**Lab: Deploying compute workloads by using images and containers**

**Student lab manual**

**Lab scenario**

Your organization is seeking a way to automatically create virtual machines (VMs) to run tasks and immediately terminate. You are tasked with evaluating multiple compute services in Microsoft Azure and determining which service can help you automatically create virtualized machines and install custom software on those machines. As a proof of concept, you have decided to try to create VMs from VHD images and container images so that you can compare the two solutions. To keep your proof of concept simple, you will create a special “IP check” application, written in .NET Core, that you will automatically deploy to your machines. Your proof of concept will evaluate the Azure Container Instances and Azure Virtual Machines services.

**Objectives**

After you complete this lab, you will be able to:

* Create a VM manually or by using tools in the Azure portal.
* Deploy a Docker container image to Azure Container Registry.
* Deploy a container from a container image in Azure Container Registry by using Azure Container Instances.
* Deploy a VM using an Azure Resource Manager template and a container image in Azure Container Registry.

**Lab Setup**

* **Estimated Time**: 75 minutes

**Exercise 1: Create a virtual machine using the Azure portal**

**Task 1: Open the Azure portal**

1. Sign in to the **Azure portal** ([https://portal.azure.com](https://portal.azure.com/)).
2. If this is your first time signing in to the Azure portal, a dialog box will appear and a tour of the portal will start. Select the **Get Started** button to skip the tour.

**Task 2: Create a resource group**

1. Create a new **resource group** with the following details:
   1. **Name**: ContainerCompute
   2. **Location**: (US) East US
2. Wait for the creation task to complete before moving forward with this lab.

**Task 3: Create a Linux virtual machine resource**

1. Create a new **Virtual Machine** with the following details:
   * **Operating system**: Ubuntu Server 18.04 LTS
   * **Name**: simplevm
   * **Disk type**: SSD
   * **Username**: Student
   * **Password**: StudentPa55w.rd
   * **Resource group**: ContainerCompute
   * **Location**: (US) East US
   * **Size**: Standard B1s
   * **Public inbound ports**: SSH (22)
2. Wait for the creation task to complete before moving on with this lab.

**Task 4: Validate the virtual machine**

1. Access the **simplevm** VM that you created earlier in this lab.
2. Open the **Connect to virtual machine** pop-up.
3. Copy the **SSH command** to connect to the VM by using port **22** and the **Public IP address**. You will use this command later in this lab.
4. Open a new **Cloud Shell** instance in the Azure Portal.
5. If the **Cloud Shell** is not already configured, configure the shell for Bash by using the default settings.
6. Within the **Cloud Shell** command prompt, use the **SSH command** that you copied earlier in this lab to connect to the VM by using SSH.
7. During the connection process, you will receive a warning that the authenticity of the host cannot be verified. Continue connecting to the host. Finally, use the password **StudentPa55w.rd** when prompted for credentials
8. When you are connected to the VM, use the following commands to view information about the machine:

uname -mnipo

uname -srv

1. Close the **Cloud Shell** pane.

**Review**

In this exercise, you created a new VM manually by using the Azure portal interface and connected to the VM by using the Cloud Shell and secure shell (SSH).

**Exercise 2: Create a virtual machine by using Azure CLI**

**Task 1: Open Cloud Shell**

1. Open a new **Cloud Shell** instance in the Azure Portal.

**Task 2: Use the Azure CLI commands**

1. Use the **az** command with the **--help** flag to view a list of subgroups and commands at the root level of the CLI.
2. Use the **az vm** command with the **--help** flag to view a list of subgroups and commands for **Virtual Machines**:
3. Use the **az vm create** command with the **--help** flag to view a list of arguments and examples for the **Create Virtual Machine** command:
4. Use the **az vm create** command to create a new **Virtual Machine** with the following settings:
   * **Resource group**: ContainerCompute
   * **Name**: quickvm
   * **Image**: Debian
   * **Username**: student
   * **Password**: StudentPa55w.rd
5. Wait for the VM creation process to complete. After the process completes, the command will return a JSON file with details about the machine.
6. Use the **az vm show** command to view a more detailed JSON file that contains various metadata about the newly created VM.
7. Use the **az vm list-ip-addresses** command to list all the IP addresses associated with the VM:

az vm list-ip-addresses --resource-group ContainerCompute --name quickvm

1. Use the **az vm list-ip-addresses** command and the **--query** argument to filter the output to only return the first IP address value:

az vm list-ip-addresses --resource-group ContainerCompute --name quickvm --query '[].{ip:virtualMachine.network.publicIpAddresses[0].ipAddress}' --output tsv

1. Use the following script to store the results of the previous command in a new Bash shell variable named *ipAddress*:

ipAddress=$(az vm list-ip-addresses --resource-group ContainerCompute --name quickvm --query '[].{ip:virtualMachine.network.publicIpAddresses[0].ipAddress}' --output tsv)

1. Use the following script to print the value of the Bash shell variable *ipAddress*:

echo $ipAddress

1. Use the following script to connect to the VM that you created earlier in this lab by using the SSH tool and the IP address stored in the Bash shell variable *ipAddress*:

ssh student@$ipAddress

1. During the connection process, you will receive a warning that the authenticity of the host cannot be verified. Continue connecting to the host. Finally, use the password **StudentPa55w.rd** when prompted for credentials
2. After you are connected to the VM, use the following commands to observe information about the machine to ensure that you are connected to the correct VM:

uname -mnipo

uname -srv

1. Close the **Cloud Shell** pane.

**Review**

In this exercise, you used the Azure Cloud Shell to create a virtual machine as part of an automated script.

**Exercise 3: Create a Docker container image and deploy it to Azure Container Registry**

**Task 1: Open the Cloud Shell and editor**

1. Open a new **Cloud Shell** instance in the Azure Portal.
2. Within the **Cloud Shell** command prompt, change the active directory to **~/clouddrive**.

**Note**: The command to change directory in Bash is **cd <path>**.

1. In the Cloud Shell command prompt, create a new directory named **ipcheck** within the **~/clouddrive** directory.

**Note**: The command to create a new directory in Linux is **mkdir <directory name>**.

1. Change the active directory to **~/clouddrive/ipcheck**.
2. Use the **dotnet new** command to create a new .NET Core console application named **ipcheck** in the current directory.

dotnet new console --name ipcheck --output .

1. Create a new file in the **~/clouddrive/ipcheck** directory named **Dockerfile**.

**Note**: The command to create a new file in Bash is **touch <file name>**. The filename **Dockerfile** is case sensitive.

1. Open the embedded graphical editor in the context of the current directory.

**Note**: You can open the editor by either using the **code .** command or selecting the editor button.

**Task 2: Create and test a .NET Core application**

1. In the graphical editor, open the **Program.cs** file and replace its contents with the following code, and then save the file:

public class Program

{

public static void Main(string[] args)

{

if (System.Net.NetworkInformation.NetworkInterface.GetIsNetworkAvailable())

{

System.Console.WriteLine("Current IP Addresses:");

string hostname = System.Net.Dns.GetHostName();

System.Net.IPHostEntry host = System.Net.Dns.GetHostEntry(hostname);

foreach(System.Net.IPAddress address in host.AddressList)

{

System.Console.WriteLine($"\t{address}");

}

}

else

{

System.Console.WriteLine("No Network Connection");

}

}

}

1. Use the **dotnet run** command in the command prompt to execute the application and validate that it finds one or more IP addresses.
2. Open the **Dockerfile** file in the graphical editor, replace its contents with the following code, and then **Save** the file:

FROM mcr.microsoft.com/dotnet/core/sdk:2.2-alpine AS build

WORKDIR /app

COPY \*.csproj ./

RUN dotnet restore

COPY . ./

RUN dotnet publish --configuration Release --output out

FROM mcr.microsoft.com/dotnet/core/runtime:2.2-alpine

WORKDIR /app

COPY --from=build /app/out .

ENTRYPOINT ["dotnet", "ipcheck.dll"]

1. Close the **Cloud Shell** pane.

**Task 3: Create an Azure Container Registry resource**

1. Create a new **container registry** with the following details:
   * **Name**: <Any globally unique name>
   * **Resource group**: ContainerCompute
   * **Location**: East US
   * **Admin user**: Disable
   * **SKU**: Basic
2. Wait for the creation task to complete before moving on with this lab.

**Task 4: Open Cloud Shell and store Azure Container Registry metadata**

1. Open a new **Cloud Shell** instance.
2. Within the **Cloud Shell** command prompt, use the **az acr list** command to view a list of all **container registries** in your subscription.
3. Use the following command to output the name of the most recently created **container registry**:

az acr list --query "max\_by([], &creationDate).name" --output tsv

1. Use the following command to save the name of the most recently created **container registry** in a Bash shell variable *acrName*:

acrName=$(az acr list --query "max\_by([], &creationDate).name" --output tsv)

1. Use the following script to print the value of the Bash shell variable *acrName*:

echo $acrName

**Task 5: Deploy a Docker container image to Azure Container Registry**

1. Change the active directory to **~/clouddrive/ipcheck**.
2. Use the **dir** command to view the contents of the current directory.

**Note**: You will know that you are in the correct directory if both the **Program.cs** and **Dockerfile** files that you edited earlier in this lab are there.

1. Use the following command to upload the source code to your **container registry** and build the container image as an **Azure Container Registry Task**:

az acr build --registry $acrName --image ipcheck:latest .

1. Wait for the build task to complete before moving forward with this lab.
2. Close the **Cloud Shell** pane.

**Task 6: Validate your container image in Azure Container Registry**

1. Access the container registry that you created earlier in this lab.
2. Select the **Repositories** link to view your images stored in the registry.
3. Proceed through the **Images** and **Tags** blades to view the metadata associated with the **ipcheck** image with the **latest** tag.

**Note**: You can also select the Run ID hyperlink to view the build task metadata.

**Review**

In this exercise, you created a .NET Core console application to display a machine’s current IP address. You then added the Dockerfile file to the application so that it could be converted into a Docker container image. Finally, you deployed the container image to Azure Container Registry.

**Exercise 4: Deploy an Azure container instance**

**Task 1: Enable Admin User in Azure Container Registry**

1. Access the container registry that you created earlier in this lab.
2. Select the **Update** button to view the settings for the Container Registry.
3. **Enable** the **Admin User**.

**Note**: Your changes are automatically saved.

**Task 2: Deploy a container image automatically to Azure container instance**

1. Select the **Repositories** link to view your images stored in the registry.
2. Select the **ipcheck** image and view the **latest** tag for that image.
3. Select/right-click the **latest** tag for the **ipcheck** container image to run a new **Azure container instance** with the following settings:
   * **Container name**: managedcompute
   * **OS type**: Linux
   * **Resource group**: ContainerCompute
   * **Location**: East US
   * **Number of cores**: 2
   * **Memory (GB)**: 4
   * **Public IP address**: No
4. Wait for the creation task to complete before moving on with this lab.

**Task 3: Deploy a container image manually to an Azure container instance**

1. Access the container registry that you created earlier in this lab.
2. Select the **Access keys** link to view the credentials necessary to access your container registry from another service. Record the following values from this section to use later in this lab:
   * **Login server**
   * **Username**
   * **Password**

**Note**: You will use these values later in this lab when you create another container instance.

1. Create a new **container instance** with the following details, using the **Access keys** credentials you recorded earlier in this lab:
   * **Container name**: manualcompute
   * **Image type**: Private
   * **Image name**: <Login server recorded earlier in the lab >/ipcheck:latest
   * **Image registry login server**: <Login server recorded earlier in the lab>
   * **Image registry user name**: <Username recorded earlier in the lab >
   * **Image registry password**: <Password recorded earlier in the lab >
   * **Resource group**: ContainerCompute
   * **Region**: East US
   * **OS type**: Linux
   * **Number of cores**: 1
   * **Memory (GB)**: 1.5
   * **Public IP address**: Yes
   * **Port**: 80
   * **Open additional ports**: No
   * **Port protocol**: TCP
   * **Restart policy**: On failure
2. Wait for the creation task to complete before moving forward with this lab.

**Task 4: Validate that the container instance ran successfully**

1. Access the **manualcompute** container instance that you created earlier in this lab.
2. Select the **Containers** link to view a list of the current containers running.
3. View the contents of the **Events** list for the container instance that ran your **ipcheck** application.
4. Select the **Logs** tab and observe the text logs from the container instance.

**Note**: You can also optionally view the **Events** and **Logs** from the **managedcompute** container instance.

**Review**

In this exercise, you used multiple methods to deploy a container image to an Azure container instance. By using the manual method, you were also able to customize the deployment further and execute task-based applications as part of a container run.

**Exercise 5: Clean up subscription**

**Task 1: Open Cloud Shell and list resource groups**

1. At the top of the Azure portal, select the **Cloud Shell** icon to open a new shell instance.
2. In the **Cloud Shell** command prompt at the bottom of the portal, type the following command and press Enter to list all resource groups in the subscription:

az group list

1. Type the following command and press Enter to view a list of possible commands to delete a resource group:

az group delete --help

**Task 2: Delete resource groups**

1. Type the following command and press Enter to delete the **ContainerCompute** resource group:

az group delete --name ContainerCompute --no-wait --yes

1. Close the **Cloud Shell** pane at the bottom of the portal.

**Task 3: Close active applications**

1. Close the currently running **Microsoft Edge** application.

**Review**

In this exercise, you cleaned up your subscription by removing the **resource groups** used in this lab.