**Challenge 1**

A 3 Tier architecture is client-server architecture that consists of presentation tier, an application tier, and a data tier. The presentation tier would communicate other tiers through API calls. The application tier will handle the logical functions and the data tier manage and store the data from the client.

Below is the simple architectural diagram from 3 tier application.

Client 🡨🡪 Server🡨🡪 Database

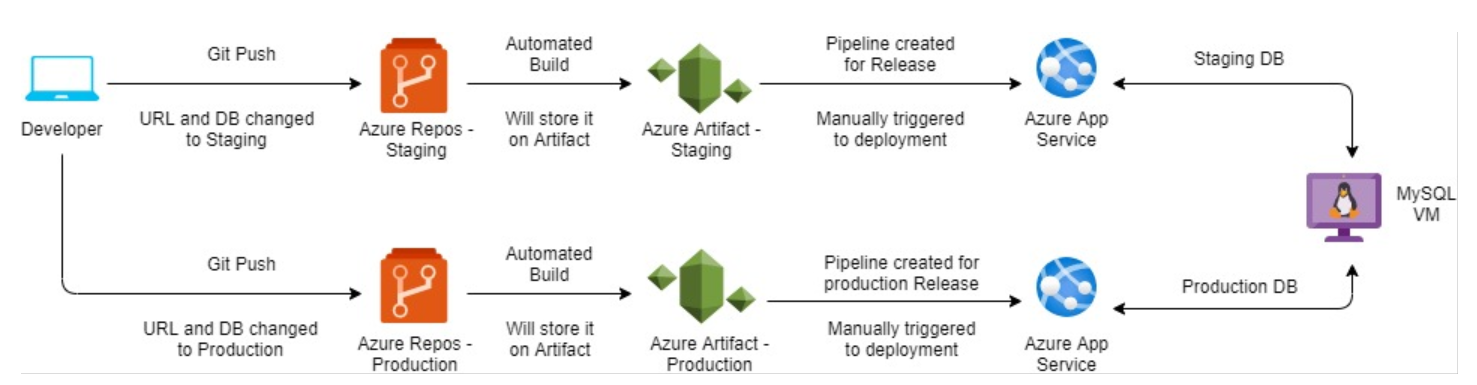
* Client is under the presentation layer what is who can access the application through endpoints like chrome, Firefox etc.
* Server would have the logical functions and the application running on it. It will get the request from client and communicate with backend server and provide the response to the client again.
* Database server will handle the request from application tier and manage and store the information of the client. Thers are lot of databases available in the market like MySQL, MS SQL, etc. We can install and configure the database server based on our requirement.

The below screenshot shows you the simple architectural diagram for deploying PHP application with MySQL backend.

Components involved in this Deployment.

1. Azure DevOps
2. Azure Subscription
3. Azure App Service
4. Azure App Service plan
5. Azure VM
6. Azure vNet
7. MySQL
8. Public IP and the DNS configuration

Simple Architectural Diagram



**Stage 1:**

* The developer completely needs to develop the source code and the application in his dev environment.
* Once the development has been completed, the source code will be pushed to the Azure repos which is for staging by changing its environment values.
* The automated build pipeline will create the build and store it on to Azure artifact for staging after the source code would be arrived on Azure repos.
* The administrator will manually trigger the build to the deployment on Azure app service by using release pipeline.

Once the deployment completed on stage 1 (Staging), we need to verify all the functionalities are working properly. Also, we need to complete UAT on this stage. The package will move forward to the production once the UAT will be completed.

**Stage 2:**

* The source code will be uploaded to the production repository on Azure with changing its environment values.
* The automated build pipeline will create the production build and store it on to Azure artifact after the source code would be arrived on Azure repos.
* The administrator will manually trigger the build to the deployment on Azure app service for production.

In this scenario, we used two repositories to manage source code. Otherwise, we can create multiple branches on the same repository to deploy staging and production.

Before that we need to install and configure all components involved and needs to communicate each other in a secure way.

Once the deployment has been done, we need to setup a DNS configuration based on our needs. So that client can easily access the application.

Azure pipeline used to take build.

# PHP as Linux Web App on Azure

# Build, package and deploy your PHP project to Azure Linux Web App.

trigger:

- master

variables:

  # Agent VM image name

  vmImageName: 'ubuntu-latest'

   Environment name

  environmentName: 'test-kpmg'

  # Root folder under which your composer.json file is available.

  rootFolder: $(System.DefaultWorkingDirectory)

stages:

- stage: Build

  displayName: Build stage

  variables:

    phpVersion: '7.3'

  jobs:

  - job: BuildJob

    pool:

      vmImage: $(vmImageName)

    steps:

    - script: |

        sudo update-alternatives --set php /usr/bin/php$(phpVersion)

        sudo update-alternatives --set phar /usr/bin/phar$(phpVersion)

        sudo update-alternatives --set phpdbg /usr/bin/phpdbg$(phpVersion)

        sudo update-alternatives --set php-cgi /usr/bin/php-cgi$(phpVersion)

        sudo update-alternatives --set phar.phar /usr/bin/phar.phar$(phpVersion)

        php -version

      workingDirectory: $(rootFolder)

      displayName: 'Use PHP version $(phpVersion)'

    - script: composer install --ignore-platform-reqs

      workingDirectory: $(rootFolder)

      displayName: 'Composer install'

    - task: CopyFiles@2

      inputs:

        SourceFolder: '$(System.DefaultWorkingDirectory)'

        Contents: |

          \*\*

          !/home/vsts/work/1/s/.git/\*\*

        TargetFolder: '$(Build.ArtifactStagingDirectory)'

    - task: ArchiveFiles@2

      displayName: 'Archive files'

      inputs:

        rootFolderOrFile: '$(Build.ArtifactStagingDirectory)'

        includeRootFolder: false

        archiveType: zip

        archiveFile: $(Build.ArtifactStagingDirectory)/$(Build.BuildId).zip

        replaceExistingArchive: true

    - upload: $(Build.ArtifactStagingDirectory)/$(Build.BuildId).zip

      displayName: 'Upload package'

      artifact: staging

The above script would help to take build after the code pushed to the repository and stored it on artifacts.

And then, we need to create and manage release pipeline on Azure DevOps for the deployment.

**Challenge 2**

I was mainly working on Azure and GCP. So, I do not have much exposure on AWS. But the task will be quite interesting, so I just tried to create an EC2 instance on AWS and had some reference from AWS user guide to get a metadata of the instance, which is created, and the source was uploaded on GitHub.

If I had a chance to work with multiple cloud technologies, I will be happy to work with and can quickly learn those technologies which is not familiar and able to adapt as much as possible.

**Challenge 3**

Access a:

----------------

if (object.ContainsKey("a"))

{

var a = object["a"]

}

Access b:

----------------

if (a.ContainsKey("b"))

{

var b = a["b"]

}

Access c:

----------------

if (b.ContainsKey("c"))

{

var c = b["c"]

}

Access d:

----------------

if (c.ContainsKey("d"))

{

var d = c["d"]

}

Access x:

----------------

if (object.ContainsKey("x"))

{

var x = object["x"]

}

Access y:

----------------

if (x.ContainsKey("y"))

{

var y = x["y"]

}

Access z:

----------------

if (y.ContainsKey("z"))

{

var z = y["z"]

}

Access a:

----------------

if (z.ContainsKey("a"))

{

var a = z["a"]

}