**Q1 - SCENARIO**

A car rental company called FastCarz has a .net Web Application and Web API which are recently migrated from on-premise system to Azure cloud using Azure Web App Service

and Web API Service.

The on-premises system had 3 environments Dev, QA and Prod.

The code repository was maintained in TFS and moved to Azure GIT now. The TFS has daily builds which triggers every night which build the solution and copy the build package to drop folder.

deployments were done to the respective environment manually. The customer is planning to setup Azure DevOps Pipeline service for below requirements:

1. *The build should trigger as soon as anyone in the dev team checks in code to master branch.*

*We can Implement ‘continuous integration trigger’ and include the branch filter as ‘master’ branch In Build pipeline.*

*Same can be done in YAML Definition as below but we need to disable the continuous integration trigger from trigger section as it is overriding YAML definition.*

*trigger:*

*branches:*

*include:*

*- master*

*2) There will be test projects which will create and maintained in the solution along the Web and API. The trigger should build all the 3 projects - Web, API and test.*

*The build should not be successful if any test fails.*

*To Build the .net project we can use* ***.net build task*** *where we can provide the “****Path to Projects****” which will build web and API.*

*We can use* ***‘.net test task’*** *which will execute all tests. we should un-select the option ‘****Continue on error****’ (continueOnError: false) from “control option” section, to fail our build if any test fails.*

*3) The deployment of code and artifacts should be automated to Dev environment.*

*In the release pipeline, we can enable the continuous deployment for each artifact and should include the specific branch name so that code pushed to the specific branch will trigger the CI and then trigger the CD*

*4) Upon successful deployment to the Dev environment, deployment should be easily promoted to QA and Prod through automated process.*

*In the release pipeline, each environment (Dev, QA, PROD) will be a stage. In QA and PROD stages we can configure “****Pre-Deployment Condition”*** *where we should choose ‘After Stage’ trigger and for QA after stage will be Dev and for PROD after stage will be QA.*

*5) The deployments to QA and Prod should be enabled with Approvals from approvers only.*

*In QA and PROD stages, we can* ***enable*** *the ‘****Pre-deployment approvals’*** *from stage’s ‘****Pre-deployment Conditions’*** *and we can provide approvers name or approvers group.*

Explain how each of the above the requirements will be met using Azure DevOps configuration.

Explain the steps with configuration details.

**Q2 - SCENARIO**

Macro Life, a healthcare company has recently setup the entire Network and Infrastructure on Azure.

The infrastructure has different components such as Virtual N/W, Subnets, NIC, IPs, NSG etc.

The IT team currently has developed PowerShell scripts to deploy each component where all the properties of each resource is set using PowerShell commands.

The business has realized that the PowerShell scripts are growing over period of time and difficult to handover when new admin onboards in the IT.

The IT team has now decided to move to Terraform based deployment of all resources to Azure.

All the passwords are stored in a Azure Service known as key Vault. The deployments needs to be automated using Azure DevOps using IaC(Infrastructure as Code).

*1) What are different artifacts you need to create - name of the artifacts and its purpose*

*In Azure devops git repo, terraform code will be store. We can build the terraform plan in CI pipeline and use the terraform plan with git files as an artifact of deploy release pipeline.*

*We may be wanting multiple iterations of the Terraform pipeline; these iterations may require an approval process or requirement to be approved prior to being released into a specific environment. Multiple people are updating Terraform code into the master branch; these approvals backup and now the current branch version is different to the original approval requirement. This is where build artifact will help.*

*Below is the terraform plan stage/ CI:*

*Git Checkout: - Normal process, checkout required branch*

*Terraform Init: - Initial Terraform using Task: - TerraformTaskV1@0*

*Terraform Plan: - Terraform Plan using Task: - TerraformTaskV1@0*

*Archive Files: - Archive the directory that has both the git content and the Terraform Plan*

*2) List the tools you will to create and store the Terraform templates.*

*- terraform HCL*

*- terraform providers*

*- one Azure SPN which will use in terraform azure provider*

*- azure devops git repo/github to store the terraform template*

*- one storage account with storage container to store terraform state*

*3) Explain the process and steps to create automated deployment pipeline.*

*1) Terraform Build & Release Tasks extension: For Azure devops to apply terraform*

*2) Add the build artifact of terraform plan and terraform templates in the release pipeline*

*3) PowerShell task: To create a storage account and create****Azure storage account****and****storage container****to store Terraform state. Get the access key of the storage account.*

*4) update the access key into terraform template.*

*5) Install terraform task*

*6) terraform init task*

*7) terraform plan task*

*8) terraform apply task*

*4) Create a sample Terraform template you will use to deploy Below services:*

*Vnet*

*2 Subnet*

*NSG to open port 80 and 443*

*1 Window VM in each subnet*

*1 Storage account*

provider "azurerm" {

  client\_id       = "cliend-id"

  client\_secret   = "client-secret"

  subscription\_id = "subscription id"

  tenant\_id       = "tenant id"

  version         = "=2.30.0"

  skip\_provider\_registration = true

  features {}

}

## Variables

variable "location" {

  type    = string

  default = "westeurope"

}

variable "prefix" {

  type    = string

  default = "demo"

}

variable "ip-config" {

  default = [

    {

      name       = "instance1-ip1"

      allocation = "Dynamic"

      primary    = true

    },

    {

      name       = "instance1-ip2"

      allocation = "Dynamic"

      primary    = false

    },

  ]

}

## RG

resource "azurerm\_resource\_group" "demo" {

  name     = "resource-group-demo"

  location = var.location

  tags = {

      env = "resource-group-demo"

  }

}

## VNET

resource "azurerm\_virtual\_network" "demo" {

  name                = "${var.prefix}-network"

  location            = var.location

  resource\_group\_name = azurerm\_resource\_group.demo.name

  address\_space       = ["10.0.0.0/16"]

}

#subnets count=2

resource "azurerm\_subnet" "demo-internal-1" {

  name                 = "${var.prefix}-internal-1"

  resource\_group\_name  = azurerm\_resource\_group.demo.name

  virtual\_network\_name = azurerm\_virtual\_network.demo.name

  address\_prefix       = "10.0.0.0/24"

}

resource "azurerm\_subnet" "demo-internal-2" {

  name                 = "${var.prefix}-internal-2"

  resource\_group\_name  = azurerm\_resource\_group.demo.name

  virtual\_network\_name = azurerm\_virtual\_network.demo.name

  address\_prefix       = "10.0.1.0/24"

}

## NSG for port 80 and port 443

resource "azurerm\_network\_security\_group" "port-80" {

    name                = "port-80"

    location            = var.location

    resource\_group\_name = azurerm\_resource\_group.demo.name

    security\_rule {

        name                       = "port-80"

        priority                   = 1001

        direction                  = "Inbound"

        access                     = "Allow"

        protocol                   = "http"

        source\_port\_range          = "\*"

        destination\_port\_range     = "80"

        destination\_address\_prefix = "\*"

    }

}

resource "azurerm\_network\_security\_group" "port-443" {

    name                = "port-443"

    location            = var.location

    resource\_group\_name = azurerm\_resource\_group.demo.name

    security\_rule {

        name                       = "port-443"

        priority                   = 1002

        direction                  = "Inbound"

        access                     = "Allow"

        protocol                   = "https"

        source\_port\_range          = "\*"

        destination\_port\_range     = "443"

        destination\_address\_prefix = "\*"

    }

}

## Storage account

resource "azurerm\_storage\_account" "demo-storage" {

  name                     = "demo-storage"

  location                 = azurerm\_resource\_group.demo.location

  resource\_group\_name      = azurerm\_resource\_group.demo.name

  account\_tier             = "Standard"

  account\_replication\_type = "LRS"

}

## VM1

resource "azurerm\_virtual\_machine" "demo-instance" {

  name                  = "${var.prefix}-vm"

  location              = var.location

  resource\_group\_name   = azurerm\_resource\_group.demo.name

  network\_interface\_ids = [azurerm\_network\_interface.demo-instance.id]

  vm\_size               = "Standard\_F2"

  delete\_os\_disk\_on\_termination = true

  delete\_data\_disks\_on\_termination = true

  source\_image\_reference {

    publisher = "MicrosoftWindowsServer"

    offer     = "WindowsServer"

    sku       = "2016-Datacenter"

    version   = "latest"

  }

  storage\_os\_disk {

    name              = "myosdisk1"

    caching           = "ReadWrite"

    create\_option     = "FromImage"

    managed\_disk\_type = "Standard\_LRS"

  }

  os\_profile {

    computer\_name  = "demo-instance"

    admin\_username = "demo"

    admin\_password = "..."

  }

}

resource "azurerm\_network\_interface" "demo-instance" {

  name                      = "${var.prefix}-instance1"

  location                  = var.location

  resource\_group\_name       = azurerm\_resource\_group.demo.name

  network\_security\_group\_id = azurerm\_network\_security\_group.port-80.id

  network\_security\_group\_id1 = azurerm\_network\_security\_group.port-443.id

  dynamic ip\_configuration {

    for\_each = var.ip-config

    content {

      name                          = lookup(ip\_configuration.value, "name")

      subnet\_id                     = azurerm\_subnet.demo-internal-1.id

      private\_ip\_address\_allocation = lookup(ip\_configuration.value, "allocation")

      public\_ip\_address\_id          = azurerm\_public\_ip.demo-instance[ip\_configuration.key].id

      primary                       = lookup(ip\_configuration.value, "primary")

    }

  }

}

resource "azurerm\_public\_ip" "demo-instance" {

    count                        = length(var.ip-config)

    name                         = "instance1-public-ip-${count.index}"

    location                     = var.location

    resource\_group\_name          = azurerm\_resource\_group.demo.name

    allocation\_method            = "Static"

}

##VM2

resource "azurerm\_virtual\_machine" "demo-instance1" {

  name                  = "${var.prefix}-vm"

  location              = var.location

  resource\_group\_name   = azurerm\_resource\_group.demo.name

  network\_interface\_ids = [azurerm\_network\_interface.demo-instance1.id]

  vm\_size               = "Standard\_F2"

  delete\_os\_disk\_on\_termination = true

  delete\_data\_disks\_on\_termination = true

  source\_image\_reference {

    publisher = "MicrosoftWindowsServer"

    offer     = "WindowsServer"

    sku       = "2016-Datacenter"

    version   = "latest"

  }

  storage\_os\_disk {

    name              = "myosdisk1"

    caching           = "ReadWrite"

    create\_option     = "FromImage"

    managed\_disk\_type = "Standard\_LRS"

  }

  os\_profile {

    computer\_name  = "demo-instance1"

    admin\_username = "demo"

    admin\_password = "..."

  }

}

resource "azurerm\_network\_interface" "demo-instance1" {

  name                      = "${var.prefix}-instance2"

  location                  = var.location

  resource\_group\_name       = azurerm\_resource\_group.demo.name

  network\_security\_group\_id = azurerm\_network\_security\_group.port-80.id

  network\_security\_group\_id1 = azurerm\_network\_security\_group.port-443.id

  dynamic ip\_configuration {

    for\_each = var.ip-config

    content {

      name                          = lookup(ip\_configuration.value, "name")

      subnet\_id                     = azurerm\_subnet.demo-internal-2.id

      private\_ip\_address\_allocation = lookup(ip\_configuration.value, "allocation")

      public\_ip\_address\_id          = azurerm\_public\_ip.demo-instance1[ip\_configuration.key].id

      primary                       = lookup(ip\_configuration.value, "primary")

    }

  }

}

resource "azurerm\_public\_ip" "demo-instance1" {

    count                        = length(var.ip-config)

    name                         = "instance1-public-ip-${count.index}"

    location                     = var.location

    resource\_group\_name          = azurerm\_resource\_group.demo.name

    allocation\_method            = "Static"

}

*5) Explain how you will access the password stored in Key Vault and use it as Admin Password in the VM Terraform template.*

* *In azure Devops pipeline, we can add a powershell task where we get the Key Vault secret as below*

*$VaultName = "vault"*

*$SecretName = "vm-password"*

*$MySecretValue = (Get-AzKeyVaultSecret -VaultName $VaultName -Name $SecretName).SecretValueText*

*Write-Host "##vso[task.setvariable variable=vmpassword]$MySecretValue"*

* *Then, we can use a task ‘replace tokens in terraform file’ which replace the secret value as VM Password*