Task 2:

The steps below will explain a complete flow of CI/CD for application code.

Branching Strategy:

There are two ways to implement branching strategy, either via Main branch or via Release branch.

Each has its pro and con, like the criticality, legacy code, dependency on other applications, monolithic or microservice architecture, Automation Regression testing and also plays a factor in selecting the flow.

The Diagram below will display both the strategies.

![Diagram, schematic

Description automatically generated]()

I will be describing below the Master branch strategy.

Diagram

Description automatically generated

The develop will push the code via development tool, rest of all the process need to take care by CI/CD.

First step is, when to trigger the CI.

1. So trigger is important :

trigger:

branches:

include:

master

feature

develop

1. Now we can set up some env variables and call the key vault to take care of sensitive information and secrets.

steps:

task: AzureKeyVault@2

inputs:

azureSubscription: '' # service connection name

KeyVaultName: '' # azure keyvault name

SecretsFilter: '' #filter the secrets based on prod or dev or staging.

RunAsPreJob: true

This will help in preventing exposure of sensitive information., we can store Access key’s, Secrets key, User name, passwords, other information in Key vaults.

1. A code analysis tool need to added into the CI in order to check code quality

There are many tools like sonarqube, fortify, sonarcloud to perform this task.

task: SonarQubePrepare@4

displayName: Prepare Analysis Configuration task

inputs:

SonarQube: ''

scannerMode: ''

configMode: ''

* + cliProjectKey:

We can add a sanity check in the CI. In order to avoid sending bad code to next faces, if the code quality has failed, it should be able to stop the build process.

# pass or fail the build based on quality gate status

if ($result.projectStatus.status -eq "OK") {

Write-Host "SonarQube Quality Gate succeeded. continuing build..."

}else{

throw "SonarQube Quality Gate failed. Please review the code analysis results from SonarQube dashboard. "

}

1. The Next stage in the CI is now building the images:

stage: Build

displayName: Build stage

jobs:

job: Build

displayName: Build

Now the build process can start , which can include installing packages, then any special command to build the image.

1. After the image has been built, it is needs to be pushed to ECR.

displayName: 'Login to AWS'

env:

AWS\_ACCESS\_KEY\_ID: # can be provided via vault

AWS\_SECRET\_ACCESS\_KEY:

AWS has a feature where you can push the image to one repo and it will replicated in other environments, which makes it easier and secure as we need no create prod repo access in CI.

1. Now the Artificats, build can be published.

- task: Docker@2

displayName: Build and push an image to container registry

inputs:

command: buildAndPush

repository: ## key Vault contains the information about it

dockerfile: /// path of the docker file to build the image.

tags: $(tag). --- add tag to the build image.

1. Now the image need to deployed into the Development system first, need to deploy some common config and

- stage: Deploy

displayName: Deploy stage

dependsOn: Build

jobs:

- deployment: deploy\_common

displayName: Deploy

environment: variable need to be passed

strategy:

runOnce:

deploy:

steps:

- task: KubernetesManifest@0

displayName: Rollout common infra

inputs:

action: deploy

namespace: namespace to which the config need to be deployed

manifests: ## manifest file path and name of file to be added.

1. Finally it can deployed to Dev cluster

- task: KubernetesManifest@0

displayName: Deploy to Kubernetes cluster

inputs:

action: deploy

manifests: |

$(Pipeline.Workspace)/manifests/deployment.yaml

imagePullSecrets: |

$(imagePullSecret)

In deployement file.

There is whole lot of things that can be controlled like tls mode, version, port of container, CPU,memory and whole of other features.

trafficPolicy:

tls:

mode: ENABLED

spec:

containers:

- name:

image:

ports:

- containerPort: 8090

resources:

limits:

cpu: "100m"

memory: "64Mi"

requests:

cpu: "50m"

memory: "32Mi"

Certificate issuer like api token and key can be put into key vault.

apiVersion: cert-manager.io/v1alpha2

kind: ClusterIssuer

metadata:

name: letsencrypt-nm

……..

- dns01:

cloudflare:

apiTokenSecretRef:

key:

name:

email:

1. Once the deployement has been completed in Dev system,

We can trigger sanity testing and testing of the application.

Different tools and practises are used by teams.

jobs:

displayName: testing

environment: variable need to be passed

input:

run: npm test

run: npm run manual:tests

run: npm run docs

1. Once the Code review is performed by two team members, the same branch will be merged and code be propagated to Staging.

Code approval can be set Automated or Manual approval from dev to Staging.

It depends upon team to team and company to company.

1. Now different testing tools like DAST, Regression, Integration, Performance and load testing is performed on the system. It can be automated or manual.

Once the Staging build has satisfied the product criteria, this can be migrated to Production.

1. Production deployment process can be manual or automated depending upon the approval process on build.

Production deployment can be either blue/green in which code is deployed to standby system and cut over is performed based on weight distribution on monitor the performance of the system.

If it is cluster, it can be performed using canary deployment.

1. In order to prevent Unauthorized changes to the code, the access restriction on the repo should be performed, what access level the user has.
2. In order to merge the code from feature or dev branch to Main/master branch. It needs two dev code review approval’s . We can add that as part of the custom code to manage security.