Introduction

Azure DevOps provides a suite of capabilities for building, testing, and deploying applications through Continuous Integration and Continuous Deployment (CI/CD) pipelines. This document outlines these capabilities and provides YAML examples for implementing them in Azure Pipelines.

1. Azure DevOps Capabilities

Azure DevOps includes several services that support software development lifecycle:

1.1 Azure Repos

A version control system supporting Git repositories to manage source code and track changes.

1.2 Azure Pipelines

A CI/CD service for building, testing, and deploying applications.

1.3 Azure Artifacts

A package management system that hosts and shares Maven, npm, and NuGet packages.

1.4 Azure Test Plans

A toolset for manual and exploratory testing, as well as automated testing.

1.5 Azure Boards

An Agile planning tool for tracking work items, issues, and project progress.

Three ways to write pipeline-

- **YAML Pipelines** → Best for modern DevOps, fully automated CI/CD, and version control.
- **Classic Build** → Suitable for teams needing only CI and prefer UI-based setup.
- **Classic Release** → Good for enterprise deployments requiring manual approvals

Feature	YAML Pipelines (CI/CD)	Classic Build (CI)	Classic Release (CD)
Definition	Code-based CI/CD pipeline	UI-based build pipeline	UI-based release pipeline
Storage	In repository as .yml file	Azure DevOps (not in repo)	Azure DevOps (not in repo)
CI/CD Support	Both CI & CD	Only CI (build)	Only CD (deploy)
Version Control	Yes, in Git	No (not stored as code)	No (not stored as code)
Flexibility	High (customizable)	Medium (task-based)	Medium (staged deployments)
Ease of Use	Harder to learn initially	Easier (UI-based)	Easier (UI-based)
Best for	Modern DevOps, GitOps	Legacy projects, UI-friendly CI	Enterprises needing manual approvals

1) Agents: Agents execute pipeline jobs.

```
pool:
  name: Default
```

2) Approvals: Approvals enforce manual validation before proceeding.

```
stages:
- stage: Deploy
  approval:
    approvals:
      - approvers:
          - user1@example.com
          - user2@example.com
```

3) Artifacts: Artifacts store pipeline outputs for later use.

```
steps:
- task: PublishBuildArtifacts@1
  inputs:
    pathToPublish: $(Build.ArtifactStagingDirectory)
    artifactName: drop
```

4) Caching: Caching stores dependencies to speed up builds.

```
steps:
- task: Cache@2
  inputs:
    key: 'npm | $(Agent.OS) | package-lock.json'
    path: $(NPM_CACHE_FOLDER)
```

Note: Suppose, which we need to deploy by 2 pipelines in selfhosted agent, so caching can help here.

5) Conditions Condition control job execution.

```
condition: eq(variables['Build.SourceBranch'], 'refs/heads/main'
```

6) Container Jobs Run jobs inside containers.

```
jobs:
- job: ContainerJob
  container: mcr.microsoft.com/windows/servercore:ltsc2019
  steps:
    - script: echo "Running in container"
```

7) Demands: Ensure agents meet specific requirements.

```
pool:
  name: Default
  demands:
    - java
    - maven
```

8) Dependencies: Define dependencies between jobs.

```
jobs:
- job: Job1
  steps:
    - script: echo "Job1 completed"
- job: Job2
  dependsOn: Job1
  steps:
    - script: echo "Job2 depends on Job1"
```

9) Deployment Groups: Group machines for deployment.

```
environments:
- name: Production
  resourceType: VirtualMachine
```

10) Deployment Group Jobs: Deploy to a group of machines.

```
jobs:
- deployment: DeployWebApp
  environment: Production
  strategy:
    runOnce:
      deploy:
        steps:
          - script: echo "Deploying to production"
```

11) Deployment Jobs: Define deployment-specific jobs.

```
jobs:
- deployment: DeployJob
  environment: staging
  strategy:
    runOnce:
      deploy:
        steps:
          - script: echo "Deploying..."
```

12) Environment: Define execution contexts.

```
environments:
  - name: Staging
```

13) Gates: Gates enforce quality checks before proceeding.

```
stages:
- stage: Validate
  preDeployGates:
    timeoutInMinutes: 5
    gates:
      - queueTime: 5
```

14) Job: A job is a collection of steps.

```
jobs:
- job: Build
  steps:
    - script: echo "Building..."
```

15) Service Connections: Connect to external services.

```
resources:
  repositories:
    - repository: myRepo
      type: github
      endpoint: myGitHubServiceConnection
```

16) Service Containers: Use containers in pipelines.

17) Stages: Stages group jobs together.

```
resources:
  containers:
    - container: myContainer
      image: node:14
```

18) Task Groups: Reusable sets of tasks.

```
taskGroups:
- name: MyTaskGroup
  tasks:
    - task: Bash@3
      inputs:
        targetType: inline
        script: echo "Reusable Task Group"
```

19) Tasks: Atomic units of work in a pipeline.

```
steps:
- task: Bash@3
  inputs:
    targetType: inline
    script: echo "Hello World"
```

20) Templates: Reusable pipeline configurations.

```
jobs:
- template: common-build.yml
  parameters:
    buildConfiguration: Release
```

21) Triggers: Automate pipeline execution.

```
trigger:
  branches:
    include:
      - main
```

22) Variables: Store values for reuse.

```
variables:
  buildConfiguration: Release
```

23) Variable Groups: Store and manage variables centrally.

```
variables:
  - group: MyVariableGroup
```