NetApp

NetApp Learning Services

Using Astra Trident with Kubernetes

Exercise Guide

Course ID: STRSW-ILT-UATWK
Catalog Number: STRSW-ILT-UATWK-EG

ATTENTION

The information contained in this course is intended only for training. This course contains information and activities that, while beneficial for the purposes of training in a closed, non-production environment, can result in downtime or other severe consequences in a production environment. This course material is not a technical reference and should not, under any circumstances, be used in production environments. To obtain reference materials, refer to the NetApp product documentation that is located at http://mysupport.netapp.com/.

Table of Contents

Module 0: Welcome

Study Aid Icons

In your exercises, you might see one or more of the following icons.

Warning
If you misconfigure a step marked with this icon, later steps might not work properly. Check the step carefully before you move forward.

Attention
Review this step or comment carefully to save time and avoid errors.

Information
Review information about the topic or procedure.

Exercise 1: Checking the Exercise Equipment

In this exercise, you familiarize yourself with your equipment and verify that licenses are installed.

Objectives

This exercise focuses on enabling you to do the following:

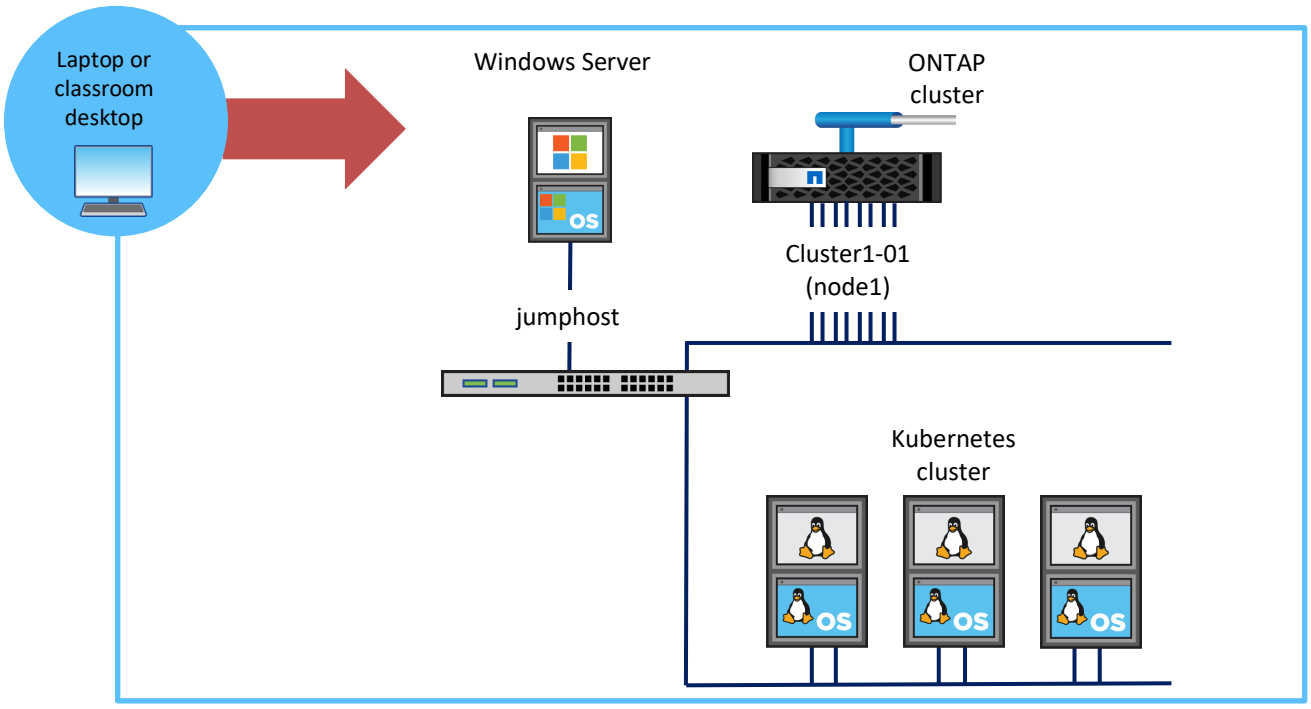
- Connect to your NetApp ONTAP cluster
- Verify that required license codes are configured
- Verify that the required software and tools are installed
- Configure Kubernetes administrator access on the jump host
- Set up your integrated development environment (IDE)
- Work with YAML files in the IDE

Lab Equipment

Your exercise environment contains the following virtual machines:

- One Windows Server system
- A 3-node Kubernetes cluster
- An ONTAP 9.9 single-node cluster (Cluster1)

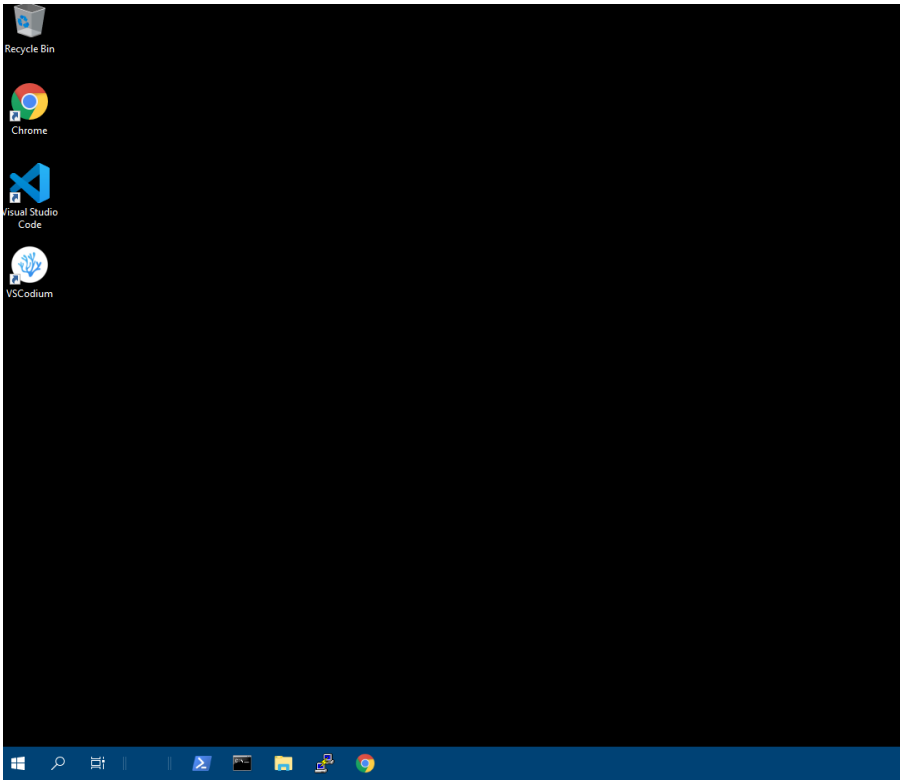
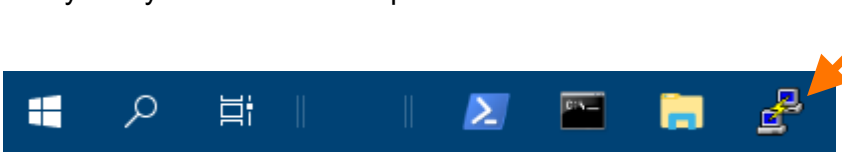
When you use the connection information that your instructor assigns to you, you first connect through Remote Desktop Connection to a Windows Server. From this Windows desktop, you connect to the other servers in your exercise environment.

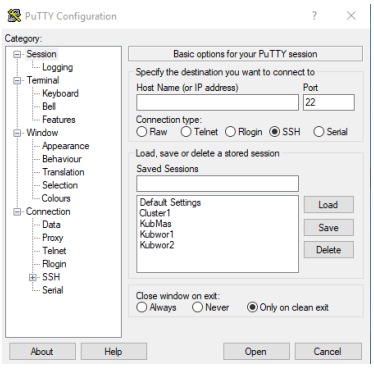


System	Host Name	IP Addresses	User Name	Password
Windows Server	Jumphost	192.168.0.5	DEMO\Administrator	Netapp1!
Kubernetes Control Plane	kubmas	192.168.0.61	root (case sensitive)	Netapp1!
Kubernetes Worker 1	kubwor1	192.168.0.62	root (case sensitive)	Netapp1!
Kubernetes Worker 2	kubwor2	192.168.0.63	root (case sensitive)	Netapp1!
ONTAP cluster-management LIF (Cluster1)	Cluster1	192.168.0.101	admin (case sensitive)	Netapp1!
node1 (Cluster1)	Cluster1-01	192.168.0.111	admin (case sensitive)	Netapp1!

Task 1: Connect to Your NetApp ONTAP Cluster

In this task, you familiarize yourself with the Windows Server desktop. You connect to the ONTAP cluster and verify the health of the cluster.

Step	Action
1-1	Verify that you see the Start page of your assigned Windows Server. 
1-2	Verify that you see the desktop and that the taskbar contains the PuTTY program. 
1-3	To connect to the ONTAP cluster UI, you browse to the NetApp ONTAP System Manager URL, which is built in to the ONTAP software. To connect to the ONTAP cluster CLI, you use PuTTY, which is a UI for the Telnet and Secure Shell (SSH) protocols.
1-4	Double-click the PuTTY shortcut. 

<table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>1-5</td><td>In the PuTTY Configuration dialog box, in the Saved Sessions list, double-click Cluster1.
 
</td></tr>
<tr><td>1-6</td><td>You can also connect to the ONTAP cluster CLI by connecting to a node in the cluster: Cluster-01 (node 1).</td></tr>
<tr><td>1-7</td><td>At the ONTAP cluster login prompt, provide the following credentials:
• Login as: <code>admin</code>
• Password: <code>Netapp1!</code>
The ONTAP cluster CLI prompt and cursor appear.</td></tr>
<tr><td>1-8</td><td>If you have any difficulty logging in to the ONTAP cluster CLI, see this table and verify that you are using the correct user name and password in the correct case (both are case-sensitive).
<table>
<tr><th>System</th><th>Host Name</th><th>IP Address</th><th>User Name</th><th>Password</th></tr>
<tr><td>ONTAP cluster-management LIF</td><td>Cluster1</td><td>192.168.0.101</td><td>admin (case-sensitive)</td><td>Netapp1!</td></tr>
</table></td></tr>
<tr><td>1-9</td><td>Verify that the single node of the ONTAP cluster is healthy and eligible:
<code>cluster show</code></td></tr>
</table>

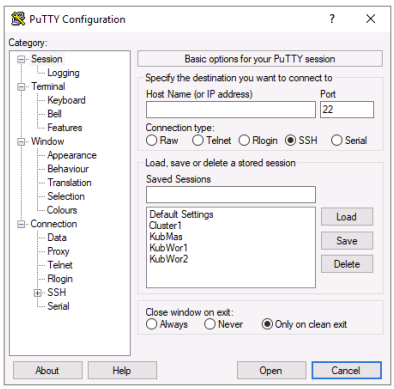
Task 2: Verify That Required License Codes Are Configured

Many advanced features of the ONTAP cluster require licenses to work. In later exercises, you use several licensed features of the ONTAP cluster. In this task, you verify that the necessary licenses are preinstalled.

Step	Action
2-1	In the cluster1 CLI, enter the following command: `license show`
2-2	Verify that the following required license codes are installed: • NFS • iSCSI
2-3	 If any of the licenses are not installed, inform your instructor.

Task 3: Verify That the Required Software and Tools Are Installed

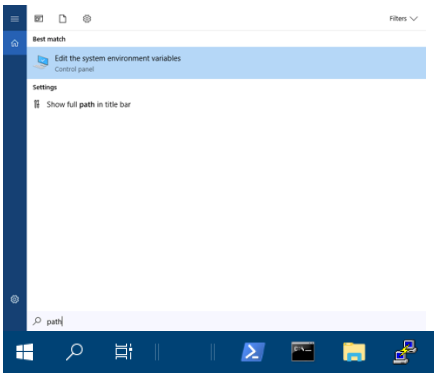
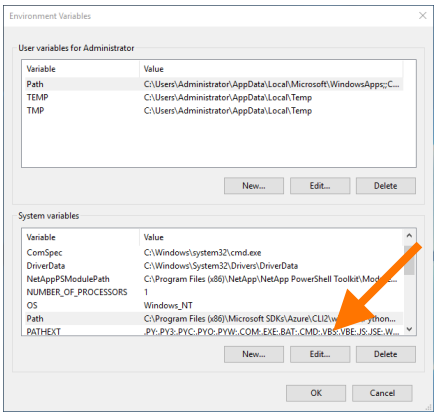
In this task, you verify that Kubernetes and container run time are installed and working properly.

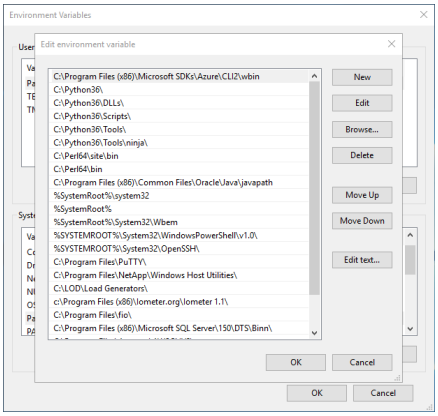
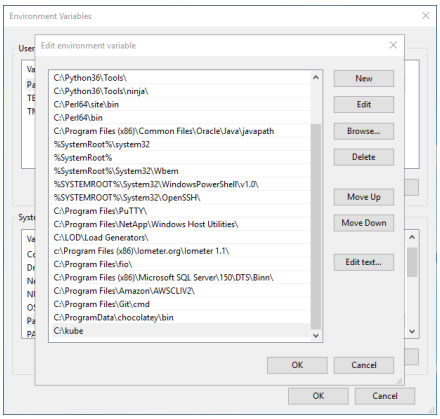
Step	Action
3-1	Close the PuTTY session, and open a new session on KubMas (Kubernetes Control Plane): 
3-2	At the Linux login prompt, provide the following credentials: • Login as: **`root`** • Password: **`Netapp1!`** The Linux CLI prompt and cursor appear.
3-3	Run the following command to verify that the Docker daemon is running: **`ps -aef | grep dockerd`**
3-4	Verify that containers are running: **`docker ps`**
3-5	Verify the connection to the public Docker repositories: **`docker search netapp`**
3-6	Verify that the docker0 interface is configured on the host: **`ifconfig docker0`**
3-7	Verify that the Kubernetes services are running: **`ps -aux | grep kubernetes`**
3-8	If you want, you can repeat the previous seven steps to test kubwor1 (Kubernetes worker 1) and kubwor2 (Kubernetes worker 2).
3-9	Make sure that automatic completion is enabled for kubectl: **`source <(kubectl completion bash)`**
3-10	Optionally, you can create an alias for the most used Kubernetes command: **`alias k='kubectl'`** *Note: If you use the alias, you can replace all the upcoming "kubectl" commands with "k".*

Step	Action
3-11	Verify the version of Kubernetes: **`kubectl version`**
3-12	Review the Kubernetes configuration: **`kubectl config view`**
3-13	If you use multiple kubeconfig files at the same time and you want to merge the views, you can use the following command: **`KUBECONFIG=~/.kube/config:~/.kube/kubconfig2 kubectl config view`**
3-14	Get the Kubernetes version in JSON (`-o json`) or YAML (`-o yaml`) format: **`kubectl version -o json`** **`kubectl version -o yaml`**
3-15	Verify the node type to which you are connected: **`kubectl cluster-info`**
3-16	Verify that the Kubernetes cluster is showing you the three nodes: **`kubectl get nodes`**
3-17	Use the `-o` option to change the output (this option can be used in many commands): **`kubectl get nodes -o wide`** **`kubectl get nodes -o json`**
3-18	Show the labels on all the nodes: **`kubectl get nodes --show-labels`**
3-19	Notice that the workers are labeled with the recognizable tag of `node-role.kubernetes.io/worker=`, which sets the node to a role of none.
3-20	Review the Kubernetes configuration file: **`cat $HOME/.kube/config`**
3-21	View the Kubernetes configuration through the kubectl tool: **`kubectl config view`**

Task 4: Configure Kubernetes Administrator Access on the Jump Host

Set up Kubernetes administrator access on the jump host.

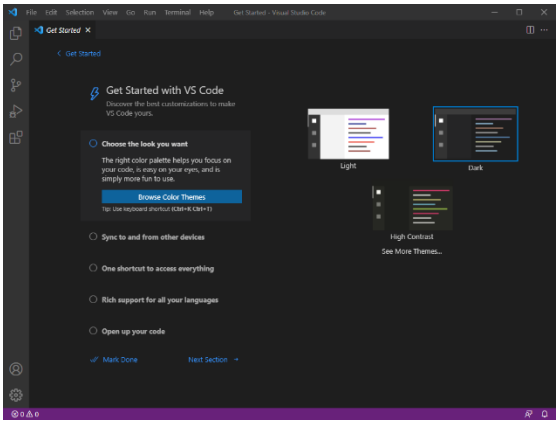
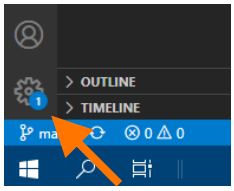
Step	Action
4-1	Start by installing kubectl on your jump host, and create a folder on your jump host: `C:\kube`
4-2	Within Chrome, enter **https://storage.googleapis.com/kubernetes-release/release/v1.23.1/bin/windows/amd64/kubectl.exe**
4-3	After you download the executable, move it to a new directory: `C:\kube`
4-4	Add the new path to your PATH environment variable, and by using the Windows search, type **PATH**. 
4-5	In Control Panel, select **Edit the system environment variables**.
4-6	Select **Environment Variables**.
4-7	With Path selected in the System variables (the bottom list of variables in the dialog box), click the **Edit** button. 

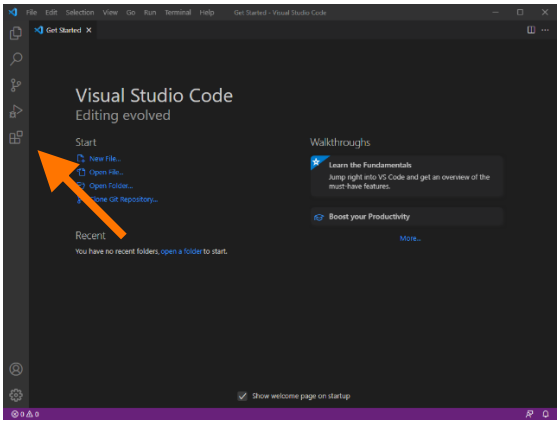
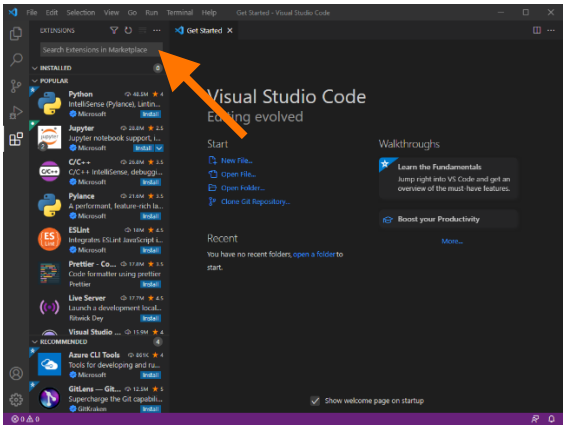
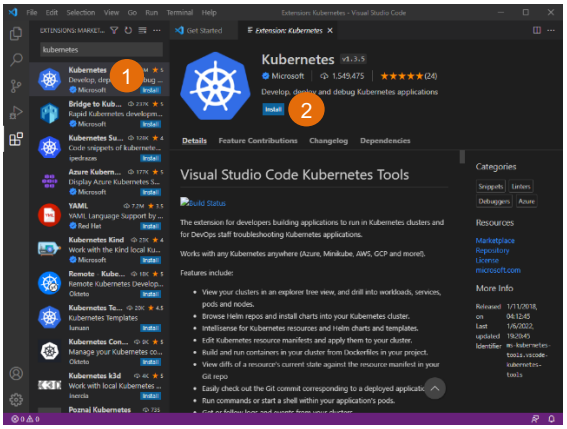
Step	Action
4-8	Click **New** to create an environment variable. 
4-9	In the new field, type the directory path **C:\kube**, and press **Enter**. 
4-10	Click **OK** to close the Edit dialog box.
4-11	Click **OK** to close the Environment Variables dialog box.
4-12	Click **OK** to close the System Properties dialog box.
4-13	Open a Command prompt window, and see whether the new PATH variable is working by typing: **`kubectl`** and then pressing **Enter**. You should receive output from the `kubectl` tool. If you do not, you should go back to Step 2-4 and verify that you have provided the correct directory.
4-14	Verify the installation and the version at a command prompt: **`kubectl version --client -o json`**
4-15	Verify the kube config: **`kubectl config view`**
4-16	 NOTE: The results are empty. You have not currently authenticated with the cluster and have no authorization to run any commands. You correct this situation now.
4-17	Create a folder called kube under your home directory (`C:\Users\Administrator.DEMO`): **`mkdir .kube`**

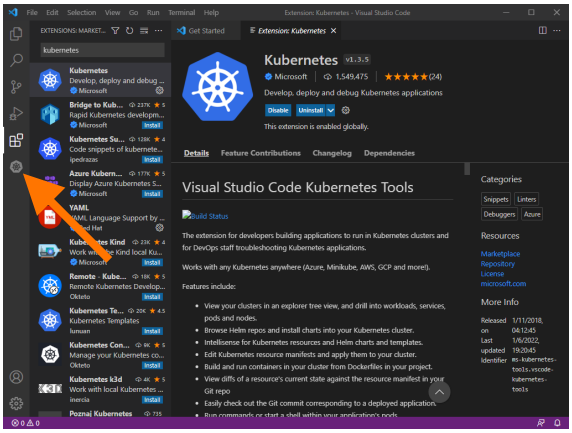
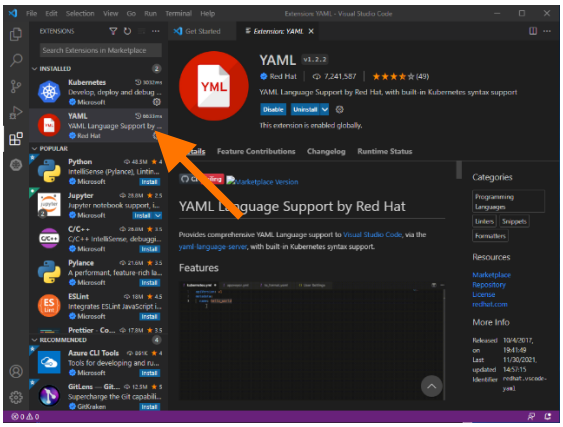
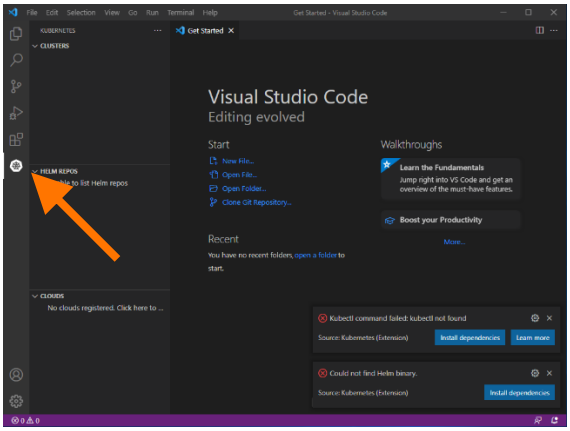
Step	Action
4-18	Use PuTTY Secure Copy tool to transfer the “config” file from `/root/.kube/config` on the KubMas control plane to the `.kube` folder on the jump host: **`pscp root@kubmas:/root/.kube/config C:\Users\Administrator.DEMO\.kube`**
4-19	Enter the root’s password when prompted: **`Netapp1!`** The result should be the `config` file copied to the `C:\Users\Administrator.DEMO\.kube`.
4-20	 If you get an error that the process could not create the file, you might have typed in a wrong path.
4-21	Reverify your credentials: **`kubectl config view`**
4-22	Verify that you can now communicate with your Kubernetes cluster: **`kubectl get nodes`**
4-23	 You have now successfully created a management host.

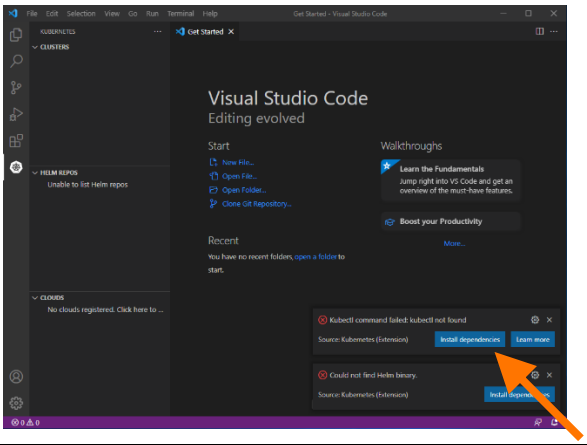
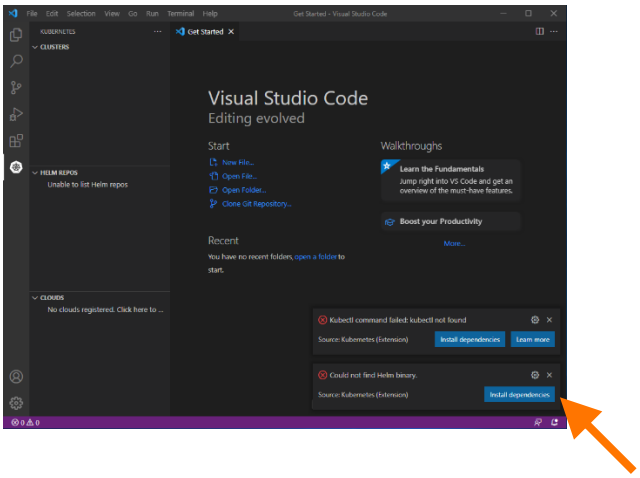
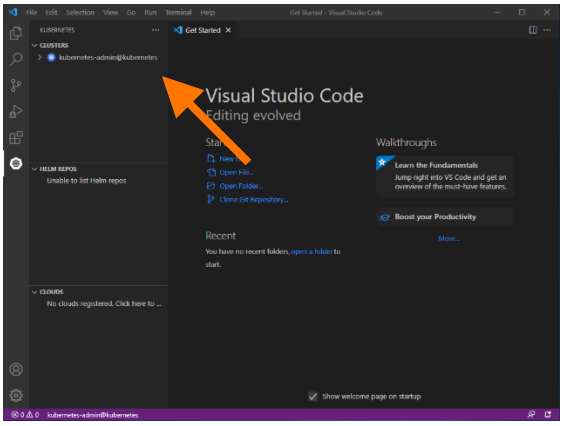
Task 5: Add the Courseware to the Integrated Development Environment

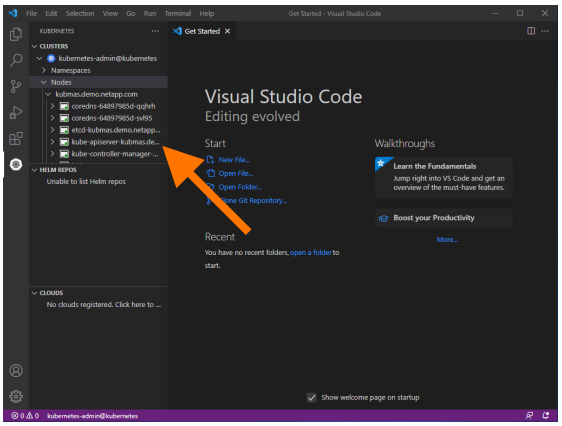
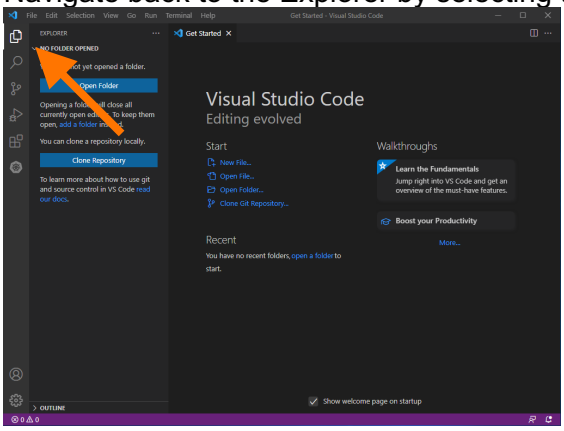
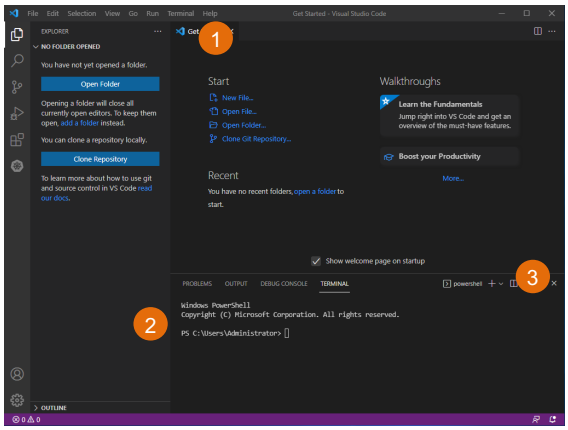
Throughout this course, you are editing and running YAML files. If you plan to take the Kubernetes certification exams, you should be comfortable creating and editing YAML files by using NANO on a Linux host. However, this procedure is not practical for daily operations. In this task, you configure an integrated development environment (IDE) to use during this class. Visual Studio Code (VSC) is one of the world's most popular IDEs. In this course, you see IDE and VSC used interchangeably.

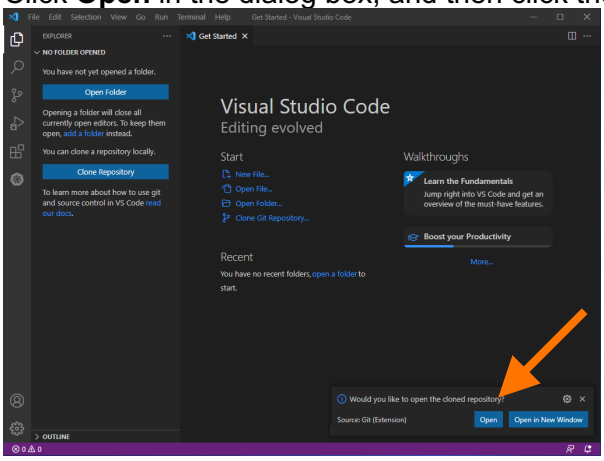
Step	Action
5-1	On your jump host, open Visual Studio Code. 
5-2	The jump host also has VSCodium installed. This program is an open-community version of the Visual Studio Code. Visual Studio Code is created and supported by Microsoft. Even though Visual Studio Code is free to use, telemetry data is collected by Microsoft. VSCodium is a non-Microsoft supported version of the Visual Studio Code binaries that has telemetry completely disabled. If you choose to use VSCodium, it has not been tested with these exercises.
5-3	Verify that the IDE opens. 
5-4	Set the look (theme) that you want, and click **Mark Done** at the bottom.
5-5	Periodically, Visual Studio Code might release a new updated. If available, you can select the gear icon with the badge indicator in the bottom left-hand corner of your IDE and then select **Restart to Update**. 

Step	Action
5-6	Navigate to the Extensions tab. 
5-7	In the Search Extensions in Marketplace section, enter **Kubernetes**. 
5-8	Select and install the Kubernetes extension by Microsoft. 

Step	Action
5-9	Verify that the extension has been added by noticing the new Kubernetes icon in the tab bar. 
5-10	Also notice that this extension added a second extension called YAML by Red Hat. 
5-11	Close the extensions page by selecting the **X** in the Extensions: YAML page, and navigate to the Kubernetes tab. 

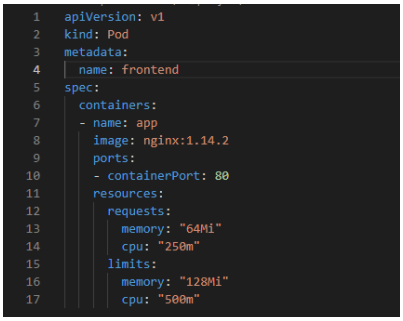
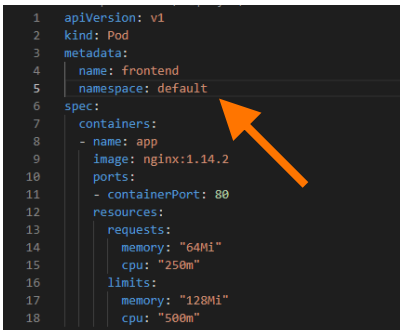
Step	Action
5-12	In the dialog box at the bottom right of the IDE, install the dependencies for the kubectl tool. 
5-13	When the previous step finishes, install the dependencies for Helm. However, you do not use Helm in this course. 
5-14	You can close the output panel if you want. The Kubernetes extension should now be authenticated with your Kubernetes cluster. The Kubernetes extension enables you to explore your Kubernetes cluster visually. 

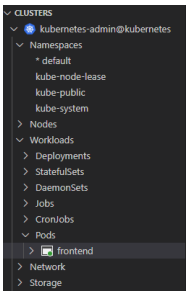
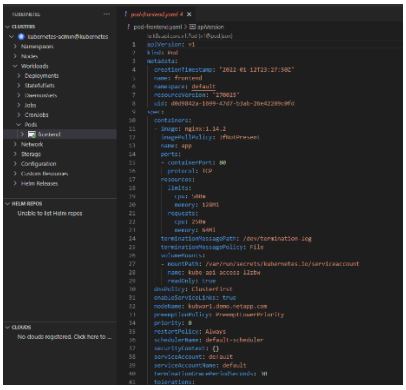
Step	Action
5-15	Check the nodes list, and verify that you see kubmas, kubwor1, and kubwor2. 
5-16	Explore to see what information is available through this extension.
5-17	Navigate back to the Explorer by selecting the Explorer tab. 
5-18	In your IDE, select **Terminal,** and then select **New Terminal** from the menus. A new terminal window should open. The default terminal is Windows PowerShell. However, if you are more comfortable with a bash or the command prompt terminal, you can select your favorite terminal with the down arrow next to the + sign in the terminal section. 

Step	Action
5-19	If you have properly set up the kubectl tool on your jump host, you should be able use it in a Terminal window of your IDE.
5-20	Verify the fact by running the **`kubectl version`** command in the Terminal window.
5-21	Close the Terminal window. This tool is available for you if you need to investigate or run any commands and use the visual Kubernetes extension.
5-22	To download the source files that you use in this course, type **Ctrl-Shift P**, type **Git: Clone**, and clone the repository from: `https://github.com/NetApp-Learning-Services/STRSW-ILT-UATWK`
5-23	Select a location to store your local clone of the courseware repository. For example, you can create a `repo` folder under `C:\Users\Administrator.DEMO` and select that folder.
5-24	Click **Open** in the dialog box, and then click the **Yes, I trust the authors** button. 
5-25	You should now have an explorer view of the files and folders.

Task 6: Work with YAML Files in the IDE

You use your IDE to deploy a pod within your Kubernetes cluster.

Step	Action
6-1	Within your IDE's explorer window, navigate to the folder Exercise 0 (this folder represents the exercise for Module 0), and select the file **`exercise0task6.yaml`**. 
6-2	Under the `name` key, type **`namespace: default`** to ensure that this pod runs in the default namespace. Spacing matters. You must ensure that this key-value pair is a sibling of the `name` key-value. 
6-3	 If you have difficulty, a completed version of the YAML file can be found in this exercise's subfolder that is entitled Solutions.
6-4	Notice the white dot on the tab with the file name. This dot indicates that the file has changed and needs to be saved.
6-5	Enter **Ctrl-S** to save the file.
6-6	With this file open, click **Ctrl-Shift-P,** and start typing **`Kubernetes`**. You see a list of Kubernetes-related commands.
6-7	Select **`Kubernetes: Create`** to execute this YAML definition file. You should see a dialog box that tells you that the pod/front end was created.

Step	Action
6-8	Navigate to the Kubernetes extension in your IDE and see whether you can find the pod that was created.  Help: You can navigate to `Workloads > Pods` and select the front-end pod. 
6-9	Double-click the front-end pod to review the pod's running configuration. 
6-10	Right-click the front-end pod in the extension, select **Delete Now** to remove the pod, and confirm the deletion in the dialog box.
6-11	Close the **`exercise0task6.yaml`** file in the IDE.

End of exercise

Module 1: Kubernetes Storage Overview

Exercise 1: Working with Kubernetes Storage Volumes

In this exercise, you explore native Kubernetes storage objects, including emptyDir, hostPath, and nfs volume types. You also create a manual persistent volume (PV) and persistent volume claim (PVC).

Objectives

This exercise focuses on enabling you to do the following:

- Set up an emptyDir volume
- Configure a hostPath volume
- Deploy a pod with two containers that share resources
- Configure an NFS server by using NetApp ONTAP software
- Set up an NFS volume
- Configure a persistent volume and persistent volume claim

Exercise Equipment

In this exercise, you use the following systems.

System	Host Name	IP Addresses	User Name	Password
Windows Server	Jumphost	192.168.0.5	DEMO\Administrator	Netapp1!
Kubernetes Worker 1	kubwor1	192.168.0.62	root (case sensitive)	Netapp1!
Kubernetes Worker 2	kubwor2	192.168.0.63	root (case sensitive)	Netapp1!

Task 1: Set Up an emptyDir Volume

In this task, you create the simplest volume option available in Kubernetes: emptyDir.

Step	Action

1-1	Review and update the `exercise1Task1.yaml` file with the following, and then save the file.
	• volumeMounts
	  ○ mountPath: **`/opt/this`**
	  ○ name: **`myvol`**
	• volumes
	  ○ name: **`myvol`**
	  ○ emptyDir: **`{}`**

1-2	Create an instance of the `exercise1Task1.yaml` file: **`kubectl create -f exercise1Task1.yaml`**
1-3	Describe the pod again and review the events: **`kubectl describe pod emptydir-pod`**

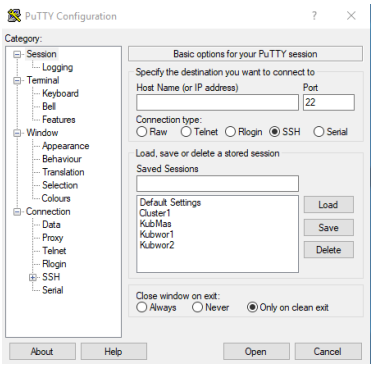
Step	Action
1-4	Answer the following question: Is the pod running?
1-5	Connect to the pod: **`kubectl exec -it emptydir-pod -- /bin/sh`**
1-6	From the container's prompt, list the directory of the `/opt/this` directory.
1-7	From the container's prompt, create a file with touch in the `/opt/this` directory.
1-8	Exit the container's prompt.
1-9	Delete emptyDir pod: **`kubectl delete pod emptydir-pod`**
1-10	 Throughout this course, CHALLENGE STEPS have been included to help you to explore deeper sections of the Kubernetes ecosystem. NOTE: These CHALLENGE STEPS are optional, but you cannot skip steps if you want to complete the entire challenge.
1-11	CHALLENGE STEP: Run Steps 1-2 through 1-6 again, and answer the following question: Does the file that you created in Step 1-7 still exist?
1-12	CHALLENGE STEP: Exit the container's prompt, and delete the second instance of the emptyDir pod.

Task 2: Configure a hostPath Volume

In this task, you create a Debian container and provide access to a local mount point on the hosting Worker node.

Step	Action

2-1	Review and update the `exercise1Task2-1.yaml` file with the following definitions, and then save the file.
	• spec volumes:
	  ○ name: **`local-vol`**
	  ○ hostPath:
	  ▪ path: **`/hostVol`**
	  ▪ type: **`Directory`**
	• Containers volumeMounts:
	  ○ mountPath: **`/root/`**
	  ○ name: **`local-vol`**

2-2	There is a solution in the subfolder.
2-3	Create an instance of the `exercise1Task2-1.yaml` file: **`kubectl create -f exercise1Task2-1.yaml`**
2-4	Identify on which node the pod is running by using the **`kubectl get pods -o wide`** command.
2-5	Describe the pod, review the events, and answer the following questions: Is the pod running? Why or why not?
2-6	Using PuTTY, open a connection to the node that is running the pod. 
2-7	At the Linux login prompt, provide the following credentials: • Login as: **`root`** • Password: **`Netapp1!`** The Linux CLI prompt and cursor appear.

Step	Action
2-8	From the Worker's prompt, create the required directory: **`mkdir /hostVol`**
2-9	From the Worker's prompt, create a local file in the directory: **`touch /hostVol/exercise1Task2`**
2-10	Describe the pod again, and review the events: **`kubectl describe pod hostpath-pod`**
2-11	Answer the following question: Is the pod running?
2-12	 If not, you might delete it and re-create it.
2-13	Connect to the pod: **`kubectl exec -it hostpath-pod -- /bin/sh`**
2-14	From the container's prompt, list the directory of the `/root` directory.
2-15	Answer the following question: Do you see the file that you created from the local Worker node?
2-16	From the container's prompt, create a file in the `/root` directory.
2-17	Answer the following question: Do you see the file that you created from a Secure Shell (SSH) session on the local Worker node?
2-18	Answer the following question: What would happen to access to the file if the pod were destroyed and re-created on the other worker node?
2-19	Delete the pod: **`kubectl delete pod hostpath-pod`**
2-20	Copy the completed YAML and rename the copy as **`exercise1Task2-2.yaml`** file.

2-21	Change the following in the `exercise1Task2-2.yaml` file, and then save the file.
	• Pod name: **`hostPath2-pod`**
	• hostPath:
	  ○ path: **`/hostVol2`**
	  ○ type: **`DirectoryOrCreate`**

2-22	Create an instance of the `exercise1Task2-2.yaml` file: **`kubectl create -f exercise1Task2-2.yaml`**

Step	Action
2-23	Describe the pod again, and review the events: **`kubectl describe pod hostpath2-pod`**
2-24	Answer the following question: Is the pod running?
2-25	Notice that the /hostVol2 was created on the appropriate Worker node.
2-26	Connect to the pod: **`kubectl exec -it hostpath2-pod -- /bin/sh`**
2-27	From the container's prompt, list the directory of the /root directory.
2-28	From the container's prompt, create a file in the mounted path.
2-29	Click **Ctrl-D** to exit to the execute command.
2-30	Delete the hostpath2-pod pod.
2-31	CHALLENGE STEP: Delete and re-create the pod, see which node is selected to host the second instance of the pod, and answer the following question: What happens to the hostPath directory on the local Worker node?

Task 3: Deploy a Pod with Two Containers That Share Storage

In this task, you define a volume named `shared-data` and its type is `emptyDir`. The first container runs nginx server and has the shared volume mounted to the directory `/usr/share/nginx/html`. The second container is based on the Debian image. It has the shared volume mounted to the directory `/pod-data` that runs a while loop that writes the current date and time to the file `index.html` that is in the shared volume. The container then waits 10 second before repeating the loop.

Step	Action

3-1	Review and update the `exercise1Task3.yaml` file with the following definitions, and then save the file.
	• spec volumes:
	  ○ name: **`shared-vol`**
	  ○ Add an emptyDir volume defintion
	• In both containers' volumeMounts defintion:
	  ○ name: **`shared-vol`**

3-2	 There is a solution in the subfolder.
3-3	Create an instance of the `exercise1Task3.yaml` file: **`kubectl create -f exercise1Task3.yaml`**
3-4	Connect to the `first` container of the pod: **`kubectl exec -it two-pod -c first -- /bin/bash`**
3-5	From the container's prompt, verify that the `index.html` page is being updated every second: **`# tail /usr/share/nginx/html/index.html`**
3-6	Click **Ctrl-D** to exit to the execute command.
3-7	Create a service that enables a Worker node access to `two-pod`: **`kubectl expose pod two-pod --type=NodePort --port=80`**
3-8	Identify the service nodeport: **`kubectl describe service two-pod`**

3-9	Open a web browser to the following: **`http://[a worker's IP address]:[nodeport]`**.
	Sample output:
	`Fri Feb 25 17:05:43 UTC 2022 Hello from the second container`
	`Fri Feb 25 17:05:44 UTC 2022 Hello from the second container`
	`Fri Feb 25 17:05:45 UTC 2022 Hello from the second container`
	`Fri Feb 25 17:05:46 UTC 2022 Hello from the second container`

3-10	Delete the pod named two-pod and remove the service.

Task 4: Configure an NFS Server by Using NetApp ONTAP Software

In this task, you create an NFS server in a storage VM (storage virtual machine, also known as SVM) in your assigned ONTAP cluster. You expose a volume to read/write access through an NFS export.

Step	Action
4-1	Open a browser to **https://192.168.0.101** (which is your Cluster 1 management LIF's IP address).
4-2	Authenticate with your ONTAP cluster by providing the following credentials: • Login as: **`admin`** • Password: **`Netapp1!`**
4-3	Click **Sign In**.
4-4	Using the left pane, navigate to Storage > Storage VMs.
4-5	Click **Add** to start the wizard to create an SVM.

4-6	Enter the following information:
	• Storage VM Name: **svm0**
	• Select **SMB/CIFS, NFS** tab
	• Access Protocol: Select **Enable NFS**
	• Allow NFS client access: Selected
	• Rule: Add a rule:
	  ○ Client Specification: **0.0.0.0/0**
	  ○ Protocols: **NFS** (both **v3** and **v4**)
	  ○ Read-Only: Selected
	  ○ Read/Write: Selected
	  ○ Click **Save**.
	• Under Network Interface:
	  ○ IP Address: **192.168.0.31**
	  ○ Subnet mask: **255.255.255.0**
	  ○ Broadcast domain: Default
	• Manage administrator account: Selected
	  ○ User name: **vsadmin**
	  ○ Password: **Netapp1!**
	  ○ Confirm Password: **Netapp1!**

4-7	At the end of the dialog box, click **Save**. You should now see the svm0 in the list of Storage VMs.
4-8	Click the newly created **svm0** link. You should see the Overview of the svm0.
4-9	Click the **Edit** button in the upper-right corner.

Step	Action
4-10	Select the Resource Allocation checkbox to prefer local tiers, and make sure that Cluster1_01_FC_1 is selected in the list of local tiers.
4-11	Click **Save**.
4-12	Navigate to Storage > Volumes.
4-13	Click **Add** to create a volume.
4-14	Add the following details: • Name: **nfs** • Size: **1 GB** • Click the **More** button. • Ensure that the Export via NFS checkbox is selected. • Ensure that the default rule has all NFS protocols selected with the clients of 0.0.0.0/0.
4-15	Click **Save** to create the volume.
4-16	Identify and select the link of the new nfs volume in the list of volumes.
4-17	While viewing the Overview of the nfs volume, click the **Edit** button in the right corner.
4-18	In the details of the volume, perform the following: • Ensure that the security type is UNIX • Under UNIX Permissions, select **Read**, **Write**, and **Execute** so that all checkboxes are selected.
4-19	Click **Save** to complete the volume's edit.

Task 5: Set Up an NFS Volume

In this task, you create a pod that directly connects to the NFS export that you created in the Task 4 of this exercise.

Step	Action

5-1	Review and update the `exercise1Task5.yaml` file with the following definitions and save the file.
	• spec volumes:
	  ○ name: **`nfsvol`**
	  ○ nfs:
	  ▪ server: **`192.168.0.31`**
	  ▪ path: **`/nfs`**
	  ▪ readOnly: **`false`**

Step	Action
5-2	 There is a solution in the subfolder.
5-3	Create an instance of the `exercise1Task5.yaml` file: **`kubectl create -f exercise1Task5.yaml`**
5-4	Connect to the alpine pod that you just created: **`kubectl exec direct-nfs-pod -it -- /bin/sh`**
5-5	From the container's prompt, change the directory to volume mount: **`# cd /opt/this`**
5-6	From the container's prompt, create a file and do other operations at this location to demonstrate that you have read/write access: **`# echo "<html><body>hello</body></html>" > index.html`** **`# ls`** **`# cat index.html`**
5-7	Press **Ctrl-D** to exit the container's prompt.
5-8	Delete the pod: **`kubectl delete pod direct-nfs-pod`**
5-9	 In the ONTAP System Manager, under the `nfs` volume page, you can select the Explorer tab to view the folders and files created in the pod. The data is persisted in the ONTAP cluster.

Task 6: Configure a PV and a PVC

In this task, you create a PV that is manually attached to the NFS export that you created in Task 4. Then you bind that PV to a PVC. Finally, you use the PVC in a pod so that the pod's container can access the NFS export. This process of creating PVs is automated later by using NetApp Astra Trident.

Step	Action

6-1	Review and update the `exercise1Task6-1.yaml` file with the following PV spec definitions, and save the file.
	• Add an `nfs` definition with the following values:
	  ○ server: **`192.168.0.31`**
	  ○ path: **`/nfs`**
	  ○ readOnly: **`false`**

6-2	 There is a solution in the subfolder.

Step	Action
6-3	Create an instance of the PV in the `exercise1Task6-1.yaml` file: **`kubectl create -f exercise1Task6-1.yaml`**
6-4	Verify that the `manual-nfs-pv` PV was created: **`kubectl get pv`**
6-5	Describe the `manual-nfs-pv` PV: **`kubectl describe pv manual-nfs-pv`**
6-6	Review and update the `exercise1Task6-2.yaml` file with the following PVC spec definitions, and save the file. • accessModes: **`ReadWriteMany`** • storageClassName: **`''  # empty string`** • volumeName: [name of the pv from Step 6-3] • resources:requests:storage: **`1Gi`**
6-7	 There is a solution in the subfolder.
6-8	Create an instance of the PVC in the `exercise1Task6-2.yaml` file: **`kubectl create -f exercise1Task6-2.yaml`**
6-9	Verify that the `manual-nfs-pvc` PVC was created **`kubectl get pvc`**
6-10	Describe the `manual-nfs-pvc` PVC: **`kubectl describe pv manual-nfs-pvc`**
6-11	Make sure that the PVC has a status of bound, which means that the PVC was mapped to the PV.
6-12	Review and update the `exercise1Task6-3.yaml` file with the following Pod volumes definitions, and save the file. • name: **`nfs-storage`** • persistententVolumeClaim:claimName: [name of the pvc that you created in Step 6-8]
6-13	 There is a solution in the subfolder.
6-14	Create an instance of the PVC in the `exercise1Task6-3.yaml` file: **`kubectl create -f exercise1Task6-3.yaml`**
6-15	Verify that the `manual-nfs-pod` pod was created **`kubectl get pod`**

Step	Action
6-16	Describe the `manual-nfs-pod` pod: **`kubectl describe pod manual-nfs-pod`**
6-17	Ensure that the pod is using the PV backed by the NFS export in `svm0`.
6-18	CHALLENGE STEP: Execute a shell into the manual-nfs-pod and navigate to the mount path of the PV. Then verify that the index.html file created in the previous task is available to the nginx server.
6-19	CHALLENGE STEP: Create a NodePort service for the manual-nfs-pod and open the index.html file you created in a browser.
6-20	Delete the `manual-nfs-pod` pod: **`kubectl delete pod manual-nfs-pod`**

End of exercise

Module 2: Astra Trident Installation

Exercise 1: Installing Astra Trident

In this exercise, you install NetApp Astra Trident by using the manual operator method. Other approaches that are available but not discussed are installing Astra Trident operator by using Helm or installing Astra Trident by using the `tridentctl` install method.

Objectives

This exercise focuses on enabling you to do the following:

- Download and set up the Astra Trident operator
- Deploy instances of Astra Trident
- Set up the `tridentctl` tool
- Prepare worker nodes

Exercise Equipment

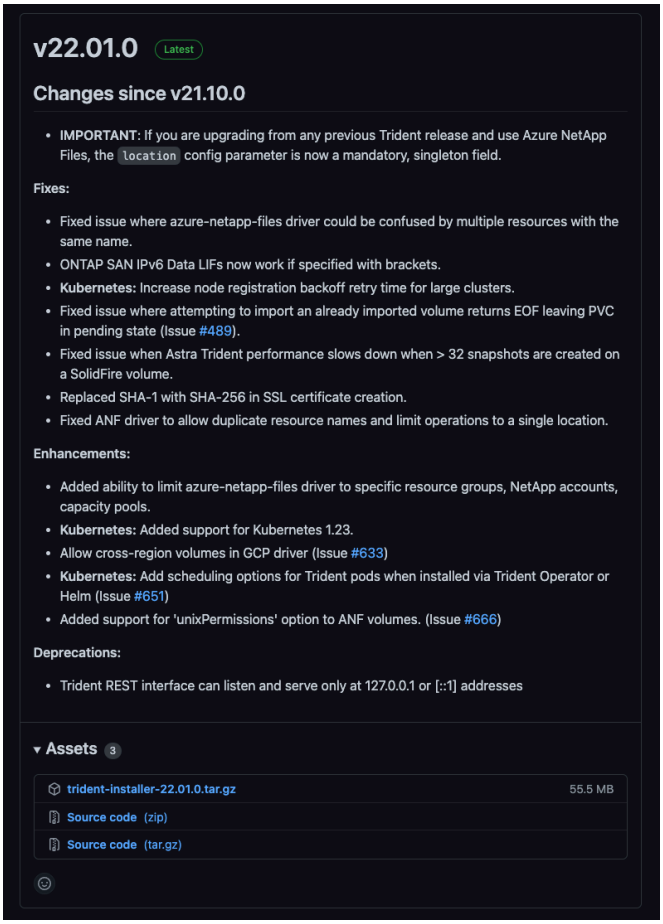
In this exercise, you use the following systems.

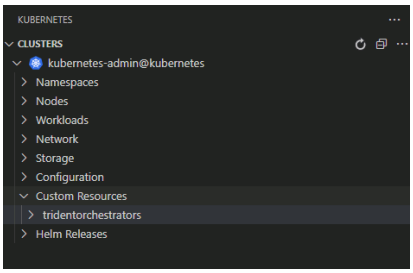
System	Host Name	IP Addresses	User Name	Password
Windows Server	Jumphost	192.168.0.5	DEMO\Administrator	Netapp1!
Kubernetes Control Plane	kubmas	192.168.0.61	root (case sensitive)	Netapp1!
Kubernetes Worker 1	kubwor1	192.168.0.62	root (case sensitive)	Netapp1!
Kubernetes Worker 2	kubwor2	192.168.0.63	root (case sensitive)	Netapp1!

Task 1: Download and Set Up the Astra Trident Operator

In this task, you verify that you have access to the Kubernetes cluster and download and set up the Astra Trident operator.

Step	Action
1-1	If desired, you can follow along with this exercise on the Astra Trident operator deployment: https://docs.netapp.com/us-en/trident/trident-get-started/kubernetes-deploy-operator.html#deploy-the-trident-operator-manually.
1-2	Verify that you have administrative access to the Kubernetes cluster: `kubectl auth can-i '*' '*' --all-namespaces`
1-3	In a future exercise, you implement Container Storage Interface (CSI) topologies. To support this effort, you label each of the Worker nodes with different labels. These labels should be present on nodes in the cluster before Astra Trident is installed for Astra Trident to be topology aware.
1-4	Label Worker 1 as Zone 1 and a region of Astra Trident: `kubectl label node kubwor1.demo.netapp.com topology.kubernetes.io/region=trident topology.kubernetes.io/zone=zone1`

Step	Action
1-5	Label Worker 2 as Zone 2 and a region of Astra Trident: **`kubectl label node kubwor2.demo.netapp.com topology.kubernetes.io/region=trident topology.kubernetes.io/zone=zone2`**
1-6	Navigate to https://github.com/Netapp/trident/releases by using a Web browser.
1-7	Identify the latest version at the top of the page: 
1-8	If desired, you can download a newer version (however, the exercises have not been tested with any version other than 22.01.0. If you want to work with this version of the exercises, you can find the tar.gz file already downloaded in your Exercise 2 folder for your class files.
1-9	Open a terminal within your integrated development environment (IDE).
1-10	In the terminal, change directory to the Exercise 2 subfolder: **`cd 'Exercise 2'`**
1-11	Unzip the Astra Trident file: **`tar -xf trident-installer-22.01.0.tar.gz`** A new subfolder under the Exercise 2 folder should appear called trident-installer.
1-12	Within your IDE browser, navigate to the `Exercise 2 > trident-installer > deploy` folder. This path is going to be the relative path for all future paths in this task.

Step	Action
1-13	Navigate to the `crds` subfolder.
1-14	There are several potential custom resource definition (CRD) YAML files in this folder. As you review them, you notice that two of the definition files have `crd` in their names and five files have `cr` in their names.
1-15	Create the CRD definitions by using the `trident.netapp.io_tridentorchestrators_crd_post1.16.yaml` file: **`kubectl create -f deploy/crds/trident.netapp.io_tridentorchestrators_crd_post1.16.yaml`**
1-16	Within the Kubernetes Extension of your IDE, you should see the `tridentorchestrators` CRD.  If you see an error under the trident CRD, you should click the **Refresh** button to make it disappear.
1-17	Create the `trident` namespace: **`kubectl create -f deploy/namespace.yaml`**
1-18	Open, review, and create the service account that is associated with the Astra Trident operator: **`kubectl create -f deploy/serviceaccount.yaml`**
1-19	Open, review, and create the cluster role that is associated with the operator: **`kubectl create -f deploy/clusterrole.yaml`**
1-20	Open, review, and create the cluster role binding that is associated with the operator: **`kubectl create -f deploy/clusterrolebinding.yaml`**
1-21	Open, review, and create the operator that is associated with Astra Trident: **`kubectl create -f deploy/operator.yaml`**
1-22	Open, review, and create the pod security policy that is associated with Astra Trident: **`kubectl create -f deploy/podsecuritypolicy_unprivileged.yaml`**
1-23	Steps 1-14 through 1-18 could have been run with `deploy/bundle.yaml`. You could also have used kustomize to change the `Trident` default namespaces and generate a different `bundle.yaml` file if you wanted to place these objects in a namespace other than `Trident`. For more information regarding kustomize, you can see https://kustomize.io/.

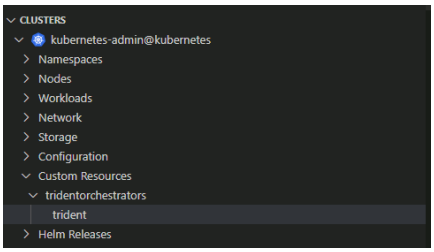
Step	Action

1-24	Verify that all objects are created:
	`kubectl get all -n trident`
	Sample output:
	`PS C:\Users\Administrator.DEMO\repo2\UsingAstra> kubectl get all -n trident`
	`NAME READY STATUS RESTARTS AGE`
	`pod/trident-operator-7cdc56fd67-j6nnw 1/1 Running 0 5s`
	
	`NAME READY UP-TO-DATE AVAILABLE AGE`
	`deployment.apps/trident-operator 1/1 1 1 5s`
	
	`NAME DESIRED CURRENT READY AGE`
	`replicaset.apps/trident-operator-7cdc56fd67 1 1 1 5s`

1-25	 There should be only *one instance* of the operator in a Kubernetes cluster. You must not create multiple deployments of the Astra Trident operator.

Task 2: Deploy Instances of Astra Trident

You are now ready to deploy Astra Trident by using the operator. This action requires creating `TridentOrchestrator` custom resource (CR). The Astra Trident installer comes with example definitions for creating `TridentOrchestrator`. The CR kicks off an installation in the `trident` namespace.

Step	Action
2-1	Review the `deploy/crds/tridentorchestrator_cr.yaml` file.
2-2	The TridentOrchestrator CR enables you to customize the Astra Trident operator. You can view this URL for more details: https://docs.netapp.com/us-en/trident/trident-get-started/kubernetes-customize-deploy.html There are several examples of modifications in the crds subfolder.
2-3	Create an instance of the `TridentOrchestrator` CR: **`kubectl create -f deploy/crds/tridentorchestrator_cr.yaml`**
2-4	Within the Kubernetes Extension of your IDE, you should see the `trident` instance under the `tridentorchestrators` CRD. 
2-5	Review the details by double-clicking the trident entry in the hierarchy, or use the following: **`kubectl describe torc trident`**
2-6	Answer the following question: In the events section, what is the last event type and reason?

Step	Action

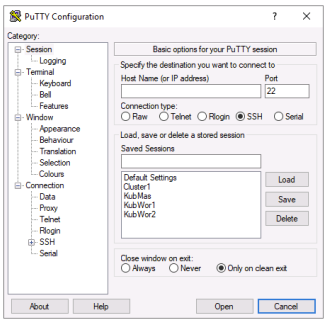
2-7	Verify that all objects are created:
	`kubectl -n trident get all`
	Sample output:
	`PS C:\Users\Administrator.DEMO\repo2\UsingAstra> kubectl get all -n trident`
	`NAME READY STATUS RESTARTS AGE`
	`pod/trident-csi-59xjj 2/2 Running 0 9m43s`
	`pod/trident-csi-6hq2c 2/2 Running 0 9m43s`
	`pod/trident-csi-7f5df5ffcf-vcv28 6/6 Running 0 9m43s`
	`pod/trident-csi-wjq9d 2/2 Running 0 9m43s`
	`pod/trident-operator-7cdc56fd67-j6nnw 1/1 Running 0 22m`
	
	`NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE`
	`service/trident-csi ClusterIP 10.110.41.241 <none> 34571/TCP,9220/TCP 9m43s`
	
	`NAME DESIRED CURRENT READY UP-TO-DATE AVAILABLE`
	`NODE SELECTOR AGE`
	`daemonset.apps/trident-csi 3 3 3 3 3`
	`kubernetes.io/arch=amd64,kubernetes.io/os=linux 9m43s`
	
	`NAME READY UP-TO-DATE AVAILABLE AGE`
	`deployment.apps/trident-csi 1/1 1 1 9m43s`
	`deployment.apps/trident-operator 1/1 1 1 22m`
	
	`NAME DESIRED CURRENT READY AGE`
	`replicaset.apps/trident-csi-7f5df5ffcf 1 1 1 9m43s`
	`replicaset.apps/trident-operator-7cdc56fd67 1 1 1 22m`

2-8	There are four additional pods with `csi` in the name, indicating that Astra Trident is now completely deployed. Three of the `csi` pods (with the shorter names) are created using the daemonset `trident-csi` and has one `csi` pod installed on each node (including the control-plane master node). The other csi pod (with the longest name) is created using the `trident-csi` deployment and is running on one of the worker nodes.
2-9	Stop the deployed Astra Trident pods by deleting the `TridentOrchestrator` CR: **`kubectl delete torc trident`**
2-10	Verify that all pods with `csi` in their name are deleted and only the Astra Trident operator is running: **`kubectl get pods -n trident -o wide`**
2-11	New in Astra Trident 22.01, you can now designate which nodes should run Astra Trident with taints and tolerations. This new feature is supported when deploying Astra Trident by using either the operator or with a Helm chart.
2-12	You try to run Astra Trident CSI deployment only on kubmas
2-13	Create taints on the worker nodes: **`kubectl taint node kubwor1.demo.netapp.com trident=true:NoSchedule`** **`kubectl taint node kubwor1.demo.netapp.com trident=true:NoSchedule`**

Step	Action
2-14	Edit the `deploy/crds/tridentorchestrator_cr.yaml` file to add an appropriate toleration: • Definition: **`nodePluginTolerations`** (see https://docs.netapp.com/us-en/trident/trident-get-started/kubernetes-customize-deploy.html) • Key: **`"trident"`** • Operator: **`"Equal"`** • Value: **`"true"`** • Effect: **`"NoSchedule"`** You can find the solution of this step in `exercise2Task2-toleration.yaml`.
2-15	Create an instance of the `TridentOrchestrator` CR: **`kubectl create -f [the edited yaml from the previous step]`**
2-16	Verify that one of the `csi`-named pod from the deployment is pending: **`kubectl -n trident get pods -o wide`**
2-17	*i* Regarding the deployment `csi`-named pod, notice that only one node has a taint of master (kubmas) that the pod did not tolerate, and 2 nodes (kubwor1 and kubwor2) has a taint of trident: true that the pod did not tolerate.
2-18	Remove the taint on one of the worker nodes: **`kubectl taint node kubwor1.demo.netapp.com trident=true:NoSchedule-`**
2-19	Verify that the deployment `csi`-named pod is either being created or is running on worker 1: **`kubectl -n trident get pods -o wide`**
2-20	CHALLENGE STEP: It is possible to update an existing instance of the Trident CR, use the patch command (for example, if you want to turn off debug logs, use the following): **`kubectl -n trident patch torc trident --type=json /`** **`-p '[{"op": "replace", "path": "/spec/debug", "value": "false"}]'`**
2-21	Verify that the debug logs are on.

Task 3: Set Up the `tridentctl` Tool

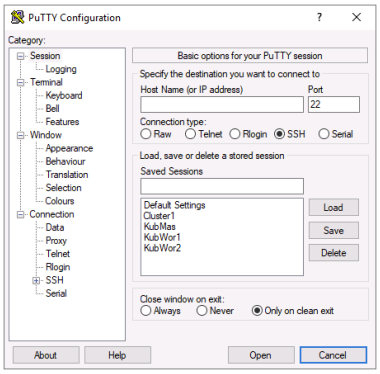
You copy the `tridentctl` tool to the Kubernetes control plane node.

Step	Action
3-1	Use PuTTY Secure Copy tool to transfer the `tridentctl` file to `/root` on the KubMas control plane from the jump host: **`pscp "C:\Users\Administrator.DEMO\[repository location]\STRSW-ILT-UATWK\Exercise 2\trident-installer\tridentctl" root@kubmas:/root`**
3-2	Enter the root's password when prompted: **`Netapp1!`** The result should be the `tridentctl` file copied to the `/root`.
3-3	Open a new session on KubMas (Kubernetes Control Plane): 
3-4	At the Linux login prompt, provide the following credentials: • Login as: **`root`** • Password: **`Netapp1!`** The Linux CLI prompt and cursor appear.
3-5	Verify that the tridentctl file was successfully transferred: **`ls -al`** Sample output: … `-rw-r--r--  1 root root 76238848 Jan 31 22:44 tridentctl`
3-6	Notice that the permissions of the file prevent execution, and change the permissions: **`chmod -R 777 tridentctl`**
3-7	Verify that the tridentctl file can now run: **`ls -al`** Sample output: … `-rwxrwxrwx  1 root root 76238848 Jan 31 22:44 tridentctl`
3-8	Review the tridentctl subcommands: **`./tridentctl`**

Step	Action
3-9	See the version of Astra Trident that is installed: **`./tridentctl -n trident version`** Sample output: `[root@kubmas ~]# ./tridentctl -n trident version` `+----------------+----------------+` `| SERVER VERSION	CLIENT VERSION |` `+----------------+----------------+` `| 22.01.0	22.01.0 |` `+----------------+----------------+`

Task 4: Prepare Worker Nodes

In this task, you verify that the worker nodes can use volumes that are provided by Astra Trident. You install the correct storage protocol tooling on the worker nodes. Currently, there is a beta feature that enables automatic worker node preparation by using the `tridentctl` tool. For more information, you can see https://docs.netapp.com/us-en/trident/trident-use/automatic-workernode.html.

Step	Action
4-1	Open a new session on KubWor1 (Kubernetes Worker 1): 
4-2	At the Linux login prompt, provide the following credentials: • Login as: **`root`** • Password: **`Netapp1!`** The Linux CLI prompt and cursor appear.
4-3	Verify that the NFSv3 protocol can run on the worker, and verify that nfsd is running: **`ps aux | grep nfsd`**
4-4	Verify that portman and mountd are also running.
4-5	Check that the nfs-utils was installed: **`yum list installed | grep nfs`**
4-6	Check to see whether iscsi-initiator is installed: **`yum list installed | grep iscsi`**
4-7	Verify that the multipath and device-mapper are installed.
4-8	Verify that iscsid and multipathd are running: **`sudo systemctl status iscsid multipathd`**
4-9	⚠ If /etc/multipath.conf does not exist, generate the file: **`mpathconf --enable`** Then restart the processes: **`sudo systemctl restart iscsid multipathd`**
4-10	Verify your initiator nodename: **`cat /etc/iscsi/initiatorname.iscsi`**
4-11	Repeat this task on Worker 2 (kubwor2).

End of exercise

Module 3: Astra Trident Configuration

Exercise 1: Working with Astra Trident

Objectives

This exercise focuses on enabling you to do the following:

- Create a NAS back end
- Create a storage class for a NAS back end
- Provision a persistent volume claim (PVC) with a NAS back end
- Mount the volumes in a pod
- Perform back end management by using the `tridentctl` tool
- Perform back end management with the `kubectl` tool

Exercise Equipment

In this exercise, you use the following systems.

System	Host Name	IP Addresses	User Name	Password
Windows Server	Jumphost	192.168.0.5	DEMO\Administrator	Netapp1!
Kubernetes Control Plane	kubmas	192.168.0.61	root (case sensitive)	Netapp1!

Task 1: Create a NAS Back End

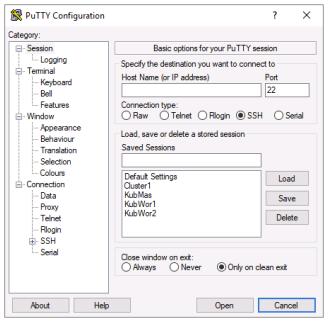
In this exercise, you use a back end with NetApp ONTAP NAS drivers. There are numerous other back ends you can create. For more information, see https://docs.netapp.com/us-en/trident/trident-use/backends.html. NOTE: When you unzip the trident tar file, the path `trident-installer/sample-inputs` provides many examples of configuration files.

Step	Action
1-1	Steps 1-2 through 1-11 repeat the steps of Exercise 1, Task 4, Steps 4-1 through 4-11. If you created the storage VM (storage virtual machine, also known as SVM) that provides NFS access in Exercise 1, begin at Step 1-12 in this task.
1-2	To create an SVM to use, open a browser to https://192.168.0.101 (which is your Cluster 1 management LIF's IP address).
1-3	Authenticate with your ONTAP cluster by providing the following credentials: • Login as: **admin** • Password: **Netapp1!**
1-4	Click **Sign In**.
1-5	Using the left pane, navigate to Storage > Storage VMs.
1-6	Click **Add** to start the wizard to create an SVM.

Step	Action

1-7	Enter the following information, and then click **Save**.
	• Storage VM Name: **svm0**
	• Select **SMB/CIFS, NFS tab**
	• Access Protocol: Select **Enable NFS**
	• Allow NFS client access: Selected
	• Rule: Add a rule:
	  ○ Client Specification: **0.0.0.0/0**
	  ○ Protocols: **NFS** (both **v3** and **v4**)
	  ○ Read-Only: Selected
	  ○ Read/Write: Selected
	  ○ Click **Save**
	• Under Network Interface:
	  ○ IP Address: **192.168.0.31**
	  ○ Subnet mask: **255.255.255.0**
	  ○ Broadcast domain: Default
	• Manage administrator account: Selected
	  ○ User name: **vsadmin**
	  ○ Password: **Netapp1!**
	  Confirm Password: **Netapp1!**

1-8	When you see the svm0 in the list of Storage VMs, click the newly created **svm0** link.
1-9	When you see the Overview of the svm0, click the **Edit** button in the upper-right corner.
1-10	Select the Resource Allocation checkbox to prefer local tiers, and make sure that Cluster1_01_FC_1 is selected in the list of local tiers.
1-11	Click **Save**.
1-12	Modify the `exercise3Task1.json` file to add the appropriate settings: • Version: **`1`** • Storage Driver Name: **`"ontap-nas"`** • Backend Name: **`"ontap-nas-backend"`** • Management LIF: **`"192.168.0.101"`** • Data LIF: **`"192.168.0.31"`** • SVM: **`"svm0"`** • Username: **`"admin"`** • Password: **`"Netapp1!"`**
1-13	Save the JSON file.

Step	Action
1-14	The back-end definition is the only place that the credentials are stored in plain text. After the back end is created, usernames and passwords are encoded with Base64 and stored as Kubernetes secrets. The creation or update of a back end is the only step that requires knowledge of the credentials. It should be an admin-only operation.
1-15	Use PuTTY Secure Copy tool to transfer the JSON file to `/root` on the KubMas control plane from the jump host: **`pscp "C:\Users\Administrator.DEMO\[repository location]\STRSW-ILT-UATWK\ Exercise 3\exercise3Task1.json" root@kubmas:/root`**
1-16	Open a new session on KubMas (Kubernetes Control Plane): 
1-17	At the Linux login prompt, provide the following credentials: • Login as: **`root`** • Password: **`Netapp1!`** The Linux CLI prompt and cursor appear.
1-18	Verify that you are in the present working directory for root and that the `tridentctl` tool and the back-end JSON are present.

1-19	Create the back end by using the `tridentctl` tool:
	`./tridentctl -n trident create backend -f exercise3Task1.json`
	
	Sample output:
	`[root@kubmas ~]# ./tridentctl -n trident create backend -f exercise3Task1.json`
	`+------------------+----------------+--------------------------------------+--------+---------+`
	`|  NAME  | STORAGE DRIVER |  UUID  | STATE  | VOLUMES |`
	`+------------------+----------------+--------------------------------------+--------+---------+`
	`| ontap-nas-backend | ontap-nas  | 9904a37b-3b91-4960-8ac6-5fe0ca38cf4c | online |  0 |`
	`+------------------+----------------+--------------------------------------+--------+---------+`

1-20	Review the `tridentctl` logs: **`./tridentctl -n trident logs | tail -n 10`**

Task 2: Create a Storage Class for a NAS Back End

In this task, you create a storage class that uses the NAS back end that you created in Task 1.

Step	Action
2-1	In your integrated development environment (IDE), open the `exercise3Task2.yaml` file.
2-2	Add the correct `backendType` to the parameters. This value is the `storageDriverName` from the back end JSON.
2-3	Create the storage class: **`kubectl create -f exercise3Task2.yaml`**

2-4	Verify that the storage class is created:
	`kubectl get sc nfs-basic`
	
	Sample output:
	`PS C:\Users\Administrator.DEMO\repo2\UsingAstra\Exercise 3> kubectl get sc nfs-basic`
	`NAME  PROVISIONER  RECLAIMPOLICY VOLUMEBINDINGMODE  ALLOWVOLUMEEXPANSION  AGE`
	`nfs-basic  csi.trident.netapp.io  Delete  Immediate  false  45s`

2-5	In the KubMas session, review the storage class by using the `tridentctl` tool:
	`./tridentctl -n trident get storageclass nfs-basic -o json`
	
	Sample output:
	`[root@kubmas ~]# ./tridentctl -n trident get storageclass basic -o json`
	`{`
	`  "items": [`
	`    {`
	`      "Config": {`
	`        "version": "1",`
	`        "name": "nfs-basic",`
	`        "attributes": {`
	`          "backendType": "ontap-nas"`
	`        },`
	`        "storagePools": null,`
	`        "additionalStoragePools": null`
	`      },`
	`      "storage": {`
	`        "ontap-nas-backend": [`
	`          "Cluster1_01_FC_1"`
	`        ]`
	`      }`
	`    }`
	`  ]`
	`}`

Task 3: Provision a Persistent Volume Claim with a NAS Back End

In this task, you create a persistent volume claim (PVC) for a volume that uses the storage class that you just created.

Step	Action
3-1	In your IDE, open the `exercise3Task3.yaml` file.
3-2	Update the `storageClassName` with the name of the storage class that you created in the previous task, and save the file.
3-3	Create the PVC to be used by a pod later: **`kubectl create -f exercise3Task3.yaml`**

3-4	Verify that the PVC is created:
	`kubectl get pvc nfs-basic`
	Sample output:
	`PS C:\Users\Administrator.DEMO\repo2\UsingAstra\Exercise 3> kubectl get pvc basic`
	`NAME STATUS VOLUME CAPACITY ACCESS MODES STORAGECLASS AGE`
	`nfs-basic Bound pvc-b876def3-8720-4738-a788-923013f9723e 1Gi RWO nfs-basic 21s`

3-5	Navigate to ONTAP System Manager, and see the new NetApp Astra Trident created volume: https://192.168.0.101/sysmgr/v4/storage/volumes

Task 4: Mount the Volumes in a Pod

You create an NGINX pod that uses the persistent volume (PV) and create a default webpage in the persistent volume. There is a challenge step that asks you to expose the pod with a NodePort service and view your custom webpage.

Step	Action
4-1	In your IDE, review the `exercise3Task4-1.yaml` file.
4-2	Set the `claimName` definition to the name of the PVC that you created in the previous task.
4-3	Save the file.
4-4	Create the pod to use the Astra Trident volume: **`kubectl create -f exercise3Task4-1.yaml`**
4-5	Verify that the pod is created: **`kubectl get pod nfs-pod`**
4-6	Connect to the pod: **`kubectl exec -it nfs-pod -- bash`**
4-7	Change directory to the Astra Trident persistent volume: **`# cd /usr/share/nginx/html`**
4-8	Create a file in this location: **`# echo '<html><body>Hello [your name] using NFS</body></html>' > index.html`**
4-9	Press **Ctrl-D** to exit the container's shell.
4-10	CHALLENGE STEP: Create a NodePort service and view the webpage with your custom message.

Task 5: Perform Back End Management by Using the `tridentctl` Tool

In this task, you investigate `tridentctl` commands and delete the pod that hosted the custom web page. You are then asked to re-create the pod and notice that the persistent volume reattached to the new pod and your custom page has persisted.

Step	Action
5-1	Make sure that you have a Secure Shell (SSH) session with your control plane node Kubmas.
5-2	Install JQuery and answer **Y** to complete the install: **`sudo yum install jq`**

5-3	Use the tridentctl tool to identify the storage class used:
	`./tridentctl get backend -n trident -o json | jq '[.items[] | {backend: .name, storageClasses: [.storage[].storageClasses]|unique}]'`
	Sample output:
	`[root@kubmas ~]# ./tridentctl get backend -n trident -o json | jq '[.items[] | {backend: .name, storageClasses: [.storage[].storageClasses]|unique}]'`
	`[`
	`  {`
	`  "backend": "ontap-nas-backend",`
	`  "storageClasses": [`
	`  [`
	`  "nfs-basic"`
	`  ]`
	`  ]`
	`  }`
	`]`

5-4	*i* You can delete and update the back end by using the `tridentctl` tool. For more information, see https://docs.netapp.com/us-en/trident/trident-use/backend_ops_tridentctl.html#create-a-backend
5-5	Delete the NFS-supported pod: **`kubectl delete pod nfs-pod`**
5-6	Investigate the logs and see whether the volume was deleted when the pod was deleted: **`./tridentctl logs -n trident | tail -n 20`**
5-7	How would you delete the volume automatically when you delete the pod?
5-8	Navigate to ONTAP System Manager, and notice that the Astra Trident created volume for the NFS-based PVC is still there: https://192.168.0.101/sysmgr/v4/storage/volumes
5-9	Re-create the pod to use the Astra Trident NFS-provided volume: **`kubectl create -f exercise3Task4-1.yaml`**
5-10	CHALLENGE STEP: If you created the NodePort service in the previous task, reconnect the webpage and see the webpage with your custom message.

Task 6: Perform Back End Management with the `kubectl` Tool

Previously, you created a back end with the `tridentctl` tool. In this task, you create a back end by using the new `TridentBackendConfig` custom resource (CR) with the credentials that are stored in a Kubernetes secret. For this task, you create a new SVM with iSCSI protocol configured. NOTE: SVMs allow multiple protocols. You could add iSCSI configuration to `svm0`. Creating another SVM was done to keep functionality separate. You also use the SVM administrator (vsadmin) and a separate management path for Astra Trident to communicate with the SVMs.

Step	Action
6-1	To create an SVM to use, open a browser to https://192.168.0.101 (which is your Cluster 1 management LIF's IP address).
6-2	Authenticate with your ONTAP cluster by providing the following credentials: • Login as: **`admin`** • Password: **`Netapp1!`**
6-3	Click **Sign In**.
6-4	Using the left pane, navigate Storage > Storage VMs.
6-5	Click **Add** to start the wizard to create an SVM.

6-6	Enter the following information, and then click **Save**.
	• Storage VM Name: **svm1**
	• Select the **iSCSI** tab
	• Access Protocol: Select **Enable iSCSI**
	• Under Network Interface 1:
	  ○ IP Address: **192.168.0.41**
	  ○ Subnet mask: **255.255.255.0**
	  ○ Broadcast domain: Default
	  ○ Select **Use the same subnet mask, gateway, and broadcast domain**
	• Under Network Interface 1:
	  ○ IP Address: **192.168.0.42**
	• Manage administrator account: Selected
	  ○ User name: **vsadmin**
	  ○ Password: **Netapp1!**
	  ○ Confirm Password: **Netapp1!**
	  ○ Select **Add a network interface for storage VM management**
	  ▪ IP Address: **192.168.0.40**
	  ▪ Subnet mask: **255.255.255.0**
	  ▪ Broadcast domain: Default

6-7	When you see svm1 in the list of Storage VMs, click the newly created **svm1** link.
6-8	When you see the Overview of the svm1, click the **Edit** button in the upper-right corner.

Step	Action
6-9	Select the Resource Allocation checkbox to prefer local tiers, and make sure that Cluster1_01_FC_1 is selected in the list of local tiers.
6-10	Click **Save**.
6-11	Update the `exercise3Task6-1.yaml` file with the details of the iSCSI SVM: • Username: **`vsadmin`** • Password: **`Netapp1!`** • Management LIF: **`192.168.0.40`** • Data LIF: **`192.168.0.41`** • SVM: **`svm1`**
6-12	Save the changes.
6-13	Create the secret and the back end by using the `kubectl` tool: **`kubectl create -f exercise3Task6-1.yaml`**
6-14	Within the Kubernetes IDE extension, navigate to `Clusters > kubernetes-admin@kubernetes > Custom Resources > tridentbackendconfigs > san-backend`. This back end is the one that you created.
6-15	Verify that the back-end configuration was created: **`kubectl -n trident get tbc -o wide`**
6-16	Get details on the back-end configuration that was created: **`kubectl -n trident describe tbc san-backend`**
6-17	Review, update the back end name in the YAML, and create the storage class in the `exercise3Task6-2.yaml` file: **`kubectl create -f exercise3Task6-2.yaml`**
6-18	Review, update the storage class name in the YAML, and create the PVC in the `exercise3Task6-3.yaml` file: **`kubectl create -f exercise3Task6-3.yaml`**
6-19	Navigate to ONTAP System Manager, and see the new Astra Trident created volume: https://192.168.0.101/sysmgr/v4/storage/volumes
6-20	Navigate to LUNs in ONTAP System Manager, and see the new Astra Trident created LUN: https://192.168.0.101/sysmgr/v4/storage/luns
6-21	Review, update the claim name in the YAML, and create the pod in the `exercise3Task6-4.yaml` file: **`kubectl create -f exercise3Task6-4.yaml`**
6-22	Verify that the pod is created: **`kubectl get pod san-pod`**

Step	Action
6-23	Connect to the pod: `kubectl exec -it san-pod -- bash`
6-24	Change directory to the Astra Trident persistent volume: `# cd /usr/share/nginx/html`
6-25	Create a file in this location: `# echo '<html><body>Hello [your name] using iSCSI</body></html>' > index.html`
6-26	Press **Ctrl-D** to exit the container's shell.
6-27	CHALLENGE STEP: Create a NodePort service and view the custom webpage.
6-28	CHALLENGE STEP: Delete the pod and verify that the LUN still exists.
6-29	CHALLENGE STEP: Re-create the pod and view the webpage with your custom message.
6-30	 Do not destroy any objects. You use them in the next exercise.

End of exercise

Module 4: Astra Trident Use Scenarios

Exercise 1: Working with Storage Managed by Astra Trident

In this exercise, you explore and manage point-in-time copies of a persistent volume (PV). You also expand a persistent volume and import a volume that is not controlled by NetApp Astra Trident as a persistent volume that is managed by Astra Trident.

Objectives

This exercise focuses on enabling you to do the following:

- Manage Snapshot copies
- Expand volumes
- Import volumes
- Deploy Astra Trident storage in multiple zones

Exercise Equipment

In this exercise, you use the following systems.

System	Host Name	IP Addresses	User Name	Password
Windows Server	Jumphost	192.168.0.5	DEMO\Administrator	Netapp1!
Kubernetes Control Plane	kubmas	192.168.0.61	root (case sensitive)	Netapp1!

Task 1: Managing Snapshot Copies

In this task, you configure the required custom resource definitions (CRDs) to use point-in-time copies snapshots of volumes in Kubernetes. Then you create the snapshot controller.

Step	Action
1-1	In your integrated development environment (IDE), make sure that you are in the `Exercise 4` folder of the course contents.
1-2	Create the `volumeshapshotclass` custom resource definition (CRD): **`kubectl create -f exercise4Task1-1.yaml`**
1-3	Create the `volumeshapshotcontents` CRD: **`kubectl create -f exercise4Task1-2.yaml`**
1-4	Create the `volumeshapshot` CRD: **`kubectl create -f exercise4Task1-3.yaml`**
1-5	Create the service account, roles, and role bindings for the `snapshot-controller` in the `kube-system` namespace: **`kubectl create -f exercise4Task1-4.yaml`**
1-6	Create the `snapshot-controller` stateful set in the `kube-system` namespace: **`kubectl create -f exercise4Task1-5.yaml`**

Step	Action
1-7	Create a `volumesnapshotclass` custom resource (CR) that points to the Astra Trident Container Storage Interface (CSI) driver: **`kubectl create -f exercise4Task1-6.yaml`**  NOTE: `deletionPolicy` can be `Delete` or `Retain`. When set to `Retain`, the underlying physical snapshot on the storage cluster is retained even when the VolumeSnapshot object is deleted.

1-8	Create a snapshot definition in an `exercise4Task1-7.yaml` file, and then save the file.
	• apiVersion: **`snapshot.storage.k8s.io/v1`**
	• kind: **`VolumeSnapshot`**
	• metadata name: **`nfs-snap`**
	• Under spec:
	  ○ volumeSnapshotClassName: [name of the snapshotclass created previously]
	  ○ source persistentVolumeClaimName: [name of the NFS-backed persistent volume claim (PVC) created in the previous exercise]

1-9	Now, create a snapshot of the NFS-back end PVC that you created in the previous exercise: **`kubectl create -f exercise4Task1-7.yaml`**

1-10	Verify that the snapshot was created:
	`kubectl get volumesnapshots`
	
	Sample output:
	`PS C:\Users\Administrator.DEMO\repo2\UsingAstra\Exercise 4> kubectl get volumesnapshots`
	`NAME READYTOUSE SOURCEPVC SOURCESNAPSHOTCONTENT RESTORESIZE SNAPSHOTCLASS SNAPSHOTCONTENT`
	`nfs-snap true nfs-basic 400Ki trident-snapshotclass snapcontent-7789e1eb`

Step	Action

1-11	Describe the snapshot, and notice the Ready to Use parameter:
	`kubectl describe volumesnapshots nfs-snap`
	
	Sample output:
	`PS C:\Users\Administrator.DEMO\repo2\UsingAstra\Exercise 4> kubectl describe volumesnapshots nfs-snap`
	`Name: nfs-snap`
	`Namespace: trident`
	`Labels: <none>`
	`Annotations: <none>`
	`API Version: snapshot.storage.k8s.io/v1`
	`Kind: VolumeSnapshot`
	`Metadata:`
	`Creation Timestamp: 2022-02-02T19:10:56Z`
	`Finalizers:`
	`snapshot.storage.kubernetes.io/volumesnapshot-as-source-protection`
	`snapshot.storage.kubernetes.io/volumesnapshot-bound-protection`
	`Generation: 1`
	`Managed Fields:`
	`API Version: snapshot.storage.k8s.io/v1`
	`Fields Type: FieldsV1`
	`fieldsV1:`
	`f:spec:`
	`.:`
	`f:source:`
	`.:`
	`f:persistentVolumeClaimName:`
	`f:volumeSnapshotClassName:`
	`Manager: kubectl-create`
	`Operation: Update`
	`Time: 2022-02-02T19:10:56Z`
	`API Version: snapshot.storage.k8s.io/v1`
	`Fields Type: FieldsV1`
	`fieldsV1:`
	`f:metadata:`
	`f:finalizers:`
	`.:`
	`v:"snapshot.storage.kubernetes.io/volumesnapshot-as-source-protection":`
	`v:"snapshot.storage.kubernetes.io/volumesnapshot-bound-protection":`
	`Manager: snapshot-controller`
	`Operation: Update`
	`Time: 2022-02-02T19:10:56Z`
	`API Version: snapshot.storage.k8s.io/v1`
	`Fields Type: FieldsV1`
	`fieldsV1:`
	`f:status:`
	`.:`
	`f:boundVolumeSnapshotContentName:`
	`f:creationTime:`
	`f:readyToUse:`
	`f:restoreSize:`
	`Manager: snapshot-controller`
	`Operation: Update`
	`Subresource: status`
	`Time: 2022-02-02T19:10:57Z`
	`Resource Version: 1222882`
	`UID: 7789e1eb-0b7b-4d70-8d56-a69cd99cfaa7`
	`Spec:`
	`Source:`
	`Persistent Volume Claim Name: nfs-basic`
	`Volume Snapshot Class Name: trident-snapshotclass`
	`Status:`
	`Bound Volume Snapshot Content Name: snapcontent-7789e1eb-0b7b-4d70-8d56-a69cd99cfaa7`
	`Creation Time: 2022-02-02T19:11:07Z`
	`Ready To Use: true`
	`Restore Size: 400Ki`
	`Events:`
	`Type Reason Age From Message`
	`---- ------ ---- ---- -------`
	`Normal CreatingSnapshot 48s snapshot-controller Waiting for a snapshot trident/nfs-snap to be created by the CSI driver.`
	`Normal SnapshotCreated 47s snapshot-controller Snapshot trident/nfs-snap was successfully created by the CSI driver.`
	`Normal SnapshotReady 47s snapshot-controller Snapshot trident/nfs-snap is ready to use.`

Step	Action

1-12	Create a PVC in an `exercise4Task1-8.yaml` file with the following defintions and then execute the YAML:
	• Metadata name: **`nfs-snap`**
	• Spec:
	  ○ accessModes: **`ReadWriteOnce`**
	  ○ resource requests storage: **`1Gi`**
	  ○ storageClassName: **`nfs-basic`**
	  ○ dataSource:
	  ▪ name: **`nfs-snap`**
	  ▪ kind: **`VolumeSnapshot`**
	  ▪ apiGroup: **`snapshot.storage.k8s.io`**

1-13	Verify that the PVC was created:
	`kubectl get pvc`
	
	Sample output:
	`PS C:\Users\Administrator.DEMO\repo2\UsingAstra\Exercise 4> kubectl get pvc`
	`NAME  STATUS  VOLUME  CAPACITY  ACCESS MODES  STORAGECLASS  AGE`
	`nfs-basic  Bound  pvc-b876def3-8720-4738-a788-923013f9723e  1Gi  RWO  nfs-basic  21h`
	`san-basic  Bound  pvc-a462584b-6519-4a09-8ec3-595976f3274a  1Gi  RWO  san-basic.  18h`
	`nfs-snap  Pending  nfs-basic  9s`

1-14	Describe the PVC that was created: **`kubectl describe pvc [name of pvc]`** Sample abbreviated output: `Events:` `Type  Reason  Age  From  Message` `----  ------  ----  ----  -------` `Normal  Provisioning  26s (x3 over 50s)  csi.trident.netapp.io_trident-csi-b75cd9bc6-qvx98_6dee966b-2ae0-4bee-9fda-70c05254faca  External provisioner is provisioning volume for claim "trident/nfs-snap"` `Normal ProvisioningFailed 26s (x3 over 50s) csi.trident.netapp.io failed to create cloned volume pvc-4737e591-c2f5-49c1-96d9-842a5e9f783c on backend exercise3Task1: error creating clone: API status: failed, Reason: You do not have a valid license for "FlexClone". Reason: Package "FlexClone" is not licensed in the cluster., Code: 13001` `Warning ProvisioningFailed 26s (x3 over 50s) csi.trident.netapp.io_trident-csi-b75cd9bc6-qvx98_6dee966b-2ae0-4bee-9fda-70c05254faca failed to provision volume with StorageClass "basic": rpc error: code = Unknown desc = failed to create cloned volume pvc-4737e591-c2f5-49c1-96d9-842a5e9f783c on backend exercise3Task1: error creating clone: API status: failed, Reason: You do not have a valid license for "FlexClone". Reason: Package "FlexClone" is not licensed in the cluster., Code: 13001` `Normal ExternalProvisioning 17s (x4 over 50s) persistentvolume-controller`
1-15	Answer the following question: Why has the PVC not been provisioned?
1-16	Add the FlexClone license to your NetApp ONTAP cluster. The license is provided by your instructor.

Step	Action
1-17	Verify that the PVC is now provisioned (you might have to wait up to a minute): **`kubectl describe pvc [name of pvc]`** Sample abbreviated output: `Events:` `  Type  Reason  Age  From  Message` `  ----  ------  ----  ----  -------` ` Normal ProvisioningFailed 52s (x12 over 5m47s) csi.trident.netapp.io failed to create cloned volume pvc-4737e591-c2f5-49c1-96d9-842a5e9f783c on backend exercise3Task1: error creating clone: API status: failed, Reason: You do not have a valid license for "FlexClone". Reason: Package "FlexClone" is not licensed`  `in the cluster., Code: 13001`  ` Warning ProvisioningFailed 52s (x12 over 5m47s) csi.trident.netapp.io_trident-csi-b75cd9bc6-qvx98_6dee966b-2ae0-4bee-9fda-70c05254faca failed to provision volume with StorageClass "basic": rpc error: code = Unknown desc = failed to create cloned volume pvc-4737e591-c2f5-49c1-96d9-842a5e9f783c on backend exercise3Task1: error creating clone: API status: failed, Reason: You do not have a valid license for "FlexClone". Reason: Package "FlexClone" is not licensed in the cluster., Code: 13001`  ` Normal ExternalProvisioning 29s (x23 over 5m47s) persistentvolume-controller waiting for a volume to be created, either by external provisioner "csi.trident.netapp.io" or manually created by system administrator`  ` Normal Provisioning 22s (x13 over 5m47s) csi.trident.netapp.io_trident-csi-b75cd9bc6-qvx98_6dee966b-2ae0-4bee-9fda-70c05254faca External provisioner is provisioning volume for claim "trident/nfs-snap"`  ` Normal ProvisioningSuccess 19s csi.trident.netapp.io provisioned a volume`  ` Normal ProvisioningSucceeded 19s csi.trident.netapp.io_trident-csi-b75cd9bc6-qvx98_6dee966b-2ae0-4bee-9fda-70c05254faca Successfully provisioned volume pvc-4737e591-c2f5-49c1-96d9-842a5e9f783c`
1-18	Create a pod similar to `exercise3Task4-1.yaml` entitled **`exercise4Task1-9.yaml`** that has the following attributes: • Metadata name: **`nfs-snap-pod`** • Metadata label: app: **`nfs-snap-web`** • Spec volumes persistentVolumeClaim claimName: **`nfs-snap`**
1-19	CHALLENGE STEP: Create a NodePort and view the custom webpage. If desired, you could also change the custom webpage to make it unique from the previous exercise's NFS-back end pod.
1-20	CHALLENGE STEP: Repeat Steps 1-8 through 1-19 with the SAN-backed PVC and pod. However, there is no need to add the FlexClone license a second time. Solution files can be found in `exercise4Task1-11.yaml` through `exercise4Task1-14.yaml` files.

Task 2: Expanding Volumes

You edit the storage class definitions for the NFS-backed storage class to enable volume expansion and then expand the NFS-backed persistent volume (PV). There is a challenge step to do the same for the SAN-backed storage class and persistent volume.

Step	Action
2-1	Edit the `exercise4Task2-1.yaml` file, add the **`allowVolumeExpansion: true`** definition, and save the file.
2-2	Apply the updated definition: **`kubectl apply -f exercise4Task2-1.yaml`** Alternatively, use **`kubectl edit sc nfs-basic.`**

2-3	Identify the PVC used by the NFS-backed pod:
	`kubectl get pvc`
	
	Sample output:
	`PS C:\Users\Administrator.DEMO\repo2\UsingAstra\Exercise 4> kubectl get pvc`
	`NAME STATUS VOLUME CAPACITY ACCESS MODES STORAGECLASS AGE`
	`nfs-basic Bound pvc-b876def3-8720-4738-a788-923013f9723e 1Gi RWO basic 21h`
	`san-basic Bound pvc-a462584b-6519-4a09-8ec3-595976f3274a 1Gi RWO basic-san 18h`
	`nfs-snap Bound pvc-4737e591-c2f5-49c1-96d9-842a5e9f783c 1Gi RWO basic 156m`
	`san-snap Bound pvc-8e9e4573-52a1-439b-9b3a-04f0f1e2deea 1Gi RWO basic-san 128m`

2-4	Describe the PVC used by the NFS-backed pod:
	`kubectl describe pvc nfs-basic`
	
	Sample output:
	`Name: nfs-basic`
	`Namespace: trident`
	`StorageClass: nfs-basic`
	`Status: Bound`
	`Volume: pvc-b876def3-8720-4738-a788-923013f9723e`
	`Labels: <none>`
	`Annotations: pv.kubernetes.io/bind-completed: yes`
	`pv.kubernetes.io/bound-by-controller: yes`
	`volume.beta.kubernetes.io/storage-provisioner: csi.trident.netapp.io`
	`volume.kubernetes.io/storage-provisioner: csi.trident.netapp.io`
	`Finalizers: [kubernetes.io/pvc-protection]`
	`Capacity: 1Gi`
	`Access Modes: RWO`
	`VolumeMode: Filesystem`
	`Used By: nfs-pod`
	`Events: <none>`

2-5	Identify the PV used by the NFS-backed pod:
	`kubectl get pv`
	
	Sample output:
	`PS C:\Users\Administrator.DEMO\repo2\UsingAstra\Exercise 4> kubectl get pv`
	`NAME CAPACITY ACCESS MODES RECLAIM POLICY STATUS CLAIM STORAGECLASS REASON AGE`
	`pvc-4737e591-c2f5-49c1-96d9-842a5e9f783c 1Gi RWO Delete Bound trident/nfs-snap nfs-basic 152m`
	`pvc-8e9e4573-52a1-439b-9b3a-04f0f1e2deea 1Gi RWO Delete Bound trident/san-snap san-basic 126m`
	`pvc-a462584b-6519-4a09-8ec3-595976f3274a 1Gi RWO Delete Bound trident/basic-san san-basic 20h`
	`pvc-b876def3-8720-4738-a788-923013f9723e 1Gi RWO Delete Bound trident/basic nfs-basic 24h`

Step	Action

2-6	Describe the PV used by the NFS-backed pod:
	`kubectl describe pv [name of the pv that is associated with nfs-basic claim]`
	
	Sample output:
	`PS C:\Users\Administrator.DEMO\repo2\UsingAstra\Exercise 4> kubectl describe pv pvc-b876def3-8720-4738-a788-923013f9723e`
	`Name: pvc-b876def3-8720-4738-a788-923013f9723e`
	`Labels: <none>`
	`Annotations: pv.kubernetes.io/provisioned-by: csi.trident.netapp.io`
	`Finalizers: [kubernetes.io/pv-protection external-attacher/csi-trident-netapp-io]`
	`StorageClass: nfs-basic`
	`Status: Bound`
	`Claim: trident/nfs-basic`
	`Reclaim Policy: Delete`
	`Access Modes: RWO`
	`VolumeMode: Filesystem`
	`Capacity: 1Gi`
	`Node Affinity: <none>`
	`Message:`
	`Source:`
	`Type: CSI (a Container Storage Interface (CSI) volume source)`
	`Driver: csi.trident.netapp.io`
	`FSType:`
	`VolumeHandle: pvc-b876def3-8720-4738-a788-923013f9723e`
	`ReadOnly: false`
	`VolumeAttributes: backendUUID=9904a37b-3b91-4960-8ac6-5fe0ca38cf4c`
	`internalName=trident_pvc_b876def3_8720_4738_a788_923013f9723e`
	`name=pvc-b876def3-8720-4738-a788-923013f9723e`
	`protocol=file`
	`storage.kubernetes.io/csiProvisionerIdentity=1643697171296-8081-csi.trident.netapp.io`
	`Events: <none>`

2-7	Edit the NFS-backed PVC in the `exercise4Task2-2.yaml` file, change the storage definition from `1Gi` to **`2Gi`**, and save the file.
2-8	Apply the updated definition of the PVC: **`kubectl apply -f exercise4Task2-2.yaml`**

Step	Action

2-9	Verify that the expanded volume of the PVC is used by the NFS-backed pod:
	`kubectl describe pvc nfs-basic`
	
	Sample output:
	`Name: nfs-basic`
	`Namespace: trident`
	`StorageClass: nfs-basic`
	`Status: Bound`
	`Volume: pvc-b876def3-8720-4738-a788-923013f9723e`
	`Labels: <none>`
	`Annotations: pv.kubernetes.io/bind-completed: yes`
	`pv.kubernetes.io/bound-by-controller: yes`
	`volume.beta.kubernetes.io/storage-provisioner: csi.trident.netapp.io`
	`volume.kubernetes.io/storage-provisioner: csi.trident.netapp.io`
	`Finalizers: [kubernetes.io/pvc-protection]`
	`Capacity: 2Gi`
	`Access Modes: RWO`
	`VolumeMode: Filesystem`
	`Used By: nfs-pod`
	`Events:`
	`Type Reason Age From Message`
	`---- ------ ---- ---- -------`
	`Warning ExternalExpanding 4m36s volume_expand Ignoring the PVC: didn't find a plugin capable of expanding the volume; waiting for an external controller to process this PVC.`
	`Normal Resizing 4m36s external-resizer csi.trident.netapp.io External resizer is resizing volume pvc-b876def3-8720-4738-a788-923013f9723e`
	`Normal VolumeResizeSuccessful 4m36s external-resizer csi.trident.netapp.io Resize volume succeeded`

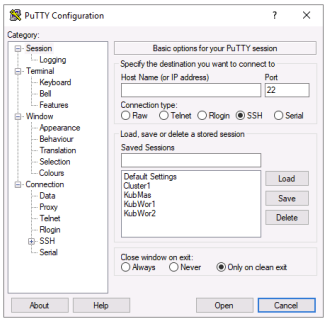
2-10	Describe the PV used by the NFS-backed pod:
	`kubectl describe pv [name of the pv that is associated with nfs-basic claim]`
	
	Sample output:
	`Name: pvc-b876def3-8720-4738-a788-923013f9723e`
	`Labels: <none>`
	`Annotations: pv.kubernetes.io/provisioned-by: csi.trident.netapp.io`
	`Finalizers: [kubernetes.io/pv-protection external-attacher/csi-trident-netapp-io]`
	`StorageClass: nfs-basic`
	`Status: Bound`
	`Claim: trident/nfs-basic`
	`Reclaim Policy: Delete`
	`Access Modes: RWO`
	`VolumeMode: Filesystem`
	`Capacity: 2Gi`
	`Node Affinity: <none>`
	`Message:`
	`Source:`
	`Type: CSI (a Container Storage Interface (CSI) volume source)`
	`Driver: csi.trident.netapp.io`
	`FSType:`
	`VolumeHandle: pvc-b876def3-8720-4738-a788-923013f9723e`
	`ReadOnly: false`
	`VolumeAttributes: backendUUID=9904a37b-3b91-4960-8ac6-5fe0ca38cf4c`
	`internalName=trident_pvc_b876def3_8720_4738_a788_923013f9723e`
	`name=pvc-b876def3-8720-4738-a788-923013f9723e`
	`protocol=file`
	`storage.kubernetes.io/csiProvisionerIdentity=1643697171296-8081-csi.trident.netapp.io`
	`Events: <none>`

2-11	CHALLENGE STEP: Repeat Steps 1-8 through 1-18 with the SAN-backed PVC and pod. However, there is no need to add the FlexClone license a second time. Solution files can be found in `exercise4Task2-3.yaml` through `exercise4Task2-4.yaml` files.

Step	Action
2-12	There are two scenarios when resizing an iSCSI PV: • If the PV is attached to a pod, Astra Trident expands the volume on the storage back end, rescans the device, and resizes the file system. • When attempting to resize an unattached PV, Astra Trident expands the volume on the storage back end. After the PVC is bound to a pod, Trident rescans the device and resizes the file system. Kubernetes then updates the PVC size after the expand operation has successfully finished.

Task 3: Importing Volumes

In this task, you create a FlexClone volume of the NFS volume that you created in the previous exercise. You then import the cloned volume as a persistent volume and associate it with a persistent volume claim. There is a challenge step to complete this activity with the LUN that you created in the previous exercise.

Step	Action
3-1	Log in to ONTAP System Manager: https://192.168.0.101.
3-2	Authenticate with your ONTAP cluster by providing the following credentials: • Login as: **`admin`** • Password: **`Netapp1!`**
3-3	Navigate to **Storage** > **Volumes**.
3-4	Select one of the Astra Trident volumes managed by storage VM svm0. These volumes are the NFS volumes.
3-5	Click the three vertical dots next to the volume's name, and select **Clone** from the menu. 
3-6	In the Clone Volume dialog box, provide the following details, and then click **Clone**. • Name: **import_vol** • Enable thin provisioning: Selected • Clone Parent Snapshot Copy: **Add a Snapshot Copy**
3-7	Verify that the new volume was created. If desired, you could also split the clone.
3-8	Back in your IDE, open the `exercise4Task3-1.yaml` file, and review it, and make sure that the storage size is equal to the cloned volume that you created in the previous steps.
3-9	Use PuTTY Secure Copy tool to transfer the YAML file to `/root` on the KubMas control plane from the jump host: **`pscp "C:\Users\Administrator.DEMO\[repository location]\STRSW-ILT-UATWK\Exercise 4\exercise4Task3-1.yaml" root@kubmas:/root`**
3-10	Open a new session on KubMas (Kubernetes Control Plane): 

Step	Action
3-11	At the Linux login prompt, provide the following credentials: • Login as: **`root`** • Password: **`Netapp1!`** The Linux CLI prompt and cursor appear.
3-12	Verify that you are in the present working directory for root and that the `tridentctl` tool is present.
3-13	Identify the existing NFS back end by using the `tridentctl` tool and using the `ontap-nas` driver: **`./tridentctl -n trident get backends`**

3-14	Create the back end by using the `tridentctl` tool:
	`./tridentctl -n trident import volume ontap-nas-backend import_vol -f exercise4Task3-1.yaml`
	Sample output:
	`[root@kubmas ~]# ./tridentctl -n trident import volume ontap-nas-backend import_vol -f exercise4Task3-1.yaml`
	`+----------+----------+----------+----------+----------+----------+----------+`
	`| NAME | SIZE | STORAGE CLASS | PROTOCOL | BACKEND UUID | STATE | MANAGED |`
	`+----------+----------+----------+----------+----------+----------+----------+`
	`| pvc-a786e345-efa2-4ad0-95b9-7f5849fffee6 | 1.0 GiB | nfs-basic | file | 9904a37b-3b91-4960-8ac6-5fe0ca38cf4c | online | true |`
	`+----------+----------+----------+----------+----------+----------+----------+`

3-15	Identify the PVC that was created for the imported volume:
	`kubectl get pvc`
	Sample output:
	`PS C:\Users\Administrator.DEMO\repo2\UsingAstra\Exercise 4> kubectl get pvc`
	`NAME STATUS VOLUME CAPACITY ACCESS MODES STORAGECLASS AGE`
	`nfs-basic Bound pvc-b876def3-8720-4738-a788-923013f9723e 2Gi RWO nfs-basic 25h`
	`san-basic Bound pvc-a462584b-6519-4a09-8ec3-595976f3274a 2Gi RWO san-basic 21h`
	`nfs-import Bound pvc-a786e345-efa2-4ad0-95b9-7f5849fffee6 1Gi RWO nfs-basic 55s`
	`nfs-snap Bound pvc-4737e591-c2f5-49c1-96d9-842a5e9f783c 1Gi RWO nfs-basic 3h34m`
	`san-snap Bound pvc-8e9e4573-52a1-439b-9b3a-04f0f1e2deea 1Gi RWO san-basic 3h5m`

Step	Action

3-16	Describe the PV that is used by the NFS-backed pod:
	`kubectl describe pvc nfs-import`
	
	Sample output:
	`PS C:\Users\Administrator.DEMO\repo2\UsingAstra\Exercise 4> kubectl describe pvc nfs-import`
	`Name: nfs-import`
	`Namespace: trident`
	`StorageClass: nfs-basic`
	`Status: Bound`
	`Volume: pvc-a786e345-efa2-4ad0-95b9-7f5849fffee6`
	`Labels: <none>`
	`Annotations: pv.kubernetes.io/bind-completed: yes`
	`pv.kubernetes.io/bound-by-controller: yes`
	`trident.netapp.io/importBackendUUID: 9904a37b-3b91-4960-8ac6-5fe0ca38cf4c`
	`trident.netapp.io/importOriginalName: import_vol`
	`trident.netapp.io/notManaged: false`
	`volume.beta.kubernetes.io/storage-provisioner: csi.trident.netapp.io`
	`volume.kubernetes.io/storage-provisioner: csi.trident.netapp.io`
	`Finalizers: [kubernetes.io/pvc-protection]`
	`Capacity: 1Gi`
	`Access Modes: RWO`
	`VolumeMode: Filesystem`
	`Used By: <none>`
	`Events:`
	`Type Reason Age From Message`
	`---- ------ ---- ---- -------`
	`Normal Provisioning 2m36s csi.trident.netapp.io_trident-csi-b75cd9bc6-qvx98_6dee966b-2ae0-4bee-9fda-70c05254faca External provisioner is provisioning volume for claim "trident/nfs-import"`
	`Normal ExternalProvisioning 2m35s (x3 over 2m36s) persistentvolume-controller waiting for a volume to be created, either by external provisioner "csi.trident.netapp.io" or manually created by system administrator`
	`Normal ProvisioningSuccess 2m26s csi.trident.netapp.io provisioned a volume`
	`Normal ProvisioningSucceeded 2m26s csi.trident.netapp.io_trident-csi-b75cd9bc6-qvx98_6dee966b-2ae0-4bee-9fda-70c05254faca Successfully provisioned volume pvc-a786e345-efa2-4ad0-95b9-7f5849fffee6`

3-17	Identify the PV that is used by the nfs-import claim:
	`kubectl get pv`
	
	Sample output:
	`PS C:\Users\Administrator.DEMO\repo2\UsingAstra\Exercise 4> kubectl get pv`
	`NAME CAPACITY ACCESS MODES RECLAIM POLICY STATUS CLAIM STORAGECLASS REASON AGE`
	`pvc-4737e591-c2f5-49c1-96d9-842a5e9f783c 1Gi RWO Delete Bound trident/nfs-snap nfs-basic 3h30m`
	`pvc-8e9e4573-52a1-439b-9b3a-04f0f1e2deea 1Gi RWO Delete Bound trident/san-snap san-basic 3h4m`
	`pvc-a462584b-6519-4a09-8ec3-595976f3274a 2Gi RWO Delete Bound trident/basic-san san-basic 21h`
	`pvc-a786e345-efa2-4ad0-95b9-7f5849fffee6 1Gi RWO Delete Bound trident/nfs-import nfs-basic 115s`
	`pvc-b876def3-8720-4738-a788-923013f9723e 2Gi RWO Delete Bound trident/basic nfs-basic 25h`

Step	Action

3-18	Describe the PV that is used by the nfs-import claim:
	kubectl describe pv [name of the pv that is associated with nfs-import claim]
	
	Sample output:
	`PS C:\Users\Administrator.DEMO\repo2\UsingAstra\Exercise 4> kubectl describe pv pvc-a786e345-efa2-4ad0-95b9-7f5849fffee6`
	`Name: pvc-a786e345-efa2-4ad0-95b9-7f5849fffee6`
	`Labels: <none>`
	`Annotations: pv.kubernetes.io/provisioned-by: csi.trident.netapp.io`
	`Finalizers: [kubernetes.io/pv-protection]`
	`StorageClass: nfs-basic`
	`Status: Bound`
	`Claim: trident/nfs-import`
	`Reclaim Policy: Delete`
	`Access Modes: RWO`
	`VolumeMode: Filesystem`
	`Capacity: 1Gi`
	`Node Affinity: <none>`
	`Message:`
	`Source:`
	`Type: CSI (a Container Storage Interface (CSI) volume source)`
	`Driver: csi.trident.netapp.io`
	`FSType:`
	`VolumeHandle: pvc-a786e345-efa2-4ad0-95b9-7f5849fffee6`
	`ReadOnly: false`
	`VolumeAttributes: backendUUID=9904a37b-3b91-4960-8ac6-5fe0ca38cf4c`
	`internalName=trident_pvc_a786e345_efa2_4ad0_95b9_7f5849fffee6`
	`name=pvc-a786e345-efa2-4ad0-95b9-7f5849fffee6`
	`protocol=file`
	`storage.kubernetes.io/csiProvisionerIdentity=1643828813775-8081-csi.trident.netapp.io`
	`Events: <none>`

3-19	Create a pod to use the PVC for the imported volume: **kubectl create -f exercise4Task3-2.yaml**
3-20	Create a service to use the pod for the imported volume: **kubectl create -f exercise4Task3-3.yaml**
3-21	Connect to the pod: **kubectl exec -it nfs-pod -- bash**
3-22	Change directory to the Astra Trident persistent volume: **# cd /usr/share/nginx/html**
3-23	Attempt to edit the index.html file in this location: **# echo '<html><body>Hello [your name] using an imported volume</body></html>' > index.html**
3-24	Enter **Ctrl-D** to break out of the `kubectl exec` operation.
3-25	Open a web browser to the NodePort service and verify you see the edite index.html page: **http://[the ip address of one your Kubernetes worker nodes]:[the port of the service created in step 3-20]**
3-26	CHALLENGE STEP: Complete this task by using a LUN-backed persistent volume. The solutions files are found as `exercise4Task3-4.yaml` - `exercise4Task3-6.yaml`.

Task 4: Deploying Astra Trident Storage in Multiple Zones

In this task, you assume that you need to segment nodes of your Kubernetes cluster into subgroups. You create two different back ends each to support one of the Kubernetes Worker nodes. You use different storage in each subgroup (or zone). Zone 1 uses the NFS-based storage, and Zone 2 uses iSCSI-backed storage. To complete this task, you should have appropriately labeled the Worker nodes in Module 2, Task 1, Step 1-4 and 1-5. If you have not done this step, you cannot complete this task.

Step	Action
4-1	Create the back end for Zone 1 (NAS storage): **`kubectl apply -f exercise4Task4-1.yaml`**
4-2	Create the back end for Zone 2 (SAN storage): **`kubectl apply -f exercise4Task4-2.yaml`**
4-3	 The secret should have been previous created and is commented out. If you have the secret does not exist, please uncomment this section.
4-4	Create the storage class for Zone 1 (NAS storage): **`kubectl apply -f exercise4Task4-3.yaml`**
4-5	Create the storage class for Zone 2 (SAN storage): **`kubectl apply -f exercise4Task4-4.yaml`**
4-6	 `volumeBindingMode` is set to `WaitForFirstConsumer` (default value: `Immediate`), which means that the PVC is not created until referenced in a pod.
4-7	Create a new namespace called `topology` and a PVC for Zone 1 (NAS storage): **`kubectl apply -f exercise4Task4-5.yaml`**
4-8	Create a PVC for Zone 2 (SAN storage): **`kubectl apply -f exercise4Task4-6.yaml`**
4-9	Investigate the PVCs, and answer the following question: Are there any matching PVs created yet?
4-10	Describe one of the PVCs and review the events. All the PVC, PV, and later pods are added to the `topology` namespace.
4-11	Create a pod for Zone 1 to use NAS storage: **`kubectl apply -f exercise4Task4-7.yaml`**
4-12	Create a pod for Zone 2 to use SAN storage: **`kubectl apply -f exercise4Task4-8.yaml`**
4-13	Investigate the PVCs, and answer the following question: Are there any matching PVs created yet?

Step	Action
4-14	NOTE: If you want to do Appendix 3, you might skip this step. Clean up: `kubectl delete namespace topology`
4-15	CHALLENGE STEP: Create another NAS storage pool and use a single storage class for both Zone 1 and Zone 2. For an example of this scenario, see https://github.com/YvosOnTheHub/LabNetApp/tree/master/Kubernetes_v4/Scenarios/Scenario15.

End of exercise

Appendix 1: Kubernetes-Related Certifications

There is no exercise for this appendix.

Appendix 2: An Introduction to Operators

There is no exercise for this appendix.

Appendix 3: Astra Trident Monitoring

Exercise 1: Monitoring Astra Trident

In this exercise, you install Prometheus and Grafana in your Kubernetes cluster and configure a Grafana dashboard for the Kubernetes content. You create a sample app, collect telemetry data from the sample app, and configure NetApp Astra Trident as a Prometheus target.

Objectives

This exercise focuses on enabling you to do the following:

- Host Prometheus and supporting applications in Kubernetes
- Configure a sample app as a Prometheus target
- Configure Astra Trident as a Prometheus target

Exercise Equipment

In this exercise, you use the following system.

System	Host Name	IP Addresses	User Name	Password
Windows Server	Jumphost	192.168.0.5	DEMO\Administrator	Netapp1!

Task 1: Host Prometheus and Supporting Applications in Kubernetes

In this task, you configure Prometheus and its supporting applications to run in Kubernetes. For more information about this procedure, see https://github.com/prometheus-operator/kube-prometheus.

Step	Action
1-1	Within a terminal window in your integrated development environment (IDE), change directory to the `Extras/monitoring` folder within the course content repository.
1-2	Review the subfolder `1-setup`, and from the terminal, apply all the custom resource definitions (CRDs) to your Kubernetes cluster: **`kubectl create -f 1-setup/`**
1-3	 You should see output that seven CRDs were applied and the `monitoring` namespace was created. These definition files were taken from https://github.com/prometheus-operator/kube-prometheus/tree/main/manifests/setup.
1-4	Review the subfolder `2-manifests`, and from the terminal, apply all the CRDs to your Kubernetes cluster: **`kubectl create -f 2-manifests/`**
1-5	 You should see a long output showing many objects that were created. These definition files were taken from https://github.com/prometheus-operator/kube-prometheus/tree/main/manifests.
1-6	View the pods that are being created: **`Kubectl -n monitoring get pods`**

Step	Action
1-7	The following pods are created: • Prometheus Operator (one pod): Is used to manage all other Prometheus pods and deploys two Prometheus instances • Prometheus k8s (two pods, tuned to one pod for this exercise): Is used to aggregate metrics • Alertmanager (three pods, tuned to one pod for this exercise): Is used to communicate Prometheus data with external services • Kube State Metrics (one pod): Is used to scrape telemetry data from your Kubernetes control plane services • Node Explorer (three pods, one for each node): Is used to scrape telemetry data from your Kubernetes nodes • Grafana (one pod): Is used to provide a front-end UI for the Prometheus data • Blackbox Exporter (one pod): Enables blackbox probing of endpoints over HTTP, HTTPS, DNS, TCP, and Internet Control Message Protocol (ICMP). • Prometheus Adapter (two pods): Contains an implementation of the Kubernetes resource metrics, custom metrics, and external metrics APIs to use with Horizontal Pod Autoscaler
1-8	Review and run the `appendix3Task1-1.yaml` file, which creates a NodePort service for Prometheus: **`kubectl create -f appendix3Task1-1.yaml`**
1-9	Open a web browser tab to **`http://[the ip address of one your Kubernetes worker nodes]:30900`**
1-10	Explore the default Prometheus interface.
1-11	Click the **Alerts** menu, and see how all the preconfigured Alerts are created. Green means that the condition is normal, and red indicates that the condition is overcommitted.
1-12	Prometheus keeps all its data in memory. You can configure Prometheus for remote-write to persist the data to some external service for management and archive.
1-13	List all the service monitors available: **`kubectl -n monitoring get servicemonitors`**
1-14	In the Prometheus UI, under the Status > Targets menu, make sure that you see a list of service monitors that have been already configured for you.
1-15	To explore how these Service Monitors work, describe them: **`kubectl -n monitoring describe servicemonitors kube-apiserver`**

Step	Action
1-16	Notice a section at the end of the describe output that has a selector and matches the labels `component:apiserver` and `provider:kubernetes`. If you investigate the API Server's service that is running in the `default` namespace called `kubernetes`, that service's labels are `component:apiserver` and `provider:Kubernetes`. With this relationship, the service monitor retrieves data from the `kubernetes` service that is used to communicate with the control plane's API Service.
1-17	Review and run the `appendix3Task1-2.yaml` file, which creates a NodePort service for Grafana: **`kubectl create -f appendix3Task1-2.yaml`**
1-18	Open a web browser tab to **`http://[the ip address of one your Kubernetes worker nodes]:30901`**
1-19	At the Grafana login page, provide the following credentials: • Login as: **`admin`** • Password: **`admin`** • When prompted to change the password, choose **Skip**. You should be authenticated with Grafana.
1-20	In the left menu buttons, select the **Configuration** menu: 
1-21	Under Data sources, you find a Prometheus entry.
1-22	Select this entry as the **prometheus data source**.
1-23	At the bottom of the Prometheus data source, click **Test**. The data source should report that it is working correctly. However, even though it says it working properly, there seems to be bug. We will create another data course.
1-24	From the existing prometheus data source, copy the URL: http://prometheus-k8s.monitoring.svc:9090
1-25	Click **Back** to navigate back to the Configuration menu and ensure the Data sources tab is visible.
1-26	Click **Add data source**.
1-27	Select **Prometheus** as the time series database and then **Select**.
1-28	Rename this new data source: **Prometheus** (specifiy a capital P).
1-29	Paste or type in the URL you copied from step 1-24.
1-30	Leaving all other values their default, click **Save & test**.
1-31	You should now have two data source pointing to the Prometheus database.

Step	Action
1-32	On the Grafana page, click the **+** sign in the left menu bar, and then click **Import**.
1-33	Within your course materials, open the `appendix3Task1-3.json` file and copy all the contents by pressing **Ctrl-A** and then **Ctrl-C**.
1-34	In the Grafana import page, paste the contents of `appendix3Task1-3.json` into the import via panel JSON text box.
1-35	At the bottom of the page, click **Load**.
1-36	Within the Options page, select the **Prometheus data source** from the list, and keep all other options as the defaults.
1-37	Click **Import** to create the dashboard.
1-38	Review the dashboard that was created with the Prometheus data. Find additional dashboards at https://grafana.com/grafana/dashboards/.

Task 2: Configure a Sample App as a Prometheus Target

In this task, you configure a sample deployment that runs three replicas of a pod that produces telemetry data. Then you create a service monitor to register these replicas as a Prometheus target. Finally, you configure a Grafana dashboard to view the metrics from the sample application.

Step	Action
2-1	Review and run the `appendix3Task3-1.yaml` file, which creates a deployment and service of a sample app: **`kubectl create -f appendix3Task3-1.yaml`**
2-2	Review and run the `appendix3Task3-2.yaml` file, which creates a service monitor for the sample app: **`kubectl create -f appendix3Task3-2.yaml`**
2-3	 In the Prometheus Status Targets page, you should see a `serviceMonitor/monitoring/example-app` target.
2-4	On the Grafana page, click the **+** sign in the left menu bar, and then click **Import**.
2-5	Within your course materials, open the `appendix3Task2-3.json` file, and copy all the contents by pressing **Ctrl-A** and then pressing **Ctrl-C**.
2-6	In the Grafana import page, paste the contents of the `appendix3Task2-3.json` into the import via panel JSON text box.
2-7	At the bottom of the page, click **Load**.
2-8	Within the Options page, select the **Prometheus data source** from the list, and keep all other options as the defaults.
2-9	Click **Import** to create the dashboard.
2-10	Review the dashboard that was created with the Kubernetes deployment data, and in the upper-left corner of the dashboard, select example-app from the list of deployments.

Task 3: Configure Astra Trident as a Prometheus Target

In this task, you use Prometheus to monitor the trident namespace. Then you investigate how the Astra Trident service is configured with an endpoint that exposes the metric port configured in the Astra Trident pod. You verify that telemetry data comes from this endpoint. Then you create a service monitor to enable Astra Trident to become a Prometheus target. Finally, you configure the Grafana dashboard to view the Astra Trident telemetry.

Step	Action
3-1	Check to see whether the `prometheus-k8s` service account can monitor the `trident` namespace: **`kubectl -n trident auth can-i get pods --as=system:serviceaccount:monitoring:prometheus-k8s`**
3-2	Answer the following question: What was response? The answer should be yes, or you do not get telemetry data from Astra Trident. Unfortunately, the answer is no.
3-3	Review and run the `appendix3Task3-1.yaml` file, which creates a role and rolebinding to enable Prometheus to monitor the `trident` namespace: **`kubectl create -f appendix3Task3-1.yaml`**
3-4	Recheck to see whether the `prometheus-k8s` service account can monitor the `trident` namespace: **`kubectl -n trident auth can-i get pods --as=system:serviceaccount:monitoring:prometheus-k8s`**
3-5	Describe the trident service: **`kubectl -n trident describe svc trident-csi`**
3-6	Notice the port that is exposed for metrics: 9220/TCP. It is important to always reference this port by its name, not the port number for Prometheus. Its name is `metrics`.
3-7	Within your Windows PowerShell terminal, split the terminal (by using the menus or **Ctrl-Shift-5**) so that you have two terminals (label them TERMINAL 1 and TERMINAL 2).

3-8	Within TERMINAL 1, start a port forwarding process from the metric port (and identify the port that is being forwarded):
	`kubectl -n trident port-forward service/trident-csi :metrics`
	Sample output:
	`kubectl port-forward service/trident-csi -n trident :metrics`
	`Forwarding from 127.0.0.1:`**`65512`**` -> 8001`
	`Forwarding from [::1]:65512 -> 8001`

Step	Action
3-9	Within TERMINAL 2, because this terminal is a PowerShell terminal, issue the following command: **`Invoke-RestMethod http://127.0.0.1:[port forwarded from step 3-9] | findstr 'trident_backend_count'`** The following is the equivilant using a Linux shell: `curl -s localhost:[port forwarded from step 3-9] | grep trident_backend_count`
3-10	In TERMINAL 1, press **Ctrl-C** to exit the port-forwarding process.
3-11	Close TERMINAL 2.
3-12	Review and run the `appendix3Task3-2.yaml` file, which creates a service monitor for Astra Trident: **`kubectl create -f appendix3Task3-2.yaml`**
3-13	 The service monitor does a few things: • It looks for metrics retrieved by the trident-csi service (You can look for the `app` label that you match. This `app` label is present on the trident-csi service running in the `trident` namespace, specified in the `namespaceSelector`). • The `endpoint` for these metrics is defined to be the metrics port that the trident-csi service exposes.
3-14	From the Prometheus UI webpage, navigate to Status > Target, and verify that the Astra Trident service monitor shows in this list of targets.
3-15	From the Prometheus UI webpage, nagivate to Graph. The following is just some examples of expressions that are possible.
3-16	In the panel, add the following to the Expression line: `trident_backend_count` (total number of volumes per back end)
3-17	Click **Execute**.
3-18	Review the data in the table format.
3-19	Click **Add Panel** to add second panel.

3-20	In the second panel, add the following to the Expression line:
	`(kubelet_volume_stats_used_bytes)/(1024*1024*1024)`
	`(kubelet_volume_stats_used_bytes{namespace="default"})/(1024*1024*1024)`
	(used bytes per PVC for all namespaces)

3-21	Review the data in the table format.
3-22	Click **Add Panel** to add a third panel.

Step	Action

3-23	In the third panel, add the following to the Expression line:
	`trident_volume_count * on (backend_uuid)`
	`group_left (backend_name, backend_type) trident_backend_info`
	`or on (backend_uuid) trident_volume_count`
	(the number of volumes per back end)

3-24	Review the data in the table format.
3-25	Click **Add Panel** to add a fourth panel.

3-26	In the fourth panel, add the following to the Expression line:
	`(sum(trident_volume_allocated_bytes) by (backend_uuid)) / (1024*1024*1024)`
	`* on (backend_uuid) group_left(backend_name) trident_backend_info`
	(the number of volumes per back end)

3-27	Review the data in the table or graphic format. This would be more interesting if you did this exercise prior to Module 4 exercises.
3-28	Click **Add Panel** to add a fifth panel.
3-29	In the fifth panel, add the following to the Expression line: `rate(trident_ontap_operation_duration_in_milliseconds_by_svm_sum{op="ems-autosupport-log"}[5m])` (operations/SVM)
3-30	Review the data in the table or graphic format.
3-31	On the Grafana page, click the **+** sign in the left menu bar, and then click **Import**.
3-32	Within your course materials, open the `appendix3Task3-3.json` file, and copy all the contents by pressing **Ctrl-A** and then pressing **Ctrl-C**.
3-33	Back in Grafana's import page, paste the contents of the `appendix3Task3-3.json` into the import via panel JSON textbox.
3-34	At the bottom of the page, click **Load**.
3-35	Click **Import** to create the dashboard.
3-36	Review the dashboard that was created with the Astra Trident data.  This dashboard is located here: https://github.com/YvosOnTheHub/LabNetApp/blob/master/Kubernetes_v4/Scenarios/Scenario03.

End of exercise