Lab Module 2: Azure Kubernetes Service

> Estimated Duration: 60 minutes

Exercise: Create AKS Cluster Using Azure CLI

In this exercise you will create an AKS cluster using the Azure Command-line Interface (CLI).

You should complete this exercise **OR** the Exercise: Create AKS Cluster Using Terraform, **NOT BOTH!**

Task 1 - Create variables resource group

1. Open a Windows Terminal window (defalts to PowerShell).

> Windows Terminal allows you to open tabbed command terminals.

2. Login to Azure.

```
az login
```

3. Set the current subscription.

```
az account set --subscription "Azure Pass - Sponsorship"
```

4. Register needed providers (if not performed in Lab 1).

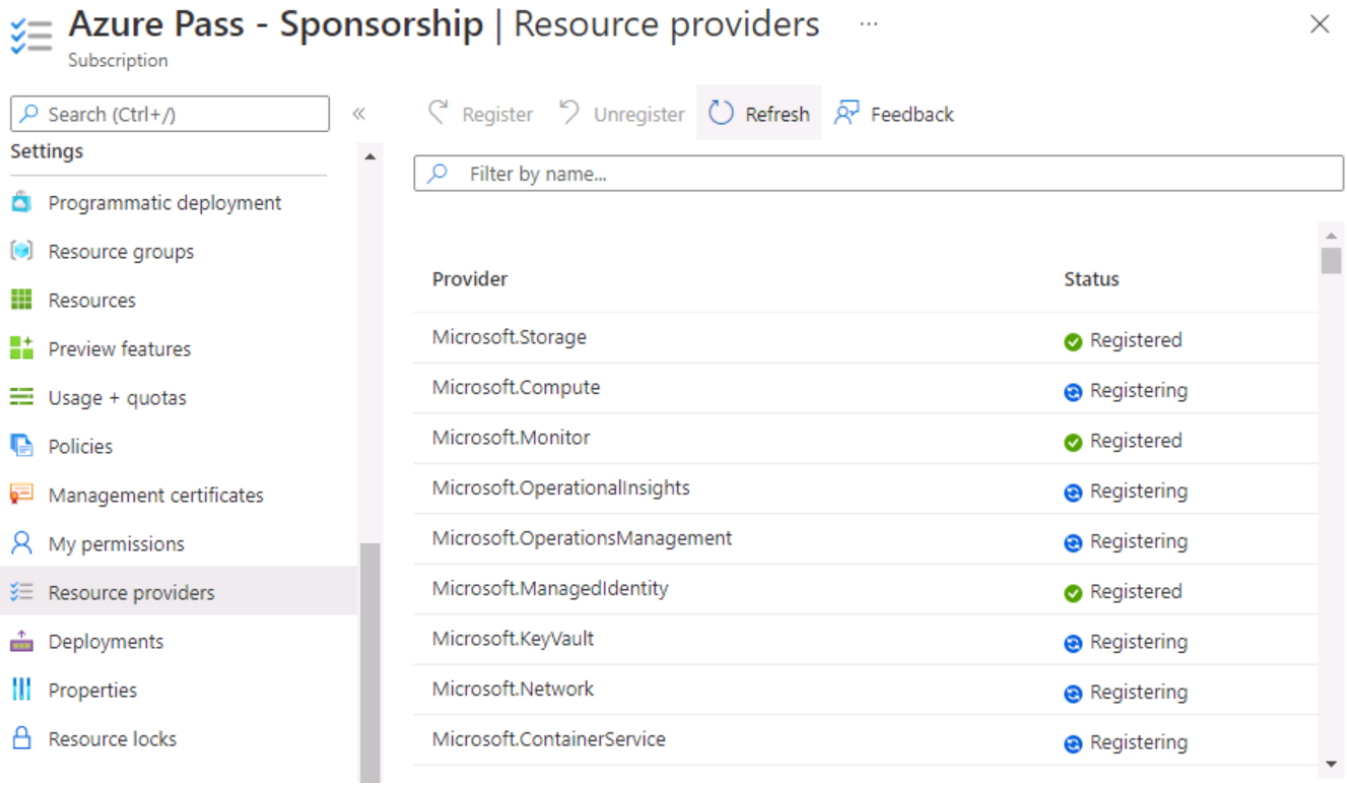
```
az provider register --namespace Microsoft.Storage
az provider register --namespace Microsoft.Compute
az provider register --namespace Microsoft.Network
az provider register --namespace Microsoft.Monitor
az provider register --namespace Microsoft.ManagedIdentity
az provider register --namespace Microsoft.OperationalInsights
az provider register --namespace Microsoft.OperationsManagement
az provider register --namespace Microsoft.KeyVault
```

```
az provider register --namespace Microsoft.ContainerService
az provider register --namespace Microsoft.Kubernetes
```

5. Open a browser and navigate to the Azure Portal: `portal.azure.com`

6. Search for and open the **Subscriptions** blade. Select your **Azure Pass - Sponsorship** subscription.

7. Scroll down and select **Resource providers**.



8. Watch the progress of the registration process until all the providers listed above have been registered. Click the `Refresh` button every few minutes to update the progess. Once everything has been registered, continue with the tasks in this lab.

Task 2 - Define variables and create resource group

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

2. Set your initials and define variables.

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

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

3. Get list of available VM sizes with 2 cores in your region.

```
az vm list-sizes --location $LOCATION --query "[?numberOfCores == ``2``].
{Name:name}" -o table
```

4. Set the VM SKU to one of the available values

```
$VM_SKU="Standard_D2as_v5"
```

5. Create Resource Group.

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

6. Create User Managed Identity

```
$AKS_IDENTITY_ID=$(az identity create --name $AKS_IDENTITY --resource-group
$AKS_RESOURCE_GROUP --query id -o tsv)
```

Task 2 - Create a Virtual Network and a Subnet

1. Define variables for network resources.

```
$AKS_VNET="aks-$($INITIALS)-vnet"
$AKS_VNET_SUBNET="aks-$($INITIALS)-subnet"
$AKS_VNET_ADDRESS_PREFIX="10.0.0.0/8"
$AKS_VNET_SUBNET_PREFIX="10.240.0.0/16"
```

2. Create an Azure Virtual Network and a Subnet.

```
az network vnet create --resource-group $AKS_RESOURCE_GROUP `
                       --name $AKS_VNET `
                       --address-prefix $AKS_VNET_ADDRESS_PREFIX `
                       --subnet-name $AKS_VNET_SUBNET `
                       --subnet-prefix $AKS_VNET_SUBNET_PREFIX
```

3. Get virtual network default subnet id

```
$AKS_VNET_SUBNET_ID=$(az network vnet subnet show --resource-group
$AKS_RESOURCE_GROUP --vnet-name $AKS_VNET --name $AKS_VNET_SUBNET --query id -o
```

```
tsv)

Write-Host "Default Subnet ID: $AKS_VNET_SUBNET_ID"
```

Task 3 - Create a Log Analytics Workspace (if needed)

1. Create a Log Analytics Workspace.

```
$LOG_ANALYTICS_WORKSPACE_NAME="aks-$($INITIALS)-law"
$LOG_ANALYTICS_WORKSPACE_RESOURCE_ID=$(az monitor log-analytics workspace create -
--resource-group $AKS_RESOURCE_GROUP --workspace-name $LOG_ANALYTICS_WORKSPACE_NAME
--query id -o tsv)
Write-Host "LAW Workspace Resource ID: $LOG_ANALYTICS_WORKSPACE_RESOURCE_ID"
```

Task 4 - Create an AKS Cluster with a System Node Pool

1. Use all the prior settings and resources to create the AKS cluster. This step will take 5-10 minutes to complete.

> See Microsoft reference: https://docs.microsoft.com/en-us/cli/azure/aks?view=azure-cli-latest#az_aks_create

```
$AKS_NAME="aks-$($INITIALS)"
Write-Host "AKS Cluster Name: $AKS_NAME"
```

```
az aks create --resource-group $AKS_RESOURCE_GROUP `
              --generate-ssh-keys `
              --enable-managed-identity `
              --assign-identity $AKS_IDENTITY_ID `
              --node-count 1 `
              --enable-cluster-autoscaler `
              --min-count 1 `
              --max-count 3 `
              --network-plugin azure `
              --service-cidr 10.0.0.0/16 `
              --dns-service-ip 10.0.0.10 `
              --docker-bridge-address 172.17.0.1/16 `
              --vnet-subnet-id $AKS_VNET_SUBNET_ID `
              --node-vm-size $VM_SKU `
              --nodepool-name system1 `
              --enable-addons monitoring `
              --workspace-resource-id $LOG_ANALYTICS_WORKSPACE_RESOURCE_ID `
              --enable-ahub `
              --name $AKS_NAME
```

2. Once the cluster is ready (after at least 5 minutes), connect your local machine to it.

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

3. List the nodes in the cluster. There should be **1** node to start with.

```
kubectl get nodes
```

4. Get the list of system nodes only. At this point, the list should return **1** node.

```
kubectl get nodes -l="kubernetes.azure.com/mode=system"
```

```
NAME                                STATUS   ROLES   AGE    VERSION
aks-system1-42326736-vmss000000     Ready    agent   108s   v1.20.9
```

5. List the node pools in the cluster. There should be 1 node pool returned.

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

```
Name     OsType    KubernetesVersion    VmSize           Count    MaxPods    ProvisioningState    Mode
-------  --------  -------------------  ---------------  -------  ---------  -------------------  ------
system1  Linux     1.20.9               Standard_DS2_v2  1        30         Succeeded            System
```

Task 5 - Create a Linux User Node Pool

1. Create a Linux user node pool.

```
az aks nodepool add --resource-group $AKS_RESOURCE_GROUP `
                    --cluster-name $AKS_NAME `
                    --os-type Linux `
                    --name linux1 `
                    --node-count 1 `
                    --enable-cluster-autoscaler `
                    --min-count 1 `
                    --max-count 3 `
                    --mode User `
                    --node-vm-size $VM_SKU
```

2. List the nodes in the cluster. There should be 2 nodes now.

```
kubectl get nodes
```

3. Get the list of system nodes only. There should still be only 1 system node.

```
kubectl get nodes -l="kubernetes.azure.com/mode=system"
```

4. List the node pools in the cluster. Again there should be 2 now.

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

```
Name     OsType    KubernetesVersion    VmSize           Count    MaxPods    ProvisioningState    Mode
-------  --------  -------------------  ---------------  -------  ---------  -------------------  ------
linux1   Linux     1.20.9               Standard_DS2_v2  1        30         Succeeded            User
system1  Linux     1.20.9               Standard_DS2_v2  1        30         Succeeded            System
```

Task 6 - Create a Windows User Node Pool

1. Create a Windows user node pool.

```
az aks nodepool add --resource-group $AKS_RESOURCE_GROUP `
                    --cluster-name $AKS_NAME `
                    --os-type Windows `
                    --name win1 `
                    --node-count 1 `
                    --mode User `
                    --node-vm-size $VM_SKU
```

2. List the nodes in the cluster. There should be 3 nodes.

```
kubectl get nodes
```

```
NAME                                STATUS   ROLES   AGE     VERSION
aks-linux1-42326736-vmss000000      Ready    agent   10m     v1.20.9
aks-system1-42326736-vmss000000     Ready    agent   16m     v1.20.9
akswin1000000                       Ready    agent   2m30s   v1.20.9
```

3. Get the list of system nodes only. There should still be only 1 node.

```
kubectl get nodes -l="kubernetes.azure.com/mode=system"
```

4. Get the list of Linux nodes. There should be 2 nodes.

```
kubectl get nodes -l="kubernetes.io/os=linux"
```

5. List the node pools in the cluster. Again there should be 3.

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

```
Name     OsType    KubernetesVersion    VmSize           Count    MaxPods    ProvisioningState    Mode
-------  --------  -------------------  ---------------  -------  ---------  -------------------  ------
linux1   Linux     1.20.9               Standard_DS2_v2  1        30         Succeeded            User
system1  Linux     1.20.9               Standard_DS2_v2  1        30         Succeeded            System
win1     Windows   1.20.9               Standard_DS2_v2  1        30         Succeeded            User
```

Task 7 - Adjust the Auto Scaler for the lab

The default settings for the AKS Auto Scaler are tuned for production environments, but take too long to scale up/down during demos and practice labs.

1. Configure the cluster-wide auto scaling profile so all the node pool auto scalers are more responsive.

```
az aks update --resource-group $AKS_RESOURCE_GROUP `
              --name $AKS_NAME `
              --cluster-autoscaler-profile `
                  scale-down-delay-after-add=1m `
                  scale-down-unready-time=1m `
                  scale-down-unneeded-time=1m `
                  skip-nodes-with-system-pods=true
```

> Refer to this link for a complete description of the options available: AKS Cluster Autoscaler

Exercise: Create AKS Cluster Using Terraform

In this exercise you will create an AKS cluster using the Terraform utility by HashiCorp.

You should complete this exercise **OR** the Exercise: Create AKS Cluster Using Azure CLI, **NOT BOTH!**

Task 1 - Ensure resource names are unique

1. Open a Windows Terminal window (defaults to PowerShell).

2. Login to Azure.

```
az login
```

3. Define variables.

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

```
$YOUR_INITIALS="$($INITIALS)".ToLower()
$env:TF_VAR_AKS_RESOURCE_GROUP="azure-$($INITIALS)-rg"
$env:TF_VAR_ACR="acr$($INITIALS)"
```

Task 2 - Go to the Main Terraform folder and Initialize It

1. Open a Windows Terminal window (defaults to PowerShell).

2. Change current folder to **Main**

```
$ROOT_PATH="C:\k8s\Labs\Module2\Terraform"
cd  C:\k8s\labs\Module2\Terraform\Main
```

3. To initialize the Terraform directory it is necessary the **main.tf** file that is already created in the **Main** folder.

```
# We strongly recommend using the required_providers block to set the
# Azure Provider source and version being used
terraform {
  required_providers {
    azurerm = {
      source  = "hashicorp/azurerm"
      version = "=3.3.0"
    }
  }
}

# Configure the Microsoft Azure Provider
provider "azurerm" {
  features {}
}
```

1. Open a Terminal in VS Code and initialize the Terraform directory.

```
terraform init
```

Task 3 - Create an Azure Resource Group

1. Add the following variable in the **rg_vars.tf** in the **Resources** folder to the **vars.tf** file in the **Main** folder. This code will use the environment variable defined earlier.

```
variable "AKS_RESOURCE_GROUP" {
    type        = string
    description = "This is the rg name"
}
```

```
$From = Get-Content -Path (Join-Path $ROOT_PATH Resources\rg_vars.tf)
Add-Content -Path (Join-Path $ROOT_PATH Main\vars.tf) -Value $From
```

2. Move the file **rg.tf** in the **Resources** folder to the **Main** folder.

```
resource "azurerm_resource_group" "arg" {
  name     = var.AKS_RESOURCE_GROUP
  location = "East US"
}
```

```
Move-Item (Join-Path $ROOT_PATH Resources\rg.tf) -Destination (Join-Path
$ROOT_PATH Main)
```

3. Have Terraform plan and apply your changes. Then this step finishes, you should have a new Resource Group available in the Azure Portal.

```
terraform plan
terraform apply -auto-approve
```

Task 4 - Create an Azure Container Registry

1. Add the following variables in the **acr_vars.tf** in the **Resources** folder to the **vars.tf** file in the **Main** folder. This code will use the environment variables defined earlier.

```
variable "ACR" {
  type        = string
  description = "This is the acr name"
}
```

```
$From = Get-Content -Path (Join-Path $ROOT_PATH Resources\acr_vars.tf)
Add-Content -Path (Join-Path $ROOT_PATH Main\vars.tf) -Value $From
```

2. Move the file **acr.tf** in the **Resources** folder to the **Main** folder.

```
resource "azurerm_container_registry" "acr" {
  name                = var.ACR
  location            = "East US"
  resource_group_name = var.AKS_RESOURCE_GROUP
  sku                 = "Standard"
  admin_enabled       = false
}
```

```
Move-Item (Join-Path $ROOT_PATH Resources\acr.tf) -Destination (Join-Path
$ROOT_PATH Main)
```

3. Have Terraform plan and apply your changes. Then this step finishes, you should have a new ACR available in the Azure Portal.

```
terraform plan
terraform apply -auto-approve
```

Task 5 - Create a Virtual Network and a Subnet

1. Define variables.

```
$env:TF_VAR_AKS_VNET="aks-$($INITIALS)-vnet"
$env:TF_VAR_AKS_VNET_SUBNET="aks-$($INITIALS)-subnet"
$env:TF_VAR_AKS_VNET_ADDRESS_PREFIX='["10.0.0.0/8"]'
$env:TF_VAR_AKS_VNET_SUBNET_PREFIX="10.240.0.0/16"
```

2. Add the following variables in the **vnet_vars.tf** in the **Resources** folder to the end of the **vars.tf** file in the **Main** folder.

```
variable "AKS_VNET" {
    type        = string
    description = "This is the vnet name"
}

variable "AKS_VNET_SUBNET" {
    type        = string
```

```
    description = "This is the subnet name"
}

variable "AKS_VNET_ADDRESS_PREFIX" {
    type        = list(string)
    description = "This is the vnet address prefix"
}

variable "AKS_VNET_SUBNET_PREFIX" {
    type        = string
    description = "This is the subnet address prefix"
}
```

```
$From = Get-Content (Join-Path $ROOT_PATH Resources\vnet_vars.tf)
Add-Content (Join-Path $ROOT_PATH Main\vars.tf) -Value $From
```

3. Move the file called **vnet.tf** in the **Resources** folder to the **Main** folder.

```
resource "azurerm_virtual_network" "network" {
  name                = var.AKS_VNET
  location            = "West US 2"
  resource_group_name = var.AKS_RESOURCE_GROUP
  address_space       = var.AKS_VNET_ADDRESS_PREFIX

  subnet {
    name           = var.AKS_VNET_SUBNET
    address_prefix = var.AKS_VNET_SUBNET_PREFIX
  }
}
```

```
Move-Item (Join-Path $ROOT_PATH Resources\vnet.tf) -Destination (Join-Path
$ROOT_PATH Main)
```

4. Have Terraform plan and apply your changes. When this step finishes, there will be a VNET created and ready to be used by your cluster.

```
terraform plan
terraform apply -auto-approve
```

Task 6 - Create a Log Analytics Workspace

1. Define variables.

```
$env:TF_VAR_LOG_ANALYTICS_WORKSPACE_NAME="aks-$($INITIALS)-law"
```

2. Add the following variables in the **law_vars.tf** in the **Resources** folder to the end of the **vars.tf** file in the **Main** folder.

```
variable "LOG_ANALYTICS_WORKSPACE_NAME" {
    type        = string
    description = "This is the law name"
}
```

```
$From = Get-Content (Join-Path $ROOT_PATH Resources\law_vars.tf)
Add-Content (Join-Path $ROOT_PATH Main\vars.tf) -Value $From
```

3. Move the file called **law.tf** in the **Resources** folder to the **Main** folder.

```
resource "azurerm_log_analytics_workspace" "log_analytics" {
  name                = var.LOG_ANALYTICS_WORKSPACE_NAME
  location            = "West US 2"
  resource_group_name = var.AKS_RESOURCE_GROUP
  sku                 = "PerGB2018"
  retention_in_days   = "90"
}

resource "azurerm_log_analytics_solution" "log_analytics" {
  solution_name         = "ContainerInsights"
  location              = azurerm_log_analytics_workspace.log_analytics.location
  resource_group_name   =
azurerm_log_analytics_workspace.log_analytics.resource_group_name
  workspace_resource_id = azurerm_log_analytics_workspace.log_analytics.id
  workspace_name        = azurerm_log_analytics_workspace.log_analytics.name

  plan {
    publisher = "Microsoft"
    product   = "OMSGallery/ContainerInsights"
  }
}
```

```
Move-Item (Join-Path $ROOT_PATH Resources\law.tf) -Destination (Join-Path
$ROOT_PATH Main)
```

4. Have Terraform plan and apply your changes. When this step finishes, there will be a Log Analytics Workspace for your cluster to send telemetry to.

```
terraform plan
terraform apply -auto-approve
```

Task 7 - Create an AKS Cluster

1. Use all the prior settings and resources to create the AKS cluster.

```
$env:TF_VAR_AKS_CLUSTER="aks-$($INITIALS)-tf"
```

2. Add the following variable in the **aks_vars.tf** in the **Resources** folder to the end of the **vars.tf** file in the **Main** folder.

```
variable "AKS_CLUSTER" {
    type        = string
    description = "This is the aks name"
}
```

```
$From = Get-Content (Join-Path $ROOT_PATH Resources\aks_vars.tf)
Add-Content (Join-Path $ROOT_PATH Main\vars.tf) -Value $From
```

3. Move the file called **aks.tf** in the **Resources** folder to the **Main** folder.

```
resource "azurerm_kubernetes_cluster" "aks" {

  name                = var.AKS_CLUSTER
  location            = "East US"
  resource_group_name = var.AKS_RESOURCE_GROUP
  dns_prefix          = var.AKS_CLUSTER
  azure_policy_enabled = true

  default_node_pool {
    name                = "systempool"
    node_count          = 1
    enable_auto_scaling = true
    min_count           = 1
    max_count           = 3
    vnet_subnet_id      = element(azurerm_virtual_network.network.subnet.*.id, 0)
    vm_size             = "Standard_D2as_v5"
    zones               = ["1"]
  }

  identity {
    type = "SystemAssigned"
  }
```

```
  oms_agent {
    log_analytics_workspace_id = azurerm_log_analytics_workspace.log_analytics.id
  }

  network_profile {
    network_plugin     = "azure"
    service_cidr       = "10.0.0.0/16"
    dns_service_ip     = "10.0.0.10"
    docker_bridge_cidr = "172.17.0.1/16"
  }
}

resource "azurerm_role_assignment" "aks_acr" {
  scope                = azurerm_container_registry.acr.id
  role_definition_name = "AcrPull"
  principal_id         =
azurerm_kubernetes_cluster.aks.kubelet_identity[0].object_id
}
```

```
Move-Item (Join-Path $ROOT_PATH Resources\aks.tf) -Destination (Join-Path
$ROOT_PATH Main)
```

4. Have Terraform plan and apply your changes. This step will take 5-10 minutes to complete.

```
terraform plan
terraform apply -auto-approve
```

5. Once the cluster is ready, connect your local machine to it.

```
az aks get-credentials --name $env:TF_VAR_AKS_CLUSTER --resource-group
$env:TF_VAR_AKS_RESOURCE_GROUP
```

6. List the nodes in the cluster. There should be 1 nodes to start with.

```
kubectl get nodes
```

Task 8 - Create a Linux User Node Pool

1. Create a Linux user node pool, add this code from the **aks_linux_node_pool.tf** in the **Resources** folder to the bottom of the **aks.tf** file.

```
resource "azurerm_kubernetes_cluster_node_pool" "linuxnp" {
  name                  = "linuxagent1"
  kubernetes_cluster_id = azurerm_kubernetes_cluster.aks.id
  os_type               = "Linux"
  node_count            = 1
  enable_auto_scaling   = true
  min_count             = 1
  max_count             = 3
  mode                  = "User"
  vm_size               = "Standard_D2as_v5"
  zones                 = ["1"]
  vnet_subnet_id        = element(azurerm_virtual_network.network.subnet.*.id, 0)
}
```

```
$From = Get-Content (Join-Path $ROOT_PATH Resources\aks_linux_node_pool.tf)
Add-Content (Join-Path $ROOT_PATH Main\aks.tf) -Value $From
```

2. Have Terraform plan and apply your changes.

```
terraform plan
terraform apply -auto-approve
```

3. List the nodes in the cluster. There should be 2 nodes now.

```
kubectl get nodes
```

Task 9 - Create a Windows User Node Pool

1. Create a Windows user node pool, add this code from the **aks_windows_node_pool.tf** in the **Resources** folder to the bottom of the **aks.tf** file.

```
resource "azurerm_kubernetes_cluster_node_pool" "winnp" {
  name                  = "win1"
  kubernetes_cluster_id = azurerm_kubernetes_cluster.aks.id
  os_type               = "Windows"
  node_count            = 1
  enable_auto_scaling   = true
  min_count             = 1
  max_count             = 3
  mode                  = "User"
  vm_size               = "Standard_D2as_v5"
  zones                 = ["1"]
  vnet_subnet_id        = element(azurerm_virtual_network.network.subnet.*.id, 0)
}
```

```
$From = Get-Content (Join-Path $ROOT_PATH Resources\aks_windows_node_pool.tf)
Add-Content (Join-Path $ROOT_PATH Main\aks.tf) -Value $From
```

2. Have Terraform plan and apply your changes.

```
terraform plan
terraform apply -auto-approve
```

3. List the nodes in the cluster. There should be 3 nodes.

```
kubectl get nodes
```

Task 10 - Create the Diagnostic Settings Module

1. Define variables.

```
$env:TF_VAR_DIAGNOSTIC_SETTING_NAME="ds-$($INITIALS)"
```

2. Add the following variable in the **ds_vars.tf** in the **Resources** folder to the end of the **vars.tf** file in the **Main** folder.

```
variable "DIAGNOSTIC_SETTING_NAME" {
    type        = string
    description = "This is the ds name"
}
```

```
$From = Get-Content (Join-Path $ROOT_PATH Resources\ds_vars.tf)
Add-Content (Join-Path $ROOT_PATH Main\vars.tf) -Value $From
```

3. Move the file called **ds.tf** in the **Resources** folder to the **Main** folder.

```
resource "azurerm_monitor_diagnostic_setting" "diagnostic_logs" {
  name               = var.DIAGNOSTIC_SETTING_NAME
  target_resource_id = azurerm_kubernetes_cluster.aks.id
  storage_account_id = azurerm_storage_account.storage.id

  dynamic "log" {
```

```
    for_each = ["kube-apiserver", "kube-controller-manager", "cluster-autoscaler",
"kube-scheduler", "kube-audit", "kube-audit-admin", "guard"]
    content {
      category = log.value
      enabled  = true

      retention_policy {
        enabled = true
        days    = 30
      }
    }
  }

  metric {
    category = "AllMetrics"
    enabled  = true

    retention_policy {
      days    = 30
      enabled = true
    }
  }
}
```

```
Move-Item (Join-Path $ROOT_PATH Resources\ds.tf) -Destination (Join-Path
$ROOT_PATH Main)
```

Task 11 - Create the Storage Account Module for the Diagnostic Settings

1. Define variables.

```
$env:TF_VAR_STORAGE_ACCOUNT_NAME="sa$($INITIALS)"
```

2. Add the following variable in the **sa_vars.tf** in the **Resources** folder to the end of the **vars.tf** file in the **Main** folder.

```
variable "STORAGE_ACCOUNT_NAME" {
    type        = string
    description = "This is the sa name"
}
```

```
$From = Get-Content (Join-Path $ROOT_PATH Resources\sa_vars.tf)
Add-Content (Join-Path $ROOT_PATH Main\vars.tf) -Value $From
```

3. Move the file called **sa.tf** in the **Resources** folder to the **Main** folder.

```
resource "azurerm_storage_account" "storage" {
  name                     = var.STORAGE_ACCOUNT_NAME
  location                 = "West US 2"
  resource_group_name      = var.AKS_RESOURCE_GROUP
  account_kind             = "StorageV2"
  account_tier             = "Standard"
  account_replication_type = "LRS"
  access_tier              = "Hot"
  enable_https_traffic_only = true
}
```

```
Move-Item (Join-Path $ROOT_PATH Resources\sa.tf) -Destination (Join-Path
$ROOT_PATH Main)
```

4. Have Terraform plan and apply your changes.

```
terraform plan
terraform apply -auto-approve
```

Task 12 - Adjust the Auto Scaler for the lab

1. The default settings for the AKS Auto Scaler are tuned for production environments, but take too long to scale up/down during demos and practice labs.
2. Configure the cluster-wide auto scaling profile so all the node pool auto scalers are more responsive. Add the following code inside the resource **"azurerm_kubernetes_cluster" "aks" {}** module, for example, below the **default_node_pool {}**. For this purpouse replace **aks.tf** file in the **Main** folder with the content of **aks2.tf** in the **Resources** folder.

```
  auto_scaler_profile {
    scale_down_unneeded         = "1m"
    scale_down_delay_after_add  = "1m"
    scale_down_unready          = "1m"
    skip_nodes_with_system_pods = true
  }
```

```
Get-Content (Join-Path $ROOT_PATH Resources\aks2.tf) | Set-Content (Join-Path
$ROOT_PATH Main\aks.tf)
```

3. Have Terraform plan and apply your changes.

```
terraform plan
terraform apply -auto-approve
```

Exercise: Deploying Workloads to Nodes

In this exercise you will deploy different pods to various node pools in your cluster.

Task 1 - Deploy a simple workload with no Node Selector

1. Change current folder to **Module2**

```
cd  C:\k8s\labs\Module2
```

2. Deploy a workload with 6 replicas and no `Node Selector`.

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

3. Get a complete list of all pods and review the results.

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

```
NAME                        READY   STATUS             RESTARTS   AGE   IP            NODE
ATES
workload-5f795755d8-8v8lb   1/1     Running            0          37s   10.240.0.28   aks-system1-42326736-vmss000000
workload-5f795755d8-gs7r8   0/1     ImagePullBackOff   0          37s   10.240.0.84   akswin1000000
workload-5f795755d8-ph4t6   0/1     ErrImagePull       0          37s   10.240.0.92   akswin1000000
workload-5f795755d8-q6zxw   1/1     Running            0          37s   10.240.0.52   aks-linux1-42326736-vmss000000
workload-5f795755d8-r4szw   1/1     Running            0          37s   10.240.0.43   aks-linux1-42326736-vmss000000
workload-5f795755d8-srn79   0/1     ImagePullBackOff   0          37s   10.240.0.71   akswin1000000
```

> Notice some of the pods got created on the Linux nodes, while others got scheduled on the Windows nodes, and thus could not be run. The Linux image specified in the Deployment is not compatible with the Windows 2019 OS running on the Windows nodes.

4. Describe any of the failed pods to see the actual error.

```
kubectl describe pod <failed pod name>
```

```
Events:
  Type     Reason          Age                  From               Message
  ----     ------          ----                 ----               -------
  Normal   Scheduled       7m3s                 default-scheduler  Successfully assigned default/workload-5f795755d8-bb4ct to akswin
000001
  Normal   SandboxChanged  6m59s                kubelet            Pod sandbox changed, it will be killed and re-created.
  Warning  Failed          6m15s (x3 over 7m)   kubelet            Failed to pull image "nginx:1.18": rpc error: code = Unknown desc
= no matching manifest for windows/amd64 10.0.17763 in the manifest list entries
  Normal   Pulling         5m23s (x4 over 7m)   kubelet            Pulling image "nginx:1.18"
  Warning  Failed          114s (x22 over 6m55s)  kubelet          Error: ImagePullBackOff
```

> Without any guidance, the Kubernetes scheduler does its best to distribute workloads evenly across all the nodes that have available resources. It doesn't examine the contents of Deployments to confirm that their images are compatible with the nodes it selects.

5. Update the **workload.yaml** file to add the following node selector:

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

6. Apply the deployment again and list the pods.

```
kubectl apply -f workload.yaml
kubectl get pods -o wide
```

> It will take a few seconds for the bad pods to be deleted, so they may show as **Terminated** for little some time.

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
workload-86dd8dfdd6-4tnzg	1/1	Running	0	13s	10.240.0.65	aks-linux1-42326736-vmss000000
workload-86dd8dfdd6-87k4c	1/1	Running	0	11s	10.240.0.7	aks-system1-42326736-vmss000000
workload-86dd8dfdd6-s9wxm	1/1	Running	0	11s	10.240.0.45	aks-linux1-42326736-vmss000000
workload-86dd8dfdd6-vb7z9	1/1	Running	0	11s	10.240.0.46	aks-linux1-42326736-vmss000000
workload-86dd8dfdd6-w7dr6	1/1	Running	0	13s	10.240.0.11	aks-system1-42326736-vmss000000
workload-86dd8dfdd6-zm8sb	1/1	Running	0	13s	10.240.0.64	aks-linux1-42326736-vmss000000

> Notice that all the pods are running now, but they're spread across both the system and user nodes. The reason for creating user nodes is to seprate your workloads from system utilities.

7. Update the **workload.yaml** file to add an additional label "**kubernetes.azure.com/mode: user**" to the Node Selector. The final `nodeSelector` section should look like this:

```
nodeSelector:
  kubernetes.io/os: linux
  kubernetes.azure.com/mode: user
```

8. Apply the deployment again and list the pods.

```

```

```
kubectl apply -f workload.yaml
kubectl get pods -o wide
```

```
NAME                        READY   STATUS    RESTARTS   AGE   IP             NODE
workload-bbc8765b7-5gnts    1/1     Running   0          23s   10.240.0.40    aks-linux1-42326736-vmss000000
workload-bbc8765b7-5mzbz    1/1     Running   0          23s   10.240.0.55    aks-linux1-42326736-vmss000000
workload-bbc8765b7-n8j9r    1/1     Running   0          20s   10.240.0.36    aks-linux1-42326736-vmss000000
workload-bbc8765b7-pzz59    1/1     Running   0          21s   10.240.0.43    aks-linux1-42326736-vmss000000
workload-bbc8765b7-rsrc5    1/1     Running   0          23s   10.240.0.42    aks-linux1-42326736-vmss000000
workload-bbc8765b7-xnsvb    1/1     Running   0          21s   10.240.0.49    aks-linux1-42326736-vmss000000
```

PERFECT!

Exercise: Scaling Nodes to Meet Demand

In this exercise you'll watch how the AKS Auto Scaler adjusts the number of nodes based on increased demand and then scales them back down when the load is reduced.

Task 1 - Verify the names and number of current nodes

1. Get a list of the Linux user nodes.

```
kubectl get nodes -l="kubernetes.azure.com/mode=user,kubernetes.io/os=linux"
```

Task 2 - Increase the number of replicas to trigger the Auto Scaler to scale up

1. Increase the number of replicas to force the Auto Scaler to create more nodes in the Linux user node pool.

```
kubectl scale --replicas=40 deploy/workload
```

2. Get the list of pods.

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

> Notice that some pods are **Pending** state.

```
NAME                          READY   STATUS    RESTARTS   AGE     IP            NODE
ATES
workload-bbc8765b7-4fptg      0/1     Pending   0          4s      <none>        <none>
workload-bbc8765b7-4tttk      0/1     Pending   0          4s      <none>        <none>
workload-bbc8765b7-5852w      0/1     Pending   0          4s      <none>        <none>
workload-bbc8765b7-5gnts      1/1     Running   0          2m3s    10.240.0.40   aks-linux1-42326736-vmss000000
workload-bbc8765b7-5mzbz      1/1     Running   0          2m3s    10.240.0.55   aks-linux1-42326736-vmss000000
workload-bbc8765b7-6d4cv      1/1     Running   0          4s      10.240.0.37   aks-linux1-42326736-vmss000000
workload-bbc8765b7-6pdxj      0/1     Pending   0          4s      <none>        <none>
workload-bbc8765b7-6qt9p      1/1     Running   0          4s      10.240.0.58   aks-linux1-42326736-vmss000000
workload-bbc8765b7-75bsh      0/1     Pending   0          4s      <none>        <none>
workload-bbc8765b7-9pjxj      0/1     Pending   0          4s      <none>        <none>
workload-bbc8765b7-b2qkr      0/1     Pending   0          4s      <none>        <none>
workload-bbc8765b7-b6r2f      0/1     Pending   0          4s      <none>        <none>
workload-bbc8765b7-c7ph6      1/1     Running   0          4s      10.240.0.41   aks-linux1-42326736-vmss000000
workload-bbc8765b7-dk748      0/1     Pending   0          4s      <none>        <none>
workload-bbc8765b7-fv9hd      1/1     Running   0          4s      10.240.0.46   aks-linux1-42326736-vmss000000
workload-bbc8765b7-hrl4h      0/1     Pending   0          4s      <none>        <none>
```

3. Describe one of the Pending pods

```
kubectl describe pod <pending pod name>
```

> Notice that the Events section describes the problem and the solution.

```
Events:
  Type     Reason            Age   From                Message
  ----     ------            ----  ----                -------
  Warning  FailedScheduling  74s   default-scheduler   0/3 nodes are available: 1 Insufficient cpu, 2 node(s) didn't match Pod's node affinity.
  Warning  FailedScheduling  74s   default-scheduler   0/3 nodes are available: 1 Insufficient cpu, 2 node(s) didn't match Pod's node affinity.
  Normal   TriggeredScaleUp  66s   cluster-autoscaler  pod triggered scale-up: [{aks-linux1-42326736-vmss 1->3 (max: 3)}]
```

4. Start a watch on the nodes

```
kubectl get nodes -l="kubernetes.azure.com/mode=user,kubernetes.io/os=linux" -w
```

In a few minutes you'll see the number of nodes increase.

```
NAME                                STATUS   ROLES   AGE     VERSION
aks-linux1-42326736-vmss000000      Ready    agent   35m     v1.20.9
aks-linux1-42326736-vmss000001      Ready    agent   2m      v1.20.9
aks-linux1-42326736-vmss000002      Ready    agent   119s    v1.20.9
```

5. When the new nodes are in a **Ready** state, press `Ctrl-C` to break out of the watch and return to the console.
6. Get a list of pods.

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

> Notice all the pening Pods are running and they're all in the same node pool, scheduled on the new nodes.

```
NAME                        READY   STATUS    RESTARTS   AGE     IP             NODE
workload-bbc8765b7-4fptg    1/1     Running   0          5m50s   10.240.0.106   aks-linux1-42326736-vmss000001
workload-bbc8765b7-4tttk    1/1     Running   0          5m50s   10.240.0.137   aks-linux1-42326736-vmss000002
workload-bbc8765b7-5852w    1/1     Running   0          5m50s   10.240.0.110   aks-linux1-42326736-vmss000001
workload-bbc8765b7-5gnts    1/1     Running   0          7m49s   10.240.0.40    aks-linux1-42326736-vmss000000
workload-bbc8765b7-5mzbz    1/1     Running   0          7m49s   10.240.0.55    aks-linux1-42326736-vmss000000
workload-bbc8765b7-6d4cv    1/1     Running   0          5m50s   10.240.0.37    aks-linux1-42326736-vmss000000
workload-bbc8765b7-6pdxj    1/1     Running   0          5m50s   10.240.0.155   aks-linux1-42326736-vmss000002
workload-bbc8765b7-6qt9p    1/1     Running   0          5m50s   10.240.0.58    aks-linux1-42326736-vmss000000
workload-bbc8765b7-75bsh    1/1     Running   0          5m50s   10.240.0.126   aks-linux1-42326736-vmss000001
```

Task 3 - Reduce workload to trigger the Auto Scaler to scale down

1. Delete the workload.

```
kubectl delete deploy/workload
```

2. Watch the nodes

```
kubectl get nodes -l="kubernetes.azure.com/mode=user,kubernetes.io/os=linux" -w
```

> In a few minutes, two of the nodes will go into a **NotReady** state and then disappear. It will probably NOT be both the new nodes that were created. The "victim" nodes are picked using an internal scale-in policy.

```
NAME                                STATUS     ROLES   AGE     VERSION
aks-linux1-42326736-vmss000000      NotReady   agent   42m     v1.20.9
aks-linux1-42326736-vmss000001      NotReady   agent   8m22s   v1.20.9
aks-linux1-42326736-vmss000002      Ready      agent   8m21s   v1.20.9
```

> The cluster autoscaler may be unable to scale down if pods can't be deleted, such as in the following situations:

- A pod is directly created and isn't backed by a controller object, such as a deployment or replica set.
- A pod disruption budget (PDB) is too restrictive and doesn't allow the number of pods to be fall below a certain threshold.
- A pod uses node selectors or anti-affinity that can't be honored by other nodes when the replacement Pod is scheduled.

3. Break out of the watch and keep getting the list of nodes manually. Repeat the command below until there is only 1 node left.

```
kubectl get nodes -l="kubernetes.azure.com/mode=user,kubernetes.io/os=linux"
```

```
NAME                                STATUS   ROLES    AGE   VERSION
aks-linux1-42326736-vmss000002      Ready    agent    10m   v1.20.9
```

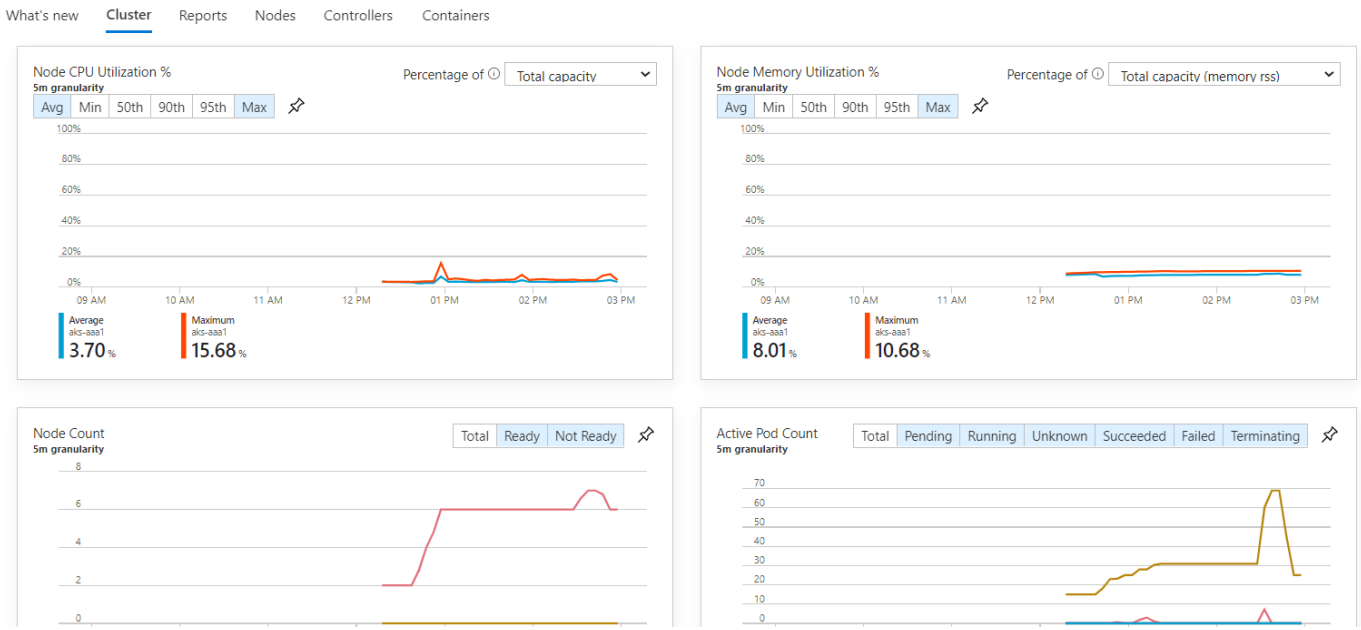
Exercise: Examine Container Insights

Now that you've used your cluster for a little while, you should have some metrics and logs in Container Insights to review.

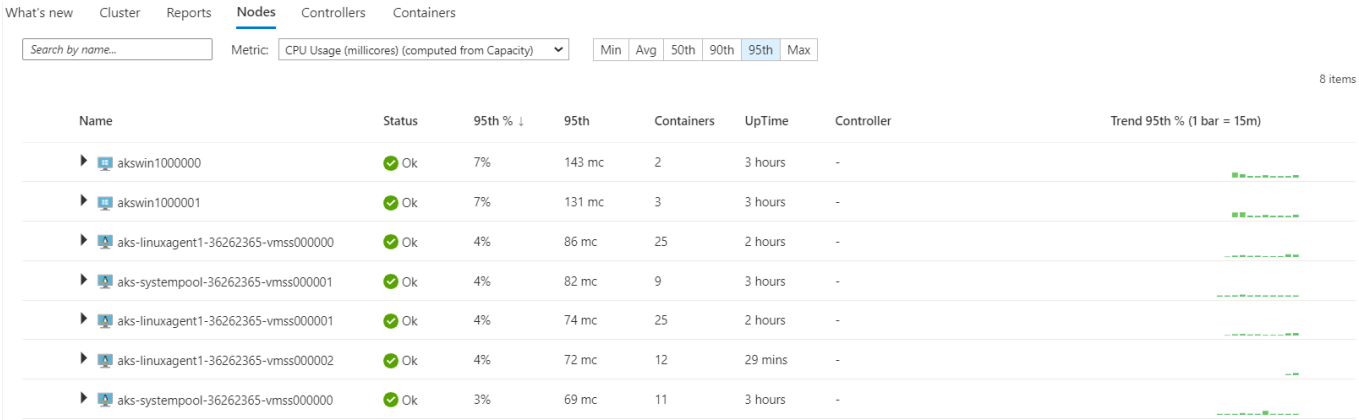
Task 1 - Review Container Insights

1. Open the Azure Portal in a browser.
2. Search for "Kubernetes". Click on your cluster.
3. Scroll down to **Insights** on the option blade.
4. Navigate through the various tabs.

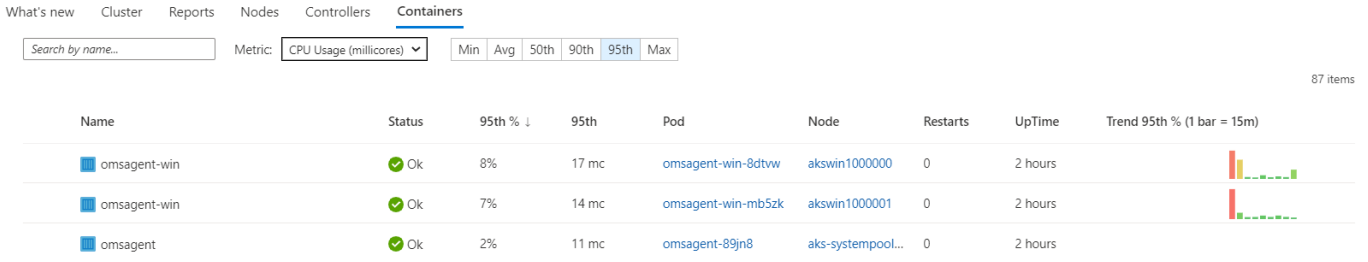
Cluster metrics overview:



Nodes list:

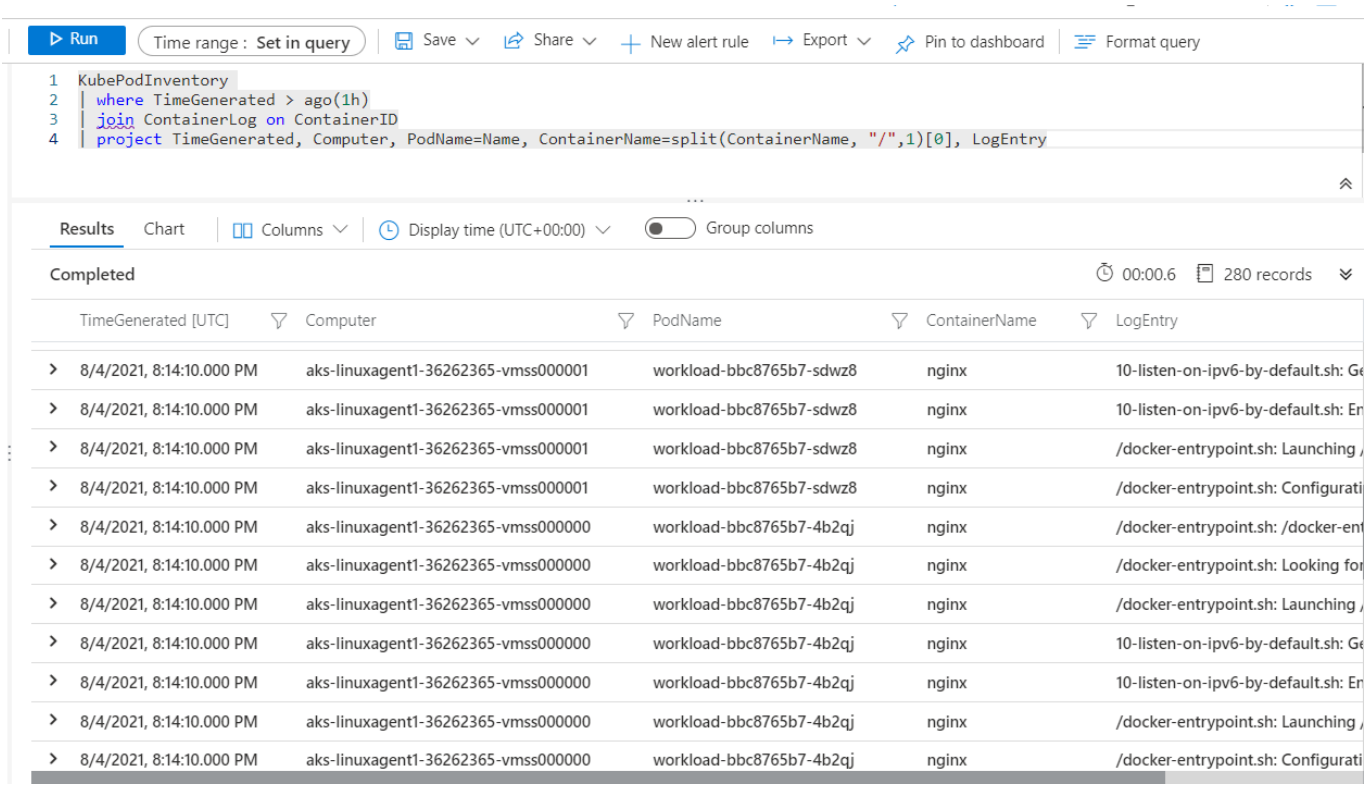


Containers:



5. Scroll down to **Logs** on the option blade.
6. Copy this query into the editor and click **Run**

```
KubePodInventory
| where TimeGenerated > ago(1h)
| join ContainerLog on ContainerID
| project TimeGenerated, Computer, PodName=Name,
ContainerName=split(ContainerName, "/",1)[0], LogEntry
```



You'll see how the container logs are sent to a Log Analytics Workspace for analysis.

Sample Kusto Queries for Container Insights

The link below list some common Kusto queries for Container Insights data:

https://docs.microsoft.com/en-us/azure/azure-monitor/containers/container-insights-log-query

Exercise: Cleanup Resources

Task 1 - Delete the cluster and its resources - Azure CLI

When you're done working with the cluster, you can delete it. You have the complete instructions here on how to recreate it.

It's a good idea to repeat this lab several times, changing some of the settings, in order to get the hang of working with AKS clusters.

1. Deleting the cluster is much easier than creating it.

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

2. You can skip deleting these resources if you're planning on recreating just the cluster at a later time.

3. Delete the Log Analytics Workspace.

```
az monitor log-analytics workspace delete --resource-group $AKS_RESOURCE_GROUP --
workspace-name $LOG_ANALYTICS_WORKSPACE_NAME
```

4. Delete the Virtual Network

```
az network vnet delete --resource-group $AKS_RESOURCE_GROUP --name $AKS_VNET
```

> You could go through and delete individual resources above or you could delete the ENTIRE resource group, which will delete everything in it. Only do this if you place to recreate all the supporting resources

5. Delete the entire resource group

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

Task 2 - Delete the cluster and its resources - Terraform

1. If you used Terraform to create your cluster, you can delete the cluster and all it supporting resources with the following command:

```
terraform destroy -auto-approve
```