



# Deploy Azure Resources Using ARM Templates

**Hands-on lab**

Applications that are deployed in Microsoft Azure often comprise different, but related cloud resources, such as virtual machines, web applications, SQL databases, virtual networks, and others. Before the introduction of Azure Resource Manager, it was necessary to define and provision these resources imperatively. Azure Resource Manager gives you the ability to define and provision these resources with their configuration and associated parameters declaratively in a JavaScript Object Notation (JSON) template file, known as an Azure Resource Manager (ARM) template.

In this lab, you will learn how to create and deploy IaaS applications using ARM templates.

Produced by HynesITe, Inc  
Version 1.0  
10/2/2015



This document supports a preliminary release of a software product that may be changed substantially prior to final commercial release. This document is provided for informational purposes only and Microsoft makes no warranties, either express or implied, in this document. Information in this document, including URL and other Internet Web site references, is subject to change without notice. The entire risk of the use or the results from the use of this document remains with the user. Unless otherwise noted, the companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted in examples herein are fictitious. No association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

Copyright 2014 © Microsoft Corporation. All rights reserved.

Microsoft Active Directory, Azure Active Directory, Azure, Hyper-V, Windows, and Windows Server 2012 are trademarks of the Microsoft group of companies.

All other trademarks are property of their respective owners.

## Contents

Deploy Azure Resources Using ARM Templates.....	4
Before You Begin.....	5
Azure Subscriptions .....	5
Hosted Workstations.....	6
Use of Own System .....	6
GitHub repository for Lab Files.....	6
Required Software .....	6
Optional Software.....	6
Access the Lab Environment .....	8
Introduction and Scenario .....	9
Prepare the Azure Infrastructure.....	10
Run the Lab01Start.ps1 script.....	10
Understand Azure Templates and Resource Groups .....	11
Understand Azure Quickstart templates .....	12
Configure Git repository and Visual Studio Code .....	20
Create GitHub account and public repository.....	20
Configure GitHub desktop and clone repository.....	21
Examine Visual Studio Code integration with Git and push commits to remote repository.....	25
Modify and Deploy Azure Quickstart Templates.....	30
Modify sample template and parameter JSON files.....	30
Deploy custom JSON files .....	37
Remove resource group used for lab.....	43
Remove Azure resource group.....	43

## Deploy Azure Resources Using ARM Templates

Applications that are deployed in Microsoft Azure often comprise different, but related cloud resources, such as virtual machines, web applications, SQL databases, virtual networks, and others. Before the introduction of Azure Resource Manager, it was necessary to define and provision these resources imperatively. Azure Resource Manager gives you the ability to define and provision these resources with their configuration and associated parameters declaratively in a JavaScript Object Notation (JSON) template file, known as an Azure Resource Manager (ARM) template.

In this lab, you will learn how to create and deploy IaaS applications using ARM templates.

## Before You Begin

In this lab, you will examine and analyze a number of quick start ARM templates that are available on GitHub. You will create a GitHub account, if you don't already have one, to host a GitHub repo for a quick start template that you will download and then modify and deploy. When you have completed the lab exercises, you will clean up the Azure resources that you created in the lab.

To as great extent as possible, the lab instructions assume the use the Azure Preview Portal, which is located at <https://portal.azure.com>. Some tasks are only available through the Azure Portal. There will be some need to switch back and forth between the two portals. Most of what you will be doing could also be done using the full portal. However, for the sake of consistency and clarity, lab instructions have only been written using the Preview Portal whenever possible. The full portal is located at <https://manage.windowsazure.com>.

For more information on the preview portal, please see <http://channel9.msdn.com/Blogs/Windows-Azure/Azure-Preview-portal> for a brief demonstration or <http://azure.microsoft.com/en-us/documentation/preview-portal/> to read the current documentation for the preview portal.

## Azure Subscriptions

This IT Camp lab requires a valid Azure subscription. While you may use an existing subscription such as a subscription associated MSDN account or existing corporate account, it would preferable to use an Azure Trial subscription for this IT camp. By using a trial subscription, you will avoid any charges against your MSDN or corporate subscription that would result from doing the exercises in this camp.

Your instructor may be able to provide you with a pre-provisioned Microsoft Account that already has an Azure subscription associated with it. Or, you may use a CLEAN and UNUSED Azure Trial account - details on how to set one up are detailed below.

To create a new Azure trial account perform the following steps.

1. Navigate to [www.live.com](http://www.live.com) and click **Sign up now**.
2. Follow the on-screen instructions to create a new Microsoft Account.
3. Navigate to [www.azure.com](http://www.azure.com) and click **Free Trial**.
4. Follow the on-screen instructions to activate a new Windows Azure Trial.
5. Navigate to [Manage.windowsazure.com](http://Manage.windowsazure.com) and sign in.
6. In Microsoft Azure portal, in the upper left, click your user name, and then click **View my bill**.
7. Click your current trial subscription, and then click **Edit subscription details**.
8. Type a name you will recognize in SUBSCRIPTION NAME, such as ITCamps, and then click the **Done** icon.

### Hosted Workstations

Labs in this camp are written to be completed on a pre-configured workstation. Additional labs require an on-premises environment consisting of multiple servers. A hosted virtual machine environment is provided for this purpose. Your instructor will provide a link to this environment.

If you are using the hosted workstation environment, use **Administrator** as the username and **Passw0rd!** as the password.

### Use of Own System

You may complete lab instructions using your own workstation (either Windows 10 or Windows 8.1), providing you download the files used for the lab from GitHub and have the following software installed.

#### GitHub repository for Lab Files

If you are not using the hosted virtual machine and are using your own workstation, any custom files the lab instruction call out can be found in a GitHub repository. The repository is located here:

<https://github.com/AZITCAMP/Labfiles>.

#### Required Software

1. Microsoft Azure PowerShell - <http://go.microsoft.com/?linkid=9811175&clid=0x409> (also installs the Web Platform Installer)
2. Visual Studio Code - <https://code.visualstudio.com/>
3. GitHub Desktop for Windows - <https://desktop.github.com/>
4. Windows Credential Store for Git (if VSCode won't authenticate with GitHub) - <http://gitcredentialstore.codeplex.com/>
5. Iometer - <http://sourceforge.net/projects/iometer/>

#### Optional Software

Any additional software that you require will be called out in the lab. The following software may be useful when working with Azure in general.

1. Remote Server Administration Tools - <http://support.microsoft.com/kb/2693643> (Windows 8.1) or <http://www.microsoft.com/en-ca/download/details.aspx?id=45520> (Windows 10)
2. AzCopy - <http://aka.ms/downloadazcopy>
3. Azure Storage Explorer - <http://azurestorageexplorer.codeplex.com/downloads/get/891668>
4. Microsoft Azure Cross-platform Command Line Tools (installed using the Web Platform Installer)
5. Visual Studio Community 2015 with Microsoft Azure SDK - 2.7.1 (installed using the Web Platform Installer)
6. Msysgit - <http://msysgit.github.io>

## Deploy Azure Resources Using ARM Templates

7. PuTTY and PuTTYgen - [www.putty.org](http://www.putty.org)
8. Microsoft Online Services Sign-In Assistant for IT Professionals RTW - <http://go.microsoft.com/fwlink/?LinkID=286152>
9. Azure Active Directory Module for Windows PowerShell (64-bit version) - <http://go.microsoft.com/fwlink/p/?linkid=236297>

Please note that these lab exercises require a minimum version of 0.9.8 of the Microsoft Azure module for PowerShell. To determine the module version installed on your system, open a Windows PowerShell prompt, type the following commands, and then press ENTER.

```
↪ import-module Azure  
↪ get-module Azure).version
```

```
PS C:\> import-module azure  
PS C:\> (get-module Azure).Version  
Major Minor Build Revision  
----
```

0	9	8	-1
---	---	---	----

### Access the Lab Environment

For this lab you will be accessing a hosted environment that contains all the VMs and resources you require. Your instructor will provide a link to the hosted lab environment.

You should be able to connect with any recent web browser, including Microsoft Edge. Once you have connected to the lab environment, take a few minutes to familiarize yourself with Launchpad.

For this course there are four VMs that you will work in. If you look at the Machines tab on the right side of the lab environment you will find a listing of all the VMs. To switch to another VM, just click on the appropriate name in the Machines list. Below you will find a listing of the VMs for this course.

Virtual Machine	Role
AZRCamp-Admin	Windows 10. A member of the Contoso.com domain. Used for Azure management.
AZRCamp-Edge	A Stand-alone Windows Server 2012 R2 Server. Routing and Remote Access has been installed and it is acting as the default gateway for all outbound traffic.
AZRCamp-DC	Windows Server 2012 R2 domain controller and DNS server.
AZRCamp-Sync	Directory Sync for use in other Labs.

The password for all logons in these VMs is "Passw0rd!".

- ★ You can type this in to the VM manually, or use the **Commands→Paste→Paste Password** sequence from the Launchpad.



## Introduction and Scenario

Contoso, Inc has successfully been using Azure for some time now. However, Contoso's use of Azure has, up until now, been limited to the use of the Azure Service Management API. The Service Management API allows Contoso to manage Azure resources, such as storage accounts, virtual machines, and virtual networks as individual entities. The Service Management API model serves Contoso well, but it does not reflect the fact that these entities are often, in fact, related and independent parts of a larger, single entity.

In contrast to the Azure Service Management API, the Azure Resource Management API allows organizations such as Contoso to deploy, manage, and monitor Azure resources as a single group. In addition to this benefit, ARM makes it possible to use declarative templates to define and provision resources in a deployment. Furthermore, Contoso could implement Role-Based Access Controls (RBAC) to provide greater security. Using ARM, Contoso would be able to apply tags to resources to organize them logically within the same subscription.

A significant and additional benefit is that by using Azure Resource Management templates, Contoso can start to realize an important goal of bringing the best of DevOps practices into its infrastructure by treating its infrastructure as code. Following this paradigm, Contoso will be able to modularize its infrastructure into various components and then join or separate the constituent components as necessary in a highly automatable and scalable manner.

Because of the benefits that Azure Resource Management can provide to Contoso, you have been asked to explore how ARM can be used to deploy and manage resources. Specifically, you have been asked to learn how templates can be used to define and deploy Azure Resources.






## Prepare the Azure Infrastructure

In this exercise, you will use the Lab01Start.ps1 script to log on to your Azure subscription and create an Azure resource group that you will use for the remaining lab exercises. The script will also determine a globally unique name that you can use to create a storage account in subsequent lab exercises.

### Run the Lab01Start.ps1 script

To perform the subsequent lab exercises, you need to create an Azure Resource Group and determine a globally unique name you can use to create a storage account.

 Perform the following tasks on **AZRCAMP-ADMIN**:

1. Open File Explorer and navigate to **C:\LabFiles\AZRITPROCamp\Lab01-Deploy Azure resources using templates**.
  -  You may also download files used for this lab from the GitHub repository for the course at <https://github.com/AZITCAMP/Labfiles>.
2. Right-click **Lab01Start.ps1**, and click **Edit**.
  -  The Windows PowerShell ISE console opens.
3. In Windows PowerShell ISE, on the upper Ribbon, click **Run Script** (green arrow).
4. When prompted, enter a lower-case string that represents your initial, and press ENTER.
  -  The storage account name must contain only lower case letters and numbers and must be globally unique.
5. In the Sign in to Windows Azure PowerShell dialog box, enter the email address of the account associated with your Azure subscription, and click **Continue**.
6. On the sign in page, enter your password, and click **Sign in**.
  -  The script starts running and then pauses to display a name verified as unique for use a storage account name.
7. Record the unique storage account name, and press ENTER to continue.
  -  You will need to know the unique name for the storage account in a subsequent lab exercise.
8. Leave the Windows PowerShell ISE console open for subsequent lab exercises.

## Understand Azure Templates and Resource Groups

Applications that are deployed in Microsoft Azure often comprise different, but related cloud resources, such as virtual machines, web applications, SQL databases, virtual networks, and others. Before the introduction of Azure Resource Manager, it was necessary to define and provision these resources imperatively, for example, with Azure PowerShell cmdlets such as `Add-AzureVMDisk`. Azure Resource Manager gives you the ability to define and provision these resources with their configuration and associated parameters declaratively in a JavaScript Object Notation (JSON) template file. The format for template files is described in detail here: <https://msdn.microsoft.com/en-us/library/azure/dn790564.aspx>.

A JSON template file is a text file that contains descriptions of the resources, configurations code and extensions. JSON templates are idempotent, which means that they can be run multiple times without changing the outcome beyond initial deployment. A consequence of this characteristic is that templates can be used to upgrade applications, for example, by scaling out applications with additional VMs. You modify the template to include the specifications for the additional virtual machines. When you deploy the template, Azure Resource Manager will recognize the resources that have previously been deployed and create only the resources that have been added.

Another advantage of template files is that they can be parameterized. This simplifies orchestration and deployment of applications because the parameterization allows you to reuse templates to create deployments with different configurations based on the parameters you define in the JSON template and that you specify at run time.

Once you create a JSON-based template, you can deploy the resources described in the template by using the Microsoft Azure Preview Portal or using Windows PowerShell commands. An advantage of using Windows PowerShell commands is that you can include them in a Windows PowerShell script that provides additional automation.

When you execute the template, the resources are provisioned to a new or existing Azure resource group. An Azure resource group is a collection of related cloud resources that you can manage as a single entity. Resource groups allow you to start, stop, or delete all resources in a group at once; to apply Role-Based Access Controls rules to provide more granular security permissions on resources; to audit operations within the resource as a whole; and to enable better tracking through the use of metadata tags.

One way to think of Azure resource groups is to consider them as containers where you place logically related resources. For example, a resource group might contain an Internet-facing web site, a backend SQL database, one or more virtual networks, and a storage account for additional assets. Alternatively, you could create a separate resource group for your backend SQL database as this component is likely to have a different lifecycle than your front-end web applications. Therefore, when considering whether resources should be placed in the same or different resource group, you should also keep in mind the lifecycle of those resources, and group resources accordingly.

In this exercise, you gain a basic understanding of Azure Resource Manager templates in general by closely examining one of the quickstart templates available on GitHub.

### Understand Azure Quickstart templates

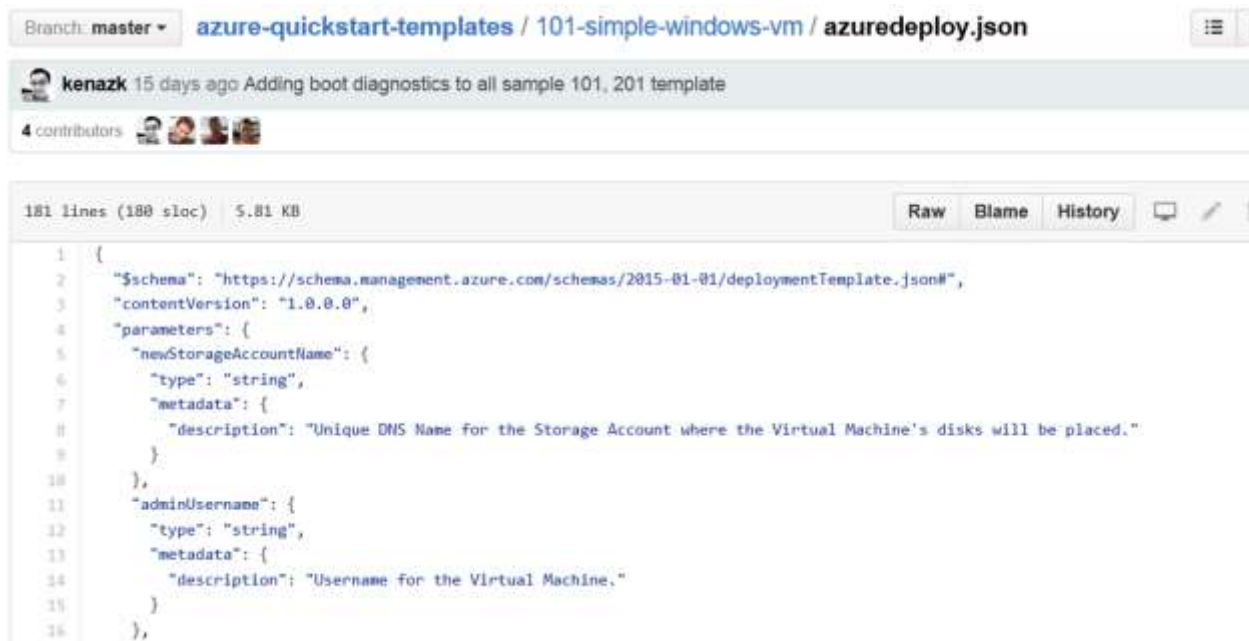
JSON templates simply provide a different way of defining the same types of resources that you may be familiar with using the classic Azure model. For example, to create a new VM using the Azure Service Management mode, you would specify the image name, the virtual machine name, the size, the administrative account, the password credential, and other values. When deploying resources using JSON templates, you specify the same information, but in a different format.

It is rarely necessary to create a JSON template from scratch. Microsoft maintains a large and growing library of community contributed Azure templates on GitHub that you can leverage to learn about JSON templates and to deploy your own applications.

In this task, you will view the quickstart templates that are available on GitHub and examine one of them in depth.

- ✎ Perform the following tasks on AZRCAMP-ADMIN logged on as **Contoso\Administrator** using **Passw0rd!** as the password:
  1. Open Microsoft Edge (or Internet Explorer) and navigate to <https://github.com/Azure/azure-quickstart-templates>.
    - ★ GitHub is a repository service, based on Git that provides source control management (SCM) and revision control. GitHub expands on the functionality of the command-line based Git by providing a GUI and collaboration features.
  2. On the Azure Quickstart Templates page, spend a few moments examining the titles and scroll down the page.
  3. Templates that have a prefix of 101 are simple templates; templates that have a prefix of 201 are more advanced and complex templates.
    - ★ A searchable index of the quickstart templates is available at <https://azure.microsoft.com/en-us/documentation/templates>.
  4. Scroll down the page, locate, and then double-click **101-simple-windows-vm**.
    - ★ Note that the folder contains 4 files: 1) The README.md is an html file that contains the description and the button that you can click to deploy the template to an Azure subscription, 2) the azuredeploy.json file is the main file used to deploy the cloud resources, 3) the azuredeploy.parameters.json is the file where you can provide values for the parameters defined in the azuredeploy.json file and pass them in at execution time, and 4) the metadata.json file provides additional data about the deploy template.
  5. Click **azuredeploy.json**.
  6. The JSON file opens as shown below:

## Deploy Azure Resources Using ARM Templates



```
1 {
2   "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
3   "contentVersion": "1.0.0.0",
4   "parameters": {
5     "newStorageAccountName": {
6       "type": "string",
7       "metadata": {
8         "description": "Unique DNS Name for the Storage Account where the Virtual Machine's disks will be placed."
9       }
10    },
11    "adminUsername": {
12      "type": "string",
13      "metadata": {
14        "description": "Username for the Virtual Machine."
15      }
16    },
```

- ✦ A JSON file is constructed of key/value pairs. For example, in the image above, "ContentVersion" is the key and "1.0.0.0" is the value. A key is always a string enclosed in quotation marks. A value can be a string, securestring, number, boolean expression, array, or object. A JSON object is enclosed in curly braces, "{}". In a key/value pair the key is always followed by a colon. Key/pairs are separated by commas.
- ✦ A JSON template may also contain functions and expressions. Expression are enclosed in square brackets, "[ ]", and can appear anywhere in a JSON string. Functions calls have the format functionName(arg1,arg2,arg3). Properties are referenced using the dot and index operators.
- ✦ A JSON template may have as many as six sections: 1) \$schema (a required element that provides the location of the file that describes the version of the template language, 2) (a required element that provides the version of the template), 3) parameters (an optional element that defines values that are passed in when the template is executed), 4) variables (an optional element that defines the values that are used when template is executed), 5) resources (a required element that defines the resources that are deployed or updated in a resource group), and 6) outputs (an option element that defines values that are returned after a deployment.)

7. At the top of the file, note the first line that references a JSON schema.

- ✦ The schema reference is used by intelligent JSON clients to determine the schema that is applicable to the JSON file and to provide additional functionality such as autocomplete and intellisense.

8. Note the Parameters key.

## Deploy Azure Resources Using ARM Templates

- ★ The value for the Parameters key is an array of parameter objects that represent the dynamic input for the JSON template. Each of the parameter objects has a name that is used to pass values in at runtime and is referenced within the JSON itself in other sections. For example, "NewStorageAccount" is the name of the parameter that is supplied as an input and used to provide the name of the storage account resource specified in the JSON file.

9. In the parameters section, note the "adminPassword" parameter, and note that the "type" value.

```
16     },
17     "adminPassword": {
18         "type": "securestring",
19         "metadata": {
20             "description": "Password for the Virtual Machine."
21         }
22     }
```

- ★ Each parameter has a type, such as string, securestring, number, etc. Securestring is a special data type that is used to ensure that the value is not persisted anywhere in the Azure platform. This ensures that sensitive information such as passwords are not displayed.

10. Scroll down to view the WindowsOSVersion parameter object.

```
29     "windowsOSVersion": {
30         "type": "string",
31         "defaultValue": "2012-R2-Datacenter",
32         "allowedValues": [
33             "2008-R2-SP1",
34             "2012-Datacenter",
35             "2012-R2-Datacenter"
36         ],
37         "metadata": {
38             "description": "The Windows version for the VM. This w:"
39         }
40     }
```

- ★ Note that a parameter can include a default value. When you execute the JSON template, you will be prompted for any parameter values that are missing from the input, unless you have specified a default value. Note also that a parameter can include an array of allowed values, which is useful if you want to constrain the input to a set of specific values, for example specific regions, etc. Finally, note the metadata key, where you can include a description of the parameter.

11. Scroll down to the variables section that begins at around line 42.

```

42  "variables": {
43    "location": "West US",
44    "imagePublisher": "MicrosoftWindowsServer",
45    "imageOffer": "WindowsServer",
46    "OSDiskName": "osdiskforwindowssimple",
47    "nicName": "myVMNic",
48    "addressPrefix": "10.0.0.0/16",
49    "subnetName": "Subnet",
50    "subnetPrefix": "10.0.0.0/24",
51    "storageAccountType": "Standard_LRS",
52    "publicIPAddressName": "myPublicIP",
53    "publicIPAddressType": "Dynamic",
54    "vmStorageAccountContainerName": "vhds",
55    "vmName": "MyWindowsVM",
56    "vmSize": "Standard_A2",
57    "virtualNetworkName": "MyVNET",
58    "vnetID": "[resourceId('Microsoft.Network/virtualNetworks',variables('virtualNetworkName'))]",
59    "subnetRef": "[concat(variables('vnetID'),'/subnets/',variables('subnetName'))]"
60  },

```

- ★ Variables, in general, represent static values that are already present in the template, in contrast to parameter values which are passed in at execution time. Variables can also be used to simplify template language expression by using expression and functions, such as "[resourceId...]" and "[concat...]", as highlighted above. In the case of the concat function, variables can be composed of other variables or parameter values that are input at run time. ResourceId is a helper function that is used to get unique IDs of resources.

12. Scroll down to the resources section that begins at around line 61.

```

61  "resources": [
62    {
63      "type": "Microsoft.Storage/storageAccounts",
64      "name": "[parameters('newStorageAccountName')]",
65      "apiVersion": "2015-05-01-preview",
66      "location": "[variables('location')]",
67      "properties": {
68        "accountType": "[variables('storageAccountType')]"
69      }

```

## Deploy Azure Resources Using ARM Templates

- ✦ In the resources array section, you define the resources that you wish to deploy or update. Each resource must include a type, name, and apiVersion. The type value is the namespace of the resource provider and the supported resource type. The value for the type is a substring of the URL you would use if you wanted to create a resources, such as a storage account, against the Azure REST (representational state transfer) API (application programming interface). When the template is deployed, the template engine makes the same HTTP put calls that would be made if you were directly interacting with the API.
- ✦ If you wish to find out the values and the properties associated with a resource, you can find the documentation on the Azure REST APIs here: <https://msdn.microsoft.com/en-us/library/azure/mt420159.aspx>. For example, to find out more information about the Microsoft.Storage/storageaccounts resource, you can consult the Azure Storage Provider REST API reference at <https://msdn.microsoft.com/en-us/library/azure/mt163683.aspx>.
- ✦ Optional key/value pairs for resource objects include location (where you define the geo-location for the resource), properties (where you define resource-specific settings), tags (that provide metadata associated with resource), resources (child resources that depend on the resource being defined), and dependsOn (resources that the resource being defined depends on).
- ✦ Scroll down to the beginning of the virtual machine resource definition, which begins at around line 131.

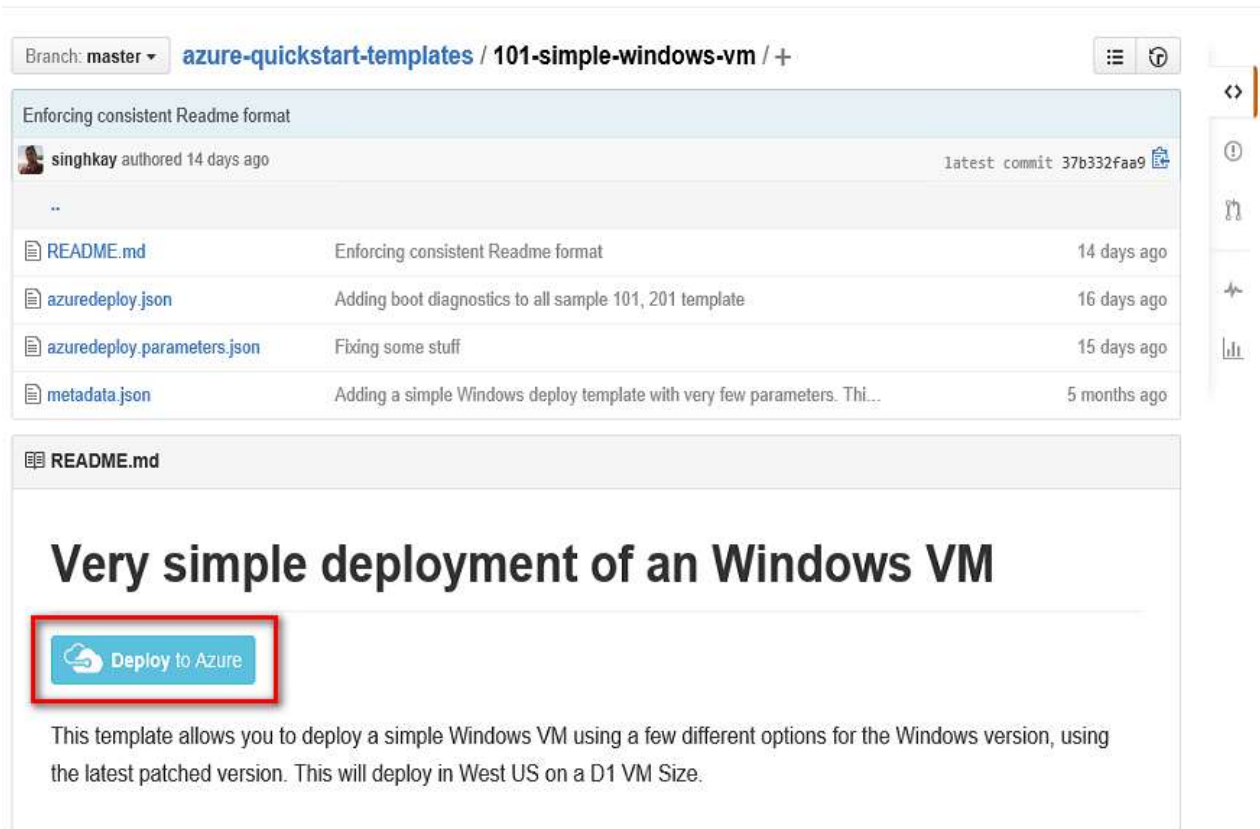
```
131     "apiVersion": "2015-06-15",
132     "type": "Microsoft.Compute/virtualMachines",
133     "name": "[variables('vmName')]",
134     "location": "[variables('location')]",
135     "dependsOn": [
136         "[concat('Microsoft.Storage/storageAccounts/', parameters('newStorageAccountName'))]",
137         "[concat('Microsoft.Network/networkInterfaces/', variables('nicName'))]"
138     ]
139 }
```

- ✦ Note the "dependsOn" key/value pair. Before the virtual machine can be created, the storage account and the network interface, which in turn has dependencies of its own (see lines 105 -111), must exist. The script is evaluated for dependencies to create objects in the appropriate order. If no dependencies are specified, Azure Resource Manager attempts to create them in parallel. This can result in much faster deployments for large and complex infrastructures over the Azure Service Manager mode.

13. In the Microsoft Edge Browser, click back to return to the previous page.

14. On the 101-simple-windows-vm page, click **Deploy to Azure**.

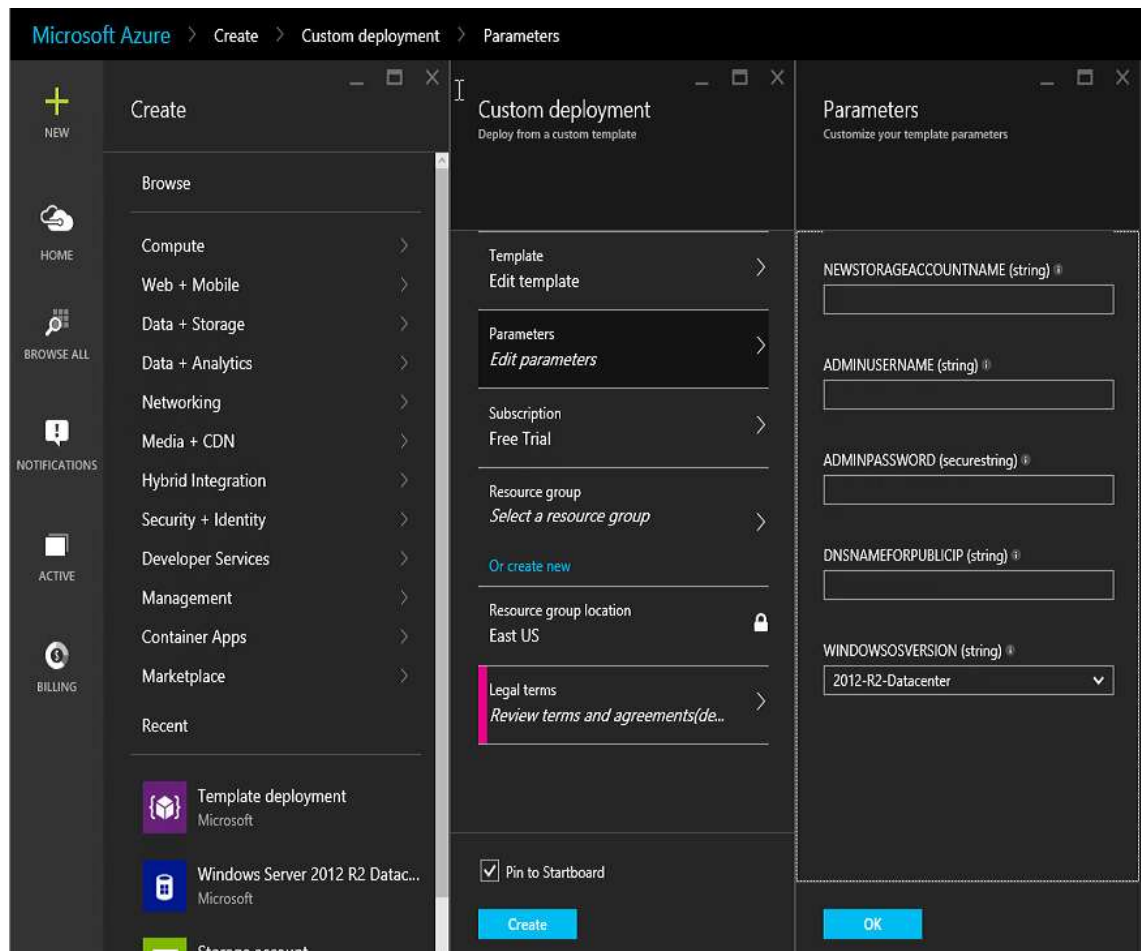




15. On the Microsoft Azure sign in page, enter the email address and password associated with your subscription, and sign in.

- ✦ The Azure Portal opens showing the Custom deployment and Parameters blades.
- ✦ Note that the Parameters blade provides fields for text input. In the case of the WINDOWSOSVERSION parameter, note the presence of a default value and a drop-list of allowed values.

## Deploy Azure Resources Using ARM Templates



16. On the Custom deployment blade, click **Edit template**.
  - ★ You can edit the template before deploying it, for example, to change variable names or modify resources that the template describes for deployment.
17. Close the Edit template blade.
18. In the Custom deployment blade, in Resource group, click **Or create new**.
19. In the Create a new resource group text box, type **test**.
20. Click **Resource group location**.
  - ★ Note that once you enter a name for a new resource group, you can also specify a location.
21. Review the locations, and then close the Location blade.
22. Close the Custom deployment blade.
  - ★ You are not going to deploy the template at this time. Rather, you will modify the template in a subsequent task and then explore other ways of deploying the template.
23. If prompted by a warning indicating your unsaved edits will be discarded, click **OK**.

24. Leave the Azure Portal open for a subsequent lab exercise.

## Configure Git repository and Visual Studio Code

As you saw in the previous exercises, GitHub is used to store ARM templates and other code-related objects. GitHub is a web-based Git repository to ensure availability and provide a version control system for files.

Version control systems record changes to files or a set of files so that they can be retrieved at a later time. There are different kinds of version control systems, the simplest being the manual use of multiple folders to store different versions of your files.

More sophisticated and fault tolerant version control systems use central or distributed databases to store the files and their changes.

Git is a distributed version control system in which clients clone entire repositories to ensure that there is no single point of failure. If a particular repository fails, the contents can be restored from a cloned copy that exists elsewhere.

In this exercise, you will learn more about GitHub and Git repositories. You will also learn how to configure Visual Studio Code to use Git.

### Create GitHub account and public repository

The remaining lab exercises require that you have a GitHub account and repository named Templates.

In this task, you create a GitHub account and then configure a public repository named Templates.

- ✦ This lab exercise is optional if you already have a GitHub account.
- ✦ Note that, if you do have a GitHub account and you have enabled 2-factor authentication for it, you may have difficulties authenticating to your account when using Visual Studio Code. If you have 2-factor authentication enabled on your Git repository, consider creating a Git account for this lab.
- ✎ Perform the following tasks on AZRCAMP-ADMIN logged on as **Contoso\Administrator** using **Passw0rd!** as the password:
  25. On AZRCAMP-ADMIN, open Microsoft Edge.
  26. Navigate to <https://github.com>
  27. On the home page, click **Sign up**.
  28. On the Join GitHub page, enter a username, email address, and password.
    - ✦ Please note that when you perform certain operations on GitHub, such as commits, your email address is publicly viewable. For more information, please see <https://help.github.com/articles/keeping-your-email-address-private/>.
  29. Click **Create an account**.
  30. On the Welcome to GitHub page, click **Finish sign up**.

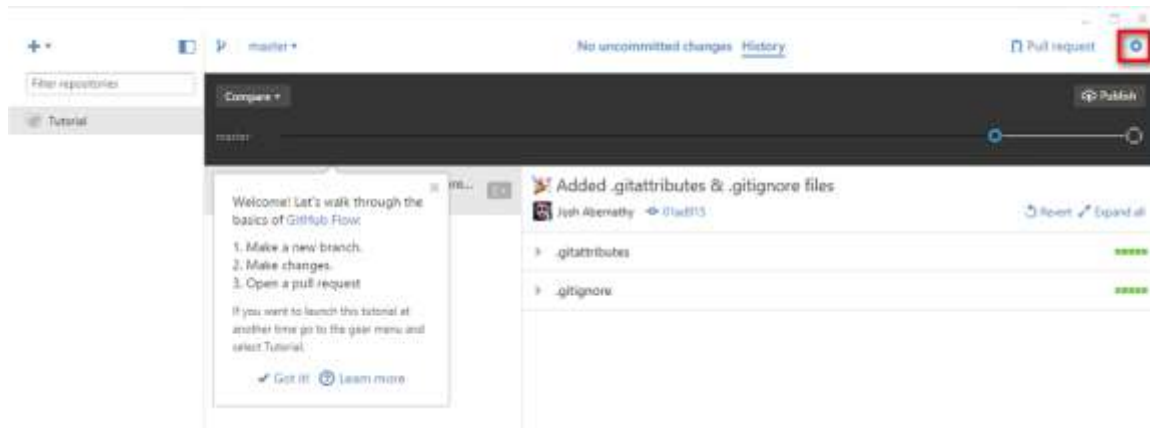
31. On the GitHub Bootcamp page, click **Create a repository**.
32. Open your email client and open the verification email sent to you by the GitHub web site.
33. Click **Verify email address**.
34. Switch to the GitHub page.
35. Click **Create a repository** (alternatively, in the upper right corner, click +, and then click **New repository**).
36. In Repository name, type **Templates**, select **Initialize this repository with a README**, and then click **Create repository**.
  - ★ You will clone the repository in the next task.
37. Leave the Microsoft Edge browser open for the next task.

### Configure GitHub desktop and clone repository

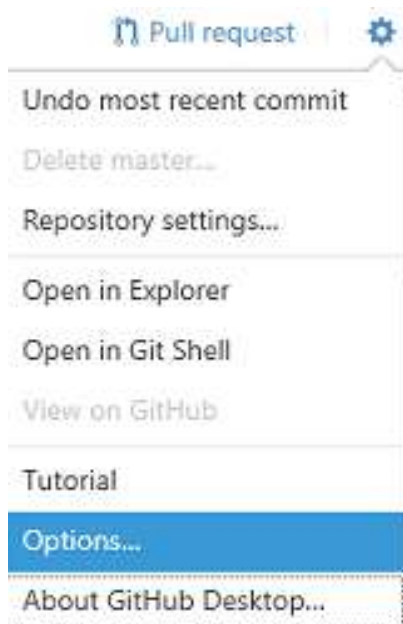
In this task, you will launch GitHub, configure settings for the application, and then clone the Templates repository you configured.

- ✎ Perform the following tasks on AZRCAMP-ADMIN logged on as **Contoso\Administrator** using **Passw0rd!** as the password:
  1. Switch to the desktop of the AZRCAMP-ADMIN
  2. On the desktop, double-click **GitHub**.
    - ★ The GitHub desktop application was installed as part of the lab setup.
  3. On the Welcome page enter your GitHub username and password, and then click **Log in**.
  4. Click **Continue**.
  5. Under No local repositories found, click **dashboard**.
    - ★ The GitHub desktop application opens showing a pre-configured Git repository named Tutorial that contains a .gitattributes and .gitignore files. Before proceeding with the next steps, feel free to launch the tutorial by clicking Got it!. Otherwise, close the tutorial.
  6. In the upper right corner, click **Settings** (the gear icon).

## Deploy Azure Resources Using ARM Templates

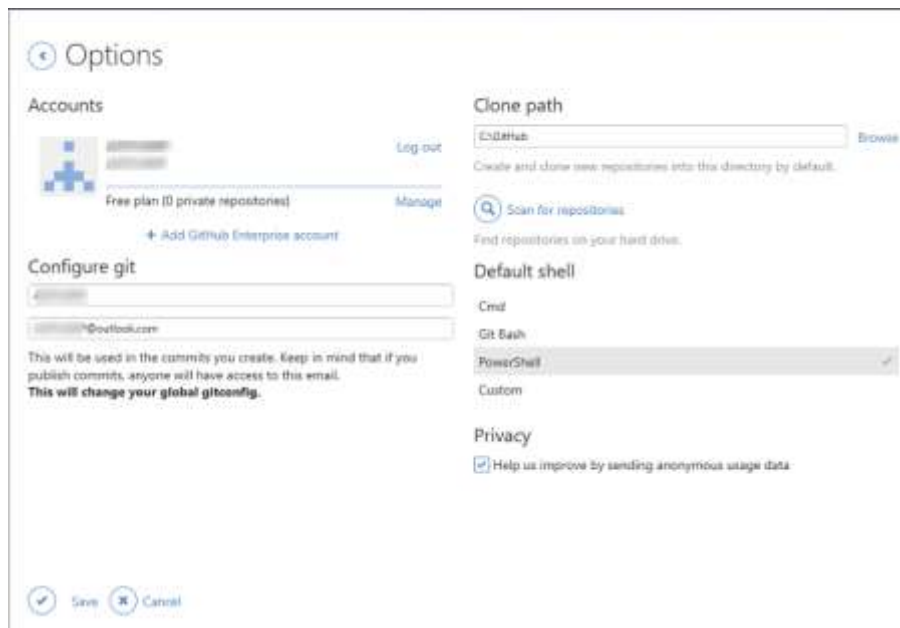


7. Click **Options**.

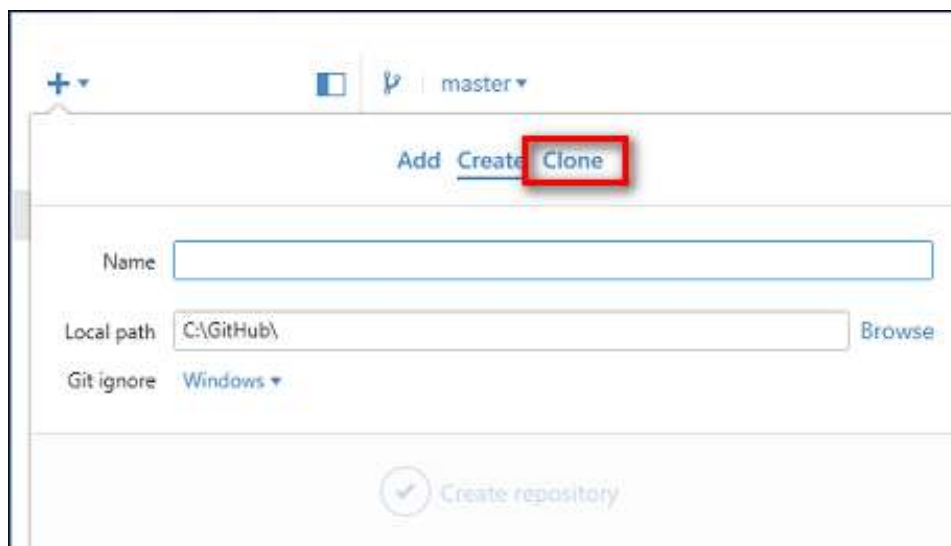


8. On the Options page, in Clone path, enter **C:\GitHub**, and then click **Save**.

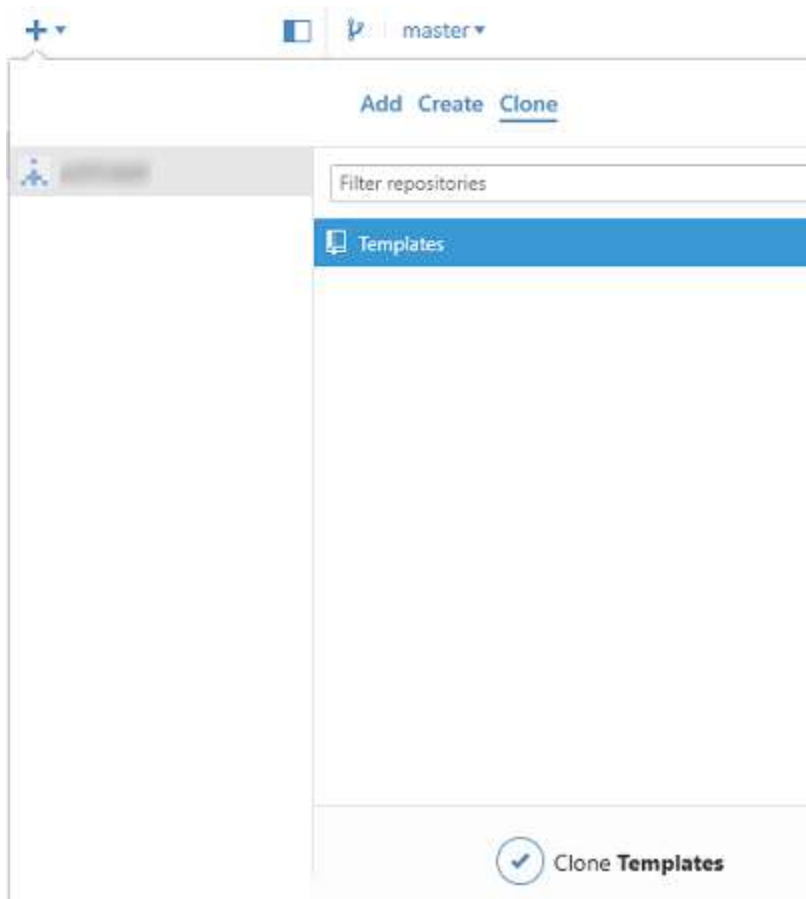
## Deploy Azure Resources Using ARM Templates



9. On the GitHub home page, in the upper left corner, click **Add** (plus sign).



10. Click **Clone**.
11. Click **Templates**, and then click **Clone Templates**.



12. In the Browse for Folder dialog box, ensure that **C:\GitHub** is selected, and then click **OK**.
  - ✦ If you receive an error, ensure that the Tutorial repository is closed/removed and perform step 12 again.
13. On the desktop, double-click **Git-Shell**.
  - ✦ A PowerShell window opens. The option to integrate the Git commands with PowerShell is configured in the GitHub client.
14. At the PowerShell prompt, type the following command and press ENTER.

```
↩ Git config --global credential.helper wincred
```

  - ✦ This command is necessary to ensure that Visual Studio code can authenticate against the GitHub web site. The command is case-sensitive.
15. At the PowerShell prompt, type the following command and press ENTER.

```
↩ Git config --list
```

  - ✦ This command shows the settings that are configured for Git. Note the bottom of the output displays your GitHub user name and email address. These settings were configured when you logged into the GitHub Windows client. Normally, you would configure Git using commands from a shell, such as Bit



## Deploy Azure Resources Using ARM Templates

Bash or the integrated PowerShell prompt you see in the lab. For more information on configuring and using Git, please see <http://git-scm.com/doc>.

16. Open File Explorer, and browse to **C:\LabFiles\AZITPROCamp\Lab01-Deploy Azure resources using templates**.

17. Select and right-click **101-simple-windows-vm**, and then click **Copy**.

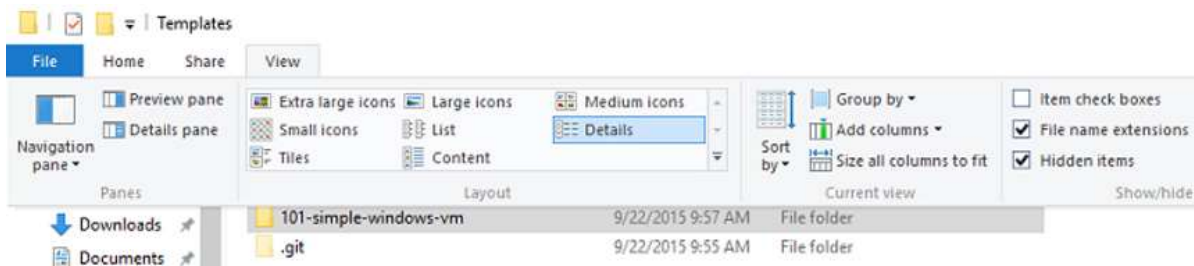
★ For your convenience, the Azure Quickstart template you examined in the previous exercise was previously downloaded as part of the lab set up.

18. In File Explorer, navigate to **C:\GitHub\Templates**.

19. Press CTRL-V to paste the folder to **C:\GitHub\Templates**.

20. In File Explorer, on the Ribbon, click **View**, and then select **Hidden items**.

★ The hidden .git folder appears. This folder is created upon initialization of Git repository. The .git folder includes Git database and other files necessary for staging and committing changes to files for source version control.



21. Close File Explorer.

## Examine Visual Studio Code integration with Git and push commits to remote repository

Visual Studio Code has integrated support for the most common Git commands, for example, to stage and commit changes.

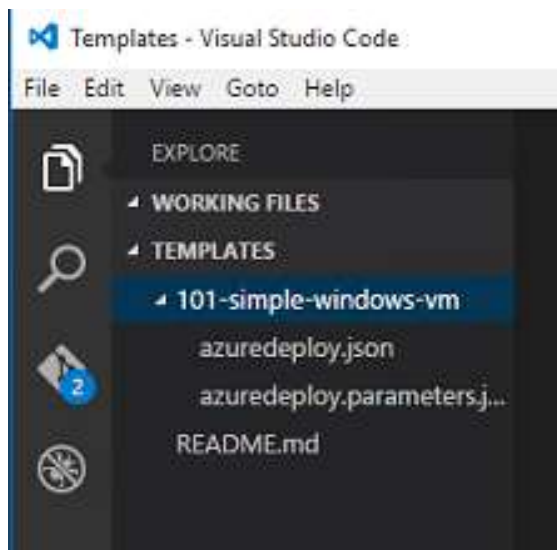
In this task, you will see a demonstration of Visual Studio Code integration with Git and push commits (changes you wish to be permanent) to the remote repository on GitHub.

✍ Perform the following tasks on AZRCAMP-ADMIN logged on as **Contoso\Administrator** using **Passw0rd!** as the password:

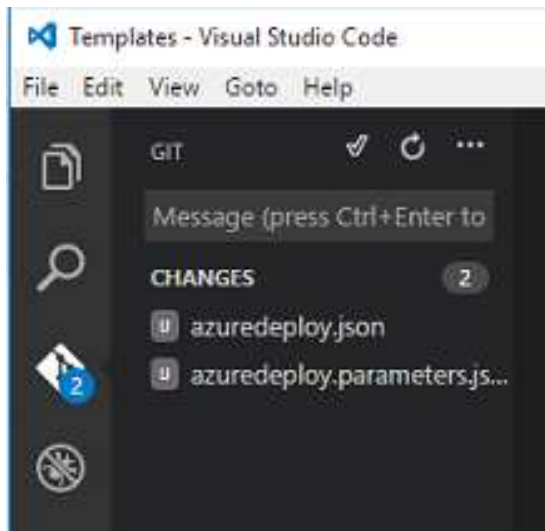
1. On the Desktop or from the taskbar, open **Visual Studio Code**.



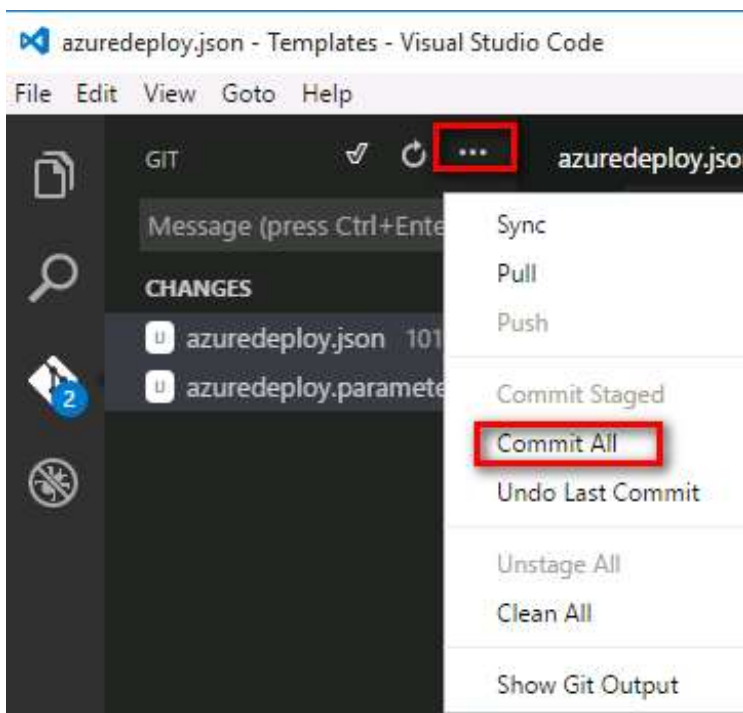
2. If you see a message indicating that you need to install Visual Studio Code, please ignore the message (or click Later) and complete the steps below.
3. In Visual Studio Code, click **File**, and click **Open Folder**.
4. In the Select Folder dialog box, navigate to **C:\GitHub**, click **Templates**, and then click **Select Folder**.
  - ✦ The folder opens and Visual Studio Code provides a visual indication that Git detects uncommitted files in the folder. These are the files you copied to the folder earlier.
  - ✦ You may have to close and reopen the folder to see the uncommitted indication as per the screen shot below. To close the folder, click File, and then click Close Folder.



5. Click the **Git icon** on the left.



6. Press the CTRL key down, and select both files.
7. Click the ellipsis, and then click **Commit All**.
  - ★ Git files have three states: modified, staged, and committed. Modified means that the files are in a workspace and have been changed but have not yet been committed to the database in the repository. Staged means the files have been moved to a staging area prior to being committed, for example, as a holding area pending review. Committed means that a snapshot of the file is committed to the database.



## Deploy Azure Resources Using ARM Templates

8. In Message, type **Initial Version**, and press CTRL+ENTER.

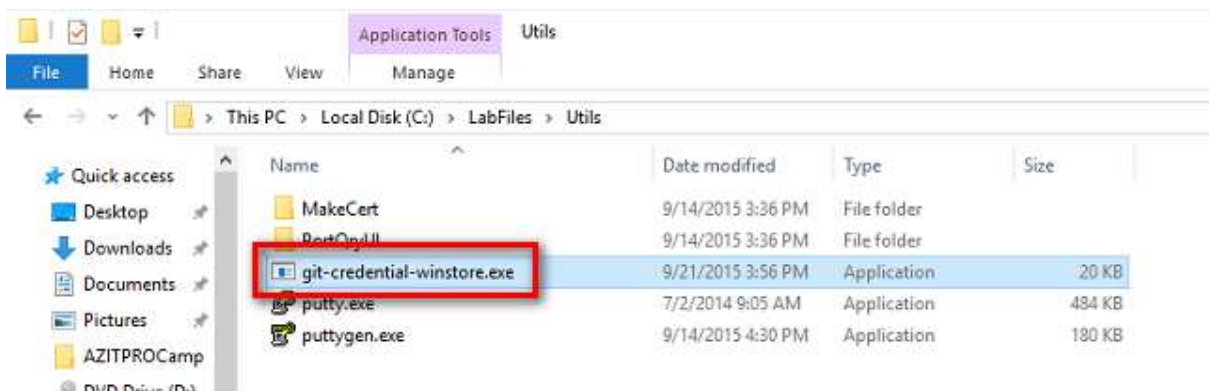
★ These files will represent a starting point for changes you will make in a subsequent exercise.



★ The changes are committed to the local repository. The files disappear from view in the Git node. In the next step, you will push the committed files to the remote repository.

9. Click the ellipses, and then click **Push**.
10. When prompted by Visual Studio Code dialog box, enter your Git username and password, and click **OK**.

★ The authentication fails. This is expected.
11. When prompted with an error message indicating that Authentication failed on the git remote, click **Close**.
12. Open File Explorer, and browse to **C:\LabFiles\Utils** and double-click **git-credential-winstore.exe**.



13. When prompted, click **Yes**.

## Deploy Azure Resources Using ARM Templates

- ✦ This small executable downloaded from [Codeplex.com](https://codeplex.com) resolves an issue with authentication to GitHub using VSCode.
- ✦ If you have enabled multi-factor authentication on your GitHub repository, this workaround may not be effective.

14. Switch to Visual Studio Code, click the ellipses, and then click **Push**.

15. When prompted by the Git Credentials dialog box, enter your Git username and password, and click **OK**.

- ✦ The committed files in the local repository are pushed to the repository on GitHub.

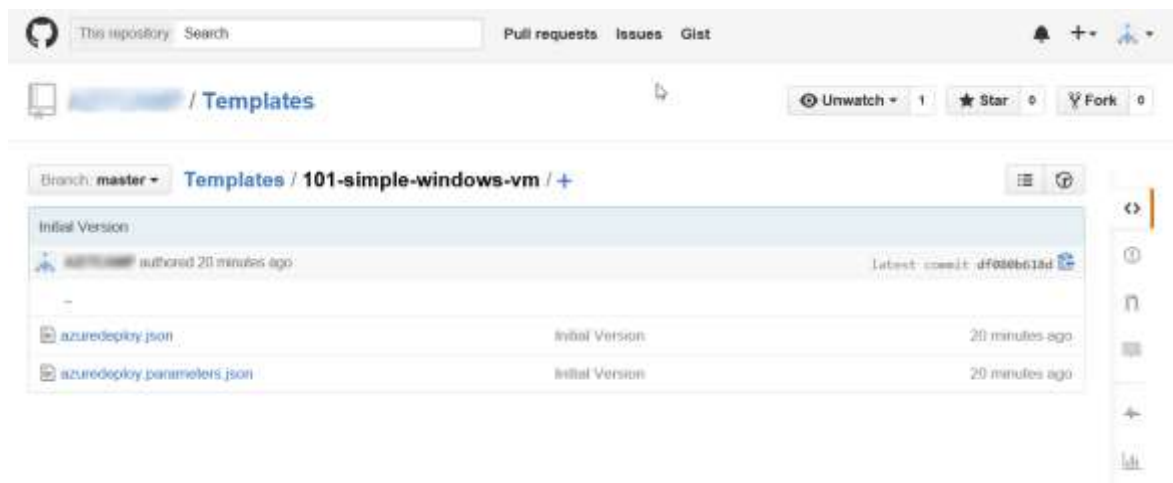
16. Leave Visual Studio Code open for subsequent lab steps, and switch to the instance of Microsoft Edge that you left open in a previous task.

17. Ensure that you are viewing the Templates page for your repository, and then press F5 to refresh the page.

- ✦ You should see the 101-simple-windows-vm folder from your local repository.

18. Click **101-simple-windows-vm**.

- ✦ You should see the two JSON files in the folder.



19. Leave the browser open for subsequent steps in the next exercise.

## Modify and Deploy Azure Quickstart Templates

### Introduction

As you learned in the previous exercise, it is rarely necessary to construct a JSON-based Azure Resource Manager template from scratch. In many instances, you will be able to find a sample template to use as a starting point for your own template. A basic understanding of how templates are constructed, combined with an appropriate sample template to use as a starting point, will allow you to create your own custom templates with relative ease.

For additional information on authoring and modifying template files, please see Authoring Azure Resource Manager templates at <https://azure.microsoft.com/en-us/documentation/articles/resource-group-authoring-templates/>.

In this exercise, you will build on the understanding that you acquired in a previous exercise to customize a sample template and then deploy it.

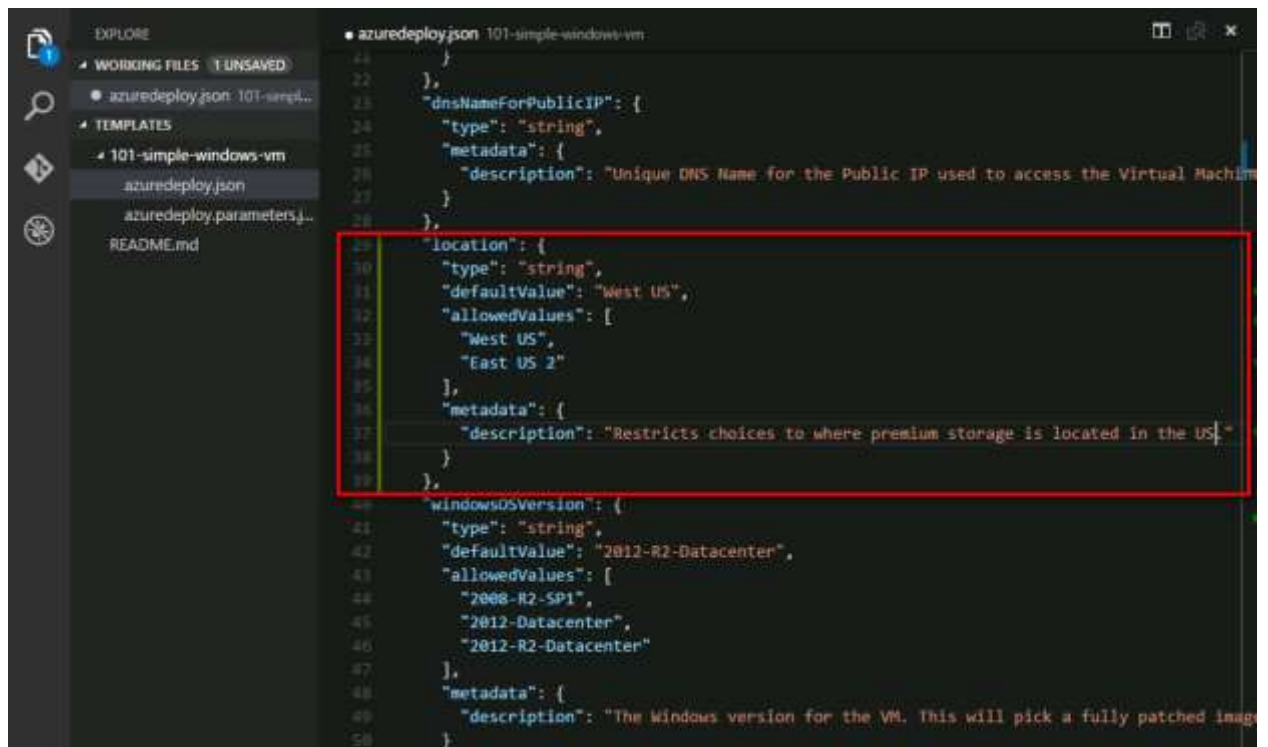
### Modify sample template and parameter JSON files

In a previous exercise, you examined the 101-simple-windows-vm template that is available on GitHub. You subsequently added this template to your own Git repository. In this task, you will customize this template and its related parameters JSON file using Visual Studio Code to meet your specific and additional requirements. Also, you will stage and commit your changes to the local and remote Git repository.

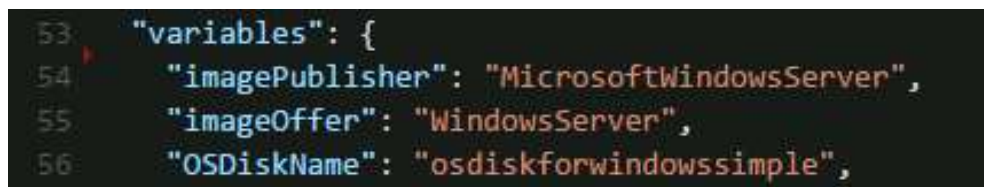
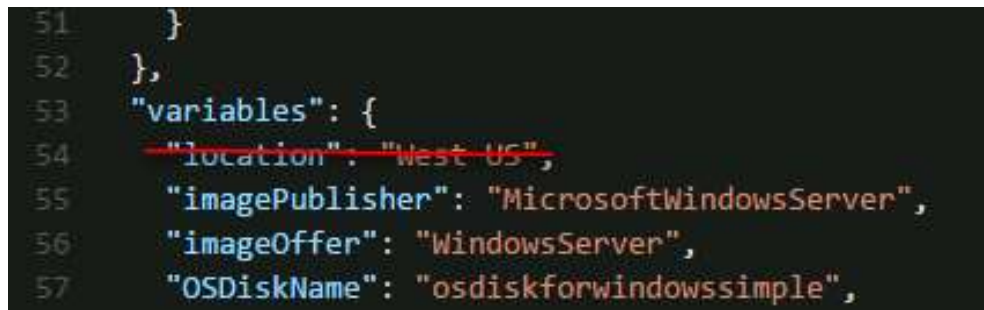
 Perform the following tasks on AZRCAMP-ADMIN logged on as **Contoso\Administrator** using **Passw0rd!** as the password:

1. If not already open, open Visual Studio Code, and navigate to the **C:\GitHub\Templates\101-simple-windows-vm** folder.
2. In the tree pane, click **azuredeploy.json**.
3. Type the following code below line 28.
  - ✦ Note that intellisense and autocomplete make the task of entering JSON easier.
  - ✦ Recall the parameters are passed to the template at runtime and allow users the ability to specify values.
  - ✦ In this case, you are creating a default value and restricting choice to locations where Premium storage is available. For a list of datacenters showing the services offered by each, please see <http://azure.microsoft.com/en-us/regions/#services>.

## Deploy Azure Resources Using ARM Templates



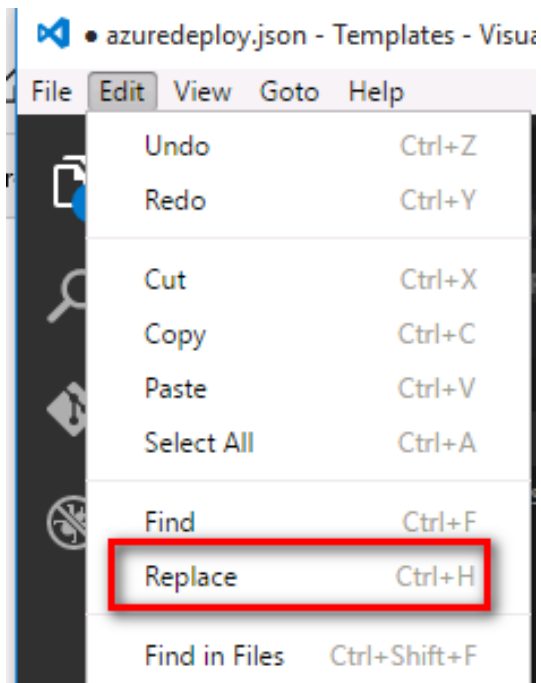
4. Scroll down to the variables section, and delete the **"location": "West US"** key/value pair.



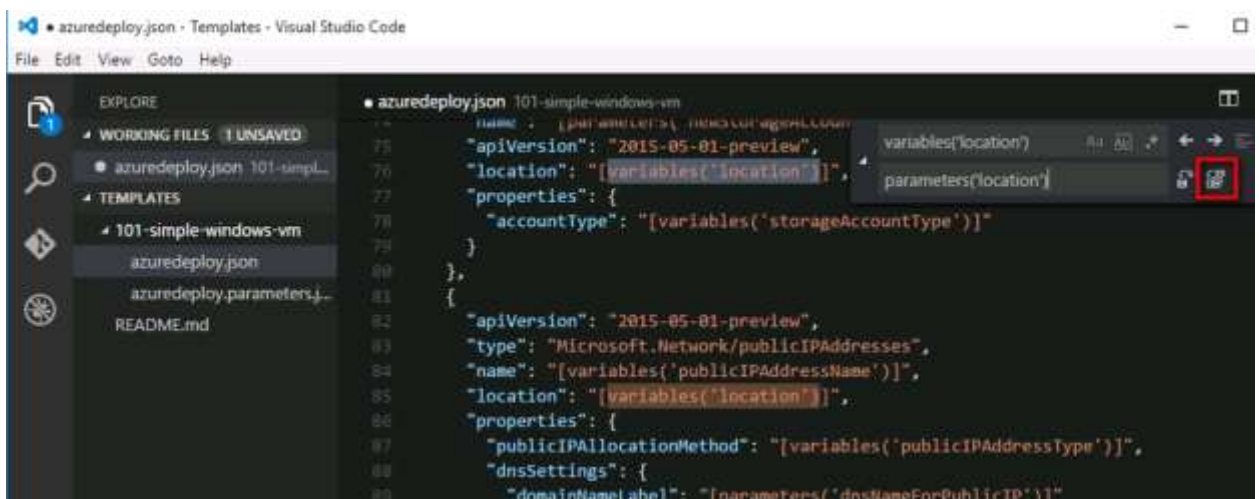
5. Click **Edit** and then click **Replace**.



## Deploy Azure Resources Using ARM Templates



6. In Find, type `variables('location')`; in Replace, type `parameters('location')`, and then click **Replace All**, as shown below.



7. Scroll to the end of the parameters section and the start of the variables section.
8. At the end of the second-last closing brace before the variables section, add a comma, as shown below.



```

49     "description": "The Windows version for the VM. T
50   }
51   },
52 },
53 "variables": {
54   "imagePublisher": "MicrosoftWindowsServer",
55   "imageOffer": "Windows Server",

```

9. Then, add the following lines of code to create a parameter for size of the data disk.

★ Tip: You can find the completed solution in the C:\LabFiles\AZIT{ROCamp\Solutions folder. If you like, you can copy and paste code from the solution into your JSON script.

```

51   },
52   "sizeOfEachDataDiskInGB": {
53     "type": "string",
54     "metadata": {
55       "description": "Size of each data disk in GB"
56     }
57   }
58 },
59 "variables": {
60   "imagePublisher": "MicrosoftWindowsServer",
61   "imageOffer": "Windows Server",

```

10. Scroll to the end of the variables section, and add a comma at the end of the subnetRef key/value pair:

```

"subnetRef": "[concat(variables('vnetID'), '/subnets/', variables('subnetName'))]",

```

11. Then, type the following on a single line at the end of the variables section:

```

"dataDisk1VhdName": "[concat('http://', parameters('newStorageAccountName'), '.blob.core.w
indows.net/', variables('vmStorageAccountContainerName'), '/', variables('vmName'), 'dataDisk1.vhd')]"

```

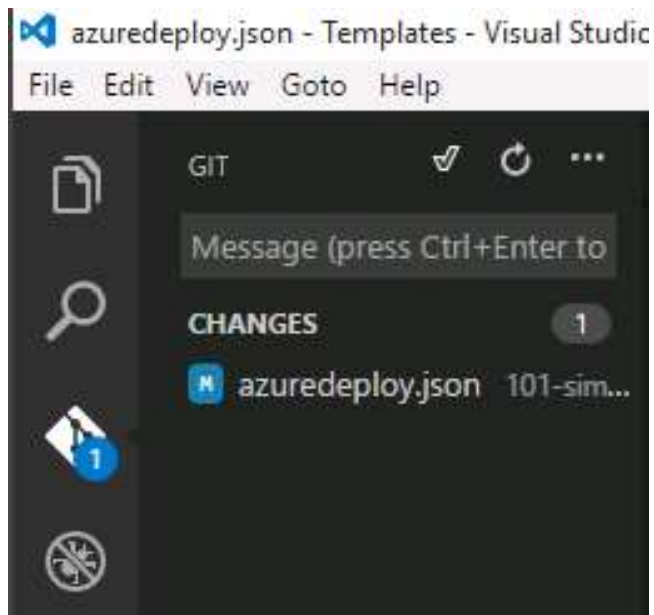
★ You can find the completed solution in the C:\LabFiles\AITPROCamp\Solutions folder.

12. Scroll down to the resources section and locate "osDisk" section.
13. Between the "osDisk" and "networkProfile" section, add the following resource:

```
171     },
172     "osDisk": {
173       "name": "osdisk",
174       "vhd": {
175         "uri": "[concat('http://',parameters('newStorageAccount'),
176       },
177       "caching": "ReadWrite",
178       "createOption": "FromImage"
179     },
180     "dataDisks": [
181       {
182         "name": "datadisk1",
183         "diskSizeGB": "[parameters('sizeOfDiskInGB')]",
184         "lun": 0,
185         "vhd": {
186           "Uri": "[variables('dataDisk1VhdName')]"
187         },
188         "createOption": "Empty"
189       }
190     ]
191   },
192   "networkProfile": {
193     "networkInterfaces": [
194       {
195         "id": "[resourceId('Microsoft.Network/networkInterfaces',
196       }
```

- ❖ Be very careful where you place this code section. Before proceeding to the next step, scroll to the end of the file and double-check for any red squiggles that might indicate improper placement of this snippet.

14. Click **File** and then click **Save**.
15. Click the Git icon on the left.



16. Under CHANGES, click **azuredeploy.json**.

✦ Two screens appear. The left screen shows the original file; the right screen shows the changes.

17. Review your changes, right-click **azuredeploy.json**, and then click **Stage**.

18. On the left, click the Explore icon.

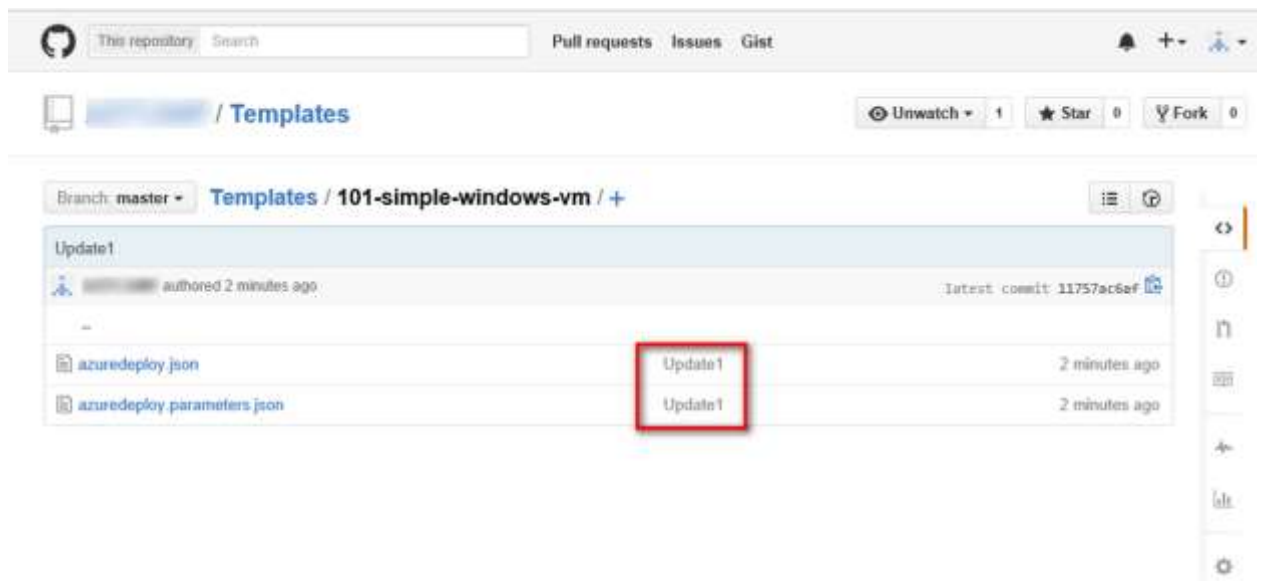
19. Click **azuredeploy.parameters.json**.

20. In **azuredeploy.parameters.json**, add key/value pairs for the **location**, **WindowsOSVersion**, and **sizeOfDiskInGB** parameters and provide unique names for the storage account and DNS name, as follows:

## Deploy Azure Resources Using ARM Templates

```
● azuredeploy.parameters.json 101-simple-windows-vm
1 {
2   "$schema": "http://schema.management.azure.com/schemas/2015-01-01/deploymentParameters.json#",
3   "contentVersion": "1.0.0.0",
4   "parameters": {
5     "newStorageAccountName": {
6       "value": "unique-name-you-recorded-in-exercise-2"
7     },
8     "adminUsername": {
9       "value": "admin123"
10    },
11    "adminPassword": {
12      "value": "Password!"
13    },
14    "dnsNameForPublicIP": {
15      "value": "your-initials-plus-4-or-more-random-integers-to-ensure-name-uniqueness"
16    },
17    "location": {
18      "value": "West US"
19    },
20    "sizeOfDiskInGB": {
21      "value": "20"
22    }
23  }
24 }
```

21. Save the azuredeploy.parameters.json file.
22. Click the Git icon in the navigation pane, right-click **azuredeploy.parameters.json**, and click **Stage**.
23. Click the ellipses, and then click **Commit Staged**.
24. In the Message box, type **Update1**, and press CTRL+ENTER.
  - ✦ A snapshot of the files is now committed to the local repository.
25. Click the ellipses, and then click **Push**.
  - ✦ Note that you are not prompted for credentials. You configured credential caching earlier with the **git config --global credential.helper wincred** command.
26. Switch to the Microsoft Edge browser you left open in the previous exercise and press F5 to refresh the page.
  - ✦ You should see that the two JSON files have been pushed to the remote repository.



27. Leave the browser open for the next task.

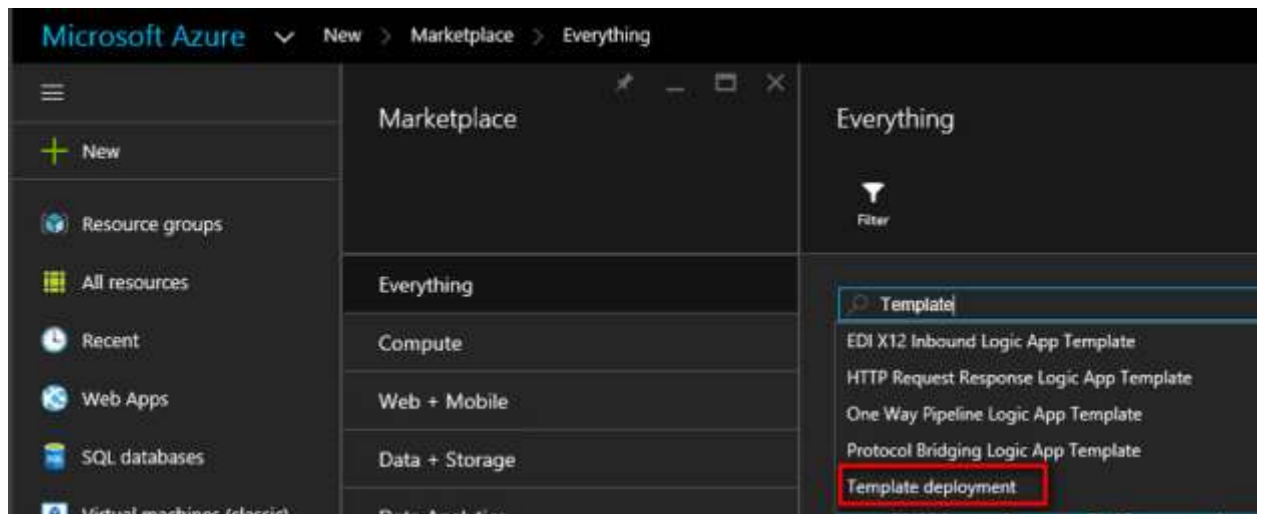
### Deploy custom JSON files

In the previous task, you modified the sample JSON to meet specific requirements and subsequently committed your changes to both a local and remote Git repository. In this task, you will examine deploying the custom template using the Azure portal and Windows PowerShell.

- ✎ Perform the following tasks on AZRCAMP-ADMIN logged on as **Contoso\Administrator** using **Passw0rd!** as the password:
  1. If not already open, open Visual Studio Code, and navigate to the **C:\GitHub\Templates\101-simple-windows-vm** folder.
  2. In the tree pane, click **azuredeploy.json**.
  3. Click anywhere in the details pane, press CTRL+A and then press CTRL+C to copy the contents of the file to the clipboard.
  4. Switch to Microsoft Edge.
  5. If not already logged on to the Microsoft Azure portal, open a new tab, navigate to <https://portal.azure.com>, and log on the account you are using for your Azure subscription.
  6. In the Microsoft Azure portal, in the navigation pane, click **New**.
  7. In the New blade, click **Marketplace**.

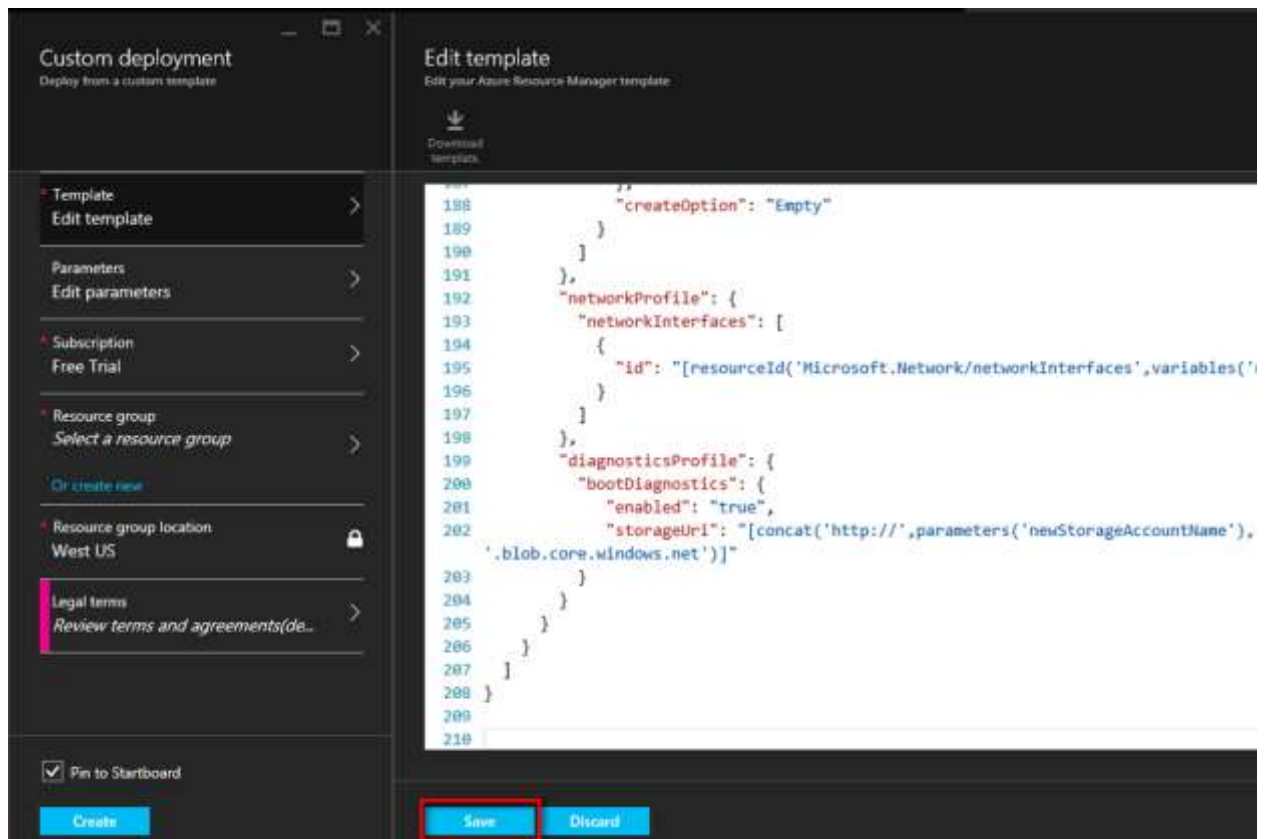
## Deploy Azure Resources Using ARM Templates

8. In Search everything text box, type **Template**, click **Template deployment** from the drop-down list, and press ENTER.



9. Double-click **Template deployment** in the results.
10. In the Template deployment blade, click **Create**.
11. The Custom deployment blade appears.
12. In the Custom deployment blade, click **Edit template**.
13. In the Edit template blade, delete all the lines of JSON script.
14. Place the cursor in the template area, and press CTRL+V to paste the JSON script that you copied to the clipboard earlier.
15. Click **Save**.

## Deploy Azure Resources Using ARM Templates

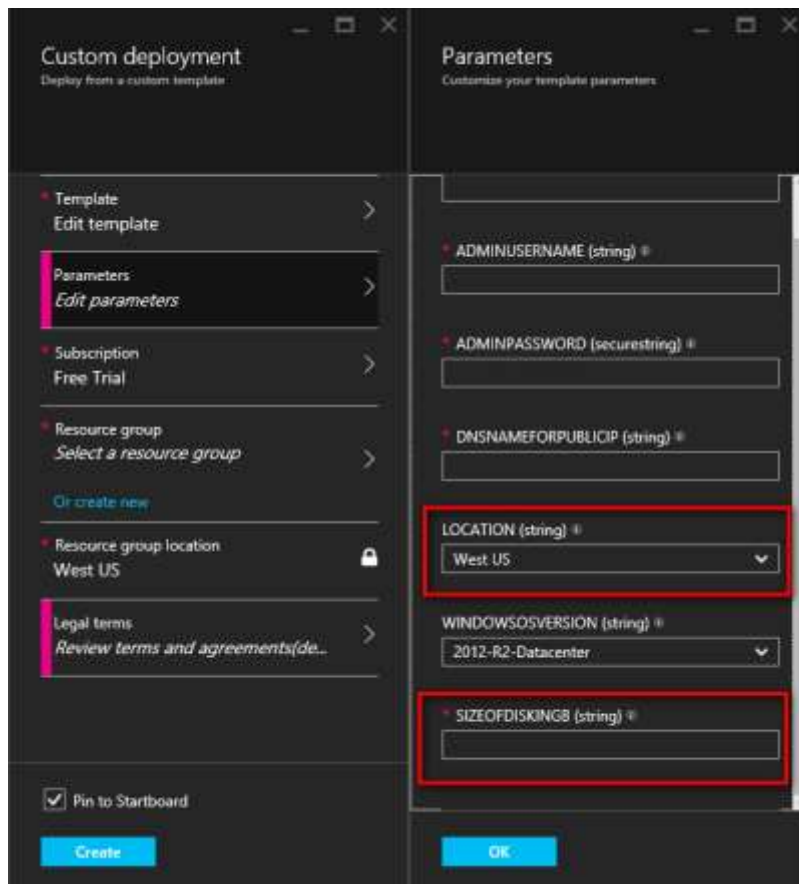


### 16. Click **Edit parameters**.

- ★ The Parameters blade should appear immediately. If it does not, this means that there is something wrong with your JSON script, most likely a mismatch between opening and closing braces or brackets.
- ★ If the template has the proper formatting, you will see small green rectangle in the upper right side of the template; otherwise, you will see small red squares on the right side that provide an approximate location of the source of the problem.

### 17. In the parameters blade, note the presence of the LOCATION (string) and SIZEOFDISKINGB (string) parameters that you added to the original JSON script.

## Deploy Azure Resources Using ARM Templates



18. Close the Parameters and the Custom deployment blades without creating the deployment.

- ✦ You are going to deploy your custom template using a PowerShell command in a later step. You examined how to deploy a custom template in the Azure portal as a demonstration and to do a final verification on your template.

19. When prompted, click **OK** to discard unsaved edits.

20. Close all open blades in the portal to return to the Start page.

21. Leave Microsoft Edge open for subsequent steps.

22. Switch to the Windows PowerShell ISE console you left open in a previous exercise.

- ✦ If you closed the console, please open it and run the `C:\LabFiles\AZITPROCamp\Lab01.\Lab01Start.ps1` script again.

23. In the command pane, type the following command on a single line, and press ENTER.

```
↩ New-AzureResourceGroupDeployment -DeploymentName "Simple-VM"  
-ResourceGroupName RG-AZITCAMP -TemplateFile  
"C:\GitHub\Templates\101-simple-windows-vm\azuredeploy.json"
```

- ✦ You will be prompted for the storage account name, as shown below. If you do not wish to be prompted for required parameters, you can add a template parameter file to the command. You will do

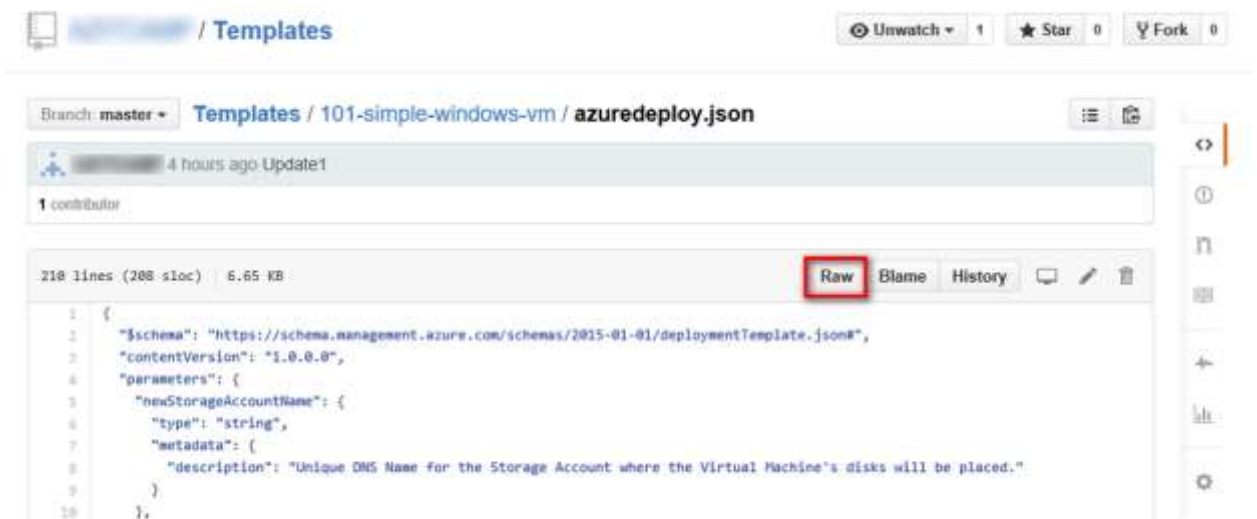


## Deploy Azure Resources Using ARM Templates

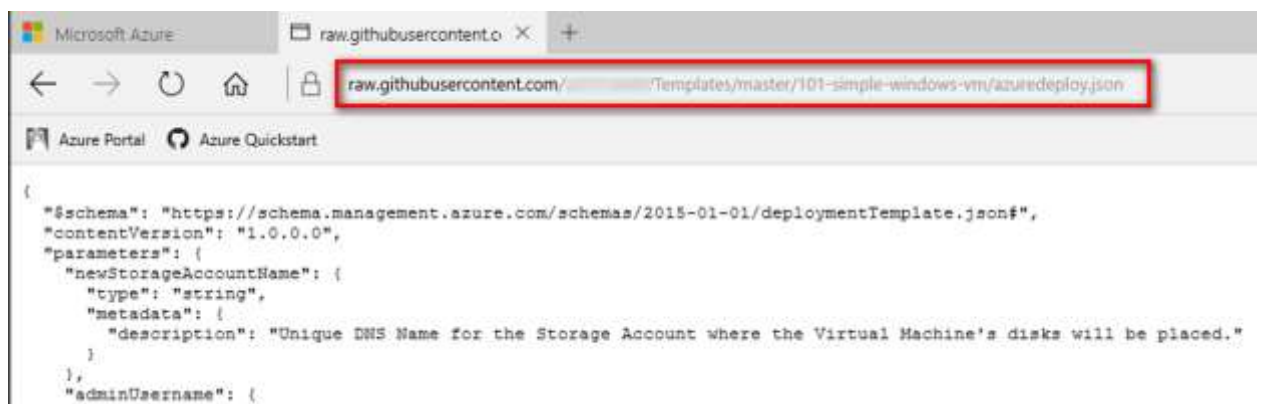
this in the next steps. However, you will use the version of the JSON scripts that are available in your remote repository.

```
PS C:\> New-AzureResourceGroupDeployment -Name "Simple-VM" -ResourceGroupName RG-AZITCAMP -TemplateFile "C:\GitHub\Templates\101-simple-windows-vm\azuredeploy.js
cadlet New-AzureResourceGroupDeployment at command pipeline position 1
Supply values for the following parameters:
(Type ? for Help.)
newStorageAccountName:
```

24. In the PowerShell command pane, press CTRL+C to break out of the command.
25. Switch to Microsoft Edge.
26. In the tab that displays the contents of your GitHub Templates repository, click **azuredeploy.json**.
27. On the azuredeploy.json page, click **Raw**.



28. Copy the URL displayed for the raw version of the azuredeploy.json file to the value of the \$templatefileURI on line 57 of the Lab01Start.ps1 script.



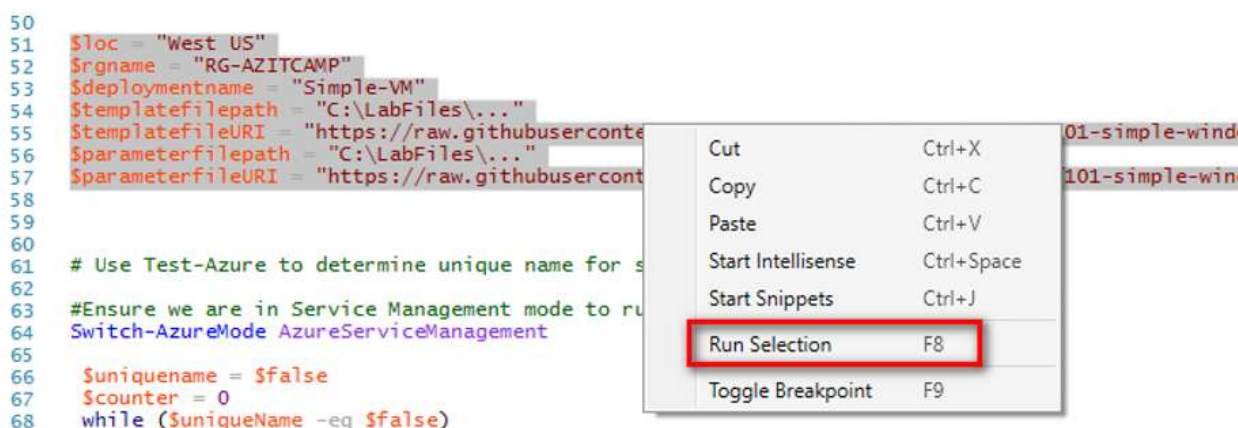
29. Repeat steps 27 - 29 to copy the URL for the azuredeploy.parameters.json file to the value for the \$parameterfileURI variable on line 59 of the Lab01Start.ps1 script.
30. The resulting section of the script should look like something this:

```

49 Select-AzureSubscription -SubscriptionName $subname
50
51 $loc = "West US"
52 $rgname = "RG-AZITCAMP"
53 $deploymentname = "Simple-VM"
54 $templatefilepath = "C:\LabFiles\..."
55 $templatefileURI = "https://raw.githubusercontent.com/.../Templates/master/101-simple-windows-vm/azuredeploy.json"
56 $parameterfilepath = "C:\LabFiles\..."
57 $parameterfileURI = "https://raw.githubusercontent.com/.../Templates/master/101-simple-windows-vm/azuredeploy.parameters.json"
58

```

31. Select the entire block of variables and click Run Selection.



```

50
51 $loc = "West US"
52 $rgname = "RG-AZITCAMP"
53 $deploymentname = "Simple-VM"
54 $templatefilepath = "C:\LabFiles\..."
55 $templatefileURI = "https://raw.githubusercontent.com/.../Templates/master/101-simple-windows-vm/azuredeploy.json"
56 $parameterfilepath = "C:\LabFiles\..."
57 $parameterfileURI = "https://raw.githubusercontent.com/.../Templates/master/101-simple-windows-vm/azuredeploy.parameters.json"
58
59
60
61 # Use Test-Azure to determine unique name for s
62
63 #Ensure we are in Service Management mode to r
64 Switch-AzureMode AzureServiceManagement
65
66 $uniqueName = $false
67 $counter = 0
68 while ($uniqueName -eq $false)

```

Context Menu:

- Cut (Ctrl+X)
- Copy (Ctrl+C)
- Paste (Ctrl+V)
- Start Intellisense (Ctrl+Space)
- Start Snippets (Ctrl+J)
- Run Selection (F8)**
- Toggle Breakpoint (F9)

32. Scroll to the bottom of the script, and remove the # at the beginning of the second command to perform a resource group deployment.

```

101 # If you want to deploy a remote template using this script, add values for the $deploymentname, $templatefileURI, and $parameterfileURI
102 # and uncomment the following line.
103
104 New-AzureResourceGroupDeployment -Name $deploymentname -ResourceGroupName $rgname -TemplateParameterUri $parameterfileURI -TemplateUri $templatefileURI
105

```

33. Select the entire line, and then click **Run Selection**.

✦ After a few minutes, the command should return a successful result. You can verify the results by viewing the new resources in the Azure portal.


34. Please leave the Windows PowerShell ISE console open for the next exercise.

## Remove resource group used for lab

Because each lab in this series begins with an empty resource and because Azure resources are potentially billable, it is necessary to remove the resource group you created and used in this lab. Also, because Azure trial accounts are limited to 4 compute cores, it is important that you remove the resource group to ensure you do not run out of resources, if you have an Azure trial account.

### Remove Azure resource group

In this task you will run a Windows PowerShell script to remove the resource group you created and used in this lab.

 Perform the following tasks on AZRCAMP-ADMIN logged on as **Contoso\Administrator** using **Passw0rd!** as the password:

1. If not already open, open Windows PowerShell ISE.
2. Click **File**, click **Open**, browse to **C:\LabFiles\AZITPROCamp\Scripts**, select **RGCleanup.ps1**, and click **Open**.
3. On the menu, click **Run**.
4. When prompted, log into your Azure subscription.
5. When prompted to delete **RG-AZITCAMP-LAB01**, click **Yes**.
6. If you used a different resource group for the lab, you can modify the PowerShell script to delete that resource group.