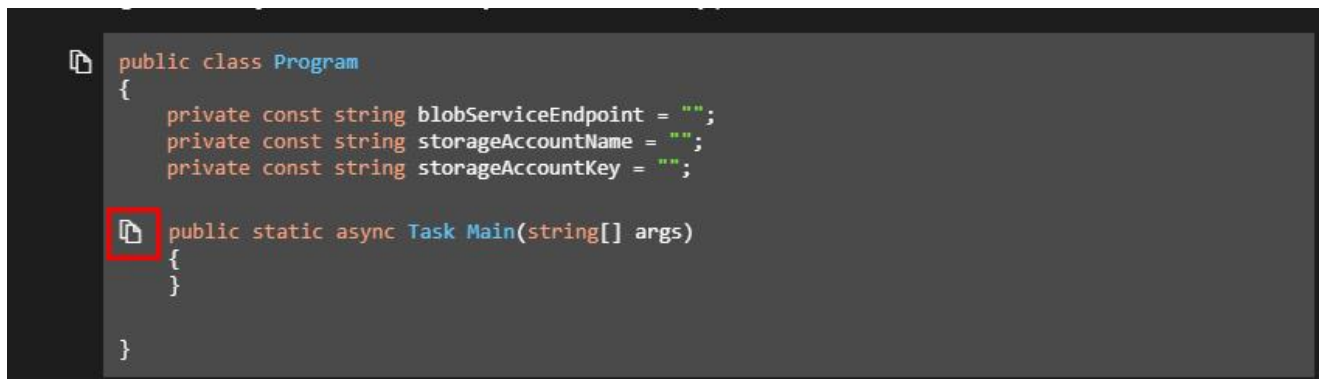


TypeText must be input into **Notepad** and then copied into **VS Code** in order to maintain formatting.

**Do not** click nested Typetext icons. Doing so will produce the wrong output.



```
public class Program
{
    private const string blobServiceEndpoint = "";
    private const string storageAccountName = "";
    private const string storageAccountKey = "";

    public static async Task Main(string[] args)
    {
    }
}
```

At the end of this lab, you can skip the **Clean Up** exercise directing you to remove the resources from your Subscription or Resource Group(s). The clean up is handled automatically, after ending your lab.

## 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're tasked with evaluating multiple compute services in Microsoft Azure and determining which service can help you automatically create VMs and install custom software on those machines. As a proof of concept, you have decided to try creating VMs from built-in images and container images so that you can compare the two solutions. To keep your proof of concept simple, you'll create a special "IP check" application written in .NET that you'll 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 by using the Azure Command-Line Interface (CLI).
- Deploy a Docker container image to Azure Container Registry.
- Deploy a container from a container image in Container Registry by using Container Instances.

#### Lab setup

- Estimated time: **45 minutes**

#### Instructions

##### Before you start

##### Sign in to the lab VM

Ensure that you're signed in to your Windows 10 VM by using the following credentials:

- Username: **Admin**
- Password: **Pa55w.rd**

## Review the installed applications

Find the taskbar on your Windows 10 desktop. The taskbar contains the icons for the applications that you'll use in this lab:

- Microsoft Edge
- File Explorer

## Exercise 1: Create a VM by using the Azure CLI

### Task 1: Open the Azure portal

1. Sign in to the Azure portal (<https://portal.azure.com>).
2. If this is your first time signing in to the Azure portal, you'll notice a dialog box offering a tour of the portal. Select **Get Started** to skip the tour.

### Task 2: Create a resource group

1. Create a new resource group with the following details:
  - a. Name: **ContainerCompute**
  - b. Location: **(US) East US**

**Note:** Wait for the creation task to complete before moving forward with this lab.

### Task 3: Open Azure Cloud Shell

1. Open a new Cloud Shell instance in the Azure portal.
2. If the Cloud Shell is not already configured, configure the shell for Bash by using the default settings.

### Task 4: Use the Azure CLI commands

1. Use the **az** command with the **--help** flag to find a list of subgroups and commands at the root level of the CLI.
2. Use the **az vm** command with the **--help** flag to find a list of subgroups and commands for Azure Virtual Machines:
3. Use the **az vm create** command with the **--help** flag to find a list of arguments and examples for the **Create Virtual Machine** command:
4. Use the **az vm create** command to create a new VM with the following settings:
  - Resource group: **ContainerCompute**
  - Name: **quickvm**
  - Image: **Debian**
  - Username: **student**

- Password: **StudentPa55w.rd**

**Note:** Wait for the VM creation process to complete. After the process completes, the command will return a JavaScript Object Notation (JSON) file with details about the machine.

5. Use the **az vm show** command to find a more detailed JSON file that contains various metadata about the newly created VM.
6. 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
```

7. 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 -
```

8. 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 --na
```

9. Use the following script to render the value of the Bash shell variable *ipAddress*:

```
echo $ipAddress
```

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

```
ssh student@$ipAddress
```

11. During the connection process, you'll receive a warning that the authenticity of the host can't be verified. Continue connecting to the host. Finally, use the password **StudentPa55w.rd** when prompted for credentials
12. After connecting to the VM, use the following command to get information about the machine to ensure that you're connected to the correct VM:

```
uname -a
```

13. Use the **exit** command to end your SSH session:

```
exit
```

14. Close the Cloud Shell pane.

## Review

In this exercise, you used Cloud Shell to create a VM as part of an automated script.

## Exercise 2: Create a Docker container image and deploy it to Container Registry

## Task 1: Open the Cloud Shell and editor

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

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

3. At the **Cloud Shell** command prompt, create a new directory named **ipcheck** in the **~/clouddrive** directory.

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

4. Change the active directory to **~/clouddrive/ipcheck**.
5. Use the **dotnet new console --output . --name ipcheck** command to create a new .NET console application in the current directory.
6. 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 file name **Dockerfile** is case sensitive.

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

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

## Task 2: Create and test a .NET 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)
    {
        // Check if network is available
        if (System.Net.NetworkInformation.NetworkInterface.GetIsNetworkAvailable())
        {
            System.Console.WriteLine("Current IP Addresses:");

            // Get host entry for current hostname
            string hostname = System.Net.Dns.GetHostName();
            System.Net.IPHostEntry host = System.Net.Dns.GetHostEntry(hostname);

            // Iterate over each IP address and render their values
            foreach (System.Net.IPAddress address in host.AddressList)
            {
                System.Console.WriteLine($"{address}");
            }
        }
        else
        {
            System.Console.WriteLine("No Network Connection");
        }
    }
}
```

```

    }
}
}

```

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

```

# Start using the .NET Core 2.2 SDK container image
FROM mcr.microsoft.com/dotnet/core/sdk:2.2-alpine AS build

# Change current working directory
WORKDIR /app

# Copy existing files from host machine
COPY . ./

# Publish application to the "out" folder
RUN dotnet publish --configuration Release --output out

# Start container by running application DLL
ENTRYPOINT ["dotnet", "out/ipcheck.dll"]

```

4. Close the Cloud Shell pane.

### Task 3: Create a Container Registry resource

- Create a new container registry with the following details:
  - Name: **Any globally unique name**
  - Resource group: **ContainerCompute**
  - Location: **East US**
  - SKU: **Basic**

**Note:** Wait for the creation task to complete before moving on with this lab.

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

1. Open a new Cloud Shell instance.
2. At the **Cloud Shell** command prompt, use the **az acr list** command to get 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
```

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

```
acrName=$(az acr list --query "max_by([], &creationDate).name" --output ts'
```

5. Use the following script to render the value of the Bash shell variable *acrName*:

```
echo $acrName
```

## Task 5: Deploy a Docker container image to Container Registry

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

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

3. Use the following command to upload the source code to your container registry and build the container image as a Container Registry task:

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

**Note:** Wait for the build task to complete before moving forward with this lab.

4. Close the Cloud Shell pane.

## Task 6: Validate your container image in Container Registry

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

**Note:** You can also select the **Run ID** link to find the build task metadata.

## Review

In this exercise, you created a .NET 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 Container Registry.

## Exercise 3: Deploy an Azure container instance

### Task 1: Enable the admin user in Container Registry

1. Access the container registry that you created earlier in this lab.
2. Select **Update** to find the settings for the container registry.
3. **Enable** the **Admin User**, **Save** your changes, and then close the **Update container registry** blade.

## Task 2: Automatically deploy a container image to an Azure container instance

1. Select the **Repositories** link to find your images in the registry.
2. Select the **ipcheck** image, and then find the **latest** tag for that image.
3. Right-click the **latest** tag or activate the shortcut menu 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**

**Note:** Wait for the creation task to complete before moving on with this lab.

## Task 3: Manually deploy a container image to Container Instances

1. Create a new container instance with the following information :
  - Resource group: **ContainerCompute**
  - Container name: **manualcompute**
  - Region: **East US**
  - Image source: **Azure Container Registry**
  - Registry: **\*Azure Container Registry resource created earlier in the lab**
  - Image: **ipcheck**
  - Image tag: **latest**

**Note:** 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 get a list of the current containers that are running.
3. Find the contents of the **Events** list for the container instance that ran your **ipcheck** application.
4. Select the **Logs** tab, and then find the text logs from the container instance.

**Note:** You can also optionally find 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 run task-based applications as part of a container run.

## Exercise 4: Clean up your subscription

### Task 1: Open Azure Cloud Shell and list resource groups

1. In the portal, select the **Cloud Shell** icon to open a new shell instance.
2. If **Cloud Shell** isn't already configured, configure the shell for Bash by using the default settings.

### Task 2: Delete resource groups

1. Enter the following command, and then select Enter to delete the **ContainerCompute** resource group:

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

2. Close the Cloud Shell pane in the portal.

### Task 3: Close the active applications

- Close the currently running Microsoft Edge application.

## Review

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

---

## Congratulations!

You have successfully completed this exercise. Click **End** to advance to the next exercise.