Agenda: IaC using ARM Templates

- About Infrastructure as Code (IaC)
- About ARM Templates
- Sample to Create Storage Account using ARM Template
- Deploy Templates using PowerShell
- Deploy Templates using Azure Portal
- Deploy Templates using Azure Pipeline
- Incremental and Complete Deployment
- Creating VM using ARM Template

About Infrastructure as Code (IaC)

This module is about expressing your infrastructure requirements as code. By adopting this practice, you can automatically provision the infrastructure that you need for your application or service.

Keeping up with changes in your infrastructure is a challenge. New resources need to be built. Old resources need maintenance. Environments need to be consistent. *Infrastructure as code* enables you to describe, through code, the infrastructure that you need for your application.

What is infrastructure as code (IaC)?

Infrastructure as code enables you to describe, through code, the infrastructure that you need for your application.

Infrastructure as code enables you to maintain both your application code and everything you need to deploy your application in a central code repository. In that repository, you can build, test, and deploy as a unit.

Your infrastructure and configuration code is stored in version control, along with the application code. You'll have lots more wall space.

As you move to infrastructure as code, you're going to see a lot of improvements.

- Consistent configurations
- Improved scalability
- Faster deployments
- Less documentation because the scripts replace it
- Better traceability

Infrastructure as code is a best practice in DevOps because it helps bring teams together. The development team gets a better understanding of where their code is being deployed to and run. Operations teams get involved in the process much earlier and can use infrastructure choices to help developers make the right design decisions.

Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

What tools can you use to apply infrastructure code?

- Azure CLI
- Azure PowerShell
- Azure SDK
- Azure Resource Manager templates
- Terraform
- Ansible

Azure Resource Manager templates

About Azure Resource Manager:

The infrastructure that makes up your application is often composed of various different components. For instance, you might simply be running a web site, but behind the scenes you have an Azure web site deployed, a Storage account for tables, blobs, and queues, a couple of VMs running a database cluster, etc…

The old way of doing things was to use the **Service Management API**. Each piece of infrastructure was **treated separately** in the Azure Portal. Deployment consisted of manually creating them, or lots of effort creating scripts using PowerShell. You would have to handle trying to initialize infrastructure in serial or parallel yourself. If you wanted to deploy just part of your infrastructure you would have to understand **what was already there** and take actions accordingly, either manually or in your scripts.

Azure Resource Manager Overview:

Azure Resource Manager enables you to work with the resources in your solution as a group. You can deploy, update, or delete all the resources for your solution in a single, coordinated operation.

- **Azure Resource Manager** allows you to define **a declarative template** for your group of resources.
- A resource manager template is **a JavaScript Object Notation (JSON)** file that defines one or more resources to deploy to a resource group.
- You use that template for deployment and it can work for different environments such as **testing, staging, and production** and have confidence your resources are deployed in a consistent state.
- You can define the dependencies between resources so they're deployed in **the correct order**.
- Azure handles things like **parallelizing** as much of the deployment as possible without any extra effort on your part.
- It also deploys in an **idempotent** way i.e. **incrementally adds** anything **missing** while leaving anything already deployed in place, so you can **rerun** the template deployment multiple times safely.

Sample to Create a Storage Account: Template.json

```
{
```

```
"$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
"contentVersion": "1.0.0.0",
"parameters": {
 "storageName": {
  "type": "string",
  "minLength": 3,
  "maxLength": 24
 },
 "storageSKU": {
  "type": "string",
  "defaultValue": "Standard_LRS",
  "allowedValues": [
   "Standard_LRS",
   "Standard_GRS",
   "Standard_RAGRS",
   "Standard_ZRS",
   "Premium_LRS",
   "Premium_ZRS",
   "Standard_GZRS",
   "Standard_RAGZRS"
  ]
 },
 "location": {
  "type": "string",
  "defaultValue": "[resourceGroup().location]"
 }
},
"resources": [
 {
  "type": "Microsoft.Storage/storageAccounts",
  "apiVersion": "2019-04-01",
  "name": "[parameters('storageName')]",
  "location": "[parameters('location')]",
  "sku": {
   "name": "[parameters('storageSKU')]"
  },
```

```
    "kind": "StorageV2",
    "properties": {
      "supportsHttpsTrafficOnly": true
    }
   }
 ],

  "outputs": {
     "storageUri": {
        "type": "string",
        "value": "[reference(parameters('storageName')).primaryEndPoints.blob]"
     }
  }
}
```

Sample: **Parameters.json**

```
{
 "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentParameters.json#",
 "contentVersion": "1.0.0.0",
 "parameters": {
   "storageName": {
     "value": "devstore"
   },
   "storageSKU": {
     "value": "Standard_LRS"
   }
 }
}
```

Deploy Templates using PowerShell

Azure Portal → Cloud Shell → Upload file

PowerShell commands to Execute the ARM Template

```
$templateFile = "$Home/Storage.json"
$parameterFile="$Home/Storage-parameters.json"
New-AzResourceGroup `
 -Name myDemoRG `
```

```
-Location "East US"

New-AzResourceGroupDeployment `
-Name devenvironment `
-ResourceGroupName myDemoRG `
-TemplateFile $templateFile `
-TemplateParameterFile $parameterFile
```

Note: Validate your deployment settings to find problems before creating actual resources.

```
Test-AzResourceGroupDeployment -ResourceGroupName Sandbox-rg -TemplateFile <PathToTemplate>
```

You have the following options for providing parameter values:

```
New-AzResourceGroupDeployment -Name ExampleDeployment -ResourceGroupName
ExampleResourceGroup -TemplateFile <PathToTemplate> -TemplateParameterFile $parameterFile -
myParameterName1 "parameterValue1" -myParameterName2 "parameterValue2"
```

Deploy the Template using Portal

To Save the template:

1. More Services → Template spec → +Add
2. Name="DemoVMTemplate", Description="This is a test template" → OK
3. Copy the content of Storage.json and paste in the Text Editor → OK
4. Click Add.

To Execute the Template

5. More Services → Search for "**Template spec**"
6. Select the template with the name you saved earlier (DemoVMTemplate)**.**
7. Template Blade → Deploy
8. Click on Edit parameters → Copy and paste content from Storage-Parameters.json → Save
9. Create a New RG and change the parameters as needed → Check I agree . . . → Purchase

CI with Azure Pipeline

1. Add the below two files to Git repository
2. Put the following in **BUILD** Stage

```
- task: CopyFiles@2
  displayName: 'Copy Files to: $(build.artifactstagingdirectory)'
  inputs:
    SourceFolder: HelloWorldApp.Templates
```

```
  TargetFolder: '$(build.artifactstagingdirectory)'
```

3. Put the following in Dev stage

```
- task: AzureResourceManagerTemplateDeployment@3
  displayName: 'ARM Template deployment: Resource Group scope'
  inputs:
    azureResourceManagerConnection: 'Azure Subs MPN 2'
    subscriptionId: 'f9baec73-91eb-4458-bd0d-965c1973526d'
    resourceGroupName: DemoRG
    location: 'East US'
    csmFile: '$(System.DefaultWorkingDirectory)/**/Storage-Template.json'
    csmParametersFile: '$(System.DefaultWorkingDirectory)/**/Storage-Parameters.json'
    templateLocation: 'Linked artifact'
    deploymentMode: 'Incremental'
```

4. Save and Run the pipeline

Incremental and Complete Deployments

When deploying your resources, you specify that the deployment is either an **incremental** update or a **complete** update.

- In complete mode, Resource Manager **deletes** resources that exist in the resource group but are not specified in the template.
- In incremental mode, Resource Manager **leaves unchanged** resources that exist in the resource group but are not specified in the template.

Existing Resource Group contains:

- Resource A
- Resource B
- Resource C

New Template defines:

- Resource A
- Resource B
- Resource D

When deployed in **incremental** mode, the resource group contains:

- Resource A
- Resource B
- Resource C

- Resource D

When deployed in **complete** mode, Resource C is deleted. The resource group contains:

- Resource A
- Resource B
- Resource D

PowerShell Command:

```
New-AzResourceGroupDeployment -Name "ExampleDeployment123" -Mode Complete -ResourceGroupName DemoRG -TemplateFile "D:\template.json" -TemplateParameterFile 'D:\parameters.json'
```

CLI Command:

```
az deployment group create --name ExampleDeployment --mode Complete --resource-group DemoRG --template-file template.json --parameter-file @parameters.json
```

Create Windows VM using ARM Template

1. Open the following files from DevOps folder
   - AzureVMTemplate.json
   - AzureVMTempalte-parameters.json

There are five resources defined by the template:

1. Microsoft.Storage/storageAccounts.
2. Microsoft.Network/publicIPAddresses.
3. Microsoft.Network/virtualNetworks.
4. Microsoft.Network/networkInterfaces.
5. Microsoft.Compute/virtualMachines.

2. Using Cloud Shell, upload the file AzureVMTempalte.json
3. Copy and Paste the following in Cloud Shell

```
$resourceGroupName = Read-Host -Prompt "Enter the resource group name"
$storageAccountName = Read-Host -Prompt "Enter the storage account name"
$newOrExisting = Read-Host -Prompt "Create new or use existing (Enter new or existing)"
$location = Read-Host -Prompt "Enter the Azure location (i.e. centralus)"
$vmAdmin = Read-Host -Prompt "Enter the admin username"
$vmPassword = Read-Host -Prompt "Enter the admin password" -AsSecureString
$dnsLabelPrefix = Read-Host -Prompt "Enter the DNS Label prefix"
```

```
New-AzResourceGroup -Name $resourceGroupName -Location $location
New-AzResourceGroupDeployment `
  -ResourceGroupName $resourceGroupName `
  -adminUsername $vmAdmin `
  -adminPassword $vmPassword `
  -dnsLabelPrefix $dnsLabelPrefix `
  -storageAccountName $storageAccountName `
  -newOrExisting $newOrExisting `
  -TemplateFile "$HOME/AzureVMTemplate.json"
```

Integrate Key Vault

open *azuredeploy.parameters.json*

```
"adminPassword": {
  "reference": {
    "keyVault": {
    "id":
"/subscriptions/<SubscriptionID>/resourceGroups/<ResGroup>/providers/Microsoft.KeyVault/vaults/<KeyVaultName>"
    },
    "secretName": "vmAdminPassword"
  }
},
```