DevOps

Week 05

Murtaza Munawar Fazal

YAML – Hello World

```yaml
name: 1.0$(Rev:.r)

# simplified trigger (implied branch)
trigger:

- main

# equivalents trigger
# trigger:
#  branches:
#    include:
#    - main

variables:
  name: John

pool:
  vmImage: ubuntu-latest

jobs:

- job: helloworld
  steps:
    - checkout: self
    - script: echo "Hello, $(name)"
```

Name

- The variable name is a bit misleading since the name is in the build number format. You'll get an integer number if you don't explicitly set a name format. A monotonically increasing number of runs triggered off this pipeline, starting at 1. This number is stored in Azure DevOps. You can make use of this number by referencing $(Rev).
- To make a date-based number, you can use the format $(Date:yyyyMMdd) to get a build number like 20221003.
- To get a semantic number like 1.0.x, you can use something like 1.0.$(Rev:.r).

Triggers

- If there's no explicit triggers section, then it's implied that any commit to any path in any branch will trigger this pipeline to run.
- However, you can be more precise by using filters such as branches or paths.
- Let's consider this trigger:

```
trigger:
  branches:
    include:


    - main
```

Exclude Branch from Trigger

```
trigger:
  branches:
    exclude:

    - main
```

Multiple Branches

```
trigger:
  branches:
    include:
     - feature/*
  paths:
    include:
     - webapp/**
```

Jobs

- A job is a set of steps executed by an agent in a queue (or pool). Jobs are atomic – they're performed wholly on a single agent. You can configure the same job to run on multiple agents simultaneously, but even in this case, the entire set of steps in the job is run on every agent. You'll need two jobs if you need some steps to run on one agent and some on another.
- A job has the following attributes besides its name:
  - displayName – a friendly name.
  - dependsOn - a way to specify dependencies and ordering of multiple jobs.
  - condition – a binary expression: if it evaluates to true, the job runs; if false, the job is skipped.
  - strategy - used to control how jobs are parallelized.
  - continueOnError - to specify if the rest of the pipeline should continue or not if this job fails.
  - pool – the name of the pool (queue) to run this job on.
  - workspace - managing the source workspace.
  - container - for specifying a container image to execute the job later.
  - variables – variables scoped to this job.
  - steps – the set of steps to execute.
  - timeoutInMinutes and cancelTimeoutInMinutes for controlling timeouts.
  - services - sidecar services that you can spin up.

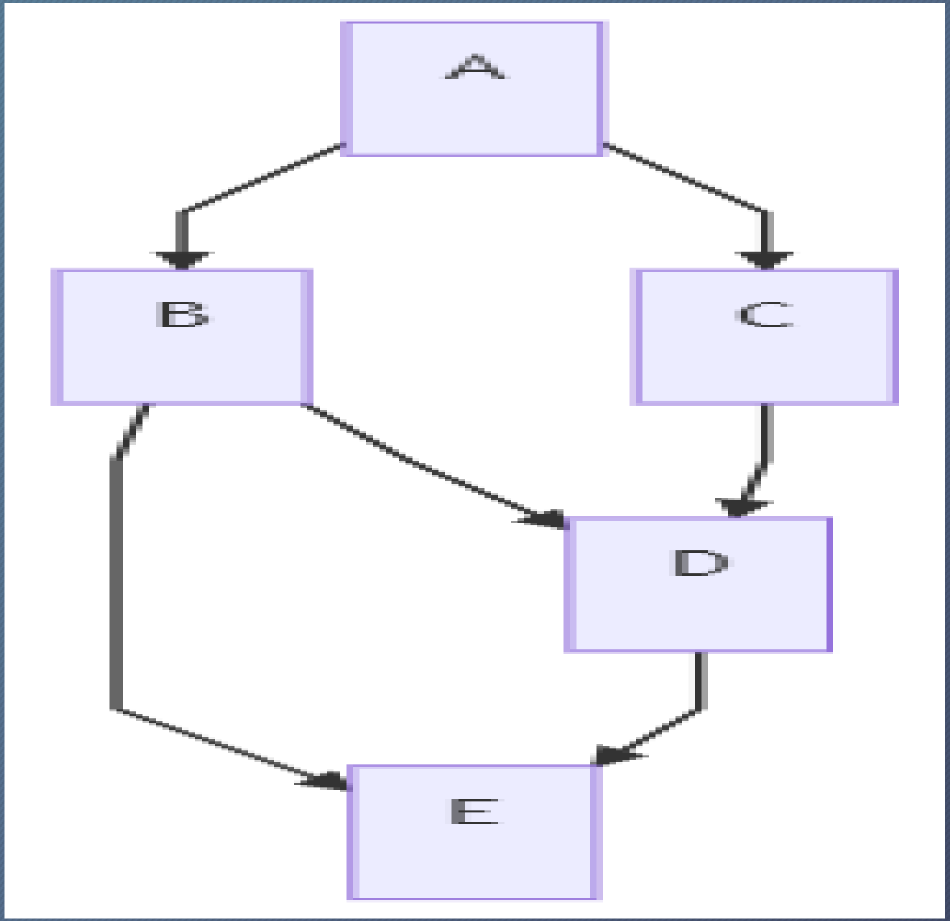
Dependent Jobs ?

```
jobs:

- job: A
  steps:
  # steps omitted for brevity


- job: B
  steps:
  # steps omitted for brevity
```

Dependent Job

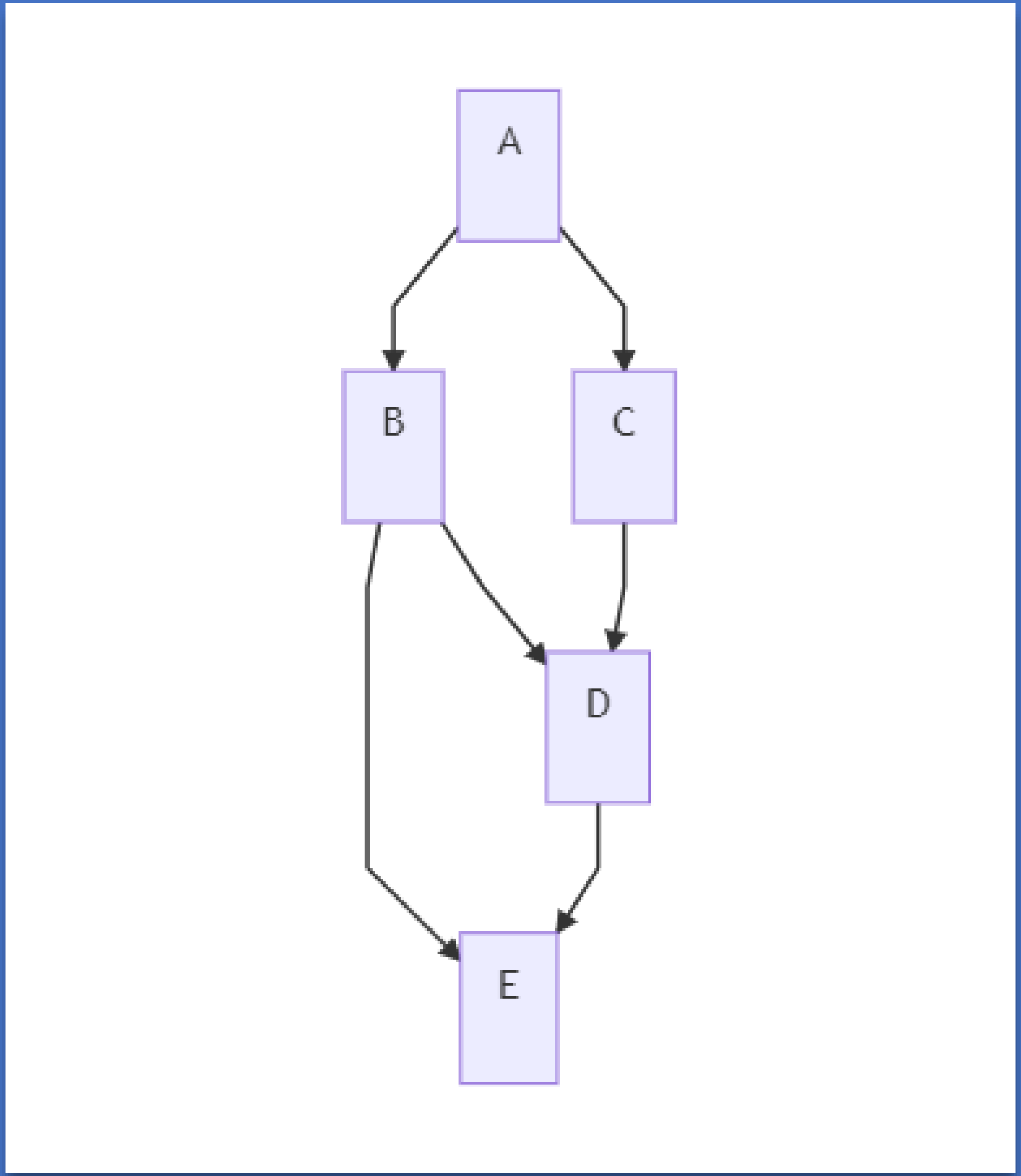
```
- job: A
  steps:
  # steps omitted for brevity


- job: B
  dependsOn: [] # this removes the implicit dependency on previous stage 
and causes this to run in parallel
  steps:
  # steps omitted for brevity
```

Dependent Job



Dependent Job

```
jobs:
- job: A
  steps:
  - script: echo' job A.'

- job: B
  dependsOn: A
  steps:
  - script: echo' job B.'

- job: C
  dependsOn: A
  steps:
  - script: echo' job C.'

- job: D
  dependsOn:
  - B
  - C
  steps:
  - script: echo' job D.'

- job: E
  dependsOn:
  - B
  - D
  steps:
  - script: echo' job E.'
```



Steps are Task

- Steps are the actual "things" that execute in the order specified in the job.
- Each step is a task: out-of-the-box (OOB) tasks come with Azure DevOps. Many have aliases and tasks installed on your Azure DevOps organization via the marketplace.

Understand the Pipeline Structure

- A pipeline is one or more stages that describe a CI/CD process.
- Stages are the primary divisions in a pipeline. The stages "Build this app," "Run these tests," and "Deploy to preproduction" are good examples.
- A stage is one or more jobs, units of work assignable to the same machine.
- You can arrange both stages and jobs into dependency graphs. Examples include "Run this stage before that one" and "This job depends on the output of that job."
- A job is a linear series of steps. Steps can be tasks, scripts, or references to external templates.

Understand the Pipeline Structure

- This hierarchy is reflected in the structure of a YAML file like:
- Pipeline
    - Stage A
        - Job 1
            - Step 1.1
            - Step 1.2
            - ...
        - Job 2
            - Step 2.1
            - Step 2.2
            - ...
    - Stage B
    - ...
- Simple pipelines don't require all these levels. For example, you can omit the containers for stages and jobs in a single job build because there are only steps.

Pipeline

- The schema for a pipeline:

- name: string # build numbering format
- resources:
  - pipelines: [pipelineResource]
  - containers: [containerResource]
  - repositories: [repositoryResource]
- variables: # several syntaxes
- trigger: trigger
- pr: pr
- stages: [stage | templateReference]

Pipeline

- If you have a single-stage, you can omit the stages keyword and directly specify the jobs keyword:
- jobs: [job | templateReference]

- If you've a single-stage and a single job, you can omit the stages and jobs keywords and directly specify the steps keyword:
- steps: [script | bash | pwsh | powershell | checkout | task | templateReference]

Pipeline - Stage

- A stage is a collection of related jobs. By default, stages run sequentially. Each stage starts only after the preceding stage is complete.
- Use approval checks to control when a stage should run manually. These checks are commonly used to control deployments to production environments.
- Checks are a mechanism available to the resource owner. They control when a stage in a pipeline consumes a resource.
- As an owner of a resource like an environment, you can define checks required before a stage that consumes the resource can start.
- This example runs three stages, one after another. The middle stage runs two jobs in parallel.

Pipeline - Stage

- stages:
  - stage: Build
    - jobs:
      - job: BuildJob
        - steps:
          - script: echo Building!
  - stage: Test
    - jobs:
      - job: TestOnWindows
        - steps:
          - script: echo Testing on Windows!
      - job: TestOnLinux
        - steps:
          - script: echo Testing on Linux!
  - stage: Deploy
    - jobs:
      - job: Deploy
        - steps:
          - script: echo Deploying the code!

Pipeline - Job

- A job is a collection of steps run by an agent or on a server. Jobs can run conditionally and might depend on previous jobs.

- jobs:
  - job: MyJob
    - displayName: My First Job
    - continueOnError: true
    - workspace:
      - clean: outputs
    - steps:
      - script: echo My first job

Pipeline - Task

- Tasks are the building blocks of a pipeline. There's a catalog of tasks available to choose from.

- steps:
  - task: VSBuild@1
    - displayName: Build
    - timeoutInMinutes: 120
    - inputs:
      - solution: '***.sln'

YAML - Resource

- Resources in YAML represent sources of pipelines, repositories, and containers.

- General schema:

- resources:
  - pipelines: [pipeline]
  - repositories: [repository]
  - containers: [container]

YAML - Pipeline Resource

- If you have an Azure pipeline that produces artifacts, your pipeline can consume the artifacts by using the pipeline keyword to define a pipeline resource.
- resources:
  - pipelines:
  - pipeline: MyAppA
  - source: MyCIPipelineA
  - pipeline: MyAppB
  - source: MyCIPipelineB
  - trigger: true
  - pipeline: MyAppC
  - project: DevOpsProject
  - source: MyCIPipelineC
  - branch: releases/M159
  - version: 20190718.2
  - trigger:
    - branches:
      - include:
        - master
        - releases/*
      - exclude:
        - users/*

YAML - Container Resource

- Container jobs let you isolate your tools and dependencies inside a container. The agent launches an instance of your specified container then runs steps inside it. The container keyword lets you specify your container images.
- Service containers run alongside a job to provide various dependencies like databases.

- resources:
- containers:
  - container: linux
    - image: ubuntu:16.04
  - container: windows
    - image: myprivate.azurecr.io/windowsservercore:1803
    - endpoint: my_acr_connection
  - container: my_service
    - image: my_service:tag
    - ports:
      - 8080:80 # bind container port 80 to 8080 on the host machine
      - 6379 # bind container port 6379 to a random available port on the host machine
    - volumes:
      - /src/dir:/dst/dir # mount /src/dir on the host into /dst/dir in the container

YAML - Repository Resource

- Let the system know about the repository if:
  - If your pipeline has templates in another repository.
  - If you want to use multi-repo checkout with a repository that requires a service connection.
- The repository keyword lets you specify an external repository.

- resources:
  - repositories:
    - repository: common
      - type: github
      - name: Contoso/CommonTools
      - endpoint: MyContosoServiceConnection

YAML - Repository Resource

- Multiple Repositories
- resources:
    - repositories:
        - repository: MyGitHubRepo # The name used to reference this repository in the checkout step.
            - type: github
            - endpoint: MyGitHubServiceConnection
            - name: MyGitHubOrgOrUser/MyGitHubRepo
        - repository: MyBitBucketRepo
            - type: bitbucket
            - endpoint: MyBitBucketServiceConnection
            - name: MyBitBucketOrgOrUser/MyBitBucketRepo
        - repository: MyAzureReposGitRepository
            - type: git
            - name: MyProject/MyAzureReposGitRepo
    - trigger:
        - main
    - pool:
        - vmImage: 'ubuntu-latest'
    - steps:
        - checkout: self
        - checkout: MyGitHubRepo
        - checkout: MyBitBucketRepo
        - checkout: MyAzureReposGitRepository
        - script: dir $(Build.SourcesDirectory)