**Challenge #1**  
  
Step 1: Define the infrastructure components:

Presentation Tier: We'll create a virtual network, subnet, and a load balancer.

Application Tier: We'll create virtual machines (VMs) that will serve as the application servers.

Data Tier: We'll create a database server using Azure SQL Database.

Step 2: Write an ARM template:

Create a new file called azuredeploy.json and open it in a text editor.

Step 3: Define parameters and variables:

Add the following code block within the parameters and variables sections of the ARM template:

"parameters": {

"vmSize": {

"type": "string",

"defaultValue": "Standard\_DS2\_v2",

"metadata": {

"description": "The size of the application VMs."

}

}

},

"variables": {

"vnetName": "myVNet",

"subnetName": "mySubnet",

"lbName": "myLoadBalancer",

"dbServerName": "myDBServer",

"dbDatabaseName": "myDatabase",

"dbAdminUsername": {

"value": "admin"

},

"dbAdminPassword": {

"value": "password"

}

}  
  
Step 4: Define resources:

Add the following code block within the resources section of the ARM template:

"resources": [

{

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

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

"apiVersion": "2021-02-01",

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

"properties": {

"addressSpace": {

"addressPrefixes": [

"10.0.0.0/16"

]

},

"subnets": [

{

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

"properties": {

"addressPrefix": "10.0.0.0/24"

}

}

]

}

},

{

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

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

"apiVersion": "2021-02-01",

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

"dependsOn": [

"[resourceId('Microsoft.Network/virtualNetworks', variables('vnetName'))]"

],

"properties": {

"frontendIPConfigurations": [

{

"name": "LoadBalancerFrontEnd",

"properties": {}

}

],

"backendAddressPools": [

{

"name": "BackendPool",

"properties": {}

}

]

}

},

{

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

"name": "[concat('nic-', copyIndex())]",

"apiVersion": "2021-02-01",

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

"copy": {

"name": "nicLoop",

"count": 2

},

"dependsOn": [

"[resourceId('Microsoft.Network/loadBalancers', variables('lbName'))]"

],

"properties": {

"ipConfigurations": [

{

"name": "ipconfig",

"properties": {

"subnet": {

"id": "[resourceId('Microsoft.Network/virtualNetworks/subnets', variables('vnetName'), variables('subnetName'))]"

}

}

}

]

}

},

{

"type": "Microsoft.Compute/virtualMachines",

"name": "[concat('vm-', copyIndex())]",

"apiVersion": "2021-03-01",

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

"copy": {

"name": "vmLoop",

"count": 2

},

"dependsOn": [

"[resourceId('Microsoft.Network/networkInterfaces', concat('nic-', copyIndex()))]"

],

"properties": {

"hardwareProfile": {

"vmSize": "[parameters('vmSize')]"

},

"osProfile": {

"computerName": "[concat('vm-', copyIndex())]",

"adminUsername": "admin",

"adminPassword": "password"

},

"networkProfile": {

"networkInterfaces": [

{

"id": "[resourceId('Microsoft.Network/networkInterfaces', concat('nic-', copyIndex()))]"

}

]

},

"storageProfile": {

"imageReference": {

"publisher": "MicrosoftWindowsServer",

"offer": "WindowsServer",

"sku": "2019-Datacenter",

"version": "latest"

},

"osDisk": {

"name": "osdisk",

"createOption": "FromImage"

}

}

}

},

{

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

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

"apiVersion": "2021-02-01-preview",

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

"dependsOn": [

"[resourceId('Microsoft.Network/virtualNetworks', variables('vnetName'))]"

],

"properties": {

"administratorLogin": "[variables('dbAdminUsername')]",

"administratorLoginPassword": "[variables('dbAdminPassword')]",

"version": "12.0",

"publicNetworkAccess": "Disabled"

}

},

{

"type": "Microsoft.Sql/servers/databases",

"name": "[concat(variables('dbServerName'), '/', variables('dbDatabaseName'))]",

"apiVersion": "2021-02-01-preview",

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

"dependsOn": [

"[resourceId('Microsoft.Sql/servers', variables('dbServerName'))]"

],

"properties": {

"collation": "SQL\_Latin1\_General\_CP1\_CI\_AS",

"edition": "Basic",

"maxSizeBytes": "1073741824",

"requestedServiceObjectiveName": "Basic"

}

}

]  
  
Step 5: Deploy the ARM template: You can use the Azure CLI to deploy the ARM template. Run the following command:  
  
az deployment group create --resource-group <resource-group-name> --template-file azuredeploy.json  
  
Replace <resource-group-name> with the name of your resource group.

The ARM template will be deployed, creating the 3-tier environment with the specified resources.

Note: The provided example is a simplified representation of a 3-tier environment and may require further customization based on your specific requirements.

**Challenge #2**

import requests

import json

def get\_aws\_metadata():

response = requests.get('http://169.254.169.254/latest/dynamic/instance-identity/document')

return response.json()

def get\_azure\_metadata():

response = requests.get('http://169.254.169.254/metadata/instance?api-version=2021-02-01', headers={'Metadata': 'true'})

return response.json()

def get\_gcp\_metadata():

response = requests.get('http://metadata.google.internal/computeMetadata/v1/instance/?recursive=true', headers={'Metadata-Flavor': 'Google'})

return response.text

def retrieve\_instance\_metadata(provider):

if provider == 'aws':

metadata = get\_aws\_metadata()

elif provider == 'azure':

metadata = get\_azure\_metadata()

elif provider == 'gcp':

metadata = get\_gcp\_metadata()

else:

return None

return json.dumps(metadata, indent=4)

# Retrieve and print the full instance metadata

aws\_metadata = retrieve\_instance\_metadata('aws')

print("AWS Instance Metadata:")

print(aws\_metadata)

azure\_metadata = retrieve\_instance\_metadata('azure')

print("Azure Instance Metadata:")

**Challenge #3**

def get\_value\_from\_nested\_object(obj, key):

keys = key.split('/')

current\_obj = obj

try:

for k in keys:

current\_obj = current\_obj[k]

return current\_obj

except KeyError:

return None

# Example usage:

obj1 = {"a": {"b": {"c": "d"}}}

key1 = "a/b/c"

value1 = get\_value\_from\_nested\_object(obj1, key1)

print(value1) # Output: d

obj2 = {"x": {"y": {"z": "a"}}}

key2 = "x/y/z"

value2 = get\_value\_from\_nested\_object(obj2, key2)

print(value2) # Output: a