**YAML Pipeline Hierarchy**

YAML (Yet Another Markup Language) is a human-readable data serialization format often used for configuration files, including in continuous integration/continuous deployment (CI/CD) pipelines. In the context of pipelines, YAML files define how the automation should run, the steps involved, and the conditions under which tasks should be executed.

The **YAML pipeline hierarchy** refers to how data is organized and structured in a pipeline YAML configuration. The hierarchy is built using indentation, nesting, and key-value pairs. Here is a detailed explanation of the various components and how they form the pipeline structure:

**Basic Structure of a YAML Pipeline**

At the highest level, the YAML pipeline file contains a series of keys and values that define the pipeline, stages, jobs, and steps. The hierarchy can be broken down as follows:

1. **Pipeline**: The outermost level defines the overall pipeline structure, which typically includes a name, trigger, and other global settings.
2. **Stages**: These are broad steps in the pipeline that can contain multiple jobs.
3. **Jobs**: Jobs are a series of steps that are executed on a specific agent (machine or container).
4. **Steps**: The smallest unit, steps define the actual tasks to execute within a job.

Let's break down these components:

**1. Pipeline (Top-Level)**

The pipeline is the outermost key in the YAML file. It might contain global settings, triggers, or environment configurations. Example:

yaml

name: ExamplePipeline

trigger:

branches:

include:

- main

- develop

* **name**: Defines the name of the pipeline.
* **trigger**: Specifies which branches trigger the pipeline (e.g., main, develop).

**2. Stages**

Stages are used to organize jobs logically. Stages help you divide your pipeline into phases (e.g., build, test, deploy). Each stage can contain multiple jobs, and jobs within a stage are executed sequentially.

yaml

stages:

- stage: Build

jobs:

- job: BuildJob

steps:

- script: echo "Building project..."

- stage: Deploy

jobs:

- job: DeployJob

steps:

- script: echo "Deploying project..."

* **stages**: A list of stages within the pipeline.
* **stage**: Each individual stage name (e.g., "Build", "Deploy").
* **jobs**: List of jobs that belong to the stage.

**3. Jobs**

A **job** is a unit of work that runs within a stage. Jobs can execute in parallel or sequentially, depending on the pipeline configuration. Each job can run on a different agent or environment.

yaml

jobs:

- job: BuildJob

pool:

vmImage: 'ubuntu-latest'

steps:

- task: UseNode@1

inputs:

versionSpec: '16.x'

- script: npm install

- job: TestJob

pool:

vmImage: 'ubuntu-latest'

steps:

- script: npm test

* **job**: Defines the name of the job.
* **pool**: Specifies the agent pool to run the job (e.g., vmImage for Azure DevOps).
* **steps**: A list of tasks or scripts to execute within the job.

**4. Steps**

A **step** represents a single task that is executed within a job. Steps can be commands, scripts, or predefined tasks. Each step can perform actions like running a script, building code, testing, or deploying.

yaml

steps:

- task: UseDotNet@2

inputs:

packageType: 'sdk'

version: '6.x'

- script: |

echo "Running tests"

dotnet test

* **script**: Run a command in the shell (bash, PowerShell, etc.).
* **task**: Use a predefined task (such as UseDotNet, PublishBuildArtifacts, etc.).

**Example of a Full Pipeline**

Here’s a full example of a simple YAML pipeline for a build and test scenario:

yaml

name: BuildAndTestPipeline

trigger:

branches:

include:

- main

- feature/\*

stages:

- stage: Build

jobs:

- job: BuildJob

pool:

vmImage: 'ubuntu-latest'

steps:

- task: UseNode@1

inputs:

versionSpec: '16.x'

- script: npm install

- stage: Test

jobs:

- job: TestJob

pool:

vmImage: 'ubuntu-latest'

steps:

- script: npm test

**Key Points of YAML Pipeline Hierarchy**

1. **Indentation**: YAML uses spaces for indentation to represent nested structures. Indentation must be consistent (typically 2 spaces per level).
2. **Key-Value Pairs**: YAML files are built using key-value pairs, where the key is followed by a colon and the value is the corresponding data.
3. **Lists**: Lists are represented by dashes (-) followed by the list item. For example, a list of steps or jobs.
4. **Nesting**: To represent hierarchical relationships, YAML uses indentation. For example, jobs are nested inside stages, and steps are nested inside jobs.

**Advanced Concepts in YAML Pipelines**

1. **Variables**: Pipelines often use variables to pass data dynamically between stages, jobs, and steps.

yaml

variables:

buildConfiguration: 'Release'

1. **Conditionals**: You can use conditions to determine whether a step or job should run.

yaml

steps:

- script: echo "This will run only on the main branch"

condition: eq(variables['Build.SourceBranchName'], 'main')

1. **Matrix**: Matrix allows you to run jobs in parallel with different configurations.

yaml

jobs:

- job: TestJob

strategy:

matrix:

Ubuntu:

imageName: 'ubuntu-latest'

Windows:

imageName: 'windows-latest'

steps:

- script: echo "Running on ${{ matrix.imageName }}"

1. **Templates**: YAML pipeline templates allow reuse of common pipeline sections (e.g., stages, jobs).

yaml

stages:

- template: build-template.yml

In the above example, a build-template.yml file could define reusable stages, jobs, or steps, which simplifies managing large pipelines.

**Conclusion**

The YAML pipeline hierarchy is a powerful and flexible way to define CI/CD workflows. It allows you to organize tasks, define dependencies, and manage automation efficiently. By structuring your pipeline in stages, jobs, and steps, you can build complex and maintainable workflows that run across various environments with different configurations.