**What is CI/CD?**

**CI/CD** stands for:

* **CI** = **Continuous Integration**
* **CD** = **Continuous Delivery** *(or)* **Continuous Deployment**

It is a **modern DevOps practice** used to automate and streamline the software **build → test → release** process.

**🔹 CI – Continuous Integration**

**Purpose:**  
Automatically build and test code whenever developers push changes to a shared repository.

**🔧 Key Activities:**

* Code is pushed (Git commit)
* Automated build is triggered
* Automated unit/integration tests are run
* Developers are notified of build/test results

**✅ Benefits:**

* Early bug detection
* Faster feedback loop
* Improved code quality
* Easier integration across teams

**🔹 CD – Continuous Delivery / Deployment**

There are **two types of CD**:

**1. Continuous Delivery**

* Automatically prepares code for **release to production**
* Final **manual approval** before production deploy

**2. Continuous Deployment**

* Goes one step further: **automatically deploys to production**
* **No manual approval** needed

**🚀 Activities:**

* Create release package or artifact
* Deploy to Dev → QA → Staging → Production
* Run post-deployment checks

**🧪 Real-world Tools for CI/CD:**

| **Stage** | **Tools** |
| --- | --- |
| Source Code | Git, GitHub, GitLab, Azure Repos |
| CI/CD | **Azure DevOps**, GitHub Actions, Jenkins, GitLab CI, CircleCI |
| Build/Test | Maven, Gradle, npm, dotnet, JUnit, Pytest |
| Deploy | Azure Pipelines, AWS CodeDeploy, Helm, Terraform, Kubernetes |

**🔹 What is Azure Pipelines?**

Azure Pipelines is a **CI/CD service** in Azure DevOps that automates:

* **Continuous Integration (CI):** Build and test your code automatically.
* **Continuous Delivery (CD):** Deploy your code to environments (Dev, QA, Production).

It supports multiple languages and platforms (Node.js, Python, .NET, Java, Docker, Kubernetes, etc.).

**Hierarchy from top to bottom**:

trigger → variables → pool → stages → jobs → steps → (task | script)

📑 **Azure DevOps YAML Pipeline Structure (CI/CD)**

| **Section** | **Purpose** |
| --- | --- |
| **trigger** | Define branch-based pipeline triggers (e.g., run on main or PRs). |
| **variables** | Declare global variables used across the pipeline (config values, secrets). |
| **pool** | Define the agent VM image (Microsoft-hosted or self-hosted) where jobs run. |
| **stages** | Top-level logical stages in your pipeline (e.g., *Build*, *Test*, *Deploy*). |
| **jobs** | A group of steps that run on a single agent inside a stage. |
| **steps** | Individual actions executed inside a job (scripts or tasks). |
| **task** | Predefined Azure DevOps action (e.g., DotNetCoreCLI, AzureWebApp, PublishBuildArtifacts). |
| **script** | Run custom shell/command-line scripts (e.g., npm install, dotnet build). |

**Example: Azure DevOps YAML Pipeline Hierarchy**

pipeline.yaml

├── trigger # Define when pipeline runs (branch, PR, schedule)

│

├── variables # Global variables and secrets

│

├── pool # Define agent VM (ubuntu, windows, self-hosted)

│

└── stages # Top-level pipeline stages (Build, Test, Deploy, etc.)

│

├── stage (Build) # One stage

│ │

│ └── jobs # Group of jobs inside stage

│ │

│ ├── job # A single job (runs on one agent)

│ │ │

│ │ └── steps # Ordered execution of steps

│ │ │

│ │ ├── task # Predefined Azure DevOps action

│ │ └── script # Custom shell command

│

└── stage (Deploy) # Another stage

│

└── jobs

└── deployment job # Special type of job for deployments

└── steps

├── task (AzureWebApp, HelmDeploy, etc.)

└── script

**🔹 Key Takeaways**

* **Pipeline** = overall YAML definition.
* **Stages** = logical divisions (e.g., CI = Build, CD = Deploy).
* **Jobs** = groups of steps, run in sequence or parallel.
* **Steps** = actual actions → either **task** (built-in) or **script** (custom).

**🛠 Hands-on: Your First Azure DevOps Pipeline**

**Step 1: Create a Project**

1. Go to Azure DevOps.
2. Create a **new project**.
3. Push some code into the repo (can be any repo — even just one .txt file).

**Step 2: Add a Pipeline**

1. In your project → go to **Pipelines** → **Create Pipeline**.
2. Choose **Azure Repos Git** (or GitHub if your code is there).
3. Select **YAML**.
4. Paste this YAML:

# azure-pipelines.yml

trigger:

- main # runs on pushes to main branch

pool:

vmImage: 'ubuntu-latest' # hosted Linux agent

stages:

- stage: Build

jobs:

- job: HelloWorldJob

steps:

- script: echo "Hello, Azure DevOps!" # simple script

displayName: "Say Hello"

- script: echo "Building project..." # pretend build step

displayName: "Build Project"

- task: PublishBuildArtifacts@1 # publish artifact

inputs:

PathtoPublish: .

ArtifactName: drop

**Step 3: Commit & Run**

1. Save the pipeline file as azure-pipelines.yml in your repo.
2. Commit → pipeline will run automatically.
3. Go to **Pipelines > Runs** and watch logs.

**Step 4: Verify Output**

* You’ll see logs:
  + Hello, Azure DevOps!
  + Building project...
* Check **Artifacts** tab → your “drop” artifact will be there.

**🛠 Step-by-Step: Classic Editor Pipeline (Hello World)**

**Step 1: Go to Pipelines**

1. Open your Azure DevOps project.
2. In the left menu, click **Pipelines → Pipelines**.
3. Click **New pipeline**.

**Step 2: Use Classic Editor**

1. At the bottom, click **“Use the classic editor”**.
   * This allows you to create pipelines with a GUI instead of YAML.

**Step 3: Select Source**

1. Choose **Azure Repos Git** (or GitHub if your code is there).
2. Select your repository.
3. Select the branch you want (e.g., main).
4. Click **Continue**.

**Step 4: Select Template**

1. Choose **Empty job** → we’ll add steps manually.
2. Click **Apply**.

**Step 5: Configure Agent Job**

1. Under **Agent job 1**:
   * Set **Agent pool**: Azure Pipelines.
   * Set **Agent specification**: ubuntu-latest (or windows-latest).

**Step 6: Add Tasks (Steps)**

1. Click **+** next to **Agent job 1** to add tasks.

**Task 1: Command Line Script**

* Display name: Say Hello
* Script:

echo "Hello, Azure DevOps!"

**Task 2: Command Line Script**

* Display name: Build Project
* Script:

echo "Building project..."

**Task 3: Publish Build Artifacts**

* Path to publish: $(Build.SourcesDirectory)
* Artifact name: drop

**Step 7: Save and Run**

1. Click **Save & Queue → Save & Run**.
2. Watch the pipeline execute.

✅ You should see:

* Hello, Azure DevOps! printed in logs.
* Building project... printed in logs.
* Artifacts published in **Artifacts tab**.

This setup gives you a **full CI pipeline** for a simple project using the **Classic Editor**.

**📑 Classic Pipeline Tasks Table (Hello World)**

| **Task** | **Display Name** | **Purpose** | **Script / Inputs** |
| --- | --- | --- | --- |
| Command Line Script | Say Hello | Print a message to verify pipeline is running | echo "Hello, Azure DevOps!" |
| Command Line Script | Build Project | Placeholder for build step | echo "Building project..." |
| Publish Build Artifacts | drop | Publish source code/artifacts for later use | Path: $(Build.SourcesDirectory) Artifact Name: drop |

**✅ Notes:**

* **Order matters**: Tasks execute top → bottom.
* **Command Line Script**: Used for custom commands.
* **Publish Build Artifacts**: Makes files available for **deployment** in later stages.