels.

(End of Exercise)

3– 14 Scale a Deployment

Description:

In this exercise, you scale out a running pod.

Dependencies

The exercise named **Update Pods in a Deployment** must be completed before performing this exercise.

Task 1: Examine a New Manifest for the Deployment

1. On the management workstation, in a terminal, enter the following commands to change to the directory containing the nginx manifests and display the contents of the **nginx-scale.yaml** file:

```
cd ~/course_files/KUB201/labs/manifests/nginx/

cat nginx-scale.yaml
```

You should see the following:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 2
  revisionHistoryLimit: 5
  minReadySeconds: 20
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 25%
      maxSurge: 2
```

```
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.9.0
        ports:
        - containerPort: 80
```

Take note of the **replicas:**, **revisionHistoryLimit:** and **strategy:** sections.

Task 2: Scale the Deployment

1. To scale the deployment, open a terminal and enter the following command:

   kubectl apply -f nginx-scale.yaml

   You should see the deployment nginx-deployment was configured.
2. Enter the following command to display the deployments:

   kubectl get deployments

   You should see that two instances of the nginx-deployment pod are now running.
3. Enter the following command to display the deployed pods:

   kubectl get pods

   You should see two instances of the nginx-deployment pod running.

Task 3: Manually Scale the Deployment Out

1. To manually scale the deployment back out, open a terminal and enter the following command:

   kubectl scale deployment nginx-deployment --replicas=10
2. Enter the following command to display the deployments:

   watch kubectl get deployments

   This will continuously run the get command. You should see that ten instances

of the nginx-deployment pod are running after a few seconds.

Hit **Ctrl+c** to close this screen.

3. Enter the following command to display the deployed pods:

   kubectl get pods

   You should see ten instances of the nginx-deployment pod running or being created.

4. Enter the following command to scale the deployment back to its original:

   kubectl apply -f nginx-update.yaml

   The deployment should have been scaled back.

5. Enter the following command to display the deployed pods:

   kubectl get pods

   You should see only 4 pods.

6. Remove the deployment:

   kubectl delete -f nginx-update.yaml

Summary:

In this exercise you examined a new manifest for an existing deployment that specified a smaller number of replicas. You then applied the updated manifest to scale in the deployment. Finally you applied the original manifest to scale the deployment back out and then deleted the entire deployment.

(End of Exercise)

3– 15 Configure Horizontal Pod Autoscaling

Description:

In this exercise, you configure and then test horizontal autoscaling of pods in a deployment.

Task 1: Deploy the AutoScaler Manifests

1. On the management workstation, in a terminal, enter the following commands to deploy the pods/services:

   ```
   cd ~/course_files/KUB201/labs/manifests/hpa
   ```

   ```
   kubectl apply -f app/
   ```

   You should see the following were created (not necessarily in this order):

   ```
   deployment.apps/php-apache created
   service/php-apache created
   horizontalpodautoscaler.autoscaling/php-apache created
   ```

2. Enter the following command to view the deployments:

   ```
   watch kubectl get deployments
   ```

   You should see the php-apache and load-generator deployments listed.

3. Open a second terminal and enter the following command to view the autoscalers:

   ```
   watch kubectl get hpa
   ```

   You should see the autoscaler php-apache listed. Notice the percentages in the TARGETS column. (It may take a minute or few for the percentage to show up on the left side of the **/** in the TARGETS column so be patient.)

4. Enter the following command to view the details of the autoscaler:

   ```
   kubectl describe hpa php-apache
   ```

You should see details about the autoscaler displayed.

5. Run the watch command for the autoscaler as well:

   ```
   watch kubectl get hpa
   ```

   The output should refresh every 2 seconds.

Task 2: Cause the Deployment To Scale Out

1. On the management workstation, in another terminal, enter the following command to deploy the load-generator deployment:

   ```
   kubectl apply -f load/
   ```

   You should see the following was created:

   ```
   deployment.apps/load-generator created
   ```

2. Look at the terminal that is watching the `kubectl get deployments` command

   Notice that when the percentages exceed 50% in the other window and the additional pods are created that the numbers here are adjusted accordingly. If the scale out goes on long enough notice that the numbers do not exceed the MAXPODS value defined in the autoscaler.

Task 3: Cause the Deployment To Scale Back

1. On the management workstation, in the second terminal, enter the following command to delete the load-generator deployment:

   ```
   kubectl delete -f load/
   ```

2. In the first terminal, after a short while you should see the percentages drop (you may see them increase before they drop due to a lag in the autoscaler getting data from the monitoring).

   A short while after the percentages drop you should see the number of php-apache pods decrease (this may take quite a while to happen so be patient).

Task 4: Clean Up the Deployments

1. In the first terminal, enter the following commands to delete all of the objects:

```
kubectl delete -f app/
```

In the terminals watching the other commands you should see the objects being removed.

2. Stop all of the watch commands in the other terminals by selecting the terminal and entering: **Ctrl+c**

Summary:

In this exercise you deployed manifests for an application, a service to export the application, and another application use to generate load and an autoscaler to scale the application. As the load increased the application was scaled out. Finally you scaled back the application and removed the objects.

(End of Exercise)

4 Application Management on Kubernetes with Kustomize

Description:

This section covers Kubernetes application management with Kustomize.

4– 1 Manage Applications with Kustomize

Description:

In this exercise you use kustomize via k**ubeclt apply -k** to deploy different application stacks.

Task 1: Examine the Base Manifests

1. On the management workstation, in a terminal, enter the following commands to change to the directory containing the kustomize manifests and display the contents of the directory tree:

   ```
   cd ~/course_files/KUB201/labs/manifests/nginx-kustomize/

   ls -l
   ```

   You should see a directory named **base** and one named **overlays**.

2. Enter the following command to display the contents of the base directory:

   ```
   ls -l base
   ```

   You should see the following files:

   ```
   deployment.yaml
   kustomization.yaml
   service.yaml
   ```

3. Enter the following command to display the contents of the **kustomization.yaml** file:

   ```
   cat base/kustomization.yaml
   ```

   You should see the following:

   ```
   apiVersion: kustomize.config.k8s.io/v1beta1
   kind: Kustomization
   resources:
     - deployment.yaml
   ```

```
  - service.yaml
```

Notice that the file refers to 2 resources, the deployment and the service. If you were to deploy the base application using kustomize these are what would be deployed.

4. Enter the following commands to display the contents of the deployment and service manifests:

```
cat base/deployment.yaml
```

You should see the following:

```
apiVersion:apps/v1
kind: Deployment
metadata:
  name: nginx-k-deployment
  labels:
    owner: examplecorp
spec:
  selector:
    matchLabels:
      app: nginx
    replicas: 1
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
          - containerPort: 80
```

Notice the name, labels, replicas and image version tag.

```
cat base/service.yaml
```

You should see the following:

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-k-service
spec:
  type: NodePort
  ports:
```

```
    - port: 80
  selector:
    app: nginx
```

Notice the name and the lack of NodePort.

Task 2: Deploy the Base Application

1. Enter the following command to deploy the base application:

   ```
   kubectl apply -k base
   ```

   The deployment and service should have been created.
2. Enter the following commands to display the deployments and services:

   ```
   kubectl get deployments
   ```

   Yous should see the **nginx-k-deployment** listed.

   ```
   kubectl get services
   ```

   You should see the **nginx-k-service** listed.

   Notice the nodePort assigned to the service is a randomly chosen port.
3. Enter the following command to display additional information about the deployment:

   ```
   kubectl describe deployment nginx-k-deployment
   ```

   Notice the name, labels, number of replicas and the container name and version.

Task 3: Examine the Overlay Manifests

1. Enter the following command display the contents of the overlays/prod directory tree:

   ```
   ls -l overlays/prod
   ```

   You should see the following files:

   ```
   kustomization.yaml
   nodeport.yaml
   replicas.yaml
   ```

2. Enter the following command to display the contents of the **kustomization.yaml** file:

   ```
   cat overlays/prod/kustomization.yaml
   ```

   You should see the following:

   ```
   apiVersion: kustomize.config.k8s.io/v1beta1
   kind: Kustomization
   namePrefix: prod-
   commonLabels:
     variant: products
   bases:
   - ../../base
   patches:
   - replicas.yaml
   - nodeport.yaml
   images:
     - name: nginx
       newTag: 1.9.0
   ```

   Notice the relative path to the base manifests, additional label, the patch files and the image with its version tag.

3. Enter the following commands to display the contents of the deployment and service manifests:

   ```
   cat overlays/prod/replicas.yaml
   ```

   You should see the following:

   ```
   apiVersion:apps/v1
   kind: Deployment
   metadata:
     name: nginx-k-deployment
   spec:
     replicas: 4
   ```

   Notice the greatly abbreviated deployment manifest with only the replicas property being set.

   ```
   cat overlays/prod/nodeport.yaml
   ```

   You should see the following:

   ```
   apiVersion: v1
   kind: Service
   metadata:
   ```

```
  name: nginx-k-service
spec:
  type: NodePort
  ports:
    - port: 80
      nodePort: 30100
```

Notice the greatly abbreviated service manifest with only the nodePort property being set.

4. Using the same commands as above, examine the **stage/** and **dev/** directory trees and files

   Note how they differ from the base and from each other.

Task 4: Deploy the Prod, Stage and Dev Applications

1. Enter the following command to deploy the base application:

   ```
   kubectl apply -k overlays/prod
   ```

   The deployment and service should have been created.

2. Enter the following commands to display the deployments and services:

   ```
   kubectl get deployments
   ```

   You should see the **prod-nginx-k-deployment** listed in addition to the base **nginx-deployment** deployment.

   ```
   kubectl get services
   ```

   You should see the **prod-nginx-k-service** listed in addition to the base **nginx-service** service.

   Notice the nodePort assigned to the service is not a randomly chosen port but the one specified in the overlay manifest.

3. Enter the following command to display additional information about the deployment:

   ```
   kubectl describe deployment prod-nginx-k-deployment
   ```

   Notice the name, labels, number or replicas and the container name and version. Notice how they are different from the base deployment.

4. Using the same commands as above, deploy the **stage** and **dev** version of the apps

Notice the difference from the base deployment/service and the other deployments/services.

Task 5: Delete the Base, Prod, Stage and Dev Applications

1. Enter the following commands to delete the **base**, **prod**, **stage** and **dev** applications:

   ```
   kubectl delete -k base

   kubectl delete -k overlays/prod

   kubectl delete -k overlays/stage

   kubectl delete -k overlays/dev
   ```

   The deployments and services should have been deleted.
2. Enter the following command to display the deployments and services:

   ```
   kubectl get deployments
   ```

   You should see none of the deployments.

   ```
   kubectl get services
   ```

   You should see none of the services

Task 6: (OPTIONAL) Experiment with Modifying the Different Applications

1. To explore kustomize in greater depth, try doing the following:
   - Modify the **prod** application to use a different application port and number of replicas without affecting the other applications by editing the least number of files.
   - Modify all of the applications to use a different application port by editing the least number of files.
   - Configure the **dev** application to deploy an additional busybox deployment in addition to the nginx deployment.

Summary:

In this exercise you examined the different manifests for the base application stack and prod, stage and dev application stacks. You deployed the base application stack as well as the prod, stage and dev application stacks and compared them. You then deleted all of the application stacks. Optionally you experimented with modifying the different application stacks.

(End of Exercise)

5 Application Management on Kubernetes with Helm

Description:

This section covers Kubernetes application management with Helm.

5– 1 Add a Repository to Helm

Description:

In this exercise you add the **Bitnami** Helm Repository.

Task 1: Add the Bitnami Repository

1. In a terminal, enter the following command to add the default repository to your Helm configuration:

   ```
   helm repo add bitnami https://charts.bitnami.com/bitnami
   ```

2. Enter the following command to confirm that the repository is now added:

   ```
   helm repo list
   ```

   You should now see the **bitnami** repository listed.

3. Enter the following command **to update** the **repositories**:

   ```
   helm repo update
   ```

   The repositories should have been updated.

Summary:

In this exercise you added the **Bitnami** repository to Helm.

(End of Exercise)

5- 2 Deploy an Application with Helm

Description:

In this exercise, you deploy an application from a Helm chart using the **helm** command.

Dependencies:

The bitnami Helm repository must be added before you perform this exercise.

Task 1: Create Helm Chart Config File

1. On the management workstation, in a terminal, enter the following command to search for a dockuwiki Helm chart:

   ```
   helm search repo dokuwiki
   ```

   You should see a helm chart named **bitnami/dokuwiki** listed with its available chart version.

2. Enter the following command to view the default configuration for the dokuwiki chart:

   ```
   helm inspect values bitnami/dokuwiki | less
   ```

   You should see the configuration displayed in the less pager. Page through the configuration to see what variables are being set.

3. In the text editor of your choice, create/open the **~/dokuwiki-values.yaml** file

4. Enter the following in the file:

   ```
   dokuwikiUsername: user
   dokuwikiPassword: password123
   service:
     type: NodePort
   persistence:
     enabled: false
   ```

```
#  storageClass: nfs-client
#  accessMode: ReadWriteOnce
#  size: 8Gi
```

(The commented out lines are there in case you want to experiment with using a storageClass persistent storage back end.)

5. Save the file and close the text editor

Task 2: Deploy the Helm Chart

1. In a terminal, enter the following command to view the current Helm releases:

   ```
   helm list
   ```

   You should not see the dokuwiki application listed.

2. Enter the following command to deploy the chart:

   ```
   helm install mywiki -f ~/dokuwiki-values.yaml bitnami/dokuwiki
   ```

   You should see that the chart was deployed.

3. Enter the following command to view the status of the **mywiki** release:

   ```
   helm status mywiki
   ```

   You should see output similar to what was displayed when the chart was first deployed.

4. You can also enter the following **kubectl** commands:

   ```
   kubectl get deployments
   kubectl get pods
   kubectl get services
   ```

   Notice that you see that same info about the deployed deployments/pods/services as if you were to have deployed them from manifests.

Task 3: Access the Application Deployed by the Helm Chart

1. Enter the following command to list the **mywiki** service:

   ```
   kubectl get services | grep mywiki
   ```

   You should see the **mywiki-dokuwiki** service listed.

Notice the IP and port(s) the application is listening on. The NodePort(s) that the application is listening on are the number after the colon (**:**) in the **PORT(S)** column.

Example: **80:32313/TCP,443:31034/TCP**

In this example the NodePorts are **32313** for http and **31034** for https.

Record the http NodePort:

DOKUWIKI_PORT=_____

2. Open a web browser and point to:

   http://worker01.example.com:*DOKUWIKI_PORT*

   You should see the My Wiki page displayed.
3. On the top right of the page click: **Log In**
4. Enter the following credentials:

   Username: **user**
   Password: **password123**

   You should be logged in.

   (After logging in you may see a number of warnings of available hotfixes and/or updates. You may ignore these warnings.)

Task 4: Delete a Deployed Helm Chart Release

1. In a terminal, enter the following command to delete the **mywiki** release:

   ```
   helm uninstall mywiki --keep-history
   ```

   You should see that the release was deleted.
2. Enter the following command to list the current Helm releases:

   ```
   helm list
   ```

   You should no longer see the dokuwiki chart named **mywiki** displayed.
3. Enter the following command to display the status of the **mywiki** release:

   ```
   helm status mywiki
   ```

   Notice that the STATUS is **UNINSTALLED** but also that it remembered that it had been deployed with that date listed in LAST DEPLOYED.

4. Enter the following command to remove the **mywiki** release from the history:

   ```
   helm uninstall mywiki
   ```

   The release should be uninstalled and its history should be deleted.

5. Enter the following command to display the status of the **mywiki** release:

   ```
   helm status mywiki
   ```

   You should no longer see the history of the release.

Summary:

In this exercise, you first deployed a Helm chart. You then accessed the application that was deployed. Finally you deleted the release.

(End of Exercise)

6 Ingress Networking with an Ingress Controller in Kubernetes

Description:

This section covers ingress networking in Kubernetes with an ingress controller.

6– 1 Configure Ingress for an Application

Description:

In this exercise you use the Ingress Controller to create the simple websites and then access them through a single address.

Dependencies:

An Ingress controller (Nginx, Traefik, etc) must be deployed in the cluster before performing this exercise.

Task 1: Deploy Websites

1. On the management workstation, in a terminal, enter the following commands to switch to the directory containing the ingress manifests and display the contents of one of the app yaml files:

   ```
   cd ~/course_files/KUB201/labs/manifests/ingress
   ```

   ```
   cat blue.yaml
   ```

   You should see the following:

   ```
   kind: Pod
   apiVersion: v1
   metadata:
     name: blue-app
     labels:
       app: blue-app
   spec:
     containers:
       - name: blue-app
         image: hashicorp/http-echo
         args:
           - "-text=blue"

   ---
   ```

```
kind: Service
apiVersion: v1
metadata:
  name: blue-service
spec:
  selector:
    app: blue-app
  ports:
    - port: 5678 # Default port for image
```

Notice that this manifest defines both an application and a service for the application. In this case the "blue-app" will return the text "blue" when queried. The red and green apps are configured similarly and will return their respective colors.

2. Enter the following commands to deploy the test websites:

```
kubectl apply -f green.yaml

kubectl apply -f blue.yaml

kubectl apply -f red.yaml
```

3. Confirm that the pods are ready:

```
kubectl get pods
```

Each pod should be in a running status.

Task 2: Deploy the Ingress Rules

1. On the management workstation, in a terminal, enter the following command to display the contents of the **color-routes.yaml** file:

```
cat color-routes.yaml
```

You should see the following:

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: color-ingress
  annotations:
    ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - http:
```

```
paths:
  - path: /red
    backend:
      serviceName: red-service
      servicePort: 5678
  - path: /green
    backend:
      serviceName: green-service
      servicePort: 5678
  - path: /blue
    backend:
      serviceName: blue-service
      servicePort: 5678
```

Notice in the rules section that each of the color named apps is given a corresponding path that references the app's service port. This will create a new ingress for the green, blue, and red websites by forwarding any request to the **/red**, **/green**, or **/blue** directories to the service with that name.

2. Enter the following command to deploy the ingress routing for the websites:

   ```
   kubectl apply -f color-routes.yaml
   ```

3. Confirm that the ingress is available:

   ```
   kubectl get ingresses.extensions
   ```

   You should see the **color-ingress** available on port **80**.

4. Enter the following command to review the ingress:

   ```
   kubectl describe ingresses.extensions color-ingress
   ```

   In the rules sections you should see:

   ```
   Host  Path  Backends
   ----  ----  --------
   *
         /red     red-service:5678 (10.42.4.63:5678)
         /green   green-service:5678 (10.42.4.63:5678)
         /blue    blue-service:5678 (10.42.4.63:5678)
   ```

   As in the **color-route.yaml** file, the **/red** directory will be forwarded to the **red-service**, etc.

Task 3: Test the Ingress

1. On the management workstation, in a terminal, enter the following command

to test the ingress using the curl command:

```
curl http://worker01.example.com/red
```

```
curl http://worker01.example.com/green
```

```
curl http://worker01.example.com/blue
```

The output from each command should reflect the URL that command is pointing to.

Summary:

In this exercise you tested the ingress on your cluster. You were able to create an ingress route that routes traffic from one URL to 3 different pods via the Kubernetes ingress.

(End of Exercise)

7 Storage in Kubernetes

Description:

This section covers persistent storage in Kubernetes.

7– 1 Configure Persistent Storage with NFS

Description:

In this exercise, you configure a persistent volume on an NFS server. You then create a pod that updates a file on the persistent volume and a pod that exports the file via http.

Task 1: Create the Persistent Volume on the NFS Server

1. In the management workstation, in a terminal, enter the following commands to create the directories to use as the persistent volumes:

```
sudo mkdir -p /srv/nfs/vol-01

sudo chmod 777 /srv/nfs/vol-01

sudo mkdir -p /srv/nfs/vol-02

sudo chmod 777 /srv/nfs/vol-02
```

The **/srv/nfs/vol-01** and **/srv/nfs/vol-02** directories should now exist. The **/etc/exports** file was already pre-configured to export the **/srv/nfs** directory.

Task 2: Examine the Manifests for the Persistent Volumes

1. Enter the following commands to change to the directory containing the nfs-pv manifests and display the contents of the **nfs-pv-vol-01.yaml** file:

```
cd ~/course_files/KUB201/labs/manifests/nfs-pv/

cat nfs-pv-vol-01.yaml
```

You should see the following:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: nfs-vol-01
  labels:
    volname: "vol-01"
spec:
  capacity:
    storage: 1Mi
  accessModes:
    - ReadWriteMany
  persistentVolumeReclaimPolicy: Recycle
  nfs:
    server: 172.30.201.2
    path: "/srv/nfs/vol-01"
```

Note label assigned to the volume and how the NFS shares are defined.

2. Enter the following command to display the contents of the **nfs-pv-vol-02.yaml** file:

```
cat nfs-pv-vol-02.yaml
```

You should see the following:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: nfs-vol-02
  labels:
    volname: "vol-02"
spec:
  capacity:
    storage: 1Mi
  accessModes:
    - ReadWriteMany
  persistentVolumeReclaimPolicy: Recycle
  nfs:
    server: 172.30.201.2
    path: "/srv/nfs/vol-02"
```

Note label assigned to the volume and how the NFS shares are defined.

Task 3: Examine the Manifest for the Persistent Volume Claim

1. Enter the following command to display the contents of **nfs-pvc.yaml** file:

```
cat nfs-pvc.yaml
```

You should see the following:

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: nfs-claim
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 1Mi
  storageClassName: ""
  selector:
    matchLabels:
      volname: "vol-02"
```

Notice how the Persistent Volume Claim doesn't define the storage volume, only the Persistent Volume by name. Also notice that the storageClassName is empty and notice which label is specified in the selector section.

Task 4: Examine the Manifest for the Busybox Instance

1. Enter the following command to display the contents of **nfs-busybox-deployment.yaml** file:

```
cat nfs-busybox-deployment.yaml
```

You should see the following:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nfs-busybox
spec:
  selector:
    matchLabels:
      app: nfs-busybox
  replicas: 1
  template:
    metadata:
      labels:
        app: nfs-busybox
    spec:
```

```
      containers:
      - image: busybox
        command:
          - sh
          - -c
          - 'while true; do date > /mnt/index.html; hostname >> /
mnt/index.html; sleep $(($RANDOM % 5 + 5)); done'
        imagePullPolicy: IfNotPresent
        name: busybox
        volumeMounts:
          # name must match the volume name below
          - name: nfs-vol
            mountPath: "/mnt"
      volumes:
      - name: nfs-vol
        persistentVolumeClaim:
          claimName: nfs-claim
```

Notice how the manifest is describing which mounts and which Persistent Volume Claim is being utilized to use the NFS storage.

Task 5: Examine the Manifest for the Webserver Instance

1. Enter the following command to display the contents of **nfs-web-deployment.yaml** file:

   ```
   cat nfs-web-deployment.yaml
   ```

   You should see the following:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nfs-web
spec:
  selector:
   matchLabels:
     app: nfs-web
  replicas: 1
  template:
    metadata:
      labels:
        app: nfs-web
    spec:
      containers:
      - name: web
```

```
      image: nginx:1.9.0
      ports:
        - name: web
          containerPort: 80
      volumeMounts:
          - name: nfs-vol
            mountPath: "/usr/share/nginx/html"
    volumes:
    - name: nfs-vol
      persistentVolumeClaim:
        claimName: nfs-claim
```

Notice how the manifest is describing which mounts and which Persistent Volume Claim is being utilized to use the NFS storage.

Task 6: Examine the Manifest for the Web Service

1. Enter the following command to display the contents of **nfs-web-service.yaml** file:

   ```
   cat nfs-web-service.yaml
   ```

   You should see the following:

   ```
   apiVersion: v1
   kind: Service
   metadata:
     name: nfs-web
   spec:
     type: NodePort
     ports:
       - port: 80
         nodePort: 30080
     selector:
       app: nfs-web
   ```

   Note the nodePort used to access the web server (30080).

Task 7: Deploy the Objects

1. To deploy the volumes/pods/service, open a terminal and enter the following command:

   ```
   kubectl apply -f ./
   ```

   You should see the following were created (not necessarily in this order):

```
deployment.apps/nfs-busybox created
persistenvolume/nfs-vol-01 created
persistenvolume/nfs-vol-02 created
persistenvolumeclaim/nfs-claim created
deployment.apps/nfs-web created
service/nfs-web created
```

2. Enter the following command to view the deployments:

   ```
   kubectl get deployments
   ```

   You should see the nfs-busybox and nfs-web deployments listed.

3. Enter the following command to view the pods:

   ```
   watch kubectl get pods
   ```

   You should see the pods for the nfs-busybox and nfs-web deployments listed.

4. Enter the following command to view the persistent volumes:

   ```
   kubectl get pv
   ```

   You should see the persistent volumes nfs-vol-01 and nfs-vol-02 listed. Notice that for **nfs-vol-01**:

   Its RECLAIM POLICY is **Recycle**,
   Its STATUS is **Available**
   The CLAIM it is bound to is empty

   Notice that for **nfs-vol-02:**

   Its RECLAIM POLICY is **Recycle**,
   its STATUS is **Bound**
   The CLAIM it is bound to is **default/nfs-sc-claim**

5. Enter the following command to view the persistent volume claims:

   ```
   kubectl get pvc
   ```

   You should see the persistent volume claim **nfs-claim** listed.
   Notice that its STATUS is **Bound** and the VOLUME it is bound to is **nfs-vol-02**.

Task 8: Test the Persistent Data

1. On the management workstation, open a web browser and point to:

   http://worker01.example.com:30080

You will see the content of the **index.html** file. When you refresh the page you should see the time stamp updating. (Also notice that if you change the URL to the different worker nodes you will see the same thing.)

Task 9: Remove the Objects from the Cluster

1. In the first terminal, enter the following command to delete all of the objects:

   ```
   kubectl delete -f ./
   ```

   You should see that the objects were deleted.

Summary:

In this exercise, you examined manifests for two persistent volumes, a persistent volume claim, a pod to that attached to the volume and writes data to an **index.html** file in the volume, and a web server that attaches to the volume and displays the **index.html** file. You then discovered which volume the pods were attached to and verified that the **index.html** file was being updated by looking at the file both on the NFS volume and the web server. Finally you removed the deployments.

(End of Exercise)

7– 2 Configure Persistent Storage with an NFS StorageClass

Description:

In this exercise, you configure a persistent volume claim against an NFS storageClass. You then create a pod that updates a file on the persistent volume and a pod that exports the file via http.

Task 1: Deploy the NFS storageClass

1. On the management workstation, enter the following commands to add the helm repository that contains the storageClass provisioner

```
helm repo add nfs-subdir-external-provisioner \
  https://kubernetes-sigs.github.io/nfs-subdir-external-provisioner/

helm repo update
```

   You should see that the repo was added and all helm repos were updated.

2. Enter the following command to deploy the storageClass provisioner:

```
helm install nfs-subdir-external-provisioner \
  nfs-subdir-external-provisioner/nfs-subdir-external-provisioner \
  --set nfs.server=172.30.201.2 \
  --set nfs.path=/srv/nfs
```

   The storageClass provisioner should be deployed.

3. Enter the following commands to verify the deployment:

```
kubectl get deployments
```

   You should see the **nfs-subdir-external-provisioner** deployment listed.

```
kubectl get storageclasses
```

   You should see the **nfs-client** storageClass listed.

Task 2: Examine the Manifest for the Persistent Volume

4. On the management workstation, enter the following commands to change to the directory containing the **nfs-pv-storage-class** manifests and display the contents of the **nfs-sc-pvc.yaml** file:

   ```
   cd ~/course_files/KUB201/labs/manifests/nfs-pv-storage-class/

   cat nfs-sc-pvc.yaml
   ```

   You should see the following:

   ```
   kind: PersistentVolumeClaim
   apiVersion: v1
   metadata:
     name: nfs-sc-claim
   spec:
     accessModes:
       - ReadWriteMany
     storageClassName: "nfs-client"
     resources:
       requests:
         storage: 1Mi
   ```

   Notice a value is specified in the **storageClassName**.

Task 3: Examine the Manifest for the Busybox Instance

1. Enter the following command to display the contents of **nfs-sc-busybox-deployment.yaml** file:

   ```
   cat nfs-sc-busybox-deployment.yaml
   ```

   You should see the following:

   ```
   apiVersion: apps/v1
   kind: Deployment
   metadata:
     name: nfs-sc-busybox
   spec:
     selector:
       matchLabels:
         app: nfs-sc-busybox
     replicas: 1
   ```

```
  template:
    metadata:
      labels:
        app: nfs-sc-busybox
    spec:
      containers:
      - image: busybox
        command:
          - sh
          - -c
          - 'while true; do date > /mnt/index.html; hostname >> /
mnt/index.html; echo "[storageClass]" >> /mnt/index.html; sleep $
(($RANDOM % 5 + 5)); done'
        imagePullPolicy: IfNotPresent
        name: busybox
        volumeMounts:
          # name must match the volume name below
          - name: nfs-sc-vol
            mountPath: "/mnt"
      volumes:
      - name: nfs-sc-vol
        persistentVolumeClaim:
          claimName: nfs-sc-claim
```

Notice how the manifest is describing which mounts and which Persistent Volume Claim is being utilized to use the NFS storage.

Task 4: Examine the Manifest for the Webserver Instance

1. Enter the following command to display the contents of **nfs-sc-web-deployment.yaml** file:

   ```
   cat nfs-sc-web-deployment.yaml
   ```

   You should see the following:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nfs-sc-web
spec:
  selector:
   matchLabels:
     app: nfs-sc-web
  replicas: 1
  template:
```

```
  metadata:
    labels:
      app: nfs-sc-web
  spec:
    containers:
    - name: web
      image: nginx:1.9.0
      ports:
        - name: web
          containerPort: 80
      volumeMounts:
          - name: nfs-sc-vol
            mountPath: "/usr/share/nginx/html"
    volumes:
    - name: nfs-sc-vol
      persistentVolumeClaim:
        claimName: nfs-sc-claim
```

Notice how the manifest is describing which mounts and which Persistent Volume Claim is being utilized to use the NFS storage.

Task 5: Examine the Manifest for the Web Service

1. Enter the following command to display the contents of **nfs-sc-web-service.yaml** file:

   ```
   cat nfs-sc-web-service.yaml
   ```

   You should see the following:

   ```
   apiVersion: v1
   kind: Service
   metadata:
     name: nfs-sc-web
   spec:
     type: NodePort
     ports:
       - port: 80
         nodePort: 30180
     selector:
       app: nfs-sc-web
   ```

   Note the **nodePort** used to access the web server (30180).

Task 6: Deploy the Objects

1. To deploy the volumes/pods/service, open a terminal and enter the following command:

   ```
   kubectl apply -f ./
   ```

   You should see the following were created (not necessarily in this order):

   ```
   deployment.apps/nfs-sc-busybox created
   persistenvolumeclaim/nfs-sc-claim created
   deployment.apps/nfs-sc-web created
   service/nfs-sc-web created
   ```

2. Enter the following command to view the deployments:

   ```
   kubectl get deployments
   ```

   You should see the nfs-sc-busybox and nfs-sc-web deployments listed.

3. Enter the following command to view the pods:

   ```
   watch kubectl get pods
   ```

   You should see the pods for the nfs-sc-busybox and nfs-sc-web deployments listed.

4. Enter the following command to view the persistent volume claims:

   ```
   kubectl get pv
   ```

   You should see the persistent volume named pvc-<random_string> listed.

   Notice that:

   Its RECLAIM POLICY is **Delete**
   Its STATUS is **Bound**
   The CLAIM it is bound to is **default/nfs-sc-claim**

5. Enter the following command to view the persistent volume claims:

   ```
   kubectl get pvc
   ```

   You should see the persistent volume claim nfs-sc-claim listed.

   Notice that its STATUS is **Bound** and the VOLUME it is bound to is the one in the previous step.

Task 7: Examine the Persistent Storage

1. On the management workstation, enter the following command to display the contents of the NFS exported directory:

   ```
   ls -l /srv/nfs/
   ```

   You should see a new directory named:

   default-nfs-sc-claim-pvc-<random_numbers_and_letters>

   This is the volume that was created by the NFS storageClass.

2. Enter the following command to to view the contents of the **index.html** file in that directory:

   ```
   cat /srv/nfs/default-nfs-sc-claim*/index.html
   ```

   You should see the data that was written by the busybox container.

Task 8: Test the Persistent Data

1. On the management workstation, open a web browser and point to:

   http://worker01.example.com:30180

   You will see the content of the **index.html** file. When you refresh the page you should see the time stamp updating. (Also notice that if you change the URL to the different worker nodes you will see the same thing.)

Task 9: Remove the Objects from the Cluster

1. In the terminal, enter the following commands to delete all of the objects:

   ```
   kubectl delete -f ./
   ```

   You should see that the objects were deleted.

Task 10: Reexamine the Persistent Storage

1. Enter the following command to display the contents of the NFS exported directory:

   ```
   ls -l /srv/nfs/
   ```

You should see a new directory named:

```
archived-default-nfs-sc-claim-pvc-<random_numbers_and_letters>
```

Notice that the persistent volume directory that was created has now been renamed by the NFS storageClass. This behavior is configurable as you can edit a value in the storage class's deployment that tells it to delete the persistent volume directory rather then rename (archive) it.

2. Enter the following command to remove the directory:

```
sudo rm -rf /srv/nfs/archived-default-nfs-sc-claim*
```

The directory should be gone.

Summary:

In this exercise, you deployed the storageClass provisioner, created manifests for a persistent volume claim using a storageClass, a pod to that attached to the volume and writes data to an **index.html** file in the volume, and a web server that attaches to the volume and displays the **index.html** file. You then verified that the **index.html** file was being updated by looking at the file both on the NFS volume and the web server. Finally you removed the deployments and the directory on the NFS server that corresponded to the persistent volume.

(End of Exercise)

8 Resource Usage Control in Kubernetes

Description:

This section covers resource usage control in Kubernetes using Limits, Requests and Quotas.

8– 1 Define Default Limits for Pods in a Namespace

Description:

In this exercise, you define limits for containers and pods in the Kubernetes cluster and test those limits.

Task 1: Create a New Namespace in the Cluster

1. On the management workstation, at the command line, enter the following command to create a new namespace in the Kubernetes cluster:

   ```
   kubectl create namespace limit-example
   ```

   You should see that a new namespace was created.
2. Enter the following command to display the namespaces:

   ```
   kubectl get namespaces
   ```

   You should see the new namespace listed.

Task 2: Examine the Manifest that Defines the Limits

1. Enter the following commands to change to the directory containing the limits manifests and display the contents of the **default-limits.yaml** file:

   ```
   cd ~/course_files/KUB201/labs/manifests/limits
   ```

   ```
   cat default-limits.yaml
   ```

   You should see the following:

   ```
   apiVersion: v1
   kind: LimitRange
   metadata:
     name: cpu-and-memory-limits
   spec:
     limits:
   ```

```
- default:
    cpu: "1"
    memory: 200Mi
  defaultRequest:
    cpu: 500m
    memory: 100Mi
  max:
    cpu: "2"
    memory: 1Gi
  min:
    cpu: 200m
    memory: 3Mi
  type: Container
```

Notice that there is no resources section specifying limits or requests.

Task 3: Apply the Limits to the Namespace

1. To deploy the pod, open a terminal and enter the following commands:

   ```
   cd ~/course_files/KUB201/labs/manifests/limits/
   ```

   ```
   kubectl -n limit-example apply -f default-limits.yaml
   ```

   You should see the limitrange cpu-and-memory-limits was created.

2. Enter the following command to display the limitranges:

   ```
   kubectl -n limit-example describe limitranges
   ```

   You should see the cpu-and-memory-limits limitrange listed. Compare this with the limits set in the manifest file in Task 2.

3. Another way to review the limits of a namespace is to describe the namespace:

   ```
   kubectl describe namespace limit-example
   ```

Summary:

In this exercise, you created a new namespace in the Kubernetes cluster. You then defined default limits for pods and applied them to the new namespace.

(End of Exercise)

8– 2 Define Limits for Containers and Pods

Description:

In this exercise, you test pod resource limits and requests in a namespace.

Dependencies:

The "Define Default Limits for Pods in a Namespace" exercise must be completed before performing this exercise.

Task 1: Deploy a Pod with No Limits or Requests

1. On the management workstation, in a terminal, enter the following commands to change to the directory containing the limits manifests and display the contents of the **cpu-defaults-pod.yaml** file:

   ```
   cd ~/course_files/KUB201/labs/manifests/limits

   cat cpu-defaults-pod.yaml
   ```

   You should see the following:

   ```
   apiVersion: v1
   kind: Pod
   metadata:
     name: cpu-defaults-pod
   spec:
     containers:
     - name: cpu-defaults-pod
       image: nginx
   ```

   Notice that there is no resources section specifying limits or requests.

2. Enter the following command to attempt to deploy the pod:

   ```
   kubectl -n limit-example apply -f cpu-defaults-pod.yaml
   ```

   You should see the pod was deployed.

3. Enter the following command to display the pod's specification:

   ```
   kubectl -n limit-example get pod cpu-defaults-pod --output=yaml
   ```

   You should see a resource section with the default limits and requests that are set for the namespace (**limits.cpu: 1** and **requests.cpu: 500m**).

Task 2: Deploy a Pod with Only a Limit

1. Enter the following command to display the contents of the **cpu-limit-only-pod.yaml** file:

   ```
   cat cpu-limit-only-pod.yaml
   ```

   You should see the following:

   ```
   apiVersion: v1
   kind: Pod
   metadata:
     name: cpu-limit-only-pod
   spec:
     containers:
     - name: cpu-limit-only-pod
       image: nginx
       resources:
         limits:
           cpu: "1"
   ```

   Notice that in the resources section only a **cpu** limit is specified.

2. Enter the following command to attempt to deploy the pod:

   ```
   kubectl -n limit-example apply -f cpu-limit-only-pod.yaml
   ```

   You should see the pod was deployed.

3. Enter the following command to display the pod's specification:

   ```
   kubectl -n limit-example get pod \
     cpu-limit-only-pod --output=yaml
   ```

   You should see a resource section with the cpu requests set to the same value as the limit (**limits.cpu: 1** and **requests.cpu: 1**).

Task 3: Deploy a Pod with Only a Request

1. Enter the following command to display the contents of the **cpu-request-only-pod.yaml** file:

   ```
   cat cpu-request-only-pod.yaml
   ```

   You should see the following:

   ```
   apiVersion: v1
   kind: Pod
   metadata:
     name: cpu-request-only-pod
   spec:
     containers:
     - name: cpu-request-only-pod
       image: nginx
       resources:
         requests:
           cpu: "0.75"
   ```

   Notice that in the resources section only a **cpu request** is specified.

2. Enter the following command to attempt to deploy the pod:

   ```
   kubectl -n limit-example apply -f cpu-request-only-pod.yaml
   ```

   You should see the pod was deployed.

3. Enter the following command to display the pod's specification:

   ```
   kubectl -n limit-example get pod cpu-request-only-pod \
     --output=yaml
   ```

   You should see a resource section with the cpu requests set to value specified in the pod's manifest and the limit is set to the namespace default (**limits.cpu: 1** and **requests.cpu: 750m**).

Task 4: Deploy a Pod Requesting Too Much CPU

1. Enter the following command display the contents of the **too-much-cpu-pod.yaml** file:

   ```
   cat too-much-cpu-pod.yaml
   ```

   You should see the following:

   ```
   apiVersion: v1
   ```

```
kind: Pod
metadata:
  name: too-much-cpu-pod
spec:
  containers:
  - name: too-much-cpu-pod
    image: nginx
    resources:
      limits:
        cpu: "3"
        memory: 100Mi
```

Notice that in the resources section the **cpu** limit is set to **3** (which is more than the maximum limit set in the **LimitRange** set on the namespace).

2. Enter the following command to attempt to deploy the pod:

```
kubectl -n limit-example apply -f too-much-cpu-pod.yaml
```

You should see the pod "too-much-cpu-pod" was not created because it tried to exceed the maximum **CPU** limit of **2**.

```
Error from server (Forbidden): error when creating "too-much-cpu-
pid.yaml": pods "too-much-cpu-pod" is forbidden: [maximum cpu
usage per Container is 2, but limit is 3]
```

Task 5: Deploy a Pod Requesting Too Little CPU

1. Enter the following command display the contents of the **too-little-cpu-pod.yaml** file:

```
cat too-little-cpu-pod.yaml
```

You should see the following:

Notice that in the resources section the **cpu** limit is set to **100m** (which is less than the minimum limit set in the **LimitRange** set on the namespace).

2. Enter the following command to attempt to deploy the pod:

```
kubectl -n limit-example apply -f too-little-cpu-pod.yaml
```

You should see the pod "too-little-cpu-pod" was not created because it tried to exceed the minimum **CPU** limit of **200m**.

```
Error from server (Forbidden): error when creating "too-little-
cpu-pid.yaml": pods "too-litle-cpu-pod" is forbidden: [minimum
cpu usage per Container is 200m, but limit is 100m]
```

Task 6: Delete the Namespace and Objects from the Cluster

1. Enter the following command to delete a namespace from the Kubernetes cluster:

   ```
   kubectl delete namespace limit-example
   ```

   You should see that the namespace was deleted.

2. Enter the following command to display the namespaces:

   ```
   kubectl get namespaces
   ```

   You should no longer see the namespace listed.

Summary:

In this exercise, you tested pod resource limits and requests in a namespace and then deleted the namespace.

(End of Exercise)

8– 3 Define Quotas for a Namespace

Description:

In this exercise you create a new namespace and then create quotas for that namespace.

Task 1: Create a New Namespace in the Cluster

1. Enter the following command to create a new namespace:

   ```
   kubectl create namespace quota-example
   ```

   The namespace should have been created.

2. Enter the following command to display the quota usage for the new namespace:

   ```
   kubectl -n quota-example describe quota
   ```

   No quotas should be displayed as none have been defined for the namespace yet.

Task 2: Examine the Manifests that Define the Quotas

1. On the management workstation, in a terminal, enter the following commands to change to the directory containing the quota manifests and display the contents of the **quota-high.yaml** file

   ```
   cd ~/course_files/KUB201/labs/manifests/quotas
   ```

   ```
   cat quota-high.yaml
   ```

   You should see the following:

   ```
   apiVersion: v1
   kind: ResourceQuota
   metadata:
   ```

```
  name: pods-high
spec:
  hard:
    cpu: “10”
    memory: 20Gi
    pods: 10
```

Note the quotas for cpu, memory and pods.

2. Enter the following command to display the contents of the **quota-low.yaml** file:

   ```
   cat quota-low.yaml
   ```

   You should see the following:

   ```
   apiVersion: v1
   kind: ResourceQuota
   metadata:
     name: pods-low
   spec:
     hard:
       cpu: “3”
       memory: 3Gi
       pods: 5
   ```

   Note the quotas for cpu, memory and pods. Compare them to the values for the **quotas-high.yaml** file.

Task 3: Set Quotas for a Namespace

1. Enter the following command to create the higher set of quotas on the namespace:

   ```
   kubectl -n quota-example create -f quota-high.yaml
   ```

   The quotas should have been created.

2. Enter the following command to display the quota usage for the new namespace:

   ```
   kubectl -n quota-example describe quota
   ```

   Notice that the quotas match those in the **quota-high.yaml** file.

Summary:

In this exercise you first created a new namespace and then verified that there are no quotas set in the namespcae. You then created quotas in the namespace and verified they were created.

(End of Exercise)

8– 4 Test Quotas for a Namespace

Description:

In this exercise you deploy an application in a namespace that has quotas set. You then scale the application out and back to see the effects the quotas have.

Dependencies:

The "**Define Quotas for a Namespace**" exercise must be completed before performing this exercise.

Task 1: Deploy a Pod in a Namespace that Contains Quotas

1. On the management workstation, in a terminal, enter the following commands to change to the directory containing the quota manifests and display the contents of the **opensuse-deployment.yaml** file:

```
cd ~/course_files/KUB201/labs/manifests/quotas

cat opensuse-deployment.yaml
```

You should see the following:

```
apiVersion: apps/v1
kind: Deployment metadata:
  name: opensuse-depoyment
  labels:
    env: "app"
    owner: opensuse
spec:
  selector:
    matchLabels:
      app: opensuse
  replicas: 1
  template:
```

```
metadata:
  labels:
    app: opensuse
spec:
  containers:
  - name: opensuse
    image: opensuse/leap
    command: ["/bin/sh"]
    args: ["-c","while true; do echo hello; sleep 10; done"]
    resources"
      requests:
        cpu: "1.5"
        memory: "1Gi"
      limits:
        cpu: "2"
        memory: "1Gi"
```

Note the resource requests and limits.

2. Enter the following command to deploy the pod:

```
kubectl -n quota-example apply -f opensuse-deployment.yaml
```

The pod should be deployed.

3. Enter the following commands to verify that pod was deployed:

```
kubectl -n quota-example get deployments

kubectl -n quota-example get pods
```

You should see that there is a deployment with a single replica and a single opensuse pod.

Task 2: Check Quota Usage in a Namespace

1. Enter the following command to display the quota usage in the namespace:

```
kubectl -n quota-example describe quota
```

You should see something like the following:

```
Name:       pods-high
Namespace:  quota-example
Resource    Used   Hard
--------    ----   ----
cpu         1500m  10
memory      1Gi    20Gi
```

```
pods            1       10
```

Notice the quotas that are set. Also notice the values in the **Used** column for each quota.

Task 3: Scale Out a Deployment

1. Enter the following command to scale the deployment out:

```
kubectl -n quota-example \
  scale deployment opensuse-deployment \
  --replicas=5
```

The deployment should have been scaled.

2. Enter the following command to display info about the deployment:

```
kubectl -n quota-example get deployment
```

You should see that there are 5 replicas running.

3. Enter the following command to display the running pods:

```
kubectl -n quota-example get pods
```

You should see that there are indeed 5 pods running that correspond to the 5 replicas in the deployment.

4. Enter the following command to display the quota usage in the namespace:

```
kubectl -n quota-example describe quota
```

You should see something like the following:

```
Name:       pods-high
Namespace:  quota-example
Resource    Used   Hard
--------    ----   ----
cpu         7500m  10
memory      5Gi    20Gi
pods        5      10
```

Notice the values in the **Used** column for each quota have increased but are below the hard quota.

5. Enter the following command to scale the deployment again:

```
kubectl -n quota-example \
  scale deployment opensuse-deployment \
  --replicas=10
```

The deployment should have been scaled.

6. Enter the following command to display info about the deployment:

```
kubectl -n quota-example get deployment
```

You should see that there are only 6 of the requested 10 replicas running.

7. Enter the following command to display the running pods:

```
kubectl -n quota-example get pods
```

You should see that there are indeed 6 pods running that correspond to the 6 of 10 replicas in the deployment.

8. Enter the following command to display the quota usage in the namespace:

```
kubectl -n quota-example describe quota
```

You should see something like the following:

```
Name:       pods-high
Namespace:  quota-example
Resource    Used   Hard
--------    ----   ----
cpu         9      10
memory      6Gi    20Gi
pods        6      10
```

Notice the values in the **Used** column for each quota have increased but are below the hard quota.

Note that the deployment was not able to scale out to the requested 10 replicas because it would have exceeded the cpu quota set in the namespace.

Task 4: Change Quotas for a Namespace

1. Enter the following commands to create the lower set of quotas on the namespace:

```
kubectl -n quota-example delete -f quota-high.yaml

kubectl -n quota-example create -f quota-low.yaml
```

The old quotas should have been deleted and the new quotas should have been created.

2. Enter the following command to display the quota usage for the new namespace:

```
kubectl -n quota-example describe quota
```

You should see something like the following:

```
Name:       pods-low
Namespace:  quota-example
Resource    Used   Hard
--------    ----   ----
cpu         9      3
memory      6Gi    3Gi
pods        6      5
```

Notice that the quotas in the **Hard** column match those in the **quota-low.yaml** file.

Also notice that the values in the **Used** column exceed the values in the **Hard** column because the pods were already running before the quotas changed.

Task 5: Scale a Deployment Again

1. Enter the following command to scale the deployment back:

```
kubectl -n quota-example \
  scale deployment opensuse-deployment \
  --replicas=5
```

The deployment should have been scaled back.

2. Enter the following command to display info about the deployment:

```
kubectl -n quota-example get deployment
```

You should see that there are 5 of 5 replicas running.

3. Enter the following command to display the running pods:

```
kubectl -n quota-example get pods
```

You should see that there are indeed 5 pods running that correspond to the 5 replicas in the deployment.

4. Enter the following command to display the quota usage in the namespace:

```
kubectl -n quota-example describe quota
```

You should see something like the following:

```
Name:       pods-low
Namespace:  quota-example
Resource    Used   Hard
--------    ----   ----
cpu         9      3
memory      6Gi    3Gi
pods        6      5
```

Notice the values in the **Used** column for each quota have not changed even thought the deployment has been scaled back.

5. Enter the following command to scale the deployment back even farther:

```
kubectl -n quota-example \
  scale deployment opensuse-deployment \
  --replicas=2
```

The deployment should have been scaled back.

6. Enter the following command to display info about the deployment:

```
kubectl -n quota-example get deployment
```

You should see that there are 2 replicas running.

7. Enter the following command to display the running pods:

```
kubectl -n quota-example get pods
```

You should see that there are indeed 2 pods running that correspond to the 2 replicas in the deployment.

(Depending on how quickly you run the command after the scale back there may be some pods still in the Terminating state. Wait for these to finish terminating before moving on to the next step.)

8. Enter the following command to display the quota usage in the namespace:

```
kubectl -n quota-example describe quota
```

You should see something like the following:

```
Name:       pods-low
Namespace:  quota-example
Resource    Used   Hard
--------    ----   ----
cpu         3      3
memory      2Gi    3Gi
pods        2      5
```

Notice the values in the **Used** column for each quota have decreased and are at or below the hard quota.

9. Enter the following command to attempt to scale the deployment out again:

```
kubectl -n quota-example \
  scale deployment opensuse-deployment \
  --replicas=3
```

The deployment should have been scaled back.

10. Enter the following command to display info about the deployment:

```
kubectl -n quota-example get deployment
```

You should see that there are only 2 of the 3 requested replicas running.

11. Enter the following command to display the running pods:

```
kubectl -n quota-example get pods
```

You should see that there are indeed 2 pods running that correspond to the 2 replicas in the deployment.

12. Enter the following command to display the quota usage in the namespace:

```
kubectl -n quota-example describe quota
```

You should see something like the following:

```
Name:       pods-low
Namespace:  quota-example
Resource    Used   Hard
--------    ----   ----
cpu         3      3
memory      2Gi    3Gi
pods        2      3
```

Notice the values in the **Used** column for each quota have decreased and are at or below the hard quota.

Task 6: Delete Quotas for a Namespace

1. Enter the following command to create the lower set of quotas on the namespace:

```
kubectl -n quota-example delete -f quota-low.yaml
```

The quotas should have been deleted.

2. Enter the following command to display the quota usage for the new namespace:

   ```
   kubectl -n quota-example describe quota
   ```

   You should see that there are no longer quotas for the namespace.

3. Enter the following command to display info about the deployment:

   ```
   kubectl -n quota-example get deployment
   ```

   You should see that there are now 3 of the 3 requested replicas running. (Depending on how quickly you ran this command after deleting the quotas you may still see only 2 replicas running. Wait a minute and rerun the command and you should see the additional replica.)

Task 7: Delete the Namespace

1. Enter the following command to delete the namespace:

   ```
   kubectl delete namespace quota-example
   ```

   The namespace and all quotas/deployments/posts in it should should have been deleted.

2. Enter the following command to verify that namespace was deleted:

   ```
   kubectl get namespace
   ```

   You should no longer see the namespace listed.

Summary:

In this exercise you deployed an application in a namespace that has quotas set. You then scaled out the application, first to a number that would adhere to the quotas and then to one that would exceed the quotas and observed the behavior. You then changed the quotas and observed what happened to the application as you tried to scale it back and then out again. You then deleted the quotas and observed what happened to the scaled out application. Finally you deleted the namespace.

(End of Exercise)

9 Role Based Access Controls Security in Kubernetes

Description:

This section covers Role Based Access Control (RBAC) in Kubernetes.

9– 1 Create Service Accounts

Description:

In this exercise you create service accounts for some additional Kubernetes users.

Task 1: Create a New Namespace

1. On the management workstation, in a terminal, enter the following command to create a new namespace for the users:

   ```
   kubectl create namespace walnuts
   ```

   The namespace should have been created.

Task 2: Create Service Account Manifests

1. On the management workstation, in the text editor of your choice, create/open the file **~/charlie-sa.yaml** to be edited
2. Enter the following in the file:

   ```
   kind: ServiceAccount
   apiVersion: v1
   metadata:
     namespace: walnuts
     name: charlie
   ```

3. Save the file
4. Repeat the previous steps to create manifests for the following users:

   lucy
   linus

   Name the manifest files ***USERNAME*-sa.yaml** (where ***USERNAME*** is the user's name) and modify the **name:** key to match the user's name.

Task 3: Create the Service Accounts in the Namespace

1. Enter the following commands to create the service accounts in Kubernetes:

   ```
   kubectl apply -f ~/charlie-sa.yaml

   kubectl applu -f ~/lucy-sa.yaml

   kubectl apply -f ~/linus-sa.yaml
   ```

   You should have three new service accounts created.

2. Enter the following command to display the service accounts in the namespace:

   ```
   kubectl get serviceaccounts -n walnuts
   ```

   You should see the three new service accounts displayed.

Summary:

In this exercise you createD a new namespace. You then created manifests for new service accounts and then added the service accounts to the cluster.

(End of Exercise)

9– 2 Create kubeconfig Files for Service Accounts

Description:

In this exercise you create kubeconfig files for service accounts.

Task 1: Create kubeconfig Files

1. On the management workstation, in a terminal, enter the following commands to retrieve the cluster info for a kubeconfig file:

```
kubectl config view --flatten --minify | \
  grep 'certificate-authority-data:'

kubectl config view --flatten --minify | \
  grep 'server:'

kubectl config view --flatten --minify | \
  grep 'name:' | head -1
```

   The output of the first command will be referred to as ***CA_DATA***.

   The output of the second command will be referred to as ***SERVER***.

   The output of the third command will be referred to as ***CLUSTER_NAME***.

2. Enter the following command to retrieve the secret for the charlie serviceaccount:

```
kubectl describe serviceaccounts -n walnuts charlie \
  | grep Tokens: \
  | awk '{ print $2 }'
```

   We will refer to this as ***USER_SECRET***.

3. Enter the following command to retrieve the authentication token for the charlie user:

```
kubectl describe secrets -n walnuts USER_SECRET \
  | grep token: \
```

```
| awk '{ print $2 }'
```

We will refer to this as ***USER_TOKEN***.

4. In the text editor of your choice, create/open the file **~/kubeconf-charlie** to be edited
5. Enter the following into the file (replacing the CA_DATA, SERVER, CLUSTER_NAME, USER_TOKEN lab variables with the values retrieved above):

```
apiVersion: v1
kind: Config
clusters:
- cluster:
    certificate-authority-data: CA_DATA
    server: SERVER
  name: CLUSTER_NAME
contexts:
- context:
    cluster: CLUSTER_NAME
    user: charlie
  name: CLUSTER_NAME
current-context: CLUSTER_NAME
users:
- name: charlie
  user:
    token: USER_TOKEN
```

6. Save the file
7. Repeat steps 2-5 in this task for the following serviceaccounts (replacing 'charlie' with these serviceaccount names):

lucy
linus

Summary:

In this exercise you created the cluster and serviceaccount authentication information and used that to create kubeconfig files for the serviceaccounts.

(End of Exercise)

9– 3 Create Roles and ClusterRoles

Description:

In this exercise you will create roles in a namespace and cluster roles all from manifests via the command line. You will then confirm that they are available.

Dependencies:

The **walnuts** namespace should have been created before performing this exercise.

Task 1: Create Roles with from Manifests

1. On the management workstation, in a terminal, enter the following commands to review a role manifest:

   ```
   cd ~/course_files/KUB201/labs/manifests/rbac/roles/
   ```

   ```
   cat role-walnuts-pod-watcher.yaml
   ```

   You should see the following text:

   ```
   apiVersion: rbac.authorization.k8s.io/v1
   kind: Role
   metadata:
     namespace: walnuts
     name: walnuts-pod-watcher
   rules:
   - apiGroups: [""]
     resources: ["pods"]
     verbs: ["get", "watch", "list"]
   ```

   Pay attention to the resource specified in the **resources:** section and the verbs in the **verbs:** section as these will indicate what resource and which actions are allowed.

   If desired, repeat the **cat** command for the remaining yaml files in this directory paying attention to the resources and verbs in those files.

2. Enter the following command to create the roles defined by the files in this directory:

   ```
   kubectl apply -f ./
   ```

3. Enter the following command to display the roles created in the walnuts namespace:

   ```
   kubectl -n walnuts get roles
   ```

   You should see roles that correspond with the roles defined in the yaml files.

4. Confirm that a role is now set:

   ```
   kubectl -n walnuts describe role walnuts-pod-watcher
   ```

   You should see the role and the permissions that it contains.

   You can repeat this command for each of the other roles created if desired.

Task 2: Create Cluster Roles from Manifests

1. On the management workstation, in a terminal, enter the following commands to review a cluster role manifest:

   ```
   cd ~/course_files/KUB201/labs/manifests/rbac/clusterroles/
   ```

   ```
   cat clusterrole-pod-watcher.yaml
   ```

   You should see the following text:

   ```
   apiVersion: rbac.authorization.k8s.io/v1
   kind: ClusterRole
   metadata:
     name: cluster-pod-watcher
   rules:
   - apiGroups: [""]
     resources: ["pods"]
     verbs: ["get", "watch", "list"]
   ```

   As you can see, there is no namespace listed for a cluster role.

   If desired, repeat the `cat` command for the remaining yaml files in this directory paying attention to the resources and verbs in those files.

2. Enter the following command to create the cluster roles defined by the files in this directory:

   ```
   kubectl apply -f ./
   ```

3. Enter the following command to display the cluster roles created in the cluster:

   ```
   kubectl get clusterroles
   ```

   You should see cluster roles that correspond with the roles defined in the yaml files.

4. Confirm that the cluster role is now set:

   ```
   kubectl describe clusterrole cluster-pod-watcher
   ```

Summary:

In this exercise you created new roles and cluster roles for users that could be in your cluster. You then confirmed that each was available and what permissions they contain.

(End of Exercise)

9– 4 Create RoleBindings and ClusterRoleBindings

Description:

In this exercise you will create role bindings and a cluster role bindings that bind users to a set of permissions contained in roles and cluster roles.

Dependencies:

The "Log Into a CaaS Platform Cluster" exercise must be completed before performing this exercise.

Task 1: Create Role Bindings from Manifests

1. On the management workstation, in a terminal, enter the following commands to review a role binding:

```
cd ~/course_files/KUB201/labs/manifests/rbac/rolebindings/

cat rolebinding-charlie-walnuts-deployment-manager.yaml
```

You should see the following text:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: charlie-walnuts-deployment-manager
  namespace: walnuts
subjects:
- kind: User
  name: charlie
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: walnuts-deployment-manager
  apiGroup: rbac.authorization.k8s.io
```

The subject is the user and the roleRef is the role that will be bound together in the walnuts namespace.

If desired repeat the **cat** command for the remaining files in this directory taking note of which users are being linked to which roles.

2. Enter the following command to create the role bindings from the file in this directory:

   ```
   kubectl apply -f ./
   ```

   The roles should have been created.

3. Enter the following command to list the role bindings created in the namespace:

   ```
   kubectl -n walnuts get rolebindings
   ```

   You should see the role bindings listed that correspond to the role bindings listed in the yaml files.

4. Enter the following command to review a role binding:

   ```
   kubectl -n walnuts describe rolebinding \
     charlie-walnuts-deployment-manager
   ```

   The user charlie has been bound to the role walnuts-deployment-manager.

   If desired, repeat this command for the other role bindings that were created.

Task 2: Create Cluster Role Bindings from Manifests

1. On the management workstation, in a terminal, enter the following commands to review a cluster role binding:

   ```
   cd ~/course_files/KUB201/labs/manifests/rbac/clusterrolebindings/

   cat clusterrolebinding-linus-pod-watcher.yaml
   ```

   You should see the following text:

   ```
   apiVersion: rbac.authorization.k8s.io/v1
   kind: RoleBinding
   metadata:
     name: linus-cluster-pod-watcher
     namespace: walnuts
   subjects:
   - kind: User
     name: linus
     apiGroup: rbac.authorization.k8s.io
   roleRef:
     kind: Role
   ```

```
name: cluster-pod-watcher
apiGroup: rbac.authorization.k8s.io
```

The subject is the user and the roleRef is the role that will be bound together in the walnuts namespace.

If desired repeat the **cat** command for the remaining files in this directory taking note of which users are being linked to which cluster roles.

2. Enter the following command to deploy the cluster role binding:

   kubectl apply -f ./

   The cluster roles should have been created.

3. Enter the following command to list the cluster role bindings created in the cluster:

   kubectl get clusterrolebindings

   You should see a bunch of existing cluster role bindings listed as well as the cluster role bindings listed that correspond to the role binding in the yaml files.

4. Enter the following command to review a cluster role binding:

   kubectl describe clusterrolebinding linus-cluster-pod-watcher

   You should see the user bound to the cluster role.

Summary:

In this exercise you bound users with their respective roles using role bindings and cluster role bindings and confirmed that they are active.

(End of Exercise)

9– 5 Test RBAC in Kubernetes

Description:

In this exercise you test RBAC roles and cluster roles that were configured in a previous exercise.

Dependencies:

The exercises titles "create Service Accounts", "Create kubeconfig Files for Service Accounts", "Create Roles and ClusterRoles" and "Create RoleBindings and ClusterRoleBindings" must be completed before performing this exercise.

Task 1: Test the RBAC Roles for the Charlie ServiceAccount

1. On the management workstation, in a terminal, enter the following command to test the charlie serviceaccount's ability to view and manage deployments:

   ```
   kubectl --kubeconfig=/home/tux/kubeconf-charlie \
     -n walnuts get deployments
   ```

   You should see "No resources found in walnuts namespace" displayed.

   This shows that the RBAC role assigned to the charlie serviceaccount allowed it to successfully get the deployments in the namespace. No deployments have been created in the namespace yet so the output of the command is correct.

2. Enter the following commands to create a deployment in the namespace:

   ```
   cd ~/course_files/KUB201/labs/manifests/nginx/
   ```

   ```
   kubectl --kubeconfig=/home/tux/kubeconf-charlie \
     -n walnuts apply -f nginx-deployment.yaml
   ```

   The deployment should be created.

3. Try to get the deployments again:

```
kubectl --kubeconfig=/home/tux/kubeconf-charlie \
  -n walnuts get deployments
```

You should see the state of the nginx-deployment displayed.

4. Enter the following command to display the pods created by the deployment:

```
kubectl --kubeconfig=/home/tux/kubeconf-charlie \
  -n walnuts get pods
```

You should see an error stating that the serviceaccount charlie cannot list the pods for the namespace. This is because the charlie serviceaccount was only assigned role relative to deployment resources but not pod resources in the namespace.

5. Enter the following command to display roles for the namespace:

```
kubectl -n walnuts get rolebindings
```

You should see a role binding named **charlie-walnuts-deployment-manager** listed.

6. Enter the following command to display information about the role binding:

```
kubectl -n walnuts \
  describe rolebinding charlie-walnuts-deployment-manager
```

You should see that the charlie serviceaccount is assigned a role named **walnuts-deployment-manager**.

7. Enter the following command to display information about the role:

```
kubectl -n walnuts describe role walnuts-deployment-manager
```

You should see that the resources (**deployments**) and the verbs (**get list watch create update patch delete**) match up with the permission the charlie serviceaccount was able to do in the namespace.

Task 2: Test the RBAC Rules for the Linus ServiceAccount

1. Enter the following command to test the linus serviceaccount's ability to view and manage deployments:

```
kubectl --kubeconfig=/home/tux/kubeconf-linus \
  -n walnuts get deployments
```

You should see an error stating that the serviceaccount linus cannot list the

pods for the namespace.

2. Enter the following command to display the pods created by the deployment in the namespace:

```
kubectl --kubeconfig=/home/tux/kubeconf-linus \
  -n walnuts get pods
```

You should see the pods for the deployment displayed. This is because the linus serviceaccount was assigned a role that only allows listing of pods but not deployments in the namespace.

Note the name of the first pod listed in the deployment. We will refer to this as: ***NGINX_POD***

3. Enter the following command to attempt to delete a pod created by the deployment:

```
kubectl --kubeconfig=/home/tux/kubeconf-linus \
  -n walnuts delete pod NGINX_POD
```

You should see an error stating the linus serviceaccount does not have the permissions to delete a pod. This is because the role assigned to the linus serviceaccount only allows listing of pods.

4. Enter the following command to display the pods created by the deployment in the entire cluster:

```
kubectl --kubeconfig=/home/tux/kubeconf-linus \
  get pods --all-namespaces
```

You should see the pods in all namespaces displayed. This is because the linus serviceaccount was assigned a cluster role that only allows listing of pods which applies to all namespaces.

5. Enter the following command to display roles for the namespace:

```
kubectl -n walnuts get rolebindings
```

You should see a role binding named **linus-walnuts-pod-watcher** listed.

6. Enter the following command to display information about the role binding:

```
kubectl -n walnuts \
  describe rolebinding linus-walnuts-pod-watcher
```

You should see that the charlie serviceaccount is assigned a role named **walnuts-pod-watcher**.

7. Enter the following command to display information about the role:

   ```
   kubectl -n walnuts describe role walnuts-pod-watcher
   ```

   You should see that the resources (**pods**) and the verbs (**get list watch**) match up with the permission the linus serviceaccount was able to do in the namespace.

8. Enter the following command to display cluster roles for the namespace:

   ```
   kubectl get clusterrolebindings
   ```

   You should see a cluster role binding named **linus-cluster-pod-watcher** listed.

9. Enter the following command to display information about the role binding:

   ```
   kubectl describe clusterrolebinding linus-cluster-pod-watcher
   ```

   You should see that the charlie serviceaccount is assigned a role named **cluster-pod-watcher**.

10. Enter the following command to display information abut the role:

    ```
    kubectl describe clusterrole cluster-pod-watcher
    ```

    You should see that the resources (**pods**) and the verbs (**get list watch**) match up with the permission the linus serviceaccount was able to do in all namespaces.

Task 3: Test the RBAC Rules for the Lucy ServiceAccount

1. Using the commands used in the previous tasks, explore what actions the **lucy** serviceaccount is able to perform (using the lucy serviceaccount's kubeconfig file: **~/kubeconf-lucy**)

   What is lucy able to do relative to pods in the namespace?

   What is lucy able to do relative to deployments in the namespace?

   What is lucy able to do relative to other resources in the namespace?

   What roles and cluster role bindings enable lucy to perform these actions?

Summary:

In this exercise you tested the RBAC roles and cluster **roles** for the charlie and linus servicaccounts. You verified that the permissions in the roles and cluster roles assigned to the serviceaccounts matched up with the actions they were actually able to perform. You then explored in a free form fashion what **actions** the lucy serviceaccount is able to perform.

(End of Exercise)

SUSE

Thank you

For more information, contact SUSE at:

+1 800 796 3700 (U.S./Canada)

+49 (0)911-740 53-0 (Worldwide)

Maxfeldstrasse 5

90409 Nuremberg

www.suse.com