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 service for below requirements:

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

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.

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

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

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

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

Explain the steps with configuration details.

Answer :

1. First step is to have the source code in a version control system i.e Azure Git in current scenario. We need to create an azure repo of type Git for the organization project. Optionally clone it to our local machine incase we are working with the code itself. Now that we have our version control we can create an Azure CI pipeline to trigger automatically a build whenever a code is committed
2. For this purpose, we can run 3 tests in parallel i.e parallel test runner by specifying the configuration in YAML and specify below to cancel build

failTaskOnFailedTests: true

another option is to untick the option “continue on error” in the integration test

1. We can use azure CD release here and under artifacts select drop . At the right of the **Drop** icon, select **Continuous deployment trigger**  which executes a deployment every time a new build artifact is available.Easiest way here to do is to choose an app service template in the release pipeline

CD release pipeline picks up the artifacts published by your CI build and then deploys the azure web app

4.In the azure pipeline we can create a trigger to automatically promote deployment to QA and Prod if it passes quality gate analysis sonarqube testing parameters during DEV build

1. Under azure pipelines environment for the QA and Prod select approvers and checks, Instructions to approvers, enter Approve this change when it's ready for staging. You can configure Azure DevOps to send you an email notification also when the build requires approval

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 ARM 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

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

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

4) Create a sample ARM template you will use to deploy a Windows VM of any size

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

Answers :

1

* Microsoft.Network/virtualNetworks
* Microsoft.Network/virtualNetworks/subnets
* Microsoft.Network/networkSecurityGroups
* Microsoft.Network/networkInterfaces

2

* Visual studio code or intellij to create templates. We can use azure gitrepos from our project to store ARM templates

3.We need to use azure pipeline to create an automated pipeline for this activity. Below is top level overview

* Prepare the azure git repo and push code to the remote repository
* Create an Azure DevOps project and create service connection as Azure resource manager
* Create an Azure pipeline and connect to the git repo. From the **Configure** tab, select **Starter pipeline** and then add a task ARM template deployment to fill in required details. Select save and run
* Verify the pipeline deployment by going to the resource group and ensuring intended resources like vnet,subnet and NSG rules are created

4.

{

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

    "contentVersion": "1.0.0.0",

    "parameters": {

        "virtualMachines\_ELK\_name": {

            "defaultValue": "ELK",

            "type": "String"

        },

        "disks\_ELK\_OsDisk\_1\_a8691463a45742afaf967e77fc70af43\_externalid": {

            "defaultValue": "/subscriptions/526c5b9c-41d3-4237-a7ed-09b236d825cd/resourceGroups/LEARNING/providers/Microsoft.Compute/disks/ELK\_OsDisk\_1\_a8691463a45742afaf967e77fc70af43",

            "type": "String"

        },

        "networkInterfaces\_elk187\_externalid": {

            "defaultValue": "/subscriptions/526c5b9c-41d3-4237-a7ed-09b236d825cd/resourceGroups/Learning/providers/Microsoft.Network/networkInterfaces/elk187",

            "type": "String"

        }

    },

    "variables": {},

    "resources": [

        {

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

            "apiVersion": "2019-07-01",

            "name": "[parameters('virtualMachines\_ELK\_name')]",

            "location": "centralindia",

            "tags": {

                "elk": ""

            },

            "properties": {

                "hardwareProfile": {

                    "vmSize": "Standard\_B2s"

                },

                "storageProfile": {

                    "imageReference": {

                        "publisher": "MicrosoftWindowsServer",

                        "offer": "WindowsServer",

                        "sku": "2012-R2-Datacenter",

                        "version": "latest"

                    },

                    "osDisk": {

                        "osType": "Windows",

                        "name": "[concat(parameters('virtualMachines\_ELK\_name'), '\_OsDisk\_1\_a8691463a45742afaf967e77fc70af43')]",

                        "createOption": "FromImage",

                        "caching": "ReadWrite",

                        "managedDisk": {

                            "id": "[parameters('disks\_ELK\_OsDisk\_1\_a8691463a45742afaf967e77fc70af43\_externalid')]"

                        }

                    },

                    "dataDisks": []

                },

                "osProfile": {

                    "computerName": "[parameters('virtualMachines\_ELK\_name')]",

                    "adminUsername": "ru561569",

                    "windowsConfiguration": {

                        "provisionVMAgent": true,

                        "enableAutomaticUpdates": true

                    },

                    "secrets": [],

                    "allowExtensionOperations": true,

                    "requireGuestProvisionSignal": true

                },

                "networkProfile": {

                    "networkInterfaces": [

                        {

                            "id": "[parameters('networkInterfaces\_elk187\_externalid')]"

                        }

                    ]

                },

                "diagnosticsProfile": {

                    "bootDiagnostics": {

                        "enabled": true

                    }

                },

                "licenseType": "Windows\_Server"

            }

        }

    ]

}

1. This can be achieved by using parameters such as below in the azure arm template
2. "adminPassword": {
3. "reference": {
4. "keyVault": {
5. "id": "/subscriptions/<SubscriptionID>/resourceGroups/mykeyvaultdeploymentrg/providers/Microsoft.KeyVault/vaults/<KeyVaultName>"
6. },
7. "secretName": "vmAdminPassword"
8. }
9. },

Below prerequisite steps need to be completed before previous

* Create a key vault with the enabledForTemplateDeployment property enabled. This property must be *true* before the template deployment process can access the secrets that are defined in the key vault.
* Add a secret to the key vault. The secret stores the VM administrator password.

Q3 - SCENARIO

A Toy Retail company ToyTrex has it retail application deployed as 3-tier application - Web App(UI), Web API(middle layer) and Database as Azure SQL.

The user load started increasing multiple fold every month and complex programs getting implemented, the application started performing poorly.

As a result, company decided to re-architect the middle layer as microservices using Azure Kubernetes Services.

The new architecture has below design decisions.

1) The middle layer should be implemented as Microservices using Azure AKS

2) The middle layer API should be deployed as containerized application images

3) The container images will use Azure Container Repository (ACR) as the private image repository

4) The CI/CD pipelines for microservices should be implemented using Azure DevOps services.

5) The Azure DevOps should be able to access ACR and download the container images for microservices deployment

6) The image should be deployed as templates such as <image\_name>:<build\_id>

Explain the DevOps configuration and steps in detail for above requirements

Answers:

1. Create docker image for api service with health checks   
2. Create a deployment with using the image, in service select azure loadbalancer as service.  
3. Use service name to connect database  
--Ci cd process will have below steps   
 code changes -> pushed to master > builds docker image with commit\_id   
> Image tag in deployment will be variable will be updated dynamically > final step will be kubectl apply