Lab 10 - Deploy Tanzu Services and Application

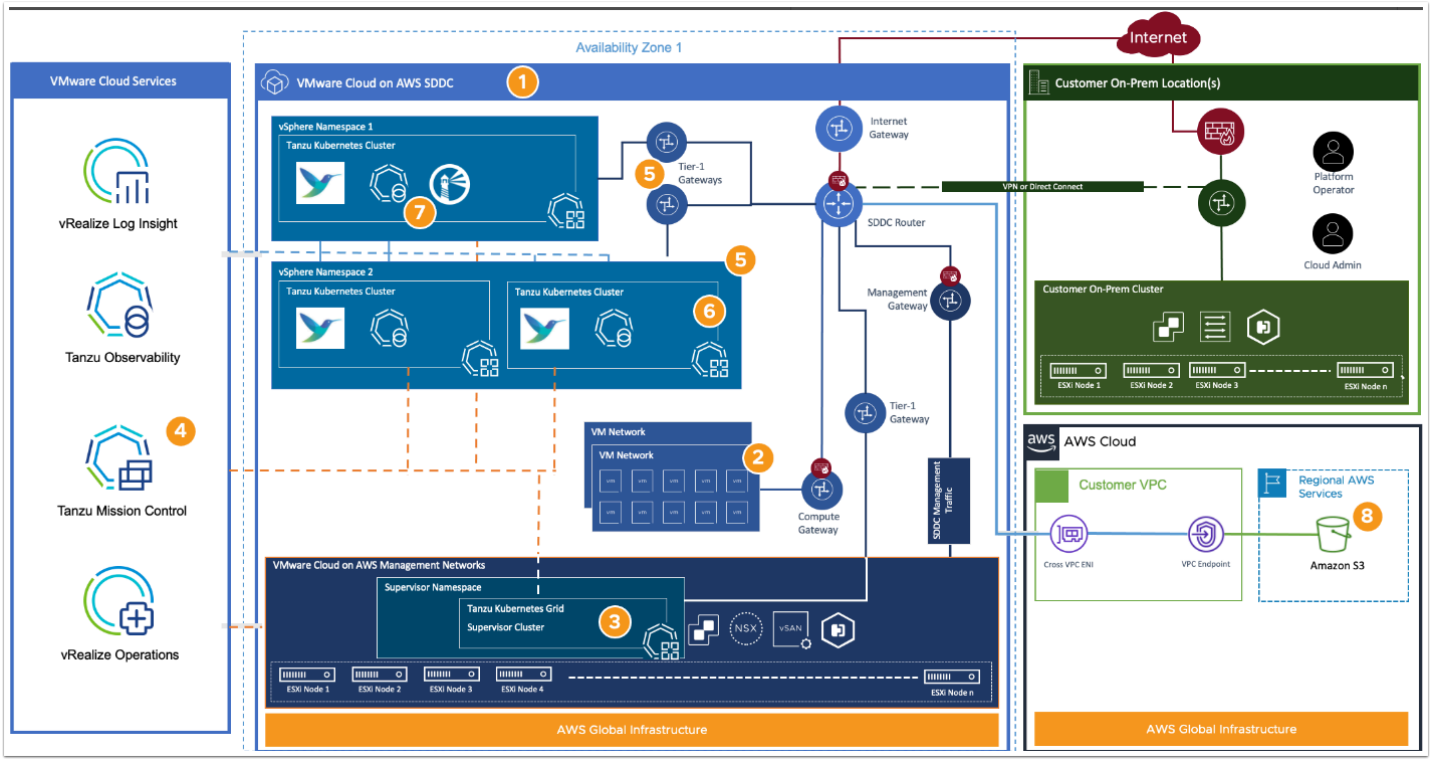
Introduction

VMware Cloud on AWS enables your IT and Operations teams to add value to your investments in AWS by extending your on-premises VMware vSphere environments to the AWS cloud. VMware Cloud on AWS is an integrated cloud offering jointly developed by Amazon Web Services (AWS) and VMware. It is optimized to run on dedicated, elastic, bare-metal Amazon Elastic Compute Cloud (Amazon EC2) infrastructure.

By running VMware Tanzu within the same infrastructure as the general VM workloads organizations can immediately start their modern application development strategy without incurring additional costs. For example, you can use SDDC spare capacity to run Tanzu Kubernetes Grid to enable next-generation application modernization, or compute power not used by disaster recovery can be used for Tanzu Kubernetes Grid clusters.

Tanzu Kubernetes Grid Managed Service Architecture

This reference architecture details how to use Tanzu services in a VMware Cloud on AWS SDDC. The Tanzu Kubernetes Grid Managed (TKG) managed service used in this architecture deploys a pair of vSphere Namespaces and some Tanzu Kubernetes clusters. Tanzu Mission Control (TMC) deploys Fluent Bit extensions for log collection and aggregation with vRealize Log Insight Cloud. TMC can also deploy Tanzu Observability to monitor Kubernetes metrics at scale. TMC configures containers and persistent storage backups through Velero using Amazon Web Services S3 as a durable object storage location.



1. Deploy the VMware Cloud on AWS SDDC in the desired region and availability zone.
2. Virtual machines run within the same SDDC on their own network segments connected to the VMware Cloud Compute Gateway and protected by firewall rules. These networks are typical but not required for the Tanzu Kubernetes Grid Service.
3. The cloud administrator activates the Tanzu Kubernetes Grid service, deploying the supervisor cluster into the VMware-managed network segments. This network placement is comparable to deploying vCenter and NSX appliances within VMware Cloud on AWS SDDC.
4. After successfully activating the TKG service, register the TKG Supervisor cluster as a management cluster within Tanzu Mission Control.
5. The cloud administrator creates vSphere Namespaces from within the vCenter's vSphere Client. These vSphere Namespaces provide resource and tenant isolation capabilities. The cloud administrator assigns resource limits for CPU, memory, and storage and then enables access to these resources for users via vCenter Single Sign-On. Each vSphere Namespace provisioned creates a new Tier-1 Gateway and connects a new network segment with the SDDC router. The vSphere Namespace network segments come from the CIDR ranges provided during the TKG activation process.
6. Platform operators deploy new Tanzu Kubernetes clusters into their assigned vSphere Namespaces through the *kubectl* CLI or Tanzu Mission Control.

7. The platform operators can use Tanzu Mission Control to set Kubernetes Polices for their clusters and use the TMC Catalog to deploy tools such as Fluent Bit configured to work with vRealize Log Insight Cloud for log aggregation, Tanzu Observability for capturing performance monitoring and metrics, or Harbor Image Registry for OCI compliant images.
8. Platform operators can configure an Amazon S3 storage location for a backup destination configured through Tanzu Mission Control.

TASKS

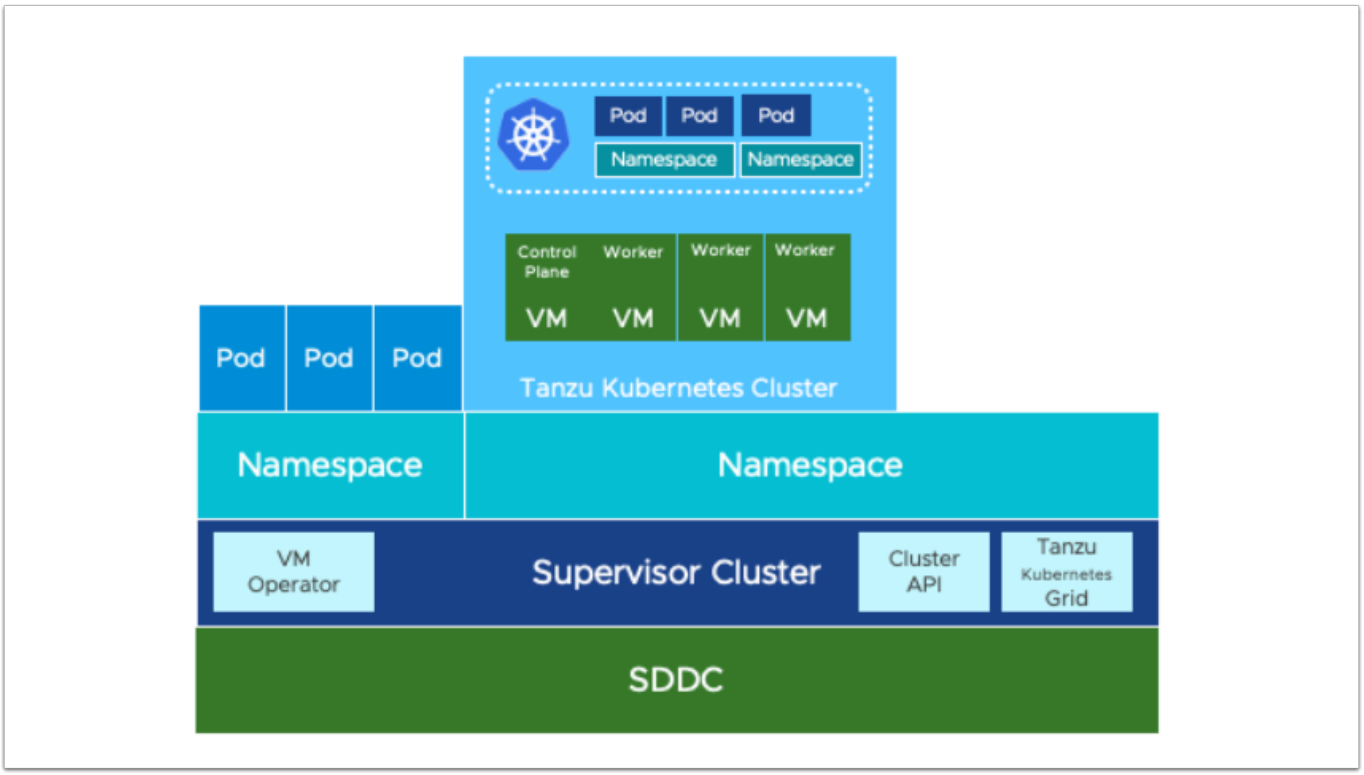
Task 1 - Create a vSphere Namespace

Unlike earlier labs in this workshop, you will share a 4-Node SDDC with other students (You do not have a dedicated SDDC for your exclusive use), for this reason, the majority of the tasks you will carry out will be related to the DevOps persona. All ITOps-related tasks, except for creating a vSphere Namespace have been completed by the instructor, those tasks include but are not limited to:

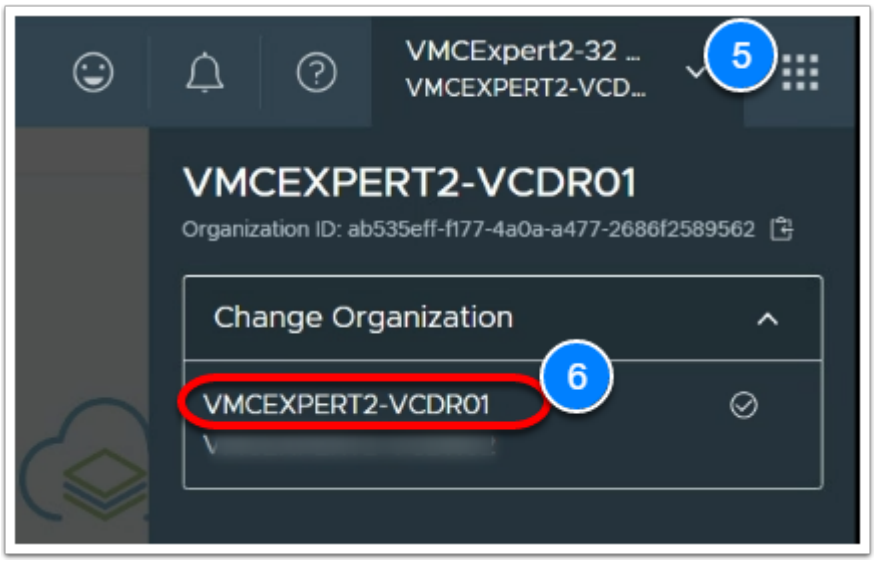
1. Activate TKG service and deploy the supervisor Cluster in vSphere
2. Setup the Gateway firewall rules to allow access for containerized applications
3. Install and Configure the CLI tools needed to manage the TKG Cluster(s)
4. etc..

While the creation of a vSphere Namespace is an ITOps task, you will perform it here, beyond that all other tasks are DevOps-related.

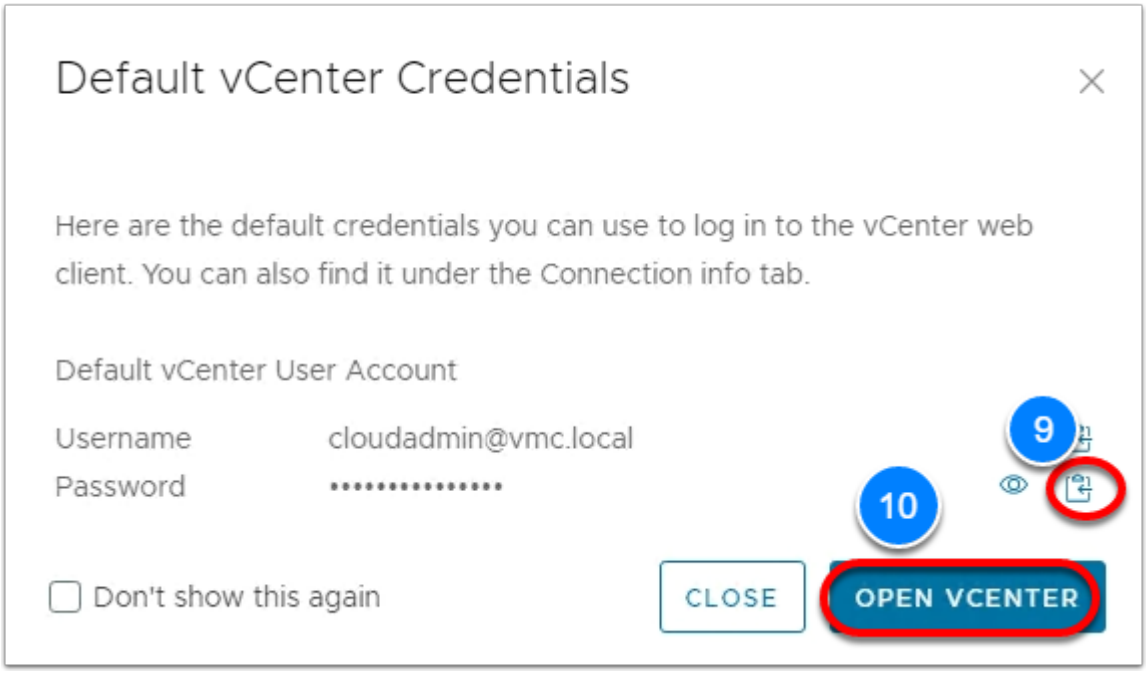
vSphere Namespaces are set up by the vSphere Admin, run in the context of the Supervisor Cluster and allow admins to control resource limits and other policies



1. If you are no longer logged into your VDI desktop or lost the RDP session to the Tanzu Desktop, then follow the instruction in Lab 10 Task 1 (Steps 1 through 8) before continuing to step 2
2. From the Tanzu Desktop Launch Firefox, Edge or Brave
   (**NOTE**: You will use this browser instance to access the SDDC vCenter as **CloudAdmin**)
3. In the browser go to https://vmc.vmware.com/sddcs/console
4. log in as:
   - **vmcexpert{1|2|3}-##@27virtual.net** (where **##** is your student number)
     i.e **vmcexpert1-02@27virtual.net**
   - **{Password-Provided-by-Instructor}**
5. In the upper right-hand corner Click the dropdown next to **<your-username>**
6. Confirm you are logged into the correct Organization, If not select it
   Note: Your correct Organization should be **{Your-VMCEXPERT-Environment}-VCDR01** e.g. **VMCEXPERT2-VCDR01**

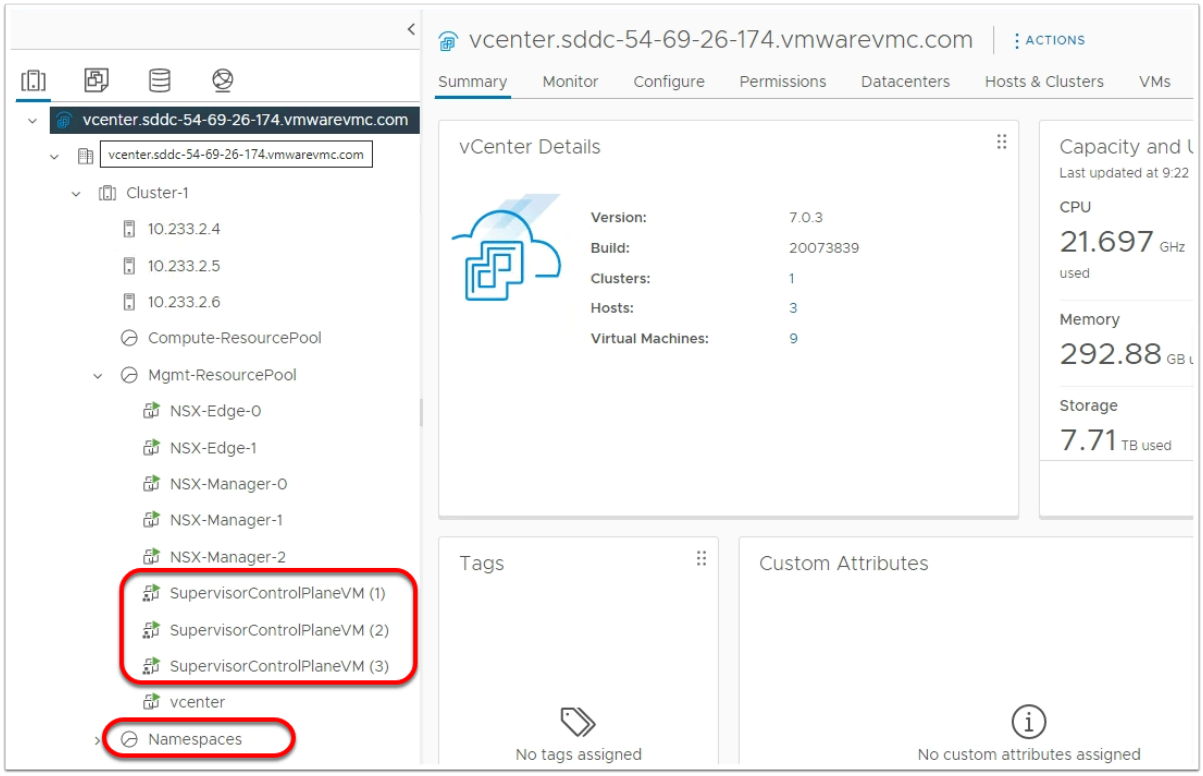


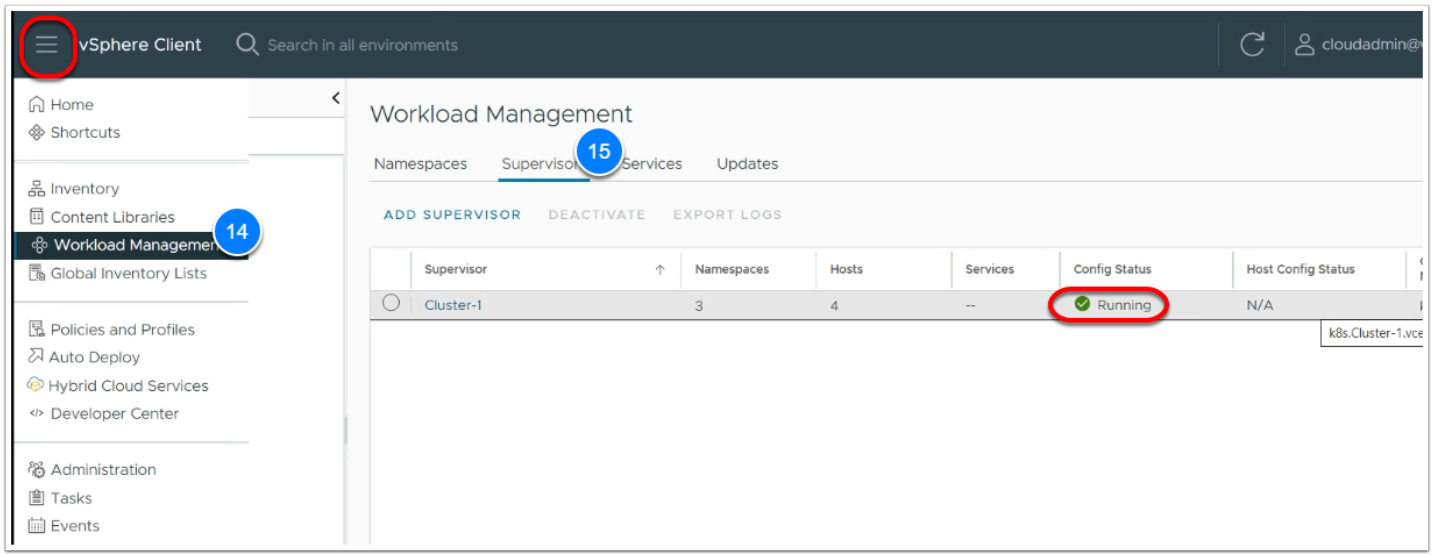
7. In the SDDC Tile, click **Open vCenter**
8. Click **Show Credential**
9. Copy the Default vCenter User **Password** (Store this for future use)
10. Click **Open vCenter**



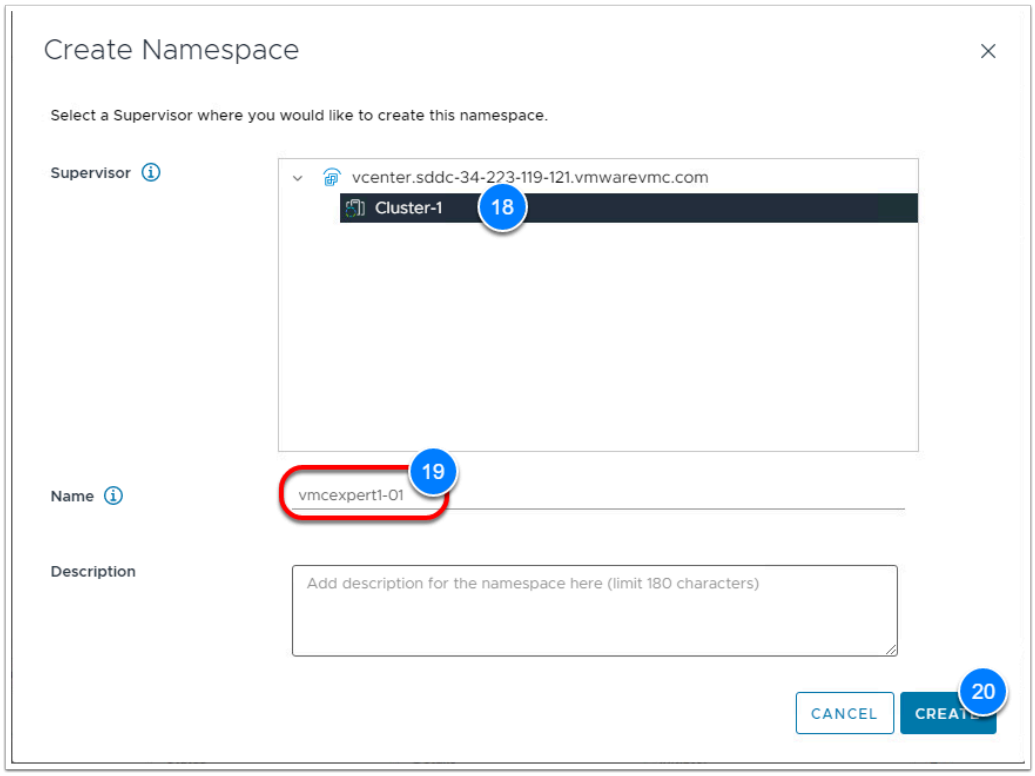
11. Log into vCenter as:
    - User Name: **Cloudadmin@vmc.local**
    - Password: (**Paste in the Password from step 9**)
12. Once Logged into vCenter, inspect the **Host & Clusters Inventory** view
13. Take note of the **Supervisor Cluster Control VMs** in the **Mgmt-ResourcePool** Resource Pool
    Also note the **NameSpaces** Resource Pool
14. In the Upper Left-hand corner, click the **Hamburger Menu,** Select **Workload Management**

15. Click the **Supervisors** tab to confirm the Supervisor Cluster exists and is in a Health state



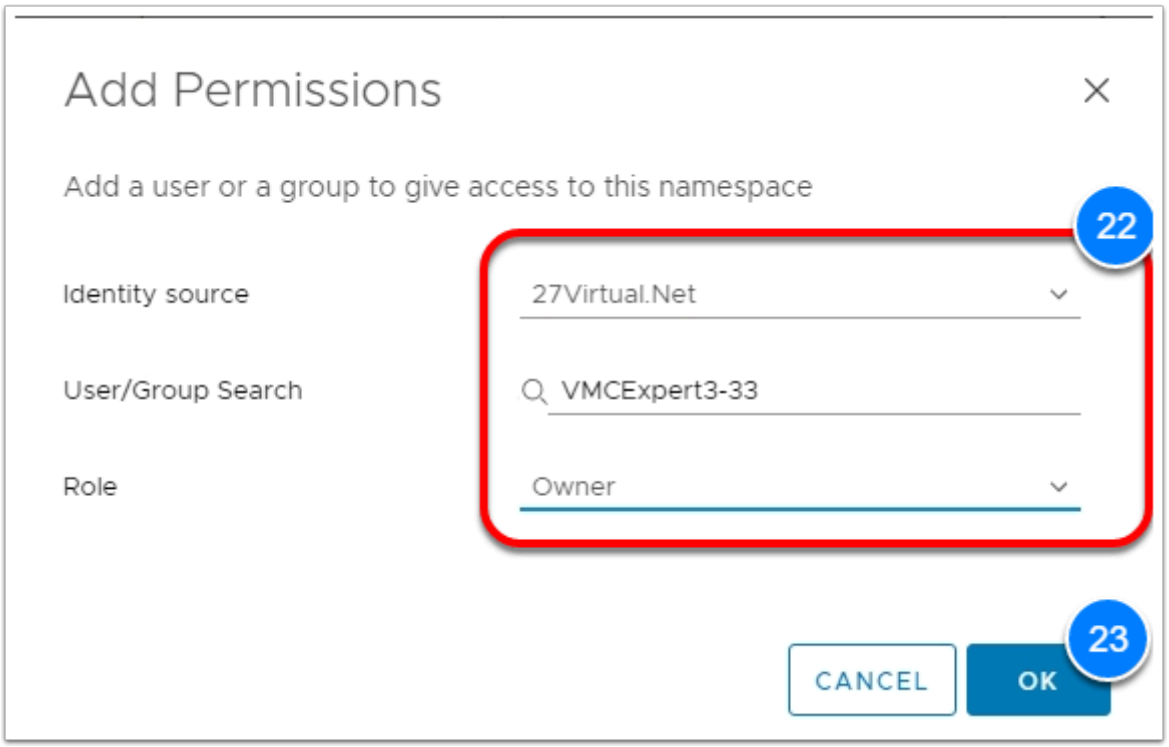


16. Click the **Namespaces** tab
17. Click **New Namespace**
18. Expand **vCenter** (**vcenter.sddc.xx.xx.xx.xx.vmwarevmc.com**) and select **Cluster-1**
19. Type {**Your Username**} in the Name field **I.E. vmcexpert3-33**
20. Click **Create**

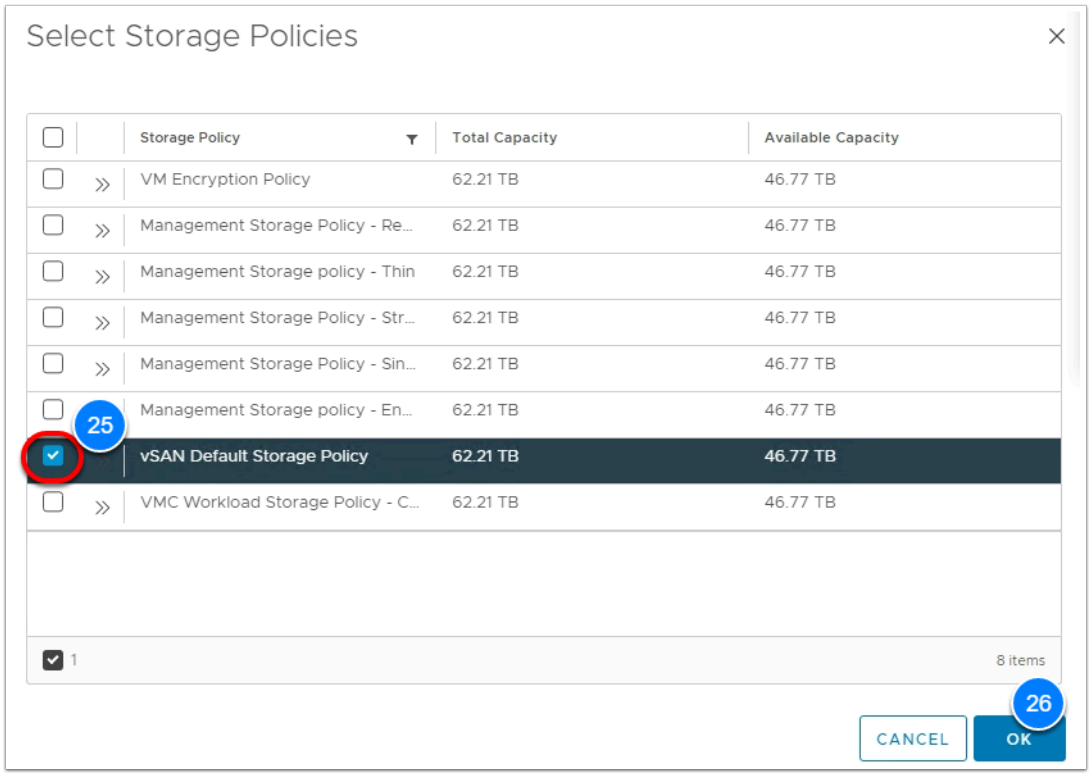


We will now add come controls around the Namespace by limiting access through RBAC, adding a Storage Policy and selecting the VM Classes that could be used to create a Tanzu Cluster. We will also log into the Namespace via CLI

21. On the Namespace you just created, Click **Add Permissions** to restrict this Namespace to your DevOps user account
22. In the Add Permissions Dialog Choose/enter the following
    - **Identity Source: 27Virtual.net**
    - **User/Group: {Your User Name} I.E. VMCEXPERT3-33**
    - **Role: Owner**
23. Click **OK**

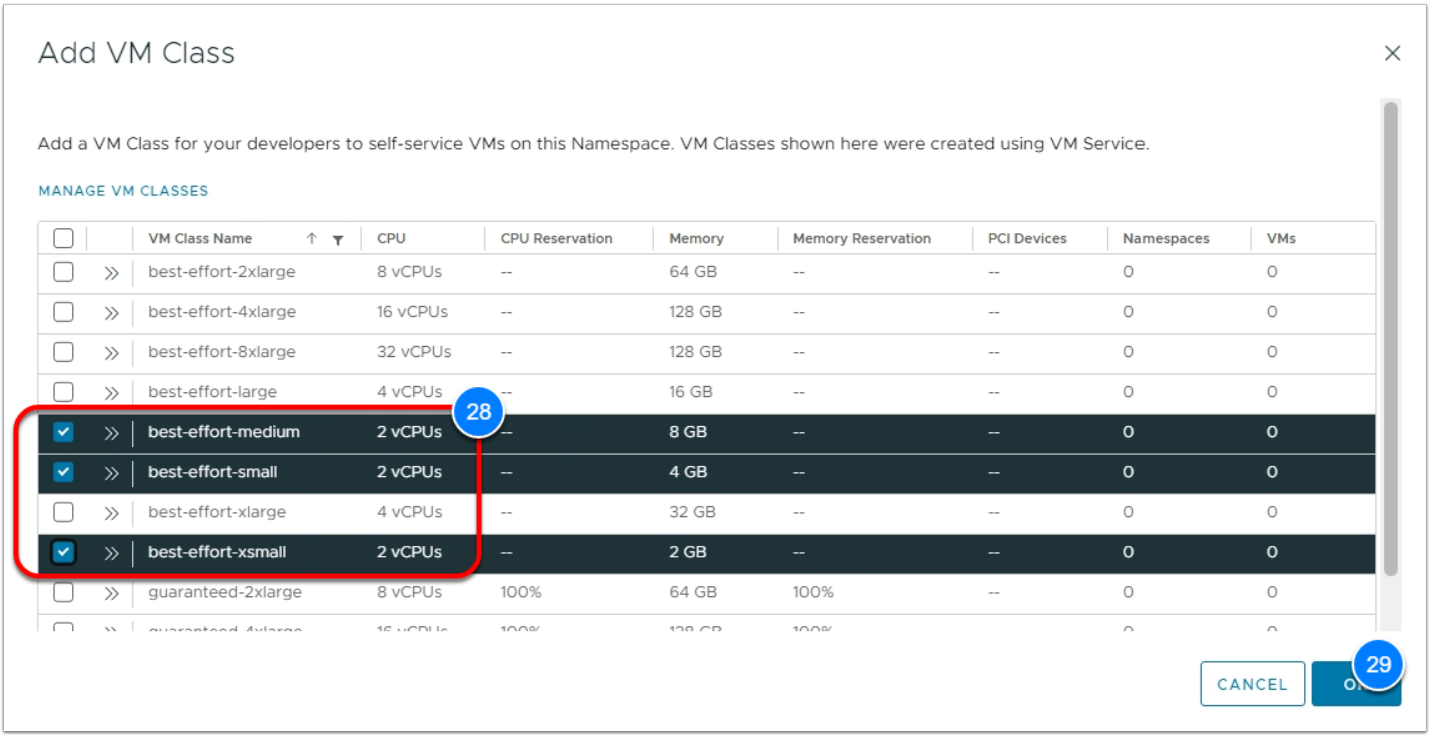


24. In the **Storage** Tile, Click **Add Storage**
25. In the Select **Storage Policies** dialog, select **vSAN Default Storage Policy**
26. Click **OK**

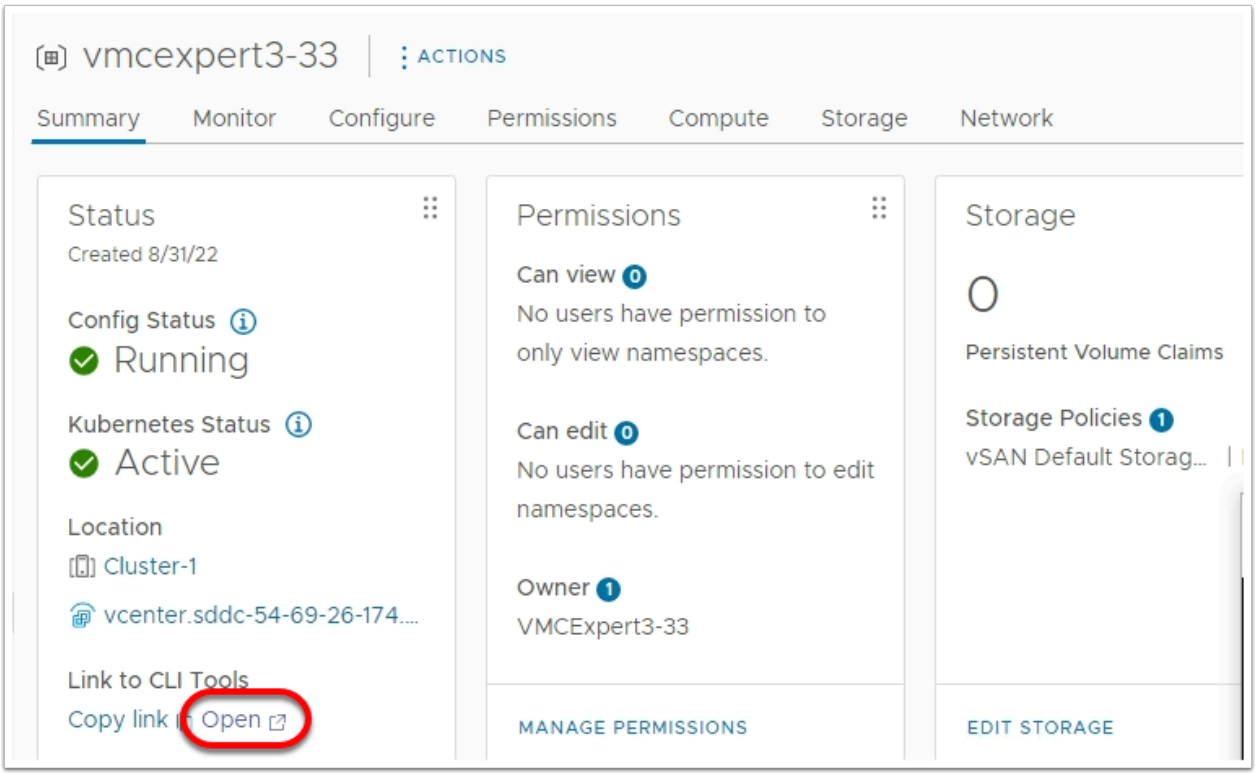


27. In the **VM Service** Tile, Click **Add VM Class**

28. In the Select **Add VM Class** dialog, select **best-effort-medium, best-effort-small** and **best-effort-xsmall**
29. Click **OK**



30. On the Status Tile In the vCenter Console, Click **Open** to open the Link to the Kubernetes Control Plane
31. Copy and store this URL (Do not include the https:// and the trailing "/").
    Note: You can store this in your lab input workbook

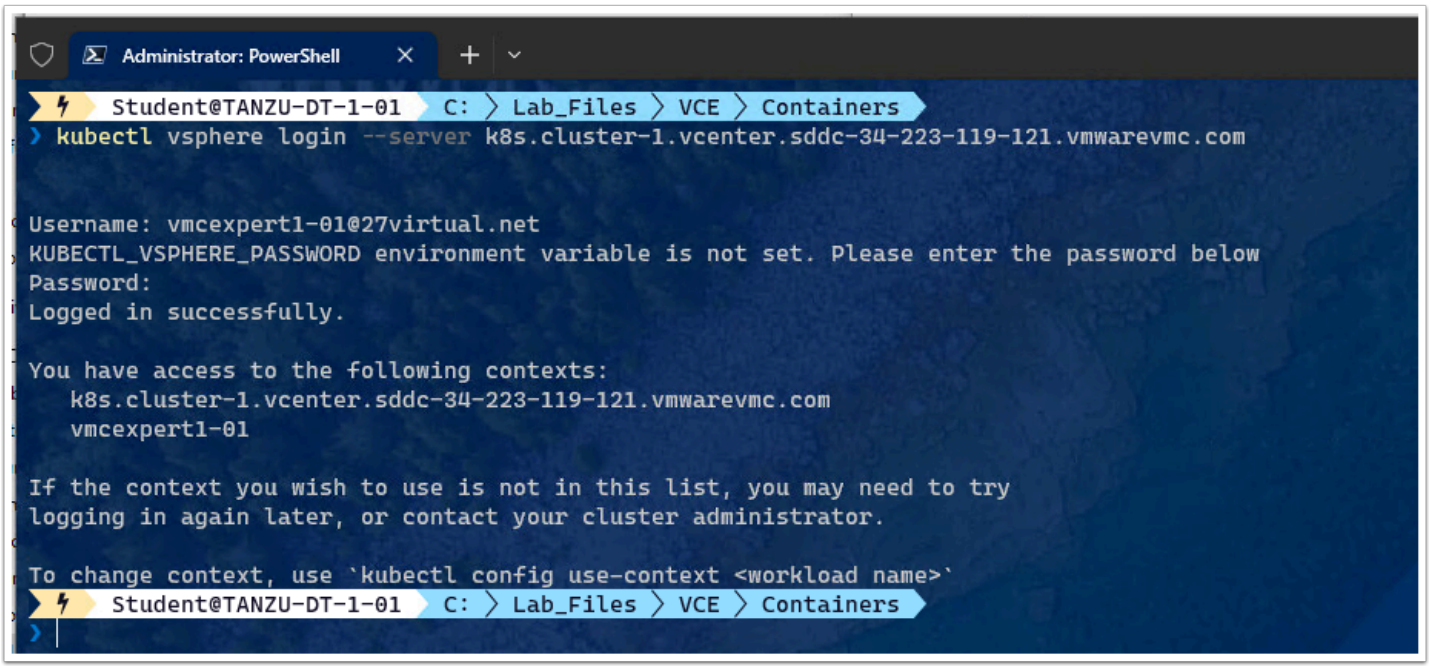


32. Launch the Windows Terminal, in the PowerShell window, type the following command to access your namespace

```
<p>kubectl vsphere login --server={Kubernetes Control Plane Endpoint}</p>
```

Click to copy

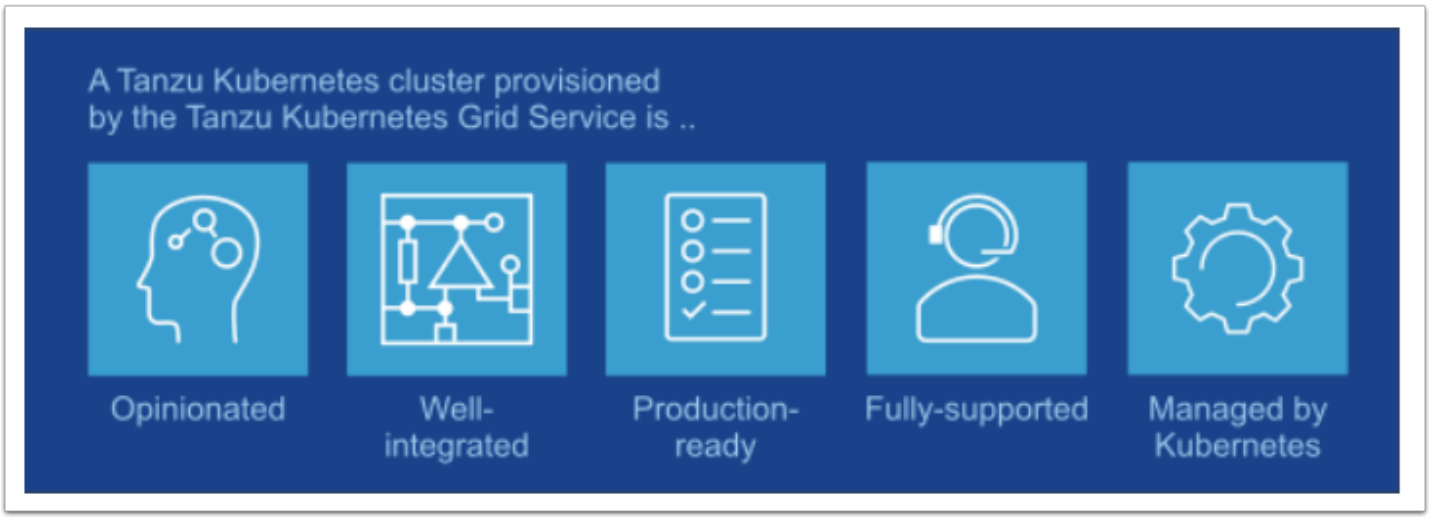
33. When Prompted for credentials:
    - Username: {**enter your DevOps Username**} **I.E. vmcexpert3-33@27virtual.net**
    - Password: **{Password-Provided-by-Instructor}**



Task 2 - Create a Tanzu Kubernetes Cluster

A Tanzu Kubernetes cluster is a full distribution of the open-source Kubernetes container orchestration platform that is built, signed, and supported by VMware. You can provision and operate Tanzu Kubernetes clusters on the Supervisor Cluster by using the Tanzu Kubernetes Grid Service. A Supervisor Cluster is a vSphere cluster that is enabled with vSphere with Tanzu.

When you deploy a workload cluster, most of the configuration for the cluster is the same as the configuration of the management cluster that you use to deploy it. Because of this, the easiest way to create a configuration file for a workload is to start with a copy of the management cluster configuration file.



1. Launch the Windows Terminal. In the PowerShell window, type the following command to access your namespace

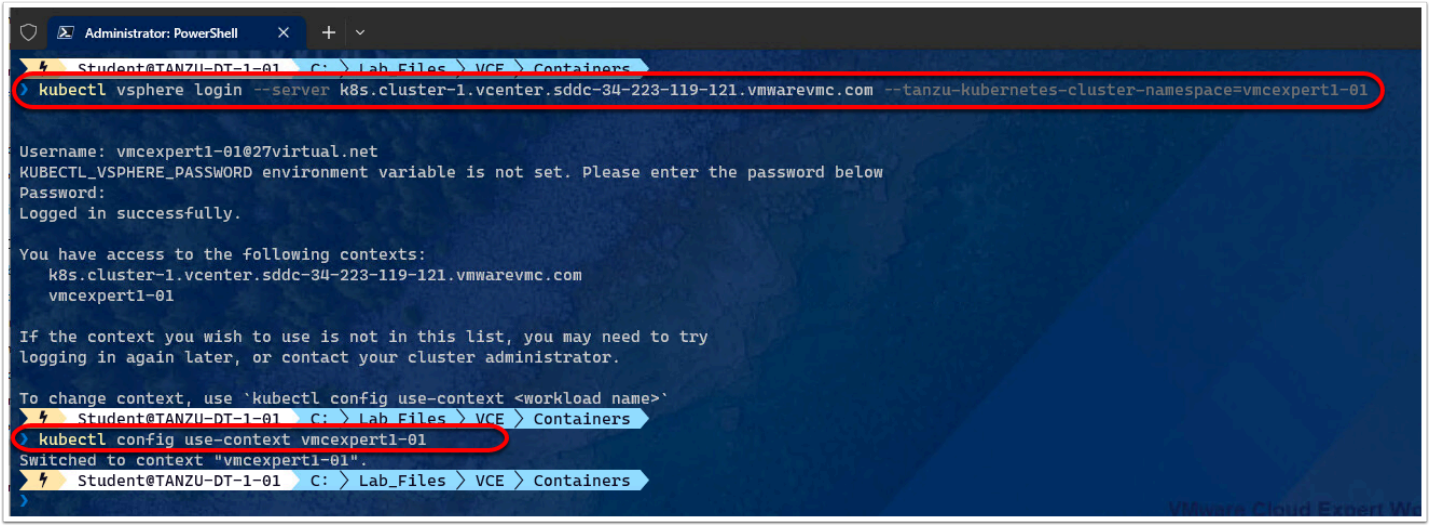
```
<p>kubectl vsphere login --server={Kubernetes Control Plane Endpoint} --tanzu-kubernetes-cluster-namespace={Your namespace created in task 1}</p>
```

Click to copy

2. When Prompted for credentials:
   - Username: {**enter your DevOps Username**} **I.E. vmcexpert3-33@27virtual.net**
   - Password: **{Password-Provided-by-Instructor}**

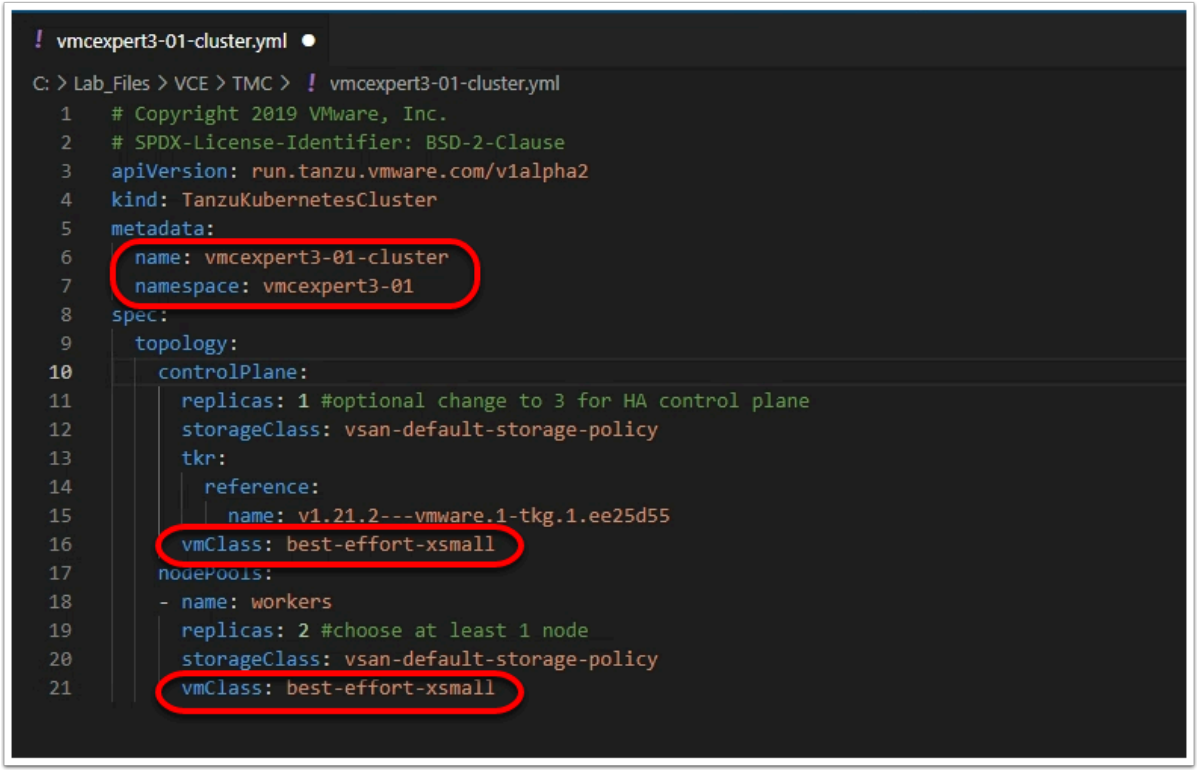
```
<p>kubectl config use-context {Your namespace created in task 1}</p>
```

Click to copy



We will now edit and save the TKC manifest file which we'll use to create the Tanzu Cluster

3. From your Tanzu Desktop, In windows explorer, navigate to **C:\lab_files\vce\TMC**
4. Locate the **vmcexpert#.XX-cluster** file
5. Double click the file to edit it in Visual Studio Code
6. With the File opened, **review it's content.** Observe the settings for StorageClass, VMClass and replicas that will be used for this cluster
7. **Edit lines 6, 7, 16 & 21**
   - name: **{Your Student name}-cluster** **NOTE: the value should have no braces {}**
   - Namespace: {**Your student name**}
   - vmClass: **best-effort-xsmall**
8. Save the file as **{Your user name}-cluster.yml**



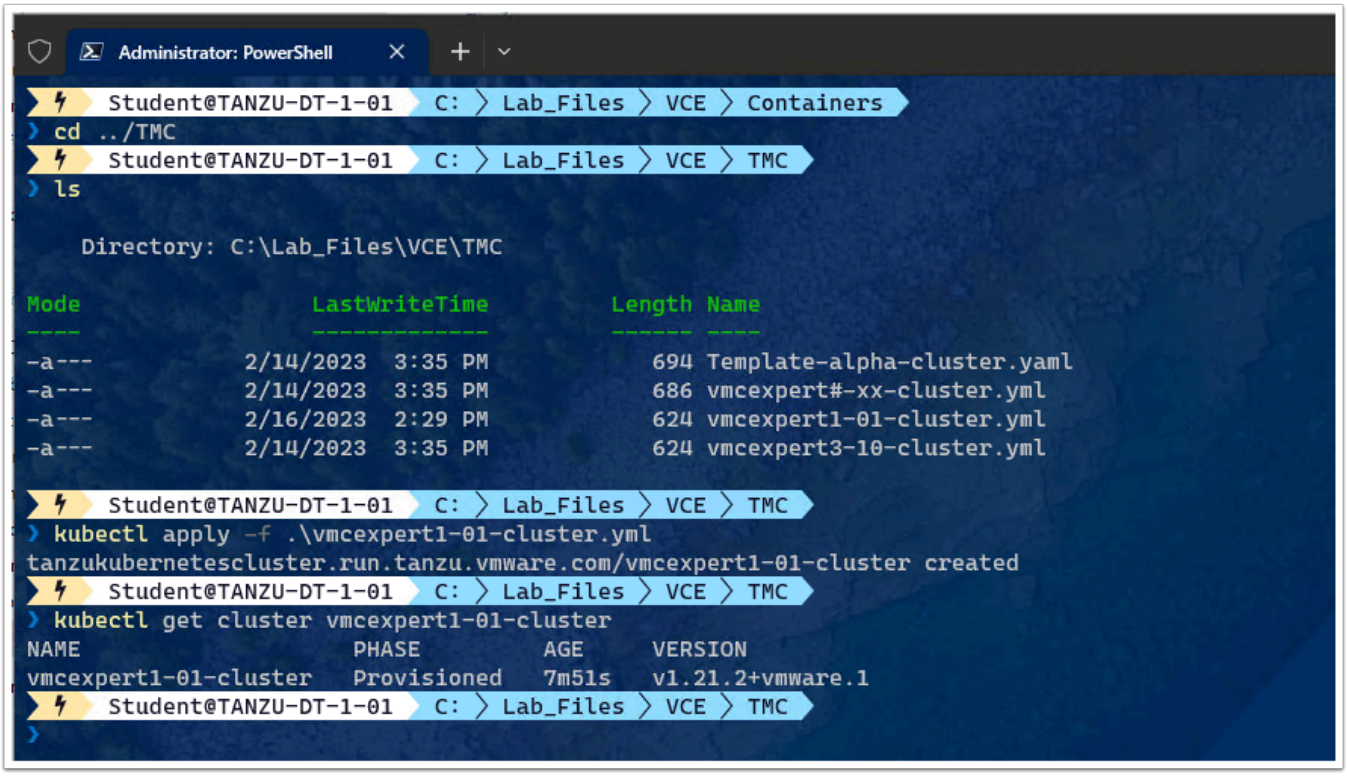
9. In Windows terminal, change directory to **c:\Lab_Files\VCE\TMC**
10. list the files, to confirm your cluster manifest file created in step 7 - 8 exists
11. run the following command to create the cluster

```
<p>kubectl apply -f .\{your username}-cluster.yml</p>
```

Click to copy

```
<p>kubectl get cluster {your username}-cluster</p>
```

Click to copy



NOTE: While the CLI may report that the Cluster has been provisioned, it may actually take up to 5 mins for all the vSphere and NSX based tasks to complete. These tasks include:

1. Deploying and configuring the Kubernetes cluster VMs
2. Creating a dedicated NSX network segment for the namespace
3. Deploying a T1 gateway for the Namespace
4. etc..

With the cluster now provisioned, we will re-authenticate to the TKG Supervisor Service to get a new cluster context in your KUBECONFIG file.

12. Run the following login command to update the KUBECONFIG file:

```
<p>kubectl vsphere login --server={Kubernetes Control Plane Endpoint} --tanzu-kubernetes-cluster-namespace={Your namespace} --tanzu-kubernetes-cluster-name={Your Cluster name}</p>
```

Click to copy

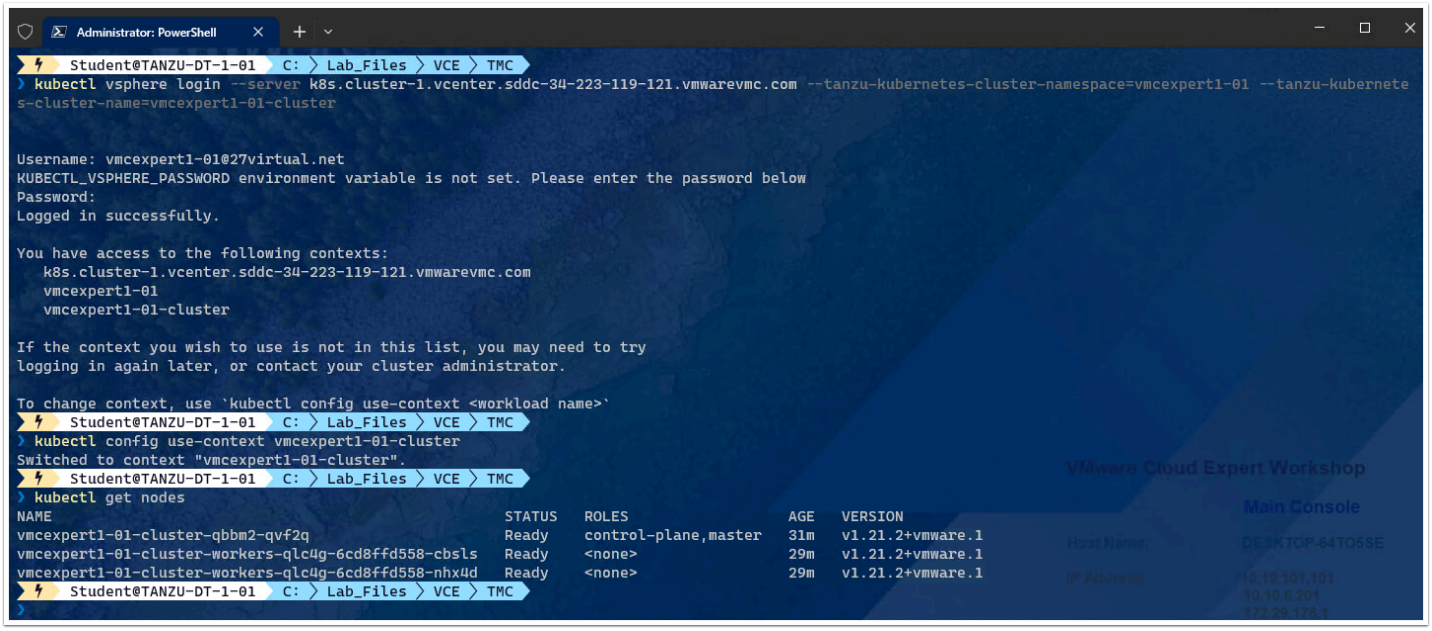
13. When Prompted for credentials:
    - Username: {**enter your DevOps Username**} **I.E. vmcexpert3-33@27virtual.net**
    - Password: **{Password-Provided-by-Instructor}**

```
<p>kubectl config use-context {Your cluster name}</p>
```

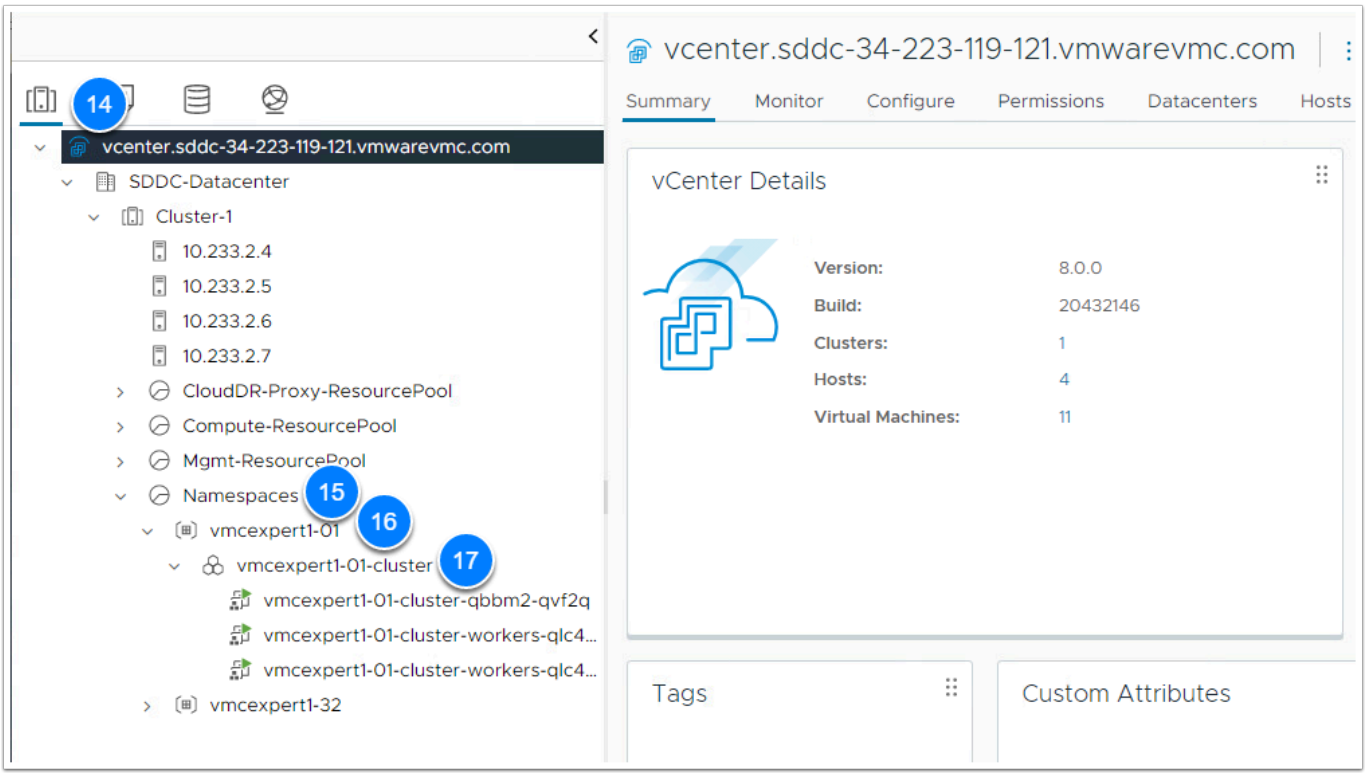
Click to copy

```
<p>kubectl get nodes</p>
```

Click to copy



14. In the vSphere UI in the browser instance, switch back to "**Hosts & Clusters**" view
15. Expand the **Namespaces** Resource Pool
16. Identify **your namespace** and expand it
17. identify **your cluster** and expand it
18. Notice the nodes (VMs) that were deployed when you created your cluster



Task 3 - Working with Pods and Deployments

Pods are the smallest execution unit in a Kubernetes cluster. In Kubernetes, containers do not run directly on cluster nodes; instead one or more containers are encased in a pod. All applications in a pod share the same resources and local network, easing communications between applications in a pod. Pods utilize an agent on each node called a kubelet to communicate with the Kubernetes API and the rest of the cluster. Although developers need API access, management of pods is transitioning to the domain of DevOps.

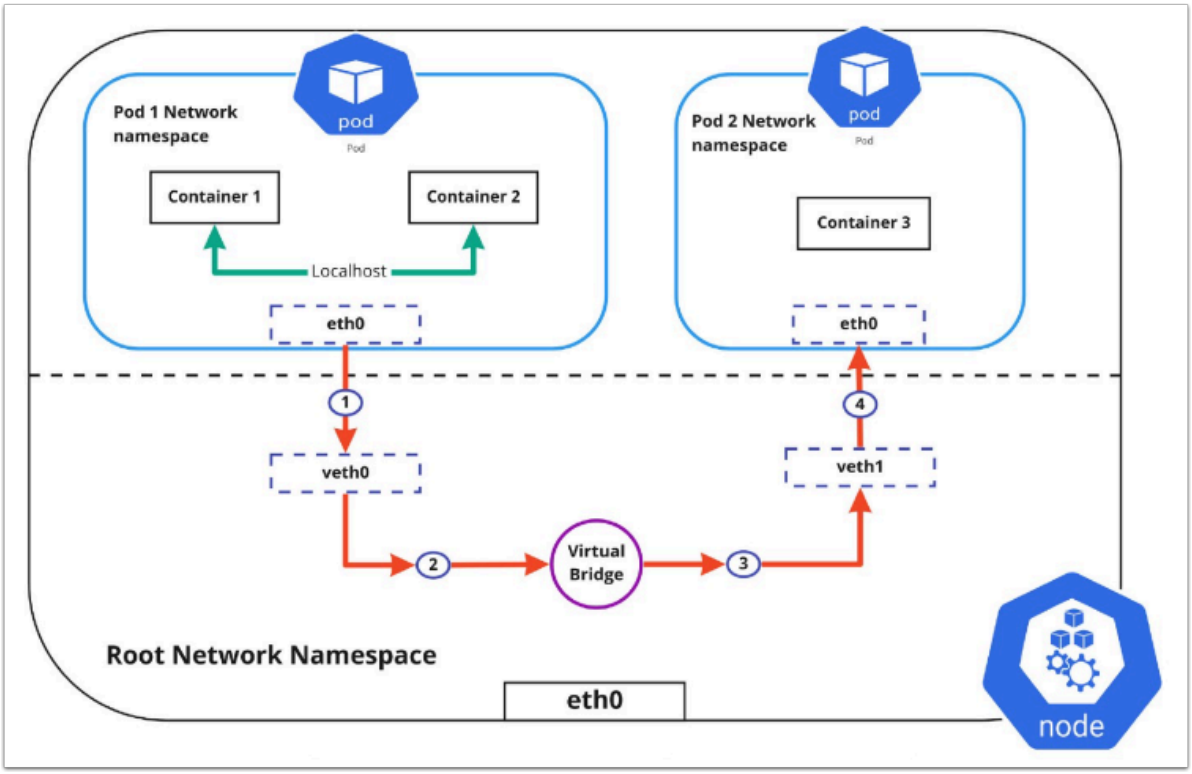
A Kubernetes pod is a collection of one or more Linux® containers, and is the smallest unit of a Kubernetes application. Any given pod can be composed of multiple, tightly coupled containers (an advanced use case) or just a single container (a more common use case).

Difference between Kubernetes pods and nodes

Pods are an abstraction of executable code, nodes are abstractions of computer hardware, so the comparison is a bit apples-and-oranges.

Pods are simply the smallest unit of execution in Kubernetes, consisting of one or more containers, each with one or more applications and their binaries.

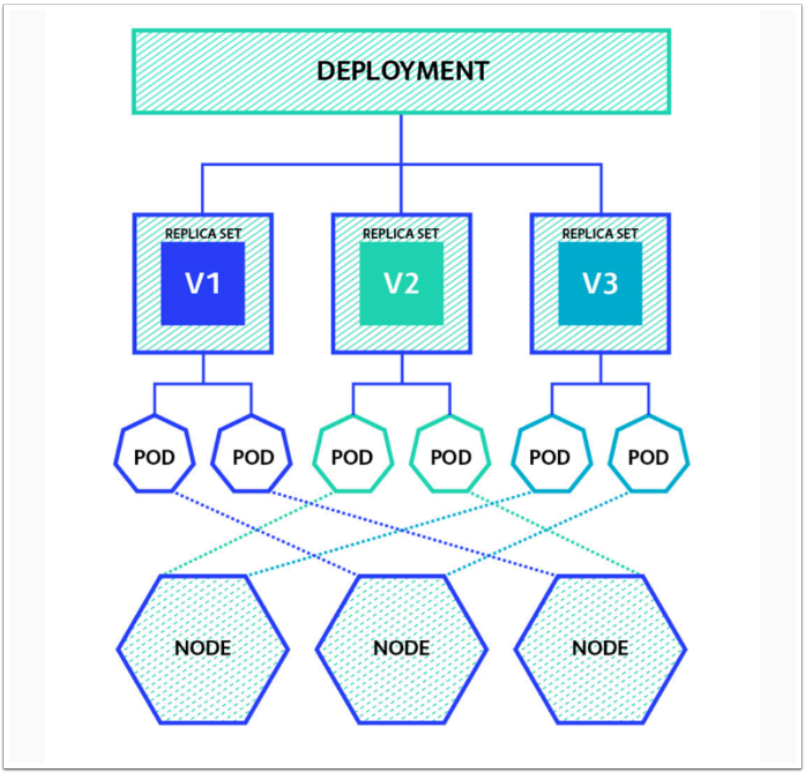
Nodes are the physical servers or VMs that comprise a Kubernetes Cluster. Nodes are interchangeable and typically not addressed individually by users or IT, other than when maintenance is required.



A Kubernetes Deployment is **used to tell Kubernetes how to create or modify instances of the pods that hold a containerized application**. Deployments can scale the number of replica pods, enable the rollout of updated code in a controlled manner, or roll back to an earlier deployment version if necessary.

Benefits of Kubernetes Deployment

- Kubernetes automates the work and repetitive manual functions that are involved in deploying, scaling, and updating applications in production.
- Since the Kubernetes deployment controller is always monitoring the health of pods and nodes, it can replace a failed pod or bypass down nodes, replacing those pods to ensure continuity of critical applications.
- Deployments automate the launching of pod instances and ensure they are running as defined across all the nodes in the cluster. More automation translates to faster deployments with fewer errors.



1. The Authentication token stored in your local KUBECONFIG file expires every 10 hours. You may want to re-authenticate to the TKG Service before starting the lab to ensure you have access to the Supervisor cluster.
   Run the following login command to update the KUBECONFIG file:

```
<p>kubectl vsphere login --server={Kubernetes Control Plane Endpoint} --tanzu-kubernetes-cluster-namespace={Your namespace} --tanzu-kubernetes-cluster-name={Your Cluster name}</p>
```

Click to copy

2. When Prompted for credentials:
   - Username: {**enter your DevOps Username**} **I.E. vmcexpert3-33@27virtual.net**
   - Password: **{Password-Provided-by-Instructor}**

```
<p>kubectl config use-context {Your cluster name}</p>
```

Click to copy

Now let's create our first Kubernetes Pod

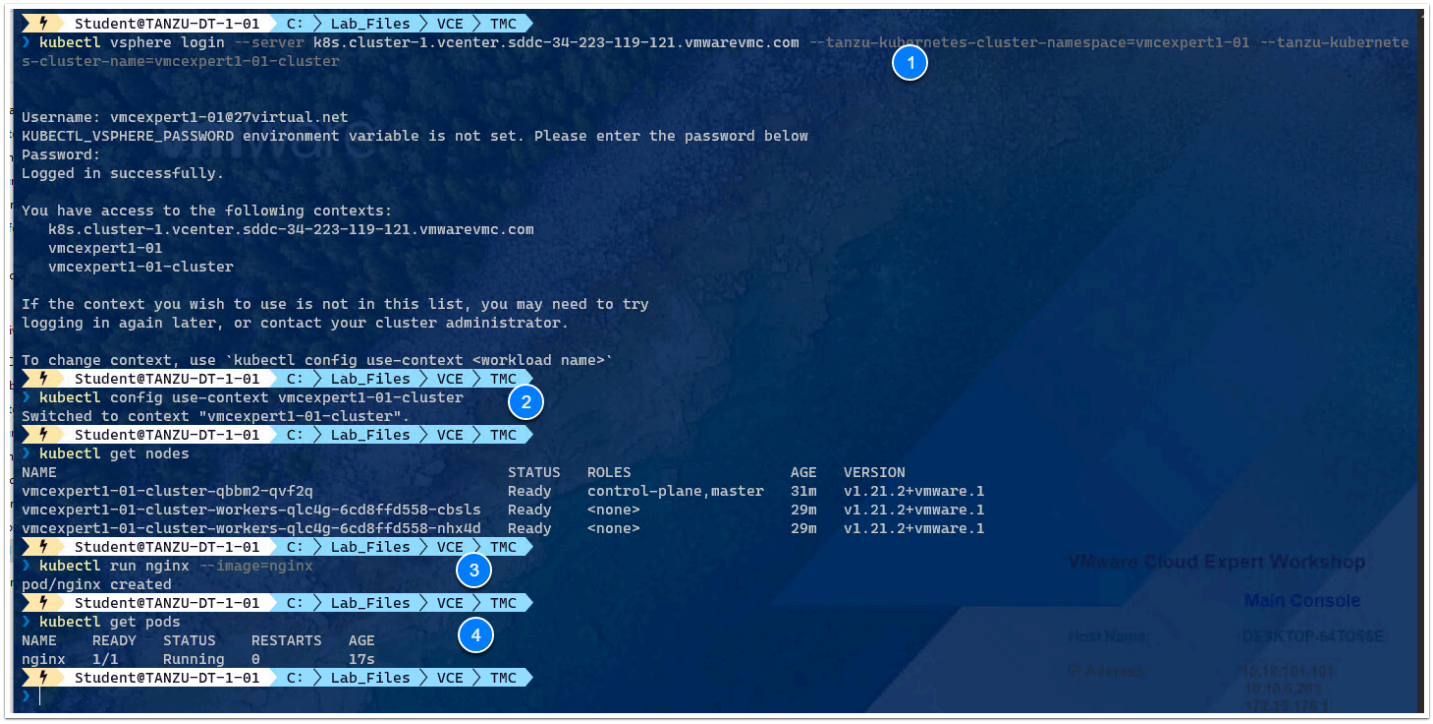
3. Run the following commands to create an NGINX pod and view it

```
<p>kubectl run nginx --image=nginx</p>
```

Click to copy

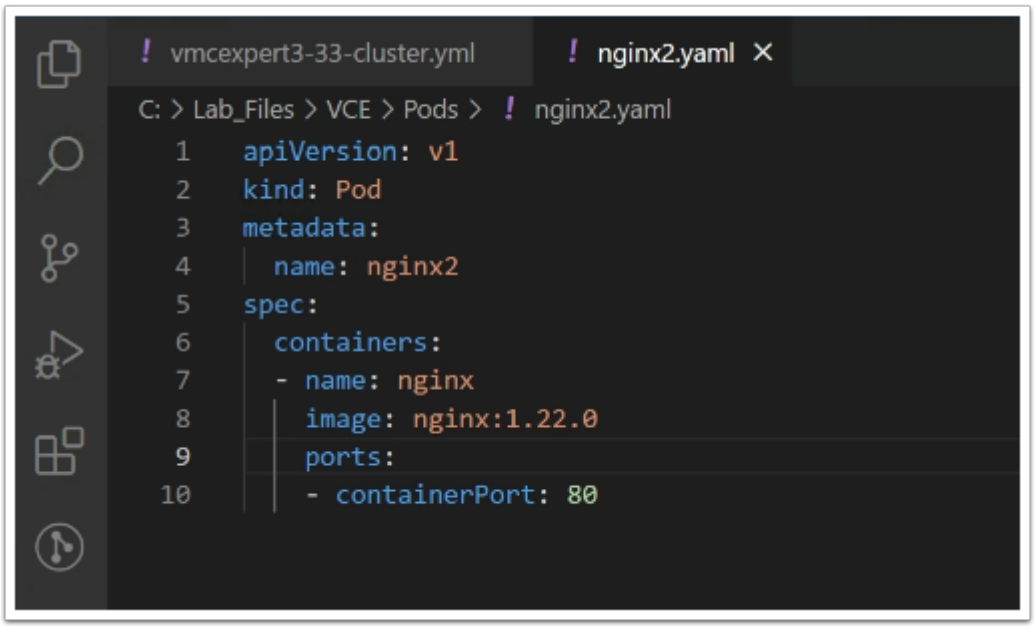
```
<p>kubectl get pods</p>
```

Click to copy



We will now deploy a 2nd pod from a YAML file

4. In Windows Explorer, navigate to **c:\Lab_Files\VCE\Pods**
5. Open the **nginx2.yaml** file
6. examine the content of the file to determine what it specifies



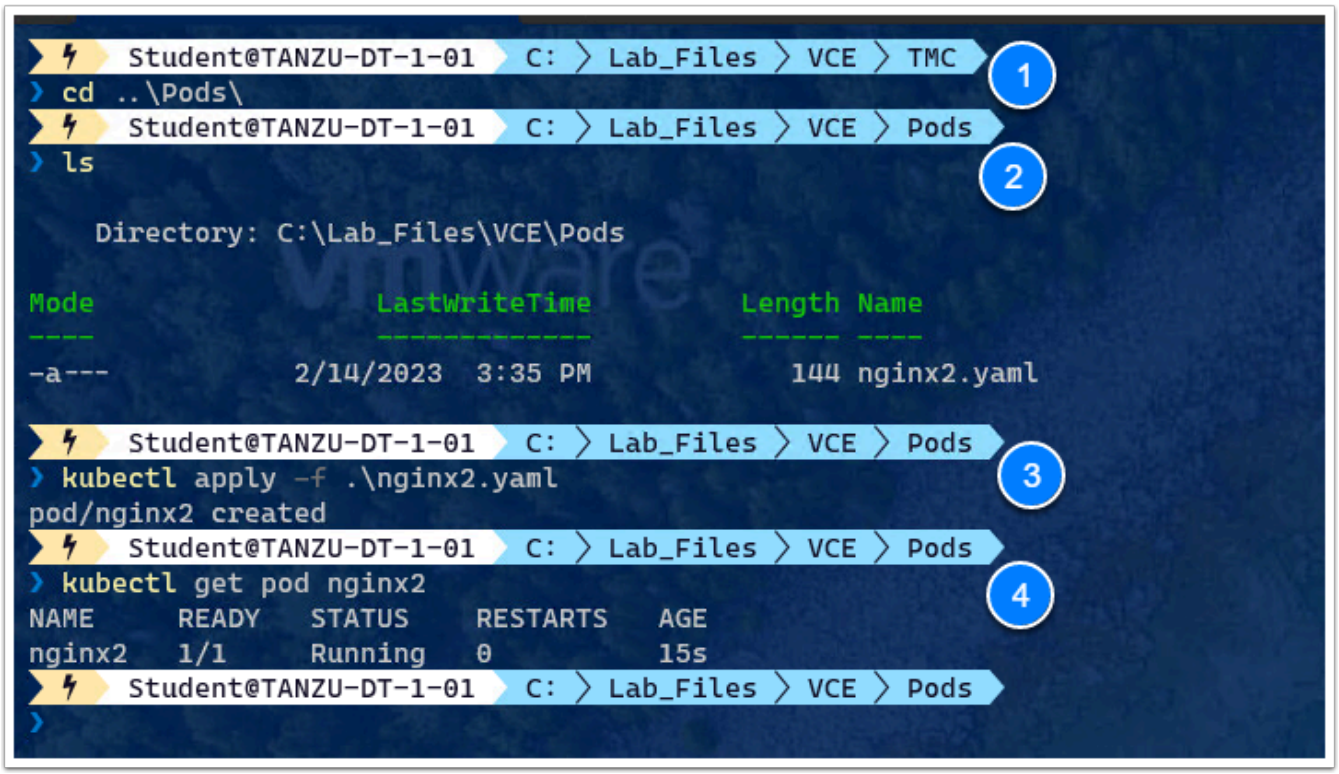
7. In Windows terminal, change directory to **c:\Lab_Files\VCE\Pods**
8. List the contents of the directory to confirm the nginx2 file exists
9. Let's create the Pod and examine it by running the following commands:

```
<p>kubectl apply -f .\nginx2.yaml</p>
```

Click to copy

```
<p>kubectl get pod nginx2</p>
```

Click to copy



Now let's create our 3rd pod and examine how to retrieve information about deployed pods

10. In Windows terminal, run the following commands to deploy an Ubuntu pod with a standard output, view details of the Pod and retrieve the standard output

```
<p>kubectl run ubuntu --image=ubuntu -- echo "Deploy human virtues of Compassion and Humanity"</p>
```

Click to copy

```
<p>kubectl get pod ubuntu</p>
```

Click to copy

```
<p>kubectl describe pod ubuntu</p>
```

Click to copy

```
<p>kubectl logs ubuntu</p>
```

Click to copy

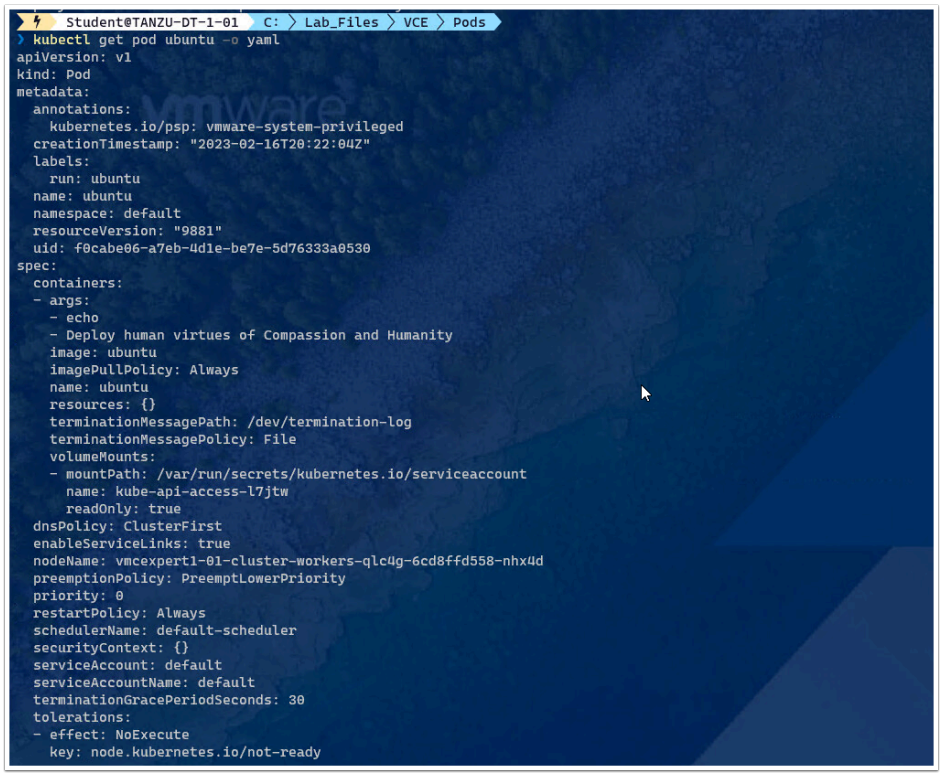
What does the log show and why?

11. To get the YAML of running Ubuntu Pod, run the following command:

```
<p>kubectl get pod ubuntu -o yaml</p>
```

Click to copy

The result is that the API server returns the declarative YAML that can be used to build a new Kubernetes manifest



12. Now, let's cleanup after ourselves and remove the pods we deployed. to do so run the following command:

```
<p>kubectl get pods</p>
```

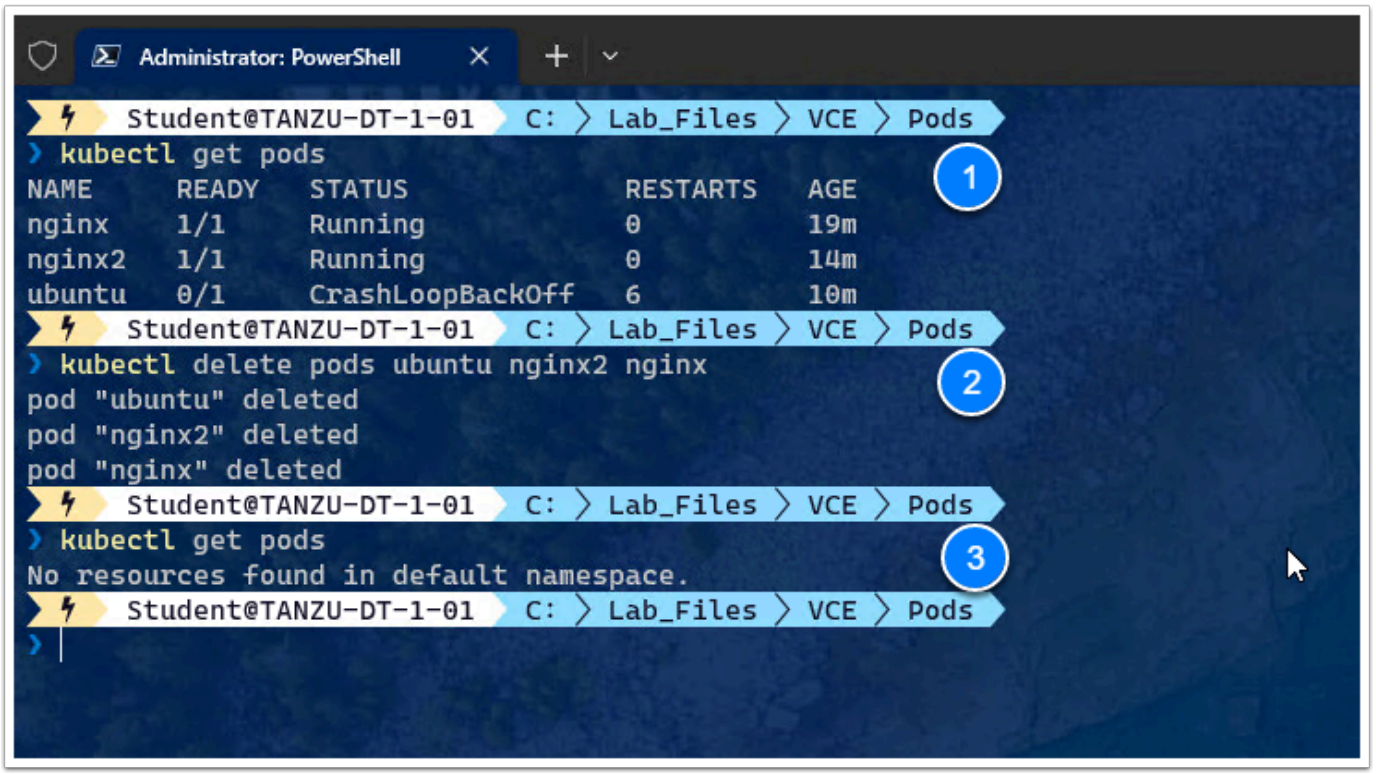
Click to copy

```
<p>kubectl delete pods ubuntu nginx2 nginx</p>
```

Click to copy

```
<p>kubectl get pods</p>
```

Click to copy



Now, we will create a Kubernetes deployment.

A *Deployment* provides declarative updates for Pods and ReplicaSets.

You describe a *desired state* in a Deployment, and the Deployment Controller changes the actual state to the desired state at a controlled rate. You can define Deployments to create new ReplicaSets, or to remove existing Deployments and adopt all their resources with new Deployments.

We will begin these lab steps by first disabling Pod Security Policies.

Pod Security Policies (PSPs) are sometimes used to limit what access pods have within a Kubernetes cluster. For example, PSPs can be used to ensure Pods don't have sudo access within the Kubernetes nodes. We disable Pod Security Polices as they have been deprecated and we are not covering them in this course. Some versions of TKG have them enabled by default.

13. In Windows Terminal run the following commands to disable PSP and deploy 3 nginx pods as part of your 1st deployment:

```
<p>cd ..\Deployments\
ls</p>
```

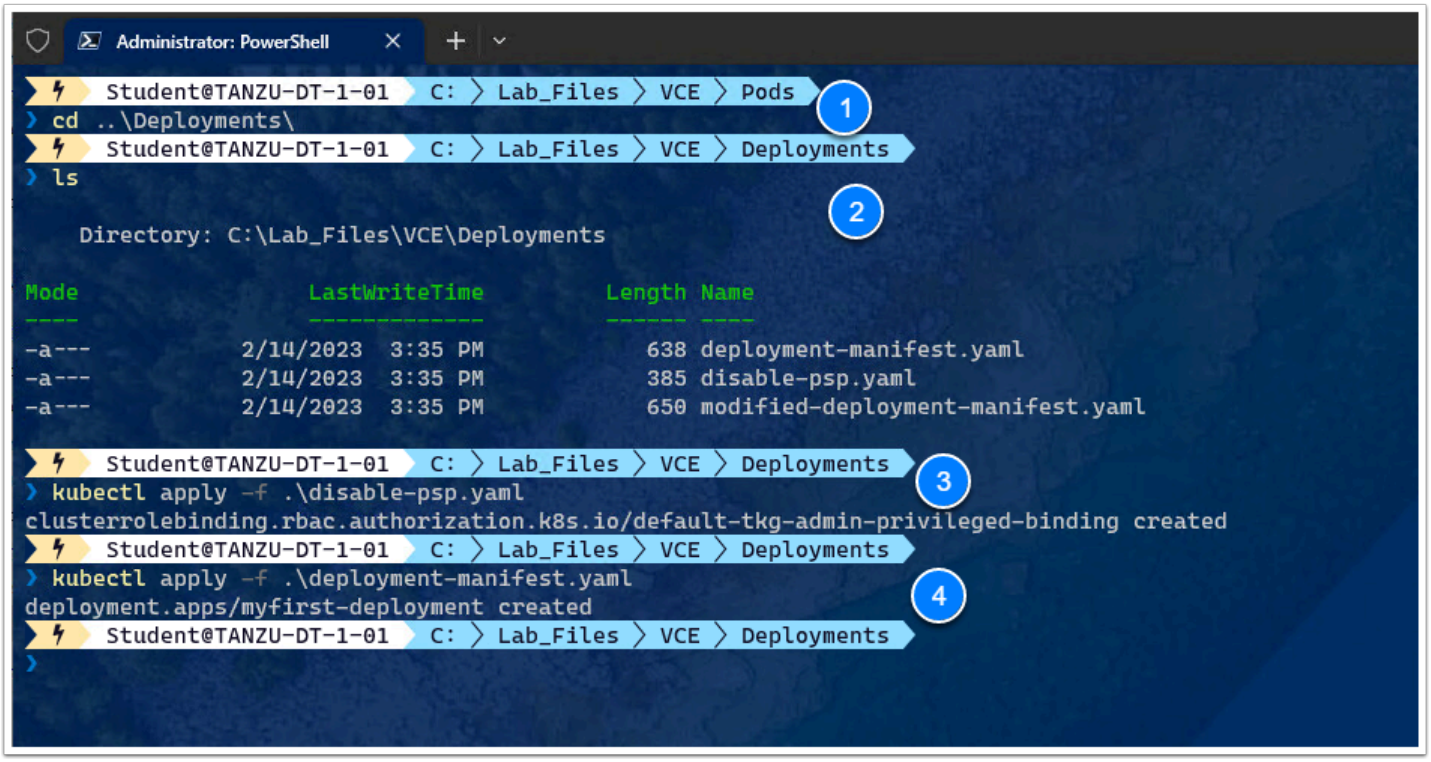
Click to copy

```
<p>kubectl apply -f .\disable-psp.yaml</p>
```

Click to copy

```
<p>kubectl apply -f .\deployment-manifest.yaml</p>
```

Click to copy



14. Now lets inspect the deployment and it's components with the following commands:

```
<p>kubectl get deployments</p>
```

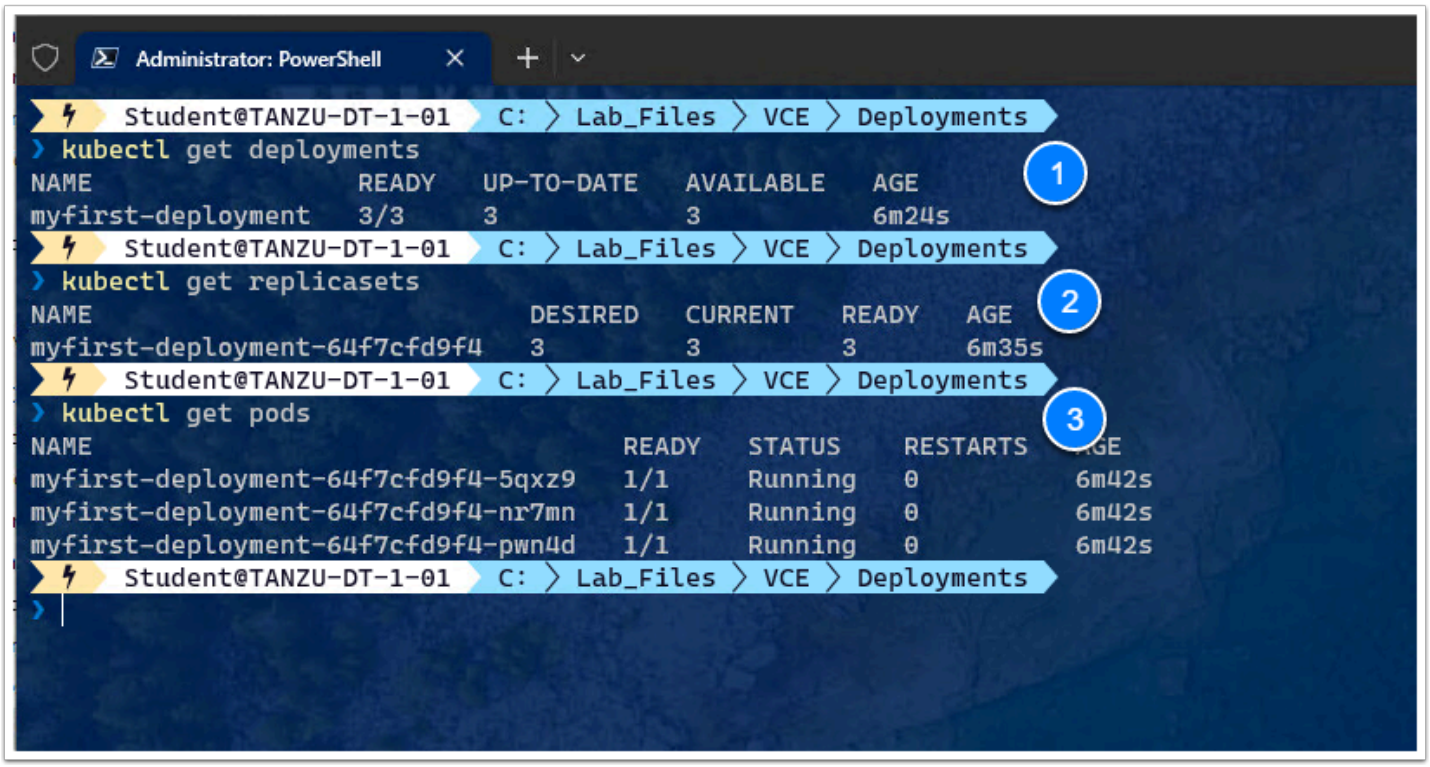
Click to copy

```
<p>kubectl get replicasets</p>
```

Click to copy

```
<p>kubectl get pods</p>
```

Click to copy



We will now modify the deployment and also see how we can rollback a previous version of the deployment

15. Modify the deployment and view its history and deployment details using the following commands

```
<p>kubectl apply -f .\modified-deployment-manifest.yaml</p>
```

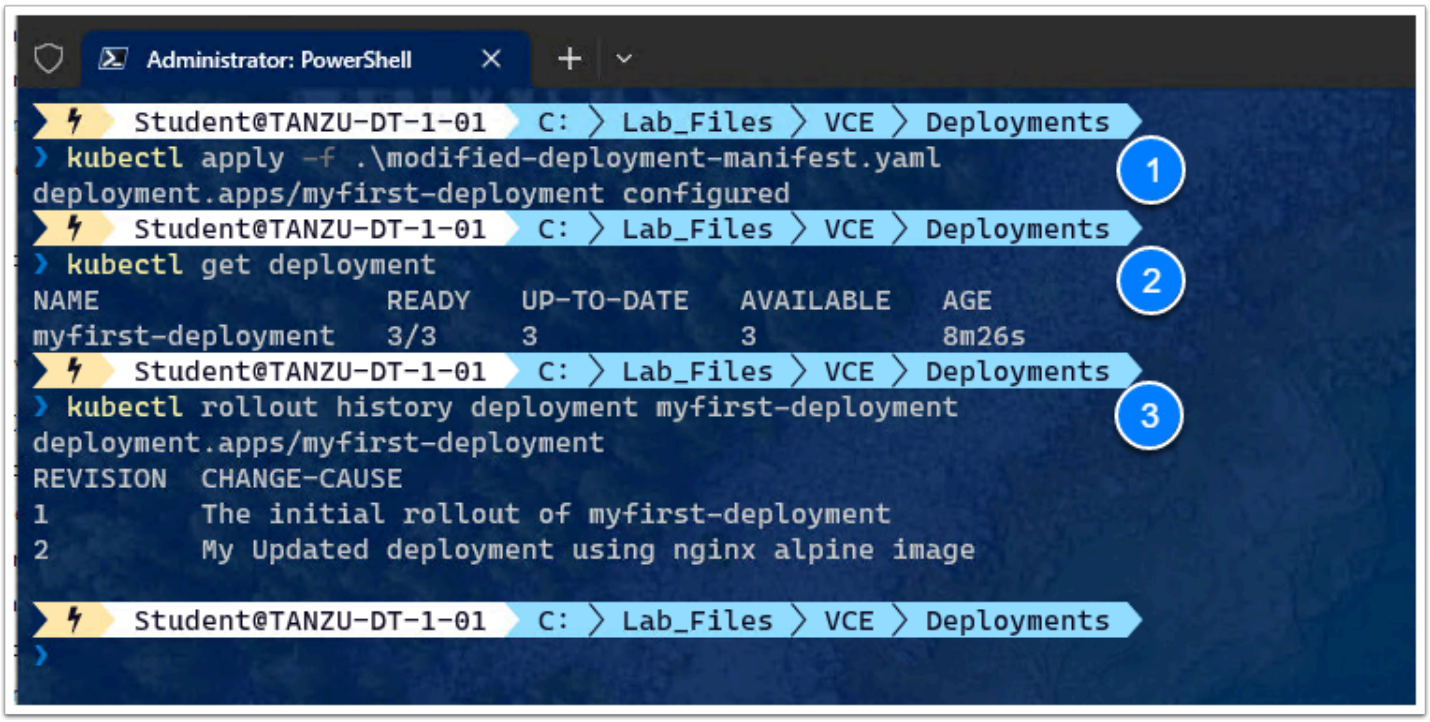
Click to copy

```
<p>kubectl get deployment</p>
```

Click to copy

```
<p>kubectl rollout history deployment myfirst-deployment</p>
```

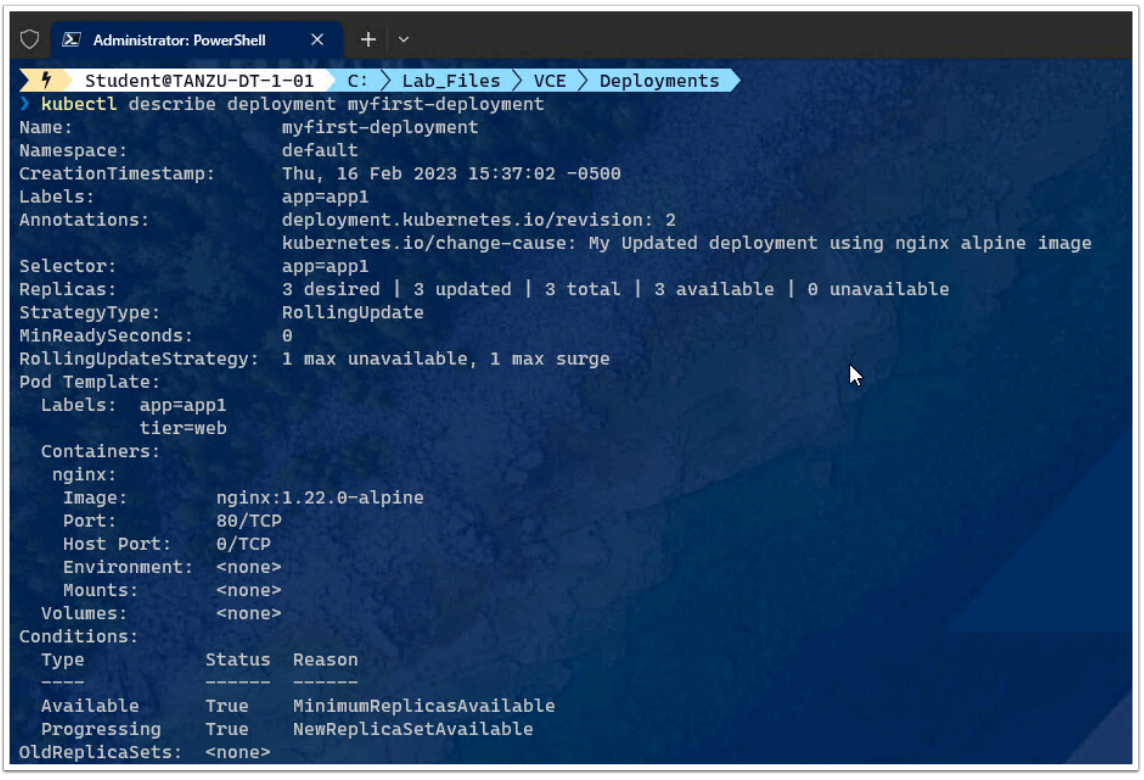
Click to copy



Let's retrieve further information about this deployment by using the following command

```
<p>kubectl describe deployment myfirst-deployment</p>
```

Click to copy



We will now rollback the deployment to the previous version and clean up the deployment once done.

16. Run the following commands to inspect the version history and rollback to a previous version

```
<p>kubectl rollout history deployment myfirst-deployment</p>
```

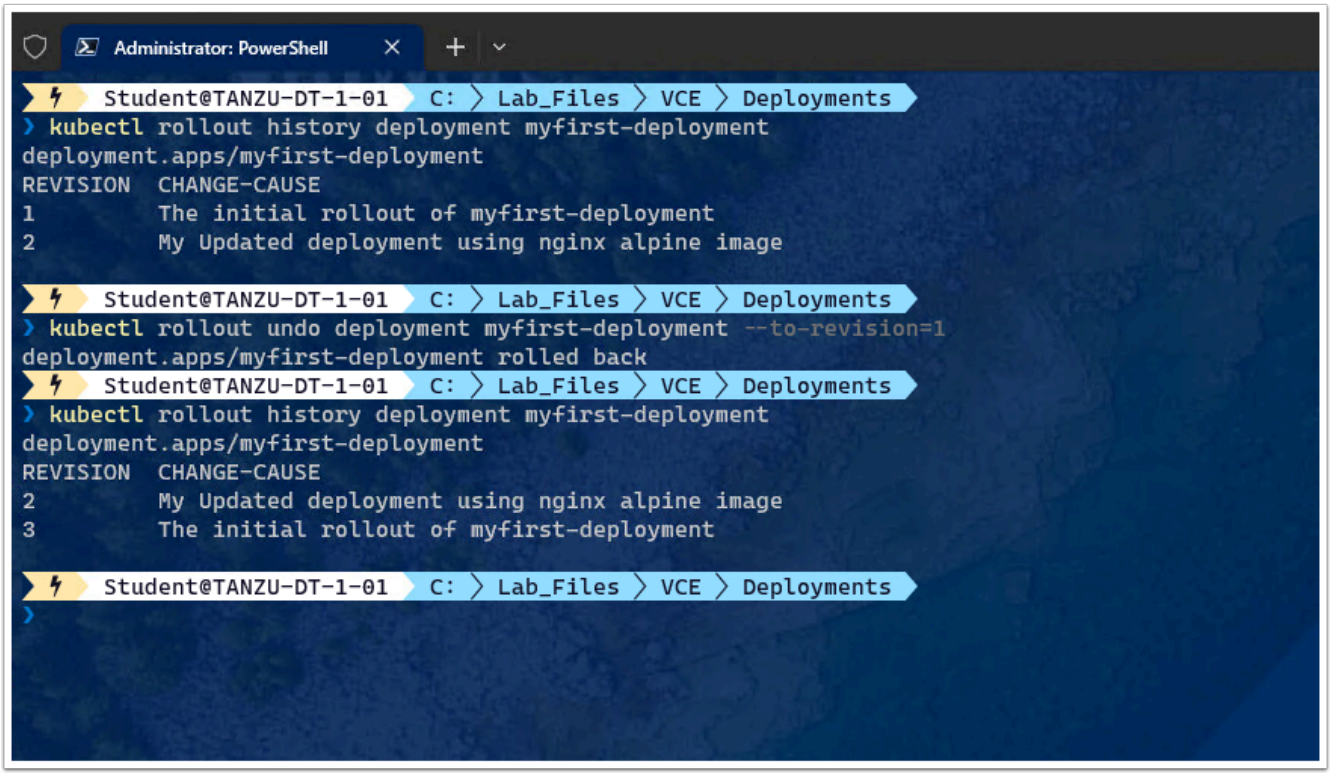
Click to copy

```
<p>kubectl rollout undo deployment myfirst-deployment --to-revision=1</p>
```

Click to copy

```
<p>kubectl rollout history deployment myfirst-deployment</p>
```

Click to copy



17. Run the following commands to cleanup your deployment and verify the cleanup removed your deployment

```
<p>kubectl delete deployment myfirst-deployment</p>
```

Click to copy

```
<p>kubectl get deployments</p>
```

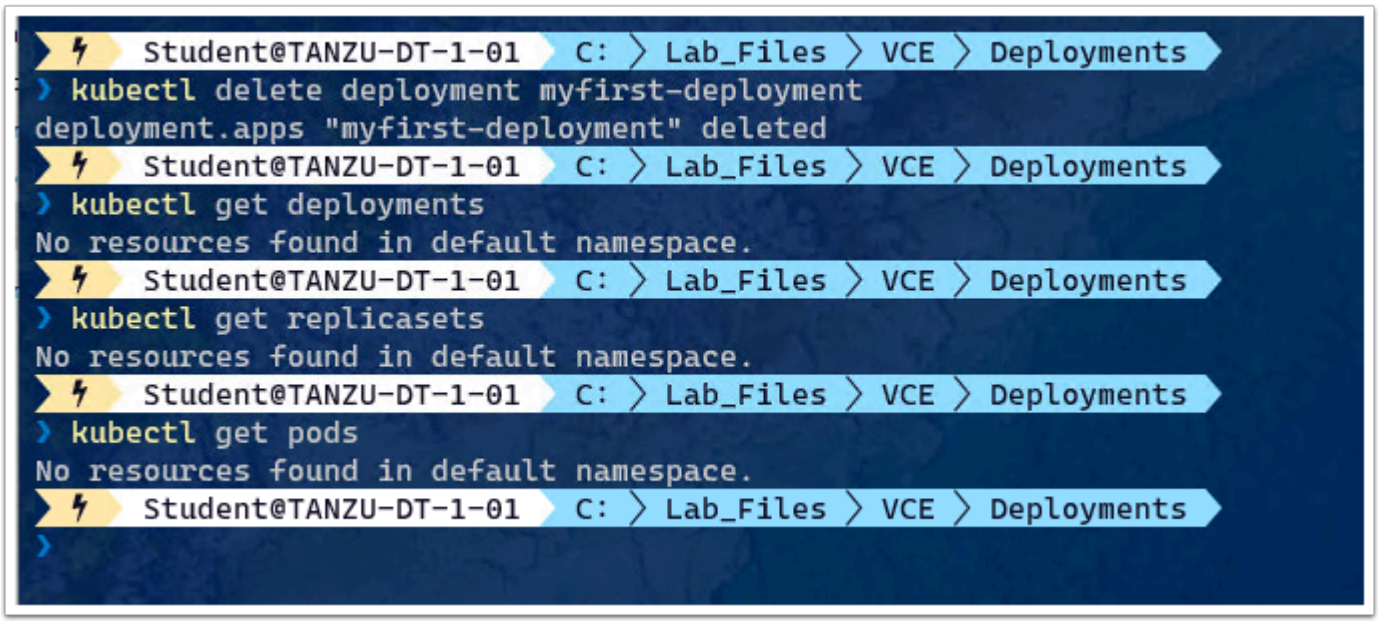
Click to copy

```
<p>kubectl get replicasets</p>
```

Click to copy

```
<p>kubectl get pods</p>
```

Click to copy



Task 4 - Working with Services & Load Balancers

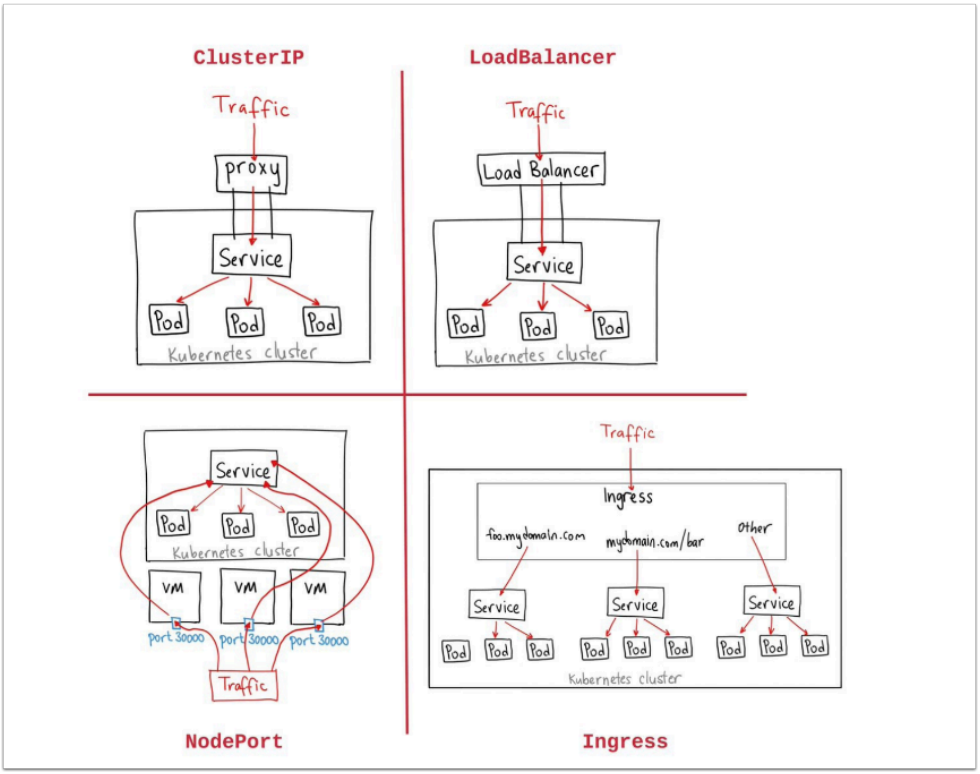
A **Kubernetes service** is a logical abstraction for a deployed group of pods in a cluster (which all perform the same function).

Since pods are ephemeral, a service enables a group of pods, which provide specific functions (web services, image processing, etc.) to be assigned a name and unique IP address (clusterIP). As long as the service is running that IP address, it will not change. Services also define policies for their access.

Difference between a deployment and a service

In Kubernetes, a deployment is a method of launching a pod with containerized applications and ensuring that the necessary number of replicas is always running on the cluster.

On the other hand, a service is responsible for exposing an interface to those pods, which enables network access from either within the cluster or between external processes and the service.



Kubernetes services connect a set of pods to an abstracted service name and IP address. Services provide discovery and routing between pods. For example, services connect an application front-end to its backend, each of which runs in separate deployments in a cluster. Services use labels and selectors to match pods with other applications. The core attributes of a Kubernetes service are:

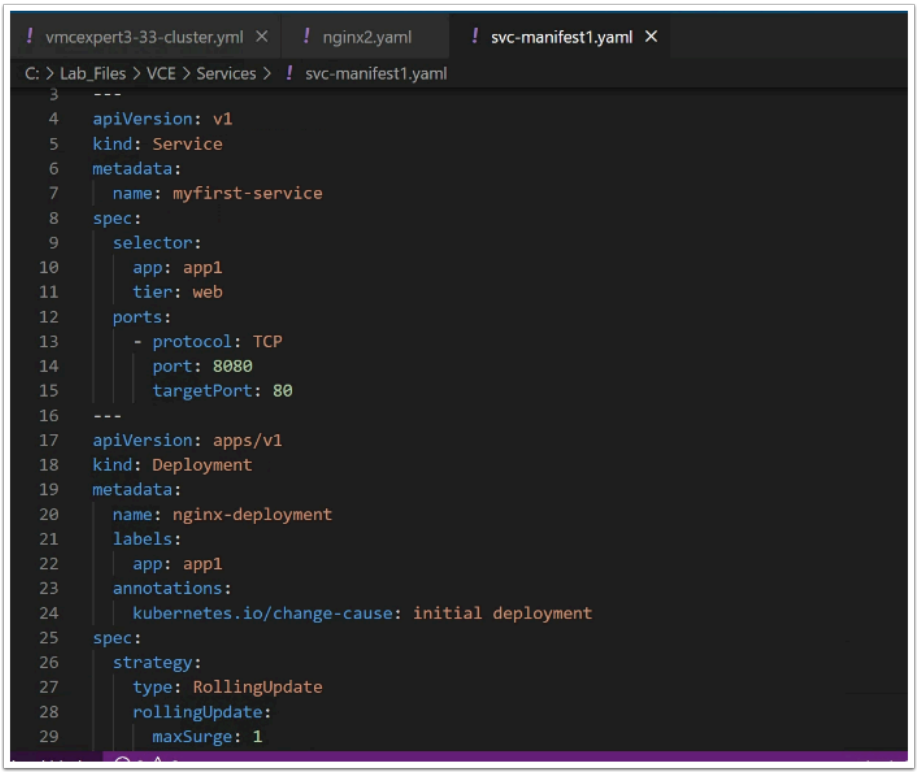
- A label selector that locates pods
- The clusterIP IP address and assigned port number
- Port definitions
- Optional mapping of incoming ports to a targetPort

Services can be defined without pod selectors. For example, to point a service to another service in a different namespace or cluster.

Service Types

- **ClusterIP**. Exposes a service that is only accessible from within the cluster.
- **NodePort**. Exposes a service via a static port on each node's IP.
- **LoadBalancer**. Exposes the service via the cloud provider's load balancer.
- **ExternalName**. Maps a service to a predefined externalName field by returning a value for the CNAME record.

1. In Windows Explorer, navigate to **C:\Lab_Files\VCE\Services**
2. Open the **svc-manifest1.yaml** file and examine its content



3. In Windows Terminal, execute the following commands to deploy the pods and service. Also, review the service deployment

```
<p>cd ..\services
ls</p>
```

Click to copy

```
<p>kubectl apply -f .\svc-manifest1.yaml</p>
```

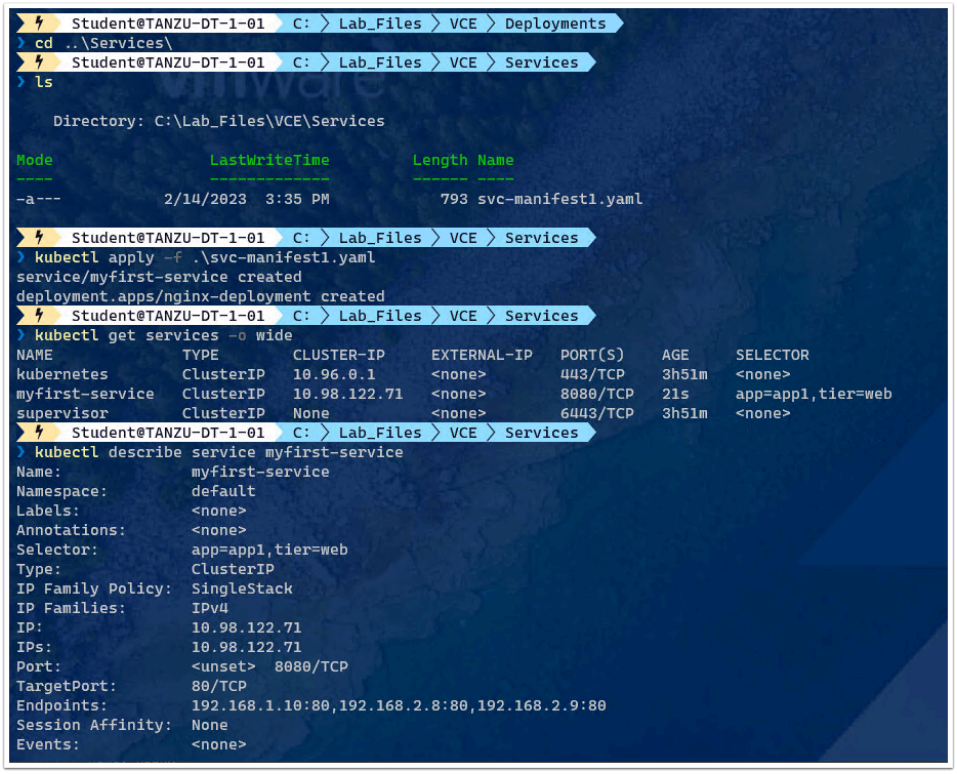
Click to copy

```
<p>kubectl get services -o wide</p>
```

Click to copy

```
<p>kubectl describe service myfirst-service</p>
```

Click to copy



4. Now, we will view the pods and delete one of them to observe what happens. Execute the following commands to do so:

```
<p>kubectl get endpoints myfirst-service</p>
```

Click to copy

```
<p>kubectl get pods</p>
```

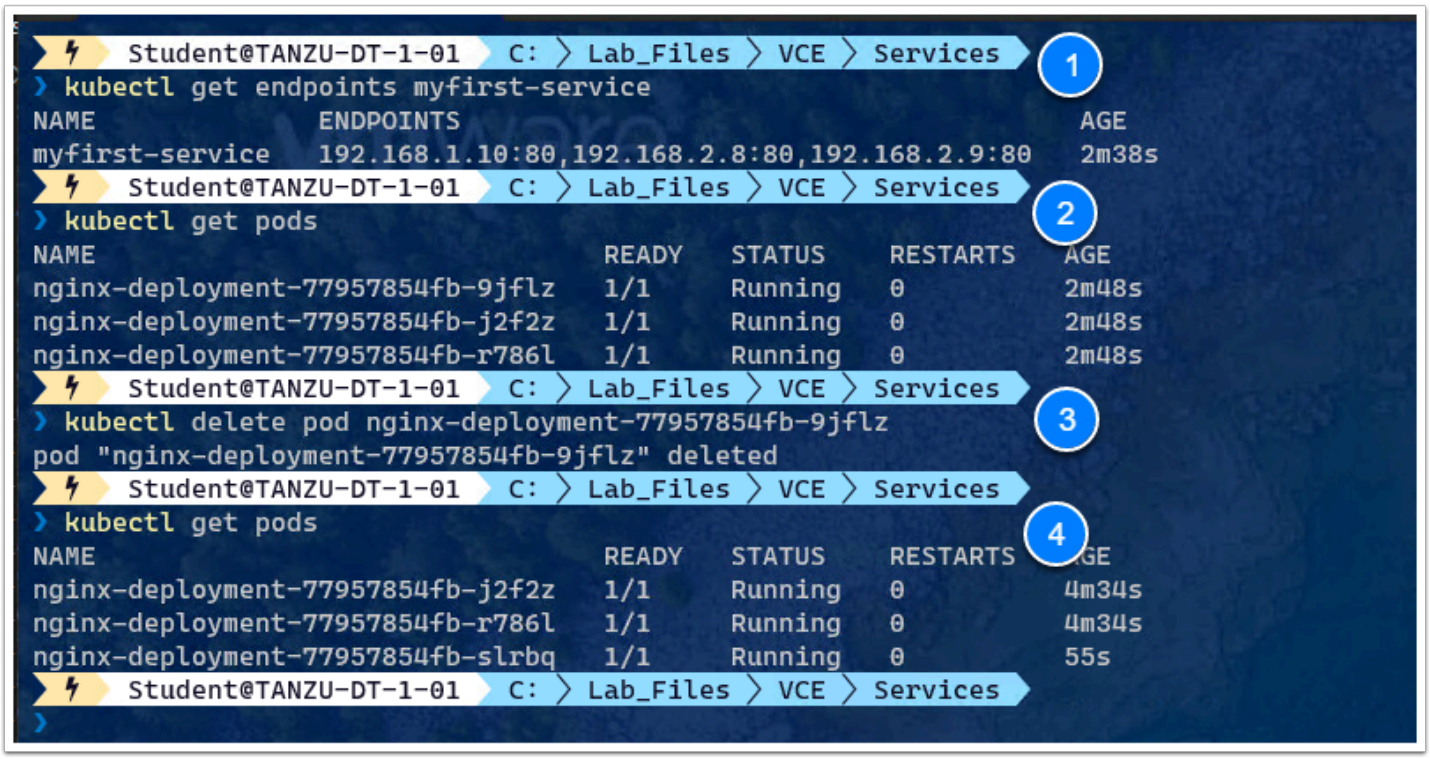
Click to copy

```
<p>kubectl delete pod {the_name_of _one of_your_nginx-pods}</p>
```

Click to copy

```
<p>kubectl get pods</p>
```

Click to copy



Question: How does deleting a pod affect the cluster?

- Did a new pod get created to replace that deleted pod?
- How did the endpoints change?
- How might this affect access from other applications?

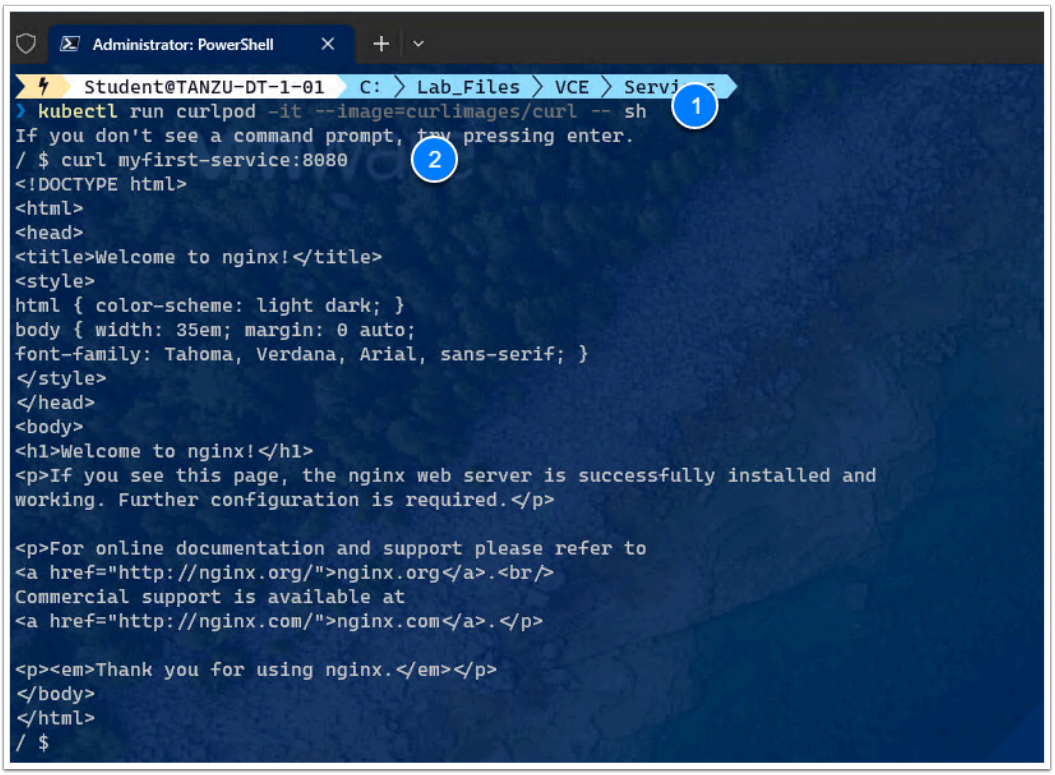
5. Now let's run a container that has the curl command installed in the image. Let's use the following imperative commands to deploy a curl container and exec into a shell

```
<p>kubectl run curlpod -it --image=curlimages/curl -- sh</p>
```

Click to copy

```
<p>curl myfirst-service:8080</p>
```

Click to copy



Now, let's perform a cleanup before moving forward. To do so we will exit Curl, delete the service and any containers. Execute the following commands:

```
<p>exit</p>
```

Click to copy

```
<p>kubectl get services
kubectl get pods</p>
```

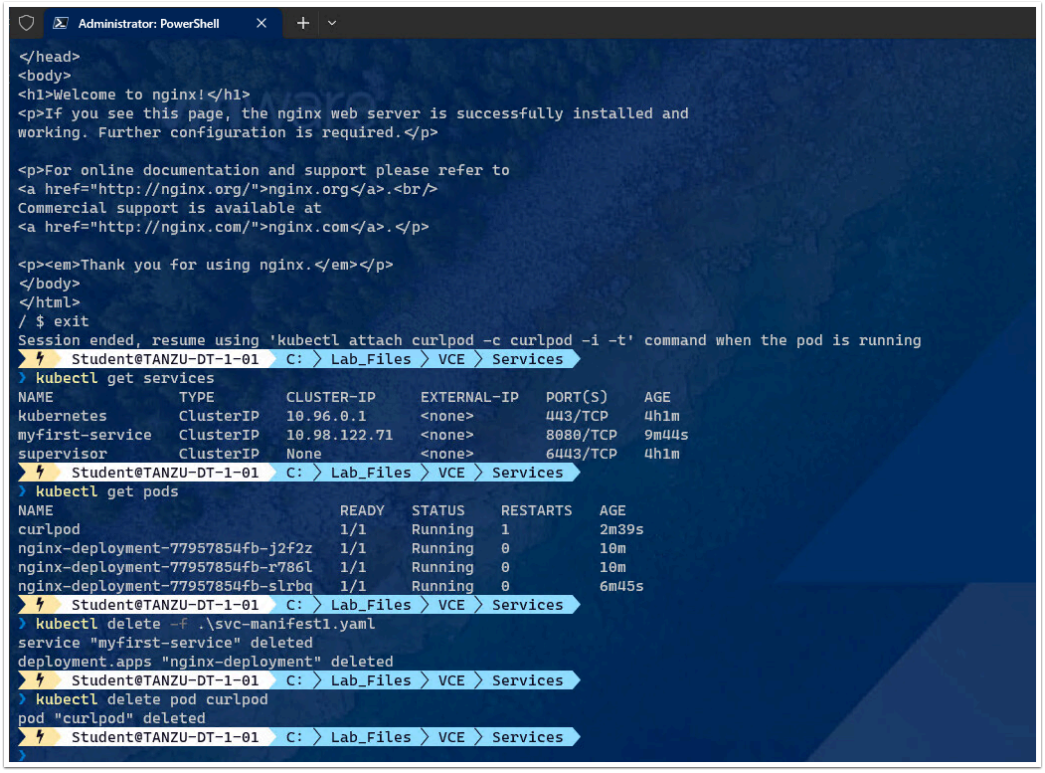
Click to copy

```
<p>kubectl delete -f .\svc-manifest1.yaml</p>
```

Click to copy

```
<p>kubectl delete pod curlpod</p>
```

Click to copy



Task 5 - Load Balancer Service

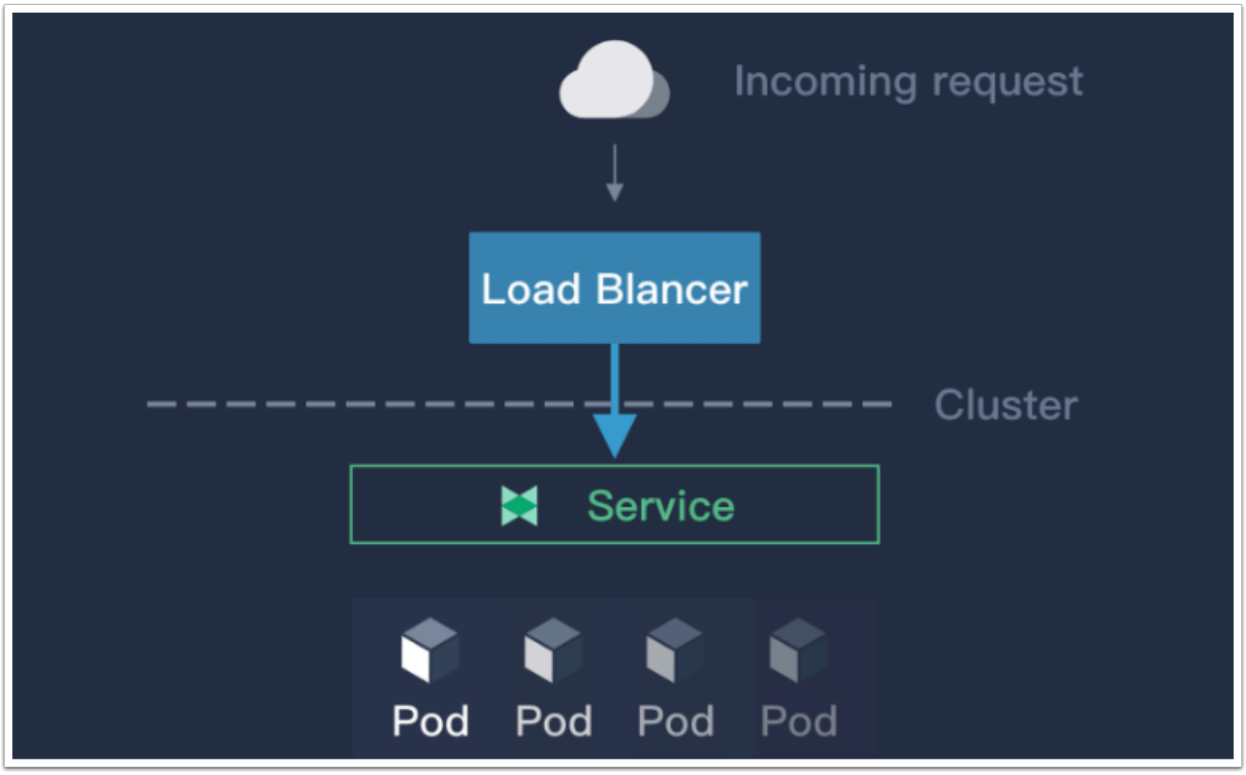
A core strategy for maximizing availability and scalability, load balancing distributes network traffic among multiple backend services efficiently. A range of options for load-balancing external traffic to pods exists in the Kubernetes context, each with its own benefits and trade-offs.

Load distribution is the most basic type of load balancing in Kubernetes. At the dispatch level load distribution is easy to implement. Each of the two methods of load distribution that exist in Kubernetes operates through the kube-proxy feature. Services in Kubernetes use the virtual IPs which the kube-proxy feature manages.

P addresses for Kubernetes pods are not persistent because the system assigns each new pod a new IP address. Typically, therefore, direct communication between pods is impossible. However, services have their own relatively stable IP addresses which field requests from external resources. The service then dispatches the request to an available Kubernetes pod.

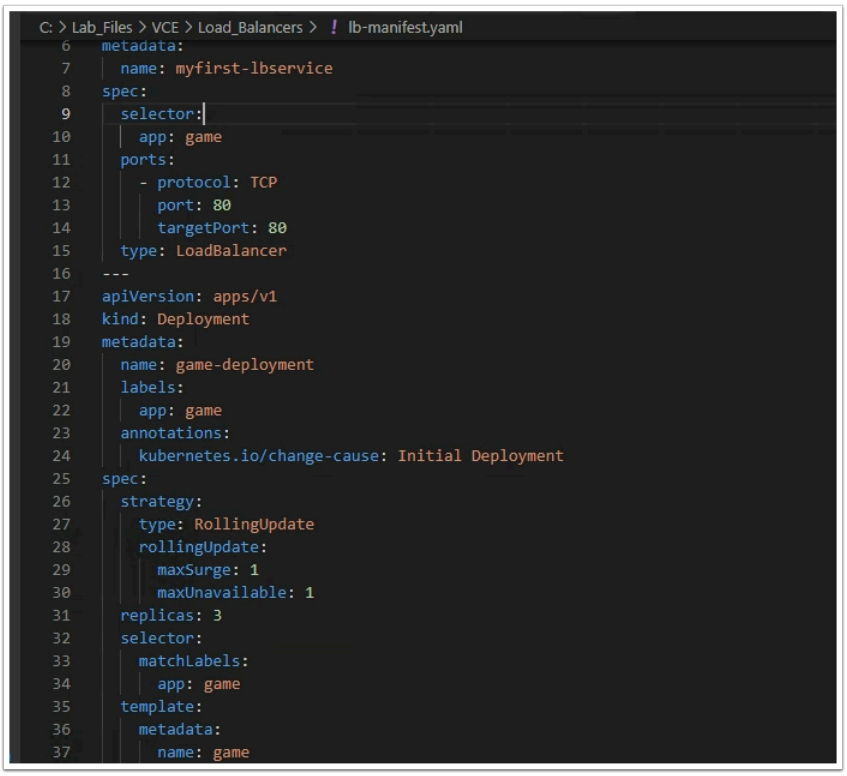
Kubernetes load balancing makes the most sense in the context of how Kubernetes organizes containers. Kubernetes does not view single containers or individual instances of

a service, but rather sees containers in terms of the specific services or sets of services they perform or provide.



In this task, we will use a supplied YAML manifest to provision a deployment and a Load Balancer

1. In Windows Explorer, navigate to **C:\Lab_Files\VCE\Load_Balancers**
2. Open the **lb-manifest.yam**l file and review its content



3. In Windows Terminal, execute the following command to provision and investigate the deployment:

```
<p>cd ..\Load_Balancers
ls</p>
```

Click to copy

```
<p>kubectl apply -f .\lb-manifest.yaml</p>
```

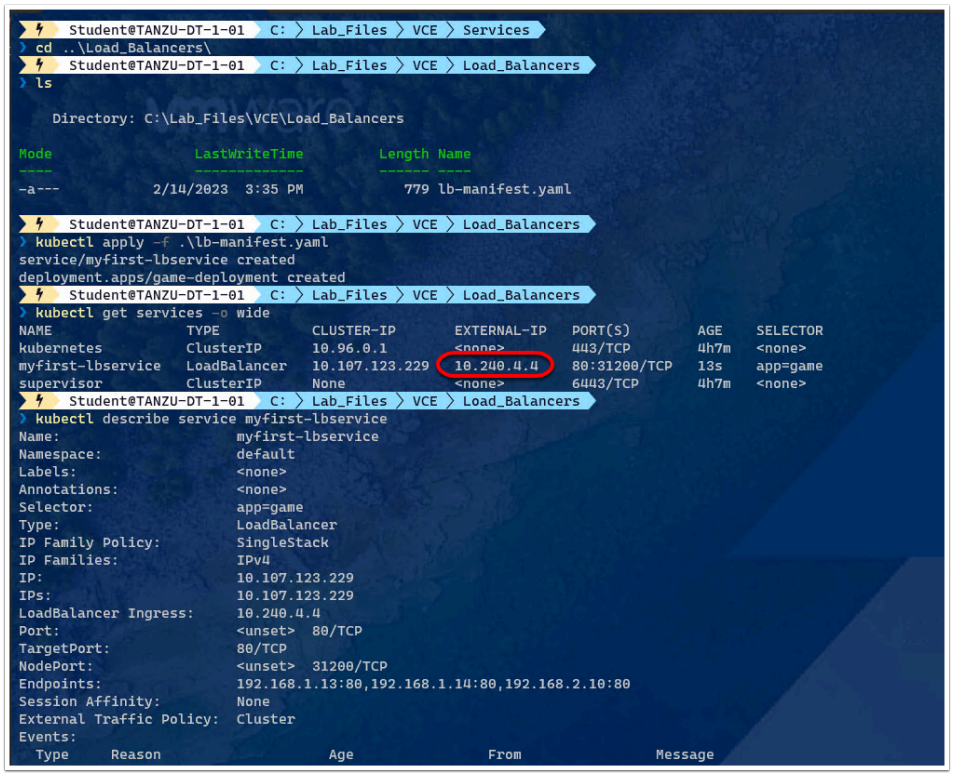
Click to copy

```
<p>kubectl get services -o wide</p>
```

Click to copy

```
<p>kubectl describe service myfirst-lbservice</p>
```

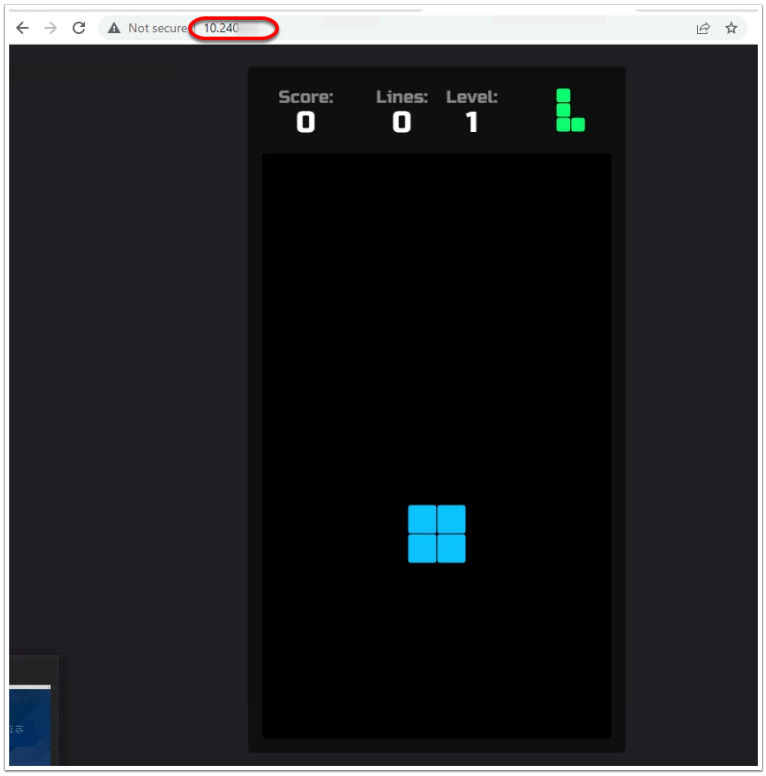
Click to copy



Questions:

1. What is the Cluster IP of the Service
2. What is the External IP of the Service
3. Which Port is the NodePort running on

4. From the Tanzu desktop, open a browser instance and navigate to the external IP of the Load Balancer service



5. Try out your Tetris skills
6. Delete the deployments, replica sets, pods and services by executing the following command

```
<p>kubectl delete -f .\lb-manifest.yaml</p>
```

Click to copy

Conclusion

Pods allow you to deploy closely coupled components together as separate containers. For instance, you can bundle an app and a proxy for that app that adds an encryption layer together so encrypted traffic goes in and out of the app without modifying the app container.

Pods in a Kubernetes cluster can be used in two main ways:

Pods that run a single container. The “one-container-per-Pod” model is the most common Kubernetes use case; in this case, you can think of a Pod as a wrapper around a single container, and Kubernetes manages the Pods rather than the containers directly.

Pods that run multiple containers that need to work together. A Pod might encapsulate an application composed of multiple co-located containers that are tightly coupled and need to share resources. These co-located containers might form a single cohesive unit of service–one container serving files from a shared volume to the public, while a separate “sidecar” container refreshes or updates those files. The Pod wraps these containers and storage resources together as a single manageable entity.