# **3. Setting Up a Basic CI Pipeline on Azure**

**B.Steps required to set up a CI pipeline using GitLab CI or Azure DevOps on Azure.**

## **Using GitLab CI/CD on Azure**

1. Set Up Your Azure Resources
2. Set Up Your GitLab Repository
3. Define GitLab CI/CD Pipeline

stages:

- build

- test

- deploy

variables:

AZURE\_CLIENT\_ID: "<your-client-id>"

AZURE\_TENANT\_ID: "<your-tenant-id>"

AZURE\_SUBSCRIPTION\_ID: "<your-subscription-id>"

AZURE\_CLIENT\_SECRET: "<your-client-secret>"

before\_script:

- apt-get update && apt-get install -y curl

- curl -sL https://aka.ms/InstallAzureCLIDeb | bash

- az login --service-principal -u "$AZURE\_CLIENT\_ID" -p "$AZURE\_CLIENT\_SECRET" --tenant "$AZURE\_TENANT\_ID"

build:

stage: build

script:

- echo "Building application"

- dotnet build

test:

stage: test

script:

- echo "Running tests"

- dotnet test

deploy:

stage: deploy

script:

- echo "Deploying to Azure"

- az webapp deployment source config-zip --resource-group MyResourceGroup --name MyWebApp --src my-app.zip

4. Configure GitLab CI/CD Variables

5. Run the Pipeline

# **Using Azure DevOps for CI/CD**

1. Set Up Azure DevOps

2. Create a Build Pipeline

3. Configure Service Connections

4. Run and Monitor the Pipeline

**C.** **Define pipeline stages:**

**1. Build Stage**

build:

stage: build

script:

- echo "Installing dependencies"

- dotnet restore # For .NET apps

- npm install # For Node.js apps

- mvn clean package # For Java apps

- echo "Building application"

- dotnet build --configuration Release # Adjust for your language/framework

artifacts:

paths:

- bin/Release/

* **Dependencies Installation**: Ensures all necessary packages are installed.
* **Build Execution**: Runs the build process.
* **Artifacts**: Stores built files for later use in the pipeline.

**Azure DevOps Configuration**

stages:

- stage: Build

jobs:

- job: Build

pool:

vmImage: 'ubuntu-latest'

steps:

- task: UseDotNet@2

inputs:

packageType: 'sdk'

version: '6.x'

- script: |

dotnet restore

dotnet build --configuration Release

- task: PublishBuildArtifacts@1

inputs:

pathToPublish: '$(Build.ArtifactStagingDirectory)'

artifactName: 'drop'

* Uses a **Microsoft-hosted agent** (ubuntu-latest).
* Publishes built artifacts for the next pipeline stages.

**2. Test Stage:**

GitLab CI/CD Configuration

test:

stage: test

script:

- echo "Running tests"

- dotnet test --logger trx # .NET example

- npm test # Node.js example

- mvn test # Java example

* Runs unit tests for different languages.
* Can include integration tests.

Azure DevOps Configuration

- stage: Test

jobs:

- job: Test

pool:

vmImage: 'ubuntu-latest'

steps:

- script: dotnet test --logger trx

displayName: 'Run Unit Tests'

- task: PublishTestResults@2

inputs:

testResultsFiles: '\*\*/\*.trx'

testRunTitle: 'Unit Tests'

* **dotnet test** runs unit tests.
* **PublishTestResults** uploads results for viewing in Azure DevOps.

**3.Deploy Stage:**

GitLab CI/CD Configuration

deploy:

stage: deploy

before\_script:

- mkdir -p ~/.ssh

- echo "$SSH\_PRIVATE\_KEY" > ~/.ssh/id\_rsa

- chmod 600 ~/.ssh/id\_rsa

script:

- echo "Deploying to Azure VM"

- scp -o StrictHostKeyChecking=no -r bin/Release/\* $AZURE\_VM\_USER@$AZURE\_VM\_IP:/var/www/app

- ssh -o StrictHostKeyChecking=no $AZURE\_VM\_USER@$AZURE\_VM\_IP "sudo systemctl restart myapp"

* Uses **SCP** to copy files to the **Azure VM**.
* Restarts the application via **SSH**.

Azure DevOps Configuration

- stage: Deploy

jobs:

- job: Deploy

pool:

vmImage: 'ubuntu-latest'

steps:

- task: CopyFilesOverSSH@0

inputs:

sshEndpoint: 'MyAzureVM'

sourceFolder: '$(Build.ArtifactStagingDirectory)/drop'

targetFolder: '/var/www/app'

- task: SSH@2

inputs:

sshEndpoint: 'MyAzureVM'

script: 'sudo systemctl restart myapp'

* Uses **CopyFilesOverSSH** to deploy artifacts.
* Restarts the service with **SSH task**.