Azure Superpowers Lab Manual

Lab 1 - Download and Install Tools

Lab Description	**This lab covers the installation of tools that you will be using in the rest of the labs**
Estimated Time to Complete	20 minutes
Key Takeaways	1. Download and install the required tools
	2. Know what version of the tools are installed
	By the end of this lab, you should have Azure PowerShell, Git, Visual Studio Code, and Azure Storage Explorer on your workstation, and they should all be up to date.
Author	Ralph Kyttle

Purpose

For those new to DevOps, Infrastructure as Code, and Source Control, there are many tools that you need to familiarize yourself with. This lab makes sure that you have all the tools required for the labs covered in this class, as well as making sure they are up to date.

Exercise - Setup Az PowerShell Module

Launch PowerShell

1. Right Click on **Windows PowerShell** from either the Desktop or the Start Menu
2. Select **Run as Administrator**

Set PowerShell Execution Policy

1. Run the following PowerShell Command to set your execution policy to Unrestricted:

```
Set-ExecutionPolicy Unrestricted
```

2. Select Yes if prompted. Execution policies determine whether you can load configuration files, such as your PowerShell profile, or run scripts and whether scripts must be digitally signed before they are run. More information on this topic can be found here: https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.security/set-executionpolicy?view=powershell-6#parameters

Install Az Module

1. Run the following PowerShell Command:

```
Install-Module -Name 'Az'
```

2. If not already installed, you may be prompted to install the NuGet provider. Select **Yes** if this is the case.
3. You may receive a warning message stating you are installing modules from an untrusted repository. Select **Yes** if this is the case. This process may take several minutes.
4. Run the following PowerShell Command to verify the module has installed correctly:

```
Get-Module -Name '*Az*' -ListAvailable
```

5. If installed, Az should be returned
6. Run the following PowerShell Command to turn off autosaving Azure credentials
(some Azure PowerShell versions have this on as the default)

```
Disable-AzContextAutosave
```

Expected return:

```
PS C:\> Disable-AzContextAutosave

Mode : Process

ContextDirectory :

ContextFile :

CacheDirectory :

CacheFile :

Settings : {}
```

Exercise - Setup Git for Windows

Download Git for Windows Installation Files

1. In a web browser, navigate to https://git-scm.com/download/win
2. The 64-bit version of **Git for Windows** should begin automatically downloading, if it does not select **64-bit Git for Windows Setup**
3. Note the location of the installation files

Install Git for Windows

1. Run the **Git for Windows** installation file
2. Accept the EULA by clicking **Next**
3. Accept the default Destination Location and click **Next**
4. Accept the default Components and click **Next**
5. Accept the default Start Menu Folder of "Git" and click **Next**
6. Accept the default editor of Vim and click **Next**
7. Accept the default "Git from the command line and also from 3rd-party software" and click **Next**
8. Accept **OpenSSL** as the default HTTPS transport backend and click **Next**
9. Accept the default **Windows-style** "Configuring the line ending conversions" and click **Next**
10. Accept the default **MinTTY** terminal emulator and click **Next**
11. Accept the default Configuring extra options and click **Next**
12. Click **Install**
13. Uncheck "View Release Notes" and click **Finish**

Exercise - Setup Visual Studio Code

Download Visual Studio Code

1. In a web browser, navigate to https://code.visualstudio.com
2. Click **Download for Windows**
3. Note the location of the installation files

Install Visual Studio Code

1. Run the **VSCodeUserSetup** installation file
2. If you are prompted with a message "This User Installer is not meant to be run as an Administrator" Click **OK**
3. Accept the EULA and click **Next**
4. Accept the default installation location and click **Next**
5. Accept the default Start Menu Folder and click **Next**
6. Select all additional tasks and click **Next** (Optional)
7. Click **Install**
8. Uncheck "Launch Visual Studio Code" and click **Finish**

Exercise - Setup Azure Storage Explorer

Download Azure Storage Explorer

1. In a web browser, navigate to https://azure.microsoft.com/en-us/features/storage-explorer/
2. Click **Download Storage Explorer free**
3. Note the location of the installation files

Install Azure Storage Explorer

1. Run the Azure Storage Explorer installation file
2. Accept the EULA and click **Install**
3. Accept the default location and click **Next**
4. Accept the Start Menu Folder and click **Next**
5. After the installation completes, uncheck "Launch Microsoft Azure Storage Explorer" and click **Finish**

Exercise - Version Check

Check the version of Az

1. Launch VS Code
2. Select the terminal. This is typically in the lower right pane of VS Code and is often tabbed with other panes, such as Problems, Output, and Debug Console. If you do not see Terminal, click View in the top menu bar and click **Terminal**.
3. Type the following PowerShell command:

```
Get-Module -Name 'Az.Compute' -ListAvailable | Select-Object -Property 'Version'
```

4. The expected output is **2.4.1** or later

Check the version of Git for Window

1. From within the VS Code Terminal, Type the following git command:

```
git --version
```

2. The expected output is **git version 2.23.0.windows.1** or later
3. If you experience any issues with this step, restart VS Code and try again

Check the version of Visual Studio Code

1. From the VS Code File Menu, click **Help** then **About**
2. The expected version is **1.37.1** or later

Check the version of Azure Storage Explorer

1. Launch Azure Storage Explorer
2. From the File Menu, click **Help** then **About**
3. The expected version is **1.9.0** or later

Troubleshooting

Azure PowerShell

We have seen issues in previous workshops if a system has multiple versions of the Azure PowerShell module installed. If you run into module related issues, remove **all Azure modules** from C:\Program Files\WindowsPowerShell\Modules and install the Azure module again.

While it is technically possible to have both the Az and AzureRM PowerShell modules installed, we recommend removing the AzureRM modules and using only the Az modules.

The Az PowerShell module requires .Net Framework Runtime 4.7.2 or higher

Azure subscriptions

TRIAL SUBSCRIPTIONS ARE NOT SUPPORTED FOR THIS WORKSHOP

Lab 2 - Tools Walkthrough

Lab Description	**This lab will show some basics of the tools and familiarize you with their interfaces/use**
Estimated Time to Complete	30 minutes
Key Takeaways	1. Gain basic familiarity with the tools to be used in the later labs
	2. At the end of this lab you should have an understanding of the different sections of VS Code and how it interfaces with the Git protocol to commit source code.
Author	Ralph Kyttle
	Wes Adams

Downloading the tools and having them on your system is great, but we need to move around in them and learn the interfaces (such as the new tendency for programs to hide buttons until you hover over them) so that in later labs we can focus on what we're doing and have less trouble getting lost in the software.

Exercise - VS Code Interface

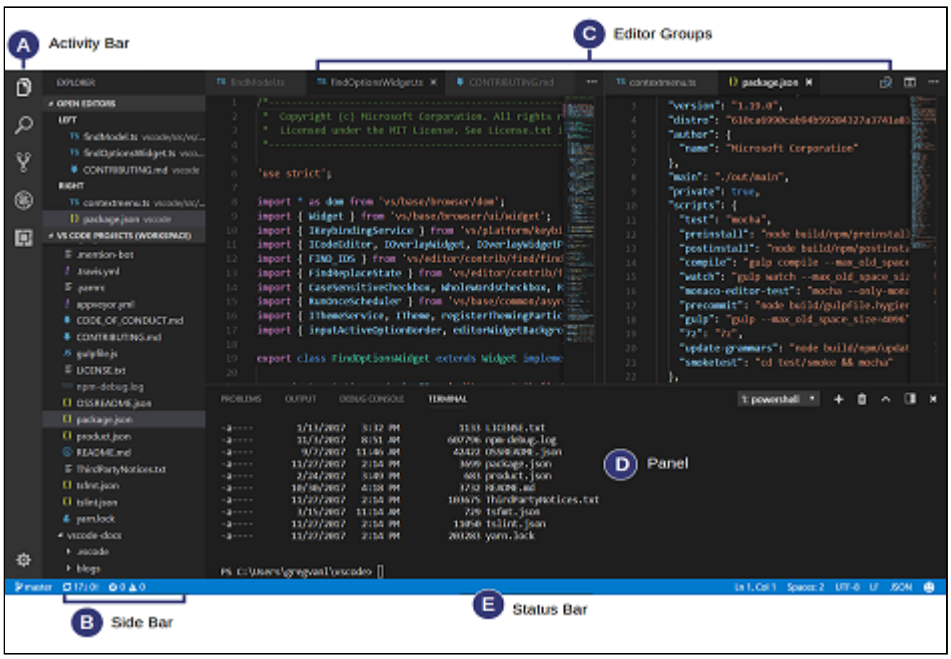
VS Code Interface Overview

1. Launch VS Code

2. Moving forward, you should be using VS Code for all code related activities, including PowerShell. You may find benefit in typing your commands in the top editor window and saving them in a file to keep a record of the commands you use in this course.

3. The interface for Visual Studio Code is divided into 5 key parts

```
a.  Activity Bar - Located on the far left-hand side, this lets you
switch between views and gives you additional context-specific
indicators like the number of unsaved files, or the number of
outgoing changes when Git is enabled.
b.  Side Bar - Contains views like the Explorer to assist you while
working on your project.
c.  Editor - The main area to edit your files. You can open as many
editors as you like side by side vertically and horizontally.
d.  Panel - You can display different panels below the editor region
for output or debug information, errors and warnings, or an
integrated terminal. Panel can also be moved to the right for
more vertical space.
e.  Status Bar - Information about the opened project and the files
you edit.
```



Make the VS Code Interface your own

1. Type **CTRL + B** to show the Side Bar. Type it again to hide it. You can always re-enable the sidebar through the View menu on the Menu Bar. You will also see similar behavior if you click on an already selected item in the Activity Bar.

2. The Activity Bar consists of 5 buttons by default.

```
a.  Explorer - View and select files for editing
b.  Search - Text level search inside your files. This will search
across all files that are available in your current workspace.
c.  Source Control - View any connected source code repositories
d.  Debug
e.  Extensions - Add extensions to enable new features within VS
Code
```

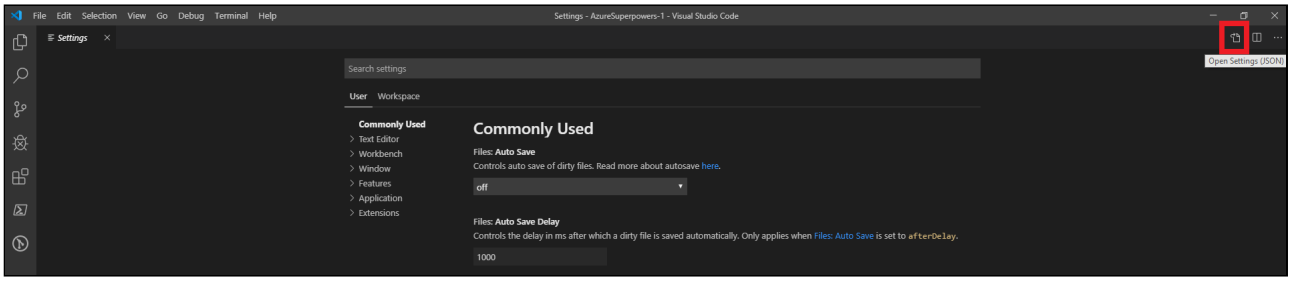
3. From the View Menu in the Menu Bar, select Appearance, and then click **Move Side Bar Right.** Notice that both the Side Bar and the Activity Bar are now on the Right Side of the Interface instead of the Left. From the View Menu, click **Move Side Bar Left** to restore the sidebar to the original location.

4. Click on the Gear Icon in the lower left-hand side of the screen and choose **Color Theme**. Notice the Command Pallet drops down and lists the available themes. Choose a theme that looks interesting to you. Choose **Dark+ (default dark)** to restore the original theme.

5. Zooming in and out within VS Code can be helpful in displaying the desired amount of information. Test this by using the commands **CTRL -** to zoom out, and **CTRL +** to zoom in. To reset zoom to the default level, go to the Menu bar and select View, Appearance, Reset Zoom.

6. To return settings back to default, Type **F1** to open the Command Pallet. Type **User** and select **Preferences: Open User Settings.**

7. Click the icon on at the top right of your window that is highlighted in the screenshot below to **Open Settings (JSON)**



8. This will open a settings.json file, which will list configured user settings that override default settings. Note that each line is comma separated. To revert changes that you have made and restore defaults, you can remove any settings that are listed.

9. An example of a user setting change the you may wish to make is setting a default file type in VS Code, which can be accomplished by adding in the following configuration:

```
"files.defaultLanguage": "powershell"
```

10. Adding the files.defaultLanguage setting listed above will ensure that any new file that you create is automatically created as a PowerShell file, as opposed to the default of plain text.

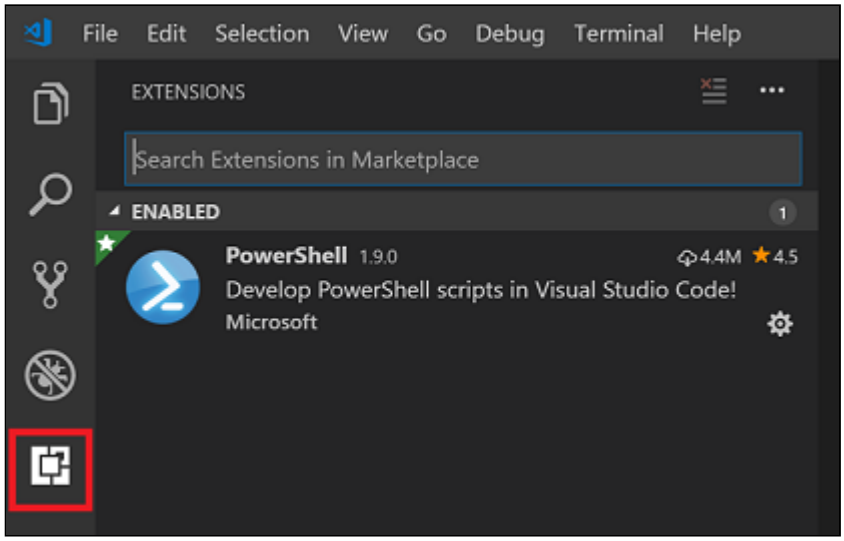
Install VS Code extensions

1. Visual Studio Code has a rich extensibility model for interacting with and adding to the tool. The features that Visual Studio Code includes out-of-the-box are just the start. VS Code extensions let you add languages, debuggers, and tools to your installation to support your development workflow.

2. You may find an extension for Azure Repos. We do not recommend using this extension in this workshop. If you already have this extension installed, please remove it for the duration of this workshop.

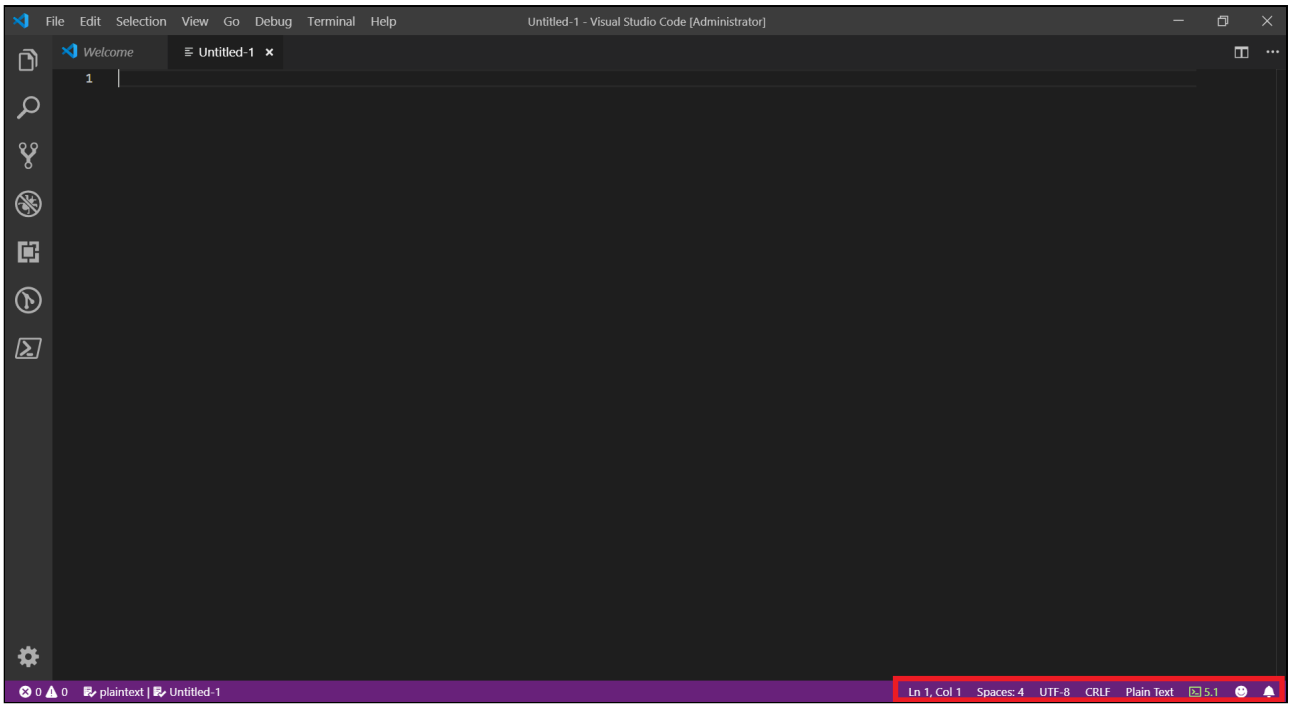
3. Search for and install the following extension, published by Microsoft. Once the extension is installed, reload VS Code.

```
a.  PowerShell (In addition to enhancements to PowerShell, this
extension includes a PowerShell ISE color theme that can found
by selecting File, Preferences, Color Theme)
b.  Install the latest available version of the PowerShell
extension, which may be newer than the version shown below.
```



4. You will not need any other VS Code extensions for this course. You may find extensions that attempt to connect you into Git repositories in Azure DevOps, but they are not needed and have been found to cause confusion with future steps in this workshop.

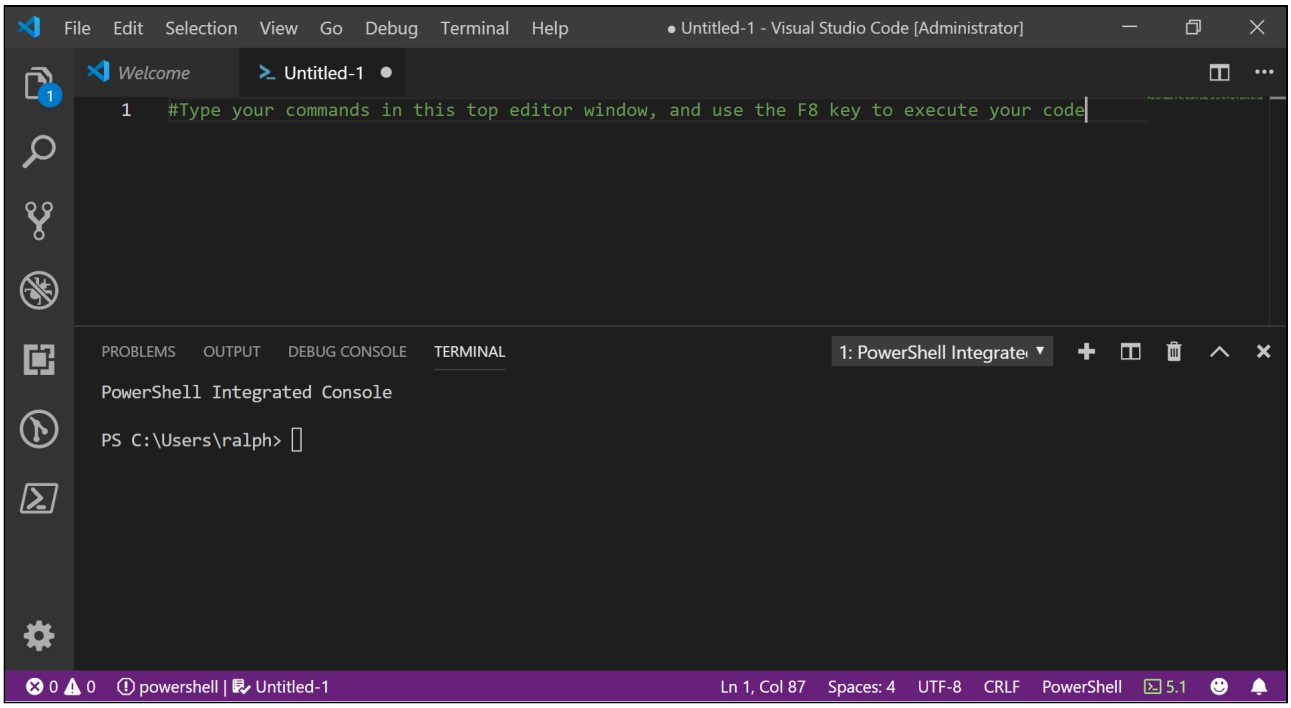
5. In VS Code, the language support for a file is based on its filename extension. However, at times you may wish to change language modes when working in a particular file. To do this, while editing a file, click on the language indicator - which is located at the bottom of the VS Code window on the right hand of the Status Bar. This will bring up the Select Language Mode drop-down where you can select another language for the current file you are working on.



6. Regarding PowerShell authoring in VS Code, you will want to have the PowerShell extension installed. You will also want to ensure that when you are working with PowerShell code, you have the VS Code language set to PowerShell, and you will want to use the PowerShell Integrated Console. **You will need have at least one PowerShell file open in VS Code for the PowerShell Integrated Console to appear.**

7. Operating in this configuration will provide you with the best PowerShell experience, and enable familiar options from the PowerShell ISE, such as pressing F8 to execute a selected area of code.

8. **NOTE:** When it comes to authoring PowerShell code in this course, we recommend that you type your commands in the top editor window, and use the F8 key to execute your code. Typing your code in the editor window allows you to save the commands you type throughout this course so you can easily refer to them later. The top editor window also enables you to be able to select and execute multiple lines of PowerShell.

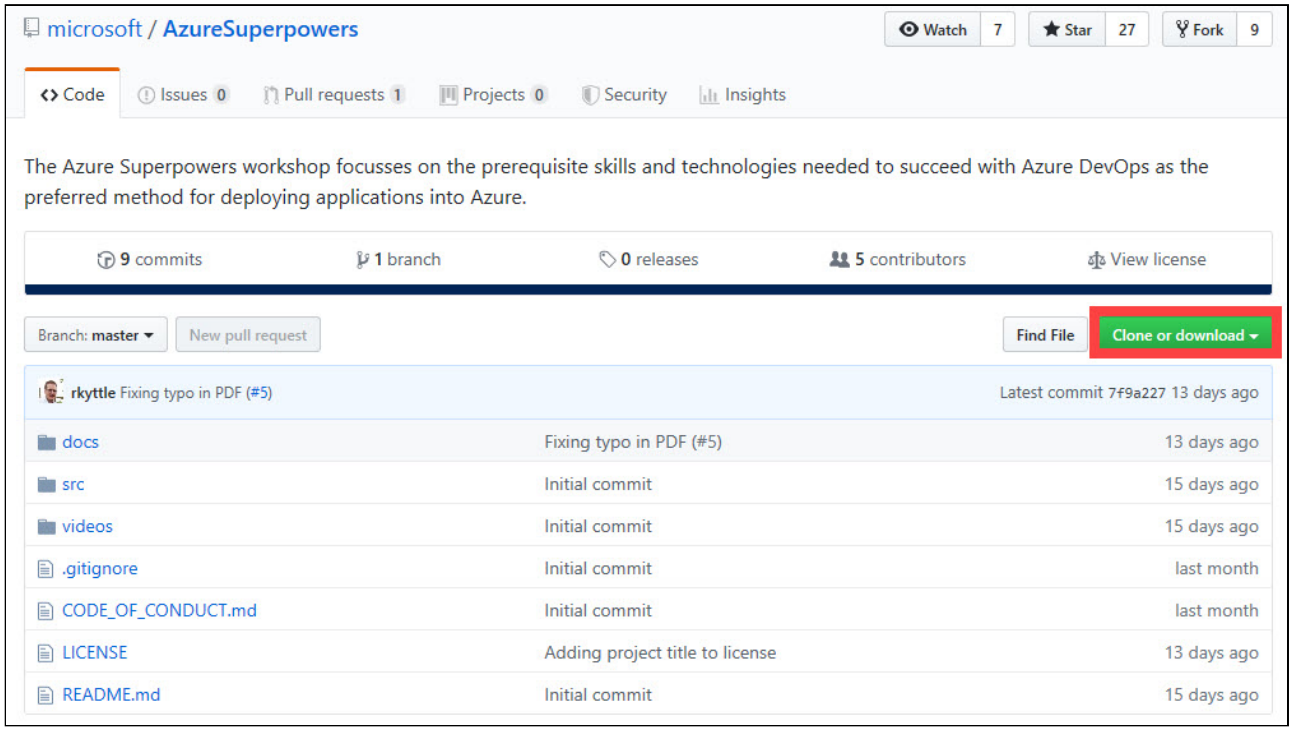

File Edit Selection View Go Debug Terminal Help
• Untitled-1 - Visual Studio Code [Administrator]
Welcome
Untitled-1
1 #Type your commands in this top editor window, and use the F8 key to execute your code
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
1: PowerShell Integrate
PowerShell Integrated Console
PS C:\Users\ralph>
0 0 powershell | Untitled-1
Ln 1, Col 87 Spaces: 4 UTF-8 CRLF PowerShell 5.1

Exercise - Clone the Azure Superpowers Repository

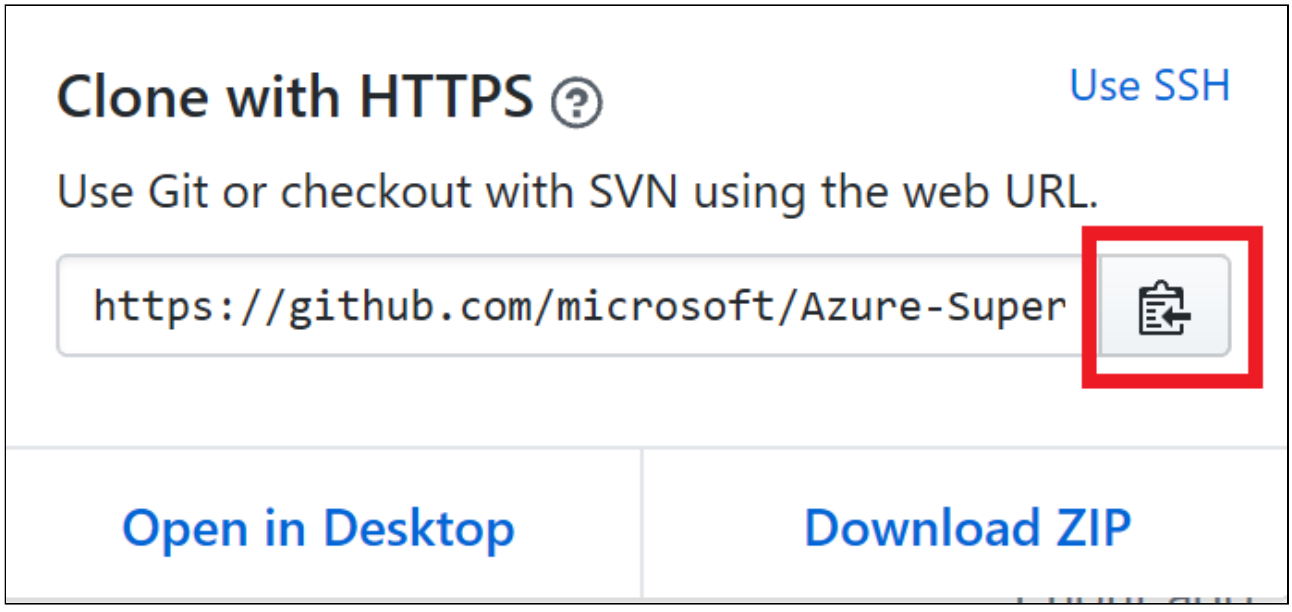
Clone Azure Superpowers via the VS Code GUI

1. Open a web browser and navigate to https://github.com/microsoft/AzureSuperpowers

2. Click on the **Clone or download** button



3. Click the icon as shown below to copy the repository URL to your clipboard so that you can paste it into VS Code in the next steps below



4. Create the following directory on your local workstation: C:\AzSuperClass
   The folder name and path listed above will be referenced by subsequent labs

5. Launch VS Code

6. Press **F1** to launch the Command Palette
7. Type the command **Git: Clone** (We will go into more detail on Git commands in a later lab)
8. The command palette will prompt for a Repository URL. Enter the URL that is currently in your clipboard (Show below for reference)

   https://github.com/microsoft/AzureSuperpowers.git

9. When prompted for a destination to clone the files, use C:\AzSuperClass

> **NOTE:** Because a repository is similar to a database, **do not** select a folder that is managed by technologies like OneDrive, Dropbox, etc. This can cause issues with both technologies fighting to track changes.

10. The cloning process will begin. Once the cloning process completes, select "Open Repository" in the lower right-hand corner. You have now cloned the Azure Superpowers Git repository from GitHub down to your local PC. This repository contains files that you will use throughout the remaining labs.

> **NOTE:** Questions have come up in the past on cloning. You only need to clone a repository once, you don't need to re-clone every time you want to work with a repository. Every time you open VS Code you will be working against a local copy of the repository, which will exist for as long as you keep the files stored on your computer.

Exercise - Az Walkthrough

What is a PowerShell module

A *module* is a set of related Windows PowerShell functionalities, grouped together as a convenient unit (usually saved in a single directory). By defining a set of related script files, assemblies, and related resources as a module, you can reference, load, persist, and share your code much easier than you would otherwise.

For more information on PowerShell Modules, refer to https://docs.microsoft.com/en-us/powershell/developer/module/understanding-a-windows-powershell-module

Az vs AzureRM

This workshop uses the latest Az PowerShell module. Starting in December 2018, the Azure PowerShell Az module is in general release and now the intended PowerShell module for interacting with Azure.

The Az module has a compatibility mode to help you use existing scripts while you update them to use the new syntax. To enable compatibility for the Az module, use the Enable-AzureRmAlias command. Aliases let you use the old cmdlet names with Az module. You can get more details on migrating from the Azure RM module to the Azure PowerShell Az module here: https://docs.microsoft.com/powershell/azure/new-azureps-module-az?view=azps-1.5.0#migrate-existing-scripts-to-az

With the new Azure PowerShell task version 4.* in build and release pipelines, we have added support for the new Az module for all platforms. Azure PowerShell task version 3.* will continue to support the AzureRM module. However, to keep up with the latest Azure services and features, we recommend that you switch to the Azure PowerShell task version 4.* as soon as possible.

Where the commands live

To see a list of the available commands in a PowerShell module, use Get-Command. Run the following command to see the available commands in the Az module:

Get-Command -Module 'Az.*'

Popular commands

Commands from the Az module that are frequently used in this workshop are shown below

```
Login-AzAccount
Get-AzContext
Get-AzEnvironment
Get-AzSubscription
Select-AzSubscription
Get-AzResourceGroup
New-AzResourceGroup
New-AzADServicePrincipal
New-AzRoleAssignment
Test-AzResourceGroupDeployment
New-AzResourceGroupDeployment
```

PowerShell Splatting

Splatting is a method of passing a collection of parameter values to a command as a unit that makes your commands shorter and easier to read. PowerShell associates each value in the collection with a command parameter. Splatted parameter values are stored in named splatting variables, which look like standard variables, but when they are called as part of a command, they begin with an At symbol (@) instead of a dollar sign ($). The @ symbol tells PowerShell that you are passing a collection of values, instead of a single value.

An example of splatting is shown below

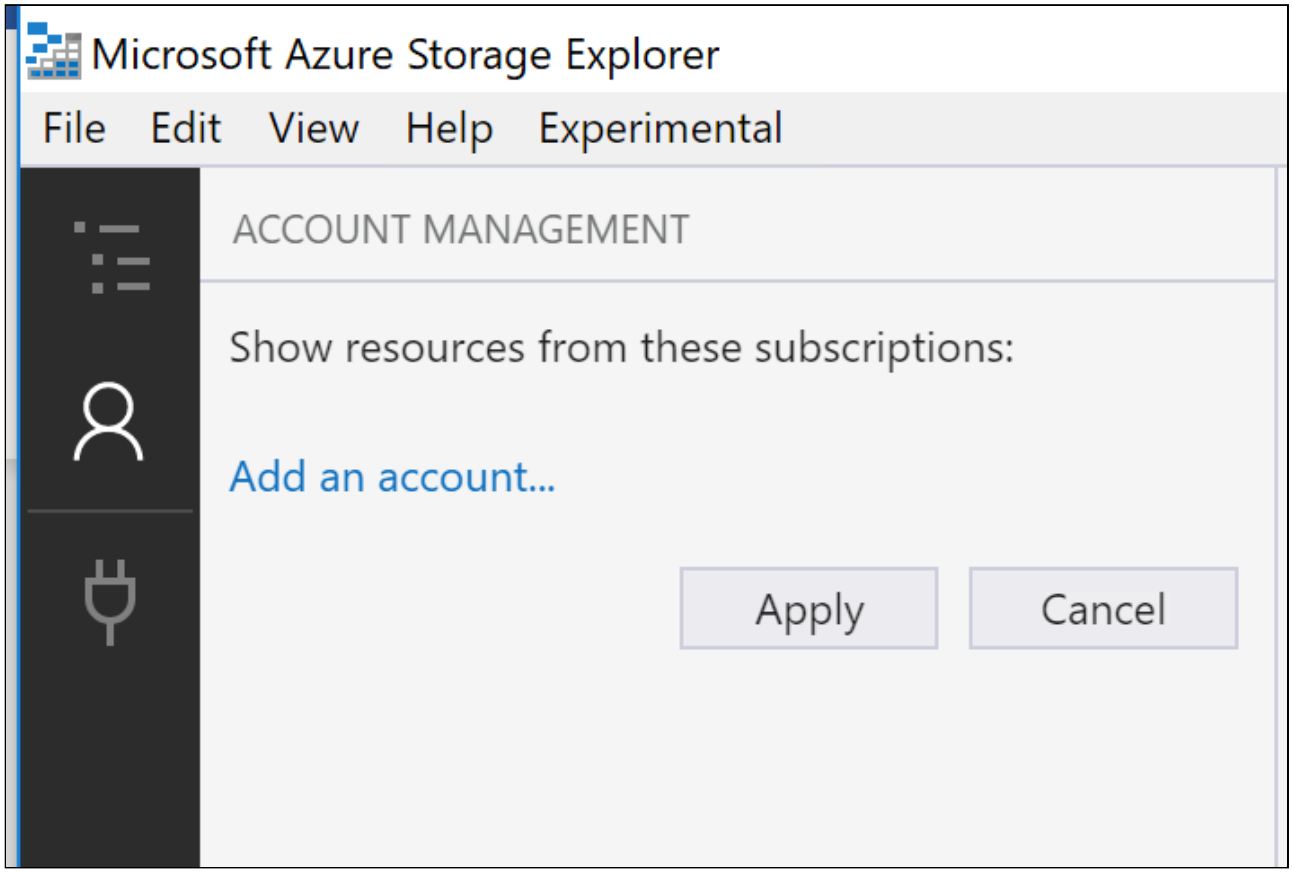
```
1.      $params = @{
2.          Name     = 'PowerShellRG'
3.          Location = 'eastUS' #or 'usgovvirginia'
4.      }
5.      New-AzResourceGroup @params
```

Note that the variable "params" is preceded by a dollar sign ($) when it's defined, but by an at sign (@) when it's called as part of a PowerShell cmdlet. Use the appropriate location for the Azure environment throughout this course.

Exercise - Azure Storage Explorer Walkthrough

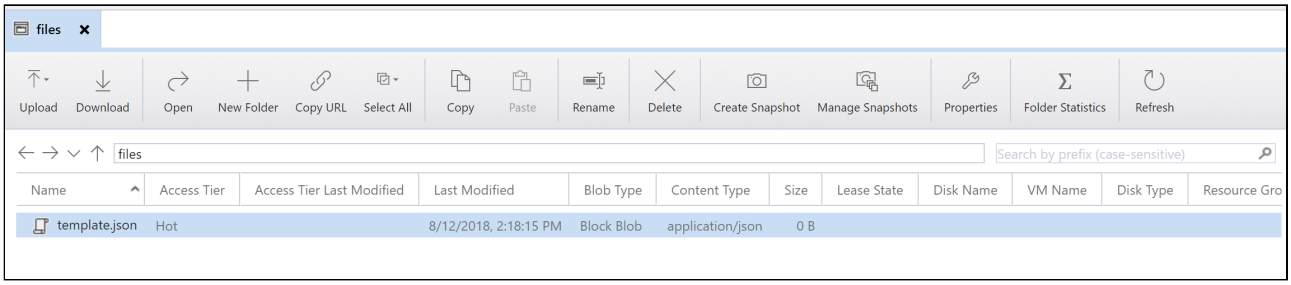
Azure Storage Explorer Walkthrough

1. Launch Azure Storage Explorer
2. From the top menu bar, click **View** and select **Account Management**
3. Click the link to Add an account



4. The "Connect to Azure Storage" wizard will launch, select **Add an Azure Account** and click **Sign in... (Pay attention to the Azure environment dropdown, and adjust if needed)**
5. Complete the authentication process with your Azure Credentials for the environment you wish to connect to
6. Under Account Management in Azure Storage Explorer, each of your subscriptions will be displayed. Select the subscriptions with the storage accounts you wish to access and click **Apply**.
7. Each subscription is displayed as a drill down list. Expand a subscription to reveal all the available storage accounts listed under the root object "Storage Account"
8. Drill into a storage account to see the types of storage available. For example, a storage account that contains Blob containers will have a list of blobs available.
9. If you select a blob in a storage account, you will see a list of all the files available in the blob. Notice the menu bar that lists available actions that can be taken with this storage account:
    - Upload

- Download
- Open
- New Folder
- Copy URL
- Select All
- Copy
- Paste
- Rename
- Delete
- Create Snapshot
- Manage Snapshots
- Properties
- Folder Statistics
- Refresh



10. Microsoft Azure Storage Explorer provides a convenient GUI for managing storage accounts. This can be useful for easily downloading and uploading files into and out of the storage account and verifying configuration. Note: You will see this menu when a blob is selected. Blobs will be created in future labs.

Troubleshooting

Simply clicking on a file in the Explorer view of VS Code will select the file, but if you click on another file it will switch to that file. To open multiple files, double click on the file to pin it in VS Code.

You will not see any storage accounts in Azure Storage Explorer if you have not already created a storage account in your Azure subscription

Ensure that you select options to login to Azure Government if your subscription is in the Microsoft Azure Government environment. We have seen issues during the Azure Storage Explorer section of this lab when the wrong environment was selected.

You can learn more about the Azure management endpoints by running the following commands

Azure Commercial: Get-AzEnvironment -Name AzureCloud -Verbose | fl * -Force
Azure Government: Get-AzEnvironment -Name AzureUSGovernment -Verbose | fl * -Force

For function keys to operate, you may need to press the **Fn** key on some computer models, like Surface

We have seen some systems experience issues with the clone button in GitHub where the URI was not copying properly to the clipboard. In this case, simply copy the URL to the clipboard by manually selecting it and copying it.

Lab 3 - Azure AD and Service Principals

Lab Description	**This lab will teach you how to create Service Principals in both the Azure Portal as well as using the Azure PowerShell cmdlets**
Glossary of Terms	*Service Principal* - an object in AzureAD that allows access to the Azure Resource Manager API layer. This means that a service (like Azure DevOps) or an application (like PowerShell's Az cmdlets) can login as a service principal and have limited rights to perform automated work in Azure
Estimated Time to Complete	60 minutes
Key Takeaways	1. Create a Service Principal and assign rights from the Azure Portal
	2. Create a Service Principal and assign rights from PowerShell
	3. Login to PowerShell as a Service Principal
	After this lab you should have a good sense of how to create a Service Principal through the Azure Portal or PowerShell, and you should understand how to give that Service Principal rights to perform actions in an Azure subscription. You should also be able to use PowerShell to perform actions against Azure as that Service Principal, however that functionality is primarily for troubleshooting. In later labs, we will enter the Service Principal information into another system, which will then use it to perform actions (instead of a human performing PowerShell actions on their workstation).
Author	Keith Hitchcock
	Ralph Kyttle
	Cory Smith

Why AAD Service Principals?

In the new cloud future, you shouldn't send a human to do a computer's job. This means that we need a non-person account that can perform actions against the Azure environment. Administrators, who are humans, should only have read access to production (for troubleshooting). Therefore, we should always use an AAD service principal in production. This may seem unusual, but in the cloud future, no one wants an administrator, who is a human, to make a change that wasn't entered into source control (therefore approved and automatically tested) by claiming that everything is 'a break-fix change'. All changes need to go through the source control process and no administrator should have the rights to circumvent that process.

AAD Service Principals

- Used for a Service (Application)
- Password or Certificate Authentication
- Supports Multiple Passwords/Certificates
- Immune to Azure AD Conditional Access
- AAD Tenant is identified by explicitly specifying it

AAD User Accounts

- Used for an individual user
- Password/Federation/Multi-factor
- Can login to the ARM Portal
- AAD Tenant is identified by @UPN Suffix

Exercise - Create Resource Group using the Portal

Login to Azure

1. Open a web browser
2. Navigate to one of the following:

   a. Commercial: https://portal.azure.com

   b. Government: https://portal.azure.us

Create a Resource Group

1. On the left-hand navigation pane, click **Resource Groups**
2. On the Resource Groups pane, click the **+Add** button
3. On the Resource Group pane, enter the resource group name, using your alias to make it unique

   PortalRG-<YOURALIAS>

   Example: PortalRG-josmith
4. Select any valid location
5. Click the **Create** button

Exercise - Create Service Principal using the Portal

Navigate to Application Registrations

1. On the left-hand side menu in Azure, select **Azure Active Directory**
2. In the Azure Active Directory pane under manage, select **App registrations**

Create a new Application Registration

1. Click the **+ New Registration** button at the top of the App registration blade
2. Enter a name for the Application: portalsp-<YOURALIAS>
   a. Example: portalsp-josmith
3. Leave Supported account types as Accounts in this organizational directory only (Default Directory)
4. Click **Register**

Generate a password for the Application Registration

1. From the menu blade, click the **Certificates & Secrets** button
2. From the Settings pane, click the **+ New client secret** button
3. From the Add a client secret pane
   a. Enter a description (Example: Key1, Password1)
   b. Select an expiration (your choice)
   c. Click the **Add** button.
   d. Value is now automatically populated. **Save this value; it is the service principal's password.**
4. Once the Key value is saved and you have a copy for your records, close the Certificates and Secrets pane.

Exercise - Grant Rights to a Resource Group using the Portal

Assign rights to Service Principal scoped to the Resource Group

1. From the Azure Portal: On the left-hand navigation pane, click **Resource Groups**
2. Click on **your new resource group name - 'PortalRG-<YOURALIAS>'**
3. From the PortalRG pane, click the **Access Control (IAM)** section
4. Click the **Role assignments** link
5. Click the **+ Add** button, then click **Add role assignment**
6. Select the **Owner Role**
7. Under **Select** type the beginning of your Service Principal name until you see it listed and then click it.
8. Click **Save** at the bottom
9. Now click the **Refresh** button and confirm that you see (under Owner) the name of your service principal

The 'Owner' role grants full rights to the scope it was granted. In this case, the Service Principal will be able to do anything in your resource group, including changing permissions.

In production it is highly recommended that a limited access role be used or a custom role created with least privileges

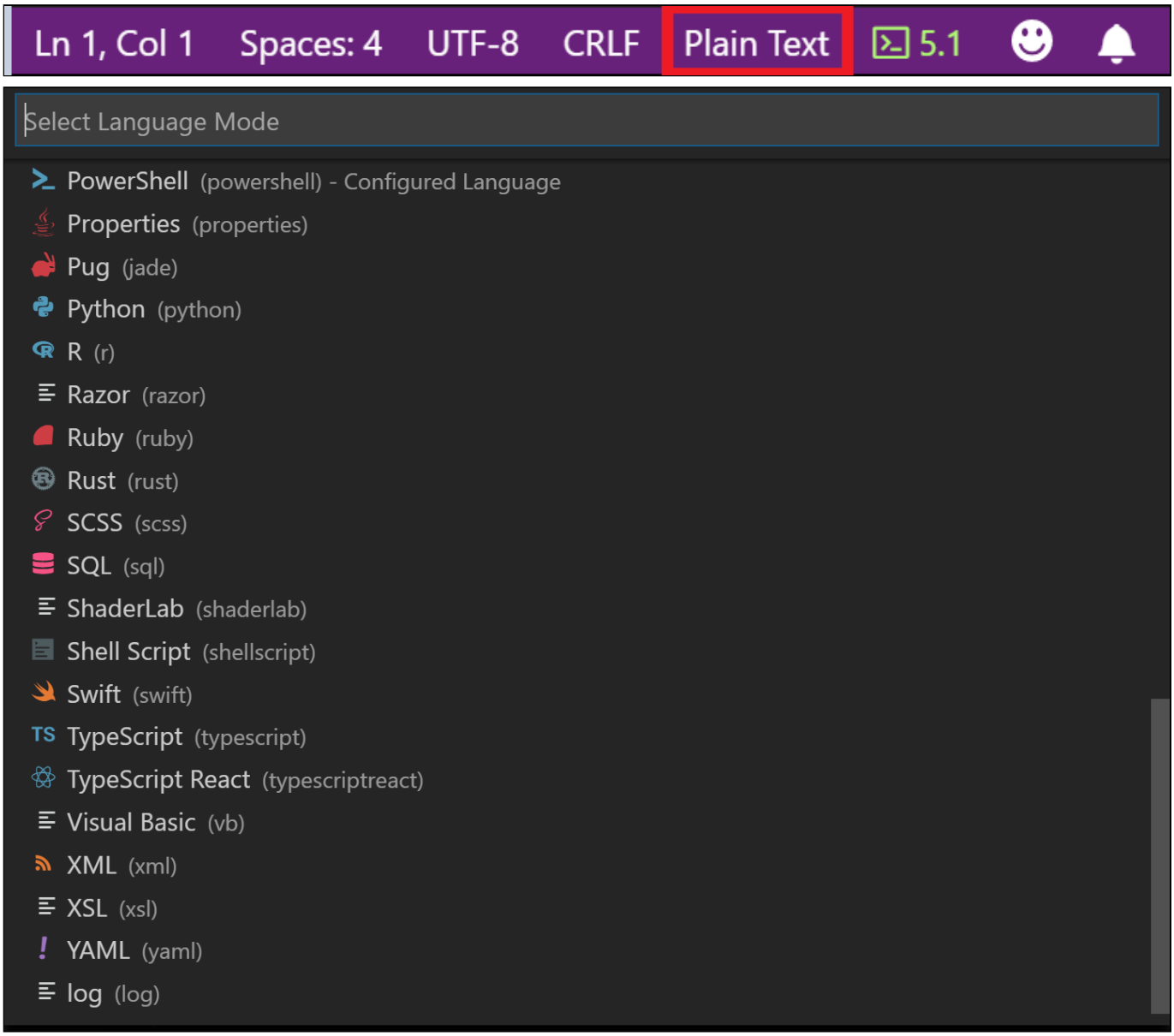
Exercise - Create Resource Group using PowerShell

Create a new PowerShell file in VS Code to store your commands

Click on **File** then **New File.**

Click on the language indicator - which is located at the bottom of the VS Code window on the right hand of the Status Bar. This will bring up the Select Language Mode drop-down where you can select another language for the current file.

If it is not already set to PowerShell, click on Plain Text and change it to PowerShell



Login to Azure via PowerShell

From a PowerShell console window, login to Azure using either:

Azure Commercial:

```
Login-AzAccount
```

-or-

Azure Government:

```
Login-AzAccount -Environment AzureUSGovernment
```

And then login into Azure (as yourself) using the pop-up dialogue box

NOTE: This will probably have popped-up under the current Window(s)

Select the desired subscription

Upon logging in, you will be signed into an Azure subscription. You may need to change this if you are associated to multiple subscriptions.

> **Pro Tip**: You may wish to create and save a .ps1 file to store all of the commands you will be typing going forward through these labs. You might need to repeat some commands and it's easier and faster to run ones you have save than to type them again. In particular, you will likely need to re-use the Select-AzSubscription command repeatedly.

Run **Get-AzSubscription** to list out the subscriptions that are available to you:

```
Get-AzSubscription
```

Run **Select-AzSubscription -Subscription '<guid>'** replacing the subscription guid with the ID from your desired subscription - note that this value will just be listed as ID, not TenantId.

The guid is alphanumeric characters in the form, xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx .

```
Select-AzSubscription –Subscription '<Id>'
```

Create a new Resource Group

Update the name listed below, and run the following PowerShell to create a new resource group:

```
$params = @{
    Name     = 'PowerShellRG-<YOURALIAS>'
    Location = 'eastUS' #or 'usgovvirginia'
}
New-AzResourceGroup @params
```

Exercise - Create Service Principal using PowerShell

Login to Azure via PowerShell

From a PowerShell console window, login to Azure using one of the following, as appropriate:

Azure Commercial:

```
Login-AzAccount
```

-or-

Azure Government:

```
Login-AzAccount -Environment AzureUSGovernment
```

And then login into Azure (as yourself) using the pop-up dialogue box (that may have popped-up under the current Window(s)).

Create a new Service Principal

From a PowerShell window logged into Azure, run the following PowerShell.
$spDisplayName should have the <YOURALIAS> replaced with your alias. (e.g. 'PowerShell-josmith')

```
1.      $spDisplayName  = 'PowerShell-<YOURALIAS>'
2.
3.      $startDate = Get-Date
4.      $KeyId = (New-Guid).Guid
5.      $KeyPassword = (New-Guid).Guid
6.
7.      $psadCredential = New-Object
Microsoft.Azure.Commands.ActiveDirectory.PSADPasswordCredential
8.      $psadCredential.StartDate = $startDate
9.      $psadCredential.EndDate = $startDate.AddYears(1)
10.     $psadCredential.KeyId = $KeyId
11.     $psadCredential.Password = $KeyPassword
12.
13.     New-AzADServicePrincipal -DisplayName $spDisplayName -PasswordCredential
$psadCredential
14.
15.     Write-Output "The password you've set is $KeyPassword (WRITE THIS DOWN)"
```

IMPORTANT: Save the service principal's password, you will need it later when logging in as the Service Principal from PowerShell.

Exercise - Grant Rights to a Resource Group using PowerShell

Login to Azure via PowerShell

From a PowerShell console window, login to Azure using one of the following, as appropriate:

Azure Commercial:

```
Login-AzAccount
```

-or-

Azure Government:

```
Login-AzAccount -Environment AzureUSGovernment
```

And then login into Azure (as yourself) using the pop-up dialogue box (that may have popped-up under the current Window(s)).

```
Select-AzSubscription –Subscription '<Id>'
```

Give rights to Service Principal scoped to the Resource Group

From a PowerShell window logged into Azure, run the following PowerShell.

```
1.      $spDisplayName = 'PowerShell-<YOURALIAS>'
2.      $appID         = (Get-AzADServicePrincipal -DisplayName
$spDisplayName).ApplicationID
3.      New-AzRoleAssignment -ApplicationID $appID -ResourceGroupName
'PowerShellRG-<YOURALIAS>' -RoleDefinitionName 'Owner'
```

Exercise - Login to Azure as Service Principal

Login to Azure via PowerShell

From a PowerShell console window, login to Azure using one of the following, as appropriate:

Azure Commercial:

```
Login-AzAccount
```

-or-

Azure Government:

```
Login-AzAccount -Environment AzureUSGovernment
```

And then login into Azure (as yourself) using the pop-up dialogue box (that may have popped-up under the current Window(s)).

Gather Information about the AzureAD Tenant and the Service Principal

From a PowerShell window logged into Azure, run the following PowerShell. Record the TenantID of Azure Active Directory and the ApplicationID of the ServicePrincipal for the next step.

Execute the following lines of code one line at a time

```
1.      $spDisplayName = 'PowerShell-<YOURALIAS>'
2.      Get-AzSubscription | Select-Object TenantID
3.      Get-AzADServicePrincipal -DisplayName $spDisplayName | Select-Object
ApplicationID
```

IMPORTANT: Record the TenantID and the ApplicationID as these will be used in future steps.
If you receive multiple different TenantID's, you will need to include a subscription name when running the Get-AzSubscription cmdlet

```
Get-AzSubscription -SubscriptionName 'My Subscription' | Select-Object TenantID
```

Log out of Azure PowerShell

From a PowerShell window logged into Azure, run the following PowerShell:

```
Logout-AzAccount
```

Define Variables

From a PowerShell window not logged into Azure, run the following PowerShell.
When prompted (line 3), enter the password for the Service Principal you created using PowerShell

Note: The password needed in this exercise should be listed in your terminal as a warning output
"The password is: $password (WRITE THIS DOWN)"

```
1.      $tenantID      = '<AAD TENANT ID>'
2.      $applicationID = '<SERVICE PRINCIPAL APPID>'
3.      $psCredObj     = Get-Credential -UserName $applicationID -Message 'SP
Password'
```

Login to Azure as Service Principal

From a PowerShell console window not already logged into Azure, run one of the following PowerShell commands, as appropriate:

Azure Commercial:

```
Login-AzAccount -ServicePrincipal -TenantId $tenantID -Credential $psCredObj
```

Azure Government:

```
Login-AzAccount -ServicePrincipal -TenantId $tenantID -Credential $psCredObj -
Environment AzureUSGovernment
```

Verify Ability to See Limited Resources

From the PowerShell window logged in as the Service Principal, run the following PowerShell:

```
Get-AzResourceGroup
```

You should only see the Resource Groups that you assigned a role to in the previous exercise. If you cannot see anything, start this exercise again and make sure you logged into the correct environment (MAG or Commercial)

Log out of Azure PowerShell

From a PowerShell window logged into Azure, run the following PowerShell:

```
Logout-AzAccount
```

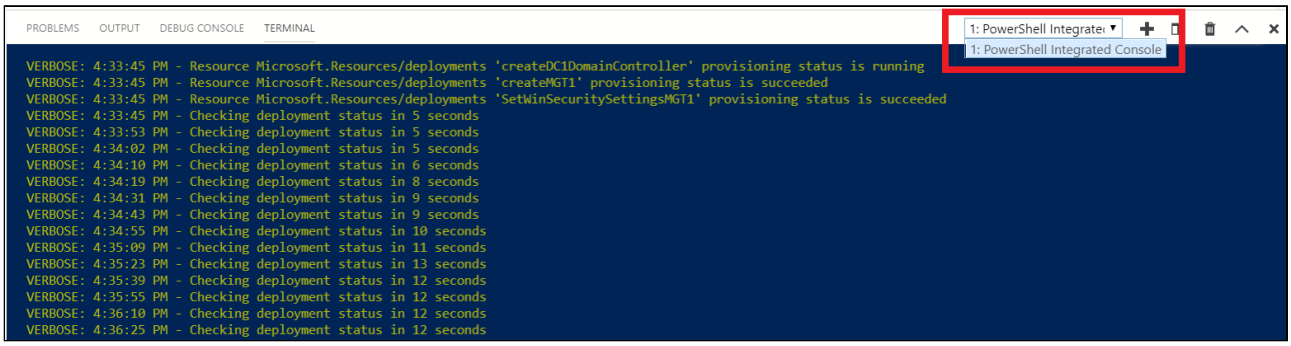
Troubleshooting

The login prompt to login to Azure via PowerShell frequently appears as a "pop-under" and may hide behind other open windows on your desktop

Each Exercise in this workshop is designed as its own discrete unit. Many exercises may begin with having you login to Azure. You can move past the login instructions if you are already logged in.

When working with PowerShell in VS Code, the following items are important

- The VS Code extension for PowerShell should be installed
- You should be executing PowerShell commands using the PowerShell Integrated Console in VS Code



- F8 should be used to execute PowerShell commands. You can execute any single line of code just by ensuring your cursor is on that line, without any text selected. To select multiple lines for execution, highlight those sections of code, then press F8.
- **IMPORTANT**: You may find it easier to type your commands in the top editor window, saving the file as a .ps1, and then select them when you would like to execute. This also allows you to capture the commands you type during this class and save them for later reference.

If you receive multiple different TenantID's in this lab, you may need to include a subscription name when using the Get-AzSubscription cmdlet

For function keys to operate, you may need to press the **Fn** key on some computer models, like Surface

Pay attention to the line numbers in the code. Printing the labs can make it hard to fit an entire line of code on one line, so line numbers are listed to indicate when a section of code needs to be listed on a single line.

Lab 4 - Basic ARM Templates

Lab Description	**In this lab, you will explore the use of Azure Resource Manager (ARM) templates to deploy resources into a Resource Group. ARM templates are JSON files that define the resources you need to deploy for your solution. To understand the concepts associated with deploying and managing your Azure solutions, see Azure Resource Manager overview**
Glossary of Terms	*ARM Template* - An ARM Template is a JavaScript Object Notation (JSON) file that defines one or more resources to deploy to an Azure resource group. It also defines the dependencies between the deployed resources. The template can be used to deploy the resources consistently and repeatedly
Estimated Time to Complete	30 minutes
Key Takeaways	1. Create your first Azure Resource Manager template
	2. Deploy a Storage Account using an ARM template with Azure PowerShell and the New-AzResourceGroupDeployment cmdlet
	3. Edit an existing ARM template by adding parameters and variables to allow for flexible, dynamic deployments
	4. Understand the impact of the declarative nature of ARM templates and how they interact with the Azure API. Once the ARM template is provided to the API, the transaction is complete as far as the user is concerned. The user could lose connectivity, or their PC could lose power and everything would still continue, as the instructions have been delivered and Azure takes it from there. This differs greatly from traditional scripting, which might require lines of code be executed independently or require constant connectivity.
Author	Ralph Kyttle

In this lab you will use an Azure Resource Manager (ARM) template, which is a JSON (JavaScript Object Notation) file to define an Azure Storage Account (which is an Azure PaaS service). Defining your infrastructure as a JSON file (which is a data format) is a great introduction to end-state deployment. The JSON describes the end state of what you want (in this case a storage account), it does not describe *how* to get there, and it does not *expect* a particular state (such as starting with an empty resource group) in order to be successful.

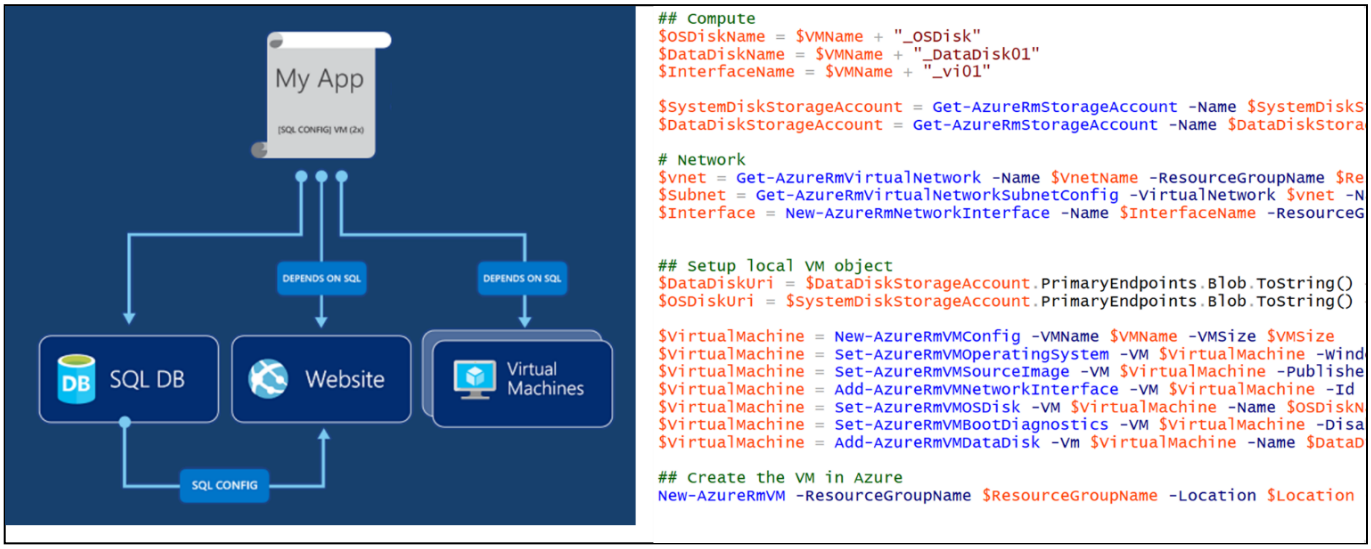
At the completion of this lab you should be familiar with the single PowerShell cmdlet that allows you to deploy an ARM template to a resource group, you should be familiar with the relationships between ARM templates and resource groups, and you should be familiar with how to manipulate an ARM template to add/remove/rename parameters/variables/resources to your liking. Understanding how to modify ARM templates allows us to make judgement calls about how complicated/simple they are, and how flexible/inflexible we want/need them to be.

ARM Template Format

JSON Copy

```
{
    "$schema": "http://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "",
    "parameters": {  },
    "variables": {  },
    "functions": {  },
    "resources": [  ],
    "outputs": {  }
}
```

ARM Templates vs Imperative PowerShell



Where can I see examples of ARM Templates?

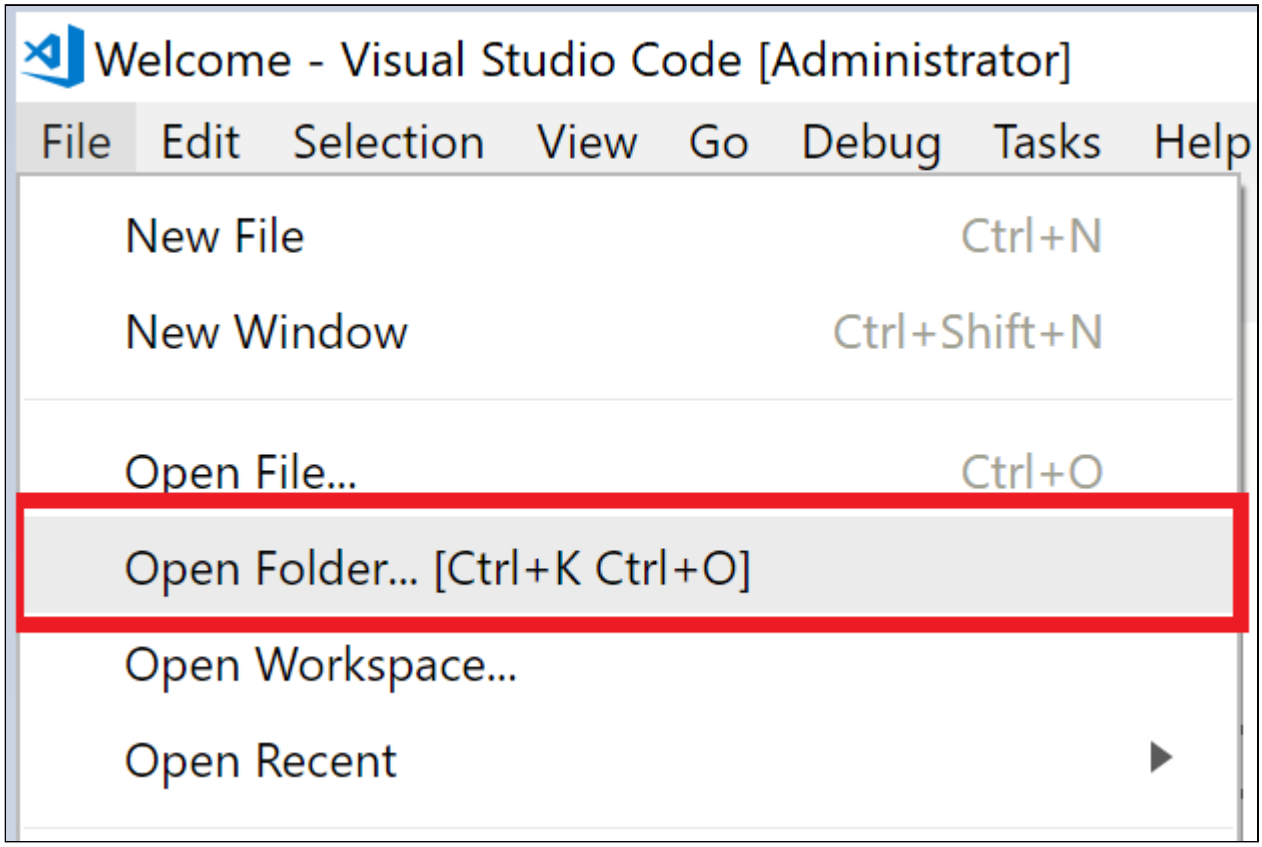
https://github.com/Azure/azure-quickstart-templates

Exercise - Deploy a Storage Account using an ARM template and Azure PowerShell

Open and review the ARM template in the Azure Superpowers repository

Launch Visual Studio Code
(Visual Studio Code is strongly recommended for creating and editing Resource Manager templates)

1. Select **File** > **Open Folder...**



2. Open the following folder: C:\AzSuperClass\AzureSuperpowers

3. Expand src\Lab - ARM Templates, select the **StorageAccount.json** ARM template file and review its contents.

   NOTE: Storage account names have several restrictions that can make them difficult to set. The name must be between 3 and 24 characters in length, use only numbers and lowercase letters, and be unique. The preceding template uses the uniqueString function to generate a hash value. To give this hash value more meaning, it adds the prefix *storage*.

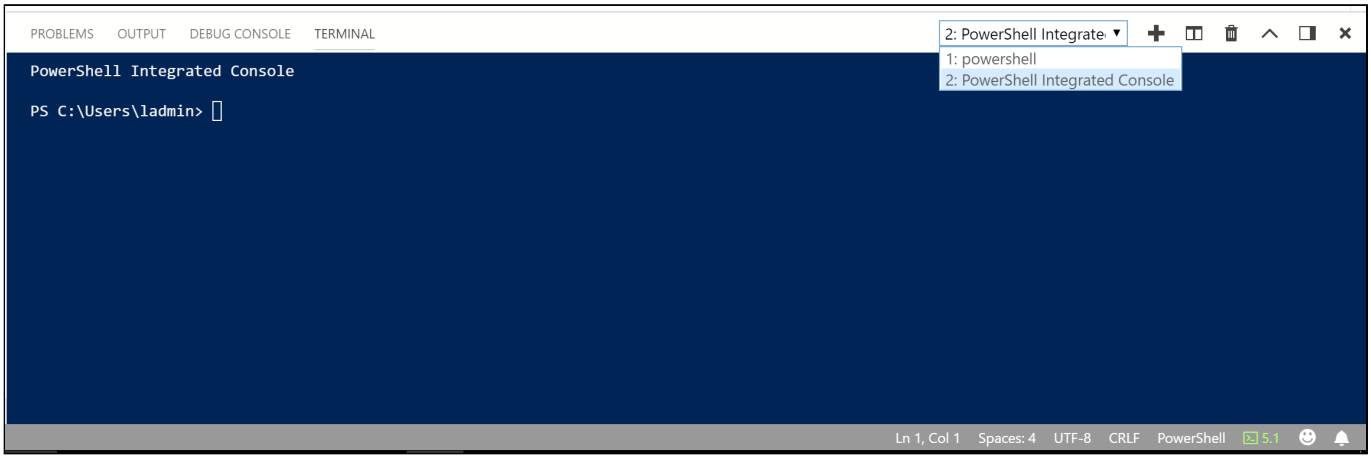
Deploy ARM template

You are now ready to deploy this template. Next, you will deploy a storage account to the PowerShellRG that you created as part of the Azure AD and Service Principals lab.
The storage account deployment should take just about 1 minute to complete.

In the PowerShell terminal of VS Code, Set your working directory to
C:\AzSuperClass\AzureSuperpowers\src\Lab - ARM Templates

Ensure that you have selected the PowerShell Integrated Console

IMPORTANT: You must have a .ps1 file open inside of VS Code in order to utilize the PowerShell Integrated Console



Login to Azure via PowerShell

From a PowerShell console window, login to Azure using either:

Azure Commercial:

```
Login-AzAccount
```

-or-

Azure Government:

```
Login-AzAccount -Environment AzureUSGovernment
```

And then login into Azure (as yourself) using the pop-up dialogue box (that may have popped-up under the current Window(s)).

```
Select-AzSubscription –Subscription '<Id>'
```

The PowerShellRG resource group should already exist from the earlier Azure AD and Service Principal's Lab. If it does not, create the Resource Group prior to this step. Execute the following PowerShell:

Run lines 1-6 first. If your template comes back as valid, run line 8.

```
1.      $params = @{
2.          ResourceGroupName = 'PowerShellRG-<YOURALIAS>'
3.          TemplateFile      = 'StorageAccount.json'
4.          Verbose           = $true
5.      }
6.      Test-AzResourceGroupDeployment @params
7.
8.      New-AzResourceGroupDeployment @params
```

The following screenshot shows the expected output

```
PS C:\AzSuperClass\Azure Superpowers\Lab - ARM Templates> $params = @{
    ResourceGroupName = 'PowerShellRG'
    TemplateFile      = 'StorageAccount.json'
    Verbose           = $true
}
Test-AzureRmResourceGroupDeployment @params
VERBOSE: 2:53:48 AM - Template is valid.
PS C:\AzSuperClass\Azure Superpowers\Lab - ARM Templates> New-AzureRmResourceGroupDeployment @params
VERBOSE: Performing the operation "Creating Deployment" on target "PowerShellRG".
VERBOSE: 2:54:22 AM - Template is valid.
VERBOSE: 2:54:23 AM - Create template deployment 'StorageAccount'
VERBOSE: 2:54:23 AM - Checking deployment status in 5 seconds
VERBOSE: 2:54:28 AM - Checking deployment status in 5 seconds
VERBOSE: 2:54:34 AM - Checking deployment status in 5 seconds
VERBOSE: 2:54:39 AM - Checking deployment status in 5 seconds
VERBOSE: 2:54:44 AM - Checking deployment status in 5 seconds
VERBOSE: 2:54:49 AM - Checking deployment status in 5 seconds
VERBOSE: 2:54:54 AM - Checking deployment status in 5 seconds
VERBOSE: 2:54:59 AM - Checking deployment status in 5 seconds
VERBOSE: 2:55:04 AM - Checking deployment status in 5 seconds
VERBOSE: 2:55:09 AM - Checking deployment status in 5 seconds
VERBOSE: 2:55:14 AM - Checking deployment status in 5 seconds
VERBOSE: 2:55:19 AM - Checking deployment status in 5 seconds
VERBOSE: 2:55:24 AM - Checking deployment status in 5 seconds
VERBOSE: 2:55:30 AM - Checking deployment status in 5 seconds
VERBOSE: 2:55:35 AM - Checking deployment status in 5 seconds
VERBOSE: 2:55:40 AM - Resource Microsoft.Storage/storageAccounts 'storagesov6fjpthrrhe' provisioning status is running
VERBOSE: 2:55:40 AM - Checking deployment status in 14 seconds
VERBOSE: 2:55:54 AM - Checking deployment status in 5 seconds
VERBOSE: 2:55:59 AM - Resource Microsoft.Storage/storageAccounts 'storagesov6fjpthrrhe' provisioning status is succeeded


DeploymentName          : StorageAccount
ResourceGroupName       : PowerShellRG
ProvisioningState       : Succeeded
Timestamp               : 11/4/2018 2:55:58 AM
Mode                    : Incremental
TemplateLink            :
Parameters              :
Outputs                 :
DeploymentDebugLogLevel :
```

Upon completion of the deployment, you can validate that your storage account resource now exists by viewing it in the Azure portal, or by running the following PowerShell command:

```
Get-AzResource -ResourceGroupName 'PowerShellRG-<YOURALIAS>'
```

The screenshot above showcases a PowerShell based deployment with verbose mode enabled, which enables a text-based output regarding the status of the deployment. You can also review the Deployments tab in the Azure portal for the resource group that you are targeting to see real-time deployment information, as well as any errors or warnings if your deployment is not successful.

Exercise - Introduce template parameters and variables

Introduction

The template from the previous exercise works fine, but it is not flexible. It always deploys a locally redundant storage account. The name is always storage followed by a hash value. To enable using the template for different scenarios, we will add parameters to the template.

The next example shows the parameters section with two parameters. The first parameter storageSKU enables you to specify the type of redundancy. It limits the values that you can pass in to the values that are valid for a storage account. It specifies a default value. The second parameter storageNamePrefix is set to allow a maximum of 11 characters. It also specifies a default value.

The list of allowedValues is worth a close look. These values do not match what you see when creating a storage account in the Azure Portal, so how did we come up with these values? Values in the Azure Portal will look more like the following: **Locally-redundant storage (LRS)**

The following articles can help us find the correct values needed for this ARM template

https://docs.microsoft.com/en-us/azure/templates/microsoft.storage/storageaccounts

https://github.com/Azure/azure-resource-manager-schemas/tree/master/schemas

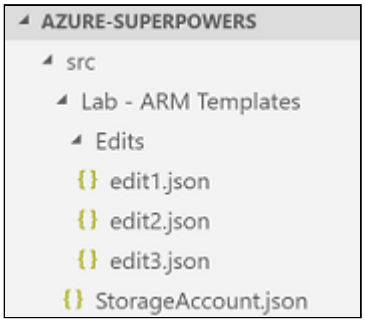
https://github.com/Azure/azure-resource-manager-schemas/blob/master/schemas/2016-01-01/Microsoft.Storage.json

You can also find values via the Azure Portal by attempting a deployment, which will generate an ARM template as part of the process, which can be reviewed to determine the proper values. The docs and GitHub may not always give you the values needed, so the Azure Portal can be useful.

Customize the ARM template

Next, you will make changes to the **StorageAccount.json** file to make it more dynamic. Edits to this file will make updates to the Parameters, Variables and Resources sections of the ARM template.

The needed edits can be found in the Azure Superpowers Git repository inside the edit files shown below.



Update your parameters section of **StorageAccount.json** to match below, using the edit1.json for the input

```
…
"parameters": {
    "storageSKU": {
        "type": "string",
        "allowedValues": [
            "Standard_LRS",
            "Standard_ZRS",
            "Standard_GRS",
            "Standard_RAGRS",
            "Premium_LRS"
        ],
        "defaultValue": "Standard_LRS",
        "metadata": {
            "description": "The type of replication to use for the storage
account."
        }
    },
    "storageNamePrefix": {
        "type": "string",
        "maxLength": 11,
        "defaultValue": "error",
        "metadata": {
            "description": "The value to use for starting the storage account
name. Use only lowercase letters and numbers.  For this example, the word error
was chosen in case you do not follow the lab instructions to supply a new storage
account prefix."
        }
    }
},
…
```

In the variables section, add a variable named storageName. It combines the prefix value from the parameters and a hash value from the uniqueString function. It uses the toLower function to convert all characters to lowercase. Storage account names must be between 3 and 24 characters in length, use only numbers and lowercase letters, and be unique. This data can be found in edit2.json

```
…
"variables": {
    "storageName": "[concat(toLower(parameters('storageNamePrefix')),
uniqueString(resourceGroup().id))]"
},
…
```

To use these new values for your storage account, change the resource definition, using the data found in edit3.json:

```
…
"resources": [
    {
        "name": "[variables('storageName')]",
        "type": "Microsoft.Storage/storageAccounts",
        "apiVersion": "2016-01-01",
        "sku": {
            "name": "[parameters('storageSKU')]"
        },
        "kind": "Storage",
        "location": "[resourceGroup().location]",
        "tags": {},
        "properties": {}
    }
],
…
```

Notice that the name of the storage account is now set to the variable that you added. The SKU name is set to the value of the parameter. The location is set the same location as the resource group.

Save your file.

Your template now looks like:

```
{
    "$schema": "http://schema.management.azure.com/schemas/2015-01-
01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "storageSKU": {
            "type": "string",
            "allowedValues": [
                "Standard_LRS",
```

```
                "Standard_ZRS",
                "Standard_GRS",
                "Standard_RAGRS",
                "Premium_LRS"
            ],
            "defaultValue": "Standard_LRS",
            "metadata": {
                "description": "The type of replication to use for the storage
account."
            }
        },
        "storageNamePrefix": {
            "type": "string",
            "maxLength": 11,
            "defaultValue": "error",
            "metadata": {
                "description": "The value to use for starting the storage account
name. Use only lowercase letters and numbers.  For this example, the word error
was chosen in case you do not follow the lab instructions to supply a new storage
account prefix."
            }
        }
    },
    "variables": {
        "storageName": "[concat(toLower(parameters('storageNamePrefix')),
uniqueString(resourceGroup().id))]"
    },
    "resources": [
        {
            "name": "[variables('storageName')]",
            "type": "Microsoft.Storage/storageAccounts",
            "apiVersion": "2016-01-01",
            "sku": {
                "name": "[parameters('storageSKU')]"
            },
            "kind": "Storage",
            "location": "[resourceGroup().location]",
            "tags": {},
            "properties": {}
        }
    ],
    "outputs": {}
}
```

Redeploy the ARM template

Redeploy the template **with new values** specified as runtime parameters as shown below.

In a PowerShell command window, ensure your working directory is set to the folder containing your template and run the following commands:

```
$params = @{
    ResourceGroupName = 'PowerShellRG-<YOURALIAS>'
    TemplateFile      = 'storageaccount.json'
    storageNamePrefix = 'newstore'
    storageSKU        = 'Standard_GRS'
    Verbose           = $true
}
Test-AzResourceGroupDeployment @params

New-AzResourceGroupDeployment @params
```

Through this task, you have now deployed a new storage account using dynamic values for the storage account name and storage SKU.

Upon completion of the deployment, you can validate that your storage account resource now exists by viewing it in the Azure portal, or by running the following PowerShell command:

```
Get-AzResource -ResourceGroupName 'PowerShellRG-<YOURALIAS>'
```

Cleanup

> When no longer needed, clean up the resources you deployed by deleting the entire resource group.

In a PowerShell command window, use the following command:

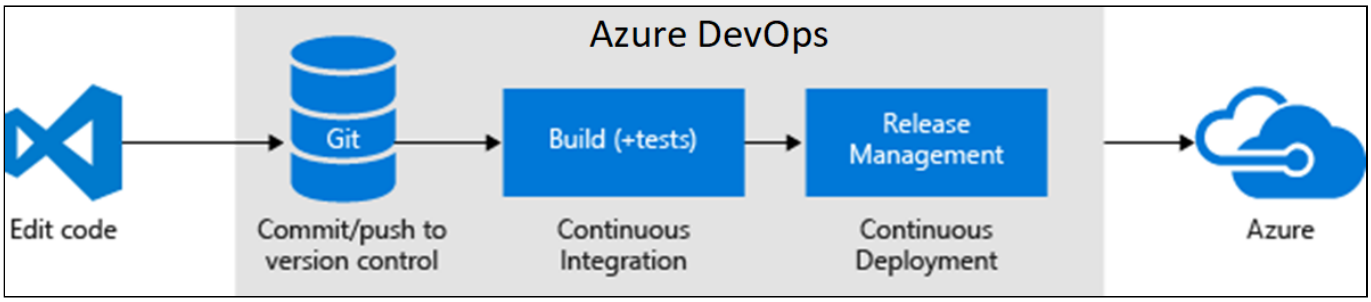
```
Remove-AzResourceGroup -Name 'PowerShellRG-<YOURALIAS>'
```

Troubleshooting

Lab 5 - Azure DevOps

Lab Description	**In this lab you will create a new Azure DevOps organization, a new project, and a new code repository. You will initialize the code repository and clone it to your local computer, and you will add other users to your project so they can collaborate.**
Glossary of Terms	*organization* - The highest object in the Azure DevOps hierarchy, which has a DNS name https://dev.azure.com/<organizationname> and can have many projects under it
	project - a project is for managing collaboration. Users can be invited to the project, which may contain multiple code repositories
	repository - A git repository is all code files, all branches, all metadata, and commit history of changes
	clone - the 'git clone' is a copy of an entire code repository from an 'origin' (such as Azure DevOps) to a 'remote' location (such as your local workstation). Additionally, a clone knows where the origin is, so it can also 'sync' with the origin code repository.
Estimated Time to Complete	30 minutes
Key Takeaways	1. The hierarchy of Azure DevOps
	2. Familiarity with Azure DevOps projects
	3. Able to interact with the git repositories that live inside Azure DevOps projects
Author	Keith Hitchcock

Process Flow



Exercise - Create new Azure DevOps organization

Login to visualstudio.com

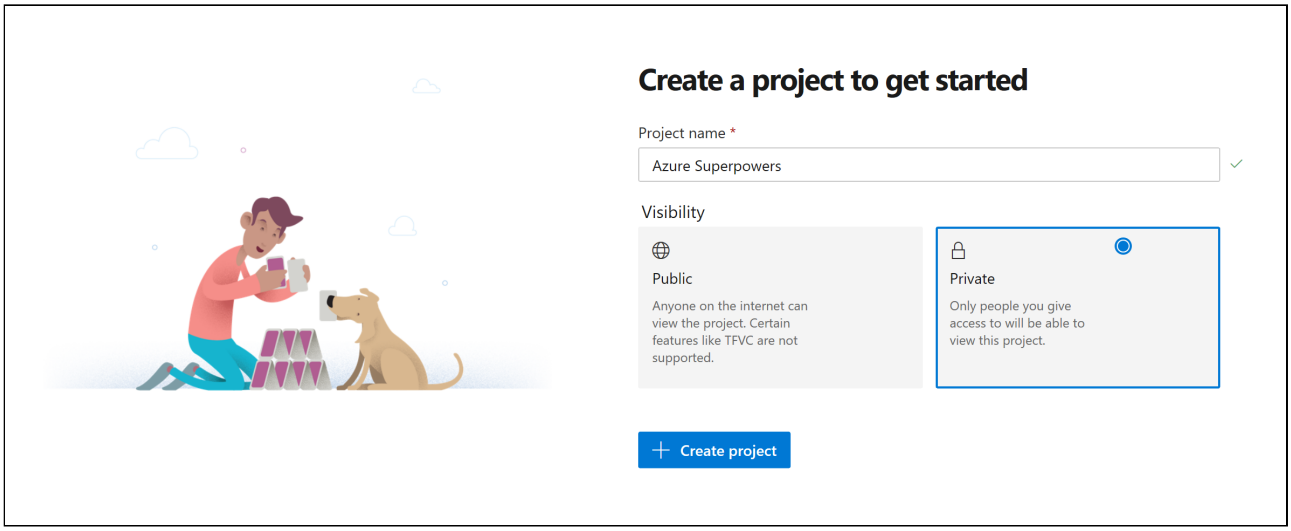
1. Open a web browser and navigate to https://www.visualstudio.com.
2. In the upper-right hand corner, you might be automatically logged in with your Microsoft Account. If not, please login.
3. Once logged in, click the logged in user in the upper-right hand corner and select 'Visual Studio Profile' from the expanded menu.

Create a new organization: https://dev.azure.com/<organizationname>

1. From the *Azure DevOps Organizations* page, you can see all the organizations that you are either a *Member* of, or *Owner* of. In the upper right-hand corner of the page, click the 'Create New Organization' button and create a new organization with a new https://dev.azure.com/<organizationname>. Because the organization name is a DNS name it must be globally unique. The specific organization name that you choose is largely irrelevant.
2. Click the 'Continue' button.

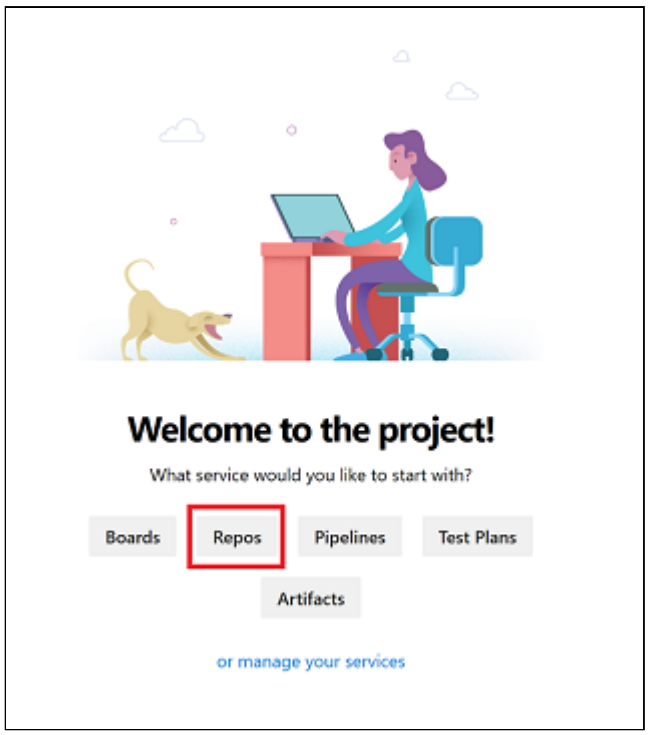
Create your first project

1. Create a project to initialize your organization



2. Project names must be unique within an organization. Name your project **Azure Superpowers**. Select Private visibility, then click **Create project**

3. Select **Repos** on the 'Welcome to the project!' screen



Initialize the default git repository

1. You should now see the initial repo page. At the bottom, you should see a section to 'initialize with a README or gitignore'. Click the 'initialize' button. If you do not see the button, you may need to expand the section out by clicking on the name 'initialize with a README or gitignore'. The reason to initialize the repository from here is because an empty repository cannot be cloned.

Clone git repository from Azure DevOps to your local system

1. From the 'Files' section of the Project, in the upper right-hand corner there is a 'Clone' button which will allow you to copy the URL of the git repository (hosted on Azure DevOps) for this project. Click the copy button to copy the URL to your clipboard.
2. Launch the Visual Studio Code application your local system
3. View -> Command Palette
4. In the command palette, type 'git clone' and press enter
5. When prompted for 'Repository URL' paste the value obtained from Step 1 into the command palette
6. Select a folder to store the new repository locally **(Ex. C:\MyAzureProject)**. Note that you are selecting the top level folder. A subfolder will be created with the repository name automatically. Because a repository is similar to a database, **do not** select a folder that is managed by technologies like OneDrive, Dropbox, etc. This can cause issues with both technologies fighting to track changes.
7. You may be prompted to authenticate, please enter credentials that are valid for your Azure DevOps organization

8. When Visual Studio Code prompts you to 'Open Repository', select that option. You should now see the repository and the default README.md file that you created by the step where you initialized the repository in Azure DevOps.

Exercise - Explore Azure DevOps Local Repository

Open local git repository with VS Code

1. If VS Code is not launched, launch it
2. If the local git repository is not in the left-hand tree view (files), open it - File -> Open Folder and open the local folder that matches the repository name

Open local git repository with File Explorer

1. In VS Code, with the local git repository open, on the left-hand file menu, right-click on a top level file and select 'Reveal in Explorer'
2. Now that you have the local git repository open in file explorer, you can move (or delete) files to it the way you would normally (e.g. Copied from local drive, USB, Right-click->New->File). The files that you move into file explorer will be added to the current branch of the git repository. The current branch can be viewed in the bottom left-hand corner of VS Code. It will likely say 'master' right now, but whichever branch it indicates is where any file modifications are taking place (regardless of if you use VS Code, file explorer, notepad, etc to add or modify fles)
3. Verify that you can see the ReadMe file within the File Explorer view.

Exercise - Add Contributors to your Project

Login to visualstudio.com

1. Open a web browser and navigate to https://www.visualstudio.com.
2. In the upper-right hand corner, you might be automatically logged in with your Microsoft Account. If not, please login.
3. Once logged in, click the logged in user in the upper-right hand corner and select 'Visual Studio Profile' from the expanded menu.

Access the Azure DevOps Organization that holds your project

1. Under the header 'Azure DevOps Organizations' you will see all the organizations that you are involved with (marked as either Owner or Member)
2. Click the '>' arrow key next to an organization to expand out the listing of projects that are contained within that organization.
3. Click the link of the project name you wish to launch.

Add Members to your Project

1. At the bottom left of the project site you will see a navigation pane with a gear icon on it that says **Project settings**. Click this button.
2. Under the General tab, click 'Security'.
3. This will expand the security groups for the project. There are many groups that do a few different things, but the three basic ones are:

<ProjectName> Team	This role has basic rights to add code, run releases, etc
Readers	This role has read-only rights. They can view code on the Azure DevOps site and can see builds/releases but cannot clone, commit code or run builds/releases
Project Administrators	This role is the administrator role. Members added to this group can do anything in the project, including modifying security/group membership of other users

4. Click the group you wish to add a user to
5. On the right-hand action pane select the 'Members' column
6. Click the '+ Add...' button and type the member or group you wish to add to this security group, then click the 'Save Changes' button

Troubleshooting

When adding members to a project, we have seen instances where the user did not show up in the lookup window, but they were still able to be added successfully to the project by simply typing in the email address and clicking on Save changes. Once the user is added, refresh the members list.

Lab 6 - Git

Lab Description	**This lab will teach you some of the basics of the Git source control system**
Glossary of Terms	*commit* - a git commit is a code change as well as meta information about that change (who, what, when, and a user-entered description of the change)
	branch - a branch is a pseudo-copy of the code where commits you make do not affect the other branches. Commits you make to the new branch can either be merged into another branch (pull request) or discarded
Estimated Time to Complete	45 minutes
Key Takeaways	1. Learn about commits, syncing, and branches
	2. Make sure your changes are successfully stored on the source control server (Azure DevOps)
Author	Ralph Kyttle
	Wes Adams

- Popular framework used for source control
- Offline cloned repository
- Track and Merge changes
- Clone, Pull, Push, Fork

Exercise - Git Branch

Introduction to branching

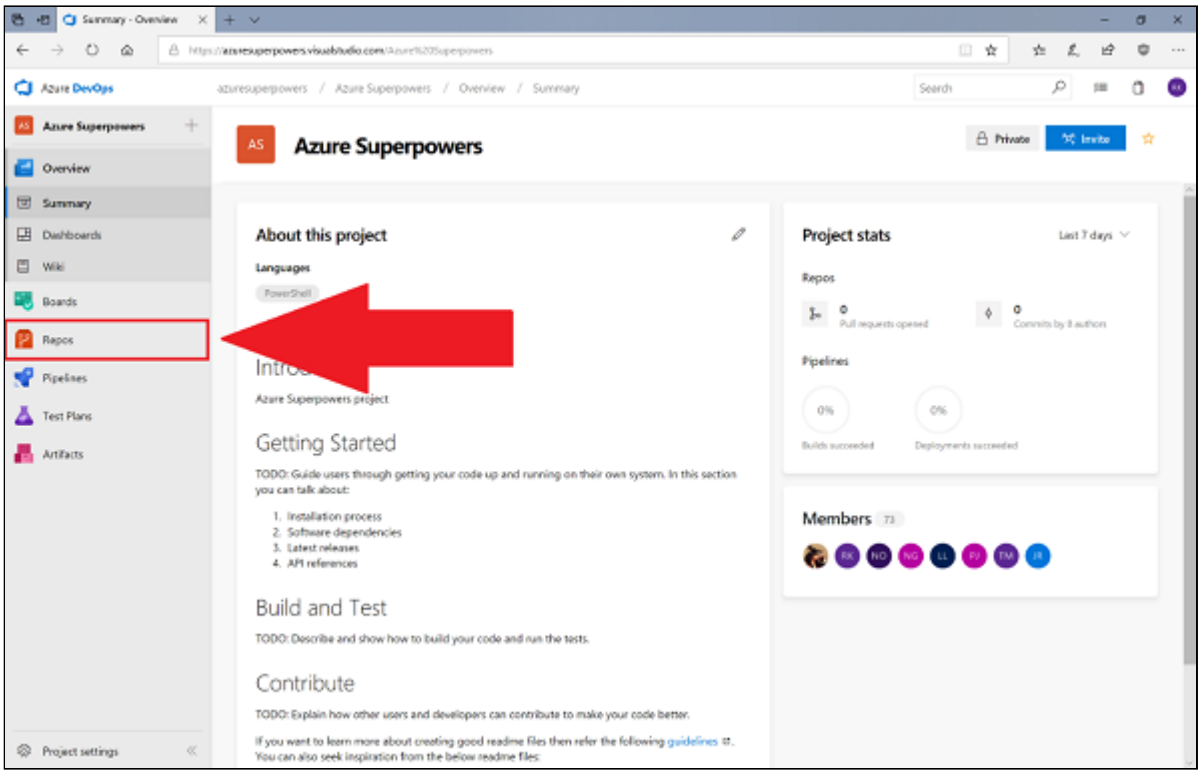
Each Git repository must have at least one branch. A branch is a sub-container within the repository for tracking file versioning. Each branch has its own history that is maintained. A branch can also be created at any time, and made as a "child" with another branch as the "parent". This means that you can start making changes to one or more files by spawning a new branch to make those changes. A branch provides an isolated environment to make file changes without affecting other branches. This can be important for maintaining the integrity of the files or source code for different environment deployments; like Production, Testing, or others.

Git branches aren't much more than a small reference that keeps an exact history of commits, so they are very cheap to create. Committing changes to a branch will not affect other branches, and you can share branches with others without having to merge the changes into the main project. Create new branches to isolate changes for a feature or a bug fix from your master branch and other work. Once the changes in a branch are completed, a pull request can be created to merge the changes in a branch into the base branch. The root branch is referred to as master.

There are a few critical branches in your repo that the team relies on always being in good shape, such as your master branch. Require pull requests to make any changes on these branches. Developers pushing changes directly to any protected branches will have their pushes rejected.

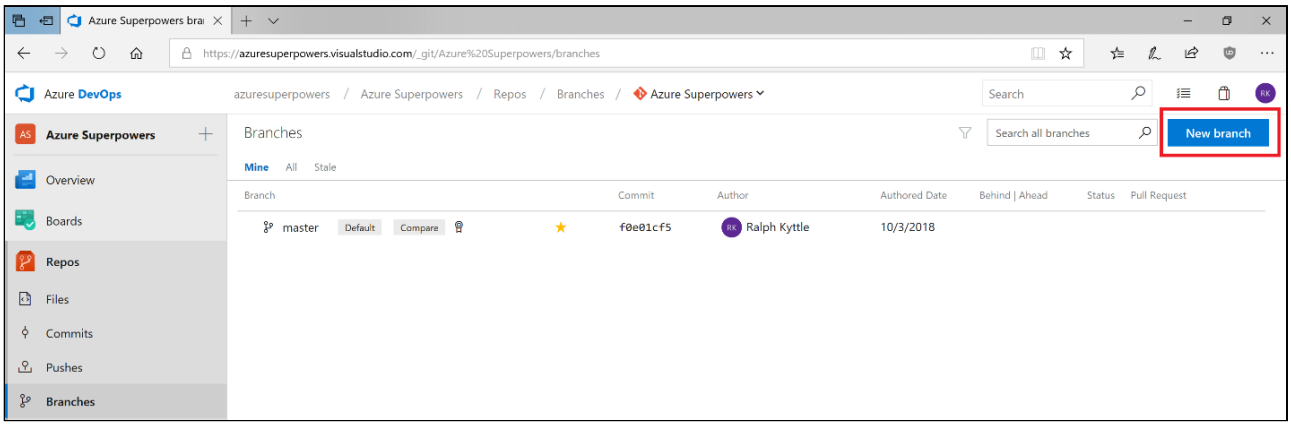
Create a new dev branch in your new project in Azure DevOps

1. Open a web browser and navigate to https://dev.azure.com/YourOrganizationName
2. Open the Azure Superpowers project
3. Once you are in the Azure Superpowers project, enter the Repos section of the project



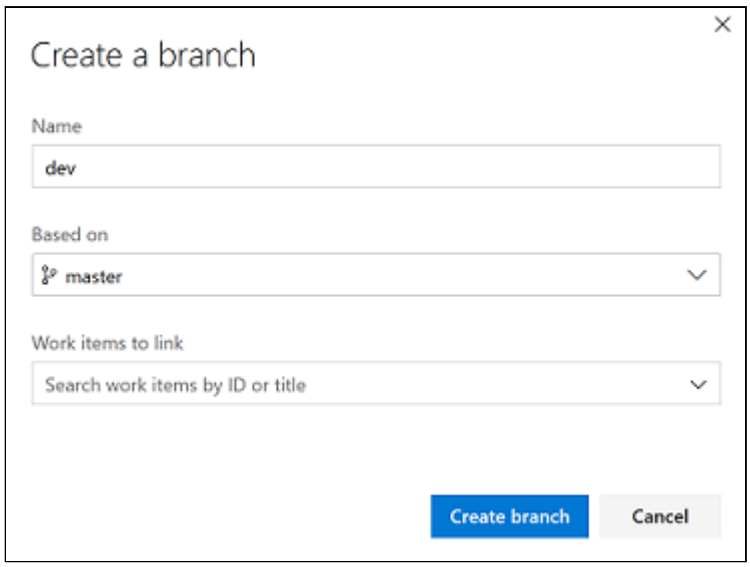
4. Under Repos, select the **Branches** button

5. Click the **New branch** button



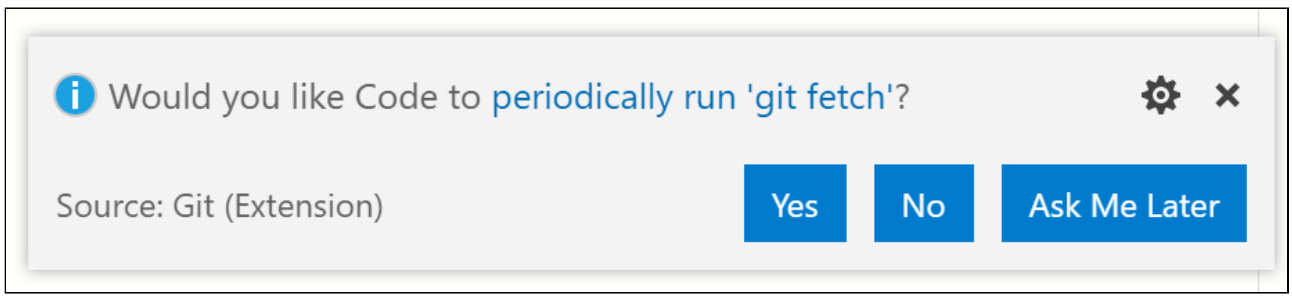
6. Create one new branch, and name it dev. Base the dev branch on the master branch.

   Example:



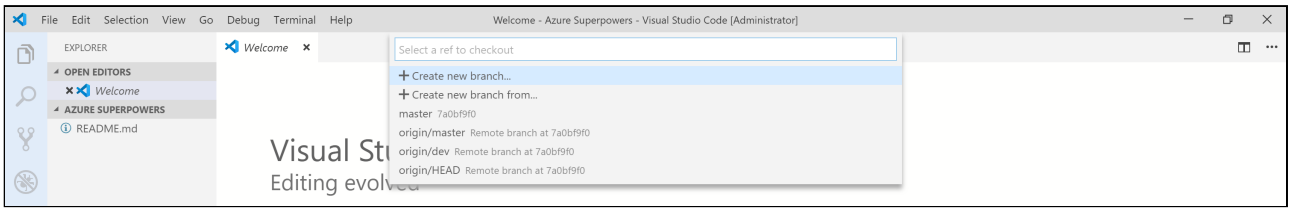
Fetching new branches in VS Code

1. Now that your new branch has been created in Azure DevOps, you will use VS Code and interact with your branches.
2. At this point, VS Code is not yet aware of your new dev branch. To make VS Code aware of new branches, you will need to perform a **git fetch** operation.
3. From within VS Code, View -> Command Palette
4. In the command palette, type **git fetch** and press enter
5. Git fetch will not return any status back to VS Code, but if it completes successfully your new branch will be available for selection in the next section of this exercise
6. If you are prompted to periodically run 'git fetch' we recommend you say **Yes** so that VS Code will automatically notice new branches when they are created



Switching between branches

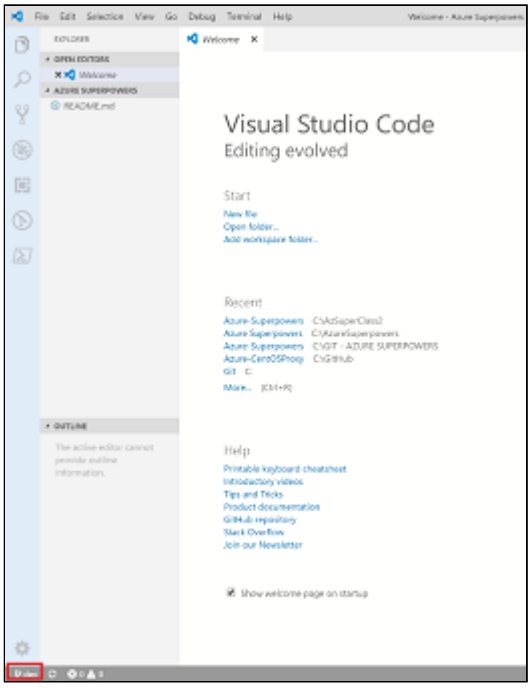
1. Press F1 to activate the Command Pallet
2. Type **git checkout to** and press enter



3. Notice that all available branches are listed in the command pallet. Click **origin/dev** which will create a local copy of your dev branch and automatically switch to that branch.

   NOTE: Branches listed beginning with origin/branch refer to branches stored in a remote repository, in this case Azure DevOps

4. Notice that VS Code will update at the bottom left of the screen with the currently selected branch which should now state **dev**. See screenshot following below for an example.



5. Press F1 to activate the Command Pallet
6. Type **git checkout to** and press enter
7. Click **master** to switch to your master branch.

8. Notice that VS Code will update at the bottom left of the screen with the currently selected branch, which should now state **master**.

   You now have the knowledge needed to switch branches in VS Code. With Git, everyone has their own local copy of code and can work simultaneously on their own branches. Git also works when you're offline since almost every operation is local.

   Branches allow for flexible and simultaneous development. The master branch contains stable, high-quality code from which you release. Feature branches contain work in progress, which you merge into the master branch upon completion. By separating your master branch from development in progress, you can protect your production environment and ensure that it only receives code that has been properly reviewed and approved.

   You can configure Azure DevOps to enforce consistent workflows and process across your team. Set up branch policies to ensure that code merges meet your requirements before completion. Branch policies can be configured to protect your important branches by preventing direct pushes, requiring reviewers, and ensuring clean builds.

Exercise - Branch policies

Introduction to branch policies

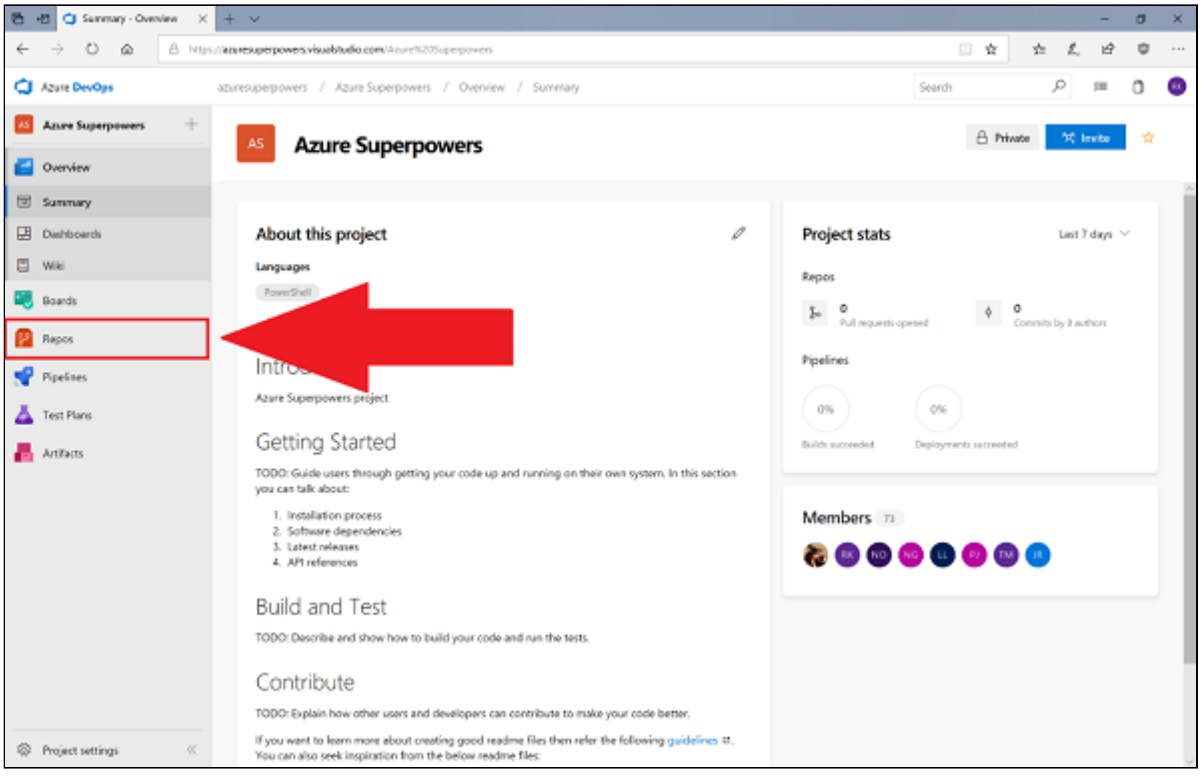
Branch policies are an important part of the Git workflow and enable you to:

- Isolate work in progress from the completed work in your master branch
- Guarantee changes build before they get to master
- Limit who can contribute to specific branches
- Enforce who can create branches and the naming guidelines for the branches
- Automatically include the right reviewers for every code change
- Enforce best practices with required code reviewers

Create a branch policy on your master branch

The steps below were adapted from information found at the following url: https://docs.microsoft.com/en-us/vsts/repos/git/branch-policies

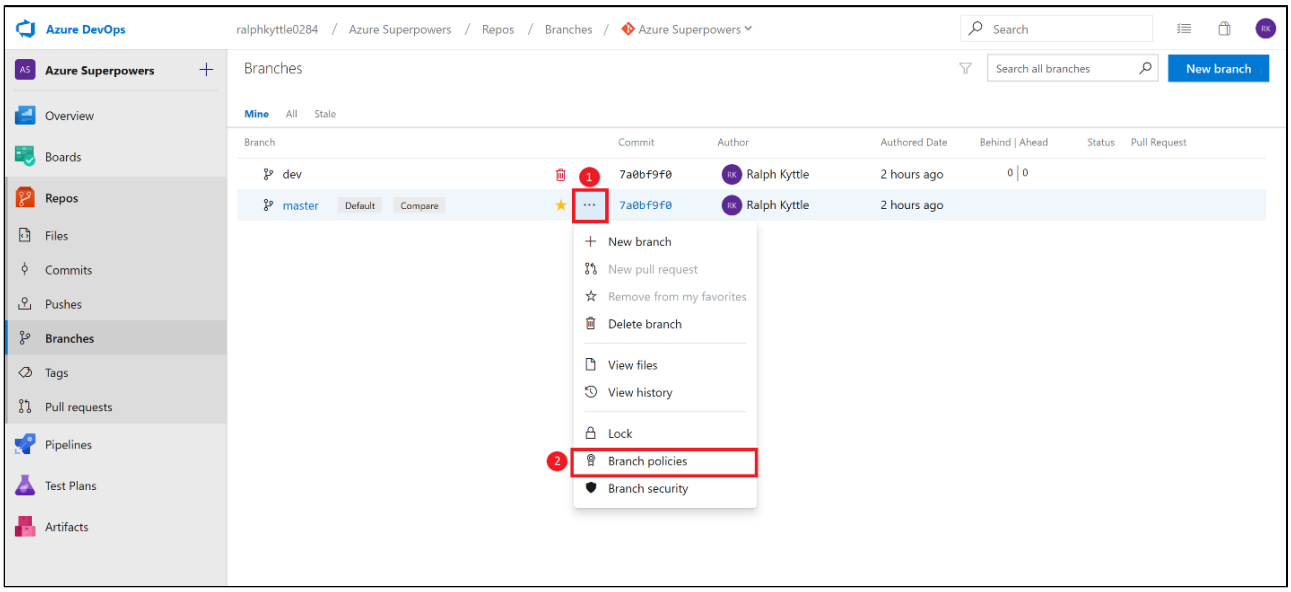
1. Open a web browser and navigate to https://dev.azure.com/YourOrganizationName
2. Open the Azure Superpowers project
3. Once you are in the Azure Superpowers project, enter the Repos section of the project



4. Under Repos, select the **Branches** button

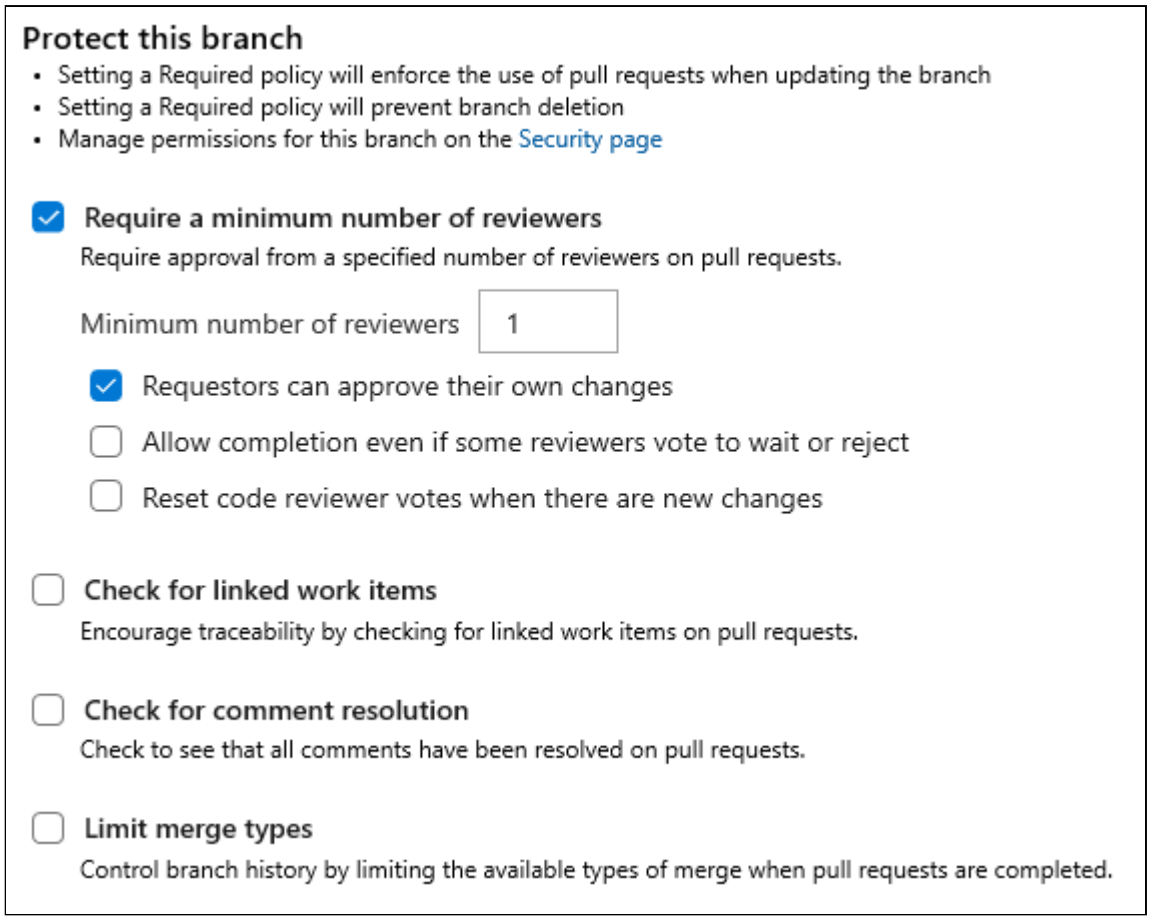
5. Locate the **master** branch in the view. You can browse the list or you can search for your branch using the Search all branches box in the upper right.

6. Open the context menu for the **master** branch by selecting the ellipsis (...) icon. Select **Branch policies** from the context menu.



7. For this example, you will setup a very simple branch policy to ensure that code cannot be pushed directly to the **master** branch.

8. Set your branch policy to match the settings shown below and save your changes to apply your new policy configuration.



9. Notice how **Requestors can approve their own changes** is selected. If **Requestors can approve their own changes** is not selected, the creator of the pull request can still vote Approve on their pull request, but their vote won't count toward the Require a minimum number of reviewers policy.

10. Save your changes. Even a simple policy like the one you just created will enforce the use of pull requests when updating the branch and will prevent branch deletion.

Exercise - Git commit

Introduction to Commit

A commit is the act of recording changes to the repository. A repository contains all files of a project and stores each file's revision history. When changes are made to one or more files, those can be saved to a branch as a commit. Each commit can contain the changes to one or multiple files.

Configure your Git identity

1. In order to be able to commit your changes to a repository, you must set your name and email address. This is important because every Git commit uses this information, and it's immutably baked into the commits you start creating.
2. Open VS Code
3. From the top menu bar, select View, Terminal
4. Enter the following commands, **filling in your own values for your name and email address**

```
git config --global user.name "Your Name"

git config --global user.email "you@example.com"
```

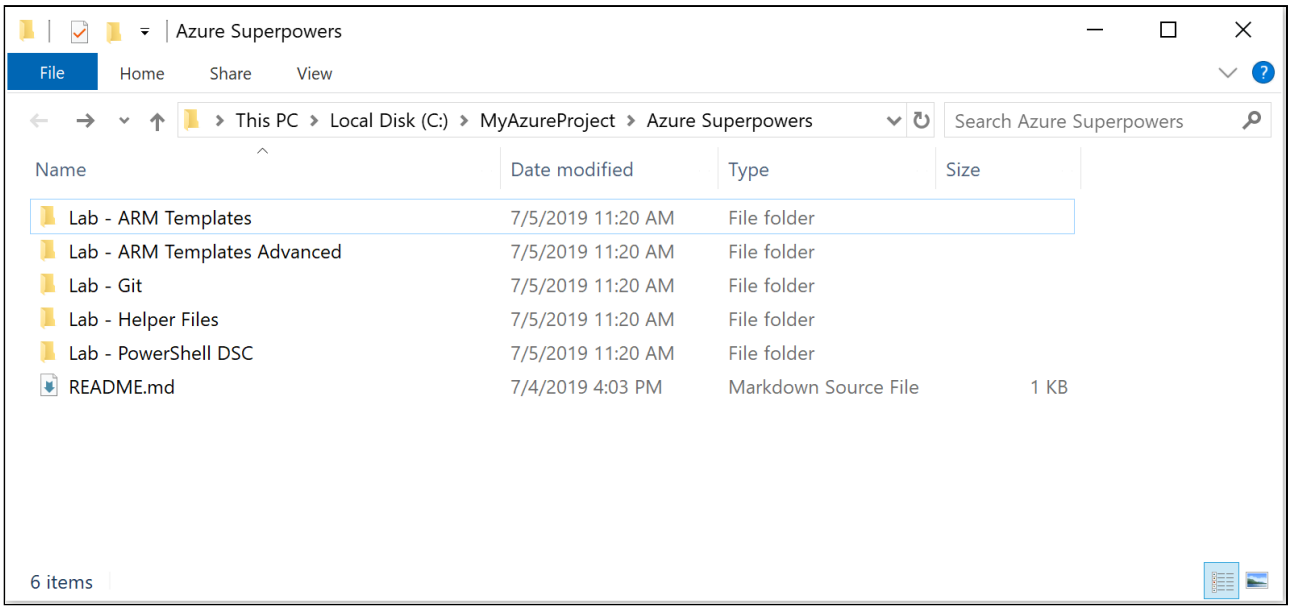
These configuration settings will be saved to your user profile at following file path: C:\Users\username\.gitconfig

You need to do this only once if you pass the --global option, because Git will always use that information for anything you do on that system. If you want to override this with a different name or email address for specific projects, you can run the command without the --global option when you're in that project.

Copy files from GitHub

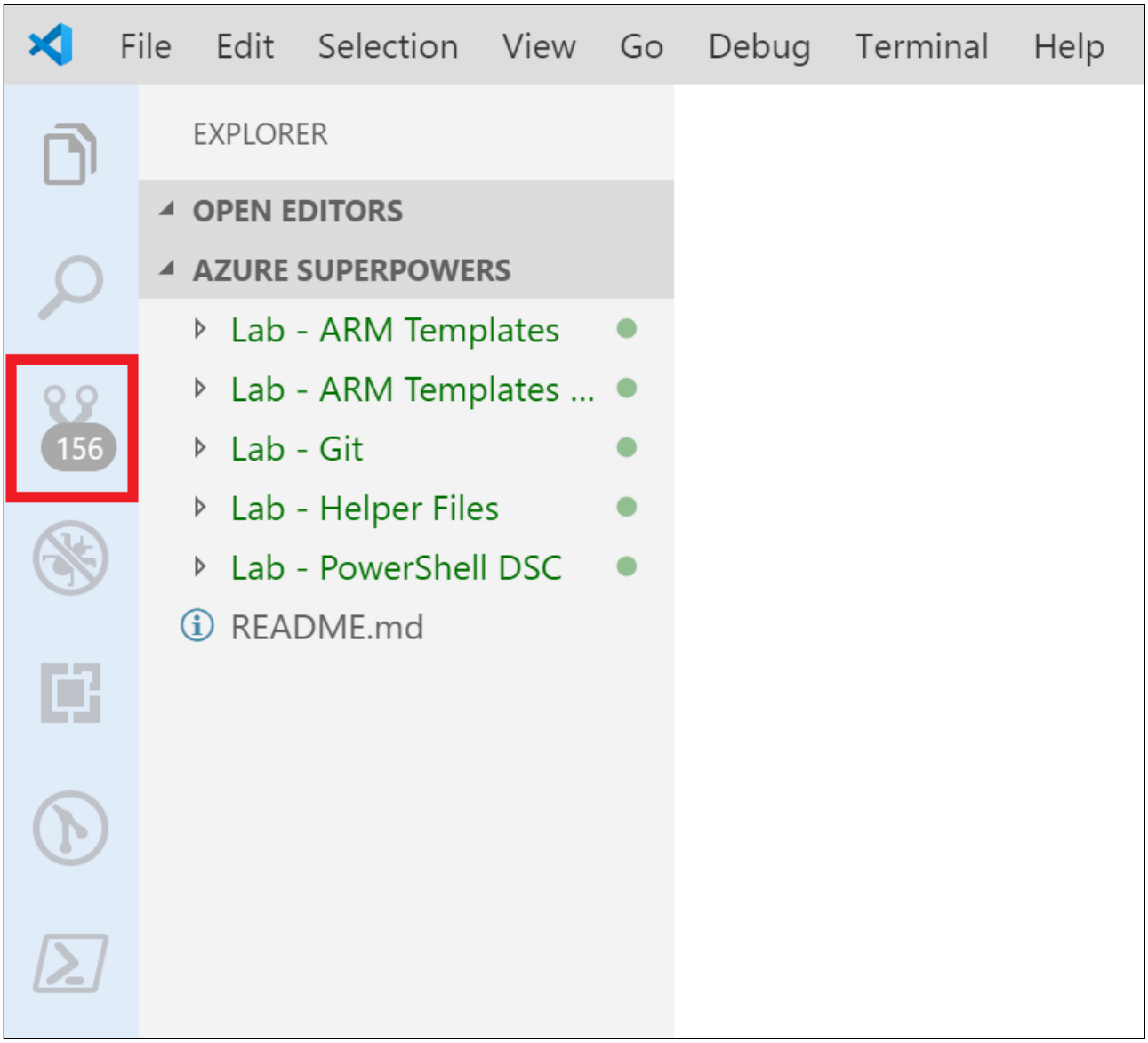
1. Your repository is currently empty, except for the default README.md file
2. In the next steps, you will copy some of the files you cloned from the Azure Superpowers repository hosted on GitHub over to a repository that you own and control hosted on Azure DevOps
3. Press F1 to activate the Command Pallet
4. Type **git checkout to** and press enter
5. Click **dev** to switch to your dev branch.
6. Inside of your VS Code window, right click on the README.md file and select **Reveal in Explorer** which should open Windows Explorer with **C:\MyAzureProject\Azure Superpowers** as the current directory
7. Copy all folders located within **C:\AzSuperClass\AzureSuperpowers\src** over to **C:\MyAzureProject\Azure Superpowers**

8. Completing the steps above should result with the following:



Performing a Commit using VS Code

1. Return to VS Code
2. Click on the Source Control icon, as indicated in the screenshot below. When there are changes to commit, the Source Control icon will indicate how many files have either been created or modified.



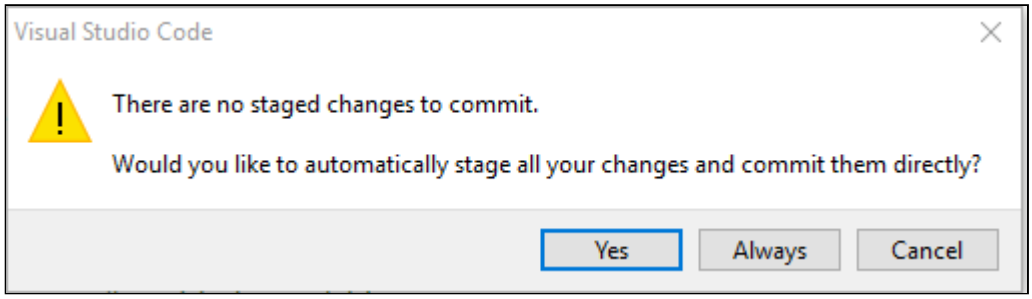
3. Clicking on the Source Control icon will change your view to show the list of changes that are able to be staged and committed to the local copy of your repository. You can click the **Stage Changes** icon (plus sign) next to each file, which appears after hovering over the file. This stages the change to be included in our commit. In real world use cases, you can select one or multiple files to stage to be part of a commit in order to group a set of like changes together.

4. You can also perform a commit in VS Code without staging any files first, and VS Code will ask if you would like to stage all the changes it has detected. This can be useful if you have made changes to many files and you would just like them all to be added as a single commit.

5. In this exercise you will have many files to commit. Instead of staging each file individually, you will follow the guidance above to perform a commit without staging any files, and you will receive a message similar to the screenshot below.

6. In the **Message** field, enter a short description of the change being made. For example, Initial Commit.

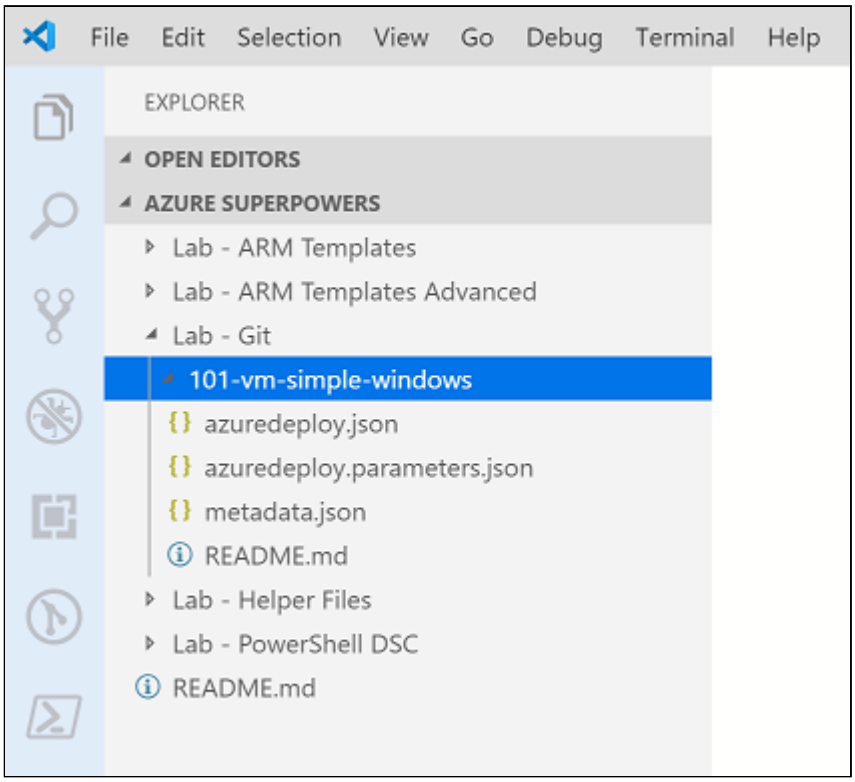
7. Click the checkmark at the top of the sidebar, located right above the Message field. This action commits the change to the repository hosted on your **local file system.** Note that the change has not yet been updated to the hosted Git repository in Azure DevOps. When prompted, select **Yes** to allow VS Code to stage and commit all of the files.



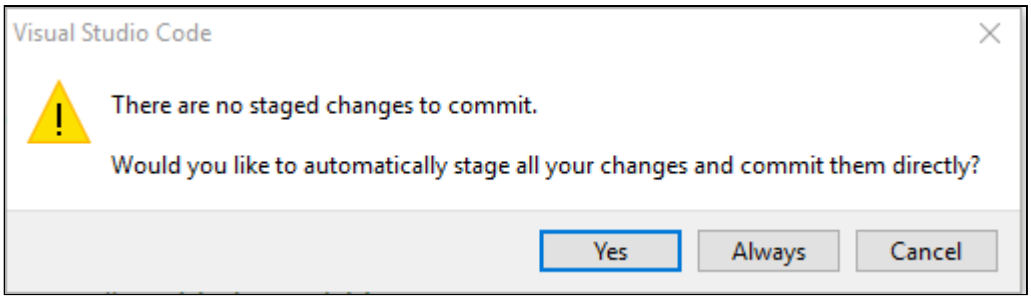
8. In the status bar at the bottom of VS Code, notice the number 1 next to the up arrow. This indicates that there is one pending commit waiting to be pushed to the Git repository stored in Azure DevOps.

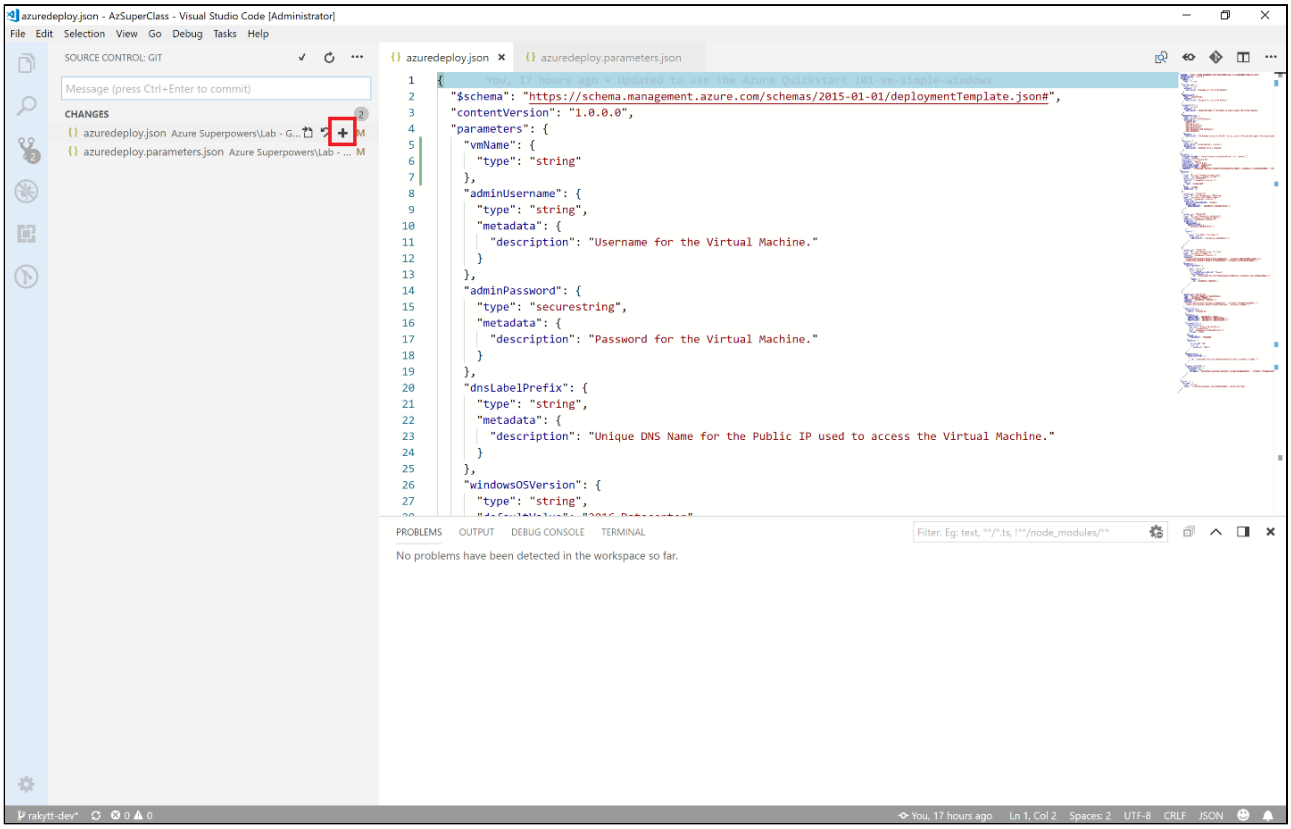
Update ARM Template and commit changes

1. Return to VS Code
2. Ensure that you have your **dev** branch selected
3. In the Explorer View (Select the File icon in the Activity Bar) expand out the folder structure for Azure Superpowers\Lab - Git\101-vm-simple-windows



4. In this exercise you will make changes to the **azuredeploy.json** and **azuredeploy.parameters.json** files, and you will commit the changes to these two files as a single commit to your dev branch of your Azure Superpowers repository.
5. These files were taken directly from the Azure Quickstarts GitHub website, and they are a useful example for showcasing ARM template structure and syntax, but they need a bit of work if we are going to deploy them in an enterprise environment.
6. Edit **azuredeploy.json**, changing vmName from being declared as a hard-coded variable, and instead make it a parameter. Edit **azuredeploy.parameters.json** as well to include a value for this new parameter, then save your changes to both files.
7. Inside VS Code, Source Control lists all changes in the project. Click the **Stage Changes** icon (plus sign) next to each file, which appears after hovering over the file. This stages the change to be included in our commit. In real world use cases, you can select one or multiple files to stage to be part of a commit in order to group a set of like changes together.
8. You can also perform a commit in VS Code without staging any files first, and VS Code will ask if you would like to stage all the changes it has detected. This can be useful if you have made changes to many files and you would just like them all to be added as a single commit.





9. In the **Message** field, enter a short description of the change being made. For example, Initial Commit.

10. Click the checkmark at the top of the sidebar. This action commits the change to the repository hosted on your **local file system.** Note that the change has not yet been updated to the hosted Git repository in Azure DevOps.

11. In the status bar at the bottom of VS Code, notice the number next to the up arrow is updated to 2. This indicates that there are now two pending commits waiting to be pushed to the Git repository stored in Azure DevOps. Throughout this exercise you committed over 100 files but you only performed the commit operation twice, so VS Code is indicating that there are two commits that are ready to be pushed to the Git repository stored in Azure DevOps.

Exercise - Git Pull and Git Push

Introduction

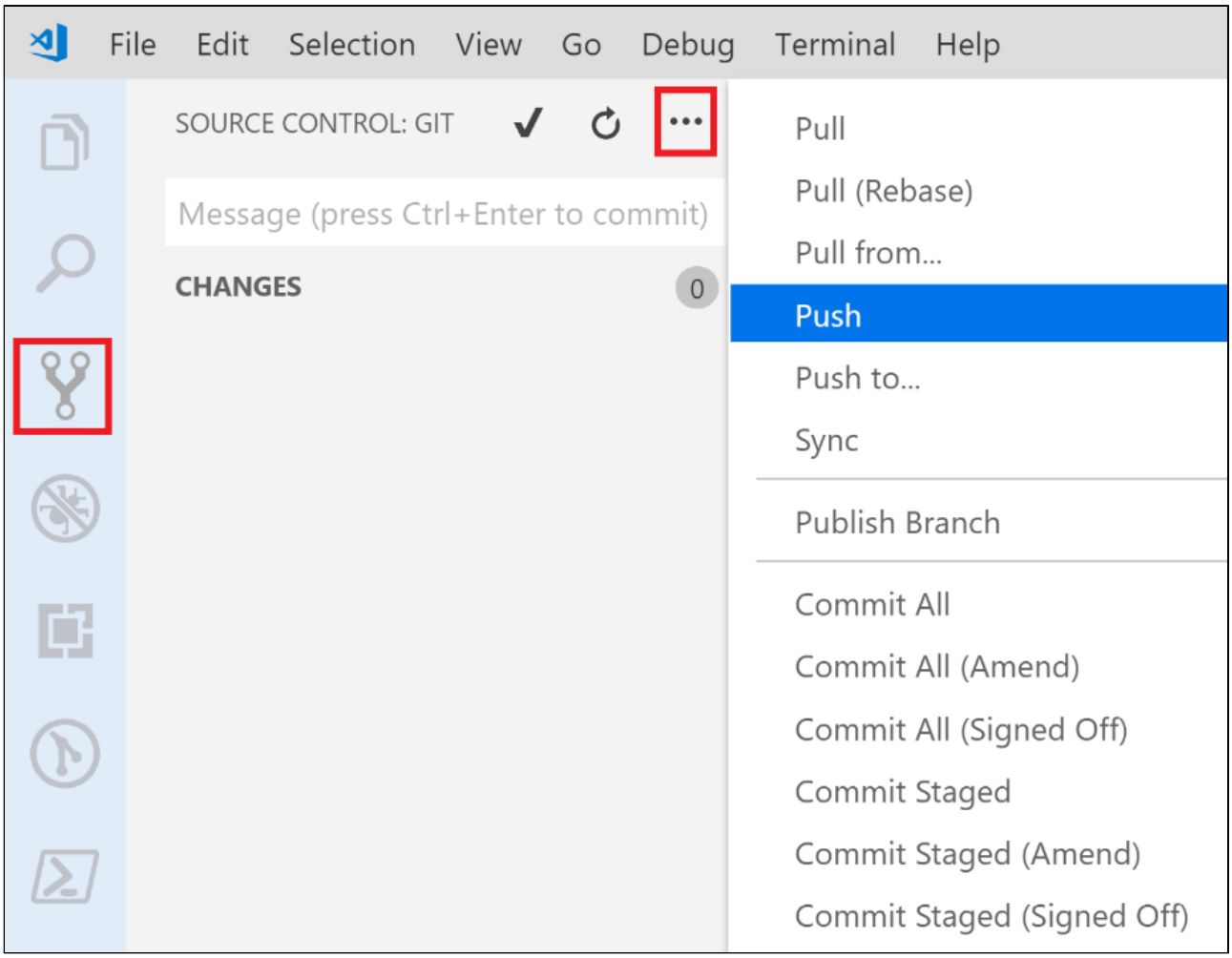
A **git pull** is the act of *pulling* in all of the changes made by others in the selected git repository branch and syncing them to your local copy of the repository. A **git push** is the act of *pushing* the changes that you have made locally so that your changes can be made available to others working in the repository.

IMPORTANT: Always pull in any changes to your local repo first before pushing new changes up

Performing a Git Push using VS Code

1. You should have two pending commits from the activities performed in Exercise - Git commit.
2. Notice the number next to the up arrow in the Status Bar. This indicates the number of commits that have been added to your local repository. In the screenshot below, there are two commits that are ready to be pushed up to Azure DevOps. In addition, in this screenshot VS Code is not aware of any commits that need to be pulled down from Azure DevOps at this time, as we see a 0 next to the arrow for commits to be pulled down.

3. In order to push your changes up to the Azure DevOps repo via a **git push**, click on **Source Control**, click on the ellipsis icon, then click **Push**



4. Depending on your device configuration, you may be prompted for credentials in order to complete the push.

5. To verify that the push has been completed, navigate to the Output view by selecting **View** in the menu bar and selecting **Output**.

6. In the Output pane, notice the drop down menu in the upper right hand corner. In this drop down menu, select **git**.

7. This will show the git activities that have occurred. If you experienced an error of any kind, this is the first place to go to troubleshoot.

Exercise - Pull Request

Merging Changes to your master branch via Pull Request

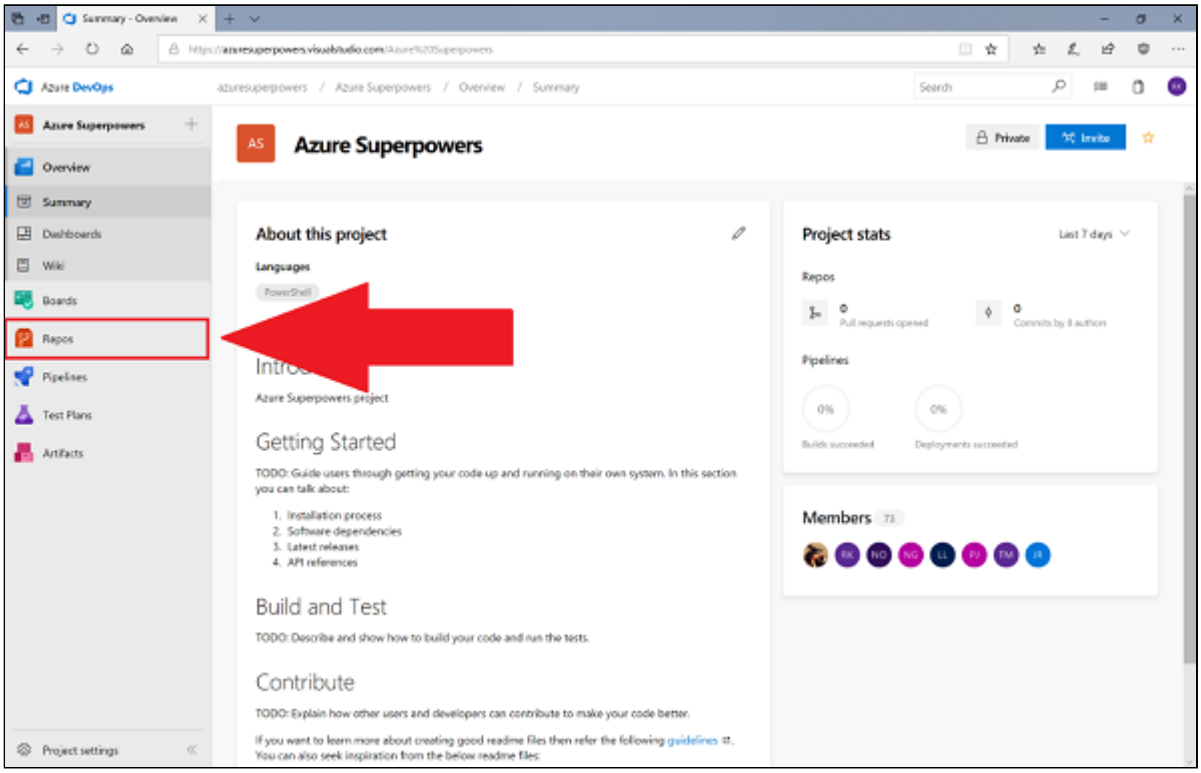
1. We learned earlier in **Exercise - Branch policies** that the branch policies that were implemented on the **master** branch will enforce the use of pull requests when updating that branch. Pull requests combine the review and merging of your code into a single collaborative process.

2. Given this policy, we will create a pull request to merge changes from the **dev** branch to the **master** branch.

3. Open a web browser and navigate to https://dev.azure.com/YourOrganizationName

4. Open the Azure Superpowers project

5. Once you are in the Azure Superpowers project, enter the Repos section of the project

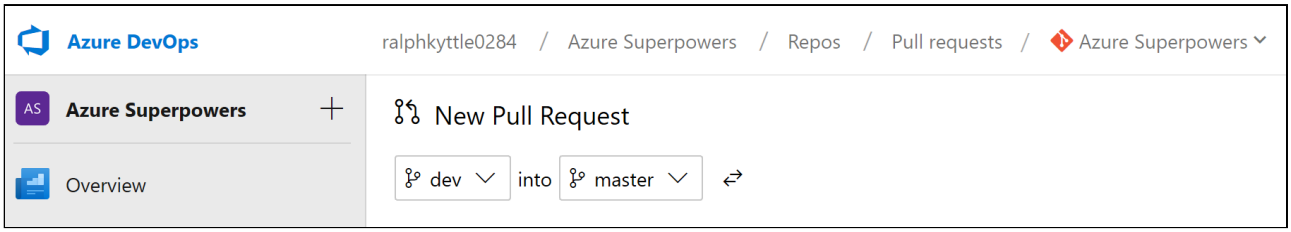


6. Select the Pull requests button, which will bring you to the following page: https://dev.azure.com/<YourOrganizationName>/_git/Azure%20Superpowers/pullrequests

7. Click the **New pull request** button

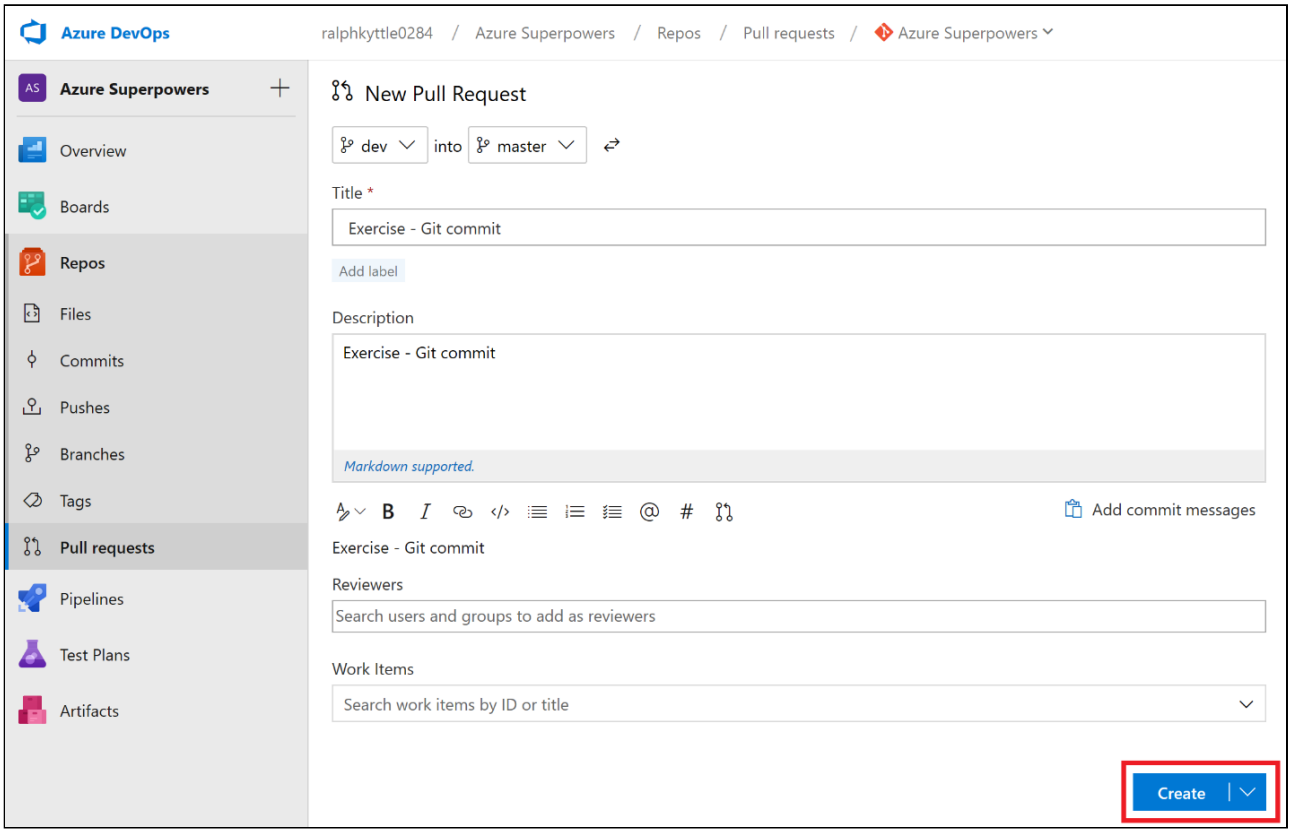
8. A pull request involves the merging of code from one branch into another branch. You may need to select the two branches that you want to have involved in the request, as Azure DevOps may default to other branches than what you are currently working with.

9. In this case, you will want to merge changes from the **dev** branch into the **master** branch.

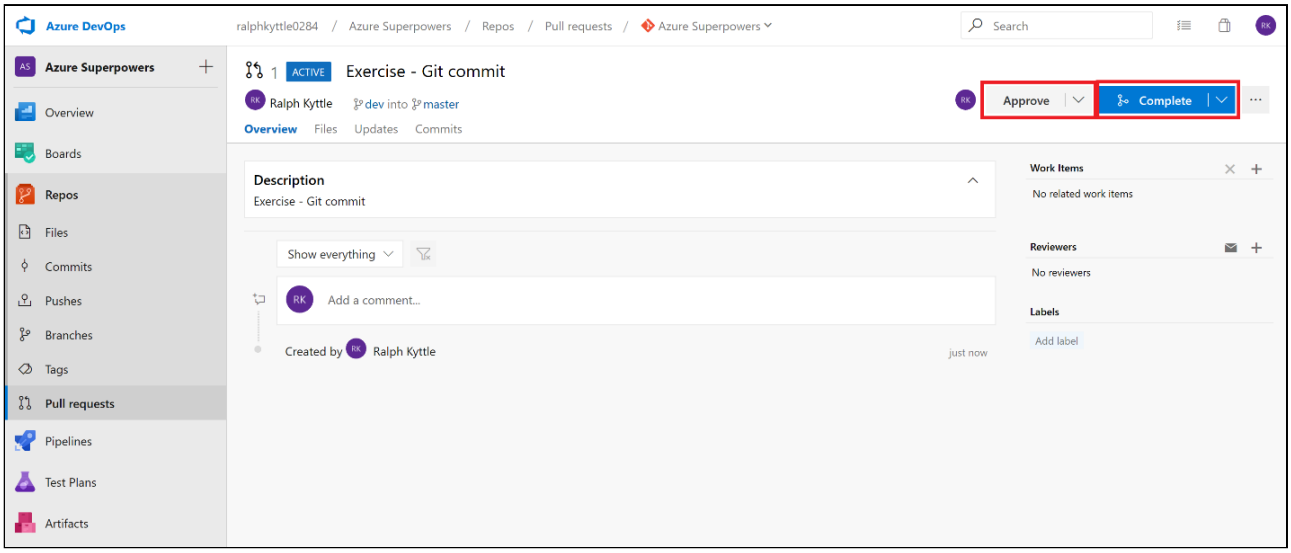


10. You should give a clear title for the pull request that describes the changes in the branch. In the description field give a clear explanation of how the changes are implemented along with any resources that might help reviewers understand the changes. You can include Azure DevOps work items and hyperlinks to allow others to have as much context as possible when reviewing your changes. You can also add any team member who you would like to review the changes.

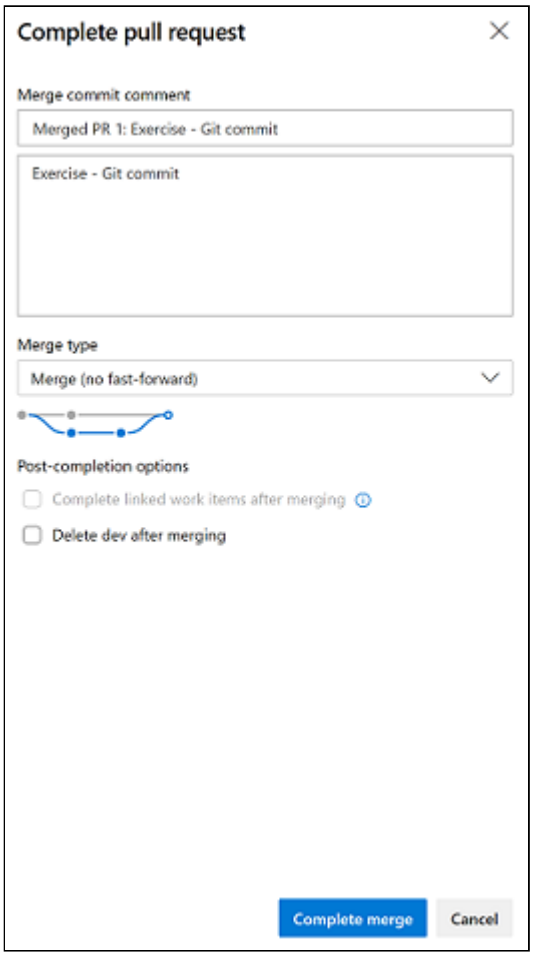
11. Click the Create button to create the pull request.



12. Approve the pull request and complete it.



13. On the **Complete pull request** window, **uncheck** the box that will delete your dev branch, and click **Complete merge**



Troubleshooting

Lab 7 - AD Super Lab Deployment

Lab Description	**This lab is a prerequisite for completing the ARM Templates (Advanced) lab. In this lab, you will deploy resources into Azure using Azure PowerShell and ARM Templates.**
Glossary of Terms	*ARM Template* - An ARM Template is a JavaScript Object Notation (JSON) file that defines one or more resources to deploy to an Azure resource group. It also defines the dependencies between the deployed resources. The template can be used to deploy the resources consistently and repeatedly.
Estimated Time to Complete	10 minutes (User tasks)
	30 minutes (Deployment of resources to Azure)
Key Takeaways	1. Deploying a complex solution can be made simple using ARM Templates
Author	Ralph Kyttle

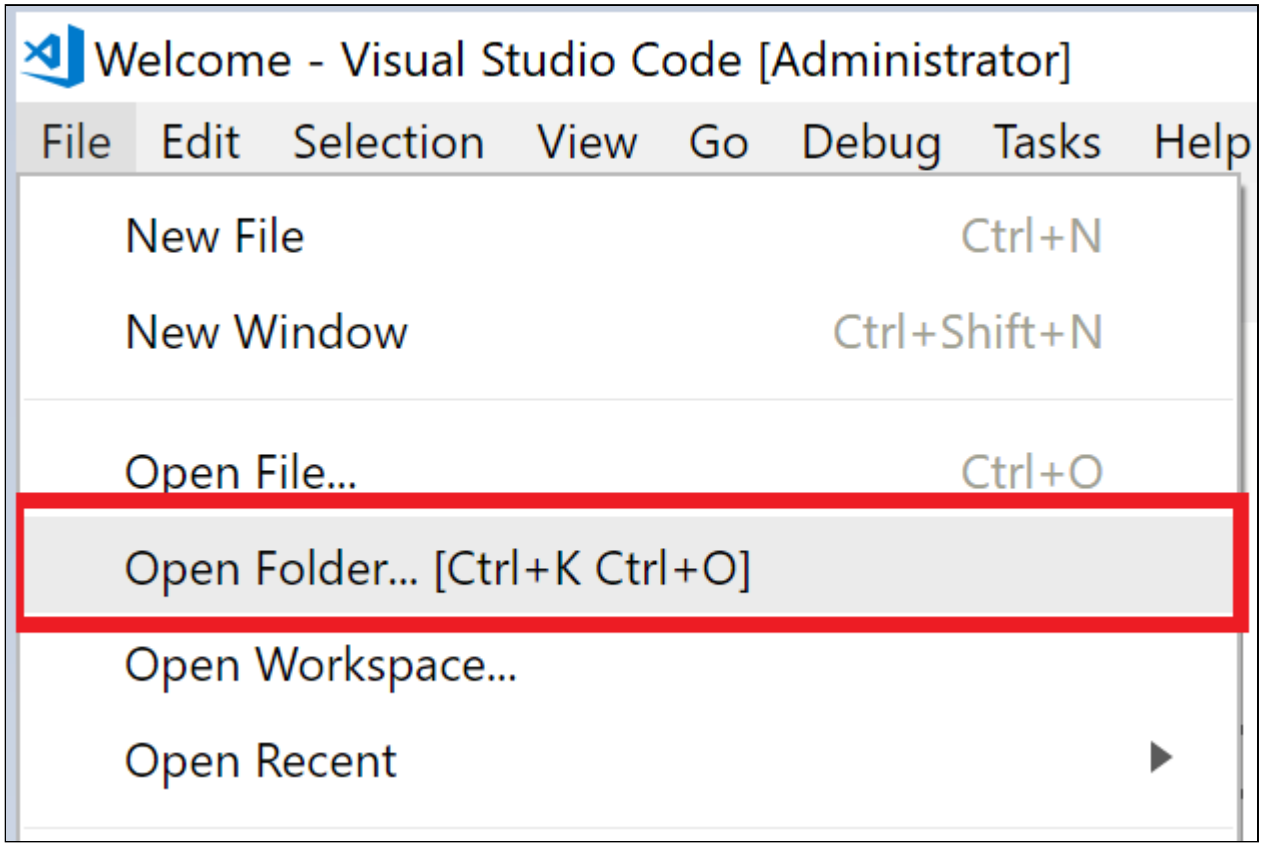
This lab should only take a few minutes to complete the user initiated actions. Once the deployment has started, it will take approximately 30 minutes to complete.

Exercise - Deploy AD Super Lab

Launch Visual Studio Code

(Visual Studio Code is strongly recommended for creating and editing Resource Manager templates)

1. Select **File** > **Open Folder...**



2. Open the following **folder**: C:\MyAzureProject\Azure Superpowers

3. Select **C:\MyAzureProject\Azure Superpowers\Lab - ARM Templates Advanced\Exercise1\adLabDeployment.ps1** and review its contents.

4. Instructions found in the ps1 file are also detailed in the next steps.

Login to Azure via PowerShell

From a PowerShell console window, login to Azure using either:

Azure Commercial:

```
Login-AzAccount
```

-or-

Azure Government:

```
Login-AzAccount -Environment AzureUSGovernment
```

And then login into Azure (as yourself) using the pop-up dialogue box (that may have popped-up under the current Window(s)).

```
Select-AzSubscription –Subscription '<Id>'
```

Next, create a new Resource Group called AzSuperADLAB-<YOURALIAS>

(Examples for Azure Commercial and Microsoft Azure Government shown below)

```
New-AzResourceGroup -Name 'AzSuperADLAB-<YOURALIAS>' -Location 'eastus'
#OR
New-AzResourceGroup -Name 'AzSuperADLAB-<YOURALIAS>' -Location 'usgovvirginia'
```

The template will take approximately 30 minutes to deploy

When you execute the deployment, it will prompt you for the following 3 values. Use the values below.

VMadminUsername: admuser

VMadminPassword: <Define a complex password, capture it so you can use it to login>

numberOfvmWorkerInstances: 3

Execute the following PowerShell

```
1.      $DeploymentParametersBuildVM = @{
2.      ResourceGroupName = 'AzSuperADLAB-<YOURALIAS>'
3.      TemplateUri       =
'https://azuresuperpowers.blob.core.windows.net/arm/Exercise%201/adLab.json?
st=2018-07-21T18%3A49%3A48Z&se=2020-12-31T18%3A49%3A00Z&sp=rl&sv=2018-03-
28&sr=b&sig=X8HpZw2HeteWtf3F9WSfry3WizXJsejv%2Brv%2BcUrmeTM%3D'
4.      Timestamp         = [system.DateTime]::Now.ToString("MM/dd/yyyy H:mm:ss
tt")
5.      CreateWorkerNodes = 'true'
6.      FullPathToFile =
'https://azuresuperpowers.blob.core.windows.net/arm/Exercise1/CustomScriptExtensio
nFiles/CheckForAD/CheckForAD.ps1?st=2019-03-27T01%3A55%3A14Z&se=2021-03-
28T01%3A55%3A00Z&sp=rl&sv=2018-03-
28&sr=b&sig=ku9y0cpYebgB7fmfVMTT4kpOCjKEjzaEqoZOs3xDKpg%3D'
7.      FileNameAndExtension =
'.\Exercise1\CustomScriptExtensionFiles\CheckForAD\CheckForAD.ps1'
8.      Verbose           = $true
9.      }
10.     Test-AzResourceGroupDeployment @DeploymentParametersBuildVM
```

If the test completes successfully, move forward with the deployment by executing the following PowerShell:

```
New-AzResourceGroupDeployment @DeploymentParametersBuildVM
```

Ensure that you capture the username and password used in your deployment, as you will need them to login.

You can review the Deployments tab in the Azure portal for the resource group that you are targeting to see real-time deployment information, as well as any errors or warnings if your deployment is not successful.

Troubleshooting

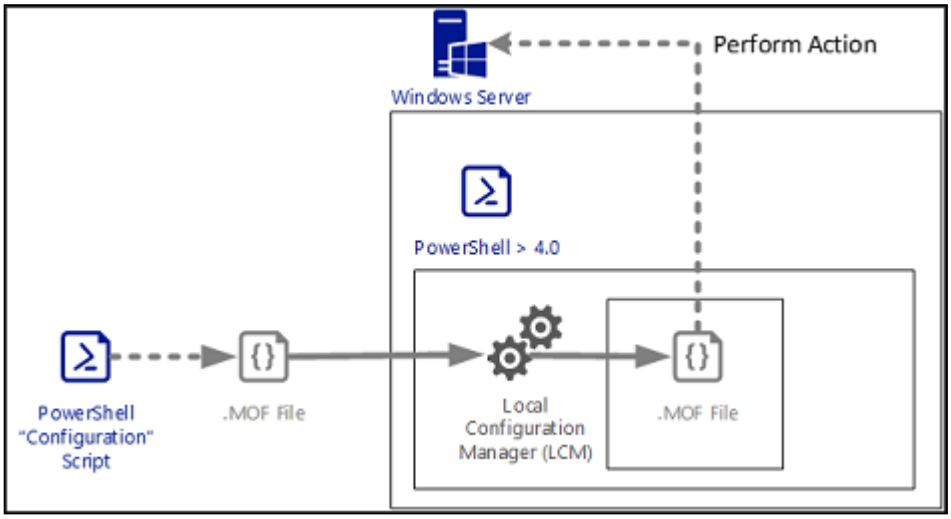
Lab 8 - PowerShell DSC

Lab Description	**This lab is to familiarize yourself with PowerShell DSC, writing a basic DSC configuration, creating a localhost.mof, testing it locally on the system, and then using Azure to deploy that configuration to an Azure Virtual Machine. This lab focusses on using DSC in the Push based model.**
Glossary of Terms	*DSC configuration* – DSC configurations are PowerShell scripts that define a special type of function. To define a configuration, you use the PowerShell keyword Configuration. Configurations are Declarative PowerShell scripts which define and configure instances of resources.
	DSC resource – The "make it so" part of DSC. They contain the code that put and keep the target of a configuration in the specified state. Resources reside in PowerShell modules.
	Local configuration manager (LCM) – The engine by which DSC facilitates the interaction between resources and configurations. Installed by default on all Windows systems with Windows PowerShell version 4.0 and above.
Estimated Time to Complete	60 minutes
Key Takeaways	1. How to author a DSC configuration
	2. How to create a ZIP file that works with Azure (Containing a .ps1 file that has the 'configuration' keyword in it)
	3. Where to put the ZIP file so that Azure can tell an IaaS VM to reach out and download/extract/apply it
	4. Why DSC (which can be applied many times and only ever produces the same result) is a good example of 'end state programming' and why end state is best for cloud technologies
	5. DSC is integral for us to involve IaaS solutions in code based deployments and release pipelines
Author	Keith Hitchcock
	Ralph Kyttle

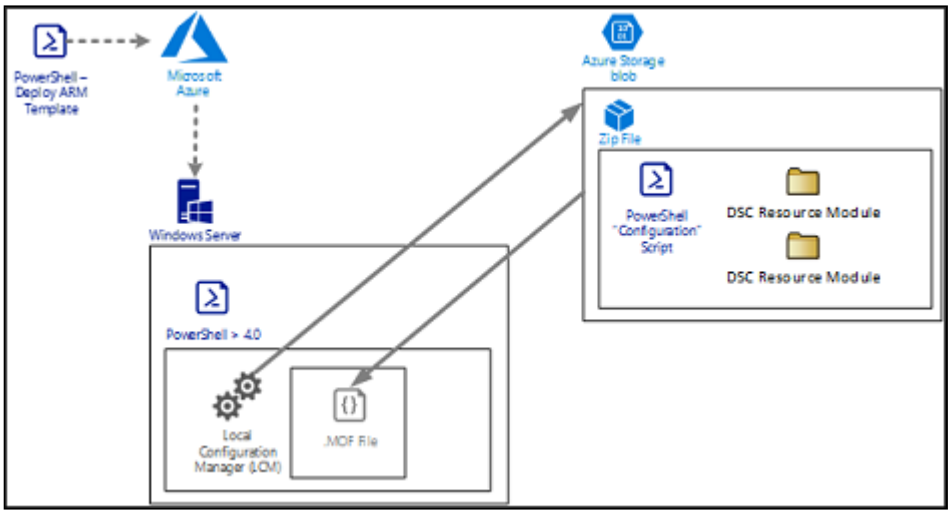
PowerShell DSC

- 'End State' programming
- The LCM (Local Configuration Manager) engine that exists on each system takes care of 'making it so'
- PowerShell creates a configuration file (.MOF format)
- Configurations can be applied and re-applied infinitely with no ill-effects
- Someone else does the hard work (DSC Resources)
- Configurations can now be kept in Source Control

Local DSC Flow



Azure DSC Flow



Basic PowerShell Function

```powershell
function DemoFunction {
    param($HappyMessage)

    Write-Host $HappyMessage
}

DemoFunction -HappyMessage 'Functions are Easy'
```

Basic DSC Configuration

```powershell
configuration DemoDSCConfiguration {
    param($Value)

    node 'localhost' {
        Environment 'EnvironmentResource' {
            Name  = 'Demo'
            Value = $Value
        }
    }
}

DemoDSCConfiguration -Value 'DemoValue'
```

Exercise - Exploring DSC Custom Resources on the Web

Explore the PowerShell Gallery

1. From any web browser, navigate to https://powershellgallery.com
2. In the search box, type 'DSC' and click the button to search for a module
3. Click into one of the modules
4. Note the PowerShell command (5.0 or higher) to Install the Module (from an Internet connected system)
5. Note the Owner/Author of the module (Modules from well-known trusted authors are recommended)
6. On the left-hand side click the 'Project Site' link. This generally links to GitHub, and you can explore the code that makes up the module. For DSC resources, each PowerShell module has a subfolder called 'DSCResources' where the DSC resources are located

Explore Github.com

1. From any web browser, navigate to https://github.com
2. In the search box, type 'sqlserverdsc' and click the button to search
3. Click the hyperlink to 'PowerShell/SqlServerDSC' which is the link to the SqlServerDsc project in github
4. Scroll down the page and note the list of 'Resources' and the detailed use of each resource. The documentation of how to use the DSC resources indicates a mature DSC resource module

Exercise - Build a new Azure VM and install custom DSC resources

Create a new Windows Server 2016 VM using the Azure Portal

1. Open a web browser
2. Navigate to either https://portal.azure.com for commercial, or https://portal.azure.us for MAG
3. Choose **Create a resource** in the upper left-hand corner of the Azure portal.
4. Under **Popular** Click on Windows Server 2016 Datacenter
5. Responses to the portal wizard are listed below. Your portal experience may look different as the portal updates frequently, but the information needed to deploy should be listed below.
6. Basics tab

   a. Make sure the correct subscription is selected and then choose to **Create new** resource group

   b. Enter a name for the resourcegroup: portaldscrg-<YOURALIAS>

   ```
   Example: portaldscrg-josmith
   ```

   c. Type DSC1 for the virtual machine name

   d. Choose East US or USGov Virginia for your Location

   e. Choose D2s_v3 (Or a similar size if this size is not in your region)

   f. Provide a username, such as azureuser and a password. The password must be at least 12 characters long and meet complexity requirements. **Make sure to remember the password as you will use it to login**.

   g. Under public inbound ports, click **Allow selected ports** and select RDP so you can login to this lab machine via its public IP

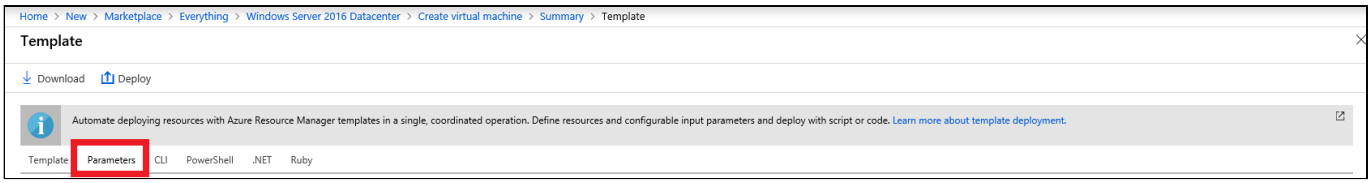
   h. At the bottom of the screen, Click **Review + create**

7. Review + create tab

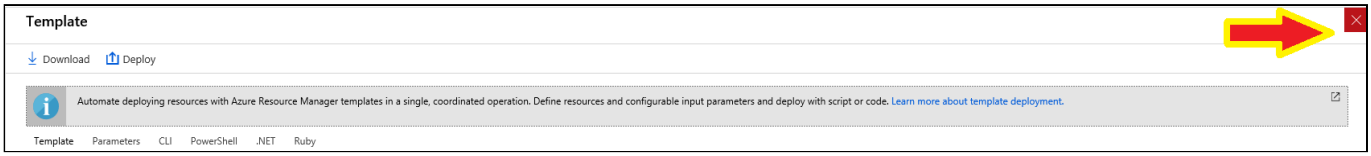
   a. **DO NOT IMMEDIATELY CLICK OK.** Instead, look to the bottom of the page and click on the link **Download a template for automation**

   b. This will present you with an ARM template that matches your deployment as you defined in the portal. This can be very helpful when you first start learning how to work with ARM templates, or with finding the values and parameters an ARM template is expecting.

   For example, note how when you click on the parameters button, the value it has listed for location looks a bit different from what you selected in the portal.



To return back to the deployment summary, click on the X to exit out from the Template



Click Create to start the deployment of your VM, which will take 5-10 minutes to complete.

Login to the VM

1. Open an RDP session to the DSC1 VM created in the previous step. Once logged into this VM, open the PowerShell ISE as an Administrator

Get a custom DSC resource from the PowerShell Gallery

1. In the section below, you may be prompted multiple times to confirm downloading nuGet and SecurityPolicyDSC, click or select yes when prompted.

2. From the PowerShell ISE console of the VM, execute:

```
Find-Module -Name '*security*'
```

3. You will see multiple answers since this is searching against the PowerShell gallery, you want the 'SecurityPolicyDSC' module, which you can download via:

```
Install-Module -Name SecurityPolicyDSC
```

Explore the required information for the custom resource

1. Execute the following PowerShell to see which DSC resources are contained within the SecurityPolicyDSC module that is now installed locally:

```
Get-DSCResource -Module SecurityPolicyDSC
```

2. Based on the output from the command above, explore the 'AccountPolicy' DSC resource of the 'SecurityPolicyDSC' module

```
Get-DSCResource -Name AccountPolicy
```

3. Although the above command just gets us the one 'AccountPolicy' DSC resource, PowerShell is hiding the important information. Use the following command to get the important part of the PowerShell object:

```
Get-DSCResource -Name AccountPolicy | Select -ExpandProperty Properties
```

4. The above command tells us about what the DSC resource needs (Mandatory) and what datatype needs to be provided (strings, integers, etc) and also the specific names of the resource properties. You'll see an example of an xFirewall resource in the next exercise. You can use this same procedure to explore the xFirewall DSC resource and look for additional configuration settings that the DSC resource can control.

Exercise - Create a DSC Configuration

Login to the VM created in the previous exercise

1. Open an RDP session to DSC1
2. Open the PowerShell ISE as an Administrator

Get a custom DSC resource from the PowerShell Gallery

1. From the PowerShell ISE console, execute:

```
Install-Module -Name NetworkingDSC
```

2. You will be prompted to confirm downloading NetworkingDSC with a message, "You are installing the modules from an untrusted repository." Click or select yes when prompted.
3. Verify what new DSC Resources we have access to now that the module is installed

```
Get-DscResource -Module NetworkingDSC
Get-DscResource -Name Firewall | Select -ExpandProperty Properties
```

Create a DSC Configuration

1. Copy/paste the following DSC configuration into the PowerShell ISE of your Azure VM named DSC1

   From: C:\MyAzureProject\Azure Superpowers\Lab - PowerShell DSC\DemoConfig.ps1

```
configuration DemoConfig {

    Import-DscResource -ModuleName NetworkingDSC
    Import-DscResource -ModuleName PSDesiredStateConfiguration

    node 'localhost' {

        Firewall 'FirewallSettingOne' {
            Name      = 'Demo Block Rule'
            Action    = 'Block'
            LocalPort = ‘5000’
            Enabled   = 'True'
            Direction = 'Inbound'
            Profile   = 'Any'
            Protocol  = 'TCP'
        }

        Registry 'ConsentPromptBehaviorAdmin' {
            Ensure    = 'Present'
```

```
            Key       =
'HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Policies\System'
            ValueName = 'ConsentPromptBehaviorAdmin'
            ValueType = 'Dword'
            ValueData = '2'
        }
        Registry 'PromptOnSecureDesktop' {
            Ensure    = 'Present'
            Key       =
'HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Policies\System'
            ValueName = 'PromptOnSecureDesktop'
            ValueType = 'Dword'
            ValueData = '1'
        }
    }
}
```

2. Execute the above code. This will not perform any immediate action, but will create the DSC 'configuration' inside the memory of the PowerShell session.

3. In order to create a DSC MOF file, execute the following command on the ISE command prompt:

```
DemoConfig
```

4. If everything was successful, you will now have a DSC MOF file created based on the configuration above located in a subfolder (called DemoConfig) of your current folder. Your MOF file will be called localhost.mof

Exercise - Test the DSC Configuration Locally

Test the DSC Configuration Locally

1. Execute the following PowerShell to test your localhost.mof file against DSC1

```
Test-DscConfiguration -Path .\DemoConfig
```

2. Based on the output of the above command, we can see that some resources are in desired state and some resources are not in desired state, but PowerShell is hiding those answers in an attempt to display well on the screen. Run the following PowerShell commands to see the full answers:

```
1.      Test-DscConfiguration -path .\DemoConfig | Select -ExpandProperty
ResourcesInDesiredState
```

```
1.      Test-DscConfiguration -path .\DemoConfig | Select -ExpandProperty
ResourcesNotInDesiredState
```

3. Expected output from the commands listed above is shown below:

```
PS C:\windows\system32> Test-DscConfiguration -Path .\DemoConfig

PSComputerName  ResourcesInDesiredState        ResourcesNotInDesiredState     InDesiredState
--------------  ------------------------        --------------------------     --------------
localhost       {[Registry]PromptOnSecureDe... {[Firewall]FirewallSettingO... False


PS C:\windows\system32> Test-DscConfiguration -Path .\DemoConfig | Select -ExpandProperty ResourcesInDesiredState


ConfigurationName     : DemoConfig
DependsOn             :
ModuleName            : PSDesiredStateConfiguration
ModuleVersion         : 1.1
PsDscRunAsCredential  :
ResourceId            : [Registry]PromptOnSecureDesktop
SourceInfo            : ::26::9::Registry
DurationInSeconds     : 0.062
Error                 :
FinalState            :
InDesiredState        : True
InitialState          :
InstanceName          : PromptOnSecureDesktop
RebootRequested       : False
ResourceName          : Registry
StartDate             : 12/5/2018 4:10:13 PM
PSComputerName        : localhost



PS C:\windows\system32> Test-DscConfiguration -Path .\DemoConfig | Select -ExpandProperty ResourcesNotInDesiredState


ConfigurationName     : DemoConfig
DependsOn             :
ModuleName            : NetworkingDsc
ModuleVersion         : 6.2.0.0
PsDscRunAsCredential  :
ResourceId            : [Firewall]FirewallSettingOne
SourceInfo            : ::8::9::Firewall
DurationInSeconds     : 0.142
Error                 :
FinalState            :
InDesiredState        : False
InitialState          :
InstanceName          : FirewallSettingOne
RebootRequested       : False
ResourceName          : Firewall
StartDate             : 12/5/2018 4:10:21 PM
PSComputerName        : localhost

ConfigurationName     : DemoConfig
DependsOn             :
ModuleName            : PSDesiredStateConfiguration
ModuleVersion         : 1.1
PsDscRunAsCredential  :
ResourceId            : [Registry]ConsentPromptBehaviorAdmin
SourceInfo            : ::18::9::Registry
DurationInSeconds     : 0.061
Error                 :
FinalState            :
InDesiredState        : False
InitialState          :
InstanceName          : ConsentPromptBehaviorAdmin
RebootRequested       : False
ResourceName          : Registry
StartDate             : 12/5/2018 4:10:21 PM
PSComputerName        : localhost
```

4. Normally, you would run Start-DscConfiguration to apply a configuration to a system to bring it into the desired state. Instead, in the next exercise we will take the PowerShell configuration file and package it up for Azure to deploy.

Exercise - Create an Azure Compatible DSC package

Create the DSC Package (.zip)

To create an Azure compatible DSC .zip file you need 2 things. You need your PowerShell configuration as a .ps1 file, and you need all the modules you used in the Import-DSCResource section of the configuration.

1. Save your DSC configuration as DemoConfig.ps1 onto the desktop of the DSC1 VM. This will make it easier to find later.

2. From File Explorer on the DSC1 VM, navigate to where PowerShell keeps its modules (C:\Program Files\WindowsPowerShell\modules) and locate the NetworkingDSC folder. Copy the NetworkingDSC folder to where your .ps1 file is (desktop).

3. On the desktop, highlight both the DemoConfig.ps1 and NetworkingDSC folder. Right-click on the DemoConfig.ps1 and select 'Send To-> Compressed (Zipped) Folder'

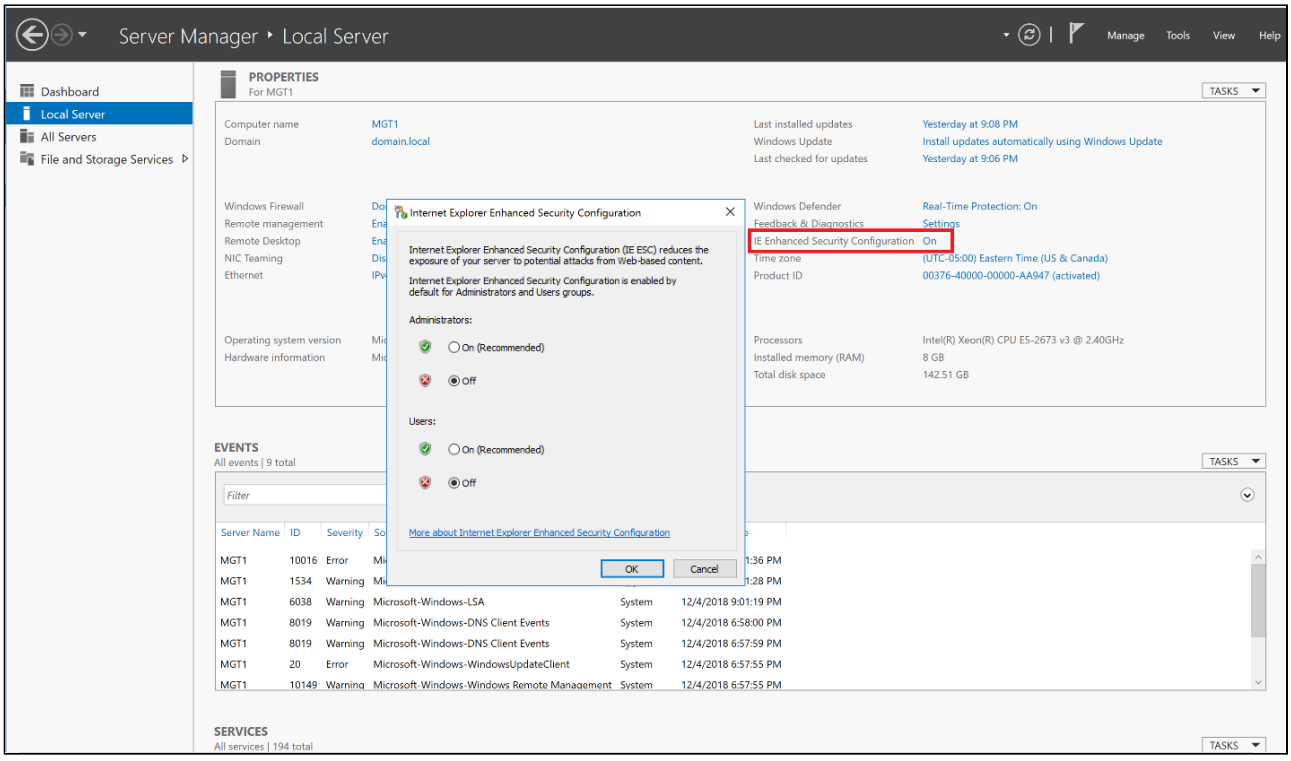
4. Rename the newly created zip file to DemoConfigZip.zip

5. If you created the DemoConfigZip.zip correctly, you should be able to double-click it and you will see both the .ps1 and the folder at the root of the zip file. If you created the zip file incorrectly, you will see a top level folder that contains files underneath it. If this is the case, delete the zip file and try again. Azure DSC expects the .ps1 and module folders in the root of the zip file.

Place the ZIP where Azure can retrieve it

In production environments, these steps can be repeated with a private Container and a Shared Access Signature (SAS) token to ensure the .zip files stay private. For this exercise, we're sticking to basics and will be using a publicly accessible URI to reduce the likelihood that something goes wrong.

1. Open Server Manager and turn off Internet Explorer Enhanced Security Configuration



2. Open the Azure portal from inside the DSC1 VM
3. In the Azure Portal, create a storage account. The details of the account don't matter. It should take about 1 minute for the storage account to deploy.
4. Access the storage account through the Azure portal and under 'Blob Service / Blobs' click the '+ Container' button and create a new container (any name) and configure the Public access level to **'Container (anonymous read access for containers and blobs)'**. Anonymous access is not the default and it's unlikely that we will use this for production configurations.
5. Click into your newly created Container. Click the 'Upload' button and select your files to upload.
6. Upload the DemoConfigZip.zip
7. Select the uploaded blob file and view the properties. Find the URL of the file and copy it.
8. Paste the value from 'URL' into the web browser **on your local workstation**. If the browser prompts you to download the file, you successfully have a public URL where anyone (especially Azure VMs) can download the .zip file and you can move to the next exercise.

Exercise - Deploy an ARM resource DSC extension

Login to Azure via PowerShell from your **LOCAL COMPUTER**

From a PowerShell console window in VS Code, login to Azure using either:

Azure Commercial:

```
Login-AzAccount
```

-or-

Azure Government:

```
Login-AzAccount -Environment AzureUSGovernment
```

And then login into Azure (as yourself) using the pop-up dialogue box (that may have popped-up under the current Window(s)).

```
Select-AzSubscription –Subscription '<Id>'
```

Deploy ARM DSC Resource

1. Review **C:\MyAzureProject\Azure Superpowers\Lab - PowerShell DSC\DSC.json**

```
1.      {
2.          "$schema": "https://schema.management.azure.com/schemas/2015-01-
01/deploymentTemplate.json#",
3.          "contentVersion": "1.0.0.0",
4.          "parameters": {
5.              "zipFileURI": {
6.                  "type": "string"
7.              },
8.              "FileName": {
9.                  "type": "string"
10.             },
11.             "ConfigName": {
12.                 "type": "string"
13.             },
14.             "VMname": {
15.                 "type": "string"
16.             }
17.         },
18.         "variables": {},
19.         "resources": [
```

```
            {
                "type": "Microsoft.Compute/virtualMachines/extensions",
                "name": "[concat(parameters('VMname'), '/DSC')]",
                "apiVersion": "2015-06-15",
                "location": "[resourceGroup().location]",
                "properties": {
                    "publisher": "Microsoft.Powershell",
                    "type": "DSC",
                    "typeHandlerVersion": "2.19",
                    "autoUpgradeMinorVersion": true,
                    "settings": {
                        "ModulesUrl": "[parameters('zipFileURI')]",
                        "ConfigurationFunction": "
[concat(parameters('FileName'), '\\', parameters('ConfigName'))]"
                    }
                }
            }
        ]
    }
```

2. Change directory over to C:\MyAzureProject\Azure Superpowers\Lab - PowerShell DSC\

3. In PowerShell, execute the following PowerShell to deploy the DSC resource, substituting the <VALUES> for your own environment.

```
    $params = @{
        VMName            = 'DSC1'
        ResourceGroupName = '<RESOURCEGROUPNAME>'
        ZipFileURI        =
'https://<ACCT>.blob.core.windows.net/<CONTAINER>/DemoConfigZip.zip'
        FileName          = 'DemoConfig.ps1'
        TemplateFile      = 'DSC.json'
        ConfigName        = 'DemoConfig'
        Verbose           = $true
        }
    New-AzResourceGroupDeployment @params
```

Once the deployment is complete, verify that the deployment was successful

1. RDP to DSC1 and verify the registry keys

   a. HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Policies\System

   ```
   i.  ConsentPromptBehaviorAdmin = Dword 2

   ii. PromptOnSecureDesktop = Dword 1
   ```

2. Verify the firewall (Start->Run->wf.msc)

   a. Click on Inbound Rules. You should see a new rule has been created, "Demo Block Rule"

   ```
   i. Action = Block

   ii. LocalPort = 5000

   iii. Enabled = Yes

   iv. Direction = Inbound

   v. Profile = All

   vi. Protocol = TCP
   ```

3. Run the following PowerShell on DSC1:

```
Test-DscConfiguration -Detailed
```

All resources should now be listed as being in the desired state.

Troubleshooting

If you receive a message that states your ZipFileURI cannot be found, make sure you are executing commands from within the correct directory that contains your DSC.json file

Pay attention to filenames

Lab 9 - Storage Accounts and SAS Tokens

Lab Description	**This lab will teach you how to upload files to Azure Storage Accounts and secure them by protecting the files with Secure Access Tokens (SAS) tokens**
Glossary of Terms	*sas* – Secure Access Token. This is a string that you append to the URL you are accessing which is an encoded/signed string that describes your rights. Read, write, Starting When, Expiring When, Can you see Containers, etc.
	uri – Uniform Resource Identifier. https://portal.azure.com is a URL. https://portal.azure.com/Default.html is a URI. URI refers to the full path to the resource, not just the DNS name of the URL.
Estimated Time to Complete	30 minutes
Key Takeaways	1. Upload files to Azure Storage
	2. Generate SAS Token
	3. Access private Azure Storage files with URI + SAS Token
Author	Wes Adams

Exercise - Create a storage account

Prepare Azure Resource Group

1. Launch VS Code
2. Navigate to the integrated terminal by creating a new PS1 file
3. Use the following commands to connect to Azure

Azure Commercial:

```
Login-AzAccount
```

-or-

Azure Government:

```
Login-AzAccount -Environment AzureUSGovernment
```

And then login into Azure (as yourself) using the pop-up dialogue box (that may have popped-up under the current Window(s)).

```
Select-AzSubscription –Subscription '<Id>'
```

4. Create a resource group that will contain the storage account used in this lab by running the following:

```
New-AzResourceGroup -Name 'StorageExample-<YOURALIAS>' -Location EastUS

#OR

New-AzResourceGroup -Name 'StorageExample-<YOURALIAS>' -Location USGovVirginia
```

Create an Azure Storage Account

1. Storage account names must be universally unique, **lower case** and alphanumeric (3-24 characters). In this example, the storage account name Aliasblobstorage is used. When this is listed, replace Alias with your alias, for example if your alias was bsmith, your storage account would be bsmithblobstorage.

2. Create a BlobStorage account using the following command:

```
$params = @{
    ResourceGroupName = 'StorageExample-<YOURALIAS>'
    AccountName       = '<ALIAS>blobstorage'
    Location          = 'eastus' #OR 'usgovvirginia'
    SkuName           = 'Standard_LRS'
}
New-AzStorageAccount @params
```

3. Open Microsoft Azure Storage Explorer and connect using your Azure Credentials

4. Notice that <ALIAS>blobstorage is listed as a storage account, expand this to show the types of storage resources available:

   a. Blob Containers

   b. File Shares

   c. Queues

   d. Tables

5. To better understand the differences between Azure Blobs, Azure Files, and Azure Disks, review: https://docs.microsoft.com/en-us/azure/storage/common/storage-decide-blobs-files-disks?toc=%2Fazure%2Fstorage%2Ffiles%2Ftoc.json

Exercise - Create a blob container

Understanding Storage Account Keys

1. A Storage Account Key is like the root admin password for your Storage Account. This information should be guarded and not distributed. Each storage account comes with two storage account keys. The following PowerShell will display the account keys for the storage account we just created:

```
$params = @{
    ResourceGroupName = 'StorageExample-<YOURALIAS>'
    Name              = '<ALIAS>blobstorage'
}
Get-AzStorageAccountKey @params
```

2. We will need the account key to create a SAS Token. A SAS Token (shared access signature) provides delegated access to resources in your storage account. You can assign full or reduced permissions to a SAS Token. SAS Tokens can also have expirations set and can more easily be revoked than a Storage Account Key. Use the following PowerShell to capture an account key into a PowerShell variable:

```
$params = @{
    ResourceGroupName = 'StorageExample-<YOURALIAS>'
    Name              = '<ALIAS>blobstorage'
}
$storageAccountKey = (Get-AzStorageAccountKey @params).Value[0]
```

3. We will need to create an Azure Storage Context. The context describes the Azure Storage Account and is needed to create a Storage Container in the next step. Uncomment the Environment parameter if you are working with something other than Azure Commercial, such as Microsoft Azure Government.

```
$params = @{
    StorageAccountName = '<ALIAS>blobstorage'
    StorageAccountKey  = $storageAccountKey
    #Environment = 'AzureUSGovernment'
}
$storageAccountContext = New-AzStorageContext @params
```

Create a Storage Container

1. Use the following PowerShell to create a Container in the Storage Account that you created previously in **Exercise- Create a storage account**.

```
$params = @{
    Name    = 'sasexample'
    Context = $storageAccountContext
}
New-AzStorageContainer @params
```

2. If you refresh the storage account in Azure Storage Explorer, you will now see the container "sasexample" listed under Blob Containers. Notice that it currently contains no files.

Create a Storage Container Policy

1. Create a Policy that will define permissions that will be associated with the SAS Token. Use the following PowerShell to complete this task with Permission defined for Read, Write, Delete, List

```
$params = @{
    Container  = 'sasexample'
    Policy     = 'policyexample'
    Permission = 'rwdl'
    ExpiryTime = (Get-Date).AddDays(2)
    Context    = $storageAccountContext
}
New-AzStorageContainerStoredAccessPolicy @params
```

2. If you refresh the Blob Container sasexample in Azure Storage Explorer, you can view the created Storage Container Policy by right clicking on sasexample and clicking **Manage Access Policies**.

3. Notice that the policy has the permissions to Read, Write, Delete, and List. A subset of these permissions could also be set, depending on the need of the application or scripts that are accessing the storage.

Create a Shared Access Signature (SAS) Token

1. Create a SAS Token (to be used later) based on the Policy and Storage Context defined earlier in this example using the following PowerShell:

```
$params = @{
    Name    = 'sasexample'
    Policy  = 'policyexample'
    Context = $storageAccountContext
}
$sasToken = New-AzStorageContainerSASToken @params
Write-Host "SAS Token: $sasToken"
```

2. The SAS Token, if correct, should be in a similar format:
   ?sv=2018-03-28&si=policyexample&sr=c&sig=6cVRHdm3rGFEBCa6sSNhmvDKAWZxruM4EQX8wpv2NS4%3D

Exercise - Create Test Files

Create Test Files

1. Create a directory in the root of C: called "SASExample"
2. Create a text file called "File1.txt"
3. Create a text file called "File2.txt"
4. Create a text file called "File3.txt"

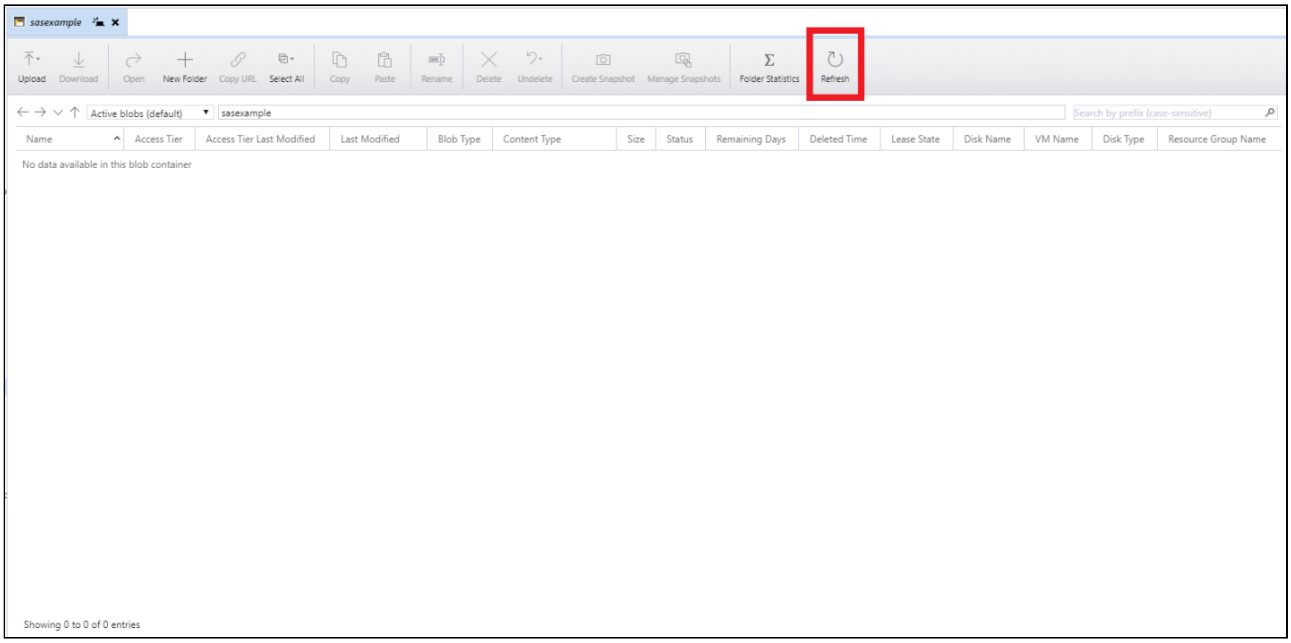
Exercise - Upload one text file using the portal

Connect to the Azure Portal

1. Login to the Azure Portal
2. Select **Resource Groups**
3. Select the Resource Group **StorageExample-<YOURALIAS>**
4. Select the StorageAccount **<ALIAS>blobstorage**
5. Under Services, select **Blobs**
6. Select the Storage Container **sasexample**
7. Click **Upload**
8. Select "**C:\sasexample\file1.txt**"
9. Click **Upload**

Verify the File Upload in Azure Storage Explorer

1. From Azure Storage Explorer, select the storage container **sasexample**
2. **Refresh** the files within the **sasexample** storage container (Note the location of the Refresh button in the screenshot below)



3. Notice **File1.txt** is visible from Azure Storage Explorer

Exercise - Upload a File Using Azure Storage Explorer

Upload File

1. From Azure Storage Explorer, Select the storage container **sasexample**
2. Click **Upload**
3. Click **Upload Files...**
4. Under "Files" select **C:\sasexample\file2.txt**
5. Click **Upload**

Verify the File Upload in Azure Storage Explorer

1. From Azure Storage Explorer, select the storage container **sasexample**
2. **Refresh** the files within the **sasexample** storage container
3. Notice **File2.txt** is visible from Azure Storage Explorer

Exercise - Upload one file using PowerShell

Create a Client Context

1. In order to upload files to our storage account via PowerShell, we will need to create a storage context, which describes our Storage Account and the SAS Token we will use for the upload.

2. Use the following PowerShell to create the client context:

```
$params = @{
    sasToken           = $sasToken
    storageAccountName = '<ALIAS>blobstorage'
    #Environment = 'AzureUSGovernment'
}
$clientContext = New-AzStorageContext @params
```

Test the Context by listing all files in the storage container

1. Run the following PowerShell to list all files in the Storage Container:

```
$params = @{
    Container = 'sasexample'
    Context   = $clientContext
}
Get-AzStorageBlob @params
```

2. The files we have uploaded in previous steps should be outputted.

Upload one file using PowerShell

1. Run the following PowerShell to upload file3.txt to our Storage Container:

```
$params = @{
    Container = 'sasexample'
    File      = 'c:\sasexample\File3.txt'
    Context   = $clientContext
}
Set-AzStorageBlobContent @params
```

2. File3.txt is now uploaded to the storage container

Verify the File Upload in Azure Storage Explorer

1. From Azure Storage Explorer, select the storage container **sasexample**
2. **Refresh** the files within the **sasexample** storage container
3. Notice **File3**.**txt** is visible from Azure Storage Explorer
4. Notice that the **File3.txt** has a different ContentType from **File1.txt** and **File2.txt.**
   Note: ContentType relates to how a web browser will interact with the file (Download, Open, Display) and can be changed if desired.
5. If desired, you could ensure that File3.txt be uploaded with a certain ContentType by including the following in your splatted parameters

```
Properties = @{"ContentType" = "text/plain"}
```

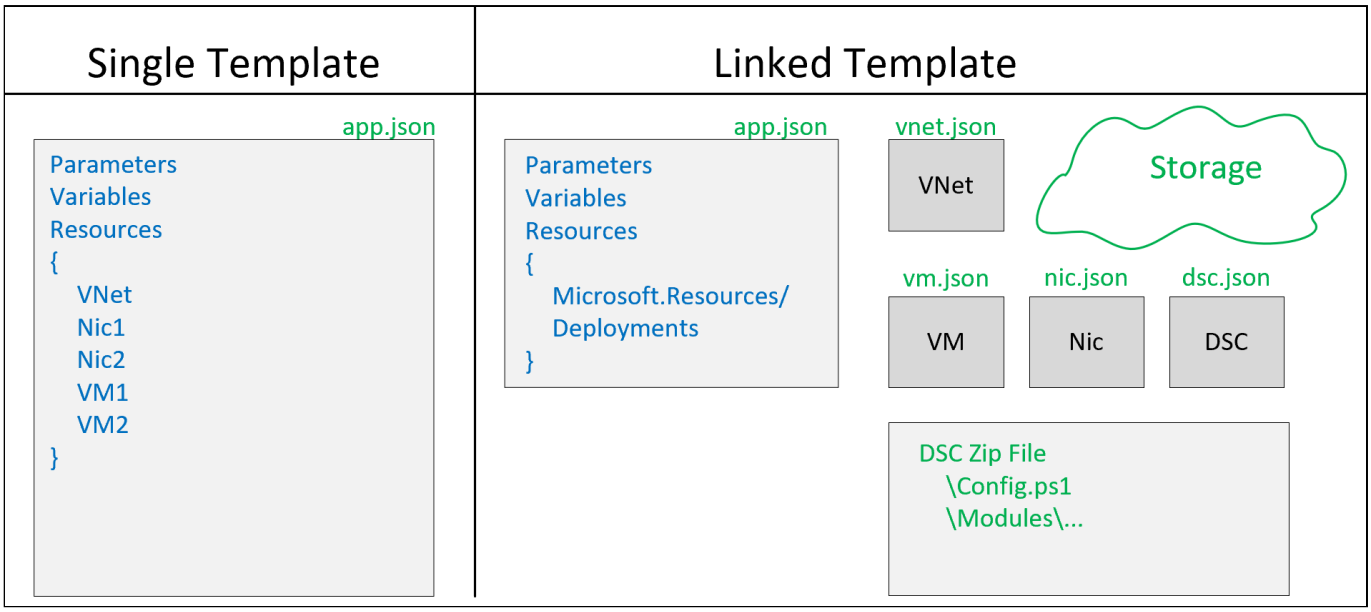
Troubleshooting

If you use the search option within Azure Storage Explorer, you may not see the expected items below, such as Blob Containers, File Shares, Queues, and Tables. The search option will only pull up instances that are a direct match, such as the storage account name itself.

Lab 10 - ARM Templates (Advanced)

Lab Description	**In this lab, you will explore some advanced usage of Azure Resource Manager (ARM) templates to deploy resources into a Resource Group.**
	To deploy a solution, you can use either a single template or a main template with many related sub-templates. The related template can be either a separate file that is linked to from the main template, or a template that is nested directly within the main template.
	Linked templates enable you to break down the solution into targeted components, and facilitate code reuse. When using linked template, you create a main template that receives the all parameter values needed for the deployment. The main template contains all the needed values that are expected by the linked templates and passes those values on as needed.
Glossary of Terms	*ARM Template* – An ARM Template is a JavaScript Object Notation (JSON) file that defines one or more resources to deploy to an Azure resource group. It also defines the dependencies between the deployed resources. The template can be used to deploy the resources consistently and repeatedly.
	PowerShell DSC extension – The Azure PowerShell DSC extension uses the Azure VM Agent framework to deliver, enact, and report on DSC configurations running on Azure VMs. The DSC extension accepts a configuration document and a set of parameters. The extension requires that the DSC configuration and any required DSC resource modules be combined into a ZIP file that is made accessible to the Azure VM at the time the extension is called.
Estimated Time to Complete	60 minutes
Key Takeaways	1. ARM templates allow for a lot of customization and flexibility
	2. Linked templates should be designed as modular building blocks that can be reused across multiple solutions, and can be deployed independently if desired
	3. The PowerShell DSC extension can be called multiple times in a template, as long as the same name is always used when defining each Microsoft.Compute/virtualMachines/extensions DSC resource. Following this strategy, along with the Microsoft.Resources/deployments resource allows you to create modular components, executed in a desired order using the ARM dependsOn element.
Author	Ralph Kyttle

Single Templates vs Linked Templates



Exercise - Review an existing Advanced ARM Template

Task 1: Review the main ARM template

Review the following ARM template in VS Code (Ensure that you are viewing the master branch)

C:\MyAzureProject\Azure Superpowers\Lab - ARM Templates Advanced\Exercise 1\adLab.json

This template showcases the following:

- A top level main template that describes the entire deployment
    - adLab.json
- Modular sub-templates that act as building blocks for the deployment (Microsoft.Resources/deployments resource)
    - Defined in-line, directly in the main template
        - Lines 56 - 102
        - Lines 150 - 194
    - Linked templates that are called via their URI, stored on either public or private storage
        - Lines 103 - 149
        - Lines 195 - 244
        - Lines 245 - 270
        - Lines 271 - 302
        - Lines 303 - 328
        - Lines 329 - 370
- Multiple calls of the PowerShell DSC extension
    - DSC to configure VM settings - includes security lockdowns typically done with GPO
    - DSC to install software
    - DSC using custom resources
    - DSC extensions created as modular components nested in their own individual ARM templates, allowing DSC extension resources to all have the same name which allows multiple configurations to be applied to a system during deployment, but also supporting DependsOn logic as the ARM templates can be uniquely named
- DSC storage locations
    - Public - GitHub

        - Line 171
    - Private - Azure Storage Account and SAS tokens
        - SAS tokens provided via template variables
            - Lines 110, 202, 252, 278, 310, 337
        - SAS tokens provided directly in the path to the file
            - Lines 139, 142, 234, 237
- DSC protected settings
    - How to pass in values such as passwords to the DSC extension. The benefit of doing it this way is that the password will never be able to be retrieved through looking at the extension properties, and it will be encrypted on disk (which is handled by the extension). This helps us keep our passwords secure and doesn't create the risk of storing passwords in plain text and easily retrievable formats.
        - Line 183
- Parameters
    - Parameters passed from the main template into sub-templates, and from sub-templates into DSC
    - Default parameter values
        - Lines 13, 24, 28
    - Allowed parameter values (Used to limit accepted values, or provide a list of valid values)
        - Line 14
    - Metadata descriptions in the absence of comments
        - Line 19, 33, 45
    - Special parameters to handle DSC configuration caching by the ARM Engine
        - Lines 42, 132, 227, 294, 320
    - VM name
        - Lines 256, 282, 314
    - VM size
        - Lines 135, 230, 353
    - IP address
        - Lines 123, 215
    - Reference OS image

        - Lines 209-238 of the VM sub template
    - Timezone
        - Line 266 of the VM sub template
- Passing output from the deployment of one template as input to another template
    - Vnet is passed from one deployment to the next
        - Lines 94-99, Line 118, Line 210
- Conditional logic
    - condition element
    - Example: Allow someone to decide during deployment whether to create a resource
    - Line 330
- DependsOn
    - Waiting for an Azure resource to deployed before kicking off the deployment of another Azure resource
        - Lines 146, 191, 241, 325
    - Ensuring that a DSC configuration running on one system completes before kicking off a DSC configuration on another system
        - Lines 267, 299, 367
- Custom Script Extension running a PowerShell script on a VM
    - Lines 245-270
- Resource Manager template functions
    - concat
        - Combines multiple arrays and returns the concatenated array, or combines multiple string values and returns the concatenated string
    - equals
        - Checks whether two values equal each other. Used in conditional logic.
    - deployment
        - Returns information about the current deployment operation. You can use deployment to link to another template based on the URI of the parent template.
    - parameters

        - Returns a parameter value. The specified parameter name must be defined in the parameters section of the template. This allows you to pass parameters from the top-level main template down to sub-templates.
    - variables
        - Returns the value of variable. The specified variable name must be defined in the variables section of the template. This allows you to pass variables from the top-level main template down to sub-templates.
    - copyIndex
        - Returns the index of an iteration loop. Along with the copy element, copyIndex allows someone to decide during deployment to create one or more instances of a resource.
    - reference
        - Returns an object representing a resource's runtime state. This is used when passing output from the deployment of one template as input to another template
    - resourceGroup
        - Returns an object that represents the current resource group. A common use of the resourceGroup function is to create resources in the same location as the resource group.
    - resourceId
        - Returns the unique identifier of a resource. You can use this function when the resource name is ambiguous or not provisioned within the same template. It is typically used with a template's outputs section so a resource can be passed onto another template for further work, or as part of a separate template directly referencing a resource that was deployed as part of a different template or deployment.
    - uri
        - Creates an absolute URI by combining the baseUri and the relativeUri string. Used along with the deployment function to link to another template based on the URI of the parent template.
- PublicIP creation - selectable per VM with conditional logic
- Domain Controller deployment
- Creation of N number of servers
- Security settings applied via DSC
- VM Domain join

Task 2: Review the linked ARM templates

Template name: vm.json

C:\MyAzureProject\Azure Superpowers\Lab - ARM Templates Advanced\Exercise 1\vm.json

Used in the following deployments:

- createDC1
- createMGT1

Stored in the location shown below

https://azuresuperpowers.blob.core.windows.net/arm/Exercise 1/vm.json?st=2018-07-31T20%3A19%3A15Z&se=2020-12-31T20%3A19%3A00Z&sp=rl&sv=2018-03-28&sr=b&sig=OzFQnQCyI5t8gMAJ8DEVeZm8DjAmi3wyXetCakNKg8k%3D

Template name: AddRSATTools.json

C:\MyAzureProject\Azure Superpowers\Lab - ARM Templates Advanced\Exercise 1\resourceTemplates\AddRSATTools\AddRSATTools.json

Used in the following deployments:

- AddRSATToolsToMGT1

Stored in the location shown below

https://azuresuperpowers.blob.core.windows.net/arm/Exercise 1/resourceTemplates/AddRSATTools/AddRSATTools.json?st=2018-11-05T02%3A04%3A47Z&se=2021-01-01T02%3A04%3A00Z&sp=rl&sv=2018-03-28&sr=b&sig=NAVnJS%2BtjU3iawlruLdR3dp%2BFGihAL%2BEgfCvcw0Xb%2Bk%3D

Template name: multiplevms.json

C:\MyAzureProject\Azure Superpowers\Lab - ARM Templates Advanced\Exercise 1\multiplevms.json

Used in the following deployments:

- CreateWorkerNodes

Stored in the location shown below

https://azuresuperpowers.blob.core.windows.net/arm/Exercise 1/multiplevms.json?st=2018-07-31T20%3A52%3A18Z&se=2020-12-31T20%3A52%3A00Z&sp=rl&sv=2018-03-28&sr=b&sig=OdLDqGLuFLzDoyY8PwS4Ci4LUPeFxfAfwdbkoFXCo2k%3D

Exercise - Edit the existing Advanced ARM Template and deployment

Task 1: Redeploy the main ARM template with an additional worker VM

Execute the same PowerShell based deployment that you performed in the AD Super Lab Deployment, except this time specify **numberOfvmWorkerInstances: 4** when you are prompted

View the deployment status in the PowerShell output and in the Azure Portal. You should see a 4th node being built.

Task 2: Login to MGT1 using RDP and its public IP and verify environment settings

- Login to MGT1 **(DOMAIN\User)** using RDP and its public IP. The name of the domain that was created is domain.local, so you will login using DOMAIN\User, replacing User with the name of the user you created.
- Open Start Menu, type Administrative Tools and click the button to launch
- Verify that you have access to Active Directory Users and Computers, and DNS. These RSAT tools do not come installed by default on a Windows Server. They were added as part of the deployment of MGT1 by using DSC.
    - adLab.json Lines 271-296 call the sub template to add the RSAT tools. Inside this sub template is a call to the DSC extension, which will then add the Windows features to the system
- Open Active Directory Users and Computers
- Select the Computers container. You should see MGT1, and your worker nodes listed here as well. Depending on whether the previous task completed or not, you should see either 3 or 4 worker nodes.

Task 3: Instructor led exercise - empty the contents of your resource group by deploying an empty template with complete mode

- We will review the file shown below

  C:\MyAzureProject\Azure Superpowers\Lab - ARM Templates Advanced\emptyResourceGroup.ps1

Troubleshooting

Lab 11 - Azure DevOps Service Connections

Lab Description	**In this lab you will create a new Service Connection in Azure DevOps which will connect to Azure and allow build/release steps in Azure DevOps to interact with and control an Azure subscription.**
Glossary of Terms	*Service Connection* - An Azure DevOps a service connection holds the connection info (subscription, etc) as well as the authentication info (service principal ID, tenant ID, and password) to an external entity (like Azure) which can be used in automated Azure DevOps build steps to automatically perform actions on behalf of the project. This keeps authentication out of source controlled code.
Estimated Time to Complete	30 minutes
Key Takeaways	1. In order for Azure DevOps to interact with Azure you need to create a Service Connection in your project
	2. Any PowerShell scripts you write or use do not need to login or specify their Azure subscription target, Azure DevOps will handle all that prior to running your scripts in Azure
Author	Keith Hitchcock

Exercise - Create a Service Principal

Complete previous lab - Azure AD and Service Principals

The prior Azure AD and Service Principal lab walks you through creating a service principal in Azure AD and giving it access to an Azure subscription.

Before proceeding to the next step, make sure that you have:

- a service principal created
- a resource group created
- the service principal has rights to that resource group (owner)

Exercise - Create a Service Connection in Azure DevOps

Login to visualstudio.com

1. Open a web browser and navigate to https://www.visualstudio.com.
2. In the upper-right hand corner, you should be automatically logged in with your Microsoft Account. If not, please login.
3. Once logged in, please click the logged in user in the upper-right hand corner and select 'Visual Studio Profile' from the expanded menu.

Access the Azure DevOps Organization that holds your project

1. Under the header 'Azure DevOps Organizations' you will see all the organizations that you are involved with (marked as either Owner or Member)
2. Click the '>' arrow key next to an organization to expand out the listing of projects that are contained within that organization.
3. Click the link of the project name you wish to launch.
4. (If you have no accounts/projects, please complete the prior lab - Azure DevOps)

Initialize the default git repository

1. Click on Repos
2. You may see a section to 'initialize with a README or gitignore'. If you do, click the 'initialize' button. The reason to initialize the repository from here is because an empty repository cannot be cloned or used in a build.

Create a new Service Connection

1. In the bottom left corner of the project click the 'Project Settings' button
2. Under 'Pipelines' click the 'Service connections' button
3. Click the '+ New Service Connection' button and select 'Azure Resource Manager' from the drop-down menu.
4. From the 'Add Azure Resource Manager Service Connection' dialogue window, **click the hyperlink 'use the full version of the service connection dialog**.' If you do not click this hyperlink, Azure DevOps will attempt to use your currently logged in Azure DevOps credentials to create a new Service Principal and give it rights to the subscription, instead of associating an existing one.

5. From the expanded 'Add Azure Resource Manager Service Connection' dialog window please enter the requested information. **Please remove any values that have been auto-populated. Review each value**:

```
a.  Connection Name - This name can be anything. A meaningful name might
include info about the purpose of the service principal (i.e. 'DEV
Connection' or 'AZGOV LIMITED', or 'AIRS Demo'

b.  Environment - select 'AzureCloud' for commercial Azure,
'AzureUSGovernment' for MAG, or whichever environment the Azure subscription
you want Azure DevOps to connect to exists in.

c.  Subscription ID - This value is the subscription ID of the Azure
subscription you wish to connect to. This information can be obtained from
the Get-AzSubscription cmdlet in PowerShell or from the Azure Portal.

d.  Subscription Name - This value is the Name of the Azure subscription you
wish to connect to. It matters that the name entered here is the name that
is associated with Subscription ID in the last field. This information can
be obtained from the Get-AzSubscription cmdlet in PowerShell or through the
Azure Portal.

e.  Service Principal Client ID – This is also known as the ‘AppID’ or
‘Application ID’.  This information can be obtained from the Get-
AzADServicePrincipal -DisplayName <SPNAME> cmdlet in PowerShell or through
the Azure Portal in Azure Active Directory under ‘App Registrations’.

f.  Service Principal Key - This is the Service Principal's password.

g.  Tenant ID - The ID of the Azure Active Directory tenant that the Service
Principal was created in
```

Click the Verify Connection hyperlink

The information must successfully verify before proceeding to the next exercise. If it does not, doublecheck the information listed above, especially Environment. If you believe all the information above is correct but you still cannot verify, attempt to login to Azure via PowerShell with the information you have provided (this procedure is described in the Azure AD and Service Principals Lab). Please note that a service principal must have some rights (any) in the Azure subscription in order to be successfully verified.

Exercise - Verify Service Connection

Create a new Azure DevOps Build

1. From the Azure DevOps web site, select the 'Pipelines -> Builds' button from the left of the page
2. Click the 'New Pipeline' button to create a new Build
3. **You will need to click the link, "Use the classic editor" for the next steps to work**
4. Under 'Select a source' keep the default settings and click the 'Continue' button
5. Under 'Select a template' click the 'Empty job' hyperlink
6. Under 'Agent Job 1' click the '+' button to add a task to Agent Job 1
7. Search for 'Azure PowerShell'
8. On the right-hand side, hover over 'Azure PowerShell and click the 'Add' button
9. On the left-hand side, select the new task which 'some settings need attention'
10. On the right side, change the Display Name to 'Run Azure PowerShell'
11. Change the Task Version to the following: **4.* (preview)**
12. Change 'Azure Subscription' to the Azure Service Connection created in the previous exercise which will be listed under "Available Azure service connections"
13. Click the 'In-line Script' radial button
14. At the bottom of the 'In-line Script' field, add the PowerShell cmdlet:

```
Get-AzResourceGroup
```

15. Under 'Preferred Azure PowerShell Version' set the value to 1.0.0
16. Click 'Save & Queue' at the top of the screen, and click 'Save & Queue'
17. In the 'Run pipeline' dialog window you can leave all options default and click the 'Save and run' button.
18. Wait for the build to find an available agent and connect to Azure, you can watch the output window scroll by.

19. Once the build completes and you receive your green check marks (may take a few minutes because of this free service tier) you can select the 'Run Azure PowerShell' section of the log that is currently displayed and it will display the output of the PowerShell cmdlet.

20. The output should include only the resource groups that your service principal has access to. You may have other resource groups in your subscription, but in this exercise we see that a service principal can only interact with resources that it has been granted permission to access.

Success!

If this worked correctly, Azure DevOps can now run any PowerShell command you give it (or script, or series of commands) without you having to provide credentials and login yourself. Azure DevOps will run any commands you want in any order you want using the Service Principal defined in the Service Connection. This now allows you to store PowerShell and JSON configs in a git database (and gain all those benefits) and then use Azure DevOps builds to execute those scripts/commands. Azure DevOps can also run the build on demand (as you did in this exercise by queuing it), on an automatic schedule, or automatically every time a change is made to the code repository.

Troubleshooting

Your Azure DevOps build is configured to utilize a hosted build agent. Because this is a hosted service it may take some time before the build begins, dependent on the other workloads being submitted to the hosted agents at the same time.

With Microsoft-hosted agents, maintenance and upgrades are taken care of for you. Each time you run a pipeline, you get a fresh virtual machine. The virtual machine is discarded after one use.

Lab 12 - Azure DevOps Prereqs

Lab Description	**In this lab, you will deploy and configure prerequisites that are needed to complete the subsequent labs on Azure DevOps Build and Release.**
Estimated Time to Complete	45 minutes
Author	Ralph Kyttle

Exercise - Azure DevOps Organization

Verify that your Azure DevOps Organization exists and you can login

1. Refer to Lab - Azure DevOps to create a new Azure DevOps Organization if you have not created one yet.
2. Open a web browser and navigate to https://dev.azure.com/organizationname
3. Verify that your Azure DevOps Organization exists and you can login

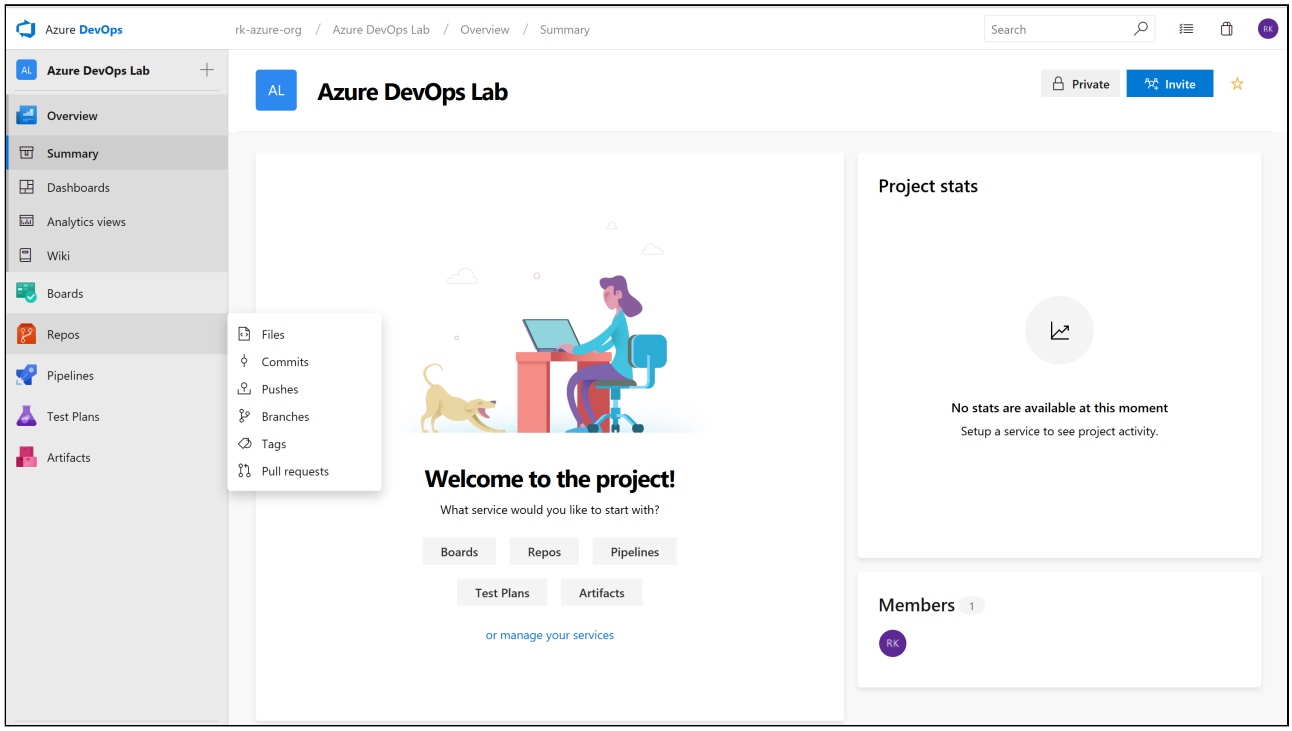
Exercise - Create a new Azure DevOps Project

Create a new Azure DevOps Project

1. From the homepage of https://dev.azure.com/organizationname click on **Create project**
2. Provide a name for your private project and click create. Project names must be unique per Azure DevOps Organization.

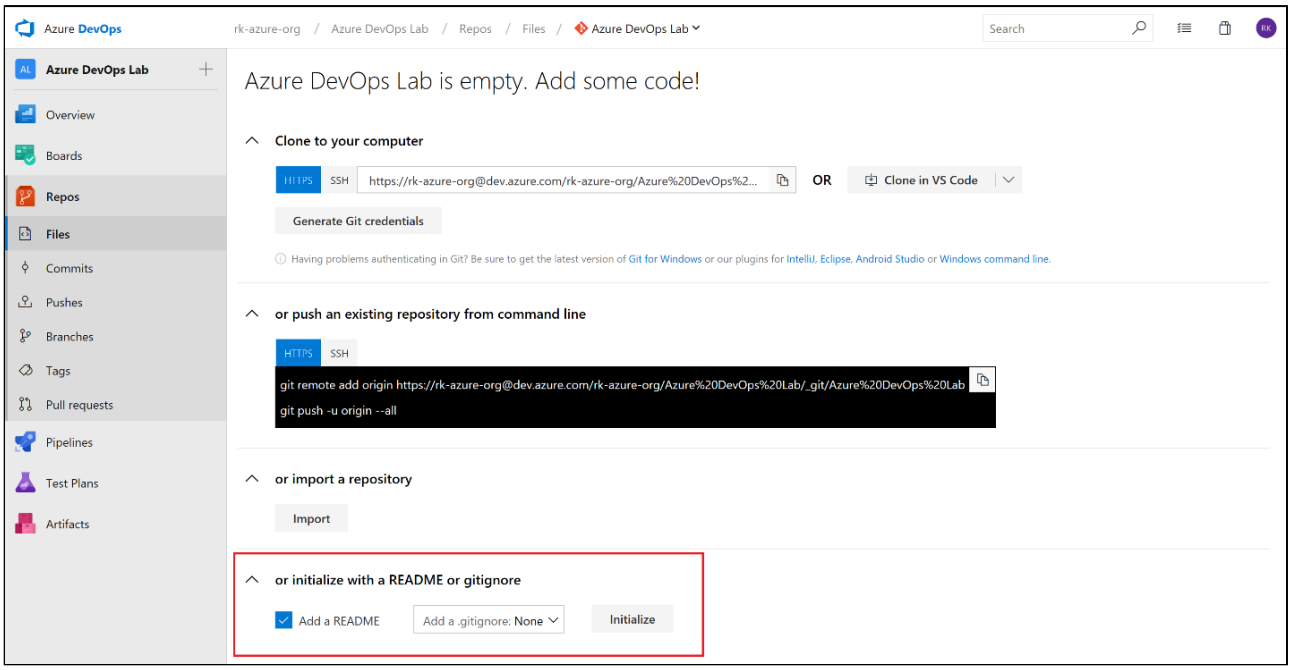
View the Project repository

1. Select the **Repos** tab



Initialize the Project repository

1. Click on the **Initialize** button to initialize the repository with a template README file



Clone git repository from Azure DevOps to your local system

1. From the 'Repos' section of the Project, in the upper right-hand corner there is a 'Clone' button which will allow you to copy the URL of the git repository (hosted on Azure DevOps) for this project. Click the copy button to copy the URL to your clipboard.

2. Launch the Visual Studio Code application your local system

3. Go to the top menu bar and select View -> Command Palette

4. In the command palette, type 'git clone' and press enter

5. When prompted for 'Repository URL' paste the value obtained from Step 1 into the command palette

6. Select a folder to store the new repository locally. Note that you are selecting the top level folder. A subfolder will be created with the repository name automatically. Because a repository is similar to a database, **do not** select a folder that is managed by technologies like OneDrive, Dropbox, etc. This can cause issues with both technologies fighting to track changes.

7. If prompted to authenticate, please enter credentials that are valid for your Azure DevOps organization

8. When Visual Studio Code prompts you to 'Open Repository', select that option. You should now see the repository and the default README.md file that you created by the step where you initialized the repository in Azure DevOps.

Exercise - Branches

Create a new dev branch in your Azure DevOps repo

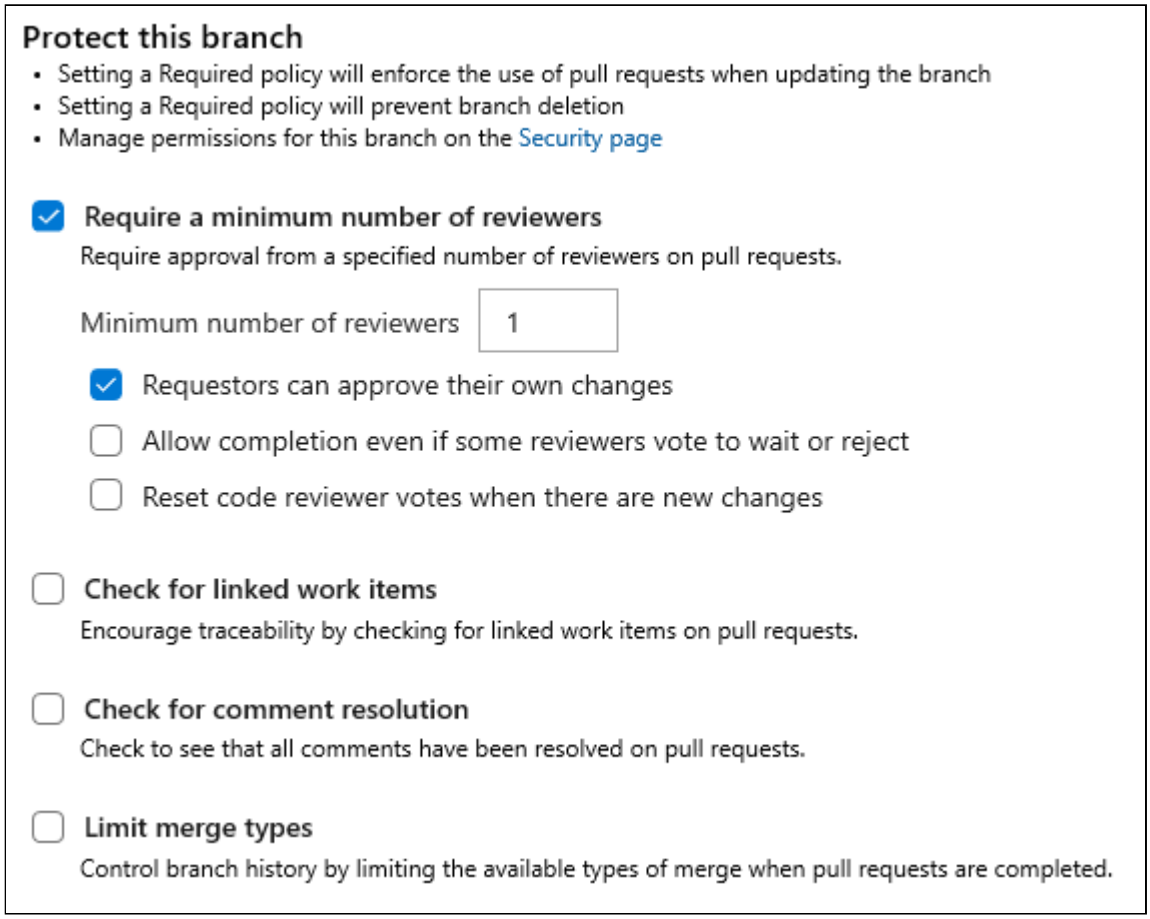
1. Select your internet browser, and head back to your Azure DevOps project
2. Under the Repos tab of your Azure DevOps project, select the Branches button
3. Click the New branch button
4. Create a new branch named dev. Base this new branch on the master branch.
5. Refer to **Exercise - Git Branch** in the Git lab for a detailed example of creating new branches

Exercise - Branch Policies

Create a branch policy on the master branch

The steps below were adapted from information found at the following url: https://docs.microsoft.com/en-us/vsts/repos/git/branch-policies

1. Locate your **master** branch in the Branches view. You can browse the list or you can search for your branch using the Search all branches box in the upper right.
2. Open the context menu for the branch by selecting the ellipsis (...) button. Select Branch policies from the context menu.
3. For this example, you will setup a very simple branch policy to ensure that code cannot be pushed directly to the **master** branch.
4. Set your branch policy to match the settings shown below and save your changes to apply your new policy configuration.



5. Notice how **Requestors can approve their own changes** is selected. If **Requestors can approve their own changes** is not selected, the creator of the pull request can still vote Approve on their pull request, but their vote won't count toward the Require a minimum number of reviewers policy.
6. Save your changes. Even a simple policy like the one you just created will enforce the use of pull requests when updating the branch and will prevent branch deletion. Refer to **Exercise - Branch policies** in the Git lab for a detailed example of creating new branch policies

Exercise - Download sample code and check files into your repository

Download sample code

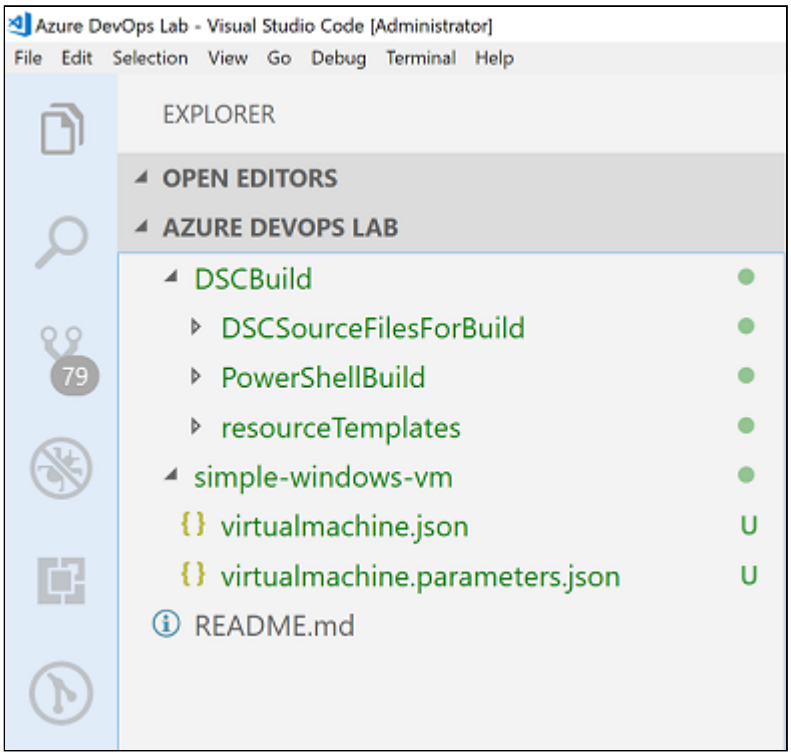
Download and save the following two zip files (Links to these files can be copied from C:\MyAzureProject\Azure Superpowers\Lab - Helper Files\Helper.txt)

https://azuresuperpowers.blob.core.windows.net/azuredevops/DSCBuild.zip?st=2018-10-04T02%3A30%3A28Z&se=2020-10-05T02%3A30%3A00Z&sp=rl&sv=2018-03-28&sr=b&sig=m2twlkmm0Pg17NId4TjnAjFFs7sLFucANFrDo4Y72Xw%3D

https://azuresuperpowers.blob.core.windows.net/azuredevops/simple-windows-vm.zip?st=2018-10-04T02%3A31%3A06Z&se=2020-10-05T02%3A31%3A00Z&sp=rl&sv=2018-03-28&sr=b&sig=SaxwRzWULbGTLN1r7C06DUgF%2FM55eRH6ZxVAusjX6b0%3D

Extract zip files and commit code to the dev branch in your repository

1. Run a 'git fetch' operation in VS Code to pull down the dev branch
2. Switch branches to the dev branch (It will be listed as origin/dev)
3. Extract the zip files to get access to the source code
4. Copy root level folders into your repository
5. The dev branch of your repository needs to look **EXACTLY** as follows



6. Ensure that your dev branch looks **EXACTLY** like the screenshot above. Typical issues encountered in this step usually relate to copying the wrong root directories.
7. Commit to your local repository dev branch and push the files to the dev branch of your Azure DevOps repository

Merge changes to the master branch via Pull Request

1. Enter the Repos section of the project in Azure DevOps
2. Select the Pull requests button
3. Click the **New pull request** button
4. A pull request involves the merging of code from one branch into another branch. You may need to select the two branches that you want to have involved in the request, as Azure DevOps may default to other branches than what you are currently working with.
5. In this case, you will want to merge changes from the **dev** branch into the **master** branch.
6. You should give a clear title for the pull request that describes the changes in the branch. In the description field give a clear explanation of how the changes are implemented along with any resources that might help reviewers understand the changes. You can include Azure DevOps work items and hyperlinks to allow others to have as much context as possible when reviewing your changes. You can add any team member who you would like to review the changes.
7. Click the Create button to create the pull request.
8. Approve the pull request and complete it.
9. On the **Complete pull request** window, uncheck the box which will delete your dev branch and click **Complete merge**

Exercise - Service Principal

Create a new Service Principal

1. Refer to Lab - Azure AD and Service Principals to create a new Service Principal in your Azure subscription.

Exercise - Create a new Resource Group to contain a storage account for DSC files

Login to Azure

1. Open a web browser
2. Navigate to either https://portal.azure.com for commercial, or https://portal.azure.us for MAG

Create a Resource Group

1. On the left-hand navigation pane, click 'Resource Groups'
2. On the 'Resource Groups' pane, click the '+Add' button
3. On the 'Resource Group' pane, enter the resource group name 'DSCFiles-<YOURALIAS>' and select any valid location. Click the 'Create' button.

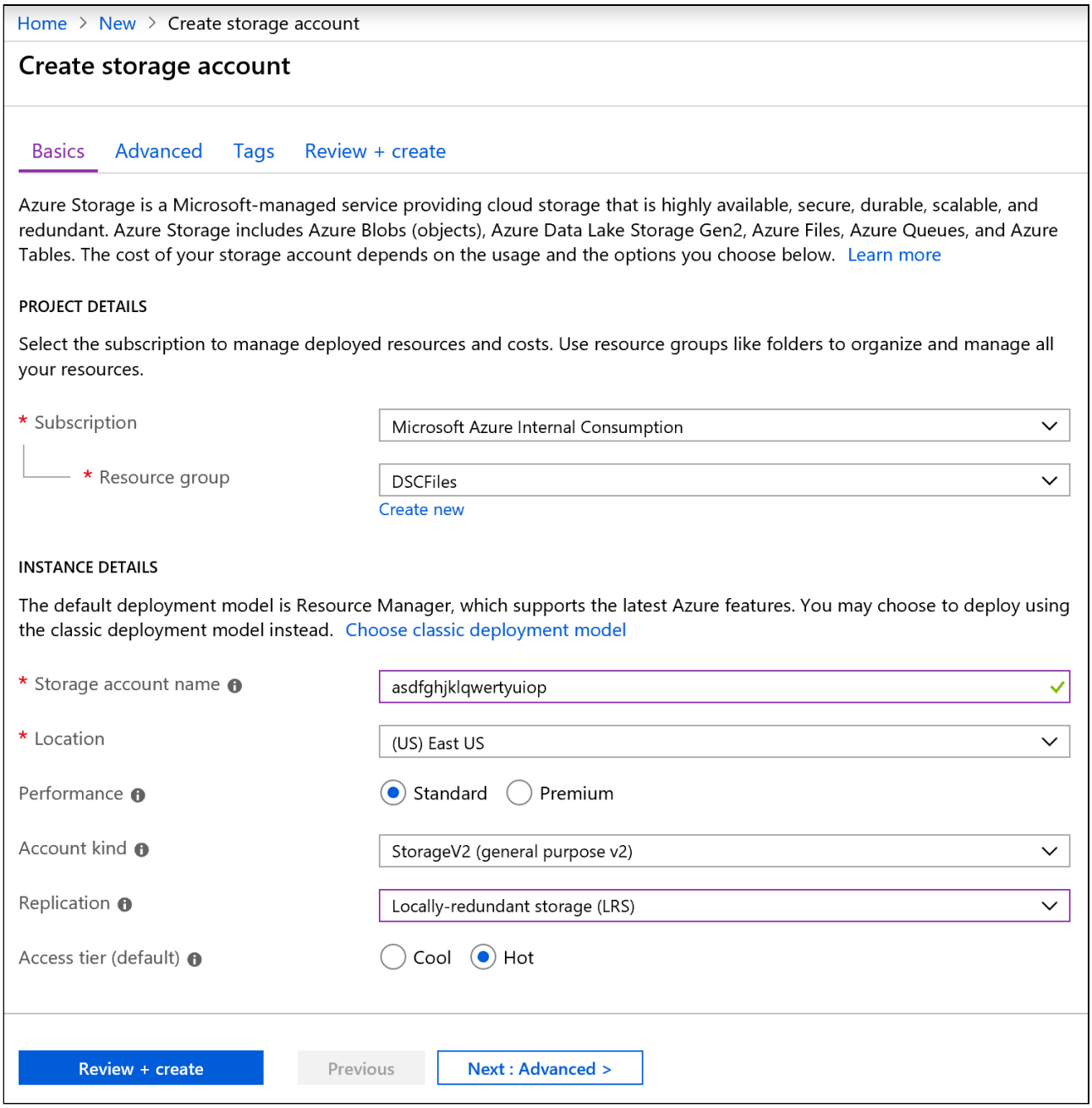
Exercise - Create a new storage account for DSC files

Login to Azure

1. Open a web browser
2. Navigate to either https://portal.azure.com for commercial, or https://portal.azure.us for MAG

Create a Storage Account

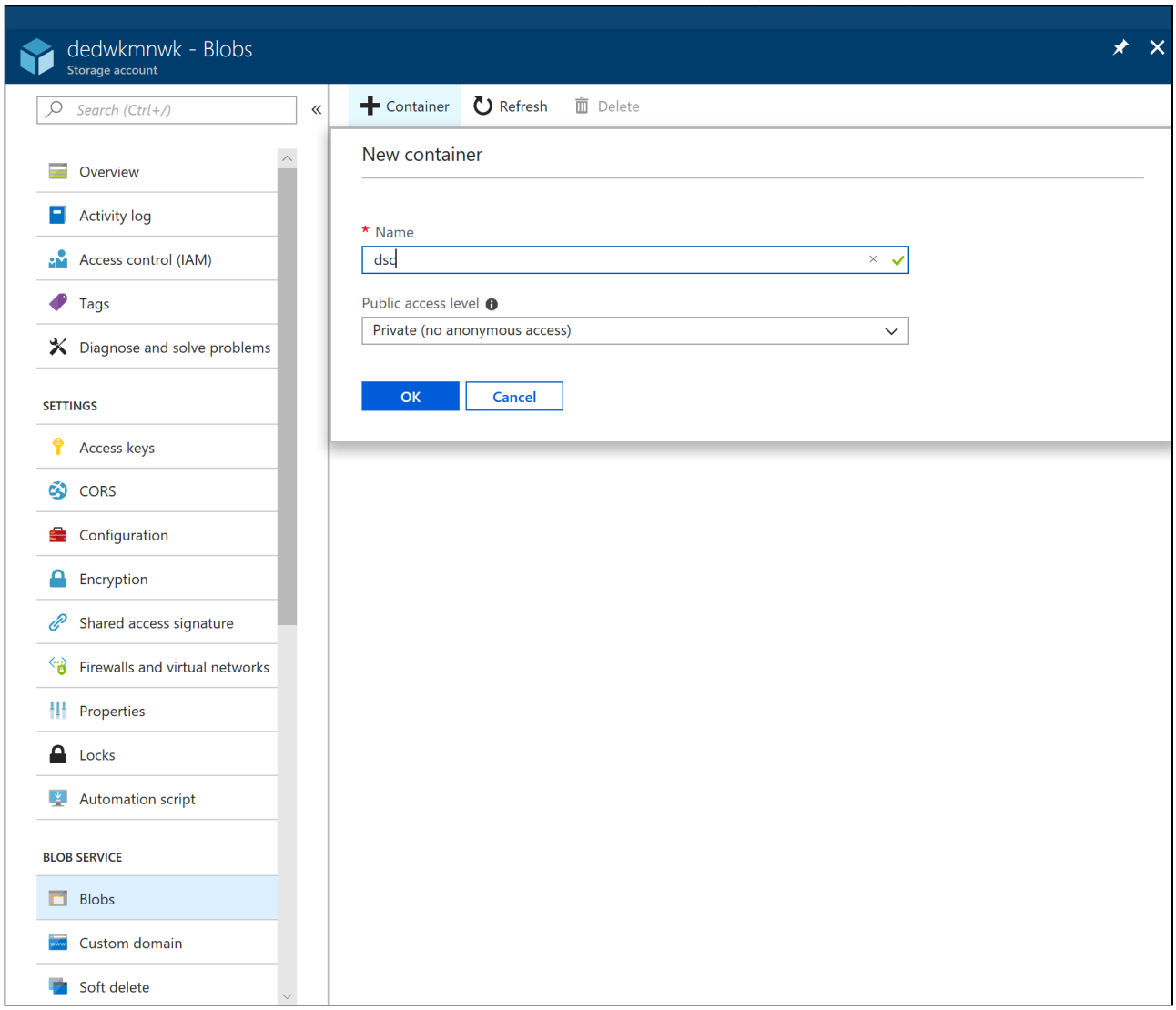
1. On the left-hand navigation pane, click 'Create a resource'
2. Search the marketplace with the following query: storage
3. Select the search result labeled 'Storage account' that is published by Microsoft, and in the new tab that opens, click Create
4. Use the following screenshot for reference on desired storage account options. You will need to pick your own globally unique name for your storage account. Ensure that you select the Resource Group that you created in the previous Exercise. Your portal experience may look different as the portal updates frequently, but the information needed to deploy should be listed below.



5. Click on the **Review + create** button

6. Click the **Create** button to create the storage account.

7. Once your storage account is created, create a new container **(Private)** in the blob storage configuration. Name the container 'dsc'



Exercise - Create two new Resource Groups for your application deployment

Login to Azure

1. Open a web browser
2. Navigate to either https://portal.azure.com for commercial, or https://portal.azure.us for MAG

Create a Resource Group

1. On the left-hand navigation pane, click 'Resource Groups'
2. On the 'Resource Groups' pane, click the '+Add' button
3. On the 'Resource Group' pane, enter the resource group name 'azure-dev-<YOURALIAS>' and select any valid location. Click the 'Create' button.
4. On the 'Resource Groups' pane, click the '+Add' button
5. On the 'Resource Group' pane, enter the resource group name 'azure-prod-<YOURALIAS>' and select any valid location. Click the 'Create' button.

Exercise - Grant Service Principal with permissions

Login to Azure

1. Open a web browser
2. Navigate to either https://portal.azure.com for commercial, or https://portal.azure.us for MAG

Edit Resource Group Permissions

1. On the left-hand navigation pane, click 'Resource Groups'
2. As a reminder, you should have created 3 new resource groups as a part of this lab that will need to have permissions edited

   1 Resource Group to contain a storage account for DSC files

   2 Resource Groups for your application deployment
3. One at a time, select each Resource Group that you created in this lab, and edit the Access Control (IAM) settings
4. For each Resource Group, edit the permissions and grant your new Service Principal with **Contributor** permissions

Exercise - Service Connection

Create a new Service Connection in your project

1. Refer to Lab - Azure DevOps Service Connection for instructions on creating a Service Connection and create a new Service Connection in your project using your Service Principal credentials.

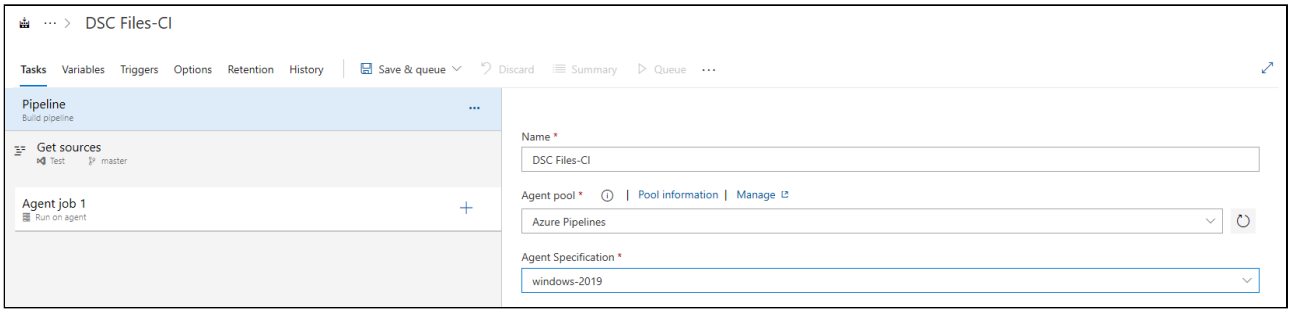
Lab 13 - Azure DevOps Build Continuous Integration

Lab Description	**In this lab, you will explore Continuous Integration in Azure DevOps.**
	Continuous Integration (CI) is the process of automating the build and testing of code every time a team member commits changes to version control. CI encourages developers to share their code and unit tests by merging their changes into a shared version control repository after every small task completion. Committing code triggers an automated build system to grab the latest code from the shared repository and to build, test, and validate the full master branch (also known as the trunk or main).
	CI emerged as a best practice because software developers often work in isolation, and then they need to integrate their changes with the rest of the team's code base. Waiting days or weeks to integrate code creates many merge conflicts, hard to fix bugs, diverging code strategies, and duplicated efforts. CI requires the development team's code be merged to a shared version control branch continuously to avoid these problems.
	CI keeps the master branch clean. Teams can leverage modern version control systems such as Git to create short-lived feature branches to isolate their work. A developer submits a "pull request" when the feature is complete and, on approval of the pull request, the changes get merged into the master branch. Then the developer can delete the previous feature branch. Development teams repeat the process for additional work. The team can establish branch policies to ensure the master branch meets desired quality criteria.
	Teams use build definitions to ensure that every commit to the master branch triggers the automated build and testing processes. Implementing CI this way ensures bugs are caught earlier in the development cycle, which makes them less expensive to fix. Automated tests can run for every build to ensure builds maintain a consistent quality.
Estimated Time to Complete	60 minutes
Key Takeaways	1. Continuous Integration can be configured as a part of your application development process to ensure code is built and tested anytime changes are made to the code.
	2. Continuous Integration can provide help meet infrastructure needs, including automating the build of DSC zip files for use in Azure.
Author	Ralph Kyttle

Exercise - Setup an Azure DevOps build that uses CI to create DSC zip files

Create an Azure DevOps build

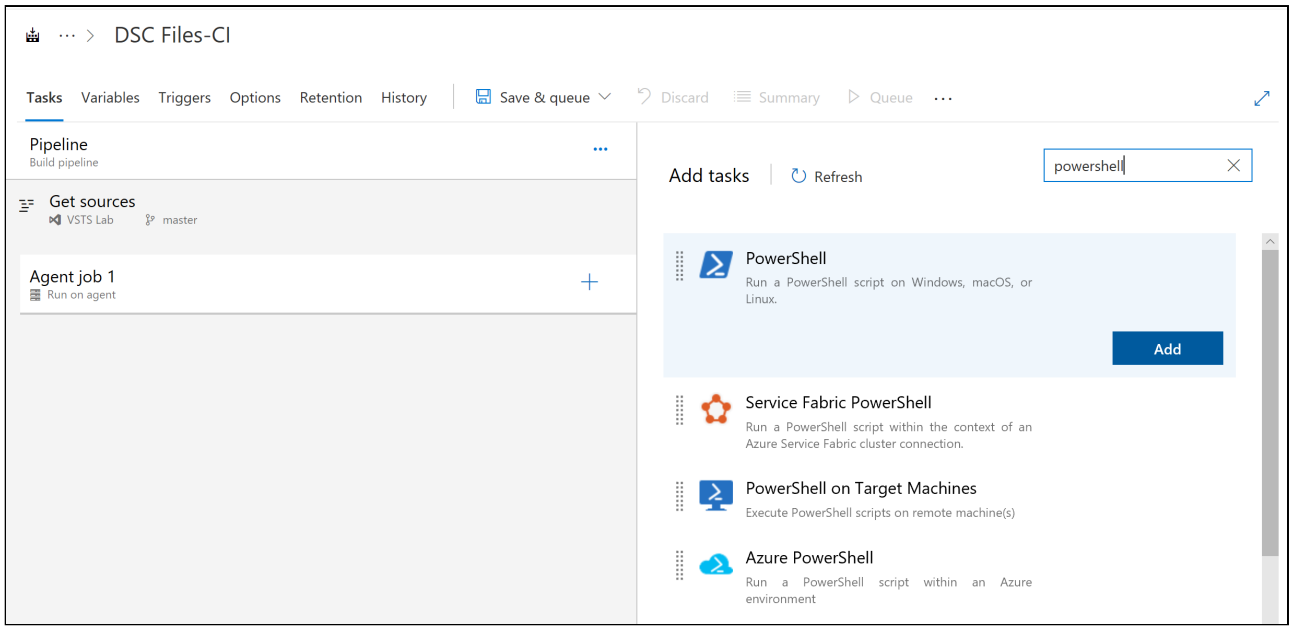
1. From within your Azure DevOps project, select the Pipelines tab
2. Under Pipelines, click Builds
3. From the Builds screen, click New pipeline
4. At the **Where is your code?** screen click the link, **"Use the classic editor"**
5. Your Azure DevOps Git Repo should be selected by default. Click Continue
6. When prompted for a template, select the option to start with an Empty job
7. Name your build 'DSC Files - CI'
8. For the **Agent pool** select 'Azure Pipelines' and for the **Agent Specification** select 'windows-2019'



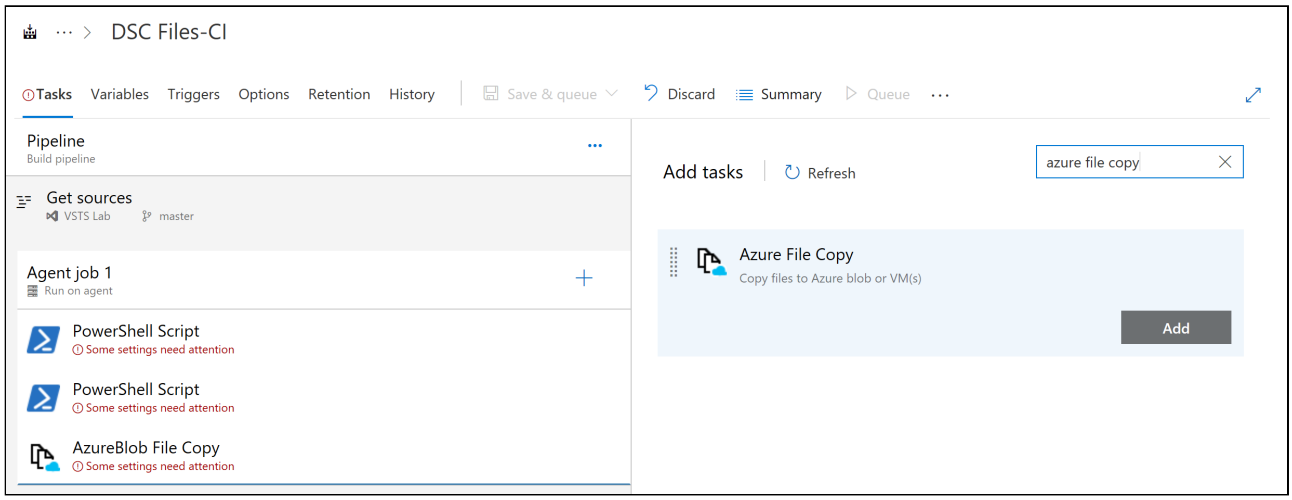
9. Under 'Agent job 1' click the + sign to add a new task

10. Add two PowerShell tasks

(Please note: This is just the PowerShell task, not the Azure PowerShell task)

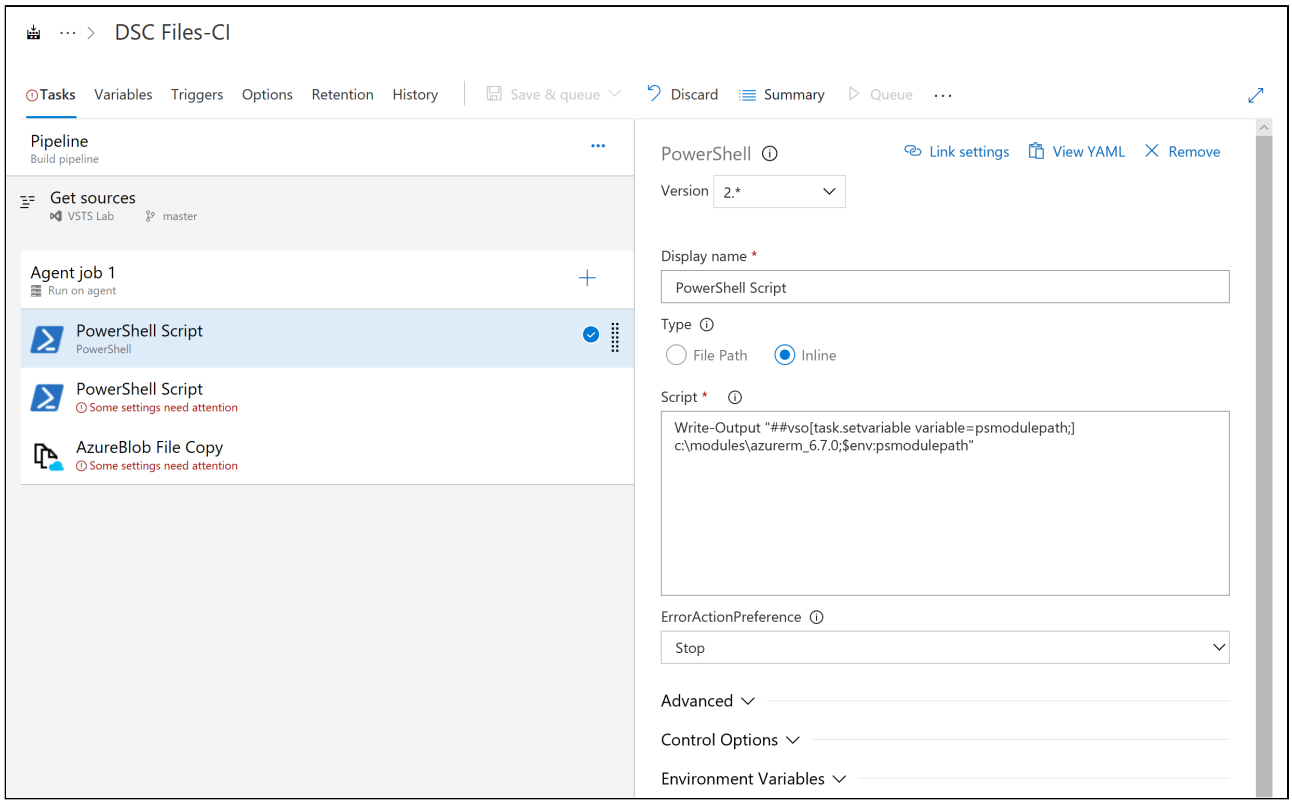


11. Add an Azure File Copy task



12. You will need to edit each task to specify required settings

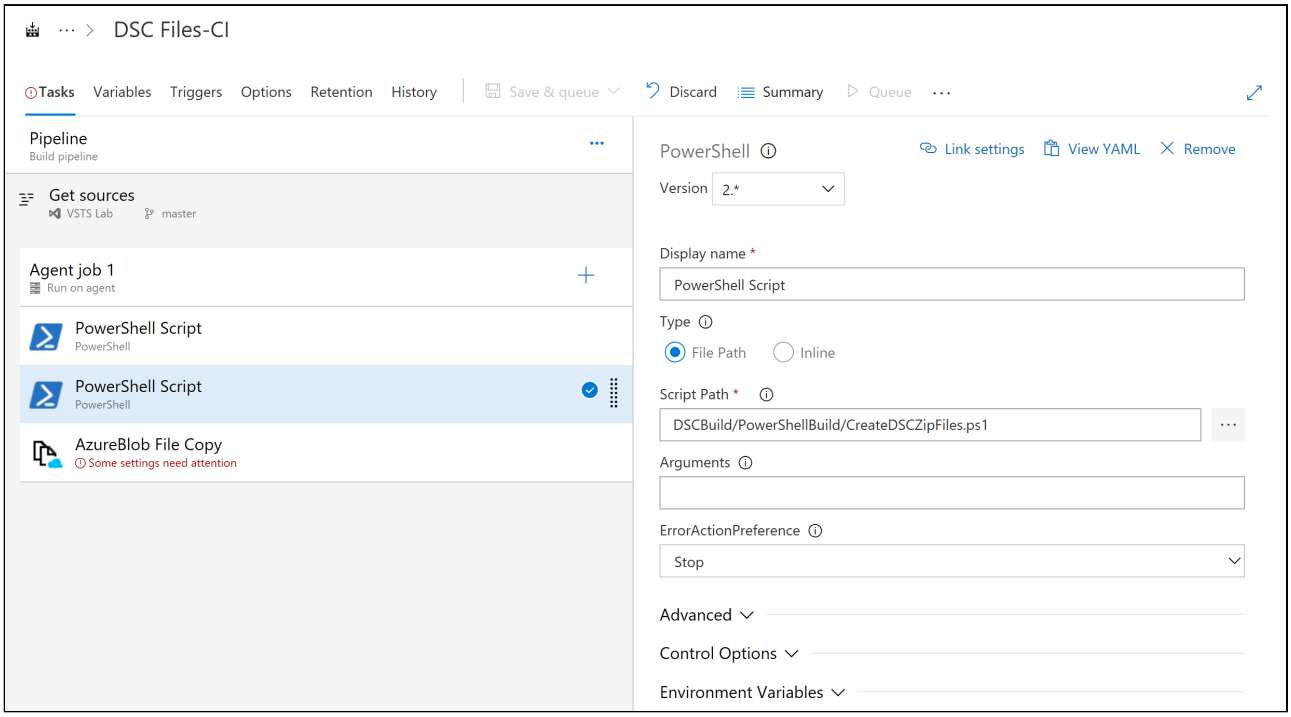
13. PowerShell Task 1



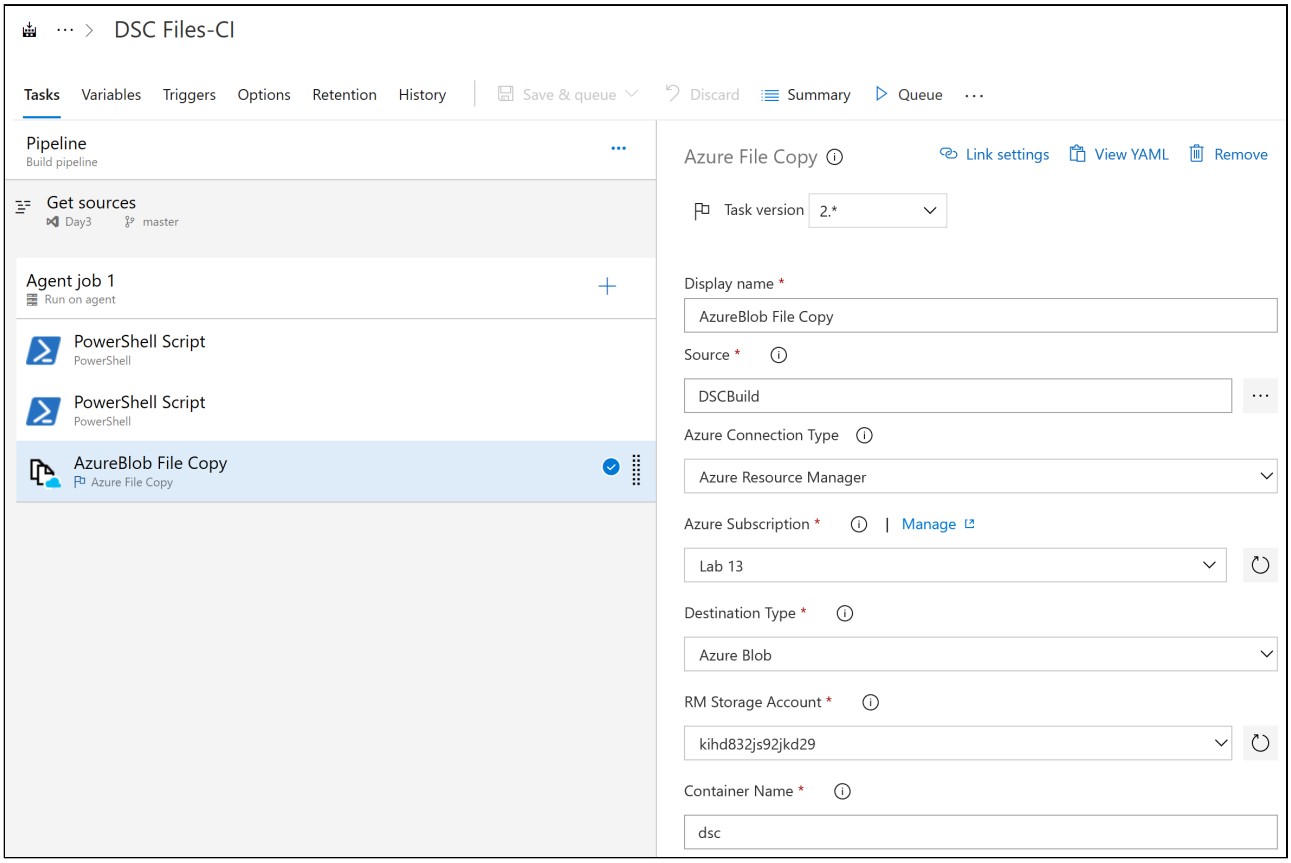
Script: Write-Output "##vso[task.setvariable variable=psmodulepath;]c:\modules\azurerm_6.7.0;$env:psmodulepath"

(Command above can be copied from C:\MyAzureProject\Azure Superpowers\Lab - Helper Files\Helper.txt)

14. PowerShell Task 2 **(Use the "..." button to browse to the file location in your project repo and select the CreateDSCZipFiles.ps1 file)**



15. Azure File Copy Task **(Use your Service Connection for Azure Subscription, Use your Storage Account created in Lab 13, Use your container that was created in the storage account)**



16. In the dropdown for Azure Subscription, select the service connection that you recently created.
17. Once you have made all edits to the tasks, save your build.
18. Queue the build to execute it and have it run the tasks that you defined.
19. Upon successful build completion, you should be able to confirm that the file upload was successful by reviewing the storage account using Azure Storage Explorer
20. Once you have seen a successful build, return to the Build Definition and edit your build **Triggers**
21. Enable Continuous Integration and save
22. Switch over to your repository in VS Code, and make sure you have the **dev** branch selected
23. As a test to view Continuous Integration in action, edit line 145 within SetWinSecuritySettings.ps1 in your local repo within VS Code, changing ValueData from 0 to 1

    This file can be found at: *DSCBuild\DSCSourceFilesForBuild\SetWinSecuritySettings*
24. Save, commit, sync
25. Perform a pull request to merge your code from your dev branch into your master branch
26. From within your Azure DevOps project, select the Pipelines tab
27. Under Pipelines, click Builds
28. Select the 'DSC Files - CI' build
29. Review the build history information, which should now indicate that a new build was automatically initiated based on your latest file update. This new build should differ from your first build, as this new build will be listed as a CI build.
30. Click into the Build to view its Summary and Logs
31. Review your storage account upon completion of your build. You should see a new zip file has been created within the SetWinSecuritySettings folder inside of resourceTemplates. You can validate that this file has been updated by reviewing the 'Last Modified' time. You can also download the zip file, extract the contents and review SetWinSecuritySettings.ps1 to verify that line 145 is listing ValueData =1.

Lab 14 - Azure DevOps Release Continuous Deployment

Lab Description	**In this lab, you will explore Continuous Delivery in Azure DevOps.**
	Continuous Delivery (CD) is the process to build, test, configure and deploy from a build to a production environment. Multiple testing or staging environments create a Release Pipeline to automate the creation of infrastructure and deployment of applications. Successive environments support progressively longer-running activities of integration, load, and user acceptance testing. Continuous Integration starts the CD process and the pipeline stages each successive environment to the next upon successful completion of tests.
	Without Continuous Delivery, software release cycles were previously a bottleneck for application and operation teams. Manual processes led to unreliable releases that produced delays and errors. These teams often relied on handoffs that resulted in issues during release cycles. The automated release pipeline allows a “fail fast” approach to validation, where the tests most likely to fail quickly are run first and longer-running tests happen after the faster ones complete successfully.
Estimated Time to Complete	45 minutes
Key Takeaways	1. Continuous Integration starts the CD process and the pipeline stages each successive environment the next upon successful completion of tests.
	2. CD automates deployment from one environment to the next and may optionally depend on an approval step, in which a decision maker signs off on the changes electronically. CD may create an auditable record of the approval in order to satisfy regulatory procedures or other control objectives.
	3. Modern release pipelines allow development teams to deploy new features fast and safely.
Author	Ralph Kyttle

Exercise - Setup an Azure DevOps release

In this lab, you will...

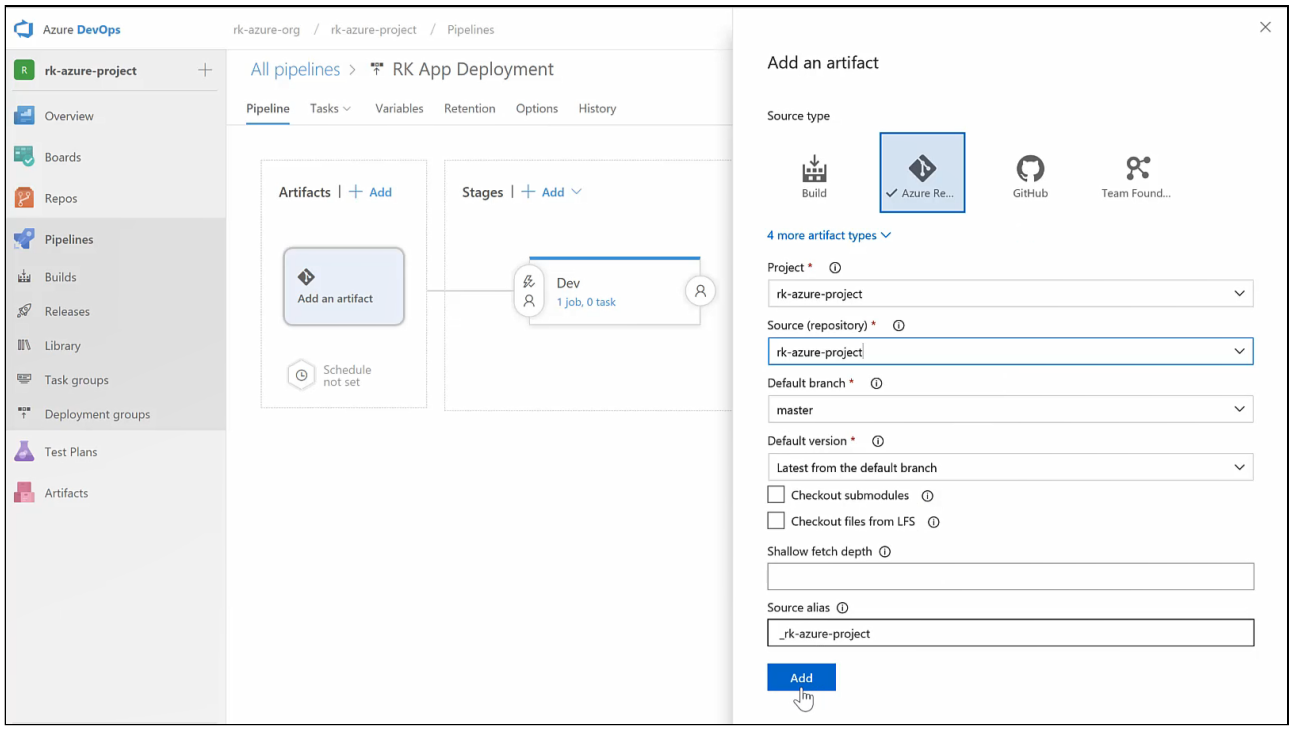
Login to your Azure DevOps account and setup a release definition that uses your Azure DevOps service endpoint to connect into Azure and deploy an ARM template into your Azure resource groups. Enable Continuous Delivery as a part of your release. The release definition should include deployments to both resource groups, dev and prod, and should require approval to deploy to prod.

Create a release. The deployment should kick off in dev, but should wait for your approval to deploy into prod. Allow the deployment to complete in dev, but reject the deployment when you are asked if it should proceed to prod.

Open VS Code and edit your ARM template. Change your VM Size to one size smaller. Save, commit, and push. CD will kick off and create a new release. This will shut your VM down in dev, then reboot it with the new VM size. Once the deployment completes in dev, approve the release moving forward into prod. Review the deployment into prod via Azure DevOps and Azure deployment logs.

Create an Azure DevOps release

1. From within your Azure DevOps project, select the Pipelines tab
2. Under Pipelines, click Releases
3. From the Releases screen, click New pipeline
4. When prompted for a template, select the option to start with an Empty job
5. Create your first stage, and name it Dev
6. Update the name of your release, by default it will be listed as 'New release pipeline', update this name and call it, 'My DevOps Release Pipeline'
7. Under the Artifacts section of your release, click Add an artifact
8. Under the artifact source type, Build will be selected by default. Select the second option 'Azure Repos Git', and point to your project and repo. Ensure that the master branch is selected, and click Add.



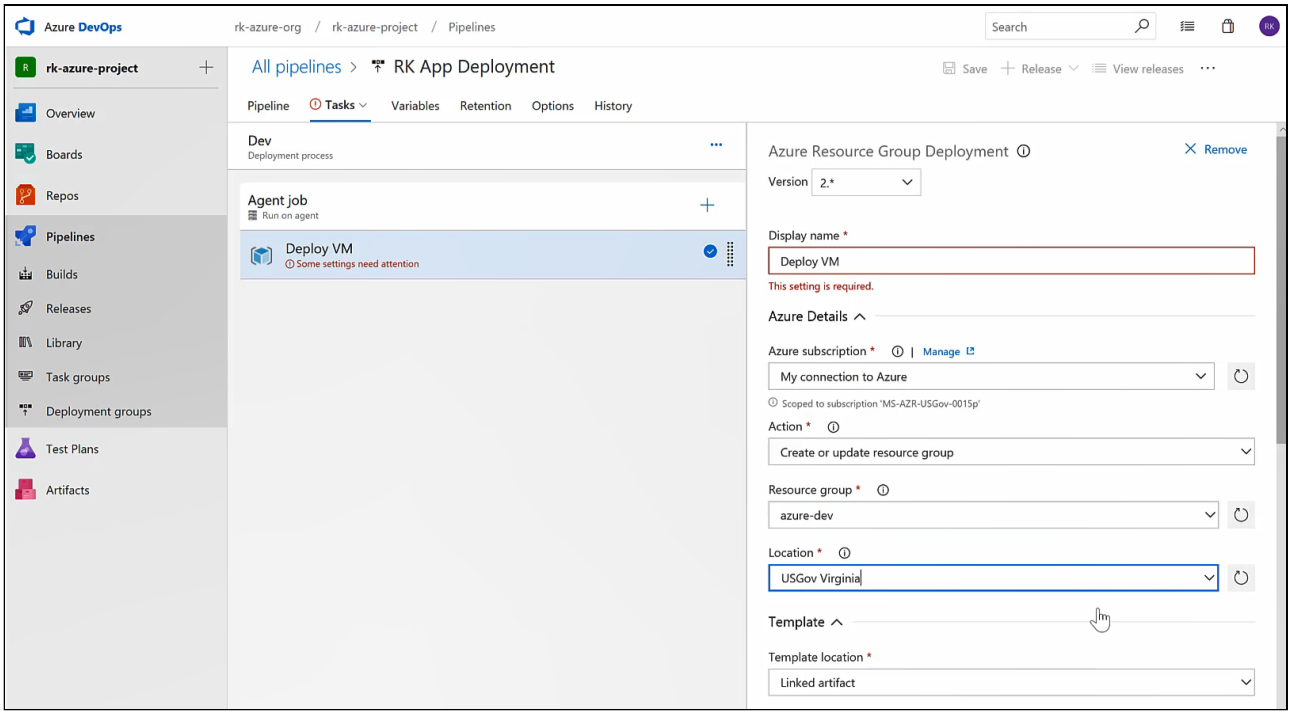
9. Click the link, '1 job, 0 task' to edit the tasks in the Dev stage

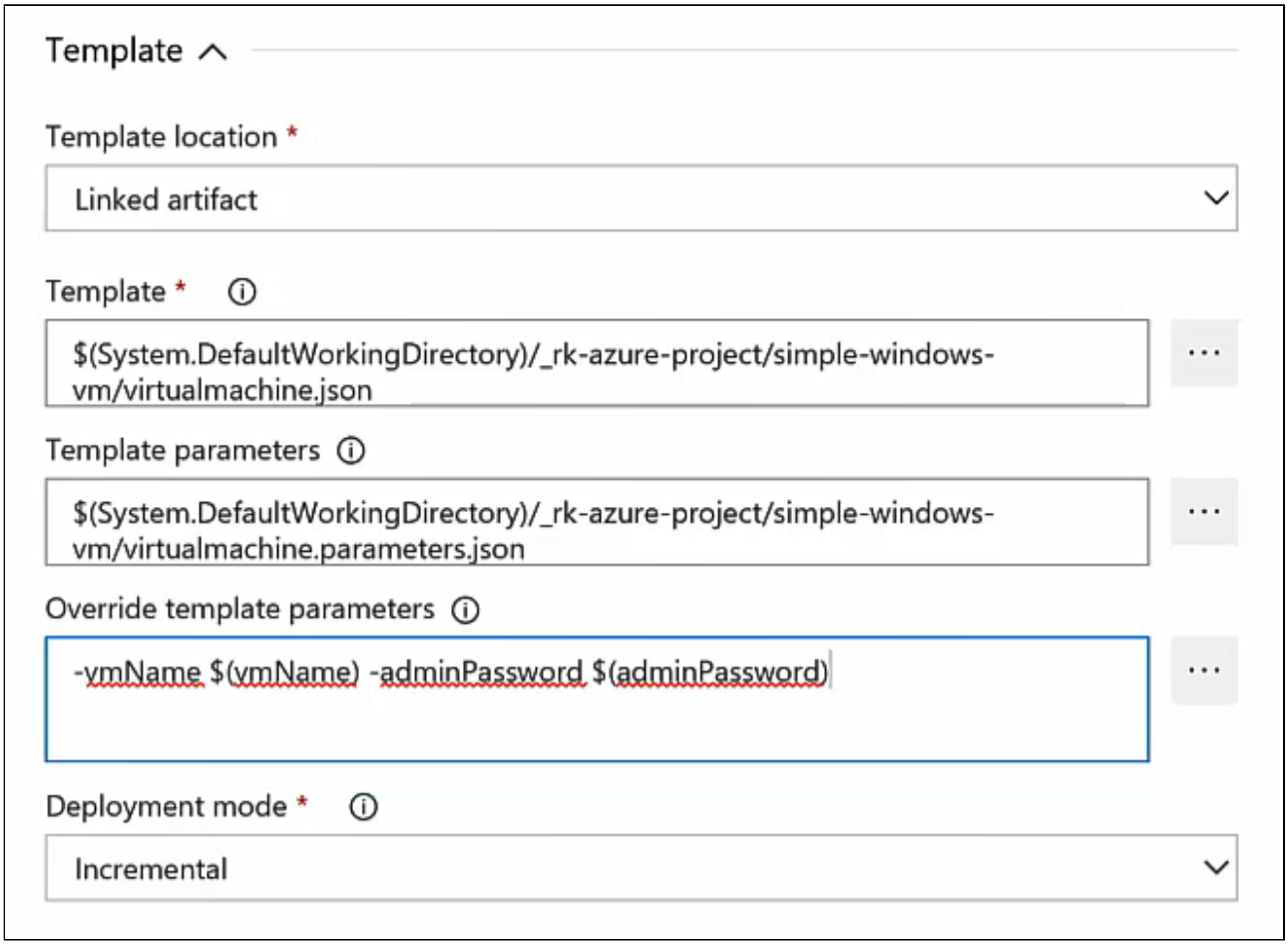
10. Next, click the plus sign to add tasks to the release to define the steps to execute whenever the release is triggered

11. Add an 'Azure Resource Group Deployment' task

12. Once the task has been added, click on it, and edit the properties. Sample information is provided below with screenshots as examples only.

```
Display name: Deploy VM

Azure Subscription: Name of service connection

Resource Group: azure-dev-alias (Example is based on using naming convention
in Lab 12)

Location: Use the location specified when creating your resource group used
above

Override template parameters: -vmName $(vmName) -adminPassword
$(adminPassword)
```



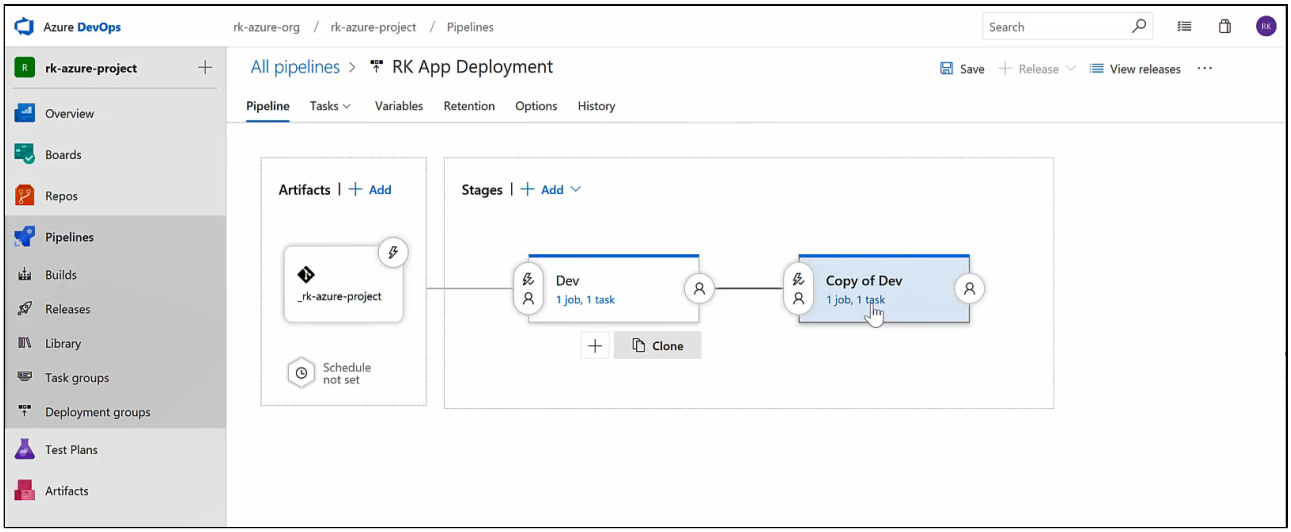


Override template parameters: -vmName $(vmName) -adminPassword $(adminPassword)

(Command above can be copied from C:\MyAzureProject\Azure Superpowers\Lab - Helper Files\Helper.txt)

13. Click save to save the pipeline before proceeding to further edits.

14. Return back to the pipeline view, and clone the Dev stage, to create a new stage for Prod (We do this before creating the variable so the Prod scope will be avaialbe in the variables setting)

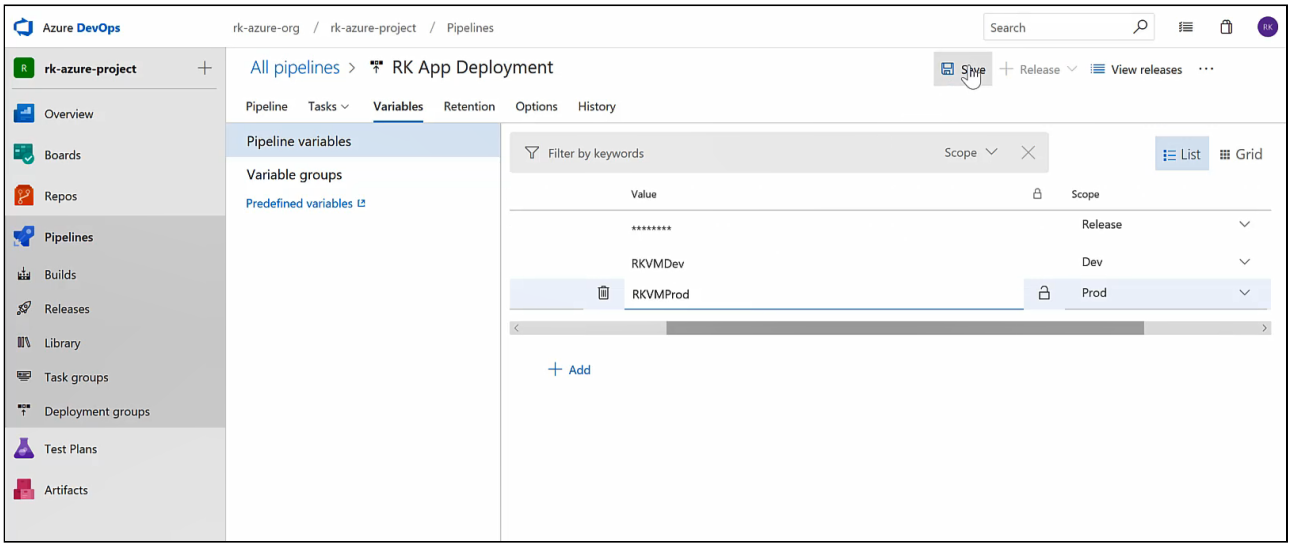


15. Name the new stage Prod, and update its values to point to proper values for Prod. In this step, update the resource group that the Prod stage deploys into.

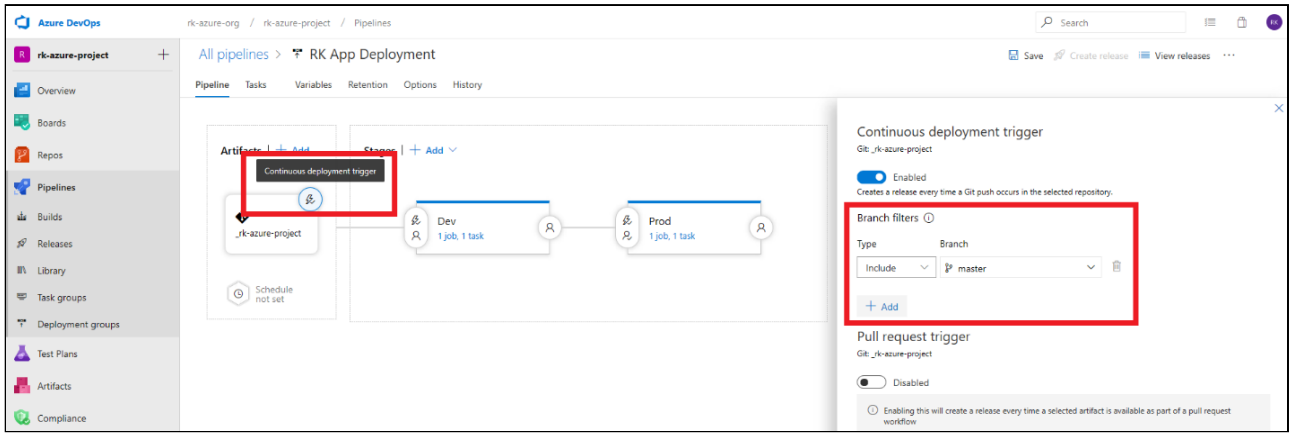
16. Since we provided the template parameters using the builtin varibles feature in Azure DevOps we will now need to update the variables section of this pipeline definition. **Alternatively, we could have simply provided the parameter in the format: -parametername parametervalue. However, items such as the adminPassword would then be passed to azure in an non-secure way.** Using variables can be helpful in a release definition, as they can enable some level of consistency within your stages, and allow for updates to be managed at the variables level

17. Create the variables to be used in your release. Sample information is provided below and includes example screenshots:

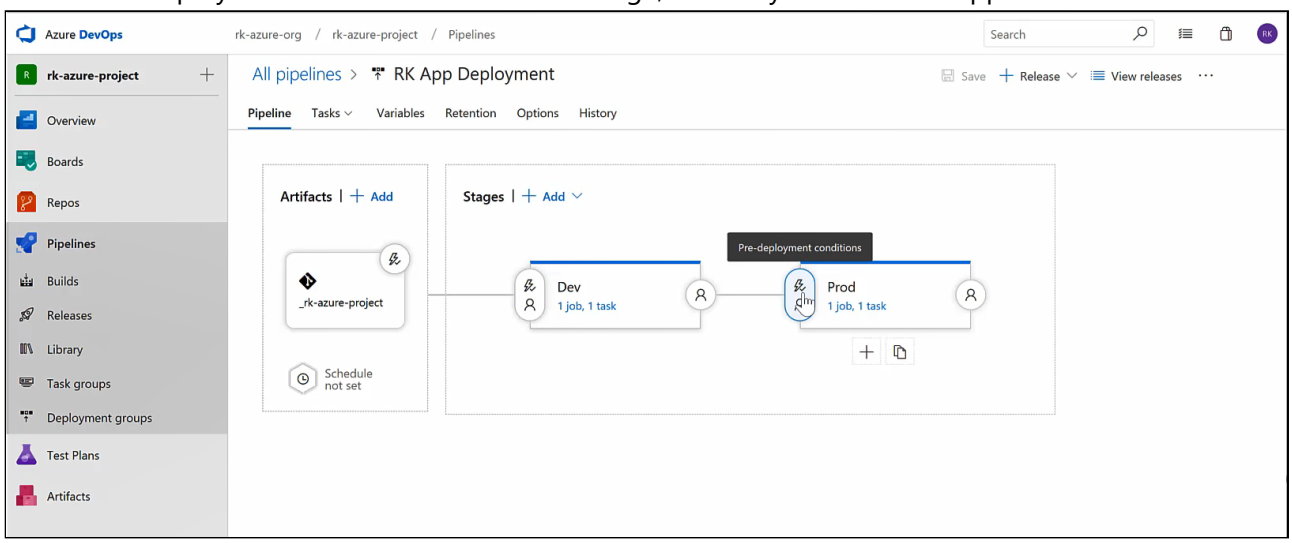
```
Variable Name: vmName          Variable Value: DevVM      Variable Scope:
Dev
Variable Name: vmName          Variable Value: ProdVM     Variable Scope:
Prod
Variable Name: adminPassword   Varialbe Value: P@ssW0rd!  Variable Scope:
Release
```

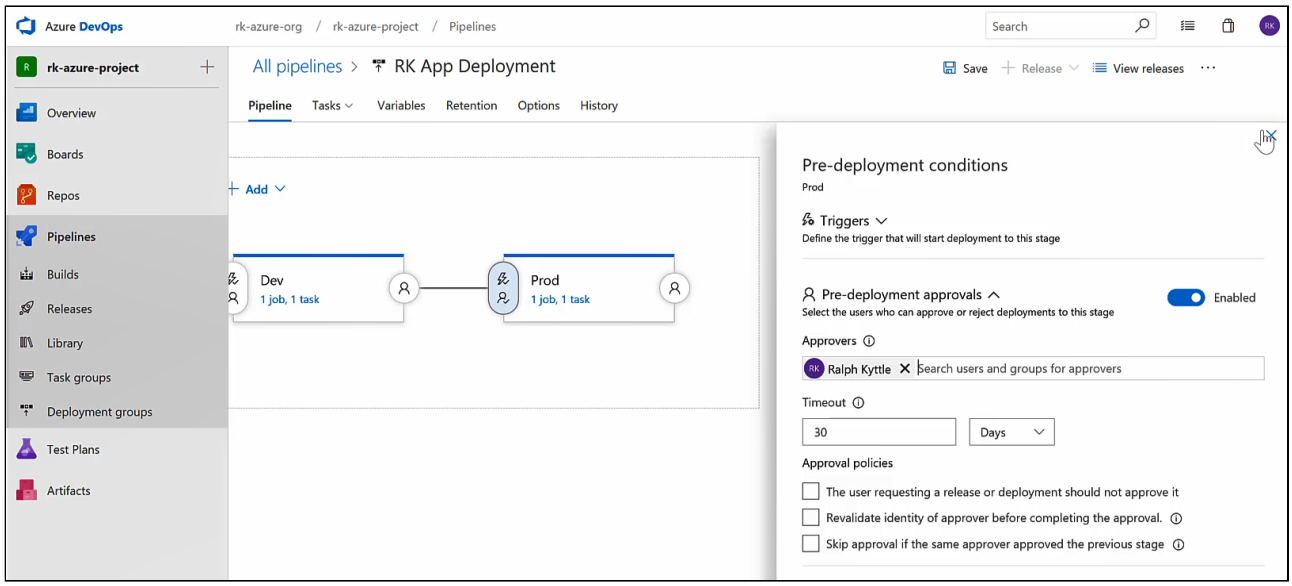


18. Click back to the Pipeline view to setup Continuous Deployment by setting the Continuous Deployment trigger on the Artifact to Enabled (See screenshot below: trigger icon is marked with red box), set a Branch filter for the master branch, and Save the release. Setting a Branch filter on the master branch ensures that continuous deployment will only automatically trigger this release whenever code has made its way into the master branch.



19. Enable Pre-deployment conditions on the Prod stage, and set yourself as the approver





20. Save the release

21. Create a Release to kick off your deployment. Keep any default values that are already selected and click Create.

22. The deployment should kick off in Dev, but it will wait for your approval to deploy into Prod. Allow the deployment to complete in Dev, but reject the deployment when you are asked if it should proceed to Prod. You should receive email notification once the release is complete in Dev and is pending deployment into Prod.

23. Head over to VS Code, and edit virtualmachine.json *(simple-windows-vm\virtualmachine.json)*

24. Edit line 10, and update the default value to Standard_D1_v2

25. Save, commit and push, then perform a Pull Request to merge code from the dev branch into the master branch

26. Thanks to continuous deployment, this change will make its way directly into Dev, but will be pending release into Prod until you approve.

27. View your release status in Azure DevOps to see that continuous delivery has already kicked off a new deployment into Dev. You can also view status in the Azure portal, by taking a look at the Deployments tab on your Dev resource group.

28. Review the deployment logs, and once the release to Dev is complete, approve the release into Prod. We should see in both Dev and Prod that the VM size deployed should be Standard_D1_v2

29. Edit the code in your repo, changing the VM size back to Standard_D2_v2

30. Save, commit, push

31. Watch as continuous delivery makes changes to the VM size in Dev, and again waits for your approval for release to Prod

32. Review deployment logs, and approve the release to Prod

Appendix

Azure DevOps Visio