Table of Contents

[ARM TEMPLATE 2](#_Toc133691074)

[BENEFITS OF ARM TEMPLATE 2](#_Toc133691075)

[STRUCTURE OF ARM TEMPLATE 2](#_Toc133691076)

[CREATING ARM TEMPLATE 3](#_Toc133691077)

[VS CODE INTELLISENSE 3](#_Toc133691078)

[DEPLOY AN EMPTY TEMPLATE USING POWERSHELL 3](#_Toc133691079)

[CREATING STORAGE ACCOUNT 4](#_Toc133691080)

[CREATING MULTIPLE RESOURCES 6](#_Toc133691081)

[ARM TEMPLATE PARAMETERS 6](#_Toc133691082)

[OUTPUT PARAMETERS 7](#_Toc133691083)

[ARM TEMPLATE DEPLOYMENT MODE 9](#_Toc133691084)

[COMPLETE MODE 9](#_Toc133691085)

[INCREMENTAL MODE 9](#_Toc133691086)

[EXAMPLE 10](#_Toc133691087)

[COMMAND 10](#_Toc133691088)

[PREVIEW ARM TEMPLATE DEPLOYMENTS 10](#_Toc133691089)

[ARM TEMPLATE VALIDATION 11](#_Toc133691090)

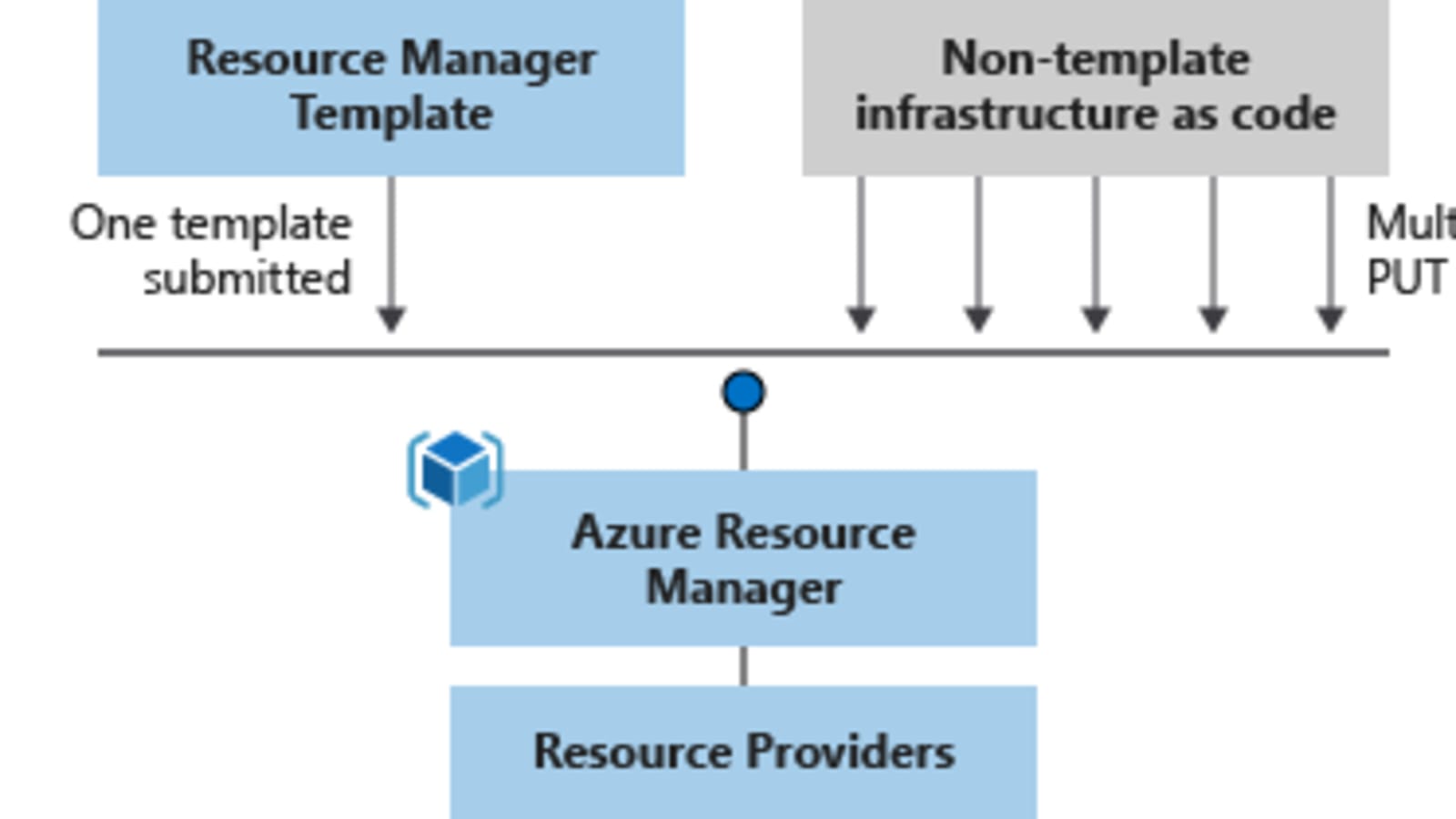
[INSTALLING ARM TEMPLATE TOOLKIT 11](#_Toc133691091)

# ARM TEMPLATE

* Infrastructure as code
* It’s a JSON file which we can be used to create the infrastructure and configuration to deploy the infrastructure on Azure.
* It has a **declarative syntax**. Its means that - we just define the configuration of the infrastructure we want to create. In declarative syntax (e.g ARM templates) we just specify the configuration of the infrastructure to be created without specifying the sequence in which they will be created.
* Note – When we want the command/operation to be executed in a certain order we make use of imperative syntax for example - PowerShell or any scripting language.

## BENEFITS OF ARM TEMPLATE

* THE APPLICATION CODE AND INFRASTRUCTURE CODE ARE PLACED IN THE SAME CODE (SAME REPO). The benefit of this approach is that if we have a different version an application which might have different infrastructure needs. We can have the application code and infrastructure put together for each version.
* **FASTER INFRASTRUCTURE CREATION**: When the template is submitted to ARM – the process are executed in parallel which make infrastructure creation faster.
* BUILT IN VALIDATION – Azure resource manager has a built in validation feature that validate the templates before execute them on Azure
* REUSABILITY
* TRACEABILITY : We can able to trace status of the template execution for example - if something fails, we can see what is the problem where this template is failing
* CI / CD FRIENDLY



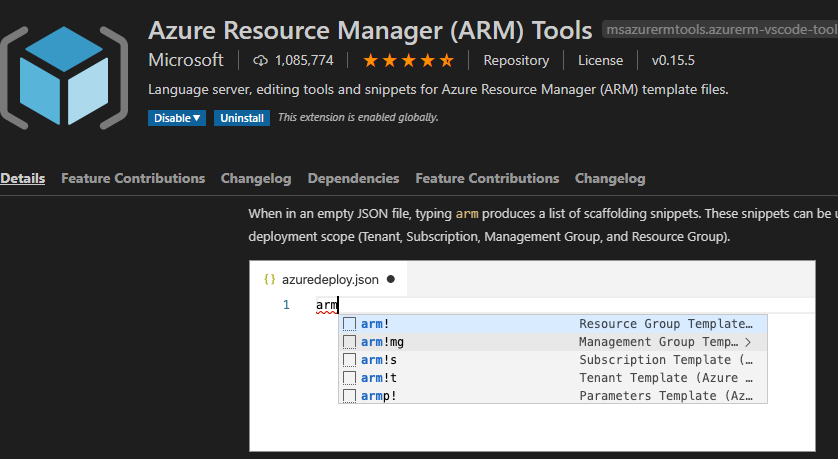
## STRUCTURE OF ARM TEMPLATE

|  |
| --- |
| {  "$schema": "",  "contentVersion": "1.0.0.0",  "parameters": {},  "functions": [],  "variables": {},  "resources": [],  "outputs": {}  } |

|  |  |
| --- | --- |
| **contentVersion** |  |
| **parameters** | Provide values during deployment that allow the same template to be used with different environments. |
| **functions** | Create customized functions that simplify your template. |
| **variables** | Define values that are reused in your templates. They can be constructed from parameter values. |
| **resources** | Specify the resources to deploy. |
| **outputs** | Return values from the deployed resources. |

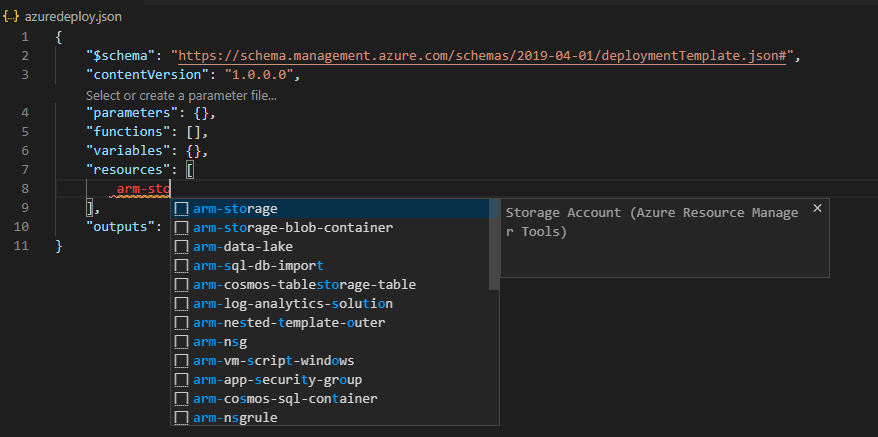
## CREATING ARM TEMPLATE

* VS CODE EXTENSION



### VS CODE INTELLISENSE

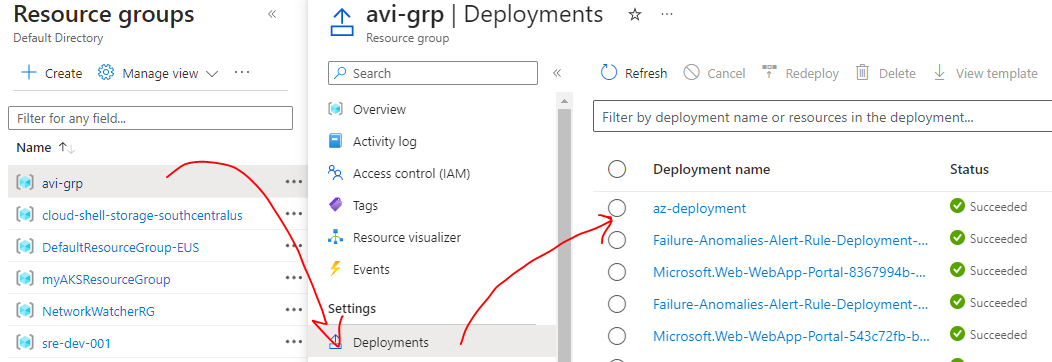
* The VS code extension has a support for ARM template.

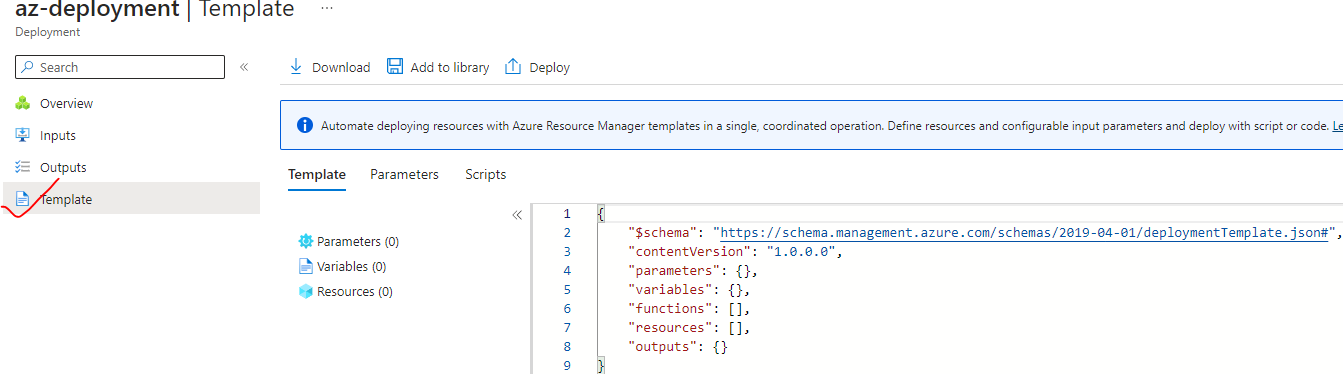


### DEPLOY AN EMPTY TEMPLATE USING POWERSHELL

|  |  |  |
| --- | --- | --- |
| We can use Power Shell in (run as administrator) to deploy the ARM template. Let deploy am empty template | | {  "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",  "contentVersion": "1.0.0.0",  "parameters": {},  "functions": [],  "variables": {},  "resources": [],  "outputs": {}  } |
| POWERSHELL COMMAND TO SUBMIT THE TEMPLATE FILE | New-AzResourceGroupDeployment -Name az-deployment -ResourceGroupName **avi-grp** -TemplateFile .\**azuredeploy.json** | |
|  | | |

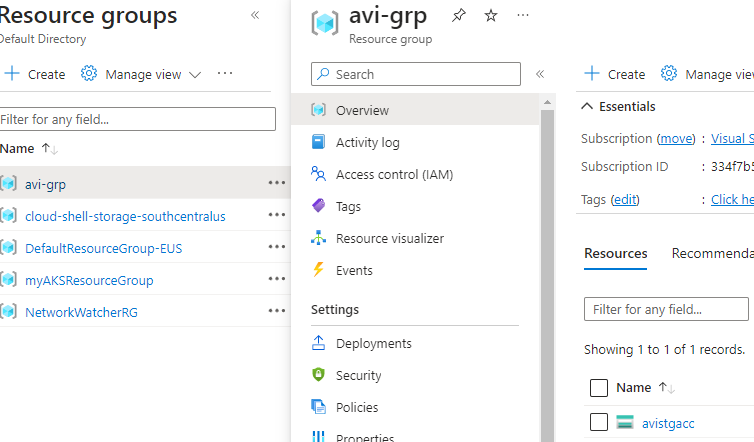
* In portals we can validate the deployment as well. Go to the **resource group 🡪 Deployment**
* As it an empty deployment there is no resource created.
* Note the deployment mode is “incremental” means –every time you run it, it continues the previous one. So, it's not going to remove what we have done before and add to that if there is something new.

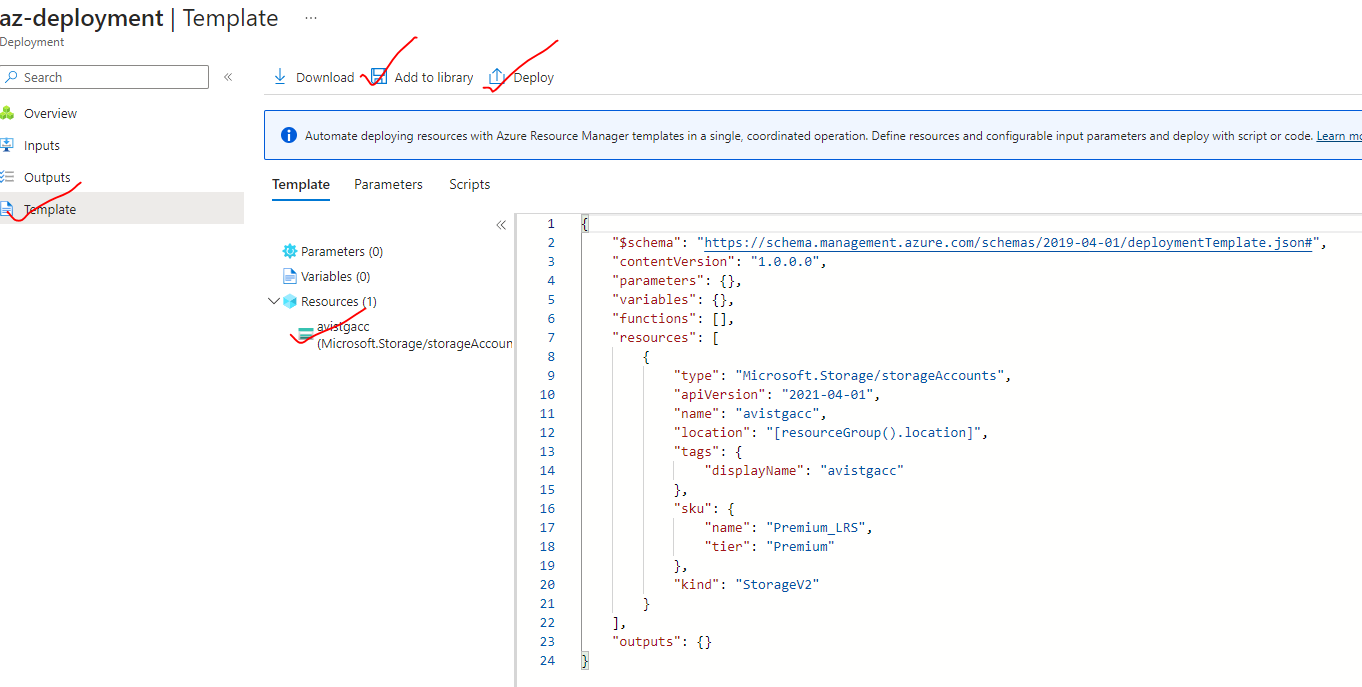




## CREATING STORAGE ACCOUNT

|  |
| --- |
| {  "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",  "contentVersion": "1.0.0.0",  "parameters": {},  "functions": [],  "variables": {},  "resources": [  {  "name": "**avistgacc**",  "type": "Microsoft.Storage/storageAccounts",  "apiVersion": "2021-04-01",  "tags": {  "displayName": "**avistgacc**"  },  "location": "[resourceGroup().location]",  "kind": "StorageV2",  "sku": {  "name": "Premium",  "tier": "Premium"  }  }  ],  "outputs": {}  } |





|  |  |
| --- | --- |
| DEPLOY | **The template can also be deployed directly from Azure portal from deploy option** |
| ADD TO LIBRARY |  |

## CREATING MULTIPLE RESOURCES

|  |  |
| --- | --- |
| * When resources like VMs are created – It in turn create multiple resources like IP, NSG , NIC etc. These resources are created in a specific order. * For example – The VM got created only after the creation of dependent resources * Note – It does not matter – in what order it has been written in template |  |

* These dependencies are maintained by using “dependsOn” property. As shown below.
* The VM can only be created after the creation of “Network Interface” and “Storage Account”

|  |
| --- |
| ....  {  “name”: “appvm250423”,  “type”: “Microsoft.Compute/virtualMachines”,  “apiVersion”: “2021-03-01”,  “location”: “[resourceGroup().location]”,  “dependsOn”: [  “[resourceId(‘Microsoft.Storage/storageAccounts’, toLower(‘appvm250423storage’))]”,  “[resourceId(‘Microsoft.Network/networkInterfaces’, ‘appvm250423-NetworkInterface’)]”  ],  .... |
| * The dependOn resource are configured using “**resourceId**” function   **resourceId(<RESOURCE\_TYPE>,<RESOURCE\_NAME>)** |

## ARM TEMPLATE PARAMETERS

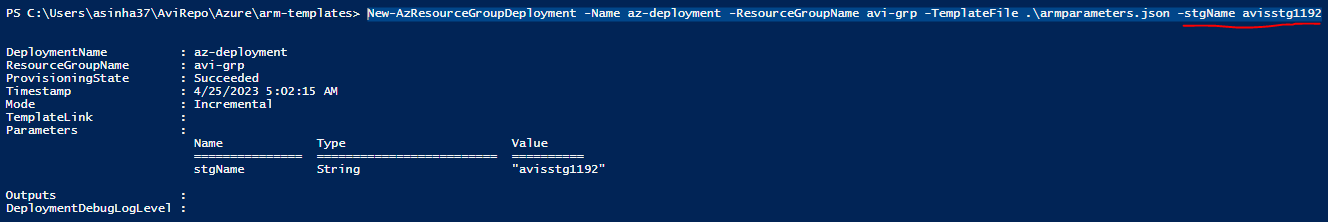
* Parameterized ARM template is reusable. We can create multiple similar resources by passing the different parameters. For example, to create Storage account.
* ARM parameters : <https://learn.microsoft.com/en-us/azure/azure-resource-manager/templates/parameters>
* Parameter type can be – ***string, secureString, integers,boolean, object, secureObject , array***

**STEP 1: CREATE ARM TEMPLATE WITH PARAMETERS**

|  |
| --- |
| {  "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",  "contentVersion": "1.0.0.0",  "parameters": {  "**stgName**": {  "type": "string",  "metadata": {  "description": "description"  }  }  },  "functions": [],  "variables": {},  "resources": [  {  "name": "[parameters(**'stgName'**)]",  "type": "Microsoft.Storage/storageAccounts",  "apiVersion": "2021-04-01",  "tags": {  "displayName": "[parameters(**'stgName'**)]"  },  "location": "[resourceGroup().location]",  "kind": "StorageV2",  "sku": {  "name": "Premium\_LRS",  "tier": "Premium"  }  }  ],  "outputs": {}  } |

**STEP 2: PASSING THE PARAMETER VALUE FROM POWERSHELL COMMAND**

**New-AzResourceGroupDeployment -Name az-deployment -ResourceGroupName avi-grp -TemplateFile .\armparameters.json -stgName avisstg1192**

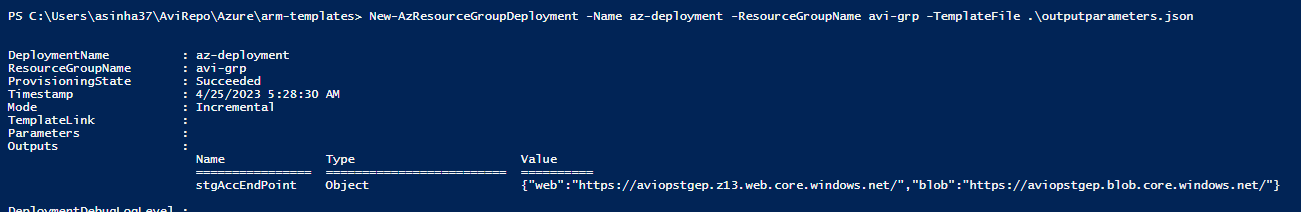


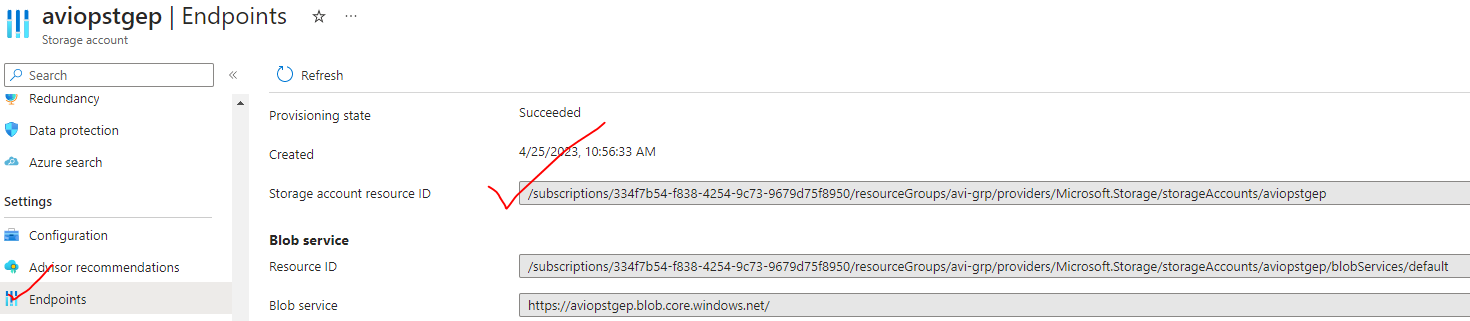
## OUTPUT PARAMETERS

* We use outputs when we need to return values from the deployed resources.
* <https://learn.microsoft.com/en-us/azure/azure-resource-manager/templates/outputs?tabs=azure-powershell>

|  |
| --- |
| {  "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",  "contentVersion": "1.0.0.0",  "parameters": {},  "functions": [],  "variables": {},  "resources": [  {  "name": "aviopstgep",  "type": "Microsoft.Storage/storageAccounts",  "apiVersion": "2021-04-01",  "tags": {  "displayName": "aviopstgep"  },  "location": "[resourceGroup().location]",  "kind": "StorageV2",  "sku": {  "name": "Premium\_LRS",  "tier": "Premium"  }  }  ],  "outputs": {  "stgAccEndPoint": {  "type": "object",  "value": "[reference('aviopstgep').primaryEndpoints]"  }  }  } |

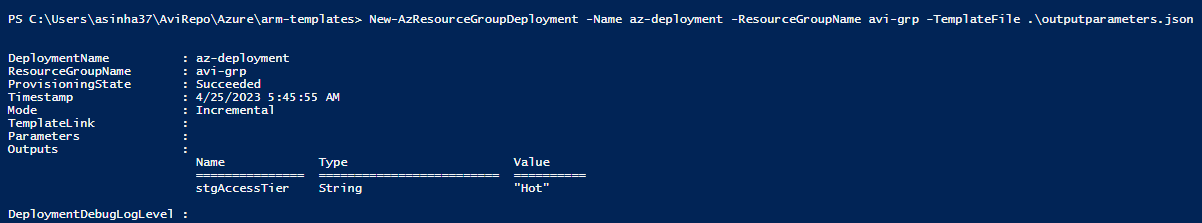
POWERSHELL OUTPUT





EXAMPLE

|  |  |
| --- | --- |
| TO GET ACCESS TIER OF STORAGE ACCOUNT | "outputs": {  "stgAccessTier": {  "type": "string",  "value": "[reference('aviopstgep').**AccessTier**]"  }  } |



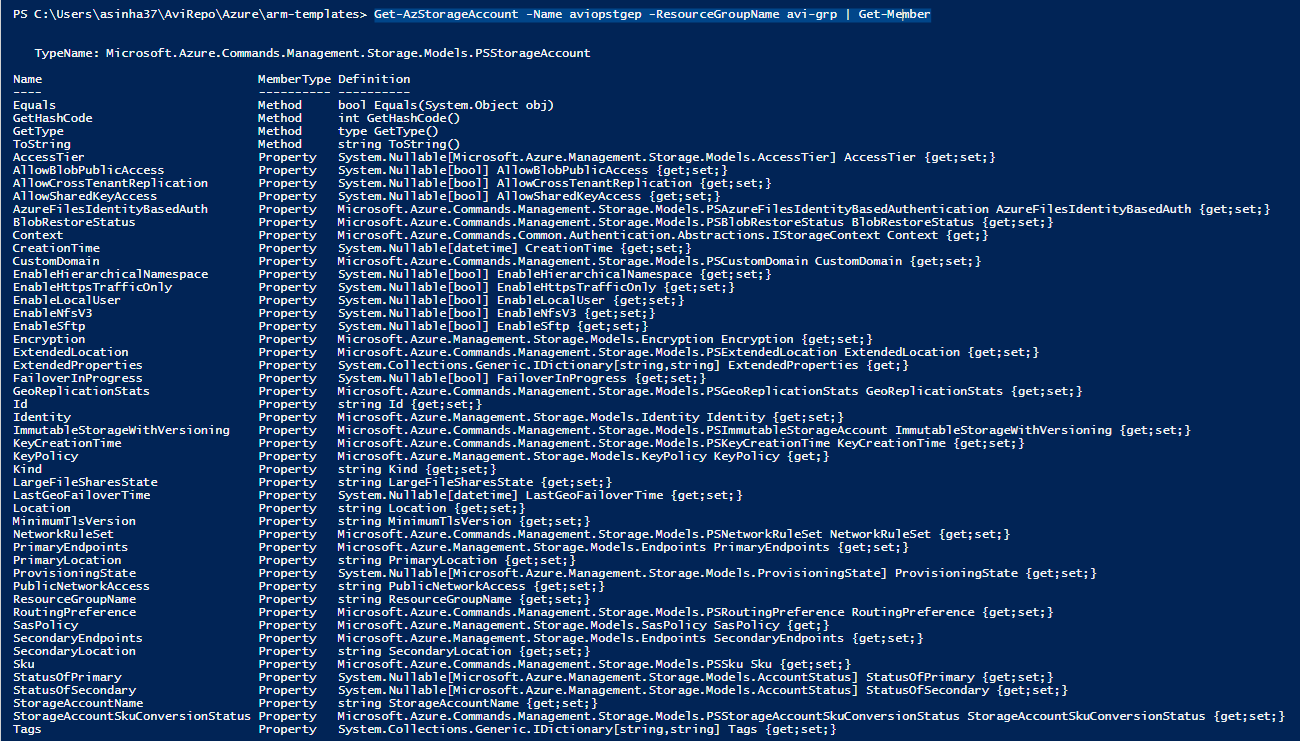
Question – What are the different output property can be used?

Answer – For example –in case of Storage account - The list of output properties can be retrieved using following

**STEP 1: GET THE MEMBER/ PROPERTY OF STORAGE ACCOUNT USING FOLLOWING COMMAND**

*Get-AzStorageAccount -Name aviopstgep -ResourceGroupName avi-grp | Get-Member*

STEP 2: USE THE PROPERTY WITH IT CORRECT TYPE



## ARM TEMPLATE DEPLOYMENT MODE

* When deploying our resources, we specify that the deployment is either an
  + **INCREMENTAL UPDATE**
  + **COMPLETE UPDATE**.
* The difference between these two modes is how Resource Manager handles existing resources in the resource group that aren't in the template.
* For both modes, Resource Manager tries to create all resources specified in the template. **If the resource already exists in the resource group and its settings are unchanged, no operation is taken for that resource**. If you change the property values for a resource, the resource is updated with those new values.
* Note - If you try to update the location or type of an existing resource, the deployment fails with an error. Instead, deploy a new resource with the location or type that you need.
* ***The default mode is incremental***.

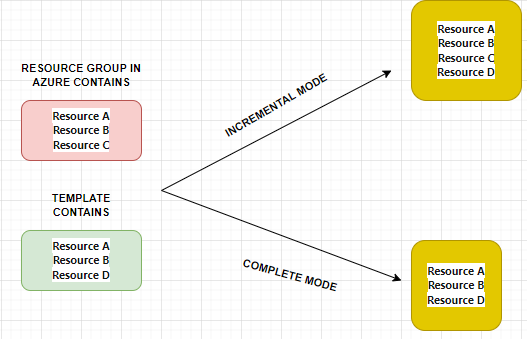
### COMPLETE MODE

* In complete mode, Resource Manager **deletes** resources that exist in the resource group but aren't specified in the template.

### INCREMENTAL MODE

* In incremental mode, Resource Manager **leaves unchanged** resources that exist in the resource group but aren't specified in the template. Resources in the template **are added** to the resource group.

### EXAMPLE



### COMMAND

|  |  |
| --- | --- |
| POWERSHELL | New-AzResourceGroupDeployment `  **-Mode Complete** `  -Name ExampleDeployment `  -ResourceGroupName ExampleResourceGroup `  -TemplateFile c:\MyTemplates\storage.json |
| AZURE CLI | az deployment group create \  **--mode Complete** \  --name ExampleDeployment \  --resource-group ExampleResourceGroup \  --template-file storage.json |

### PREVIEW ARM TEMPLATE DEPLOYMENTS

* We can be able to preview the deployment – it help in avoiding unforeseen issues.
* Azure Resource Manager provides the **what-if operation** to highlight the changes when you deploy a template. The what-if operation doesn't make any changes to existing resources. Instead, it predicts the changes if the specified template is deployed at a resource group and subscription level.
* Using the what-if operation compares the current state model to the desired state model. The what-if operation confirms if the changes made by your template match your expectations *without* applying those changes to real resources or to the state of those resources.

|  |  |
| --- | --- |
| PREVIEW/ PREDICT DEPLOYMENT using WhatIf  <https://learn.microsoft.com/en-us/training/modules/arm-template-whatif/3-what-if?pivots=bicepcli> | **New-AzResourceGroupDeployment -Name az-dpmode -ResourceGroupName avi-grp -TemplateFile .\deploymentmode.json -Mode Complete -WhatIf** |

#### CONTROLLING THE OUTPUT OF THE DEPLOYMENT PREVIEW

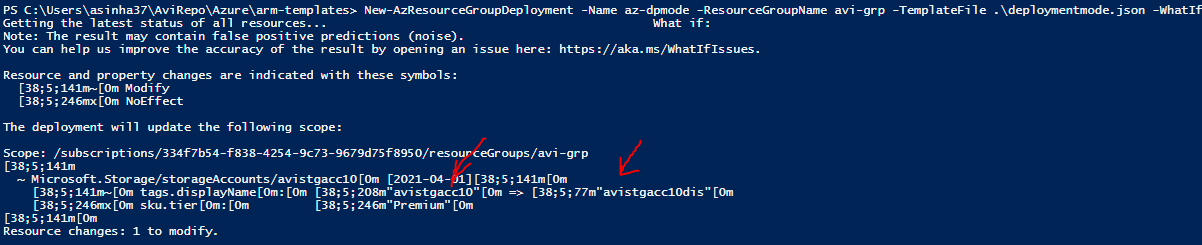
We can control the amount of text output of the what-if operation by using one of these result formats:

1. **FullResourcePayloads**. By including this parameter, we get a verbose output that consists of a list of resources that will change. The output also shows details about all the properties that will change in accordance with the template.
2. **ResourceIdOnly**. This mode returns a list of resources that will change, but not all the details.

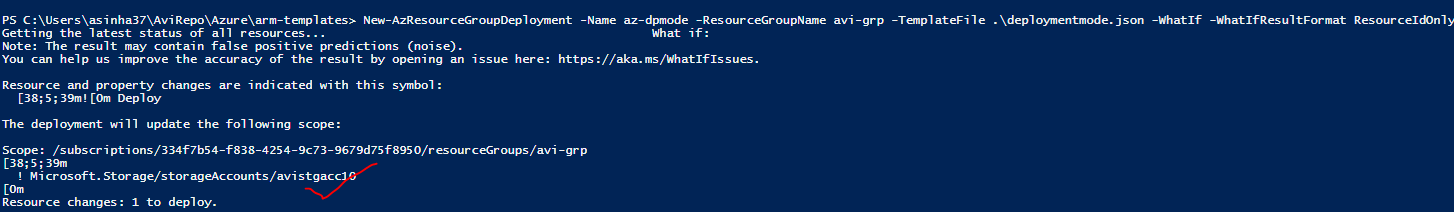
##### EXAMPLE

For example, assume that you're changing the storage type in a template that deploys a single storage account to an existing environment.

1. **COMMAND:** New-AzResourceGroupDeployment -Name az-dpmode -ResourceGroupName avi-grp -TemplateFile .\deploymentmode.json -WhatIf -WhatIfResultFormat **FullResourcePayloads**

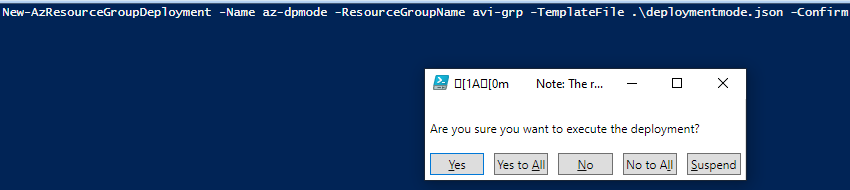


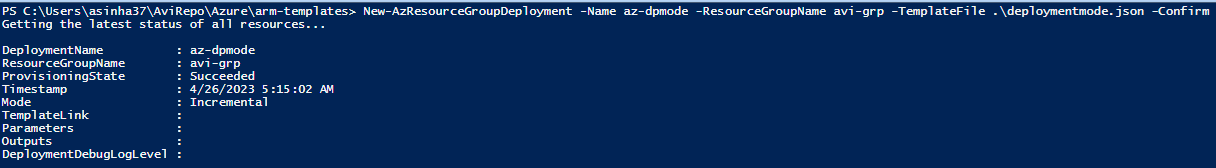
1. **COMMAND:** New-AzResourceGroupDeployment -Name az-dpmode -ResourceGroupName avi-grp -TemplateFile .\deploymentmode.json -WhatIf -WhatIfResultFormat **ResourceIdOnly**



1. **COMMAND:** New-AzResourceGroupDeployment -Name az-dpmode -ResourceGroupName avi-grp -TemplateFile .\deploymentmode.json -**Confirm**

This command will ask for the confirmation before making the changes to the resources





## ARM TEMPLATE VALIDATION

* For ARM templates - we need a validation tool to avoid mistakes and for best practices. The validation help us in avoiding issues like security (sensitive information like passwords tokens,secrets etc.)
* For validation of ARM template we make use of - **ARM Template Test Toolkit**.( <https://learn.microsoft.com/en-us/azure/azure-resource-manager/templates/test-toolkit> )

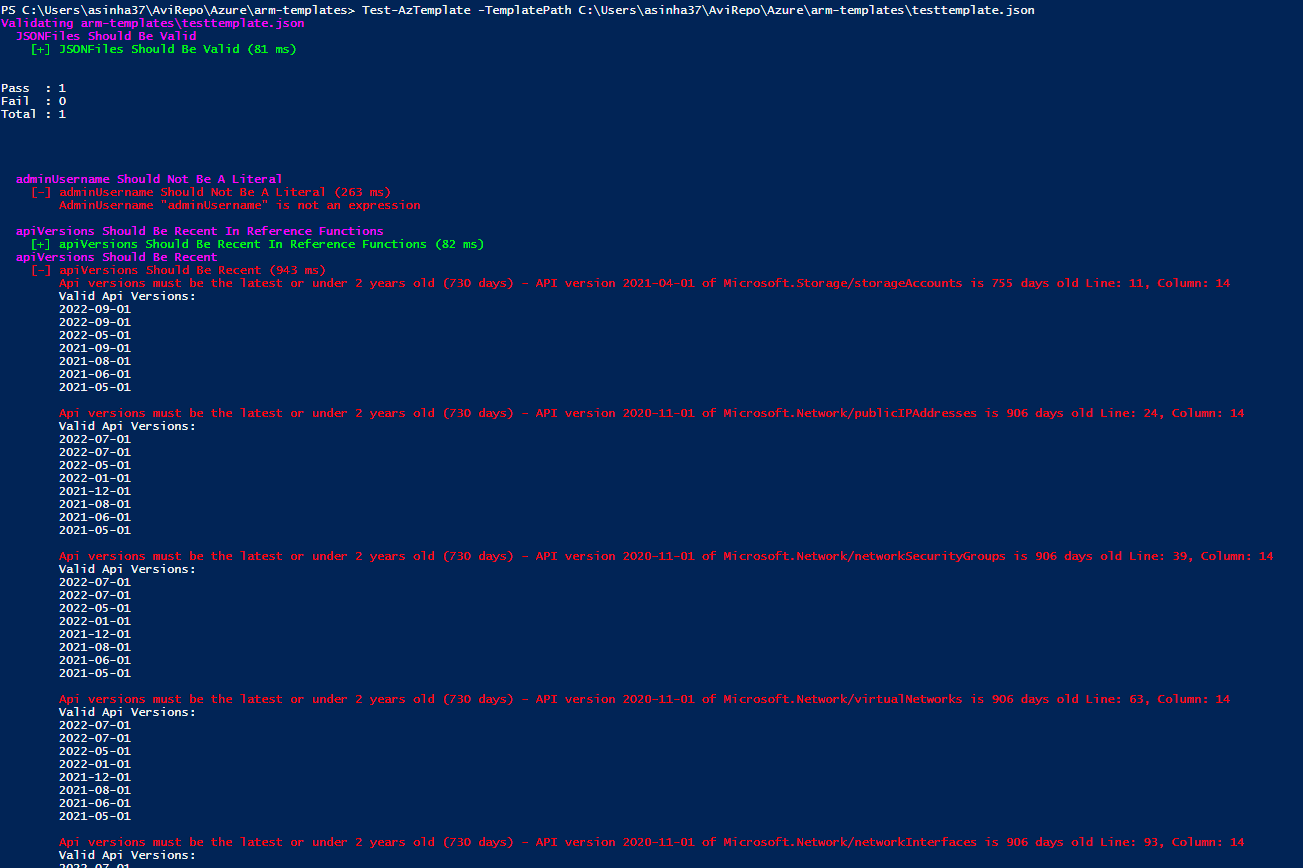
### INSTALLING ARM TEMPLATE TOOLKIT

* Step 1: Clone the Repo : <https://github.com/Azure/arm-ttk.git>
* Step 2: Import the module in PowerShell : **Import-Module .\arm-ttk\arm-ttk.psd1**
* Step 3: Run the test for any template : **Test-AzTemplate -TemplatePath \path\to\template**

#### EXAMPLE

|  |  |
| --- | --- |
|  | * In this example – lets make use of default template provided by VS code for a VM. * As per validation rule hardcoding the credential is not recommended. Hence the toolkit will flag an error while running the validation |

ARM TOOL KIT OUTPUT



#### FIXING THE TEMPLATE ISSUES