Lab 09c - Implement Azure Kubernetes Service

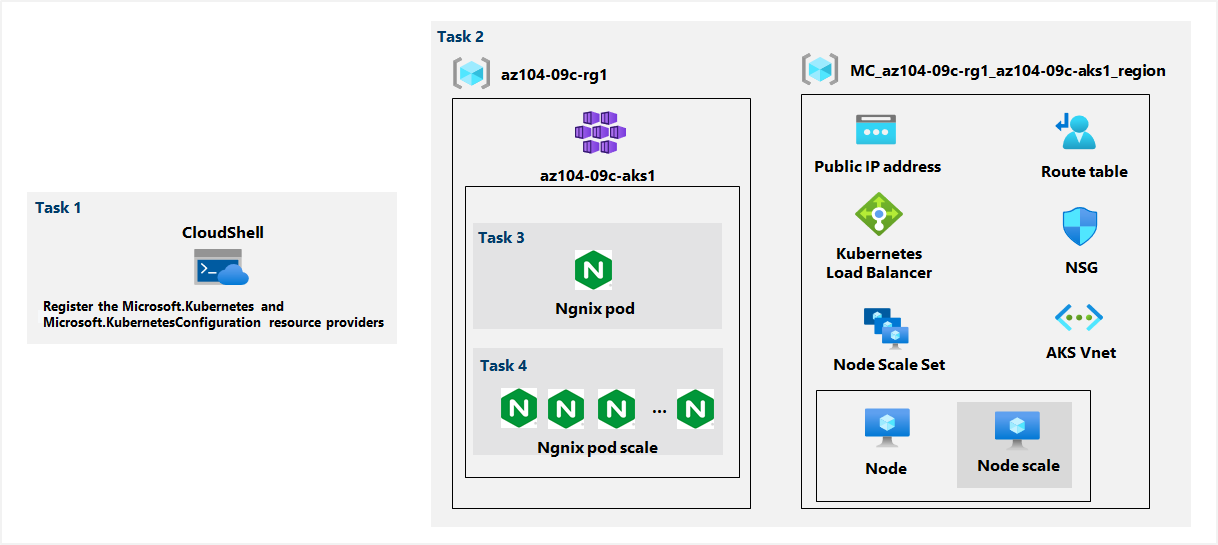
Lab scenario

Contoso has a number of multi-tier applications that are not suitable to run by using Azure Container Instances. In order to determine whether they can be run as containerized workloads, you want to evaluate using Kubernetes as the container orchestrator. To further minimize management overhead, you want to test Azure Kubernetes Service, including its simplified deployment experience and scaling capabilities.

Objectives

In this lab, you will:

* Task 1: Register the Microsoft.Kubernetes and Microsoft.KubernetesConfiguration resource providers.
* Task 2: Deploy an Azure Kubernetes Service cluster
* Task 3: Deploy pods into the Azure Kubernetes Service cluster
* Task 4: Scale containerized workloads in the Azure Kubernetes service cluster



Exercise 1

Task 1: Register the Microsoft.Kubernetes and Microsoft.KubernetesConfiguration resource providers.

We sign in to the Azure portal and open the Azure Cloud Shell.

From there we select PowerShell and run the following to register the Microsoft.Kubernetes and Microsoft.KubernetesConfiguration resource providers:

Register-AzResourceProvider -ProviderNamespace Microsoft.Kubernetes

Register-AzResourceProvider -ProviderNamespace Microsoft.KubernetesConfiguration

Text

Description automatically generated

Task 2: Deploy an Azure Kubernetes Service cluster

In the Azure portal we search for Kubernetes services and then create a Kubernetes cluster.

On the basics tab we specify the following settings:

Resource group: **az104-09c-rg1**

**Cluster present configuration: Dev/Test ($)**

**Kubernetes cluster name: az104-9c-aks1**

**Region: East US**

Availability zones: **None**

Scale method: **Manual**

Node count: **1**

We leave the rest to the default values.

On the Node Pools tab we make sure that the value of Enable virtual nodes is **Disabled** which is the default.

On the Access tab we leave the settings with their default values:

Resource identity: **System-assigned managed identity**

Authentication method: **Local accounts with Kubernetes RBAC**

On the Networking tab we specify the following settings:

Network configuration: **kubenet**

DNS name prefix: **vlatko283**

On the Integrations tab we specify the following settings:

Container monitoring: **Disable**

Enable recommended alert rules: **Uncheck**

This was the last step so we can now go to Review+create and then Create the cluster.

Graphical user interface, text

Description automatically generated with medium confidence

#### Task 3: Deploy pods into the Azure Kubernetes Service cluster

We navigate to the new resource and in the Settings section, click **Node pools.** Here we can verify that the cluster consists of a single pool with one node.

We go back to the Azure Cloud Shell and switch to **Bash**. From here we run the following to retrieve the credentials to access the AKS cluster:

RESOURCE\_GROUP='az104-09c-rg1'

AKS\_CLUSTER='az104-9c-aks1'

az aks get-credentials --resource-group $RESOURCE\_GROUP --name $AKS\_CLUSTER

And then we verify the connectivity to the AKS cluster by typing **kubectl get nodes.**

Now we need to deploy the nginx image from the Docker Hub:

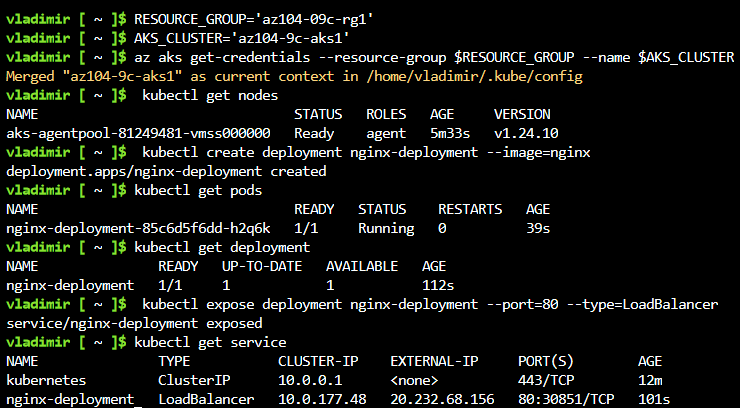
kubectl create deployment nginx-deployment --image=nginx

and we verify the creation with **kubectl get pods.**

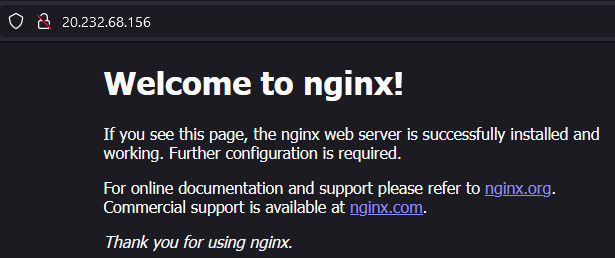
We identify the state of the deployment with **kubectl get deployment**, and to make the pod available from the internet we enter the following command:

kubectl expose deployment nginx-deployment --port=80 --type=LoadBalancer

The last thing to do is check If a public IP address has been provisioned, and we can check that by typing **kubectl get service**

****

We can now open a browser window and navigate to the IP address to see the **Welcome to nginx!** page



#### Task 4: Scale containerized workloads in the Azure Kubernetes service cluster

From the Cloud Shell we increase the number of pods to 2:

kubectl scale --replicas=2 deployment/nginx-deployment

and enter **kubectl get pods** to verify the outcome

Now we scale out the cluster by increasing the number of nodes to 2:

RESOURCE\_GROUP='az104-09c-rg1'

AKS\_CLUSTER='az104-9c-aks1'

az aks scale --resource-group $RESOURCE\_GROUP --name $AKS\_CLUSTER --node-count 2

We verify the outcome with the command **kubectl get nodes**

In order to scale the deployment we enter the following:

kubectl scale --replicas=10 deployment/nginx-deployment

and then verify with the command **kubectl get pods.**

We run the following command to review the pods distribution across cluster nodes:

kubectl get pod -o=custom-columns=NODE:.spec.nodeName,POD:.metadata.name

In order to delete the deployment we enter the following:

kubectl delete deployment nginx-deployment

