**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. *To automate the build trigger we need to enable “****Continuous Integration****” in build pipeline. To configure the same we need to follow the below steps: - 1) Configure a Build pipeline with required tasks with “****Azure Git repository****” for “****FastCarz Project****” with “Default branch for Manual and Scheduled build” as “****Master****” branch [we can choose any other branch as well] 🡪 Go to “****Triggers****” Tab in the pipeline options 🡪 “****Check****” the option “****Enable Continuous Integration****”.*

*Whenever any Dev Team member create a pull request to merge the code into “****Master****” branch “****ContinosIntegration****” will trigger a build. For the best practices direct Commit to the Master branch should be blocked by Branch Policy.*

1. *To Configure “****Break the Build if any Test Fails****” we need to configure the “****Visual Studio Test****” task with “****Control Options****” as “****Run This Task = Only When all Previous tasks have succeeded****”. This option used to be set as* ***default*** *if we not set any custom condition in YAML build.*

*This configuration will check if any test failure, it will break the build and rest of the task in the build pipeline will be “****Skipped****”. A snapshot from my own AzDo POC project.*

|  |
| --- |
|  |

1. *To Publish the Artifact in Dev Environment through Automated process we need to “****Enable****” the “****Continuous Deployment Trigger****”. TO achieve the same, we need to follow the below steps: -*

*Create a Release pipeline with the same build artifact from the build pipeline (mentioned above). The release pipeline needs to setup with all the 3 Environments as mentioned above and the “****DEV****” environment should be configuring the trigger as “****After Release****”. We need enable the “****Continuous Deployment Trigger****” on “****Triggers****” Tab in added “****Build Artifact****” Options in “****Release pipeline****”. Below snapshot from my own AzDo project.*

|  |  |
| --- | --- |
|  |  |

1. *To automate the deployment towards the “****QA****” and “****Prod****” after successful release to the “****DEV****” environment we need to configure the “****QA****” and “****Prod****” Stages with “****Triggers****” options. To achieve the same, we need follow the below steps.*

*Go to the Release pipeline 🡪 Edit Pipeline 🡪 Add a “****QA****” stage where the triggers option will be configured as “****After Stage (need to choose: - Dev from drop-down box)*** *and Add a “****Prod****” Stage where the triggers option will be configured as “****After Stage (need to choose: - QA from drop-down box)****. We need to make sure to* ***uncheck*** *the box “****Trigger even when the selected stages partially succeed by default it used to be unchecked though****”.*

*This should automate the deployment towards smoothly “****QA****” and “****PROD****” if “****DEV****” and “****QA****” succeed respectively and no “****Pre-deployment Approval****” is configured.*

1. To configure the “**Pre-deployment Approval**” to need enable it from the respective Stages with “**TimeOut**” settings [**Days**]. If multiple approvers need to configure, we can Configure the “**Approval Order**” like “**Any Order/In Sequence/Any One User**”. For best practice we need to make sure to configure the “Approval policies” like “**The user requesting a release or deployment should not approve it**”.

**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.*

***Answer: -***

1. *To deploy a VM through Azure DevOps release pipeline we need to have 2 artifacts, one is the “****Deployment Template (xxx.json)”*** *and another one is “****Parameters file (xxx.parameters.json).***

*The deployment Template where we will configure resources with desired configuration, here we need to configure [****Schema, Content Version,******Parameters, Variables, Resources and Output****]. For the best practices we can configure the template like where we have to less dependent on “****Resources****”. Desire state will be like just change the value of parameter schema to get deployed new resources in future.*

*The Parameter schema where we can update the parameters values like username, etc, we can reference to the KV secrets as well for sensitive information like passwords, connection strings.*

1. *List of the tools that need to be created are “storageAccount”[For Boot Diagnostics], “networkInterfaceCard”, “virtualNetwork”, “subnet”, “nsgRule” and “virtualMachine”.*
2. *To automate the deployment pipeline, we need to follow the below steps: - [Lets assume the template validation completed]*
3. *Create a new Release pipeline with “****Azure Repos******Git****” artifact where the template file and parameters files will be stored [We can use a build artifact as well, in that case we need to configure the build pipeline with copy task and deploy to Azure Artifact].*
4. *We need to Configure a new service connection which will integrate the Azure Subscription to the Azure DevOps. For every subscription we need to create individual service connections.*
5. *Add a Stage “****Dev****” and add a task there called “****ARM template deployment****” and need to configure as*

* *Deployment Scope 🡪 Resource Group*
* *Azure Resource Manager Connection 🡪 The SPN we configured with subscription*
* *Action 🡪 Create or Update Resource Group*
* *Resource Group 🡪 Resource Group Name (We can use RG Variable here as well)*
* *Template Location 🡪 Linked Artifact*
* *Template 🡪 Path of the deployment file [ex: VMAssignmentdeploy.json]*
* *Template Parameters 🡪 Path of the parameter deployment file [ex: VMAssignmentdeploy.parameters.json]*
* *Deployment Mode 🡪 Incremental/Complete [By Default Incremental]*
* *Deployment Name 🡪 Name of the deployment [This is an Optional]*
* *Control Options 🡪 Enabled 🡪 “Only When all previous tasks have succeeded”*

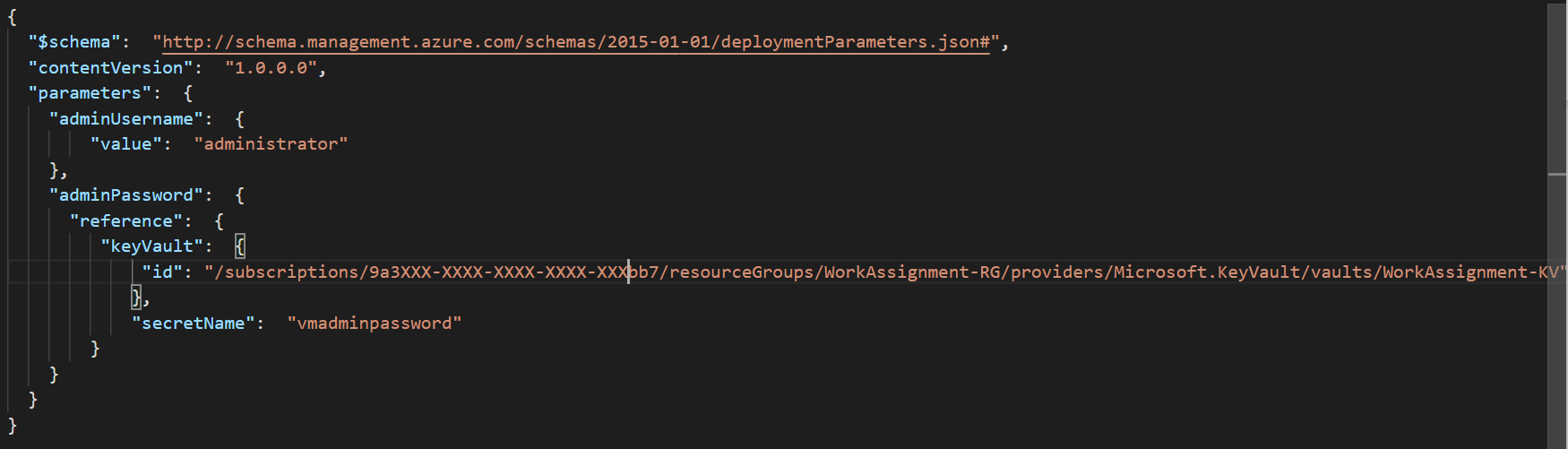
1. *We can specify parameters in “****Override Parameters****” section if we are not using any parameter file or don’t want to use any default value in template file.*
2. *We can specify the Variables as well for the best practices, so we don’t have to modify the tasks again and again, we can modify the variables only as per the environment scope.*
3. *We can add Variable Group as well where we will configure the group with key Vault, So we can get the secure secrets from key Vault.*
4. *Please find the attached “****deployment template****” and “****Parameters deployment file****”.*

* *

1. *Please find the below way to access the secure Admin password from key Vault in the VM ARM template.*

*We can add Key Vault reference in parameters deployment file like as below and we can configure the main deployment file as well, while made the deployment it will get the values for the secret “****vmadminpassword****”.*

*P.S: - Access policies need to be created for the SPN [We can create an AppRegistration and create an access policy with “****GET****” secret values].*

**

**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.*

***Answer: -***

1. *We need to setup the environment to configure middle layer API as Microservices using Azure AKS, the resources we need to setup are-*
2. *Need to create a Resource Group for this project.*

* *az group create --name aksassignmentrg --location eastus*

1. *Need to deploy an ACR in Azure (through Azure CLI or PowerShell).*

* *az acr create --resource-group aksassignmentrg --name acrassignment --sku Standard --location eastus*

1. *Need to create AKS in Azure (through Azure CLI or PowerShell).*

* *az aks create --resource-group aksassignmentrg --name aksassignment --enable-addons monitoring --kubernetes-version 1.18.0 --generate-ssh-keys --location eastus*

1. *Need to grant AKS service principal access to the ACR which will authorize the AKS cluster to connect to the azure container registry using AKS generated SPN.*

* *Create a Role assignment will achieve the same.*

1. *Need to create service connection with Azure DevOps.*
2. *Once the resources will be created, we need to configure the Build pipeline which will create an Image and push it to the “****ACR****”. Need to configure the Agent Job with suitable agent or agent pools.*
3. *We need to create a “****Docker-Compose****” file.*
4. *We need to create a “****Manifests file [Ex.- Deployment.yaml]****” which contains details of deployments, services and pods which will deployed in AKS.*
5. *Build pipeline need to configure with the below task as follows.*

* *Need to add a task called “****Docker Compose****” configure it with “****Run Service images****” action.*
* *Container Registry Type 🡪 Azure Container Registry*
* *Azure Subscription 🡪 “Set the subscription Name”*
* *Azure Container Registry 🡪 “name of the ACR from the dropdown box”*
* *Docker Compose File 🡪 “name of the Docker-Compose file”*
* *Project Name 🡪 “name of the project”*
* *Action 🡪 “Run Service Images” 🡪 Build images 🡪 Run in Background*
* *Working Directory 🡪 “$(System.DefaultWorkingDirectory)”*
* *Control Option 🡪 Only when all previous tasks have succeeded*
* *Need the same task “****Docker-Compose****” configure it with “****Build Service Images****” Action and configure like above.*
* *Additional Image Tags 🡪 $(Build.BuildId)*
* *Need the same task “****Docker-Compose****” configure it with “****Push Service Images****” Action and configure like above.*
* *Additional Image Tags 🡪 $(Build.BuildId)*
* *Need the same task “****Docker-Compose****” configure it with “****Lock Service Images****” Action and configure like above.*
* *Output Docker Compose File 🡪 “$(Build.StagingDirectory)/docker-compose.yml”*
* *Need add a task “****Publish Build Artifact****” which will publish the artifact into “****Azure Pipelines****” location so that they can utilize in Release Definition.*
* *Run the build pipeline and make sure we get the artifact in the “****Drop****” location and as well as the image should be push to the ACR repositories.*

1. *Once Build will be configured and working it as expected, we need to create the “****Release******Pipeline****” to do the AKS deployment. Need to follow the below steps- (Agent Job Need to be configured)*

* *Need to create the pipeline with Build Artifact from the above Build Pipeline*
* *Need to add a Stage and add a task called “****Deploy to Kubernetes****” and configure the same with “****Deploy****” action.*
* *Update the AKS service Connection from dropdown box.*
* *Namespace need to mention if we are looking for to deploy it in any specific Namespace in the cluster.*
* *Strategy we can put as None or Canary.*
* *We need to map the manifests file which will be deploy.*
* *Secret need to be mention which will be used to pull the image from ACR. [This secret we can create it from inside the AKS cluster by run the command “****kubectl create secret****” or we can add a task prior to the task which will create a secret (Need to use same task with “****Create Secret****” Action)]*
* *Variables need to be defined if we use any variable in the task.*
* *Now we can run the release pipeline which will deploy the image to the number of Pods we defined in the manifest file.*
* *Can configure Auto-Scaling as per matrix so we can mitigate the high traffic issue as well as we can set some health check monitor to the cluster (AKS given the observability to monitor the same from Azure Portal as well)*

*Once the Release will be complete, we can login to the AKS Cluster to see the Pods [****kubectl get Pods****] is running as expected or not. Once we will get the IP we can configure the “****coreDNS****” or “****kube-dns****” to achieve the same.*

*[P.S :- SQL connection String need to be configured in appSettings.json. Best practices will be push the connection string through a release pipeline Variable(Which can be pull from KV) for ndividual environments]*