

Module 7 - Continuous Deployment (CD)

1. Multi-Stage Pipelines (YAML)

Azure Pipelines supports **multi-stage YAML pipelines**, allowing teams to model **Build → Test → Deploy** workflows in a single, version-controlled YAML file.

Traditional CI/CD required separate pipelines (Build Pipeline + Release Pipeline).
multi-stage pipelines combine everything into a **single end-to-end artifact**, improving:

- Traceability
 - Maintainability
 - Auditability
 - Reuse
 - Standardization
-

1.1 Structure of a Multi-Stage Pipeline

A YAML multi-stage pipeline includes:

- **trigger / PR** conditions
- **variables / variable-groups / secrets**
- **stages → jobs → steps → tasks**

Example Structure:

stages:

- stage: Build

jobs:

- job: BuildJob

steps:

- task: DotNetCoreCLI@2

- stage: Test

dependsOn: Build

jobs:

- job: TestJob

```
- stage: Deploy
  dependsOn: Test
  jobs:
  - deployment: DeployWebApp
    environment: 'Prod'
    strategy:
      runOnce:
        deploy:
          steps:
            - task: AzureWebApp@1
```

1.2 Key Concepts

Stages

Logical boundaries representing CI/CD phases:

- Build
- Test
- Security Scans
- Deploy to DEV
- Deploy to QA
- Deploy to PROD

Jobs

Parallel units of work inside a stage.

Steps

The actual tasks (CLI, scripts, built-in tasks).

Dependencies

Stages use:

dependsOn: Build

This ensures proper sequencing.

2. Environments & Approvals

Azure DevOps Environments represent **physical or logical deployment targets**.

Environments can be:

- Azure App Services
 - Azure Kubernetes Service (AKS)
 - Virtual Machines
 - On-prem servers
 - Generic “logical” environments (DEV, QA, UAT, PROD)
-

2.1 Features of Environments

◆ Approvals & Checks

- Manual approval
- Business rule checks
- Azure Policy
- Deployment gates
- Integration with ServiceNow, JIRA, etc.

◆ Deployment history & audit

Shows:

- Who approved
- What was deployed
- When it was deployed
- What version was deployed

◆ Resource Mapping

You can associate:

- App Service resources
 - VMs
 - Kubernetes namespaces
-

2.2 Configuring Approvals

1. Go to **Pipelines → Environments**
2. Click the environment (e.g., Prod)
3. Select **Approvals and Checks**
4. Add:

- Mandatory manual approval
- Timeout
- Reviewer groups

5. Save

In YAML:

environment:

name: 'Prod'

resourceName: 'prod-webapp'

resourceType: 'AzureWebApp'

Azure DevOps automatically respects approval rules before the Deploy stage runs.

3. Deployment Strategies

Azure DevOps supports multiple deployment strategies for rolling out changes safely.

3.1 Basic Types

1. RunOnce (Default)

Deploy everything in a single execution.

strategy:

runOnce:

deploy:

steps:

- task: AzureWebApp@1

2. Rolling Deployment

Deploy incrementally to batches (VM scale sets, App Service slot instances).

Used when:

- You have multiple instances
- You want gradual rollout without downtime

strategy:

rolling:

maxParallel: 2

preDeploy:

steps: []

deploy:

steps: []

3. Blue-Green Deployment

Two identical environments:

- **Blue** = current version
- **Green** = new version → tested → swap

Benefits:

- Near-zero downtime
- Easy rollback (swap back to Blue)

Azure App Service supports **slot swapping**, used for Blue-Green.

YAML:

- task: AzureAppServiceManage@0

inputs:

Action: 'Swap Slots'

SourceSlot: 'staging'

ResourceGroupName: 'rg-app'

WebAppName: 'myapp'

4. Canary Deployment

Route a small percentage of traffic to a new version.

Used with:

- Application Gateway
- Azure Front Door
- Traffic Manager
- Kubernetes (Istio/Envoy)

Example concept:

- 5% → Canary
- Monitor → Increase to 25%

- Monitor → Increase to 50%
- Monitor → 100% rollout

Azure DevOps YAML example (traffic-splitting conceptual, not complete):

strategy:

canary:

increments: [5, 25, 50, 100]

(Actual implementation depends on your routing layer.)

4. Lab: Deploy Application to Azure App Service (Multi-Stage Pipeline)

This lab deploys a .NET Web App or Node/Java app.

4.1 Prerequisites

- Azure subscription
 - Azure DevOps project
 - Service connection to Azure (ARM or OIDC recommended)
 - Azure App Service created:
 - WebApp name: myapp-dev
 - Resource Group: rg-app
 - Docker or VM optional for bonus sections
-

4.2 Multi-Stage YAML Pipeline – Full Working Example

Create azure-pipelines.yml at repo root:

trigger:

branches:

include:

- main

pool:

vmImage: 'ubuntu-latest'

variables:

buildConfiguration: 'Release'

vmImage: 'ubuntu-latest'

webAppName: 'myapp-dev'

artifactName: 'drop'

stages:

Stage 1: Build

- stage: Build

displayName: 'Build and Publish'

jobs:

- job: BuildJob

displayName: 'Build'

steps:

- task: UseDotNet@2

displayName: 'Install .NET SDK'

inputs:

packageType: 'sdk'

version: '8.0.x'

- task: DotNetCoreCLI@2

displayName: 'Restore'

inputs:

command: 'restore'

projects: 'src/WebApp/WebApp.csproj'

- task: DotNetCoreCLI@2

displayName: 'Build'

inputs:

command: 'build'

```
projects: 'src/WebApp/WebApp.csproj'
arguments: '--configuration $(buildConfiguration)'
```

```
- task: DotNetCoreCLI@2
  displayName: 'Publish'
  inputs:
    command: 'publish'
    publishWebProjects: true
    arguments: '--configuration $(buildConfiguration) --output $(Build.ArtifactStagingDirectory)'
    zipAfterPublish: true
```

```
- publish: $(Build.ArtifactStagingDirectory)
  artifact: $(artifactName)
```

```
# -----
```

```
# Stage 2: Deploy to Dev
```

```
# -----
```

```
- stage: Deploy_Dev
  displayName: 'Deploy to Dev'
  dependsOn: Build
  condition: succeeded()
  jobs:
    - deployment: DeployDev
      displayName: 'Deploy Dev'
      environment: 'Dev'
      strategy:
        runOnce:
          deploy:
            steps:
              - download: current
                artifact: $(artifactName)
```

```
- task: AzureWebApp@1

  displayName: 'Deploy App to Dev'

  inputs:

    azureSubscription: 'MyServiceConnection'

    appType: 'webApp'

    appName: 'myapp-dev'

    package: '$(Pipeline.Workspace)/$(artifactName)/**/*.zip'
```

```
# -----
```

```
# Stage 3: Deploy to Prod (manual approval)
```

```
# -----
```

```
- stage: Deploy_Prod

  displayName: 'Deploy to Prod'

  dependsOn: Deploy_Dev

  condition: succeeded()

  jobs:

    - deployment: DeployProd

      displayName: 'Deploy Prod'

      environment: 'Prod'      # has approval enabled

      strategy:

        runOnce:

          deploy:

            steps:

              - download: current

                artifact: $(artifactName)
```

```
- task: AzureWebApp@1

  displayName: 'Deploy App to Prod'

  inputs:

    azureSubscription: 'MyServiceConnection'
```

appName: 'myapp-prod'

package: '\$(Pipeline.Workspace)/\$(artifactName)/**/*.zip'

4.3 What This Pipeline Demonstrates

Multi-stage pipeline

- Build
- Deploy to Dev
- Deploy to Prod

Uses environments

- Dev → auto deployment
- Prod → requires approval before deployment

Deployment Strategies Supported

- runOnce (default)
 - Can be easily extended to **blue-green** using App Service slots
 - Can be extended to **canary** using Front Door / Gateway
 - Can be extended to **rolling** for VM Scale Sets
-

5. Bonus Lab: Deploy to VM (instead of App Service)

Prerequisites

- Linux VM with:
 - SSH enabled
 - Node/Java/.NET runtime installed

YAML Example:

- stage: DeployVM

dependsOn: Build

jobs:

- deployment: VMDeploy

environment:

name: 'VM-Prod'

resourceType: VirtualMachine

strategy:

```
runOnce:

deploy:

  steps:

    - task: CopyFilesOverSSH@0

      inputs:

        sshEndpoint: 'MyVMConnection'

        contents: '**/*.zip'

        targetFolder: '/var/www/myapp/'

    - task: SSH@0

      inputs:

        sshEndpoint: 'MyVMConnection'

        runOptions: 'commands'

        commands: |

          unzip /var/www/myapp/*.zip -d /var/www/myapp/

          systemctl restart myapp
```

6. Summary

This chapter covered:

Multi-Stage YAML Pipelines

- End-to-end CI/CD in a single YAML
- Stages → Jobs → Steps hierarchy

Environments & Approvals

- Manual approvals
- Deployment gates
- Audit trail

Deployment Strategies

- RunOnce
- Rolling
- Blue-Green
- Canary

Hands-On Labs

- Build & deploy app to Azure App Service
- Optional VM deployment
- Multi-stage pipeline with artifacts and approvals