Azure Resource Manager Inside-Out

In



Vision Scope

Prepared for

Microsoft

27-Jan-16

Version 1.0 Draft

Prepared by

**Kristian Nese**

CTO

kristian.nese@lumagate.com

Contributors

**Add Contributors to Doc Properties**

Table of Contents

[1 Before you begin 3](#_Toc435786932)

[2 Lab 1: Microsoft Azure Portal 3](#_Toc435786933)

[3 Lab 2: Template Authoring 7](#_Toc435786934)

[4 Lab 3: Deployment 15](#_Toc435786935)

[5 Lab 4: RBAC and Tags 20](#_Toc435786936)

[6 Appendix: Troubleshooting ARM deployments 25](#_Toc435786937)

[6 Troubleshooting 25](#_Toc435786938)

[6.1 ARM Explorer 25](#_Toc435786939)

[6.2 Windows Azure Agent VM Logs 27](#_Toc435786940)

[6.3 PowerShell DSC Logs in Event Viewer 28](#_Toc435786941)

[6.4 Deployment Timeout from Visual Studio & PowerShell 28](#_Toc435786942)

1. Before you begin

The objective of this Lab is to learn how to interact with Azure Resource Manager API through interfaces such as Visual Studio, Portal, GitHub and PowerShell.

In order to complete these labs, you need to ensure you have the following access and tools properly configured:

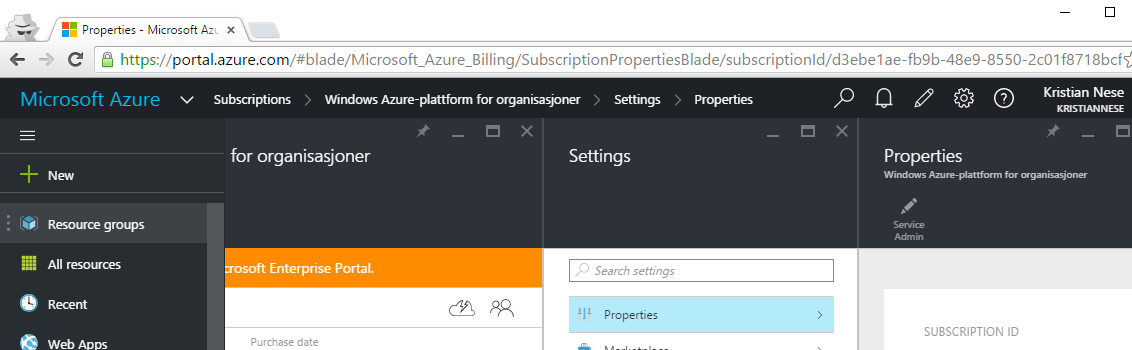
* Admin access to an Azure Subscription (minimum Trial Subscription)
* Visual Studio 2013 or 2015 Installed (minimum Community Edition)
* The latest SDK installed (at the time of writing this is version 2.8 and it can be downloaded from here: <https://azure.microsoft.com/en-us/blog/announcing-the-azure-sdk-2-8-for-net/>
* Microsoft Azure PowerShell (1.0.2 or higher)

1. Lab 1: Microsoft Azure Portal

In this lab, you will become familiar with the Azure Portal and its blade concept, as well as creating a new Resource Group and customize the portal to meet your needs.

**Part 1 – Getting familiar with the Azure Portal**

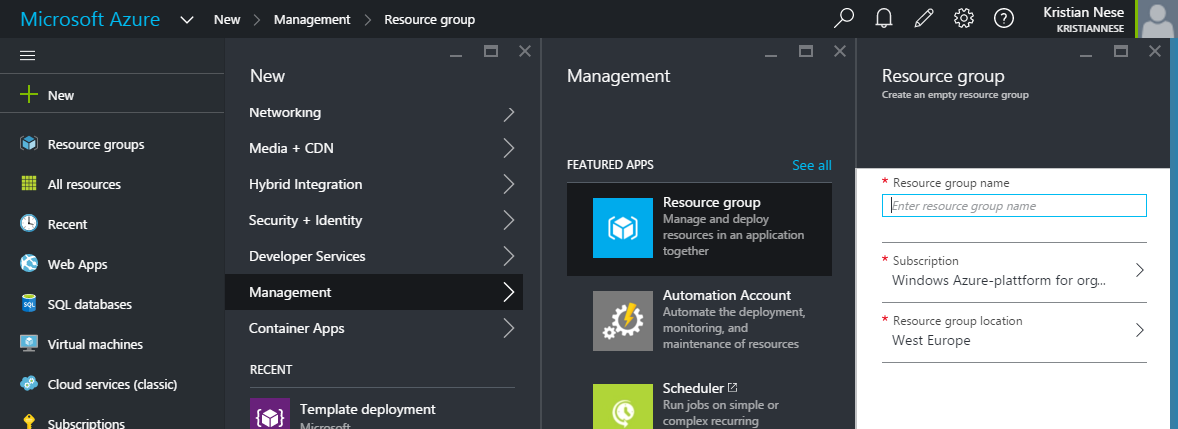
1. From your computer, open your preferred web browser and navigate to <https://portal.azure.com>
2. Sign in with your credentials that has access to an Azure subscription (you need Admin access)
3. Once you are logged in, explore the options you have available in the portal and familiarize yourself with the structure. Verify that by drilling further into each object by clicking on it and a new blad appears with further instructions



1. Close all the open blades and proceed to Part 2.

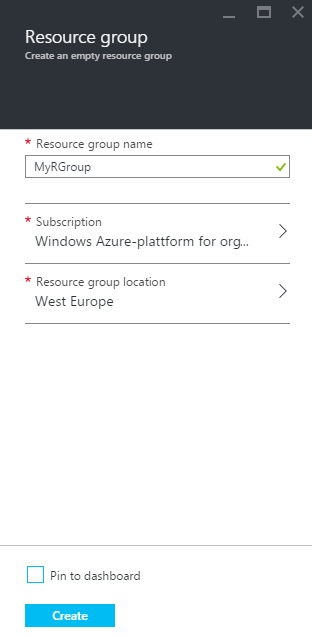
**Part 2 – Create your first Resource Group using the Azure Portal**

1. In the portal, click on ‘New’ 🡪 ‘Management’ 🡪 ‘Resource Group’

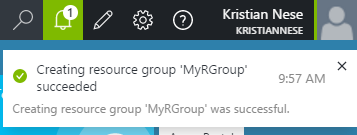


1. Assign a name to the Resource Group and select your preferred location. Also note that if your user has access to multiple subscriptions, you are also able to specify which subscription you want to deploy to on this step as well.

Ensure you un-tick the ‘Pin to dashboard’ checkbox, and then click ‘Create’.



1. Once you have clicked create, notice that you get a notification in the upper right that gives you information about the action and its status



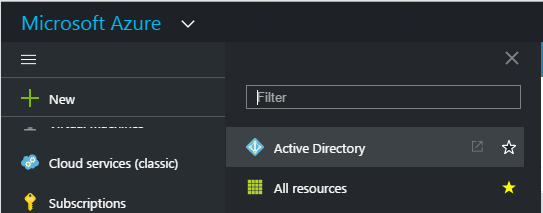
You have now successfully deployed a new Resource Group in Azure

**Part 3 – Customizing the Azure Portal**

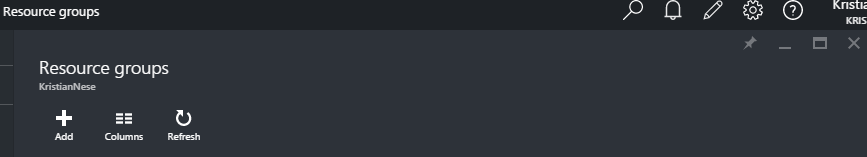
1. Down to the left in the portal, you should find a button called ‘Browse’ that let you browse all the categories you have available.



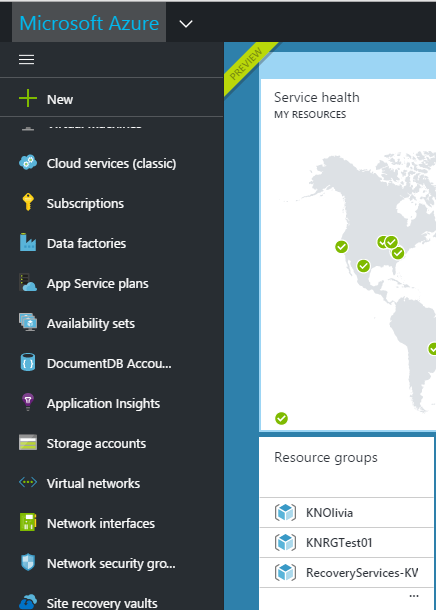
1. Click on ‘Browse’ and explore the list of available resources and categories. You can also notice the star sign on each object. If the star is yellow, it is indicated as a favorite and should be listed by default in the portal view



1. Navigate to ‘Resource Groups’ and ensure it is ‘starred’ and go further into the object by clicking on it
2. In the upper right, there’s a pin icon you can click on in order to pin this object to the welcome screen on the portal. Click on it and pin the resource group view



1. Go back to the portal and verify that the Resource Group view is present



**Part 4 – Retrieving the Resource Group using PowerShell**

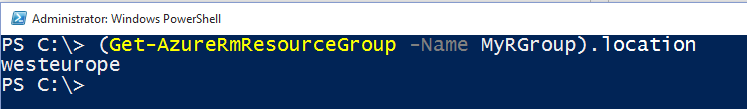
1. Start PowerShell on your computer and connect to your Azure subscription with your admin account, using the following cmdlet:

Add-AzureRmAccount -Credential (Get-Credential)

Enter the credential and ensure you have access to the correct subscription.

1. Ensure that the newly created Resource Group is deployed to the location you specified earlier by running the following cmdlet:

(Get-AzureRmResourceGroup –Name <nameOfYourResourceGroup>).location



**Summary**

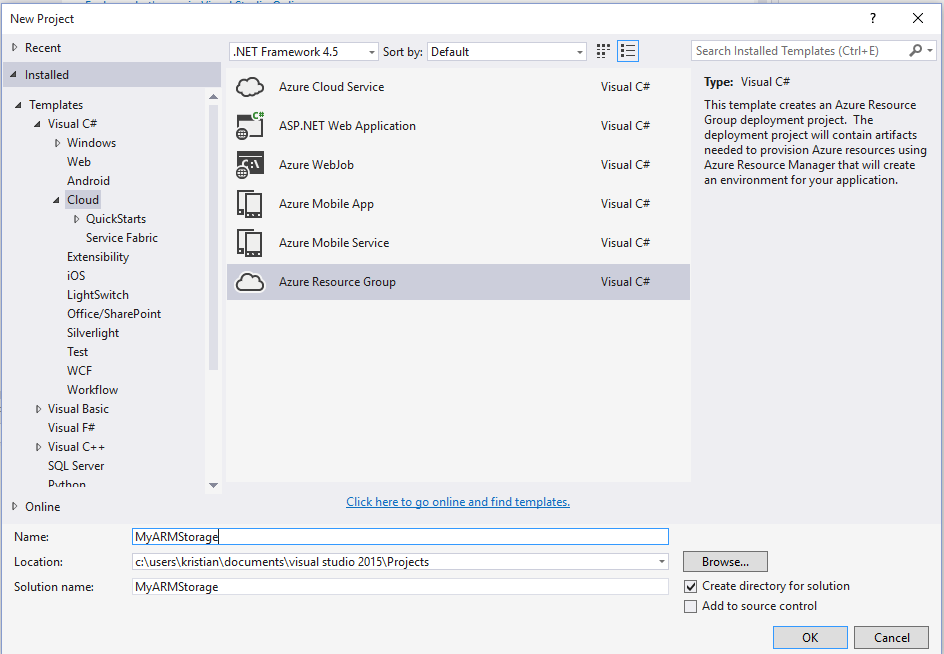
You have now successfully completed Lab 1, where you have donw the following:

* Explored the portal and become familiar with the structure and organizing of resources
* Created your first Resource Group into a preferred location and subscription
* Customized the portal and pinned the Resource Group view to the start screen
* Retreved information from your Resource Group location using PowerShell AzureRm module

1. Lab 2: Template Authoring

In this lab, you will create an Azure Resource Manager template, and also explore some existing templates available on GitHub.

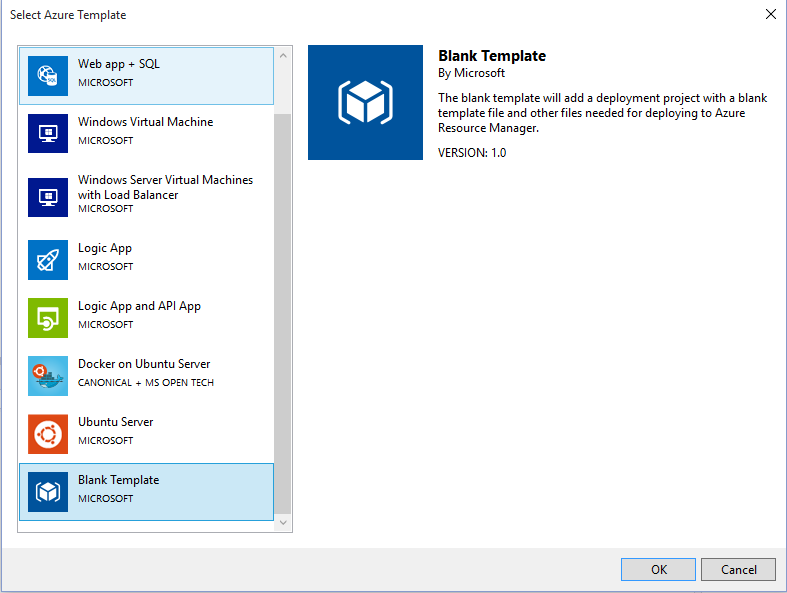
1. Launch Visual Studio on your computer, click File 🡪 New 🡪 Project
2. In the new dialogue box, expand ‘Templates’ 🡪 ‘Visual#’ 🡪 ‘Cloud’ and select ‘Azure Resource Group
3. Assign the following name to the project: ‘MyARMStorage’, leave the default options enabled and click ‘OK’



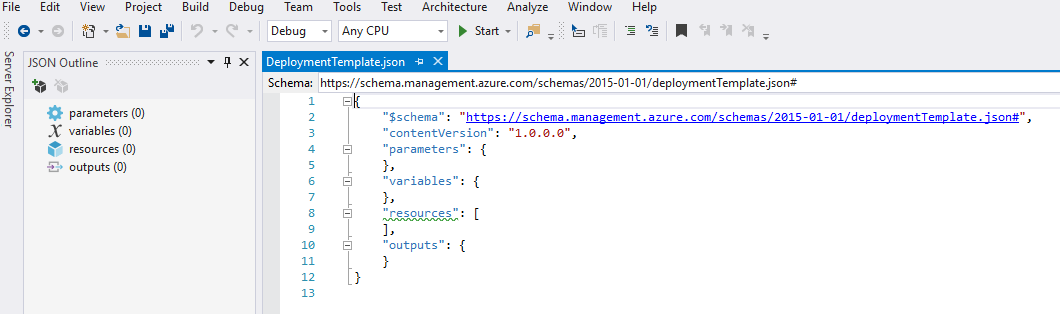
1. On the next screen, you are able to select some templates provided directly by Microsoft.

You can see that you can combine even PaaS and IaaS services in the same document, which shows that ARM is capable of deploying mutli-services.

Scroll down and select ‘Blank Template’ and click OK

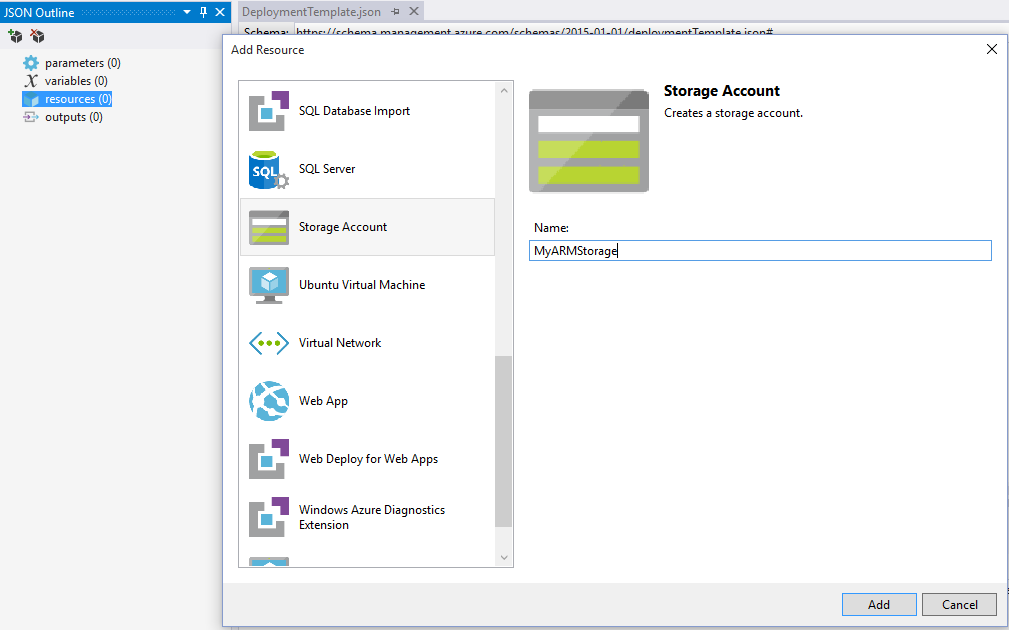


1. Ensure you have selected the template in the ‘Solution Explorer’ and you should then see the JSON template in Visual Studio with the JSON outline on the left



1. Right click on the ‘resources’ in the JSON outline and click ‘Add’. This will now open a new dialogue box that will let you add resources to your JSON template in a visual way.

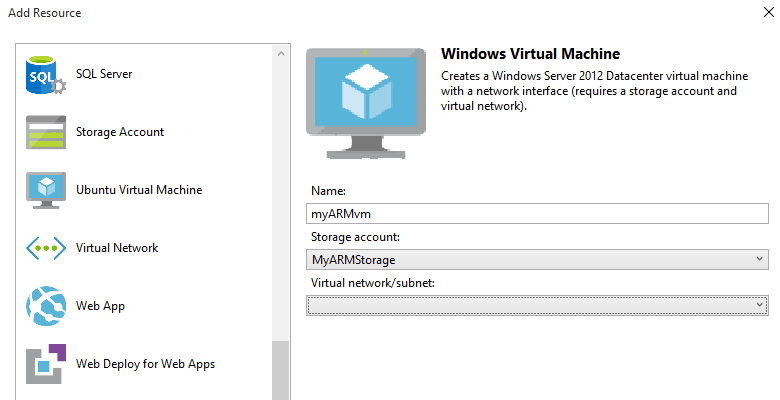
Scroll down and select ‘Storage Account’, assign a name and click ‘Add’



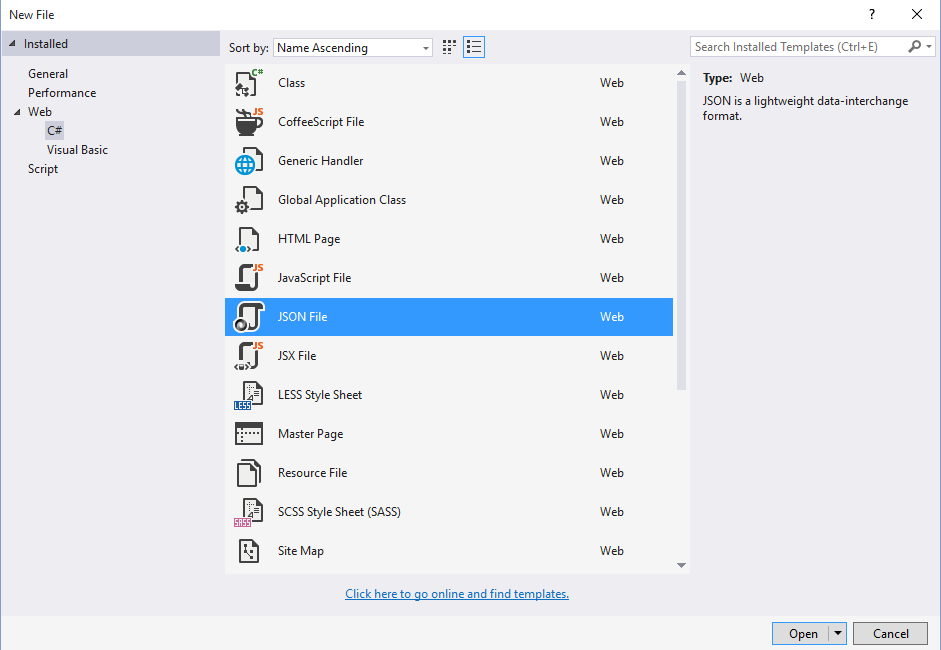
1. You can now see that the template contains a parameter, variable and a resource that will create a storage account resource type.

Spend some time and explore the template and its structure. What would be ideal to change?

1. Explore the JSON outline a bit more and try to add more resources. See how the template dynamically changes and how dependencies and references are created.



1. When you are done, save the project to your preffered location.
2. Once you have saved, go to File 🡪 New 🡪 File, expand Web 🡪 C# and select ‘JSON File’. Click ‘Open’.



This will now open a blank JSON document without any code.

1. Next, either copy – or manualy write the following code into the newly created JSON document:

{

"$schema": "http://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",

"contentVersion": "1.0.0.0",

"parameters": {

"mystoragename": {

"type": "string",

"defaultValue": "ARMStorage01"

},

"storageType": {

"type": "string",

"defaultValue": "Standard\_LRS",

"allowedValues": [

"Standard\_GRS",

"Standard\_LRS"

]

}

},

"resources": [

{

"apiVersion": "2015-05-01-preview",

"type": "Microsoft.Storage/storageAccounts",

"location": "[resourcegroup().location]",

"name": "[parameters('mystoragename')]",

"tags": {

"Department": "Sales"

},

"properties": {

"accountType": "[parameters('storagetype')]"

}

}

]

}

It is good learning experience to manually write the template yourself so that you get it “in your fingers” and becomes familiar with the syntax.

1. Next, let us add a variable to the template and reference this to simplify the language expression when adding multiple resources. Add the following section to your template in Visual Studio:
2. We have now added a variables section to the template that we can call upon whenever we add new resources and need to specify the API version we want to use.
3. Next, modify your template to be similar to the following example where we are using the variable on the resource object’s apiVersion property:

{

"$schema": "http://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",

"contentVersion": "1.0.0.0",

"parameters": {

"mystoragename": {

"type": "string",

"defaultValue": "ARMStorage01"

},

"storageType": {

"type": "string",

"defaultValue": "Standard\_LRS",

"allowedValues": [

"Standard\_GRS",

"Standard\_LRS"

]

}

},

"variables": {

"apiVersion": "2015-05-01-preview"

},

"resources": [

{

"apiVersion": "2015-05-01-preview",

"type": "Microsoft.Storage/storageAccounts",

"location": "[resourcegroup().location]",

"name": "[parameters('mystoragename')]",

"tags": {

"Department": "Sales"

},

"properties": {

"accountType": "[parameters('storagetype')]"

}

}

]

}

{

"$schema": "http://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",

"contentVersion": "1.0.0.0",

"parameters": {

"mystoragename": {

"type": "string",

"defaultValue": "ARMStorage01"

},

"storageType": {

"type": "string",

"defaultValue": "Standard\_LRS",

"allowedValues": [

"Standard\_GRS",

"Standard\_LRS"

]

}

},

"variables": {

"apiVersion": "2015-05-01-preview"

},

"resources": [

{

"apiVersion": "[variables('apiVersion')]",

"type": "Microsoft.Storage/storageAccounts",

"location": "[resourcegroup().location]",

"name": "[parameters('mystoragename')]",

"tags": {

"Department": "Sales"

},

"properties": {

"accountType": "[parameters('storagetype')]"

}

}

]

}

1. Further on, we will add a new resource to the template, that will instantiate a web site resource type, using a different apiVersion and supporting a new parameter, and also have a dependency on the storage account. Modify your template to be similar to the following:

{

"$schema": "http://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",

"contentVersion": "1.0.0.0",

"parameters": {

"mystoragename": {

"type": "string",

"defaultValue": "ARMStorage01"

},

"storageType": {

"type": "string",

"defaultValue": "Standard\_LRS",

"allowedValues": [

"Standard\_GRS",

"Standard\_LRS"

]

},

"webSiteName": {

"type": "string",

"defaultValue": "lab3-2015"

}

},

"variables": {

"StorageapiVersion": "2015-06-15",

"WebapiVersion": "2015-08-01"

},

"resources": [

{

"apiVersion": "[variables('StorageapiVersion')]",

"type": "Microsoft.Storage/storageAccounts",

"location": "[resourcegroup().location]",

"name": "[parameters('mystoragename')]",

"tags": {

"Department": "Sales"

},

"properties": {

"accountType": "[parameters('storagetype')]"

}

},

{

"apiVersion": "[variables('WebapiVersion')]",

"type": "Microsoft.Web/sites",

"location": "[resourceGroup().location]",

"name": "[parameters('webSiteName')]",

"dependsOn": [

"[concat('Microsoft.Storage/storageAccounts/', parameters('mystoragename'))]"

],

"properties": {

"name": "[parameters('webSiteName')]"

}

}

]

}

1. Once you are done, save the JSON template to a folder on your computer, for example c:\ARMTemplate\StorageAndWebDemo.JSON

You have now successfully created your first ARM template.

1. Lab 3: Deployment

In this lab you will explore how ARM is handles deployment and how to perform template deployments using different interfaces.

**Part 1 – Deploy using PowerShell**

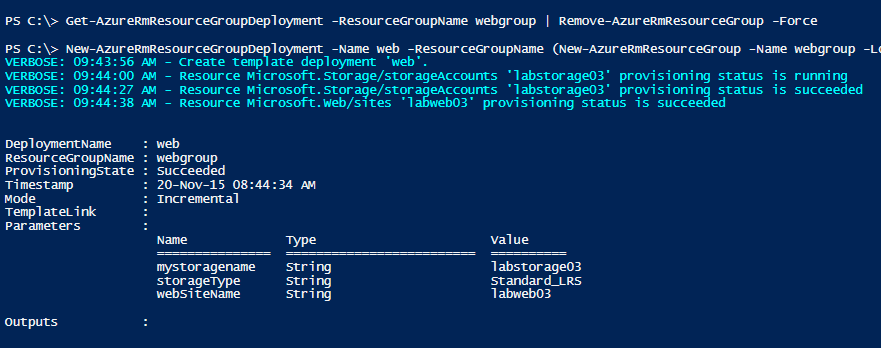
1. Open PowerShell and connect to your subscription if you haven’t already done so, by running the following cmdlet:

Add-AzureRmAccount -Credential (Get-Credential)

1. You will now deploy the template you just created, so ensure you know where you saved the JSON file because we will have to reference the template location in the cmdlet we’re using. Type the following cmdlet (on a single line):

New-AzureRmResourceGroupDeployment -Name web -ResourceGroupName (New-AzureRmResourceGroup -Name webgroup -Location "west europe").ResourceGroupName -TemplateFile C:\arm\StorageAndWebDemo.json -mystoragename labstorage03 -webSiteName labweb03 -Verbose

1. If you receive any errors during the attempt, try to determine the root cause and implement a solution
2. If the deployment goes as expected, you should have a similar output like this:



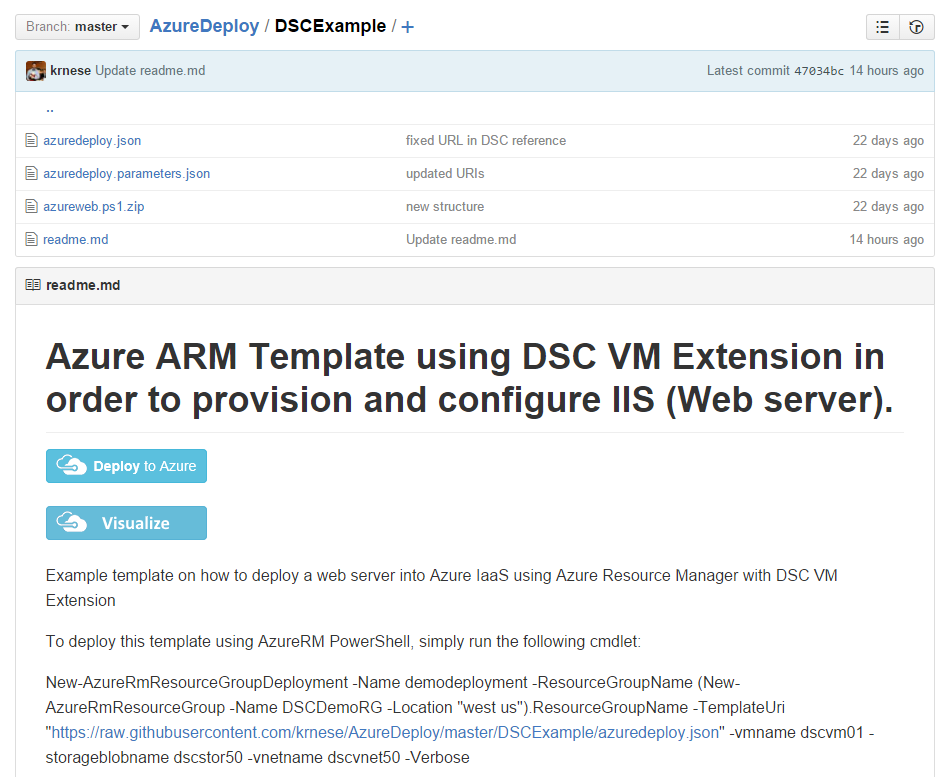
1. Try running the exact same cmdlet. What happens then?

**Part 2 – Deploy using GitHub**

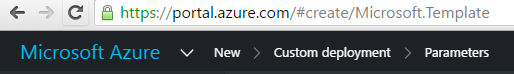
Since large templates are hard to fit into a guide like this, we would like to point you towards some public ARM templates on GitHub, for you to explore, deploy and eventually download and customize.

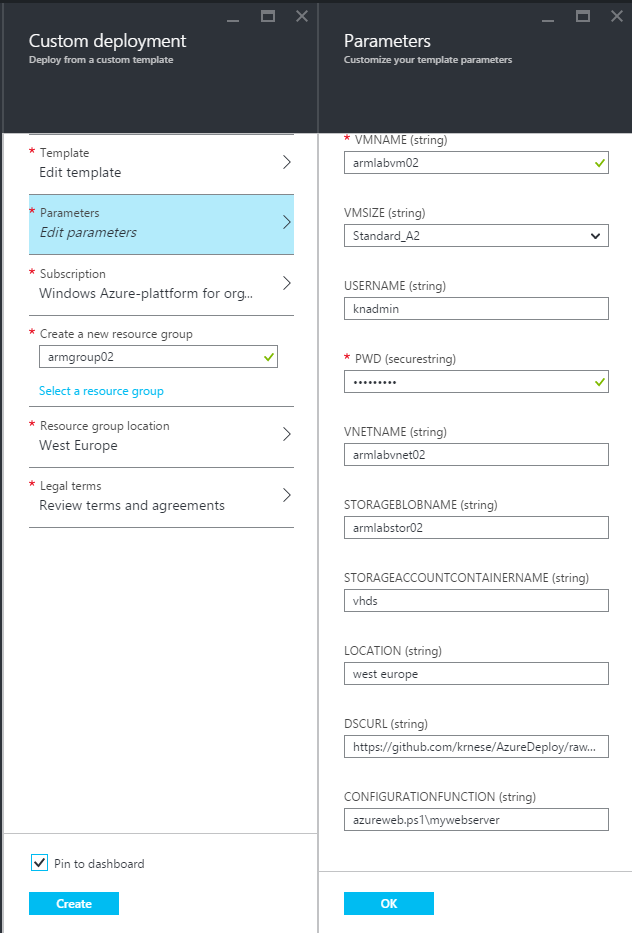
1. Access the following GitHub repository using your preffered web browser: <https://github.com/krnese/AzureDeploy/tree/master/DSCExample>
2. In this repository, you have access to an ARM template that will depoy a new virtual machine (Windows Server 2012 R2) with a storage account, virtual network, vNic, public IP address and a DSC extension that will ensure that IIS (Web-Server) will be installed and configured.

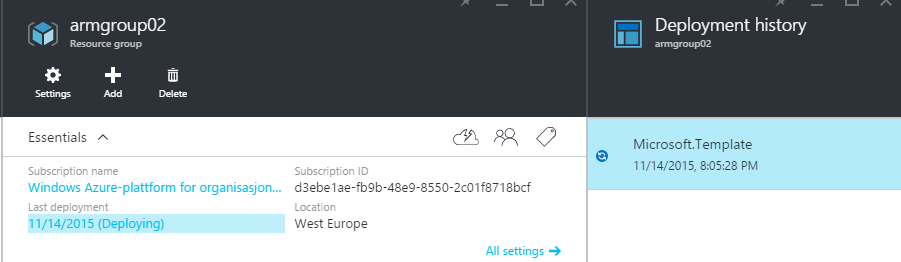
Click on the ‘Deploy’ button.



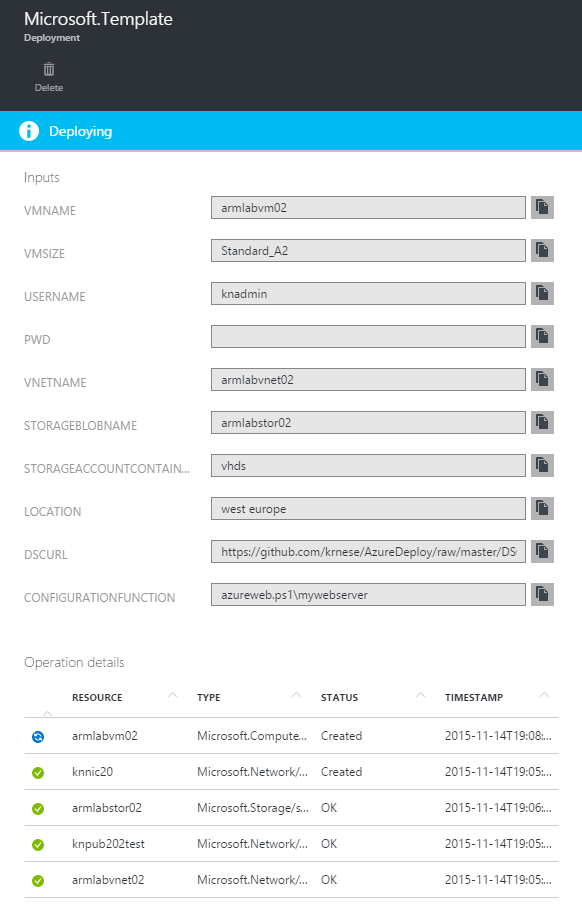
1. This will take you into the Azure portal and show the template and its parameters



1. Complete the parameters and deploy the template into a new Resource Group to your preferred subscription
2. Once the deployment has started, you can track the process directly from within the portal by drilling into the newly created Resource Group and click on the Deployment history and then the job



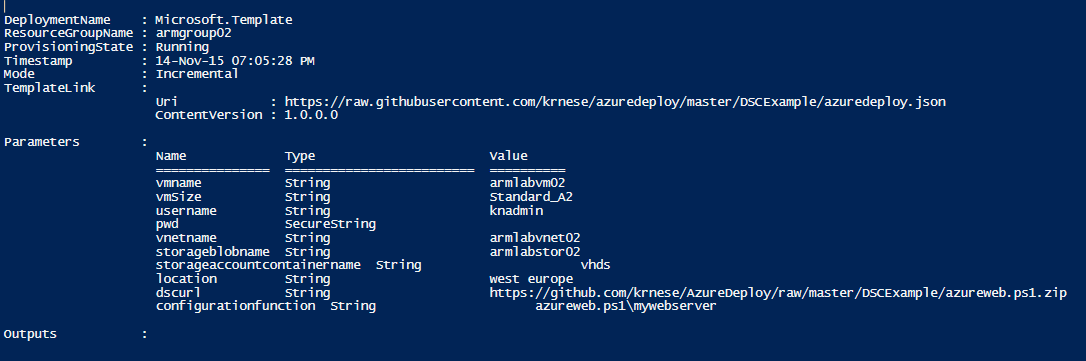
1. The deployment job shows the parameters (except the password – which is defined as a ‘securestring’). You can further drill into each step of the job to get even more details



1. In addition to the portal, you can use PowerShell to retrieve information about the current deployment by running the following cmdlet (on a single line):

Get-AzureRmResourceGroupDeployment -ResourceGroupName armgroup02 -Name microsoft.template

That should give a similar output as shown below



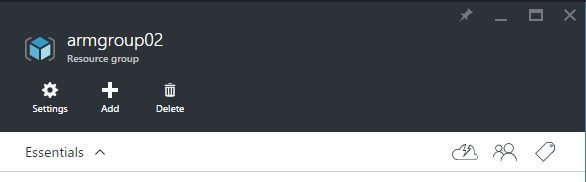
You have now successfully deployed ARM templates using PowerShell and the portal through a GitHub repository.

1. Lab 4: RBAC and Tags

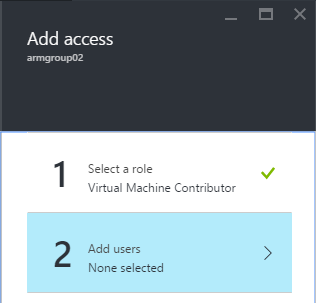
This lab will focus on Role Based Access Control and Tags.

**Part 1 – Configure RBAC on the newly created Resource Group from Lab 3**

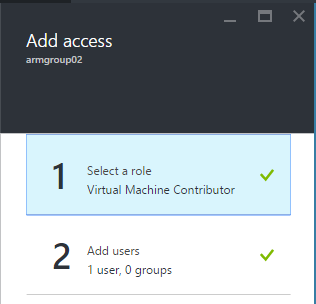
1. In the portal, navigate to the newly created Resource Group from Lab 3 and click on the RBAC icon and a new blade will open



1. Click ‘Add’🡪 ‘Select a role’ 🡪 ‘Virtual Machine Contributor’

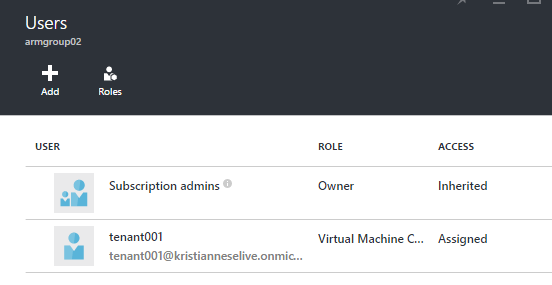


1. Next, click ‘Add users’ 🡪 and select one of your tenants you created earlier in the Technical Foundation for Active Directory. If you want to select multiple tenants, hold ‘Shift’ while selecting them from the list. Click ‘Ok’ once you are done.

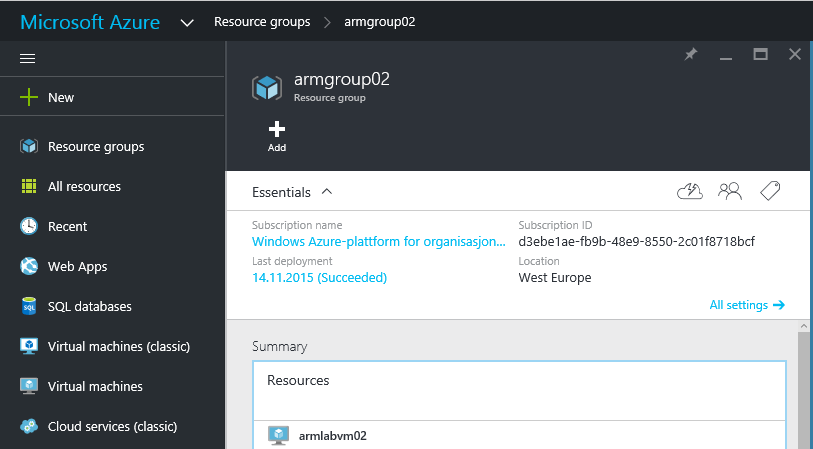


1. When you return to the ‘Add access’ blade, click OK to let ARM configure the new RBAC

Once completed, the newly added user(s) should have access

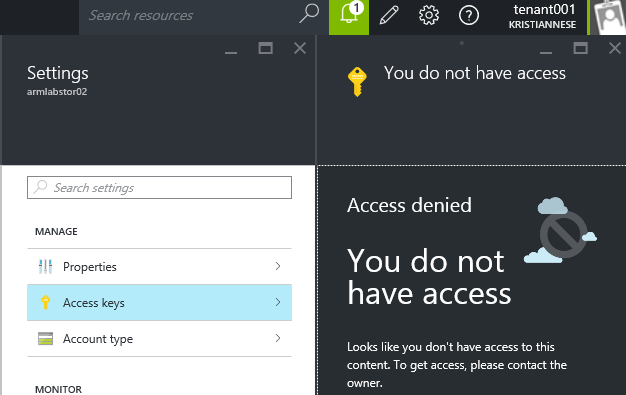


**Part 2 – Accessing the Resource Group as the tenant to verify RBAC**

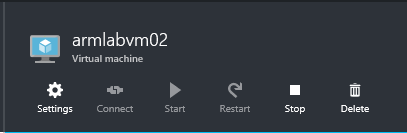
1. Start a new private web session and logon to <https://portal.azure.com> as the tenant and ensure you have the proper access that you just assigned 

In our case, we assigned the virtual machine contributor role. If you did the same, ensure you have access to the ‘Audit logs’ for the virtual machine

1. When you have the ‘Virtual Machine contributor role’ assigned, you should not have access to any of the other components within the resource group. Ensure that you don’t have access to the access keys of the storage by drilling into the properties

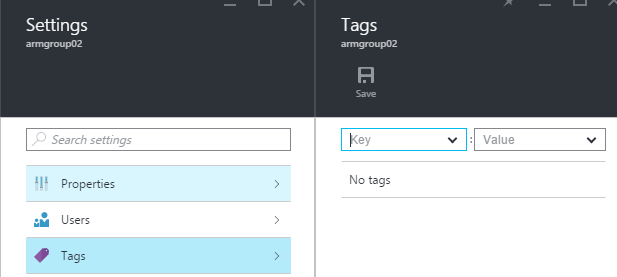


1. Verify that you can stop and start the virtual machine in the resource group through the portal



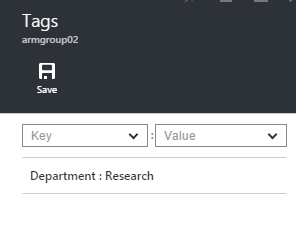
You have now verified that RBAC is successfully implemented on your newly created Resource Group

**Part 3 – Setting Tags on resources**

1. Login to <https://portal.azure.com> as the service admin and open the newly created Resource Group
2. Click on ‘All settings’ 🡪 Tags 
3. Assign a new Key and a Value. In our example we are going to use “Department” and “Research”. Once you are done you can click on ‘Save’



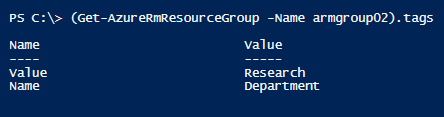
1. You have now successfully added Tags to the newly created Resource Group



**Part 4 – Retrieving Tags information using PowerShell**

1. Launch PowerShell as the Service Admin and run the following cmdlet:

(Get-AzureRmResourceGroup -Name armgroup02).tags

1. Verify that the output is what you recently created using the portal 

You have now successfully completed the lab and assigned RBAC and Tags to your Resource Group.

1. Appendix: Troubleshooting ARM deployments
2. Troubleshooting

Troubleshooting deployment issues in ARM Template Deployment isn’t always very straightforward. Below are some guidelines to help you troubleshoot failed deployments.

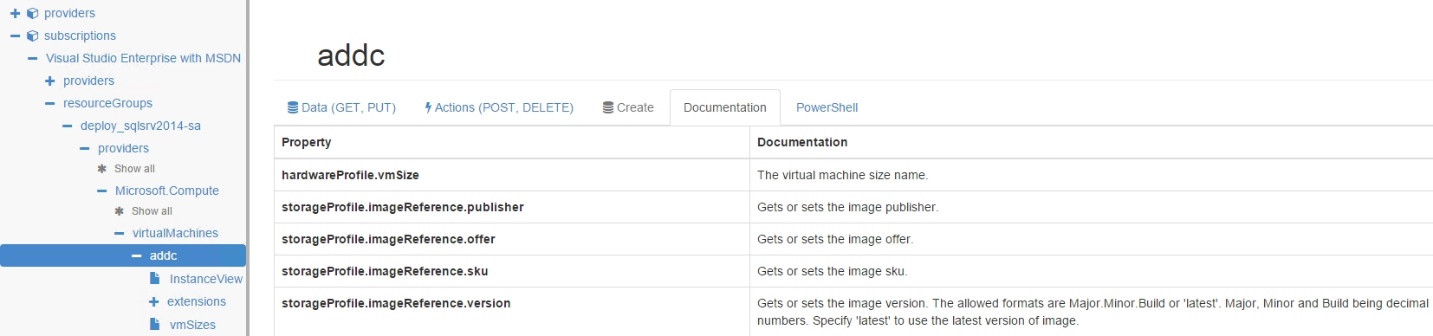
* 1. ARM Explorer

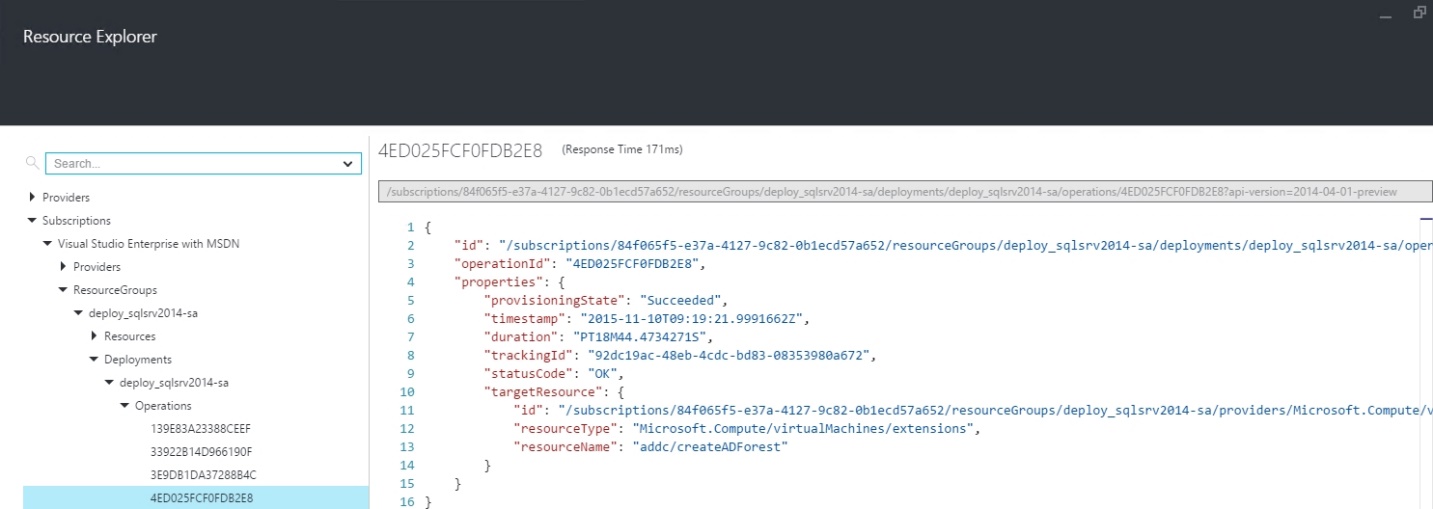
Azure Resource Explorer (also known as ARM Explorer) allows you to view ARM Deployments in Azure as they are being deployed in real-time. ARM Explorer allows you to view the API in your own subscription directly from a web browser at the URL below:

<https://resources.azure.com>

Everything displayed in ARM Explorer is in JSON, but is color coded for readability. Below is a screenshot of the output of the ADDC PowerShell DSC Resource Deployment of Active Directory resource and the respective message output:

  
  
Using ARM Explorer, you have the option of creating resources by submitting JSON directly of the properties of the resources you are creating. You also have the option of deleting resources from here as well. PowerShell Code samples for adding and removing resources are available on the PowerShell tab. Finally, if you click on a particular resource and click on the Documentation tab, you will retrieve all Property values available from the resource along with the respective documentation.

  
  
The Azure Portal also has a new option called Resource Explorer, giving you a quick glance into the API to determine if a resource was deployed successfully or not.



Unlike ARM Explorer, Resource Explorer only provides a quick view of resources and their status and doesn’t an option to manage resources directly like in ARM Explorer. This could be an option for tenants looking to have more visibility into deployments without giving too much access.

* 1. Windows Azure Agent VM Logs

If you are still having problems finding the root cause of your deployment issues, you can log into the deployed VMs and find additional information as to why an extension failed to run correctly by review the logs of the respective named extensions in the following paths:

C:\WindowsAzure\Logs\Plugins  
  
This section will have basic information as to when the extension was installed and executed successfully. When using the PowerShell Script Extension, it is possible for the PowerShell Script to fail but for the execution of the extension to return successful. That is why you should review the logs generated by the extensions in the following path below:

C:\Packages\Plugins

Under the named folder of the extension you are investigating, you should find a folder called **Status**; within this folder should be a file called **0.status**. The actions taken by the extension during runtime will available here for your review.

On Linux Hosts, you can find the same **0.status** file by looking in the following directory:

/var/lib/waagent/

For example, if you are using the Custom Script Extension for Linux, the full path to the **0.status** file might look like:

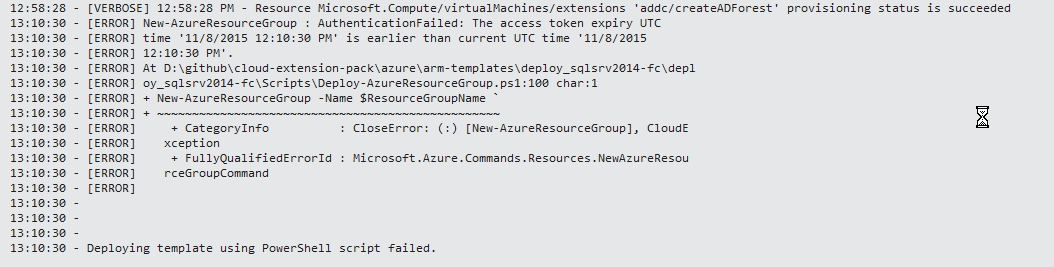
/var/lib/waagent/Microsoft.OSTCExtensions.CustomScriptForLinux-1.2.2.0/status/0.status

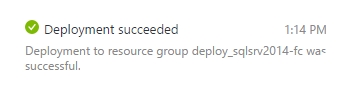
* 1. PowerShell DSC Logs in Event Viewer

Additional troubleshooting information can be gleaned from the PowerShell DSC Operational Log in Event Viewer. After opening up Event Viewer, the path to the log is **Application and Service Logs 🡪 Microsoft 🡪 Windows 🡪 Desired State Configuration 🡪 Operational**.

* 1. Deployment Timeout from Visual Studio & PowerShell

When deploying an ARM Template that may run over an hour from Visual Studio or PowerShell, the deployment may return an error message: *The access token expiry UTC time is earlier than current UTC time*, as shown in the screenshot below:



If this happens, open up ARM Explorer and verify that the deployment is still running; more often than not, the deployment will complete successfully several minutes later:  


If the customer does not have a properly configured VPN or Azure Subscription in place, please have them consider the Azure IaaS Foundations IP at http://aka.ms/MCS\_EPG\_Azure\_Iaas-Foundation.

These are just examples; update the requirements adapting to the specific customer situation if necessary. Points to consider might include: tool usage (MAP versus customer provided), server locations, availability of test lab, and so on.