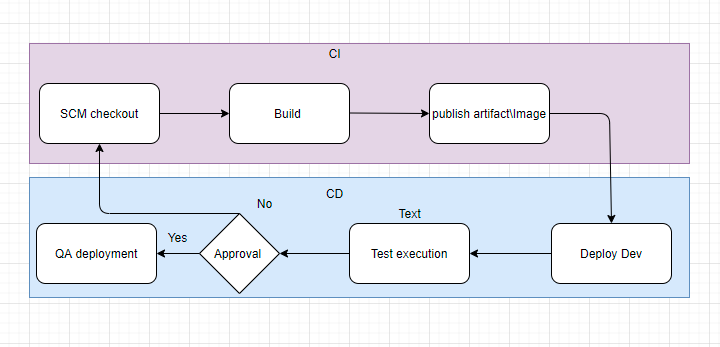
* Explain how each of the above the requirements will be met using Azure DevOps configuration.
* Explain the steps with configuration details.



**CI:**

* Initially the SCM tools will be setup with the branching strategy to be followed, say Feature, develop, release, Master.
* We can templetise the pipeline.yaml which will be stored in a shared repository as template.

From the development Code repository we can call the pass the parameters to the pipeline template by calling with the template name via **jobs** step.

* Auto trigger can be by defining in azure pipeline.

*Trigger :<branchname>*

* The Web, API, test project will need to be Build with task as *“DonNetCoreCLI”*

or

*“Maven task”*

* You can add the*“Run test task”*

Which will trigger your test and with

*“Publish test result task”*

you can publish the test result and check mark the “Fail if test failures” and fail your whole CI pipeline.

* For the build to fail on a test failure, you need to configure *Nunit for .net and Junit for Java,* where it will be defined in the build file as dependency. So whenever a unit test fails the build will automatically fail.
* Once the code quality and coverage is run from the Azure pipeline with the **Sonar task in azure**, you can end your CI cycle and start the CD cycle.
* Use *“Artifactory Build Promotion task”* to Artifactory from the target build folder, or to publish to the *Azure container registry* we can use the*“Docker”* **task** with command *“buildAndPush”*

**CD:**

* To have deployment we can create different environment under the *Azure DevOps->Environment* and from there we can set the **Approvers under** *Environment->QA or prod->approvals and checks-> add approvers* and also specify number of approvers and. Also we can define at release **pipeline** *“pre deployment approval”* **and** *“post deployment approvals”.*Specify in yaml as *“environment:QA”*
* Fetch the Artifact from the Artifact repo use the *“DownloadBuildArtifacts”* task or if you are using Kubernates based deployment then we can use the deployment manifest yaml for Kubernates where the image with ACR will be mentioned.
* For deployment we can *use [script, powershell, Kubernates deployment manifest, Helm charts, ansible]* or a third party deployment tool can also be integrated for deployments.
* We can also specify deployment strategy as **Rolling strategy, Blue/green deployment, Canary deployment** supported as in yaml*“strategy ->rolling,canary,runOnce”*

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 Terraform templates.

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

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

Vnet

2 Subnet

NSG to open port 80 and 443

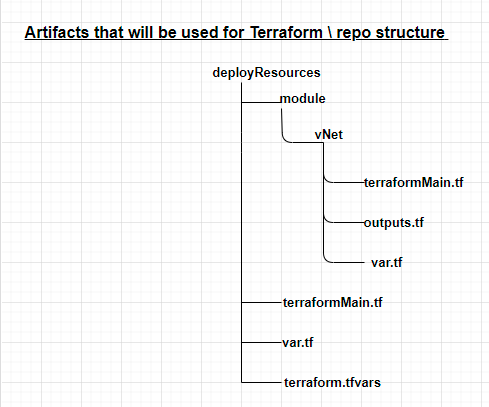
1 Window VM in each subnet

1 Storage account

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

Terraform template.

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



**deployResources:** Here will keep the configuration files and will act as the working directory.

**terraformMain.**tf: Is file to declare resource or configuration that we want to create and manage.

**vNet:** This folder will contain the files to deploy the vnet resource in Azure.

**Var.**tf : This is used as a variable file, which accepts the values from the calling module.

**Outputs.**tf : Here we accept the values of the resources that is created by the terraform and can be used as input for creating resources from the calling module.

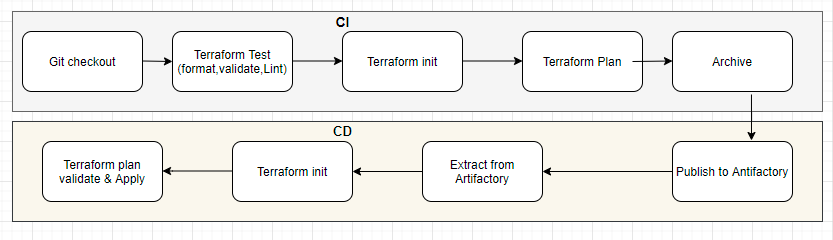
**Terraform.**tfvars: This file will be used that can be assigned to the variables.

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

Tools that will be used.

* Terraform Editor: **Atom**
* To store terraform templates: SCM tools like **GIT, Bitbucket** etc.
* Install **Azure cli**
* **Storage account** to store **terraform state file**: which holds the state of your terraform deployments.
* Azure DevOps pipeline

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



* Create repo in azure DevOps for terraform.
* Install Azure CLI
* Install Terraform extension in Azure DevOps to run the the terraform commands from Azure DevOps pipeline.
* Create a Azure Storage account to store the terraform sate file.
* Push the terraform code that you created in Atom editor to the Azure repo via git client.
* Create the service connection in azure DevOps from the projects settings of Azure DevOps.
* Select the subscription and the tenant and assign service principal with contributor access in the Azure AD and give the connection name as terraform connect. This will establish a connection of your Azure DevOps to deploy resources to Azure.

CI pipeline steps:

* Create the yaml file where you will checkout the code with first stage.
* Specify the Agent in the yaml file which will used as build agent.
* Add the **ArchiveFiles** task in you yaml to package the files of terraform and another task to zip the files.
* Use **PublishBuildArtifacts** task the zip file to the artifactory or drop. So that the artefact can be consumed by the CD stage.

CD pipeline Steps:

* Copy or download the build artefact from the drop or the artifactory with CI yaml pipeline by adding the copy steps or the artifactory download steps, to get the Artifact to the Azure DevOps Agent.
* Invoke the Terraform tool installer as next step to install the terraform to run the terraform commands from the azure Agent.
* Add the step to invoke the command via TerraformTask to invoke *Terraform init, then terraform validate, terraform plan* and then *terraform apply.*

4) Explain how will you access the password stored in Key Vault and use it as Admin Password in the VM

* Via Azure variable group we can establish connection by check mark the option to link to azure key vault and accessing it in the pipeline and then use it in Terrafrom via TF\_VAR\_
* We can pass DevOps variables that have been populated with Secrets values into the Terraform task as Environment Variables.

*export TF\_VAR\_<secretname>=xyz*