**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.**

In Azure DevOps In the Build (CI) Pipeline in Triggers Tab when we Check Enable Continuous Integration so that this build will get invoked whenever the source changes are committed.

In Branch Filters we has to Include the Branch specification (there we can include the 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.**

In the Build Pipline where we build web, Api and Test , We will be having one task created Task Publish Test Results we have to check the option Fail If there are test failures.

By Check this option to fail the task if test failures are detected in the results files.

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

In the Release (CD) Pipeline at the Artifacts section we has to Enable Continuous deployment trigger which creates a release every time a new build is available. In the same way to automate the same on to Dev Environment at the pre-deployment conditions under triggers tab we has to select trigger After Release.

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

In the Release Pipeline at pre-deployment conditions under triggers of QA and Prod Stages

For QA - Select After Stage is Dev

For Prod – select After Stage is QA

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

In the pre-deployment conditions we have to Enable the Pre-deployment appovals

And we need to add the users who can approve the release.

We have Approval order are 1.any order 2 In sequence 3. Any one user based on the management decisions we will check that option and also Timeout too.

**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 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.**

There are two main artifacts one is Template file and second is template parameter file.

Template – Templates are JSON files, which contains Schema, Parameters, Variables, Resources and Outputs as data to deploy an infrastructure on Azure through Automation.

Template Parameters - When automating deployments it can be easier to pass a set of values for your environment. Parameter files make it easier to package parameter values for a specific environment in a separate Parameters file.

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

Visual Studio Code – Used to create/modify ARM templates.

Azure Repo’s/Github – Used to store ARM templates.

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

a). Prepare ARM templates and commit them to Azure Repos / Github

b). Create a build pipeline with Azure Repo / Github as the source for templates (artifacts)

c). Use MS-Hosted agent or Self-Hosted agent for execution of Pipeline

d). Create deployment task and create Agent Job’s under the task

e). Create service connection to Azure Subscription to deploy resources from Azure Devops.

f). Provide resource group information if already exists, or create a new one using Variables

g). In the Agent Job, choose ARM template deployment (predefined) job and provide artifact information such as which template to use and which parameter file to use for deployment.

h). After pipeline is configured with required inputs, save it and run to deploy resources automatically.

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

Template is pushed to git : <https://github.com/saikumar-azure11/Saikumar-Assessment>

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

a). Create a keyvault and create a secret which will be used as Admin Password for VM

b). Click on Access Policies in the KeyVault, select “Azure Resource Manager for template deployment” option and save.

c). Create a template parameter file and add a parameter named “adminPassword” with below code to access the secret that we have created earlier.

"adminPassword": {

"reference": {

"keyVault": {

"id": "Key Vault Id Here"

},

"secretName": "Name of the secret in Azure Key Vault"

}

}

d). keyVault Id can be retrieved from the properties of KeyVault and secretName should be the name of the secret that we have create earlier.

**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.**

**Steps Explanation**

1. Create a AKS cluster using Azure ARM template
2. Create a dotnet code \java api and docker file -- to be created by developer
3. Create a service connection to ACR.

**AKS Deployment Strategy:**

1. Devloper pushes the code to source that is Azure Repository
2. Azure Pipeline triggers the build by cloning the application code from azure repo and Here the build agents will build the job and generates the deployable artifacts, which can be pushed to a drop location for the artifact staging directory.
3. Azure pipeline creates docker image with application code artifacts , tag the image and push it to the ACR.
4. Azure pipeline tag the application code that was containerized and pushes the code to Azure repo
5. Azure pipeline executes the kubectl commands to deploy pods process starts from the image pulling from the ACR.
6. AKS Deployment process: 1. ConfigMap to store non-sensitive environment variables. 2.Store the all required secrets in keyvault.
7. Create Kubernetes objects (services,Ingress object etc) to communicate with pods
8. Provision a load balancer , forwarding rules and assign external public ip address
9. Here customer access the application from the public url’s

**Azure DevOps Build Pipeline for Continuous Integration**

Build pipeline for Kubernetes usually contains the steps that will fetch the code, build the Docker image, push that image to your Docker repository and then publish the artifacts.

1. Get sources from your code repository and choose the right repository and branch.
2. Define the agent you want to run for building your code (Windows agent, Ubuntu agent etc).
3. Select the Docker task and Action “Build an image”, choose the Azure container registry with Azure subscription and the name of the registry in case you want to use Azure one (Or you can configure any outside public registry), then choose the docker file name.
4. Select the task for docker image push. Also provide Azure container registry information where we can upload docker images for further AKS deployment process. Here we can keep our Image name "$(Build.Repository.Name):$(Build.BuildId)”
5. Select the “Copy files” step to copy the files to an artifact staging directory and we will only need to copy the deployment file
6. Select task Publish build artifacts step.

**Azure DevOps Release Pipeline for Continuous Deployment**

1. In release Pipeline we will put the correct build to its correct running environment that starts by reading the Artifacts from the build pipeline and fires one of the pipeline stages according to the build code branch.
2. One stage will be enough with only two tasks, one of them to copy the building code to the deployment yaml file so we can choose the right docker version for the release, and there is a ready pre-defined step for that purpose which is “Replace Tokens”. All you need to do is to select the deployment file from the Artifacts using “Root Directory” and “Target files” fields.
3. And second task “Deploy to Kubernetes” ready task from the templates, that task actually will only run Kubectl apply command on your deployment file. From the “Configuration file” field you can choose the deployment file you will use in that command.
4. we need to write our deployment file, and the best place for that file is inside our code. So this deployment file actually is creating a service for our code we choose the name of the service and the port which the service will work on. And It will be taking image defined in the deployment file and deploy it.