* ARM (Azure Resource Manager) template is a block of code that defines the infrastructure and configuration for your project.
* These templates use a declarative syntax to let you define your deployment in the form of JSON (JavaScript Object Notation) files.
* All the resources and their properties are defined in ARM template. This helps in automating the deployment process in a constant flow.

**Advantages of ARM Templates**

* Using ARM Templates, we can declare network infrastructure, storage and any other resources.
* Over the development lifecycle, ARM Templates allows the deployment of resources repeatedly in a consistent manner.
* User can deploy templates parallelly, and only one command is sufficient to deploy all your resource settings.
* Templates can be divided into different modules. In other words, templates can be broken into multiple templates so that a parent template can consist of small templates.
* The PowerShell or Bash Scripts can be added to the templates using deployment scripts.
* The working of ARM Templates can be tested using the ARM template toolkit.
* A user can see the preview of the template. All the resources that are being created or deleted in this template will be shown in the preview.
* A user can integrate templates with Continuous Integration (CI) and Continuous Deployment (CD) tools to automate the release.

## Understanding ARM Templates

**Template Format**

The ARM Templates file contains various key-value pairs in the JSON format. For example, below, you can see a format of an ARM Templates.

**{**

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

**"contentVersion": "1.0.0",**

**"parameters":{},**

**"variables":{},**

**"functions":[],**

**"resources": [],**

**"outputs":{}**

**}**

**Schema**

This ‘schema’ defines the location of the JSON file and specifies the version of the template language that you want to use in this template. This schema depends on the purpose of your deployment. Some schema types are listed below.

* Resource Group Deployment – https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#
* Subscription Group Deployment – https://schema.management.azure.com/schemas/2018-05-01/subscriptionDeploymentTemplate.json#
* Management Group Deployment – https://schema.management.azure.com/schemas/2019-08-01/managementGroupDeploymentTemplate.json#
* Tenant Group Deployment – https://schema.management.azure.com/schemas/2019-08-01/tenantDeploymentTemplate.json#

**Content Version**

It specifies the version of the templates. This version can be any number that you want to give to your template.

**Parameters**

Parameters in a templates define the configuration. These parameters are used in run time or during deployment. In a parameter, we need to define the name, type, values and properties. We can also set some allowed values and default values to the parameters, so when a value is not passed during deployment, then the default or allowed values will be used. Below is an example of parameters that defines the type and the default value of username and password for the VM (Virtual Machine).

"parameters": {

"adminUsername": {

"type": "string",

"defaultValue": "Admin",

"metadata": {

"description": "Username for the Virtual Machine."

}

},

"adminPassword": {

"type": "securestring",

"defaultValue": "12345",

"metadata": {

"description": "Password for the Virtual Machine."

}

}

}

**Variables**

Variables define values used throughout the template. In simple words, you can define a short name for a specific value that can be used anywhere in the template. Variables also become an advantage when you want to update all the values and reference in a template. Then you can update the variable and its value only.

"variables": {

"nicName": "myVMNic",

"addressPrefix": "10.0.0.0/16",

"subnetName": "Subnet",

"subnetPrefix": "10.0.0.0/24",

"publicIPAddressName": "myPublicIP",

"virtualNetworkName": "MyVNet"

}

**Functions**

In a template, the function contains the steps and procedures to be followed. It is just like a variable that defines the steps performed when called in a templates. The below example of the function defines the unique name for the resources.

"functions": [

{

"namespace": "contoso",

"members": {

"uniqueName": {

"parameters": [

{

"name": "namePrefix",

"type": "string"

}

],

"output": {

"type": "string",

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

}

}

}}],

**Resources**

All the azure resources are defined here that makes the deployment. For creating a resource, we need to set up the type, name, location, version and properties of the resource that needs to be deployed. We can also use the variables and parameters here that are defined in the ‘variables’ section. Below is the example of declaring the resources in a templates.

"resources": [

{

"type": "Microsoft.Network/publicIPAddresses",

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

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

"apiVersion": "2018-08-01",

"properties": {

"publicIPAllocationMethod": "Dynamic",

"dnsSettings": {

"domainNameLabel": "[parameters('dnsLabelPrefix')]"

}

}

}

],

**Outputs**

Output defines the result that you want to see when a template runs. In simple words, the final words that you want to see when a template is successfully deployed. In the below example, the hostname with a value fetched from the public IP address name.

"outputs": {

"hostname": {

"type": "string",

"value": "[reference(variables('publicIPAddressName')).dnsSettings.fqdn]"

}

}