# Lab: Monitoring services deployed to Azure

## Microsoft Azure user interface

Given the dynamic nature of Microsoft cloud tools, you might experience Azure user interface (UI) changes after the development of this training content. These changes might cause the lab instructions and steps to not match up.

Microsoft updates this training course as soon as the community brings needed changes to our attention. However, because cloud updates occur frequently, you might encounter UI changes before this training content is updated. **If this occurs, adapt to the changes and work through them in the labs as needed.**

## Instructions

### Before you start

#### Sign in to the lab virtual machine

* Sign in to your **Windows 10** virtual machine by using the following credentials:
  + **Username**: Admin
  + **Password**: Pa55w.rd
* **Note**: Lab virtual machine sign-in instructions will be provided to you by your instructor.

#### Review installed applications

* Observe the taskbar located at the bottom of your **Windows 10** desktop. The taskbar contains the icons for the applications that you will use in this lab:
  + Microsoft Edge
  + File Explorer
  + Visual Studio Code
  + Windows PowerShell

#### Download the lab files

1. On the taskbar, select the **Windows PowerShell** icon.
2. In the PowerShell command prompt, change the current working directory to the **Allfiles (F):\** path:

* cd F:

1. Within the command prompt, enter the following command and press Enter to Clone the **microsoftlearning/AZ-203-DevelopingSolutionsForAzure** project hosted on GitHub into the **Labfiles** directory:

* git clone --depth 1 --no-checkout https://github.com/microsoftlearning/AZ-203-DevelopingSolutionsForMicrosoftAzure .

1. Within the command prompt, enter the following command and press **Enter** to check out the lab files necessary to complete the **AZ-201.02** lab:

* git checkout master -- Allfiles/\*

1. Close the currently running **Windows PowerShell** command prompt application.

### Exercise 1: Create and configure Azure resources

#### Task 1: Open the Azure portal

1. On the taskbar, select the **Microsoft Edge** icon.
2. In the open browser window, navigate to the [**Azure portal**](https://portal.azure.com) (portal.azure.com).
3. At the sign-in page, enter the **email address** for your Microsoft account.
4. Select **Next**.
5. Enter the **password** for your Microsoft account.
6. Select **Sign in**.

**Note**: If this is your first time signing in to the **Azure portal**, a dialog box will display an offer to tour the portal. Select **Get Started** to skip the tour and begin using the portal.

#### Task 2: Create an Application Insights resource

1. In the left navigation pane of the portal, select **+ Create a resource**.
2. At the top of the **New** blade, locate the **Search the Marketplace** field.
3. In the search field, enter **Insights** and press Enter.
4. In the **Everything** search results blade, select the **Application Insights** result.
5. In the **Application Insights** blade, select **Create**.
6. In the second **Application Insights** blade, perform the following actions:
   1. Leave the **Subscription** field set to its default value.
   2. In the **Resource group** section, select **Create new**, and then enter **MonitoredAssets**.
   3. In the **Name** field, enter **instrm[*your name in lowercase*]**.
   4. In the **Location** list, select **(US) East US**.
   5. Select **Review + create**
   6. Review the options that you entered in the previous steps.
   7. Select **Create**.
7. Wait for the creation task to complete before you move forward with this lab.
8. In the left navigation pane of the portal, select **Resource groups**.
9. In the **Resource groups** blade, select the **MonitoredAssets** resource group that you created earlier in this lab.
10. In the **MonitoredAssets** blade, select the **instrm\*** Application Insights account that you created earlier in this lab.
11. In the **Application Insights** blade, on the left side of the blade, within the **Configure** category,, select the **Properties** link.
12. In the **Properties** section, observe the value of the **Instrumentation Key** field. This key is used by client applications to connect to Application Insights.

#### Task 3: Create an API App resource

1. In the left navigation pane of the portal, select **+ Create a resource**.
2. At the top of the **New** blade, locate the **Search the Marketplace** field.
3. In the search field, enter **API** and press Enter.
4. In the **Everything** search results blade, select the **API App** result.
5. In the **API App** blade, select **Create**.
6. In the second **API App** blade, perform the following actions:
   1. In the **App name** field, enter **smpapi[*your name in lowercase*]**.
   2. Leave the **Subscription** field set to its default value.
   3. In the **Resource group** section, select **Use existing**, and then select **MonitoredAssets**.
   4. Select the **App Service plan/Location** field.
7. In the **App Service plan** blade, select **Create new**.
8. In the **New App Service Plan** blade, perform the following actions:
   1. In the **App Service plan** field, enter **MonitoredPlan**.
   2. In the **Location** list, select the **East US** location.
   3. Set the **Pricing tier** field to the value **S1 Standard**.
   4. Select **OK**.
9. Back in the **Api App** blade, select the **Application Insights** field.
10. In the **Application Insights** blade, perform the following actions:
    1. In the **Application Insights** section, select **Enable**.
    2. In the **Change your resource** section, select **Select existing resource** and then select the **instrm\*** Application Insights account that you created earlier in this lab.
    3. In the **Instrument your application** section, select the **.NET Core** tab and then select **Recommended**.
    4. In the **Profiler** section, select **On**.
    5. In the **Snapshot debugger** section, select **On**.
    6. In the **SQL Commands** section, select **On**.
    7. Select **Apply**.
11. Back in the **Api App** blade, select **Create**.
12. Wait for the creation task to complete before you move forward with this lab.
13. In the left navigation pane of the portal, select **Resource groups**.
14. In the **Resource groups** blade, select the **MonitoredAssets** resource group that you created earlier in this lab.
15. In the **MonitoredAssets** blade, select the **smpapi\*** API app you that created earlier in this lab.
16. In the **App Service** blade, on the left side of the blade, within the **Settings** category, select the **Configuration** link.
17. In the **Configuration** section, perform the following actions:
    1. Select the **Application settings** tab.
    2. Select **Show Values** to view the secrets associated with your API.
    3. Observe the value corresponding to the **APPINSIGHTS\_INSTRUMENTATIONKEY** key. This value was set automatically when you built your API App resource.
18. In the **App Service** blade, on the left side of the blade within the **Settings** category, select the **Properties** link.
19. In the **Properties** section, record the value of the **URL** field. You will use this value later in the lab to make requests against the API.

#### Task 4: Configure API App auto-scale options

1. In the **App Service** blade, on the left side of the blade, within the **Settings** category, select the **Scale out (App Service Plan)** link.
2. In the **Scale out** section, perform the following actions:
   1. Select **Enable autoscale**.
   2. In the **Autoscale setting name** field, enter **ComputeScaler**.
   3. In the **Resource group** list, select **MonitoredAssets**.
   4. In the **Scale mode** section, select **Scale based on a metric**.
   5. In the **Minimum** field within the **Instance limits** section, enter **2**.
   6. In the **Maximum** field within the **Instance limits** section, enter **8**.
   7. In the **Default** field within the **Instance limits** section, enter **3**.
   8. Select **+ Add a rule**. In the **Scale rule** window that appears, leave all fields set to their default values and then select **Add**.
   9. At the top of the section, select **Save**.
3. Wait for the save operation to complete before you move forward with this lab.

#### Review

In this exercise, you created the resources that you will use for the remainder of the lab.

### Exercise 2: Build and deploy an .NET Core Web API application

#### Task 1: Build a .NET Core Web API project

1. On the taskbar, select the **Visual Studio Code** icon.
2. On the **File** menu, select **Open Folder**.
3. In the File Explorer pane that opens, go to **Allfiles (F):\Allfiles\Labs\05\Starter\Api**, and then select **Select Folder**.
4. In the Visual Studio Code window, access the context menu or right-click the **Explorer** pane and then select **Open in Terminal**.
5. In the open command prompt, enter the following command and press Enter to create a new .NET Core Web API application named **SimpleApi** in the current directory:

* dotnet new webapi --output . --name SimpleApi

1. In the command prompt, enter the following command and press Enter to add the **2.7.1** version of the **Microsoft.ApplicationInsights.AspNetCore** package from NuGet to the current project:

* dotnet add package Microsoft.ApplicationInsights.AspNetCore --version 2.7.1

1. In the command prompt, enter the following command and press Enter to build the .NET Core web application:

* dotnet build

#### Task 2: Update application code to disable HTTPS and use Application Insights

1. On the left side of the **Visual Studio Code** window, in the **Explorer** pane, double-click the **Program.cs** file to open the file in the editor.
2. In the editor, in the **Program** class, locate the following block of code at line **20**:

* public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>  
   WebHost.CreateDefaultBuilder(args)  
   .UseStartup<Startup>();

1. Replace that block of code with the following block of code that enables **Application Insights** telemetry for the project:

* public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>  
   WebHost.CreateDefaultBuilder(args)  
   .UseStartup<Startup>()  
   .UseApplicationInsights();

1. **Save** the **Program.cs** file.
2. On the left side of the **Visual Studio Code** window, in the **Explorer** pane, double-click the **Startup.cs** file to open the file in the editor.
3. In the editor, in the **Program** class, locate and delete the following line of code at line **43**:

* app.UseHttpsRedirection();
* **Note**: This line of code forces the API App to use HTTPS. For this lab, this is unnecessary.

1. **Save** the **Startup.cs** file.
2. Locate the command prompt at the bottom of the screen. In the command prompt, enter the following command and press Enter to build the .NET Core web application.

* dotnet build

#### Task 3: Test an API application locally

1. Locate the command prompt at the bottom of the screen. In the command prompt, enter the following command and press Enter to execute the .NET Core web application.

* dotnet run

1. On the taskbar, select the **Microsoft Edge** icon.
2. In the open browser window, navigate to the **/api/values** relative path of your test application hosted at **localhost** on port **5000**.

* **Note**: The full URL is http://localhost:5000/api/values

1. In the same browser window, navigate to the **/api/values/7** relative path of your test application hosted at **localhost** on port **5000**.

* **Note**: The full URL is http://localhost:5000/api/values/7

1. Close the browser window displaying the http://localhost:5000/api/values/7 address.
2. Close the currently running **Visual Studio Code** application.

#### Task 4: View metrics in Application Insights

1. Return to your currently open browser window displaying the **Azure portal**.
2. On the left side of the portal, select **Resource groups**.
3. In the **Resource groups** blade, locate and select the **MonitoredAssets** resource group that you created earlier in this lab.
4. In the **MonitoredAssets** blade, select the **instrm\*** Application Insights account that you created earlier in this lab.
5. In the **Application Insights** blade, in the tiles located in the center of the blade, observe the metrics displayed. Specifically, observe the number of **server** **requests** that have occurred and the average **server response time**.

#### Task 5: Deploy an application to API App

1. On the taskbar, select the **Visual Studio Code** icon.
2. On the **File** menu, select **Open Folder**.
3. In the File Explorer pane that opens, go to **Allfiles (F):\Allfiles\Labs\Starter\Api**, and then select **Select Folder**.
4. In the Visual Studio Code window, access the context menu or right-click the **Explorer** pane, and then select **Open in Terminal**.
5. In the open command prompt, enter the following command and press Enter to sign in to the Azure CLI:

* az login

1. In the browser window that appears, perform the following actions:
   1. Enter the **email address** for your Microsoft account.
   2. Select **Next**.
   3. Enter the **password** for your Microsoft account.
   4. Select **Sign in**.
2. Return to the currently open **command prompt** application. Wait for the sign-in process to finish.
3. At the command prompt, enter the following command and press Enter to list all the **apps** in your **MonitoredAssets** resource group:

* az webapp list --resource-group MonitoredAssets

1. Enter the following command and press Enter to find the **apps** that have the prefix **smpapi\***:

* az webapp list --resource-group MonitoredAssets --query "[?starts\_with(name, 'smpapi')]"

1. Enter the following command and press Enter to print out only the name of the single app that has the prefix **smpapi\***:

* az webapp list --resource-group MonitoredAssets --query "[?starts\_with(name, 'smpapi')].{Name:name}" --output tsv

1. Enter the following command and press Enter to change the current directory to the **Allfiles (F):\Allfiles\Labs\05\Starter** directory that contains the deployment files:

* cd F:\Allfiles\Labs\05\Starter\

1. Enter the following command and press Enter to deploy the **api.zip** file to the **API app** that you created earlier in this lab:

* az webapp deployment source config-zip --resource-group MonitoredAssets --src api.zip --name <name-of-your-api-app>
* **Note**: Replace the **<name-of-your-api-app>** placeholder with the name of the API app that you created earlier in this lab. You recently queried this app’s name in the previous steps.

1. Wait for the deployment to complete before you move forward with this lab.
2. Close the currently running **Visual Studio Code** application.
3. In the left navigation pane of the portal, select **Resource groups**.
4. In the **Resource groups** blade, select the **MonitoredAssets** resource group that you created earlier in this lab.
5. In the **MonitoredAssets** blade, select the **smpapi\*** API app that you created earlier in this lab.
6. In the **App Service** blade, select **Browse** at the top of the blade.
7. A new browser window or tab will open and return a **404 (Not Found)** error. In the browser address bar, update the URL by appending the suffix **/api/values** to the end of the current URL and then press Enter.

* **Note**: For example, if your URL is http://smpapistudent.azurewebsites.net, the new URL would be http://smpapistudent.azurewebsites.net/api/values.

1. Observe the JSON array that is returned as a result of using the API.

#### Review

In this exercise, you created an API by using ASP.NET Core and configured it to stream application metrics to Application Insights. You then used the Application Insights dashboard to view performance details about your API App and the API running in the app.

### Exercise 3: Build a client application by using .NET Core

#### Task 1: Build a .NET Core console project

1. On the taskbar, select the **Visual Studio Code** icon.
2. On the **File** menu, select **Open Folder**.
3. In the File Explorer pane that opens, go to **Allfiles (F):\Allfiles\Starter\Console**, and then select **Select Folder**.
4. In the Visual Studio Code window, access the context menu or right-click the **Explorer** pane, and then select **Open in Terminal**.
5. In the open command prompt, enter the following command and press Enter to create a new .NET Core console application named **SimpleConsole** in the current directory:

* dotnet new console --output . --name SimpleConsole

1. In the command prompt, enter the following command and press Enter to add the **7.1.0** version of the **Polly** package from NuGet to the current project:

* dotnet add package Polly --version 7.1.0

1. In the command prompt, enter the following command and press Enter to build the .NET Core web application:

* dotnet build

#### Task 2: Add HTTP client code

1. On the left side of the **Visual Studio Code** window, in the **Explorer** pane, double-click the **Program.cs** file to open the file in the editor.
2. In the editor, add the following **using** block for the **System.Net.Http** namespace:

* using System.Net.Http;

1. In the editor, add the following **using** block for the **System.Threading.Tasks** namespace:

* using System.Threading.Tasks;

1. In the **SimpleConsole** namespace, locate the following class at line **7**:

* class Program  
  {  
   static void Main(string[] args)  
   {  
   Console.WriteLine("Hello World!");  
   }  
  }

1. Replace the entire **Program** class with the following implementation:

* class Program  
  {  
   private const string \_api = "";  
   private static HttpClient \_client = new HttpClient(){ BaseAddress = new Uri(\_api) };  
    
   static void Main(string[] args)  
   {  
   Run().Wait();  
   }  
    
   static async Task Run()  
   {  
    
   }  
  }

1. Locate the **\_api** constant at line **9**:

* private const string \_api = "";

1. Update the **\_api** constant by setting the value of the variable to the **URL** of the API app you recorded earlier in this lab:

* **Note**: For example, if your URL is http://smpapistudent.azurewebsites.net, the new line of code will be: private const string \_api = “http://smpapistudent.azurewebsites.net”;

1. Within the **Run** method, add the following line of code to asynchronously invoke the **HttpClient.GetStringAsync** method passing in a string for the relative path of **/api/values/**:

* string response = await \_client.GetStringAsync("/api/values/");

1. Within the **Run** method, add an additional line of code to write out the response from the **GET** request to the console:

* Console.WriteLine(response);

1. Your **Program.cs** file should now have the following code:

* using System;  
  using System.Net.Http;  
  using System.Threading.Tasks;  
    
  namespace SimpleConsole  
  {  
   class Program  
   {  
   private const string \_api = "http://<your-api-name>.azurewebsites.net/";  
   private static HttpClient \_client = new HttpClient(){ BaseAddress = new Uri(\_api) };  
    
   static void Main(string[] args)  
   {  
   Run().Wait();  
   }  
    
   static async Task Run()  
   {  
   string response = await \_client.GetStringAsync("/api/values/");  
   Console.WriteLine(response);   
   }  
   }  
  }

1. **Save** the **Program.cs** file.

#### Task 3: Test a console application locally

1. At the bottom of the screen, in the command prompt, enter the following command and press Enter to execute the .NET Core web application.

* dotnet run

1. Observe that the application successfully invokes the API app in Azure and returns the same JSON array that you observed earlier in this lab. Your result should appear similar to the following JSON content:

* ["value1","value2"]

1. Return to your currently open browser window displaying the **Azure portal**.
2. On the left side of the portal, select **Resource groups**.
3. In the **Resource groups** blade, locate and select the **MonitoredAssets** resource group that you created earlier in this lab.
4. In the **MonitoredAssets** blade, select the **smpapi\*** API app that you created earlier in this lab.
5. In the **App Service** blade, select **Stop** at the top of the blade to halt the execution of the API app.
6. In the **Stop web app** confirmation dialog box, select **Yes**.
7. On the taskbar, select the **Visual Studio Code** icon.
8. On the **File** menu, select **Open Folder**.
9. In the File Explorer pane that opens, go to **Allfiles (F):\Allfiles\Labs\05\Starter\Console**, and then select **Select Folder**.
10. In the Visual Studio Code window, access the context menu or right-click the **Explorer** pane, and then select **Open in Terminal**.
11. In the open command prompt, enter the following command and press Enter to execute the .NET Core web application.

* dotnet run

1. Observe that the application execution fails and displays a **HttpRequestException** message that is similar to the following exception message:

* System.Net.Http.HttpRequestException: Response status code does not indicate  
    
  success: 403 (Site Disabled).  
    
  at System.Net.Http.HttpResponseMessage.EnsureSuccessStatusCode()  
    
  at System.Net.Http.HttpClient.GetStringAsyncCore(Task\`1 getTask)  
    
  at SimpleConsole.Program.Run() in F:\Allfiles\Labs\05\Starter\Console\Program.cs:line 20
* **Note**: This exception occurs because the API app is no longer available.

#### Task 4: Add retry logic by using Polly

1. On the left side of the **Visual Studio Code** window, in the **Explorer** pane double-click the **PollyHandler.cs** file to open the file in the editor.
2. Within the **PollyHandler** class, observe lines **13-24**. These lines of code use the **Polly** library from the **.NET Foundation** to create a retry policy that will retry a failed HTTP request every five minutes.
3. On the left side of the **Visual Studio Code** window, in the **Explorer** pane, double-click the **Program.cs** file to open the file in the editor.
4. Locate the **\_client** constant at line **10**:

* private static HttpClient \_client = new HttpClient(){ BaseAddress = new Uri(\_api) };

1. Update the **\_client** constant by updating the **HttpClient** constructor to use a new instance of the **PollyHandler** class:

* private static HttpClient \_client = new HttpClient(new PollyHandler()){ BaseAddress = new Uri(\_api) };

1. **Save** the **Program.cs** file.

#### Task 5: Validate retry logic

1. At the bottom of the screen, in the command prompt, enter the following command and press Enter to execute the .NET Core web application.

* dotnet run

1. Observe that the HTTP request execution continues to fail and is re-attempted every five seconds. You will observe the following message in the console while the application is failing:

* Failed Attempt

1. Leave the console application running. It will attempt to access the API app infinitely until it is successful.
2. Return to your currently open browser window displaying the **Azure portal**.
3. On the left side of the portal, select **Resource groups**.
4. In the **Resource groups** blade, locate and select the **MonitoredAssets** resource group that you created earlier in this lab.
5. In the **MonitoredAssets** blade, select the **smpapi\*** API app that you created earlier in this lab.
6. In the **App Service** blade, select **Start** at the top of the blade to resume the API app.
7. In the **Stop web app** confirmation dialog box, select **Yes**.
8. Return to the currently running **Visual Studio Code** application.
9. Observe that the application finally successfully invokes the API app in Azure and returns the same JSON array that you observed earlier in this lab. Your result should resemble the following JSON content:

* ["value1","value2"]

1. Close the currently running **Visual Studio Code** application.

#### Review

In this exercise, you created a console application to access your API by using conditional retry logic. The application continued to work regardless of whether the API App was available.

### Exercise 4: Load test API app

#### Task 1: Run a performance test on an API app

1. Return to your currently open browser window displaying the **Azure portal**.
2. On the left side of the portal, select **Resource groups**.
3. In the **Resource groups** blade, locate and select the **MonitoredAssets** resource group that you created earlier in this lab.
4. In the **MonitoredAssets** blade, select the **smpapi\*** API app that you created earlier in this lab.
5. In the **App Service** blade, on the left side of the blade within the **Development Tools** category, select **Performance test**.
6. In the **Performance test** section, at the top of the section, select **New**.
7. In the **New performance test** blade, perform the following actions:
   1. In the **Name** field, enter **LoadTest**.
   2. In the **Generate Load From** list, select **East US (Web app Location)**.
   3. In the **User Load** field, enter **1000**.
   4. In the **Duration** field, enter **10**.
   5. Select **Configure Test Using**.
8. In the **Configure test using** blade, perform the following actions:
   1. In the **Test Type** list, select **Manual Test**.
   2. In the **URL** field, update the URL by appending the suffix **/api/values** to the end of the current URL.
   * **Note**: For example, if your URL is http://smpapistudent.azurewebsites.net, the new URL would be http://smpapistudent.azurewebsites.net/api/values.
   1. Select **Done**.
9. Back in the **New performance test** blade, select **Run test**.
10. Back in the **Performance test** section, in the list of **Recent runs,** select **LoadTest**.
11. In the **LoadTest** blade, wait for the test to start before you proceed with the lab.

* **Note**: Most load tests take about 10 to 15 minutes to gather the resources and start. You can wait at this blade because it will automatically refresh when the load testing is started.

1. Wait for the load test to finish before you proceed with the lab. Observe the live chart updating as your API app experiences increased usage.

* **Note**: The load test will take the 10 minutes you specified in the previous steps of the lab.

#### Task 2: Use Azure Monitor metrics after performance test

1. In the left navigation pane of the Azure portal, select **All services**.
2. In the **All services** blade, select **Monitor**.
3. In the **Monitor** blade, on the left side of the blade, select **Metrics**.
4. In the **Metrics** section, perform the following actions:
   1. In the **Resource** section, select **Select a resource**.
   2. In the **Select a resource** window that appears, in the **Resource group** list, select **MonitoredAssets**. Then, in the **Resource** list, select the **instrm\*** Application Insights account option. Finally, select **Apply** to close the window and confirm your selection.
   3. In the **Metric Namespace** list, in the **Standard** category, select **Standard metrics (preview)**.
   4. In the **Metric** list, in the **Performance Counter** category, select **Process CPU**.
   5. In the **Aggregation** list, select **Avg**.
   6. At the top of the section, select **Last 24 hours (Automatic - 5 minutes)**. In the window that appears, in the **Time range** category, select **Last 30 minutes** and then select **Apply** to save your selection.
   7. At the top of the section, select **Line chart**. In the menu that appears, select **Area chart**.
5. Observe your newly created chart.
6. At the top of the section, select **Add metric**.
7. In the **Metrics** section, perform the following actions with the new metric item in the list:
   1. In the **Metric Namespace** list, in the **Standard** category, select **Log-based metrics**.
   2. In the **Metric** list, in the **Server** category, select **Server response time**.
   3. In the **Aggregation** list, select **Avg**.
8. Observe your updated chart. Observe the information displayed in your chart. You can observe how the server response time correlates with the CPU time as load on the application increased.

#### Review

In this exercise, you performed a performance (load) test of your API app by using the tools available to you in Azure. After you performed the load test, you were able to measure your API app’s behavior by using metrics in the Azure Monitor interface.

### Exercise 5: Clean up subscription

#### Task 1: Open Cloud Shell

1. At the top of the Azure portal, select the **Cloud Shell** icon to open a new shell instance.
2. At the bottom of the portal in the **Cloud Shell** command prompt, 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 **MonitoredAssets** resource group:

* az group delete --name MonitoredAssets --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.
2. Close the currently running **Visual Studio Code** application.

#### Review

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