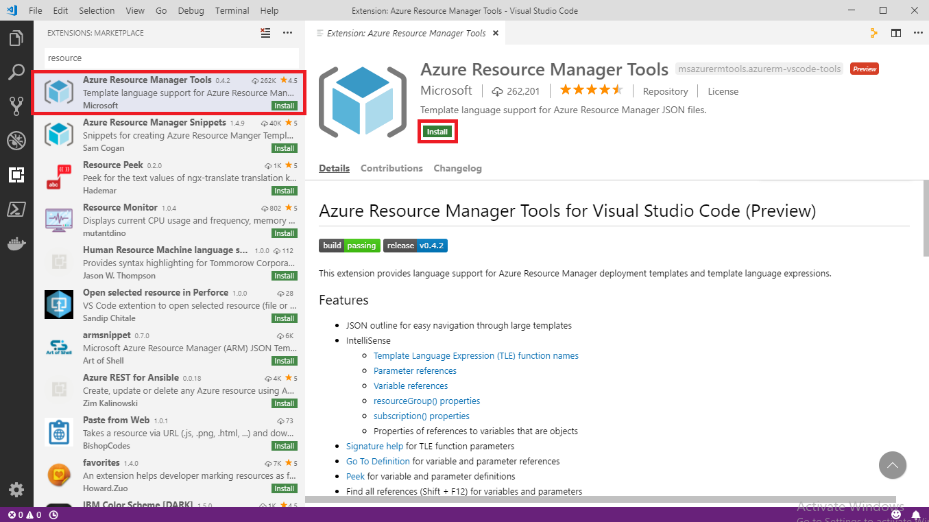
ARM templates, part 2

Get tools

1. In the task we will use **Visual Studio Code** as our editor, which can be installed from here <https://code.visualstudio.com/>.
2. In **Visual Studio Code**, go to **File** > **Preferences** > **Extensions** and in the search box type **Azure Resource Manager** tools install the [Azure Resource Manager Tools](https://marketplace.visualstudio.com/items?itemName=msazurermtools.azurerm-vscode-tools)
3. [Install Azure CLI on Windows](https://docs.microsoft.com/en-us/cli/azure/install-azure-cli-windows?tabs=azure-cli), make sure you have version 2.6 or later, to check your installed version, use: az --version.

Create your template

1. Open Visual Studio Code with the Resource Manager Tools extension installed.
2. From the **File** menu, select **New File** to create a new file.
3. From the **File** menu, select **Save as**.
4. Name the file **azuredeploy** and select the **JSON** file extension. The complete name of the file **azuredeploy.json**.
5. Save the file to your workstation. Select a path that is easy to remember because you'll provide that path later when deploying the template.
6. Copy and paste the following JSON into the file:

{

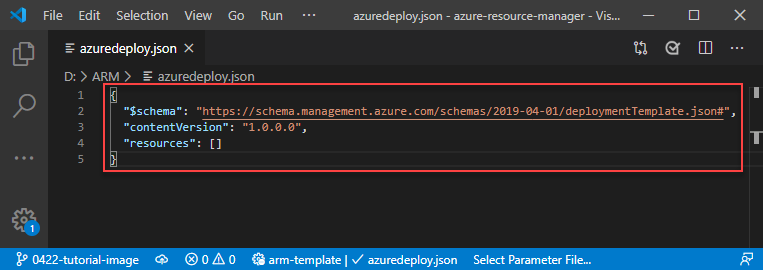
"$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",

"contentVersion": "1.0.0.0",

"resources": []

}

Here's what your VS Code environment looks like:



This template doesn't deploy any resources. We're starting with a blank template so you can get familiar with the steps to deploy a template while minimizing the chance of something going wrong.

1. Save the file.

Sign in to Azure

To start working with Azure PowerShell/Azure CLI, sign in with your Azure credentials.

az login

If you have multiple Azure subscriptions, select the subscription you want to use:

az account set --subscription [SubscriptionID/SubscriptionName]

Copy template

To create multiple resources of the same type you can use copy function:

{

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

"contentVersion": "1.0.0.0",

"parameters": {

},

"resources": [

{

"apiVersion": "2016-01-01",

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

"name": "[concat(copyIndex(),'storage', uniqueString(resourceGroup().id))]",

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

"sku": {

"name": "Standard\_LRS"

},

"kind": "Storage",

"properties": {},

"copy": {

"name": "storagecopy",

"count": 4,

"mode": "Serial",

"batchSize": 2

}

}

],

"outputs": {}

}

By default, Resource Manager creates the resources in parallel. It applies no limit to the number of resources deployed in parallel, other than the total limit of 800 resources in the template. The order in which they're created isn't guaranteed.

However, you may want to specify that the resources are deployed in sequence. For example, when updating a production environment, you may want to stagger the updates so only a certain number are updated at any one time. To serially deploy more than one instance of a resource, set mode to serial and batchSize to the number of instances to deploy at a time. With serial mode, Resource Manager creates a dependency on earlier instances in the loop, so it doesn't start one batch until the previous batch completes.

The value for batchSize can't exceed the value for count in the copy element.

Deploy template

To deploy the template, use either Azure CLI or Portal. Use the resource group that was created for you. Give a name to the deployment so you can easily identify it in the deployment history. For convenience, also create a variable that stores the path to the template file. This variable makes it easier for you to run the deployment commands because you don't have to retype the path every time you deploy.

$templateFile="{provide-the-path-to-the-template-file}"

az deployment group create `

--name copyfunction `

--resource-group myResourceGroup `

--template-file $templateFile

The deployment command returns results. Look for ProvisioningState to see whether the deployment succeeded.

Copy based on input parameters

The copy operation is helpful when working with arrays because you can iterate through each element in the array. Use the length function on the array to specify the count for iterations, and copyIndex to retrieve the current index in the array.

The following example creates one storage account for each name provided in the parameter.

{

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

"contentVersion": "1.0.0.0",

"parameters": {

"listOfSANames": {

"type": "array",

"defaultValue": [

"ic01",

"ic02czechrepublic",

"ic03baltics"

]

}

},

"resources": [

{

"apiVersion": "2017-06-01",

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

"name": "[concat(parameters('listOfSANames')[copyIndex()], uniqueString(resourceGroup().id))]",

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

"sku": {

"name": "Standard\_LRS"

},

"kind": "Storage",

"properties": {},

"copy": {

"name": "storagecopy",

"count": "[length(parameters('listOfSANames'))]"

}

}

],

"outputs": {}

}

Deploy template

You can deploy the template to create the storage account. Give your deployment a different name so you can easily find it in the history.

az deployment group create `

--name copyarray `

--resource-group myResourceGroup `

--template-file $templateFile

Verify deployment

You can verify the deployment by exploring the resource group from the Azure portal.

1. Sign in to the [Azure portal](https://portal.azure.com/).
2. From the left menu, select **Resource groups**.
3. Select the resource group you deployed to.
4. You see that a storage account has been deployed.
5. Notice that the deployment label now says: **Deployments: 2 Succeeded**.

User Functions

Within your template, you can create your own functions. These functions are available for use in your template. User-defined functions are separate from the [standard template functions](https://docs.microsoft.com/en-us/azure/azure-resource-manager/templates/template-functions) that are automatically available within your template. Create your own functions when you have complicated expressions that are used repeatedly in your template.

Copy the whole file and replace your template with its contents.

{

"$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",

"contentVersion": "1.0.0.0",

"parameters": {

"storageNamePrefix": {

"type": "string",

"maxLength": 11

}

},

"functions": [

{

"namespace": "alior",

"members": {

"uniqueName": {

"parameters": [

{

"name": "namePrefix",

"type": "string"

}

],

"output": {

"type": "string",

"value": "[concat(toLower(parameters('namePrefix')), uniqueString(resourceGroup().id))]"

}

}

}

}

],

"resources": [

{

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

"apiVersion": "2019-04-01",

"name": "[alior.uniqueName(parameters('storageNamePrefix'))]",

"location": "South Central US",

"sku": {

"name": "Standard\_LRS"

},

"kind": "StorageV2",

"properties": {

"supportsHttpsTrafficOnly": true

}

}

]

}

Deploy template

You can deploy the template to create the storage account. Give your deployment a different name so you can easily find it in the history.

az deployment group create `

--name userfunctions `

--resource-group myResourceGroup `

--template-file $templateFile `

--parameters storageNamePrefix={your-unique-name}

Clean up resources

1. From the Azure portal, select **Resource group** from the left menu.
2. Enter the resource group name in the **Filter by name** field.
3. Select the resource group name.
4. Delete all resources in selected resource group.

Conditional deployment

Sometimes you need to optionally deploy a resource in an Azure Resource Manager (ARM) template. Use the condition element to specify whether the resource is deployed. The value for this element resolves to true or false. When the value is true, the resource is created. When the value is false, the resource isn't created. The value can only be applied to the whole resource.

{

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

"contentVersion": "1.0.0.0",

"parameters": {

"ifDeploy": {

"type": "string",

"defaultValue":"No"

}

},

"resources": [

{

"condition": "[equals(parameters('ifDeploy'),'Yes')]",

"apiVersion": "2016-01-01",

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

"name": "[concat(copyIndex(),'storage', uniqueString(resourceGroup().id))]",

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

"sku": {

"name": "Standard\_LRS"

},

"kind": "Storage",

"properties": {},

"copy": {

"name": "storagecopy",

"count": 4,

"mode": "Serial",

"batchSize": 2

}

}

],

"outputs": {}

}

Deploy template

You can deploy the template to create the storage account. Give your deployment a different name so you can easily find it in the history.

az deployment group create `

--name conditionalDeployment `

--resource-group myResourceGroup `

--template-file $templateFile

Verify deployment – was anything deployed?

Change ifDeploy parameter value to Yes and redeploy

Verify deployment

You can verify the deployment by exploring the resource group from the Azure portal.

1. Sign in to the [Azure portal](https://portal.azure.com/).
2. From the left menu, select **Resource groups**.
3. Select the resource group you deployed to.
4. You see that a storage account resources have been deployed.

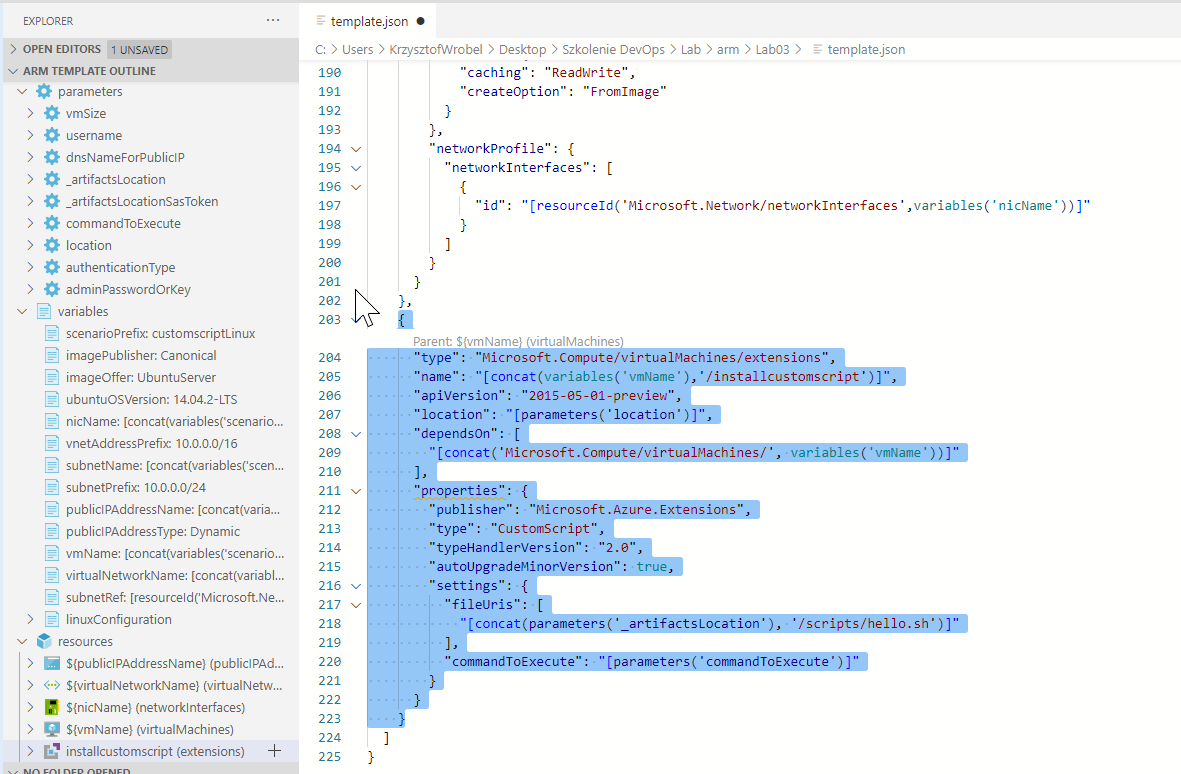
Custom Script extension

Browse <https://github.com/Azure/azure-quickstart-templates> and find an example of a VM template with Extension.

You can start with:

<https://github.com/Azure/azure-quickstart-templates/tree/master/201-customscript-extension-public-storage-on-ubuntu>

Download and review the template, find custom script extension:



Download parameters file and deploy the template to your resource group.

Clean up resources

1. From the Azure portal, select **Resource group** from the left menu.
2. Enter the resource group name in the **Filter by name** field.
3. Select the resource group name.
4. Delete all resources in selected resource group.