***ARM Template***

Reference urls:

<https://docs.microsoft.com/en-us/learn/paths/deploy-manage-resource-manager-templates/>

quick start templates:

<https://github.com/Azure/azure-quickstart-templates/tree/master/quickstarts>

What is ARM template?

Azure Resource manager (ARM) template is a way to describe your infrastructure....

{

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

"contentVersion": "1.0.0.0",

"parameters": {},

"variables": {},

"resources": [],

"outputs": {},

"functions": []

}

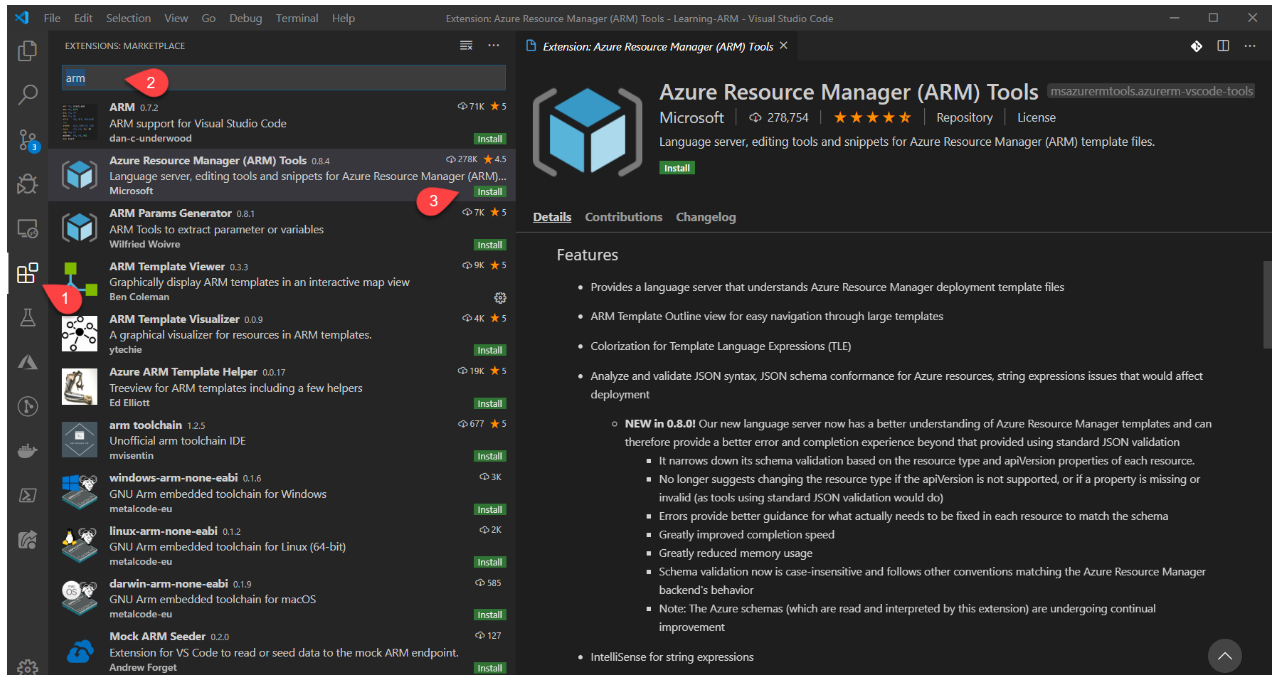
Advantages of ARM templates:

Here are some advantages to use ARM template:

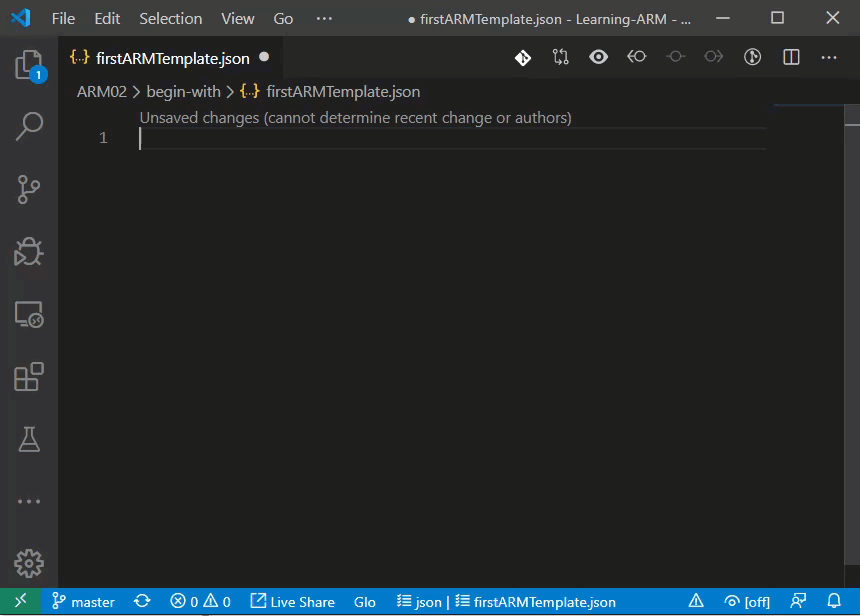
* A template file is light and easy to keep in a repository.
* It's very simple to have the exact same template deployed in multiple environments.
* The template is idempotent.
* ARM templates are really fast to deploy.
* Easy to edit/ customize/ expand.
* Easy to delete.

Create ARM Template:

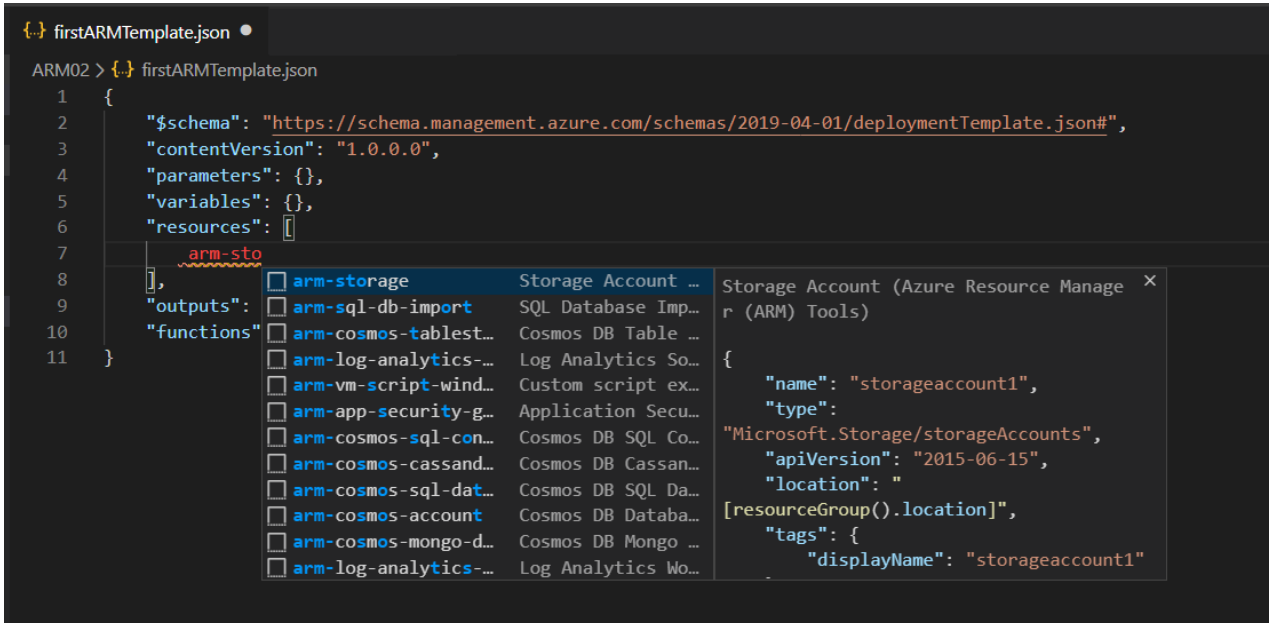
The Azure Resource Manager (ARM) Tools for Visual Studio Code is available from the marketplace (for free), or directly from VSCode. Click the **Extensions** button from the Side Bar and search for "**ARM Tools**".



Now from any JSon (.json) file the extension will automatically be active. Create a new File and save it as a JSON file (ex: firstARMTemplate.json). Start typing arm, you should see the context menu popping.

[](https://github.com/buchepalli-ramana/Learning-ARM/blob/master/ARM02/medias/skeleton.gif)

* That will add an empty template.
* Now type arm-storage, you should see a list of Azure resources filtered down to show only the storage. Click it and you should have now your first template done.



Now the tools is asking you to enter a name for your resource. Enter: learningARMStorage.

{

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

"contentVersion": "1.0.0.0",

"parameters": {},

"functions": [],

"variables": {},

"resources": [

{

"name": "storageaccount1",

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

"apiVersion": "2019-06-01",

"tags": {

"displayName": "storageaccount1"

},

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

"kind": "StorageV2",

"sku": {

"name": "Premium\_LRS",

"tier": "Premium"

}

}

],

"outputs": {}

}

Parameter Types:

To make our template dynamic we can pass some information when we call it using parameters. Remember the section parameters in the top of the templates?

{

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

"contentVersion": "1.0.0.0",

"parameters": {},

}

This is where we will define our parameters. There is multiple type of parameters but before we go list them and see some scenarios, let's understand how parameters are defined.

Parameters Definition in Azure Resource Manager templates:

It's easy to thinks about parameters like a key pair value. Something like a parameters as a name and a value. But there is much more then that.

"parameters": {

"storageSKU": {

"type": "string",

"allowedValues": [

"Standard\_LRS",

"Standard\_ZRS",

"Standard\_GRS",

"Standard\_RAGRS",

"Premium\_LRS"

],

"defaultValue": "Premium\_LRS",

"metadata": {

"description": "The type of replication to use for the storage account."

}

}

}

Parameter Types:

There is multiple type of parameter to accommodate the different type of information. We already mention the ***string*** type and other more also type like ***int, bool***, and ***array***.

Then there is ***securestring*** a very convenient way to pass a password for example. Using the securestring type will make sure the the information cannot be read after resource deployment.

Finally we have ***object*** and ***secureObject***. With those you can define a complete object with many properties it's extremely powerful. And things become very flexible when you start mixing all those type together, like passing an array of objects.

Command to deploy with parameters:

az group deployment create -g MyResourceGroup --template-file azuredeploy.json --parameters storageName=tstStorage storageKind=StorageV2

before deploy we can validate the ARM template by using the –what-if flag in the command.

az group deployment create ***–what-if*** -g MyResourceGroup --template-file azuredeploy.json --parameters storageName=tstStorage storageKind=StorageV2

Create a Parameter file:

A parameter files is just yet another JSON file. So you can easily create that in any text editor. However, there is a few really nice feature in the VS Code Extension: Azure Resource Manager (ARM) Tools that will make you save a lot of time.

To create a new parameter file simply right-click anywhere in your ARM template, to make the contextual menu to popup. Select **Select/Create Parameter File...**, then **New** and Finally **All parameters**.

This will create a new file with the following structure.

{

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

"contentVersion": "1.0.0.0",

"parameters": {

"storageSKU": {

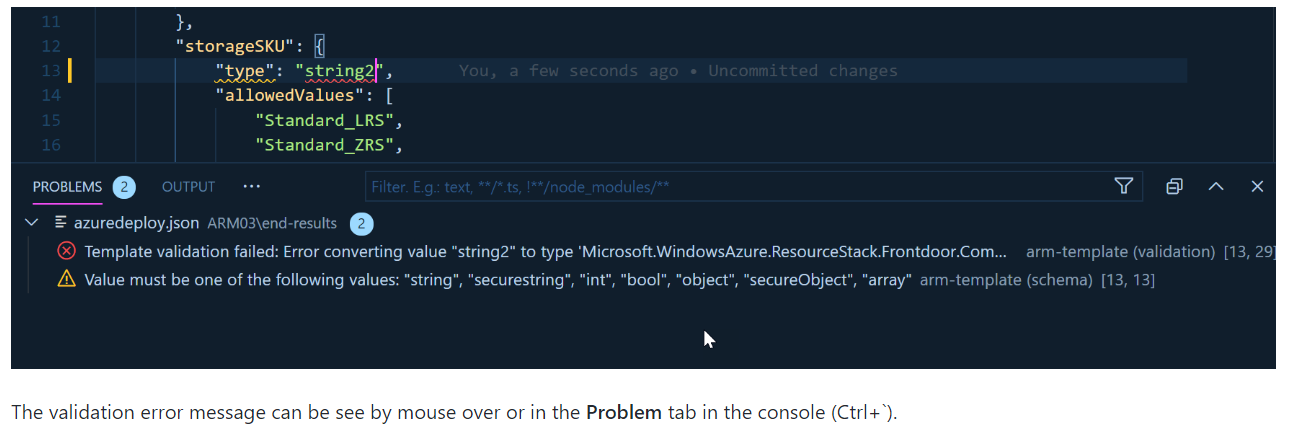
"value": "Premium\_LRS"

}

}

}

A interesting feature of using the Azure Resource Manager (ARM) Tools is that it will validate the template and the values of the parameters associated with it.



More Security:

The securestring inside the ARM template is really great. Using them will make sure our sensitive information won;t show up in the logs. But if we are using a parameters file then the value will also be present in clear text! To avoid doing this (and by mistake pushing our password in the source control, we need to use Azure KeyVault. You will need to to have a active KeyVault and you will be able to reference your secret in the parameter file like this:

In this sample we would pass the secret **secretName** contained in the KeyVault . TO know how to create your KeyVault refer to Use Azure Key Vault to pass secure parameter value during deployment

{

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

"contentVersion": "1.0.0.0",

"parameters": {

"adminPassword": {

"reference": {

"keyVault": {

"id": "/subscriptions/<subscription-id>/resourceGroups/<rg-name>/providers/Microsoft.KeyVault/vaults/<vault-name>"

},

"secretName": "ExamplePassword"

}

},

}

}

After confirming that your template and parameter file are valid, deploy it.

az group deployment create -g MyResourceGroup --template-file azuredeploy.json --parameters azuredeploy.parameters.json

How to define Variables in ARM template:

In the previous chapter, we created a suffix to make sure the resource name was unique. However, we needed to copy our "formula" all over the ARM template. Let's create a variable suffix with our formula and another one with the name of our resource.

The variables are defined in the variables section of the ARM template.

"variables": {

"suffix": "[uniqueString(resourceGroup().id)]",

"primeStorageName":"[toLower(concat(parameters('storageName'), variables('suffix')))]"

}

And now to use those variables I will use the very handy function variables() that returns the value of the variable name passed in parameter. See now how clearer is our template!

"resources": [

{

"name": "[variables('primeStorageName')]",

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

}

Parameters:

"parameters": {

"storageName": {

"type": "string",

"metadata": {

"description": "Name of your storage account. Text only no space."

},

"maxLength": 10

},

"environment": {

"type": "string",

"allowedValues": [

"DEV",

"PROD"

],

"defaultValue": "DEV",

"metadata": {

"description": "Define the type of environment and resource to deploy. "

}

}

}

Variables:

"variables": {

"suffix": "[uniqueString(resourceGroup().id)]",

"primeStorageName":"[toLower(concat(parameters('storageName'), variables('suffix')))]",

"storageInfo": {

"DEV": {

"storageSKU": "Standard\_LRS",

"storageTier": "Standard"

},

"PROD": {

"storageSKU": "Premium\_LRS",

"storageTier": "Premium"

}

}

}

The last step is to define a variable for SKU and Tier and assigned them a value base on the parameter environment.

"variables": {

"suffix": "[uniqueString(resourceGroup().id)]",

"primeStorageName":"[toLower(concat(parameters('storageName'), variables('suffix')))]",

"storageInfo": {

"DEV": {

"storageSKU": "Standard\_LRS",

"storageTier": "Standard"

},

"PROD": {

"storageSKU": "Premium\_LRS",

"storageTier": "Premium"

}

},

"storageSKU": "[variables('storageInfo')[parameters('environment')].storageSKU]",

"storageTier": "[variables('storageInfo')[parameters('environment')].storageTier]"

}

This is very powerful and useful. Now we just need to use those variables inside the template.

"resources": [

{

"name": "[variables('primeStorageName')]",

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

"apiVersion": "2019-06-01",

"tags": {

"displayName": "[variables('primeStorageName')]"

},

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

"kind": "StorageV2",

"sku": {

"name": "[variables('storageSKU')]",

"tier": "[variables('storageTier')]"

},

"properties": {

"supportsHttpsTrafficOnly": true

}

}

]

Best practices:

Here a few of the best practices that can be helpful when you work with variables (complete list [here](https://docs.microsoft.com/en-us/azure/azure-resource-manager/templates/template-best-practices#variables?WT.mc_id=learnARM-github-frbouche)):

* Use camel case for variable names.
* Use variables for values that you need to use more than once in a template. If a value is used only once, a hard-coded value makes your template easier to read.
* Use variables for values that you construct from a complex arrangement of template functions. Your template is easier to read when the complex expression only appears in variables.
* Don't use variables for apiVersion on a resource. The API version determines the schema of the resource. Often, you can't change the version without changing the properties for the resource.
* Remove unused variables.

Outputs:

In the Outputs section of your template, you can specify values that will be returned after a successful deployment. You can use those outputs in your templates to return properties from the resources you are deploying. For example, it might be helpful to get the endpoints for a newly deployed storage account, or the public IP address or a newly deployed resource.

The Output section uses the reference function to get the runtime state of the resources. To get the runtime state of a resource, you pass in the name or ID of a resource.

there are certain elements that make up the outputs section.

"outputs": {

"<output-name>": {

"condition": "<boolean-value-whether-to-output-value>",

"type": "<type-of-output-value>",

"value": "<output-value-expression>",

"copy": {

"count": <number-of-iterations>,

"input": <values-for-the-variable>

}

}

}

* **output-name**: Must be a valid JavaScript identifier.
* **condition**: (Optional) Is a boolean value that indicates whether this output value is returned. When true, the value is included in the output for the deployment. When false, the output value is skipped for this deployment. When not specified, the default value is true.
* **Type**: The types of the output value supported are the same as the types of template input parameters.
  + NOTE: If you specify securestring for the output type, the value isn't displayed in the deployment history and can't be retrieved from another template. To use a secret value in more than one template, store the secret in a Key Vault and reference the secret in the parameter file. For more information, see Use Azure Key Vault to pass secure parameter value during deployment.
* **Value**: (Optional) Template language expression that is evaluated and returned as output value.
* **Copy**: (Optional) the Copy is used to return more than one value for an output.

For more information, and for the full documentation on outputs you can refer to [Output iteration in Azure Resource Manager templates](https://docs.microsoft.com/azure/azure-resource-manager/templates/copy-outputs?WT.mc_id=learnARM-github-frbouche).

And for a complete set of examples of outputs, refer to the [Outputs in Azure Resource Manager template](https://docs.microsoft.com/azure/azure-resource-manager/templates/template-outputs?WT.mc_id=learnARM-github-frbouche) document.

Controlling deployment using Dependency:

Let's say we want to deploy a website, something simple. In Azure we will need two resources: an App Service and a Service plan. The Service plan needs to be created before the App Service. We know Azure will paralyzed as much as possible during a deployment, to make sure the order is respected we need to use dependsOn.

Let's start a new ARM template. If you are using Visual Studio Code with the extension it will only take you a few clicks to get started.

1- Create a new file named temp.json 1- Type "!arm" to get the skeleton of the template. 1- In the resources section type arm-web-app

You should end-up with:

"resources": [

{

"name": "webApp1",

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

"apiVersion": "2018-11-01",

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

"tags": {

"[concat('hidden-related:', resourceGroup().id, '/providers/Microsoft.Web/serverfarms/appServicePlan1')]": "Resource",

"displayName": "webApp1"

},

"dependsOn": [

"[resourceId('Microsoft.Web/serverfarms', 'appServicePlan1')]"

],

"properties": {

"name": "webApp1",

"serverFarmId": "[resourceId('Microsoft.Web/serverfarms', 'appServicePlan1')]"

}

}

]

Notice the dependsOn property? It's expecting a Service plan 'appServicePlan1'. If you try to validate or deploy this template it would fail because it's expecting a resource 'appServicePlan1'.

Let's add the service plan. With the extension just type "arm-plan". Now the template should look like this. And even if it's not following the best practices we learn, it is perfectly valid. Even more, we are sure that the service plan will always be deployed before the app service.

{

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

"contentVersion": "1.0.0.0",

"parameters": {},

"functions": [],

"variables": {

},

"resources": [

{

"name": "webApp1",

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

"apiVersion": "2018-11-01",

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

"tags": {

"[concat('hidden-related:', resourceGroup().id, '/providers/Microsoft.Web/serverfarms/appServicePlan1')]": "Resource",

"displayName": "webApp1"

},

"dependsOn": [

"[resourceId('Microsoft.Web/serverfarms', 'appServicePlan1')]"

],

"properties": {

"name": "webApp1",

"serverFarmId": "[resourceId('Microsoft.Web/serverfarms', 'appServicePlan1')]"

}

},

{

"name": "appServicePlan1",

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

"apiVersion": "2018-02-01",

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

"sku": {

"name": "F1",

"capacity": 1

},

"tags": {

"displayName": "appServicePlan1"

},

"properties": {

"name": "appServicePlan1"

}

}

],

"outputs": {}

}

***Nested and Linked Template:***

As ARM deployments become more complex, using linked and nested templates allow you to break these deployments down into smaller reusable components.

* ***Linked templates:*** create reusable, composable, and modular deployments comprised of many individual arm templates.
* ***Nested templates:*** allows for advanced deployments scenarios like deploying to multiple ARM scopes or multiple resource groups from a single template file.

Linked template example:

{

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

"contentVersion": "1.0.0.0",

"parameters": {

"name": {

"type": "string",

"defaultValue": "linkeddemo001"

},

"location": {

"type": "string",

"defaultValue": "eastus"

}

},

"variables": {

"linked-template": "https://raw.githubusercontent.com/neilpeterson/arm-deployment-demo/master/linked-tempate/artifacts/storage.json",

"linked-template-two": "https://raw.githubusercontent.com/neilpeterson/arm-deployment-demo/master/linked-tempate/artifacts/identity.json"

},

"resources": [

{

"name": "storage",

"type": "Microsoft.Resources/deployments",

"apiVersion": "2019-10-01",

"properties": {

"mode": "Incremental",

"templateLink": {

"uri": "[variables('linked-template')]",

"contentVersion": "1.0.0.0"

},

"parameters": {

"name": {"value": "[parameters('name')]"},

"location": {"value": "[parameters('location')]"}

}

}

},

{

"name": "identity",

"type": "Microsoft.Resources/deployments",

"apiVersion": "2019-10-01",

"dependsOn": [

"[resourceId('Microsoft.Resources/deployments','storage')]"

],

"properties": {

"mode": "Incremental",

"templateLink": {

"uri": "[variables('linked-template-two')]",

"contentVersion": "1.0.0.0"

},

"parameters": {

"name": {"value": "[parameters('name')]"},

"location": {"value": "[parameters('location')]"}

}

}

}

],

"outputs": {

"storageURI": {

"type": "string",

"value": "[reference('storage').outputs.storageEndpoint.value]"

}

}

}

Nested template example:

{

"$schema": "https://schema.management.azure.com/schemas/2018-05-01/subscriptionDeploymentTemplate.json#",

"contentVersion": "1.0.0.0",

"parameters": {

"rgName": {

"type": "string",

"defaultValue": "nestedouter001"

},

"location": {

"type": "string",

"defaultValue":"eastus"

},

"storageName": {

"type": "string",

"defaultValue": "nestedouter001"

}

},

"resources": [

{

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

"type": "Microsoft.Resources/resourceGroups",

"apiVersion": "2019-10-01",

"location": "[parameters('location')]",

"tags": {

"displayname": "resource-group"

}

},

{

"name": "storage",

"type": "Microsoft.Resources/deployments",

"apiVersion": "2019-10-01",

"resourceGroup": "[parameters('rgName')]",

"dependsOn": [

"[resourceId('Microsoft.Resources/resourceGroups', parameters('rgName'))]"

],

"properties": {

"expressionEvaluationOptions": {

"scope": "outer"

},

"mode": "Incremental",

"template": {

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

"contentVersion": "1.0.0.0",

"resources": [

{

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

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

"apiVersion": "2019-06-01",

"location": "[parameters('location')]",

"kind": "StorageV2",

"sku": {

"name": "Premium\_LRS",

"tier": "Premium"

}

}

]

}

}

}

]

}