Exercise - Deploy Kubernetes with Azure Kubernetes Service

100 XP

15 minutes

Fruit Smoothies wants to use Kubernetes as their *compute* platform. The development teams already use containers for application development and deployment, and using an orchestration platform will help them rapidly build, deliver, and scale their application.

To do this, you need to deploy the foundation of your Kubernetes environment.

In this exercise, you will:

- Create a new resource group.
- Configure cluster networking.
- Create an Azure Kubernetes Service cluster.
- Connect to the Kubernetes cluster by using `kubectl`.
- Create a Kubernetes namespace.

> **Important**
>
> You need your own Azure subscription to run this exercise and you may incur charges. If you don't already have an Azure subscription, create a free account before you begin.

Create a new resource group

You'll first need to create a resource group for your resources to deploy into.

1. Sign in to Azure Cloud Shell with your Azure account.
2. We're going to reuse some values throughout the deployment scripts. For example, you need to choose a region where you want to create a resource group, for example, **East US**. If you select a different value, remember it for the rest of the exercises in this module. You may need to redefine the value between Cloud Shell sessions. Run the following commands to record these values in Bash variables.

   Azure CLI — Copy

   ```
   REGION_NAME=eastus
   RESOURCE_GROUP=akswkorkshop
   SUBNET_NAME=aks-subnet
   VNET_NAME=aks-vnet
   ```

   > Tip
   >
   > You can use the **Copy** button to copy commands to the clipboard. To paste, right-click on a new line in the Cloud Shell window and select **Paste** or use the `Shift+Insert` keyboard shortcut (`⌘+v` on macOS).

   You can check each value using the `echo` command, for example, `echo $REGION_NAME`.
3. Create a new resource group with the name **akswkorkshop**. Deploy all resources created in these exercises in this resource group. A single resource group makes it easier to clean up the resources after you finish the module.

   Azure CLI — Copy

   ```
   az group create \
       --name $RESOURCE_GROUP \
       --location $REGION_NAME
   ```

Configure networking

We have two network models to choose from when deploying an AKS cluster. The first model is *Kubenet networking*, and the second is *Azure Container Networking Interface (CNI) networking*.

What is Kubenet networking?

Kubenet networking is the default networking model in Kubernetes. With Kubenet networking, nodes get assigned an IP address from the Azure virtual network subnet. Pods receive an IP address from a logically different address space to the Azure virtual network subnet of the nodes.

Network address translation (NAT) is then configured so that the pods can reach resources on the Azure virtual network. The source IP address of the traffic is translated to the node's primary IP address and then configured on the nodes. Note, that pods receive an IP address that's "hidden" behind the node IP.

What is Azure Container Networking Interface (CNI) networking?

With Azure Container Networking Interface (CNI), the AKS cluster is connected to existing virtual network resources and configurations. In this networking model, every pod gets an IP address from the subnet and can be accessed directly. These IP addresses must be unique across your network space and calculated in advance.

Some of the features you'll use require you to deploy the AKS cluster by using the *Azure Container Networking Interface networking* configuration.

For a more detailed comparison, see the **Learn more** section at the end of this module.

Let's create the virtual network for your AKS cluster. We will use this virtual network and specify the networking model when we deploy the cluster.

1. First, create a virtual network and subnet. Pods deployed in your cluster will be assigned an IP from this subnet. Run the following command to create the virtual network.

   Azure CLI — Copy

   ```
   az network vnet create \
       --resource-group $RESOURCE_GROUP \
       --location $REGION_NAME \
       --name $VNET_NAME \
       --address-prefixes 10.0.0.0/8 \
       --subnet-name $SUBNET_NAME \
       --subnet-prefix 10.240.0.0/16
   ```
2. Next, retrieve, and store the subnet ID in a Bash variable by running the command below.

   Azure CLI — Copy

   ```
   SUBNET_ID=$(az network vnet subnet show \
       --resource-group $RESOURCE_GROUP \
       --vnet-name $VNET_NAME \
       --name $SUBNET_NAME \
       --query id -o tsv)
   ```

Create the AKS cluster

With the new virtual network in place, you can go ahead and create your new cluster. There are two values you need to know before running the `az aks create` command. The first is the version of the latest, non-preview, Kubernetes version available in your selected region, and the second is a unique name for your cluster.

1. To get the latest, non-preview, Kubernetes version you use the `az aks get-versions` command. Store the value that returns from the command in a Bash variable named `VERSION`. Run the command below the retrieve and store the version number.

   Azure CLI — Copy

   ```
   VERSION=$(az aks get-versions \
       --location $REGION_NAME \
       --query 'orchestrators[?isPreview] | [-1].orchestratorVersion' \
       --output tsv)
   ```
2. The *AKS cluster name* must be unique. Run the following command to create a Bash variable that holds a unique name.

   bash — Copy

   ```
   AKS_CLUSTER_NAME=akswkorkshop-$RANDOM
   ```
3. Run the following command to output the value stored in `$AKS_CLUSTER_NAME`. Note this for later use. You'll need it to reconfigure the variable in the future, if necessary.

   bash — Copy

   ```
   echo $AKS_CLUSTER_NAME
   ```
4. Run the following `az aks create` command to create the AKS cluster running the latest Kubernetes version. This command can take a few minutes to complete.

   Azure CLI — Copy

   ```
   az aks create \
   --resource-group $RESOURCE_GROUP \
   --name $AKS_CLUSTER_NAME \
   --vm-set-type VirtualMachineScaleSets \
   --load-balancer-sku standard \
   --location $REGION_NAME \
   --kubernetes-version $VERSION \
   --network-plugin azure \
   --vnet-subnet-id $SUBNET_NAME_ID \
   --service-cidr 10.2.0.0/24 \
   --dns-service-ip 10.2.0.10 \
   --docker-bridge-address 172.17.0.1/16 \
   --generate-ssh-keys
   ```

   Let's review the variables in the previous command:

   - `$AKS_CLUSTER_NAME` specifies the name of the AKS cluster.
   - `$VERSION` is the latest Kubernetes version you retrieved earlier.
   - `$SUBNET_NAME_ID` is the ID of the subnet created on the virtual network to be configured with AKS.

   Note the following deployment configuration:

   - `--vm-set-type`: We're specifying that the cluster is created by using virtual machine scale sets. The virtual machine scale sets enable you to switch on the cluster autoscaler when needed.
   - `--network-plugin`: We're specifying the creation of the AKS cluster by using the CNI plug-in.
   - `--service-cidr`: This address range is the set of virtual IPs that Kubernetes assigns to internal services in your cluster. The range must not be within the virtual network IP address range of your cluster. It should be different from the subnet created for the pods.
   - `--dns-service-ip`: The IP address is for the cluster's DNS service. This address must be within the *Kubernetes service address range*. Don't use the first IP address in the address range, such as 0.1. The first address in the subnet range is used for the *kubernetes.default.svc.cluster.local* address.
   - `--docker-bridge-address`: The Docker bridge network address represents the default *docker0* bridge network address present in all Docker installations. AKS clusters or the pods themselves don't use *docker0 bridge*. However, you have to set this address to continue supporting scenarios such as *docker build* within the AKS cluster. It's required to select a classless inter-domain routing (CIDR) for the Docker bridge network address. If you don't set a CIDR, Docker chooses a subnet automatically. This subnet could conflict with other CIDRs. Choose an address space that doesn't collide with the rest of the CIDRs on your networks, which includes the cluster's service CIDR and pod CIDR.

Test cluster connectivity by using `kubectl`

kubectl is the main Kubernetes command-line client you use to interact with your cluster and is available in Cloud Shell. A cluster context is required to allow *kubectl* to connect to a cluster. The context contains the cluster's address, a user, and a namespace. Use the `az aks get-credentials` command to configure your instance of *kubectl*.

1. Retrieve the cluster credentials by running the command below.

   Azure CLI — Copy

   ```
   az aks get-credentials \
       --resource-group $RESOURCE_GROUP \
       --name $AKS_CLUSTER_NAME
   ```
2. Let's take a look at what was deployed by listing all the nodes in your cluster. Use the `kubectl get nodes` command to list all the nodes.

   bash — Copy

   ```
   kubectl get nodes
   ```

   You'll see a list of your cluster's nodes. Here's an example.

   output — Copy

   ```
   NAME                                STATUS   ROLES   AGE  VERSION
   aks-nodepool1-24503160-vmss000000   Ready    agent   1m   v1.15.7
   aks-nodepool1-24503160-vmss000001   Ready    agent   1m   v1.15.7
   aks-nodepool1-24503160-vmss000002   Ready    agent   1m   v1.15.7
   ```

Create a Kubernetes namespace for the application

Fruit Smoothies want to deploy several apps from other teams in the deployed AKS cluster as well. Instead of running multiple clusters, the company wants to use the Kubernetes features that let you logically isolate teams and workloads in the same cluster. The goal is to provide the least number of privileges scoped to the resources each team needs.

What is a namespace?

A namespace in Kubernetes creates a logical isolation boundary. Names of resources must be unique within a namespace but not across namespaces. If you don't specify the namespace when you work with Kubernetes resources, the *default* namespace is implied.

Let's create a namespace for your ratings application.

1. List the current namespaces in the cluster.

   bash — Copy

   ```
   kubectl get namespace
   ```

   You'll see a list of namespaces similar to this output.

   output — Copy

   ```
   NAME              STATUS   AGE
   default           Active   1h
   kube-node-lease   Active   1h
   kube-public       Active   1h
   kube-system       Active   1h
   ```
2. Use the `kubectl create namespace` command to create a namespace for the application called **ratingsapp**.

   bash — Copy

   ```
   kubectl create namespace *ratingsapp*
   ```

   You'll see a confirmation that the namespace was created.

   output — Copy

   ```
   namespace/ratingsapp created
   ```

Summary

In this exercise, you created a resource group for your resources. You created a virtual network for your cluster to use. You then deployed your AKS cluster, including the Azure CNI networking mode. You then connected to your cluster with *kubectl* and created a namespace for your Kubernetes resources.

Next, you'll create and configure an Azure Container Registry (ACR) instance to use with your AKS cluster and store your containerized ratings app.

Next unit: Exercise - Create a private, highly available container registry

Continue