Azure IaC – (Infrastructure as code)

## **What is Infrastructure as code (IaC)?**

Traditionally, development and operations are two distinct departments in an Independent Software Vendor, developers concern themselves with writing applications, and the folks in operations are concerned with keeping the applications running. However, for an application to function correctly, there are always explicit or implicit requirements regarding how the supporting infrastructure is configured. Unfortunately, such requirements are often lost during communication, which leads to many problems such as service outages because of misconfigurations, frictions between development and operations, and difficulties in recreating and diagnosing issues. All these problems are unacceptable in an Agile environment.

## **Why Infrastructure as Code (IaC)?**

The boundary between development and operations is shifting. The developers are required to provide consistently deployable applications instead of just application code; thus, the deployment process can be automated to rollout fixes and upgrades quickly. This shift changed the definition of application. An application is no longer just code. Instead, an application is made up of both application code and explicit, executable description of its infrastructural requirements.

## **What are the benefits of Infrastructure as Code (IaC)?**

1. Consistency: Consistently achieve standardised provisioning or deployment  
2. Accelerating: Accelerating provisioning or deployment rapidly  
3. Reusability: JSON code for repeatable or similar provisioning or deployment  
4. Extensibility: JSON code for incorporating with additional items

Have any developer ever waited for weeks or months to get a development environment delivered by their internal IT operations? This demand has driven IT industry to deliver new tool set to ensure that one can deliver on time and meet the standard requirement constantly.

## **How Infrastructure is implemented**

A short list of some of the Infrastructure as Code frameworks / platforms available

* PowerShell & PowerShell Desired State Configuration (DSC)
* Azure Resource Manager Portal
* Puppet
* Chef
* Otter
* SaltStack
* Ansible
* CFEngine

There are many Infrastructure as Code platforms / frameworks available. Each one will have its own set of benefits and drawbacks compare to the others, as is the case when comparing any set of similar tools.

On the Windows and Microsoft Azure platforms, the answers to implementing Infrastructure as Code provided by Microsoft are largely the PowerShell DSC, and even more so with Azure the Azure Resource Manager.

When implementing Infrastructure as Code in the Microsoft Azure and [Azure Stack](https://azure.microsoft.com/en-in/overview/azure-stack/) platform ecosystem the Azure Resource Manager is the tool of choice.

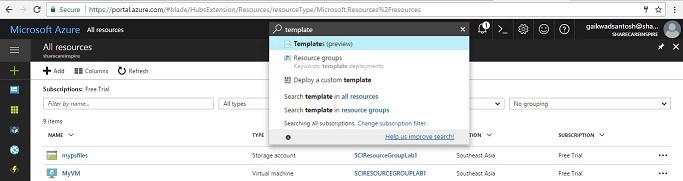
Additionally, there has been some market traction within the Microsoft Azure ecosystem in the adoption of Puppet and Chef. Microsoft even recognizes this with their [Azure certifications](https://www.microsoft.com/en-in/learning/mcsd-azure-architect-certification.aspx) by having questions on the exams expecting knowledge of what Puppet and Chef are, and what they are used for.

## **Azure Resource Manager Template (ARM)**

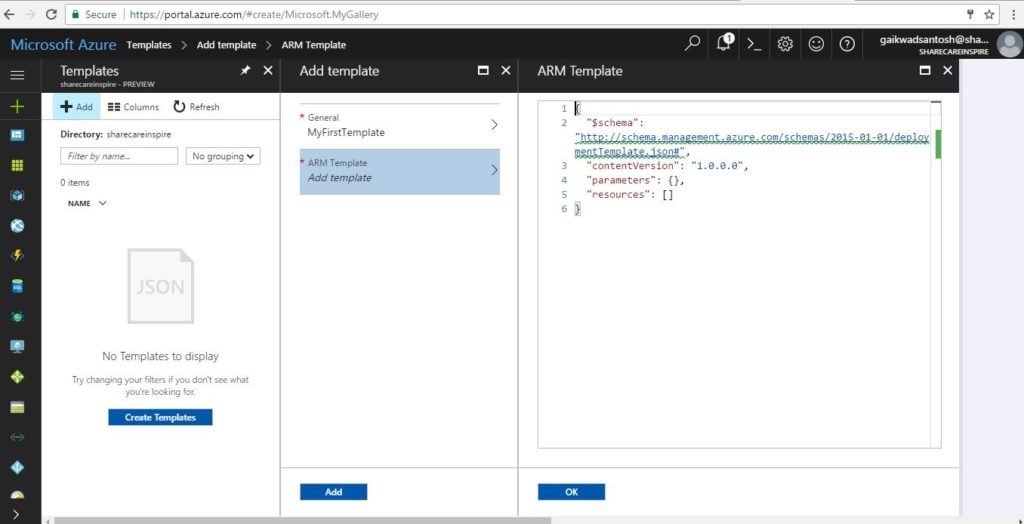
Azure Resource Templates are JSON files that capture infrastructure as code. You can capture all the Azure resources your application needs in a single JSON document that you can consistently deploy to different environments. All resources defined in an Azure Resource Template are provisioned within a Resource Group.

## **Provision resources using ARM template in ARM portal**

1. Search “Templates” on ARM portal  
Templates is a preview service currently (May 17) available on ARM portal. select “Templates” service and it will provide you the list of existing templates available in current subscription and gives you option to create/add new.

Search Templates

2. Add New Template  
Click on add button to add new template and use following code to provision web and database.

ARM JSON Template

{

   "$schema":"http://schema.management.azure.com/schemas/2014-04-01-preview/deploymentTemplate.json#",

   "contentVersion":"1.0.0.0",

   "parameters":{

      "siteName":{

         "type":"string",

         "metadata":{

            "description":"Name of azure web app"

         }

      },

      "administratorLogin":{

         "type":"string",

         "minLength":1,

         "metadata":{

            "description":"Database administrator login name"

         }

      },

      "administratorLoginPassword":{

         "type":"securestring",

         "minLength":8,

         "maxLength":128,

         "metadata":{

            "description":"Database administrator password"

         }

      },

      "databaseDTU":{

         "type":"int",

         "allowedValues":[

            50,

            100

         ],

         "defaultValue":50,

      "metadata":{

            "description":"Azure database for MySQL pricing tier"

         }

      },

      "databaseSkuName":{

         "type":"string",

         "allowedValues":[

            "MYSQLB100",

            "MYSQLB50"

         ],

         "defaultValue":"MYSQLB100",

         "metadata":{

            "description":"Azure database for MySQL sku name: MySQL Basic 50 DTU (MYSQLB50)or 100 DTU (MYSQLB100)"

         }

      },

      "databaseSkuSizeMB":{

         "type":"int",

         "allowedValues":[

            102400,

            51200

         ],

         "defaultValue":51200,

         "metadata":{

            "description":"Azure database for MySQL Sku Size "

         }

      },

      "databaseSkuTier":{

         "type":"string",

         "allowedValues":[

            "Basic"

         ],

         "defaultValue":"Basic",

         "metadata":{

            "description":"Azure database for MySQL pricing tier"

         }

      },

      "mysqlVersion":{

         "type":"string",

         "allowedValues":[

            "5.6",

            "5.7"

         ],

         "defaultValue":"5.6",

         "metadata":{

            "description":"MySQL version"

         }

      }

   },

   "variables":{

      "databaseName":"[concat(parameters('siteName'), 'database')]",

      "serverName":"[concat(parameters('siteName'), 'pgserver')]",

      "hostingPlanName":"[concat(parameters('siteName'), 'serviceplan')]"

   },

   "resources":[

      {

         "apiVersion":"2016-09-01",

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

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

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

         "properties":{

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

            "workerSizeId":"1",

            "reserved":true,

            "numberOfWorkers":0,

            "hostingEnvironment":""

         },

         "sku":{

            "Tier":"Standard",

            "Name":"S1"

         }

      },

      {

         "apiVersion":"2015-02-01",

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

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

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

         "dependsOn":[

            "[concat('Microsoft.Web/serverfarms/', variables('hostingPlanName'))]"

         ],

         "properties":{

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

            "serverFarmId":"[variables('hostingPlanName')]",

            "hostingEnvironment":""

         },

         "resources":[

            {

               "apiVersion":"2015-04-01",

               "name":"connectionstrings",

               "type":"config",

               "dependsOn":[

                  "[concat('Microsoft.Web/sites/', parameters('siteName'))]"

               ],

               "properties":{

                  "defaultConnection":{

                     "value":"[concat('Database=', variables('databaseName'), ';Data Source=', variables('serverName'), '.mysql.database.azure.com;User Id=', parameters('administratorLogin'),'@', variables('serverName'),';Password=', parameters('administratorLoginPassword'))]",

                     "type":"MySQL"

                  }

               }

            }

         ]

      },

      {

         "apiVersion":"2016-02-01-privatepreview",

         "kind":"",

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

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

         "properties":{

            "version":"[parameters('mysqlVersion')]",

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

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

            "storageMB":"[parameters('databaseSkuSizeMB')]"

         },

         "sku":{

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

            "tier":"[parameters('databaseSkuTier')]",

            "capacity":"[parameters('databaseDTU')]",

            "size":"[parameters('databaseSkuSizeMB')]",

            "family":"SkuFamily"

         },

         "type":"Microsoft.DBforMySQL/servers",

         "resources":[

            {

               "type":"firewallrules",

               "apiVersion":"2016-02-01-privatepreview",

               "dependsOn":[

                  "[concat('Microsoft.DBforMySQL/servers/', variables('serverName'))]"

               ],

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

               "name":"[concat(variables('serverName'),'firewall')]",

               "properties":{

                  "startIpAddress":"0.0.0.0",

                  "endIpAddress":"255.255.255.255"

               }

            },

            {

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

               "type":"databases",

               "apiVersion":"2016-02-01-privatepreview",

               "properties":{

                  "charset":"utf8",

                  "collation":"utf8\_general\_ci"

               },

               "dependsOn":[

                  "[concat('Microsoft.DBforMySQL/servers/', variables('serverName'))]"

               ]

            }

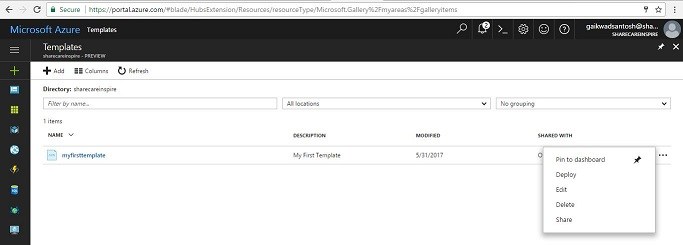
         ]

      }

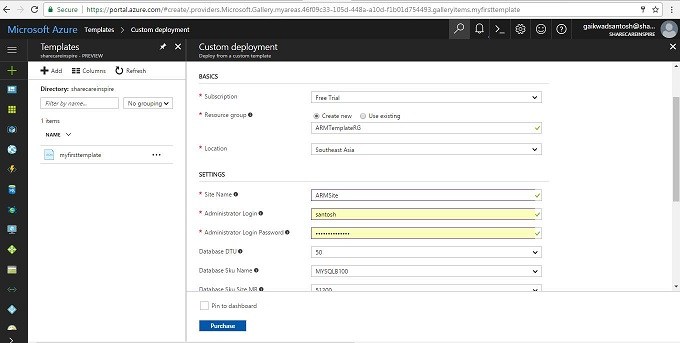
   ]

}

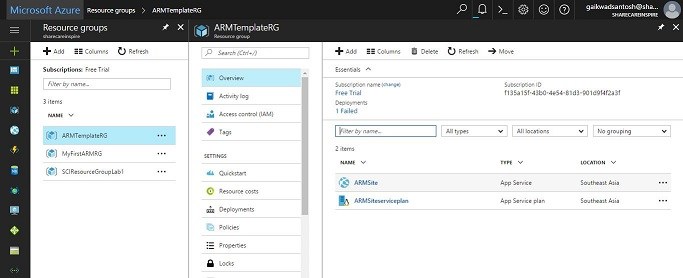
3. Deploy existing template  
Once template is created, it gets displayed in the list under templates service. click on the … button in-front of the template to deploy the same to provision defined resources in template.

Deploy ARM Template

4. Purchase resources / Custom deployment  
Deployment will prompt for the list of parameters specified in the JSON Template. provide all the required details and click on purchase button

Purchase Resources

5. Provisioned resources  
purchase resources button click start the ARM template deployment and resources provisioning process gets started, wait for some time to complete the provisioning process and open “Resource Groups” blade to see all the provisioned resources.

Resources provisioned

Deployment of ARM Templates using PowerShell is another popular practice followed by experts, will be covering same in another article.

## **What is ARM (Azure Resource Manager) Template**

Azure Resource Templates are JSON files that capture infrastructure as code. You can capture all the Azure resources your application needs in a single JSON document that you can consistently deploy to different environments. All resources defined in an Azure Resource Template are provisioned within a Resource Group, which is a logical group for managing related Azure resources.

To create and revise templates, you need a JSON editor. [Visual Studio Code](https://code.visualstudio.com/) is a lightweight, open-source, cross-platform code editor. It supports creating and editing Resource Manager Templates through an extension, however, if you have another JSON editor (like Visual Studio), you can use that editor.

## **ARM Template JSON format**

In its simplest structure, a template contains the following elements:

{

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

    "contentVersion": "",

    "parameters": {  },

    "variables": {  },

    "resources": [  ],

    "outputs": {  }

}

| Element name | Required | Description |
| --- | --- | --- |
| $schema | Yes | Location of the JSON schema file that describes the version of the template language. Use the URL shown in the preceding example. |
| contentVersion | Yes | Version of the template (such as 1.0.0.0). You can provide any value for this element. When deploying resources using the template, this value can be used to make sure that the right template is being used. |
| parameters | No | Values that are provided when deployment is executed to customize resource deployment. |
| variables | No | Values that are used as JSON fragments in the template to simplify template language expressions. |
| resources | Yes | Resource types those are deployed or updated in a resource group. |
| outputs | No | Values that are returned after deployment. |

Guidelines for creating ARM Template  
Limit the size of your template to 1 MB, and each parameter file to 64 KB. The 1-MB limit applies to the final state of the template after it has been expanded with iterative resource definitions, and values for variables and parameters.

Full Syntax of ARM Template  
Each element contains properties you can set. The following example contains the full syntax for a template:

{

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

    "contentVersion": "",

    "parameters": {

        "<parameter-name>" : {

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

            "defaultValue": "<default-value-of-parameter>",

            "allowedValues": [ "<array-of-allowed-values>" ],

            "minValue": <minimum-value-for-int>,

            "maxValue": <maximum-value-for-int>,

            "minLength": <minimum-length-for-string-or-array>,

            "maxLength": <maximum-length-for-string-or-array-parameters>,

            "metadata": {

                "description": "<description-of-the parameter>"

            }

        }

    },

    "variables": {

        "<variable-name>": "<variable-value>",

        "<variable-name>": {

            <variable-complex-type-value>

        }

    },

    "resources": [

      {

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

          "apiVersion": "<api-version-of-resource>",

          "type": "<resource-provider-namespace/resource-type-name>",

          "name": "<name-of-the-resource>",

          "location": "<location-of-resource>",

          "tags": {

              "<tag-name1>": "<tag-value1>",

              "<tag-name2>": "<tag-value2>"

          },

          "comments": "<your-reference-notes>",

          "copy": {

              "name": "<name-of-copy-loop>",

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

              "mode": "<serial-or-parallel>",

              "batchSize": "<number-to-deploy-serially>"

          },

          "dependsOn": [

              "<array-of-related-resource-names>"

          ],

          "properties": {

              "<settings-for-the-resource>",

              "copy": [

                  {

                      "name": ,

                      "count": ,

                      "input": {}

                  }

              ]

          },

          "resources": [

              "<array-of-child-resources>"

          ]

      }

    ],

    "outputs": {

        "<outputName>" : {

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

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

        }

    }

}

## **Create your first Azure Resource Manager template**

Microsoft along with community contributors is maintaining repository of quick start ARM templates in github at following location.

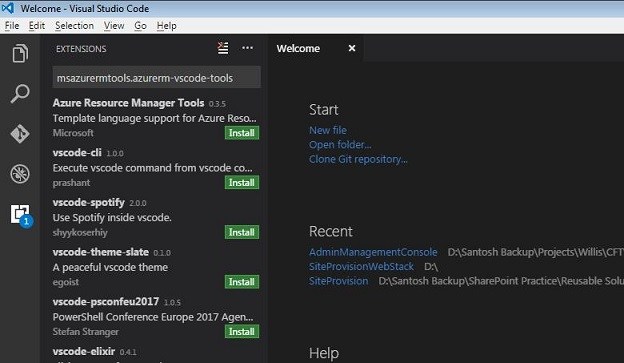
<https://github.com/Azure/azure-quickstart-templates>

Steps to create ARM Template using VS Code tool

1. Open visual studio code and press ctrl+P to search for extensions paste following command to get the extensions and install ARM extension.

ext install msazurermtools.azurerm-vscode-tools

To know more about visual studio code read [here](http://www.sharecareinspire.com/sharepoint-online-sp-2016-development-framework-spfx-using-open-source-web-stack/)

Visual Studio Code

2. Create a file  
3. Copy and paste following JSON syntax in file

{

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

  "contentVersion": "1.0.0.0",

  "parameters": {  },

  "variables": {  },

  "resources": [  ],

  "outputs": {  }

}

4. Save this file as azuredeploy.json.  
5. Let’s create this template to add storage account  
Storage account is a resource, hence we need to add the code to resources section.

{

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

  "contentVersion": "1.0.0.0",

  "parameters": {  },

  "variables": {  },

  "resources": [

    {

      "name": "string",

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

      "apiVersion": "2016-05-01",

      "sku": {

        "name": "string"

      },

      "kind": "Storage",

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

      "tags": {},

      "properties": {

      }

    }

  ],

  "outputs": {  }

}

6. In the above template I have most of the properties of storage resource in template, but let’s get name and sku values from parameters, so that users will have option to provide the name of storage account. But storage name should be unique in azure, hence there would be chances of failing script. Hence to avoid script failure create unique storage name, lets collect storage name prefix from users and attach unique resource groupID to create unique name.

7. Create parameter storageNamePrefix

"parameters": {

  "storageNamePrefix": {

    "type": "string",

    "maxLength": 11,

    "defaultValue": "storage",

    "metadata": {

      "description": "The value to use for starting the storage account name."

    }

  }

},

8. Construct variable storageName using parameter storageNamePrefix and resourceGroupID

"variables": {

  "storageName": "[concat(parameters('storageNamePrefix'), uniqueString(resourceGroup().id))]"

},

9. Now in the resources section update value of name property with variable as follows.

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

10. Now let’s update sku plan for azure storage account. Let’s plan to get the value from user, hence need to add one more parameter (named storageSKU) in parameters section. As sku values are very much internal to Azure and those have to spelled accurately, hence lets add the list of allowed values and set the default value so that user get help while executing template.

"parameters": {

  "storageNamePrefix": {

    "type": "string",

    "maxLength": 11,

    "defaultValue": "storage",

    "metadata": {

      "description": "The value to use for starting the storage account name."

    }

  },

  "storageSKU": {

    "type": "string",

    "allowedValues": [

      "Standard\_LRS",

      "Standard\_ZRS",

      "Standard\_GRS",

      "Standard\_RAGRS",

      "Premium\_LRS"

    ],

    "defaultValue": "Standard\_LRS",

    "metadata": {

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

    }

  }

},

11. Now get back to the resources section and update the sku value with parameter ‘storageSKU’

"sku": {

  "name": "[parameters('storageSKU')]"

},

12. Save file.

**Final ARM Template**After completing all the steps, final template looks as follows.

{

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

  "contentVersion": "1.0.0.0",

  "parameters": {

    "storageNamePrefix": {

      "type": "string",

      "maxLength": 11,

      "defaultValue": "storage",

      "metadata": {

        "description": "The value to use for starting the storage account name."

      }

    },

    "storageSKU": {

      "type": "string",

      "allowedValues": [

        "Standard\_LRS",

        "Standard\_ZRS",

        "Standard\_GRS",

        "Standard\_RAGRS",

        "Premium\_LRS"

      ],

      "defaultValue": "Standard\_LRS",

      "metadata": {

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

      }

    }

  },

  "variables": {

    "storageName": "[concat(parameters('storageNamePrefix'), uniqueString(resourceGroup().id))]"

  },

  "resources": [

    {

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

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

      "apiVersion": "2016-05-01",

      "sku": {

        "name": "[parameters('storageSKU')]"

      },

      "kind": "Storage",

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

      "tags": {},

      "properties": {

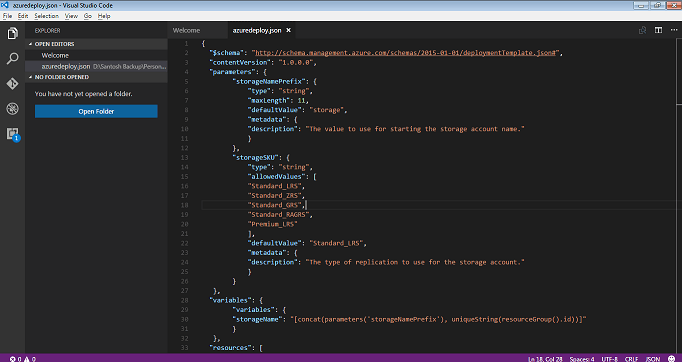
      }

    }

  ],

  "outputs": {  }

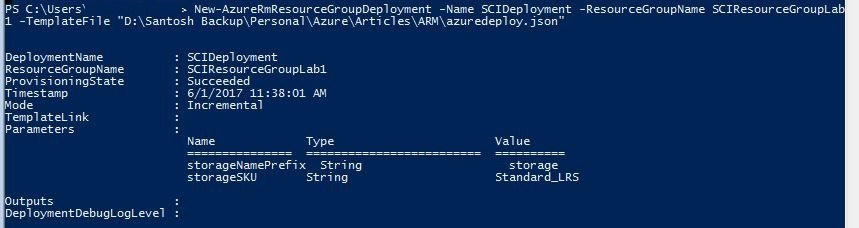
}

Final ARM Template

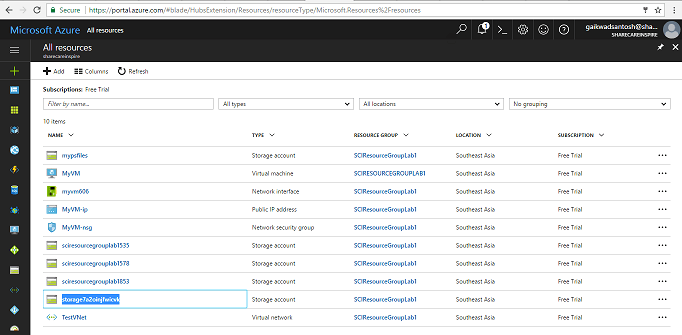
**Deploy Resources to Azure using ARM Template using PowerShell**

1. Execute ARM template with default values

New-AzureRmResourceGroupDeployment -Name ExampleDeployment -ResourceGroupName ExampleGroup -TemplateFile c:\MyTemplates\azuredeploy.json

PowerShell Azure Resource Group Deployment

You can see storage account got created in Azure ARM Portal

Storage Account created in ARM Portal

2. Execute ARM template with specific values as parameters

New-AzureRmResourceGroupDeployment -Name ExampleDeployment2 -ResourceGroupName ExampleGroup -TemplateFile c:\MyTemplates\azuredeploy.json -storageNamePrefix "sci" -storageSKU "Standard\_GRS"

3. Execute ARM Template with Parameters file  
Create Parameters file and save it as azuredeploy.parameters.json

{

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

  "contentVersion": "1.0.0.0",

  "parameters": {

     "storageNamePrefix": {

         "value": "scistorage"

     },

     "storageSKU": {

         "value": "Standard\_GRS"

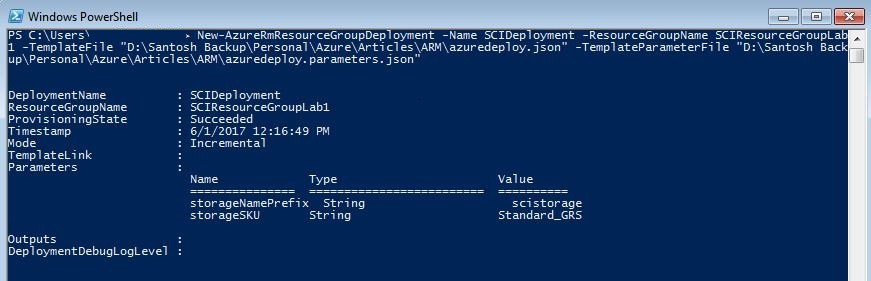
     }

  }

}

Execute following command on PowerShell command prompt.

New-AzureRmResourceGroupDeployment -Name ExampleDeployment2 -ResourceGroupName ExampleGroup -TemplateFile c:\MyTemplates\azuredeploy.json –TemplateParameterFile c:\MyTemplates\azuredeploy.parameters.json

PowerShell Azure Resource Group Deployment with parameters file

# How to make changes to pages and files on Web Apps form ARM Portal

## **How to make changes to files in web app from portal**

While doing development I came across a scenario, where I wanted to change the content of static page of a web app which is hosted in Azure. I did not want to use a visual studio and wanted a quick and easy way to make changes directly on the Azure Portal itself.

I believe this would be a common requirement among all developers, while debugging your Azure Web App deployments it is sometimes useful to view the files that are deployed to the service. There are many scenarios. E.g. update database connection strings or possibly determining which files actually got deployed and where they sit in the file system, or maybe you need to check if all your dependencies have been installed.

So how to view files if you are using a [Platform as a Service (PaaS)](https://en.wikipedia.org/wiki/Platform_as_a_service) solution such as Web Apps? Web app doesn’t allow you to remote into VM in the traditional sense and instead abstracts all that complexity for you.

Azure provides best way to manage these issues, using a tool called Kudu. It provides console gives you both command line and file browser access to your sites.

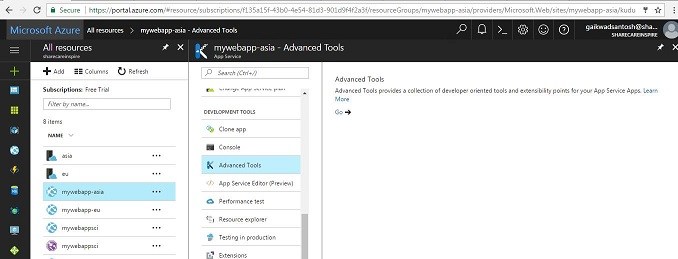
I have created a web app [https://mywebapp-asia.azurewebsites.net](https://mywebapp-asia.azurewebsites.net/), and following is the default page of the web app, I want to have my own default page. To make the changes lets follow steps as given below.

### **Step 1: Create a Web App**

Create new azure web app either from ARM portal or PowerShell, following is the default page of a newly created web app. now i want to create another web page and set that as a default page.

Default Web App page

Navigate to properties of the web app and under development tools section select advanced tools option.

Advanced Tools for Web App

As you click on go button, which appears on the rights of the screen,  page will be navigated to Kudu management console.

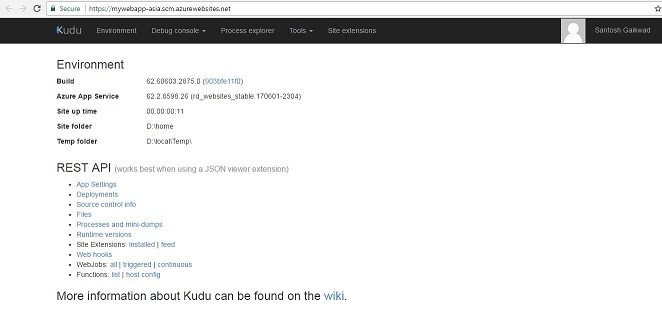
## **What is KUDU and how I can access It?**

The idea behind a PaaS offering is that you do not have to worry about the operating system or infrastructure or just stuff in general that does not give you actual business benefit.

However, sometimes you do need to access the actual environment you are running code on, usually those are the moments you are either developing a new solution or you need to troubleshoot something that went wrong. KUDU service is developed as a way to access a Microsoft Azure Web Apps environment, which includes Azure Functions.

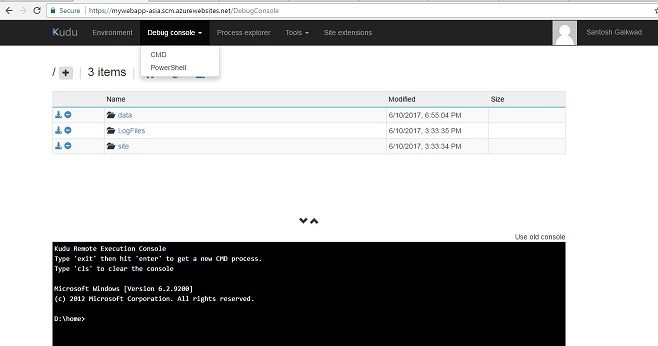
With KUDU you can get access to log files, environment information/variables and also the actual file system on the runner instance that was assigned to your Azure web app.

Accessing your KUDU console is easily done via the following URL: https://mywebapp-asia.scm.azurewebsites.net where  mywebapp-asiais the name of your web app, following screen snap shows Kudu console.

Kudu Console

### **Step: 2 Select CMD option from Debug console menu.**

As you select CMD from debug console menu, you will be presented with file explorer and a windows command console right within web browser. File explorer and command console are always in sync i.e. if you execute a command on console same will be immediately reflated in file explorer too.

Kudu Command Console

### **Step 3: Navigate to /sites/wwwroot**

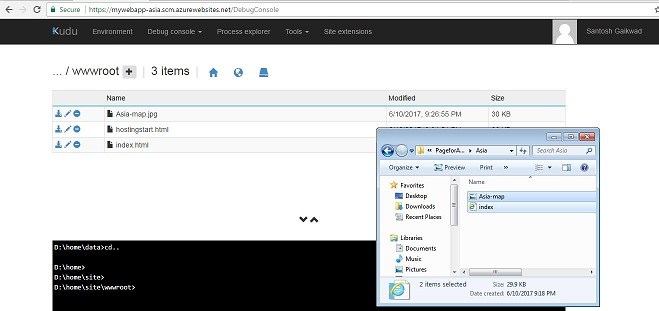
If you notice, azure separates data drive disk from operating disk drive, hence your web app contents are deployed to D drive by default and stored in D:\home\sites

All the web pages and files are stored in wwwroot folder of the web app, be default azure creates hostingstart.html file in azure web app and sets it as a default page.

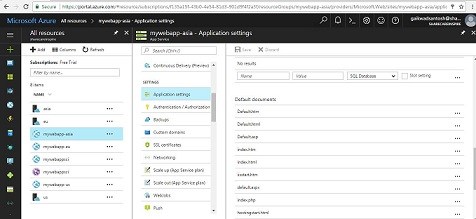
### **Step 4: Upload files**

Now it’s time to upload desired default page to the web app folder. Just drag files from windows explorer and drop it on the page (file explores), files will be automatically uploaded to web app.

For this article I have created a index.html file and used asia-map.jpg file in the page to display the region in which web app is hosted.

Upload files to Kudu Console

If you want to set a default page to the web app then make sure you follow the naming convention as shown in following screen snap. Default pages/documents are configured in the applications settings option in settings blade of web app.

Web App Default Page sequence

### **Step 5: Verify if default page is changed**

Now it’s time to test if new default page is considered for processing. Navigate to the web app URL [https://mywebapp-asia.azurewebsites.net](https://mywebapp-asia.azurewebsites.net/)

Now default page is changed as per the uploaded files.

Default page of Web app changed