Lab Module 6: Advanced Kubernetes Topics - Part 1

> Estimated Duration: 60 minutes

Create a Basic AKS Cluster

For the exercises in this module, you'll need simple AKS cluster.

Task 1 - Create an AKS cluster (or start an existing one)

1. Select the region closest to your location. Use '**eastus**' for United States workshops, '**westeurope**' for European workshops.

2. Define variables (update as needed)

```
$INITIALS="abc"
```

```
$YOUR_INITIALS="$($INITIALS)".ToLower()
$AKS_RESOURCE_GROUP="azure-$($INITIALS)-rg"
$VM_SKU="Standard_D2as_v5"
$AKS_NAME="aks-$($INITIALS)"
$NODE_COUNT="3"
$LOCATION="@lab.Variable(region)"
```

3. Create Resource Group

```
az group create --location $LOCATION `
                --resource-group $AKS_RESOURCE_GROUP
```

4. Create Basic cluster.

```
az aks create --node-count $NODE_COUNT `
              --generate-ssh-keys `
              --node-vm-size $VM_SKU `
              --name $AKS_NAME `
              --resource-group $AKS_RESOURCE_GROUP
```

5. Connect to local environment

```
az aks get-credentials --name $AKS_NAME `
                       --resource-group $AKS_RESOURCE_GROUP
```

6. Verify connection

```
kubectl get nodes
```

Exercise: Using Secrets Stored in Azure Key Vault

In this Azure Key Vault exercise, you'll perform the following actions:

1. Enable Key Vault Addon in AKS - Azure installs all the components need to integrate Key Vault with AKS
2. Create an Azure Key Vault - This will contain all the secrets you'll use
3. Grant `administrator` permissions you your account - This will allow you to create secrets in the Key Vault.
4. Create a secret in Azure Key Vault - This will represent the sensitive data you should keep outside the cluster.
5. Great `reader` permissions to the AKS cluster - This will allow the cluster to read the external secret
6. Create custom resources in your cluster to establish the connection to Key Vault - This will specify the AKS identity, Key Vault and secret that will be injected into your Pod.
7. Mount a CSI secrets volume in your Pod - This will use the information in custom resource retrieve and mount your secret.

Task 1 - Enable Key Vault Addon in AKS and create a Key Vault

1. Define variables.

```
$INITIALS="abc"
```

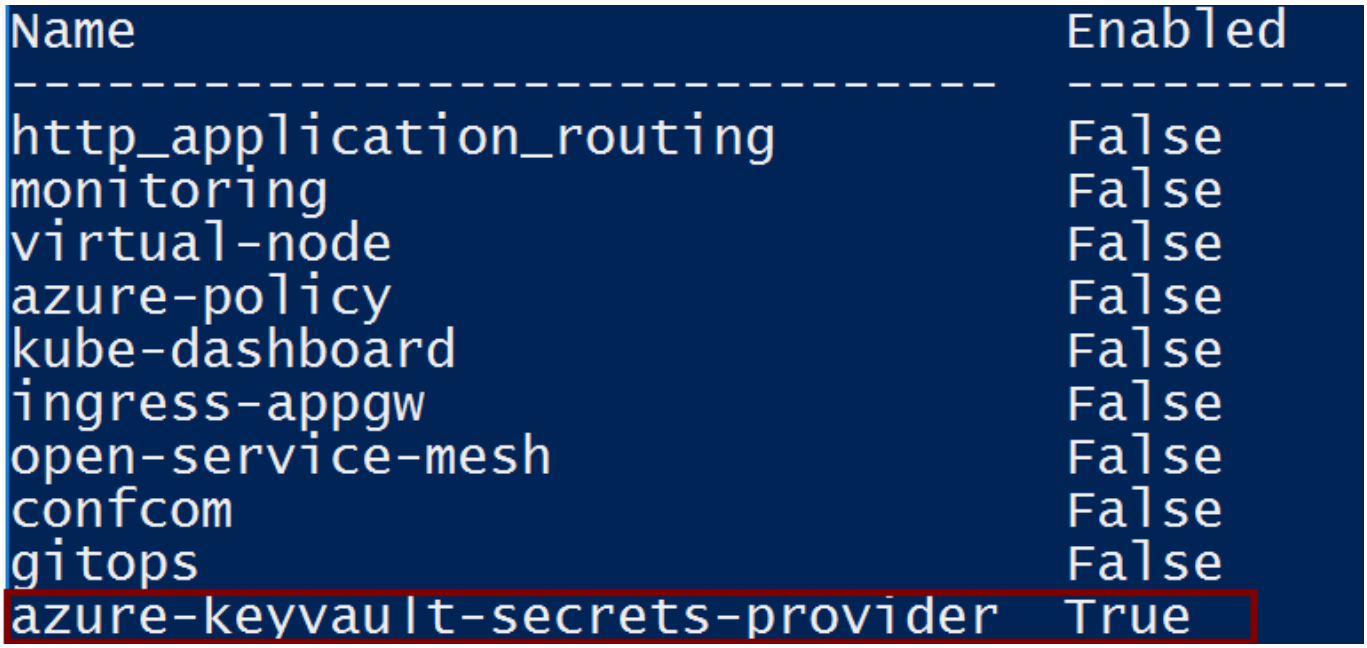
```
$YOUR_INITIALS="$($INITIALS)".ToLower()
$AKS_RESOURCE_GROUP="azure-$($INITIALS)-rg"
$LOCATION="@lab.Variable(region)"
$AKS_IDENTITY="identity-$($INITIALS)"
$AKS_NAME="aks-$($INITIALS)"
$KV_NAME="kv-$($INITIALS)"
```

2. Enable Key Vault Addon in AKS

```
az aks addon enable `
   --addon azure-keyvault-secrets-provider `
   --name $AKS_NAME `
   --resource-group $AKS_RESOURCE_GROUP
```

3. Verify addon has been enabled

```
az aks addon list --name $AKS_NAME --resource-group $AKS_RESOURCE_GROUP -o table
```



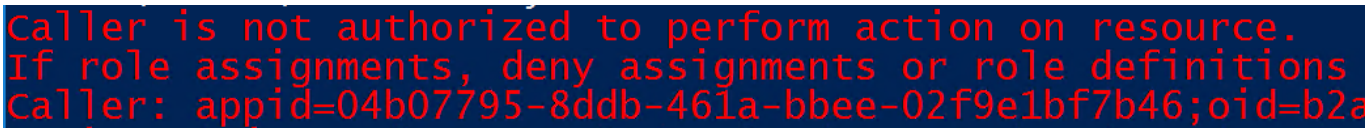
4. Create Key Vault. Notice the **--enable-rbac-authorization** option. This will allow AKS to use its managed identity to access the Key Vault.

```
az keyvault create --name $KV_NAME `
   --resource-group $AKS_RESOURCE_GROUP `
   --enable-rbac-authorization
```

Task 2 - Assign Permissions for you to create Secrets from Key Vault

1. List the secrets in the Key vault.

```
az keyvault secret list --vault-name $KV_NAME
```



Even though you created the Key Vault, you don't automatically have access to the Data Plane of the vault. You must give yourself additional permissions first.

You can assign yourself as the **Key Vault Secrets Officer**, which will allow you to maintain secrets. You may wish to use **Key Vault Administrator** instead, which will allow you create Keys and Certificates as well.

2. Get the `object id` of the Key Vault:

```
$KV_ID=(az keyvault list --query "[? name=='$($KV_NAME)'].{id:id}" -o tsv)
```

3. Get `your` object id:

```
$CURRENT_USER_ID=(az ad signed-in-user show --query "{objectId:objectId}" -o tsv)
```

4. Assign yourself the Key Vault Secrets Officer/Administrator role:

```
az role assignment create `
   --role "Key Vault Secrets Officer" `
   --assignee-object-id $CURRENT_USER_ID `
   --scope $KV_ID
```

5. List the secrets in the Key Vault again. This time you should get an empty list instead of "not authorized" error.

```
az keyvault secret list --vault-name $KV_NAME
```

6. Create a sample secret for testing

```
az keyvault secret set `
   --vault-name $KV_NAME `
   --name SampleSecret `
   --value "Highly sensitive information: <place data here>"
```

7. Verify the secret is in the Key Vault

```
az keyvault secret list --vault-name $KV_NAME -o table
```

You should now see the secret listed.

Task 3 - Assign Permissions to AKS to read Secrets from Key Vault

You'll need to assign AKS the **Key Vault Secrets User** role so it's able to read the secrets you created.

1. List all available identities. Find the managed identity created for accessing the Key Vault from AKS when the addon was enabled.

```
az identity list --query "[].{name:name,ClientId:clientId}" -o table
```

```
Name                                       ClientId
-----------------------------------------  ------------------------------------
identity-kiz                               ccced939-b36e-4062-b630-ad4851075ee5
aks-76782343-agentpool                     d6f41dfd-4030-481d-9074-553d15d90847
azurekeyvaultsecretsprovider-aks-76782343  b32ab48d-e972-48b6-a26b-f7bb8f164169
```

2. Get the object id of that managed identity AKS is using

```
$KV_IDENTITY=(az identity list --query "[?
contains(name,'azurekeyvaultsecretsprovider')].{principalId:principalId}" -o tsv)
$KV_CLIENT_ID=(az identity list --query "[?
contains(name,'azurekeyvaultsecretsprovider')].{clientId:clientId}" -o tsv)
```

3. Assign the AKS identity permission to read secrets from the Key Vault.

```
az role assignment create `
   --role "Key Vault Secrets User" `
   --assignee-object-id $KV_IDENTITY `
   --scope $KV_ID
```

Task 4 - Create Kubernetes resources and read secret from Key Vault

1. Gather the needed values

```
$TENANT_ID=(az aks show --name $AKS_NAME --resource-group $AKS_RESOURCE_GROUP --
query "{tenantId:identity.tenantId}" -o tsv)

write-host @"
Client Id: $($KV_CLIENT_ID)
Key Vault Name: $($KV_NAME)
Tenant Id: $($TENANT_ID)
"@
```

2. Change current folder to **Module6**

```
cd  C:\k8s\labs\Module6
```

3. Open the file called called **spc.yaml**.

```
code spc.yaml
```

4. Replace the placeholders with the values listed above.

```
apiVersion: secrets-store.csi.x-k8s.io/v1
kind: SecretProviderClass
metadata:
  name: azure-kv-secret
spec:
```

```
  provider: azure
  parameters:
    usePodIdentity: "false"
    useVMManagedIdentity: "true"
    userAssignedIdentityID: <client-id>
    keyvaultName: <key-vault-name>
    cloudName: ""
    objects: |
      array:
        - |
          objectName: SampleSecret
          objectType: secret
          objectVersion: ""
    tenantId: <tenant-id>
```

5. Apply the manifest.

```
kubectl apply -f spc.yaml
```

6. Review the contents of **pod-kv.yaml**.

```
kind: Pod
apiVersion: v1
metadata:
  name: pod-kv
spec:
  containers:
    - name: busybox
      image: k8s.gcr.io/e2e-test-images/busybox:1.29-1
      command:
        - "/bin/sleep"
        - "10000"
      volumeMounts:
        - name: secrets-store01
          mountPath: "/mnt/secrets-store"
          readOnly: true
  volumes:
    - name: secrets-store01
      csi:
        driver: secrets-store.csi.k8s.io
        readOnly: true
        volumeAttributes:
          secretProviderClass: "azure-kv-secret"
```

7. Apply the manifest.

```
kubectl apply -f pod-kv.yaml
```

8. View secret value in Pod

```
kubectl exec -it pod-kv -- cat /mnt/secrets-store/SampleSecret
```

You should now be able to see the content of the Key Vault secret you created earlier.

Task 5 - Create a Kubernetes Secret from a secret in Key Vault

Many Kubernetes resources use `Secret` resources (Ingress, Persistent Volumes, etc.). You can extend the configuration about to create a Kubernetes secret object when you mount the Pod.

1. Edit the **SecretProviderClass** you created earlier.

```
code spc.yaml
```

Add the indicated section.

```
apiVersion: secrets-store.csi.x-k8s.io/v1
kind: SecretProviderClass
metadata:
  name: azure-kv-secret
spec:
  provider: azure

  ########## Add this section #########
  secretObjects:
  - data:
    - key: MySecret
      objectName: SampleSecret
    secretName: k8s-secret
    type: Opaque
  ########## End of section #########

  parameters:
    usePodIdentity: "false"
    useVMManagedIdentity: "true"
  ...
```

2. Update the object.

```
kubectl apply -f spc.yaml
```

3. Edit the Pod manifest. Notice the `env` section setting the environment variable.

```
code pod-kv.yaml
```

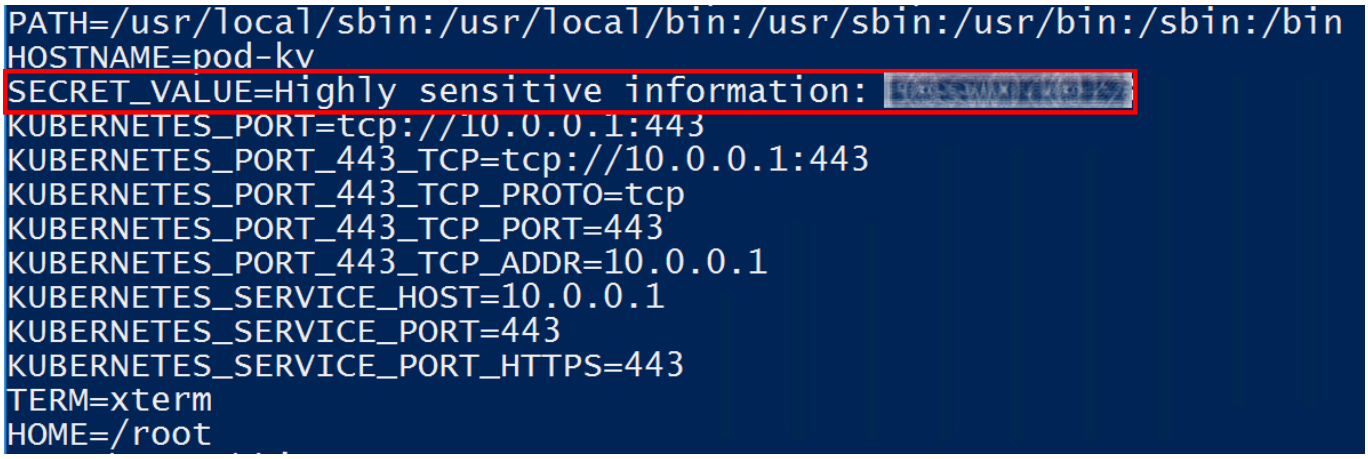
```
kind: Pod
apiVersion: v1
metadata:
  name: pod-kv
spec:
  containers:
    - name: busybox
      image: k8s.gcr.io/e2e-test-images/busybox:1.29-1
      command:
        - "/bin/sleep"
        - "10000"
      volumeMounts:
        - name: secrets-store01
          mountPath: "/mnt/secrets-store"
          readOnly: true

      ########## Add this section #########
      env:
        - name: SECRET_VALUE
          valueFrom:
            secretKeyRef:
              name: k8s-secret
              key: MySecret
      ########## End of section #########

  volumes:
    - name: secrets-store01
      csi:
        driver: secrets-store.csi.k8s.io
        readOnly: true
        volumeAttributes:
          secretProviderClass: "azure-kv-secret"
```

4. Since Pods are immutable, you'll have to delete it and then reapply the manifest.

```
kubectl delete pod pod-kv
kubectl apply -f pod-kv.yaml
```

5. View the secret in the Pod.

```
kubectl exec -it pod-kv -- printenv
```



6. Verify the Secret object has been created and is available for other Pods to use.

```
kubectl get secret
```

7. Delete the Pod and the Kubernetes Secret object will also be deleted. Once an injected Secret is no longer referenced by any Pods, it's automatically deleted.

```
kubectl delete pod pod-kv
```

```
kubectl get secret
```

Once an injected Secret is no longer referenced by any Pods, it's automatically deleted.

Exercise: Using Taints, Tolerations, Node Selectors and Affinity

In this exercise, you will work with 3 simple workloads of 12 Pods each. You'll ensure that "heavy" workloads are scheduled on Nodes that can handle the loads and "light" workloads don't clutter up those nodes and use up valuable resources which could be used by the Pods that need them.

Scenario

We assume you have 3 nodes in your cluster. If you don't, please create and AKS cluster with at least 3 nodes.

Assume the following about your AKS cluster:

1. One of the nodes has specialized hardware (GPUs) which make it ideal for hosting graphics processing workloads and for Machine Learning.
2. The other 2 nodes have basic hardware and are meant for general purpose processing.
3. You have 3 workloads you need to schedule in the cluster:

- Workload 1 – Makes use of heavy GPU hardware for processing tasks.
- Workload 2 – General purpose
- Workload 3 – General purpose

The goal of this exercise is to make sure that:

4. Workload 1 Pods are scheduled ONLY on Node 1, so they can take full advantage of the hardware.
5. Workloads 2 & 3 are NOT scheduled on Node 1, so they don't use up resources which could be used by additional replicas of Workload 1.

You'll start with 3 workload `yaml` files. The containers in them are identical, but for the purpose of this lab, you'll pretend and **`workload1.yaml`** does extensive processing that takes advantage of a GPU in the Node.

Task 1 – Add workloads to the cluster

1. Open a Window Terminal window and apply the following workloads:

```
kubectl apply -f .\workload-1.yaml
kubectl apply -f .\workload-2.yaml
kubectl apply -f .\workload-3.yaml
```

2. Verify that the Pods are running

```
kubectl get pods
```

```
workload-1-5c98b9959f-97nnd   1/1     Running   0          17s
workload-1-5c98b9959f-9pr7x   1/1     Running   0          17s
workload-1-5c98b9959f-kbnfc   1/1     Running   0          17s
workload-1-5c98b9959f-mjqxk   1/1     Running   0          17s
workload-1-5c98b9959f-qkd68   1/1     Running   0          17s
workload-1-5c98b9959f-tcpht   1/1     Running   0          17s
workload-2-74d6bffc66-748x5   1/1     Running   0          12s
workload-2-74d6bffc66-c2d5p   1/1     Running   0          12s
workload-2-74d6bffc66-g6trp   1/1     Running   0          12s
workload-2-74d6bffc66-lz4b4   1/1     Running   0          12s
workload-2-74d6bffc66-q2f7b   1/1     Running   0          12s
workload-2-74d6bffc66-tvm7p   1/1     Running   0          12s
workload-3-7b6d554d-67lg2     1/1     Running   0          5s
workload-3-7b6d554d-8dg5p     1/1     Running   0          5s
workload-3-7b6d554d-kftvq     1/1     Running   0          5s
workload-3-7b6d554d-pfkkg     1/1     Running   0          5s
workload-3-7b6d554d-s2wr9     1/1     Running   0          5s
workload-3-7b6d554d-tg7zc     1/1     Running   0          5s
```

Task 2 – Add color and label to node

1. Get a list of the Nodes.

```
kubectl get nodes
```

2. Pick on of the Nodes to be the "GPU" Node (it doesn't matter which one). Copy it's name by clicking the Right button on your mouse

```
NAME                                STATUS   ROLES   AGE
aks-agentpool-13458657-vmss000000   Ready    agent   45d
aks-userpool1-20365992-vmss000001   Ready    agent   10d
aks-userpool1-20365992-vmss00000k   Ready    agent   19h
```

3. Set a variable to the Node name you selected (replace the NodeName with your selected node)

```
$NodeName="aks-userpool1-20365992-vmss000001"
```

4. Add a **color=lime** and **process=GPU** labels to the Node to distinguish it from the others and allow the Pods to find it..

```
kubectl label node $NodeName color=lime, process=GPU --overwrite
```

Task 3 – Update the Node Selector of the Workload

1. Edit the **workload-1.yaml** file to update the node selector to ensure it's only scheduled on the selected node.

```
      nodeSelector:
        kubernetes.io/os: linux
        process: GPU
```

2. Save and apply workload1.yaml again.

```
kubectl apply -f .\workload-1.yaml
```

3. Verify that the Pods have been rescheduled on the selected node.

```
kubectl get pods -o wide
```

```
NAME                          READY   STATUS    RESTARTS   AGE   IP             NODE
workload-1-7d7499b8f4-2cwql   1/1     Running   0          86s   172.17.1.115   aks-userpool1-20365992-vmss000001
workload-1-7d7499b8f4-4zltd   1/1     Running   0          86s   172.17.1.134   aks-userpool1-20365992-vmss000001
workload-1-7d7499b8f4-fmf9t   1/1     Running   0          84s   172.17.1.111   aks-userpool1-20365992-vmss000001
workload-1-7d7499b8f4-qkxpw   1/1     Running   0          86s   172.17.1.132   aks-userpool1-20365992-vmss000001
workload-1-7d7499b8f4-r45xk   1/1     Running   0          84s   172.17.1.133   aks-userpool1-20365992-vmss000001
workload-1-7d7499b8f4-zgk8n   1/1     Running   0          84s   172.17.1.114   aks-userpool1-20365992-vmss000001
workload-2-74d6bffc66-748x5   1/1     Running   0          10h   172.17.0.137   aks-userpool1-20365992-vmss00000k
workload-2-74d6bffc66-c2d5p   1/1     Running   0          10h   172.17.1.125   aks-userpool1-20365992-vmss000001
workload-2-74d6bffc66-g6trp   1/1     Running   0          10h   172.17.1.119   aks-userpool1-20365992-vmss000001
workload-2-74d6bffc66-lz4b4   1/1     Running   0          10h   172.17.0.98    aks-agentpool-13458657-vmss000000
workload-2-74d6bffc66-q2f7b   1/1     Running   0          10h   172.17.0.119   aks-userpool1-20365992-vmss00000k
workload-2-74d6bffc66-tvm7p   1/1     Running   0          10h   172.17.0.126   aks-userpool1-20365992-vmss00000k
workload-3-7b6d554d-67lg2     1/1     Running   0          10h   172.17.0.117   aks-userpool1-20365992-vmss00000k
workload-3-7b6d554d-8dg5p     1/1     Running   0          10h   172.17.0.103   aks-agentpool-13458657-vmss000000
workload-3-7b6d554d-kftvq     1/1     Running   0          10h   172.17.1.130   aks-userpool1-20365992-vmss000001
workload-3-7b6d554d-pfkkg     1/1     Running   0          10h   172.17.0.134   aks-userpool1-20365992-vmss00000k
workload-3-7b6d554d-s2wr9     1/1     Running   0          10h   172.17.0.141   aks-userpool1-20365992-vmss00000k
workload-3-7b6d554d-tg7zc     1/1     Running   0          10h   172.17.0.10    aks-agentpool-13458657-vmss000000
```

4. Notice also that some of other pods are on the same node. If you scale **workload-1**, there might not be enough room for the additional pods to be scheduled on the selected node. Those pods will remain in a **Pending** state because any label specified in the **NodeSelector** is a required to be present on the node before a Pod can be scheduled there. Since no other nodes have the required label, there's nowhere for the scheduler to place the additional Pods.

Task 4 – Add a Taint to the Node to prevent other Pods from being scheduled on it

1. Add a **NoSchedule** taint to the selected node

```
kubectl taint nodes $NodeName allowed=GPUOnly:NoSchedule
```

2. The problem is that there are still pods on this node that were scheduled before the node was tainted. Remove those pods and let the scheduler place the on different nodes

```
kubectl delete pods --field-selector=spec.nodeName=$NodeName
```

3. Get a list of Pods

```
kubectl get pods
```

4. As you can see the other Pods were rescheduled. However, **workload-1** pods are all in a **Pending** state.

```
NAME                                READY   STATUS    RESTARTS   AGE
workload-1-7d7499b8f4-4jmck         0/1     Pending   0          15s
workload-1-7d7499b8f4-g6555         0/1     Pending   0          15s
workload-1-7d7499b8f4-lbzbv         0/1     Pending   0          15s
workload-1-7d7499b8f4-ngzj5         0/1     Pending   0          15s
workload-1-7d7499b8f4-plm5p         0/1     Pending   0          15s
workload-1-7d7499b8f4-ss5ds         0/1     Pending   0          15s
workload-2-74d6bffc66-5qbvf         1/1     Running   0          15s
workload-2-74d6bffc66-748x5         1/1     Running   0          10h
workload-2-74d6bffc66-ldlnv         1/1     Running   0          15s
workload-2-74d6bffc66-lz4b4         1/1     Running   0          10h
workload-2-74d6bffc66-q2f7b         1/1     Running   0          10h
workload-2-74d6bffc66-tvm7p         1/1     Running   0          10h
workload-3-7b6d554d-67lg2           1/1     Running   0          10h
workload-3-7b6d554d-8dg5p           1/1     Running   0          10h
workload-3-7b6d554d-dxgrl           1/1     Running   0          15s
workload-3-7b6d554d-pfkkg           1/1     Running   0          10h
workload-3-7b6d554d-s2wr9           1/1     Running   0          10h
workload-3-7b6d554d-tg7zc           1/1     Running   0          10h
```

5. Describe one of the Pods to see the problem (substitute one of your pod names)

```
kubectl describe pod workload-1-7d7499b8f4-4jmck
```

```
Events:
  Type     Reason            Age    From               Message
  ----     ------            ----   ----               -------
  Warning  FailedScheduling  3m37s  default-scheduler  0/3 nodes are available: 1 node(s) had taint {allowed: GPUOnly}, that the pod didn't tolerate, 2 node(s) didn
't match Pod's node affinity/selector.
  Warning  FailedScheduling  3m36s  default-scheduler  0/3 nodes are available: 1 node(s) had taint {allowed: GPUOnly}, that the pod didn't tolerate, 2 node(s) didn
't match Pod's node affinity/selector.
  Normal   NotTriggerScaleUp  3m28s  cluster-autoscaler  pod didn't trigger scale-up: 1 node(s) had taint {allowed: GPUOnly}, that the pod didn't tolerate
```

Task 5 – Add a Toleration to the Pods so they can be scheduled on the selected Node

1. Add a toleration to the `workload-1.yaml` file (same indentation and **NodeSelector**)

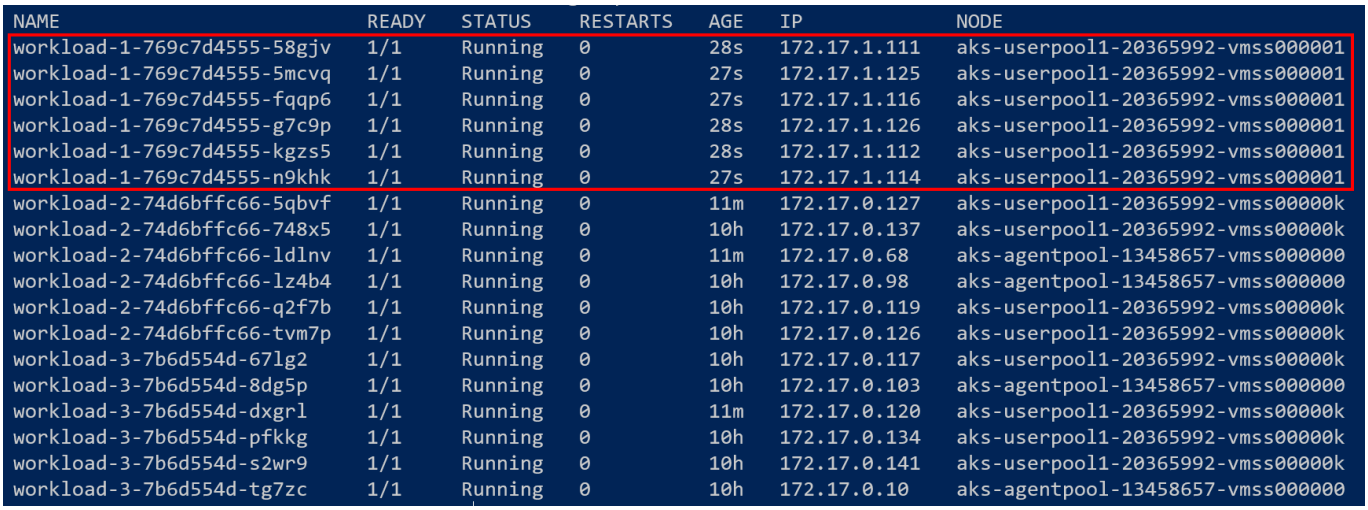
```
      tolerations:
      - key: "allowed"
        operator: "Equal"
        value: "GPUOnly"
        effect: "NoSchedule"
```

2. Save and apply the changes.

```
kubectl apply -f .\workload-1.yaml
```

3. Review the pods

```
kubectl get pods -o wide
```



```
NAME                          READY   STATUS    RESTARTS   AGE   IP             NODE
workload-1-769c7d4555-58gjv   1/1     Running   0          28s   172.17.1.111   aks-userpool1-20365992-vmss000001
workload-1-769c7d4555-5mcvq   1/1     Running   0          27s   172.17.1.125   aks-userpool1-20365992-vmss000001
workload-1-769c7d4555-fqqp6   1/1     Running   0          27s   172.17.1.116   aks-userpool1-20365992-vmss000001
workload-1-769c7d4555-g7c9p   1/1     Running   0          28s   172.17.1.126   aks-userpool1-20365992-vmss000001
workload-1-769c7d4555-kgzs5   1/1     Running   0          28s   172.17.1.112   aks-userpool1-20365992-vmss000001
workload-1-769c7d4555-n9khk   1/1     Running   0          27s   172.17.1.114   aks-userpool1-20365992-vmss000001
workload-2-74d6bffc66-5qbvf   1/1     Running   0          11m   172.17.0.127   aks-userpool1-20365992-vmss00000k
workload-2-74d6bffc66-748x5   1/1     Running   0          10h   172.17.0.137   aks-userpool1-20365992-vmss00000k
workload-2-74d6bffc66-ldlnv   1/1     Running   0          11m   172.17.0.68    aks-agentpool-13458657-vmss000000
workload-2-74d6bffc66-lz4b4   1/1     Running   0          10h   172.17.0.98    aks-agentpool-13458657-vmss000000
workload-2-74d6bffc66-q2f7b   1/1     Running   0          10h   172.17.0.119   aks-userpool1-20365992-vmss00000k
workload-2-74d6bffc66-tvm7p   1/1     Running   0          10h   172.17.0.126   aks-userpool1-20365992-vmss00000k
workload-3-7b6d554d-67lg2     1/1     Running   0          10h   172.17.0.117   aks-userpool1-20365992-vmss00000k
workload-3-7b6d554d-8dg5p     1/1     Running   0          10h   172.17.0.103   aks-agentpool-13458657-vmss000000
workload-3-7b6d554d-dxgrl     1/1     Running   0          11m   172.17.0.120   aks-userpool1-20365992-vmss00000k
workload-3-7b6d554d-pfkkg     1/1     Running   0          10h   172.17.0.134   aks-userpool1-20365992-vmss00000k
workload-3-7b6d554d-s2wr9     1/1     Running   0          10h   172.17.0.141   aks-userpool1-20365992-vmss00000k
workload-3-7b6d554d-tg7zc     1/1     Running   0          10h   172.17.0.10    aks-agentpool-13458657-vmss000000
```

4. Now you can see that all the `Workload-1` pods are on the selected node AND no other pods are scheduled (or will be scheduled) on the selected node.

Shutdown or Delete the AKS Cluster

When you're done for the day, you can shutdown your cluster to avoid incurring charges when you're not using it. This will delete all your nodes, but keep your configuration in tact.

```
az aks stop --name $AKS_NAME `
            --resource-group $AKS_RESOURCE_GROUP
```

Or you can choose to delete the entire resource group instead. You can create a new one prior to the next lab.

```
az group delete --resource-group $AKS_RESOURCE_GROUP
```