**DevOps pipeline:**

A **DevOps pipeline** is a set of automated processes and tools that allow development and operations teams to work together to build, test, and deploy software quickly and efficiently.

**Key Parts of a DevOps Pipeline:**

1. **Source Code Management**:
   * The process starts when developers commit their code to a source control system (like Git). The pipeline listens for changes in the code.
2. **Build**:
   * The pipeline automatically compiles and builds the code into an executable application (such as an app or website). This ensures that the code is free of errors and ready for testing.
3. **Test**:
   * After building the code, the pipeline runs automated tests (unit tests, integration tests, etc.) to ensure that the code works as expected and has no bugs.
4. **Deploy**:
   * If the code passes the tests, it’s automatically deployed to a staging environment where it can be tested in a more realistic setting. In some cases, it can also be deployed to production directly if all tests pass and quality is verified.
5. **Monitor**:
   * After deployment, the pipeline can also include monitoring tools that track the application's performance and catch any issues that may arise in production.

**Why is it Important?**

* **Automation**: DevOps pipelines automate repetitive tasks, saving time and reducing human errors.
* **Continuous Integration and Continuous Delivery (CI/CD)**: It ensures that code changes are integrated and deployed continuously, allowing faster releases and updates.
* **Collaboration**: DevOps pipelines bridge the gap between developers (who write code) and operations (who deploy and manage it), ensuring smooth collaboration between teams.
* **Speed and Quality**: With automation and testing, it allows teams to deliver high-quality software faster.

**Types of Pipeline:**

The difference between **Classic Pipeline** and **YAML Pipeline** is mainly in how they are defined and configured. Here's a simple explanation:

**1. Classic Pipeline:**

* **User Interface (UI) Based**: You create and configure the pipeline using a graphical interface. No coding is required.
* **Visual Configuration**: You click through steps like tasks, stages, variables, and triggers using a drag-and-drop interface.
* **More Manual**: You may need to manually set up each task and configuration.
* **Easy for Beginners**: Best for users who prefer visual interaction rather than writing code.

**Example**: You use a wizard in the Azure DevOps UI to define your pipeline steps, like build, test, and deploy.

**2. YAML Pipeline:**

* **Code-Based Configuration**: You define the pipeline using YAML (a markup language) code.
* **Text File**: The configuration is written in a .yml or .yaml file, which can be versioned and stored with the code repository.
* **Version Control**: Since it's a file, changes to the pipeline are tracked in version control (like Git).
* **More Flexibility**: Provides more control and customization, but requires knowledge of YAML syntax.

**Example**: You write a YAML file that includes build steps, triggers, and environments.

### Key Differences:

* **UI vs Code**: Classic pipeline is built with a GUI, while YAML pipeline uses code.
* **Flexibility**: YAML offers more flexibility, while Classic is simpler to set up visually.
* **Versioning**: YAML pipelines are stored as code (versioned in Git), whereas Classic pipelines are not.

Creating a YAML pipeline in Azure DevOps is a simple process once you understand the basic structure. Here’s an easy step-by-step guide to the syntax of a YAML pipeline:

### 1. ****Basic Structure of YAML Pipeline****

At its core, a YAML pipeline consists of three main sections:

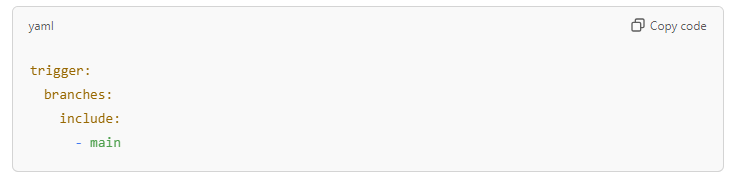
* **trigger**: Defines when the pipeline should run.
* **jobs**: Specifies what tasks or steps need to be executed.
* **steps**: Defines each individual task in the job.

**Here's a basic example:**



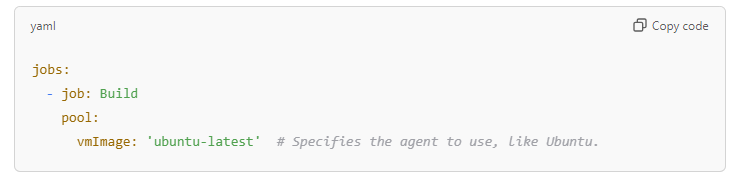
### Step-by-Step Breakdown:

#### 1. **Trigger**

The trigger section specifies when the pipeline should run. In this example, the pipeline triggers on changes to the main branch.

#### 2. **Jobs**

A pipeline consists of **jobs**, which run on agents (machines). Here, we define a single job named Build.



#### 3. **Steps**

Each **job** contains multiple **steps**. Steps define tasks to execute, such as running scripts or using pre-built tasks (e.g., using Node.js).

* **task**: Uses predefined tasks like installing Node.js.
* **script**: Runs custom shell commands (e.g., installing dependencies or building the project).

### 4. ****Other Optional Sections****:

* **variables**: Define custom variables.
* **stages**: Split the pipeline into multiple stages (e.g., Build, Test, Deploy).
* **condition**: Control whether a step or job runs under certain conditions.

**Example with Multiple Stages and Variables:**

trigger:

branches:

include:

- main

variables:

nodeVersion: '14.x'

stages:

- stage: Build

jobs:

- job: Build

pool:

vmImage: 'ubuntu-latest'

steps:

- task: UseNode@2

inputs:

versionSpec: $(nodeVersion)

displayName: 'Use Node.js $(nodeVersion)'

- script: |

npm install

npm run build

displayName: 'Install dependencies and build'

- stage: Deploy

jobs:

- job: Deploy

steps:

- script: echo "Deploying application"

displayName: 'Deploy'

### Summary of YAML Pipeline Syntax:

1. **trigger**: Defines when to run the pipeline.
2. **jobs**: Defines tasks to run (can have multiple jobs).
3. **steps**: Defines the individual tasks/commands within a job.
4. **stages** (optional): Groups jobs into stages.
5. **variables** (optional): Defines reusable variables.

This basic syntax will help you create a simple YAML pipeline to build, test, or deploy your application.

**Main Concepts in a YAML Pipeline:**

**Stages:** The pipeline is split into stages. Each stage represents a distinct phase in the workflow (e.g., build, test, deploy). Stages are executed one after another, in order.

**Jobs:** Each stage contains one or more jobs. Jobs define specific tasks that need to be done in that stage.

**Steps:** A step is the smallest unit of work in the pipeline. It represents an individual action or task, like compiling code or running tests.

**Tasks:** A task is the actual operation that the pipeline executes. This could be a predefined task (like running tests) or a custom script written by the user.

**What is pool in pipeline:**

In pipeline, a **pool** refers to a collection of **agents** (virtual or physical machines) that are used to run the tasks in the pipeline.

### Key Points about Pool in a Pipeline:

1. **Agents**:
   * An **agent** is a machine (either a virtual machine or a physical machine) that executes the tasks defined in the pipeline. The agent runs the build, test, and deployment steps.
2. **Pool**:
   * A **pool** is essentially a group or collection of these agents. The pool is used to manage where your pipeline’s tasks will run. You can specify which pool of agents should be used for a particular job in your pipeline.

**Why Use a Pool?**

* It allows you to define and manage the resources (agents) where your tasks will run. You can specify which pool of agents should run a certain part of the pipeline, making the pipeline flexible and efficient.
* You can have multiple pools for different types of workloads, such as one pool for building applications, another for deploying, and so on.

### Types of Pools:

1. **Microsoft-Hosted Pools**:
   * These are managed by Azure DevOps. They provide pre-configured virtual machines with popular software already installed (like Ubuntu, Windows, etc.).
   * Example: vmImage: 'ubuntu-latest' — Azure DevOps automatically provides a machine running the latest Ubuntu image.
2. **Self-Hosted Pools**:
   * These are pools that you configure yourself, either in your on-premises environment or in the cloud. You can install the necessary software and tools that your pipeline needs on these agents.
   * You have full control over the machines, including custom configurations.

**Example in YAML Pipeline:**

