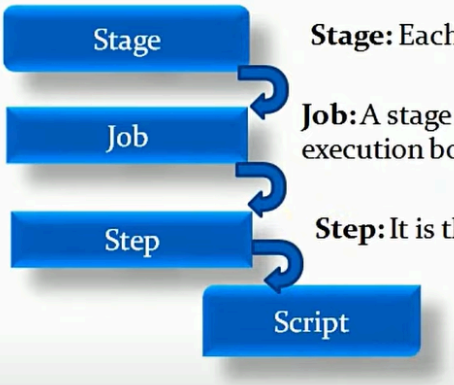
Pipeline:
It is a workflow that defines how our test, build, and deployment steps are run.

Azure Pipelines: Develop, test and deploy CI/CD workflows that are compatible with any language, platform, and cloud. Automating builds and deployments with pipelines can save time, giving scope for innovation

Azure DevOps Pipeline key concepts

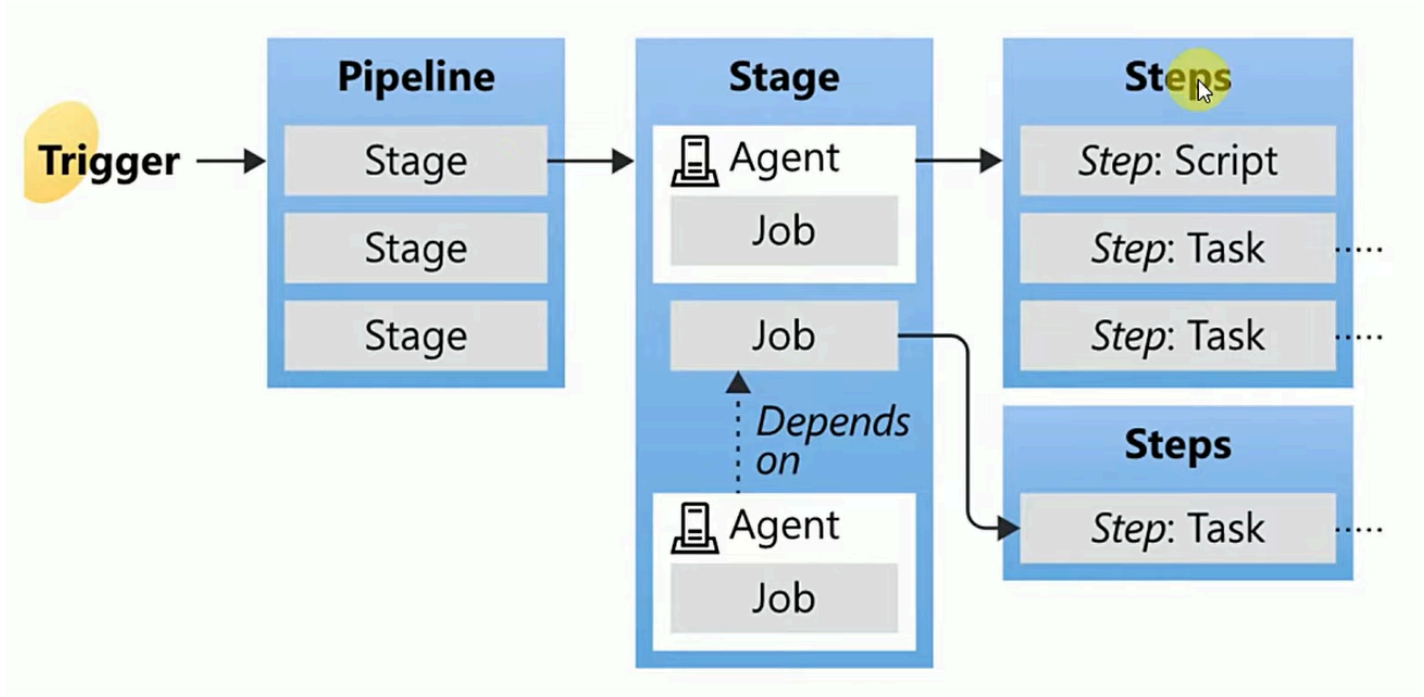


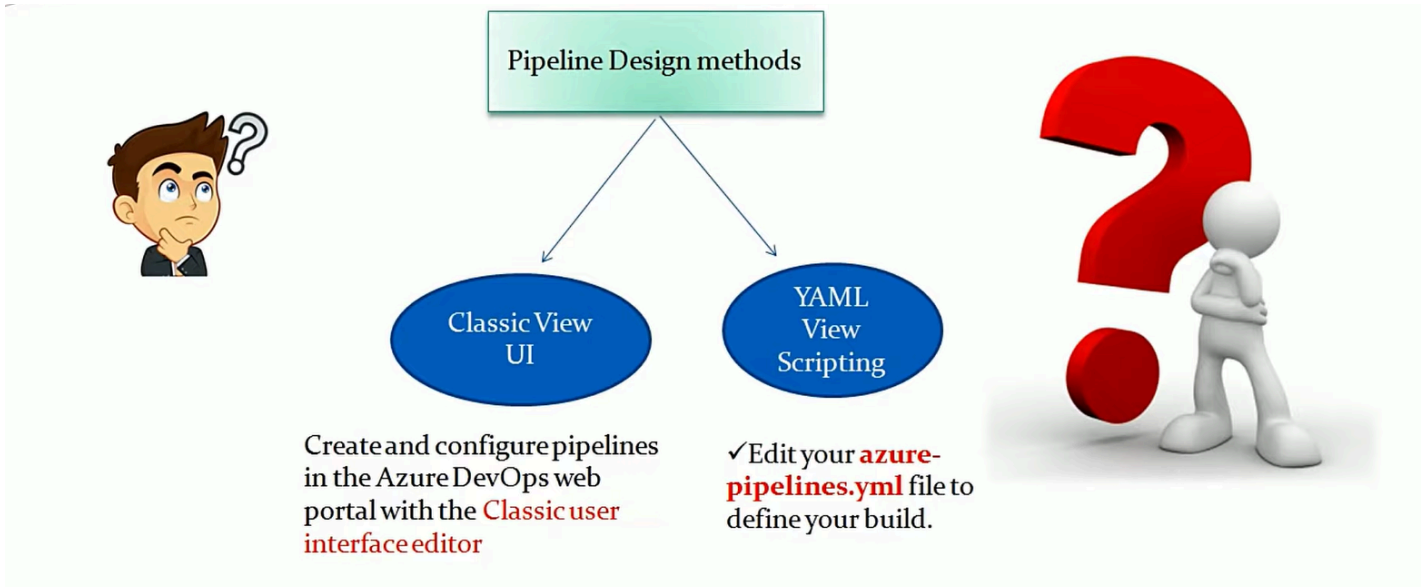
Stage: Each stage contains one or more jobs.

Job: A stage can contain one or more jobs. Each job runs on an agent. It represents an execution boundary of a set of steps.

Step: It is the smallest building block of a pipeline.

Agent and Agent pools: An agent is an installable software that runs one job at a time.


Trigger
Pipeline
Stage
Stage
Stage
Stage
Agent
Job
Job
Depends on
Agent
Job
Steps
Step: Script
Step: Task
Step: Task
Steps
Step: Task


Pipeline Design methods
Classic View UI
YAML View Scripting
Create and configure pipelines in the Azure DevOps web portal with the Classic user interface editor
✓Edit your azure-pipelines.yml file to define your build.

YAML Pipeline Design

YAML Basic understanding

Define build and release pipelines using YAML (YAML Ain't Markup Language)

YAML build definitions can be added to a project by simply adding their source file .yaml to the root of the repository

YAML is a human-readable language. It is commonly used for configuration files.

1. YAML is case sensitive language
2. YAML files have .yaml as the extension
3. YAML spaces are allowed

Basic about YAML

Block format uses **hyphen+space** to begin a new item in a specified list

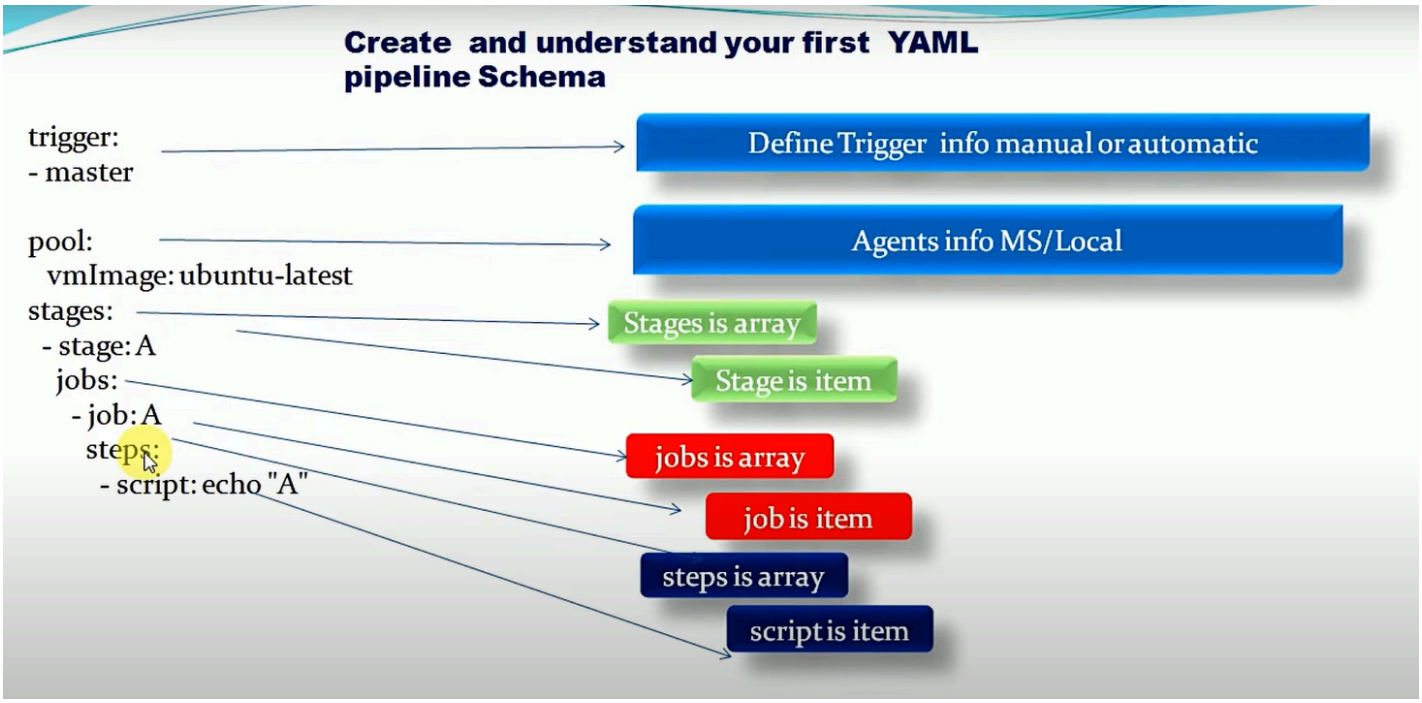
- India
- US
- UK

4. Comments in YAML begins with the (#) character.

5. Associative arrays are represented using colon (:)

6. Block sequences in collections indicate each entry with a **dash and space** (-).

Create and understand your first YAML pipeline Schema



Review your pipeline YAML

Variables | Save and run

BDDRepo / azure-pipelines.yml * | Show assistant

```yaml
trigger:
- master

pool:
  vmImage: ubuntu-latest

stages:
  - stage: A
    jobs:
      - job: Ajob
        steps:
          - script: echo "Ascrpti"
```

What are stages and how to add

Stages are the major divisions in a pipeline: "build ", "run tests", and "deploy to pre-production" are good examples of stages.

Every pipeline has at least one stage even if you do not explicitly define it.

You can also arrange stages into a dependency graph so that one stage runs before another one. There is a limit of **256 jobs for a stage**.

You can directly specify the jobs in your YAML file.

YAML

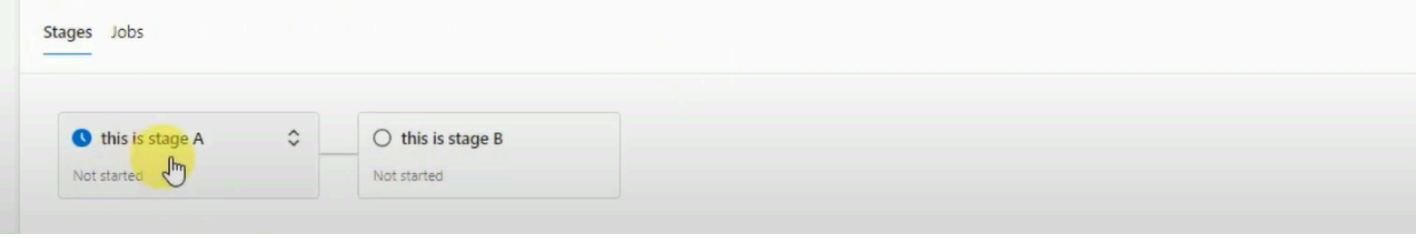
```yaml
# this has one implicit stage and one implicit job
pool:
  vmImage: 'ubuntu-latest'
steps:
- bash: echo "Hello world"
```

The full syntax to specify a stage is:

YAML

```yaml
stages:
- stage: string  # name of the stage, A-Z, a-z, 0-9, and underscore
  displayName: string  # friendly name to display in the UI
  dependsOn: string | [ string ]
  condition: string
  pool: string | pool
  variables: { string: string } | [ variable | variableReference ]
  jobs: [ job | templateReference]
```

```yaml
pool:
  vmImage: ubuntu-latest

stages:
  - stage: stageA
    displayName: "this is stage A"
    jobs:
      - job: jobA
  - stage: stageB
    displayName: "this is stage B"
    jobs:
      - job: jobB
```



How to define Multiple stages

Your pipeline may have multiple stages, each with multiple jobs. If you organize your pipeline into multiple stages, you use the stages keyword.

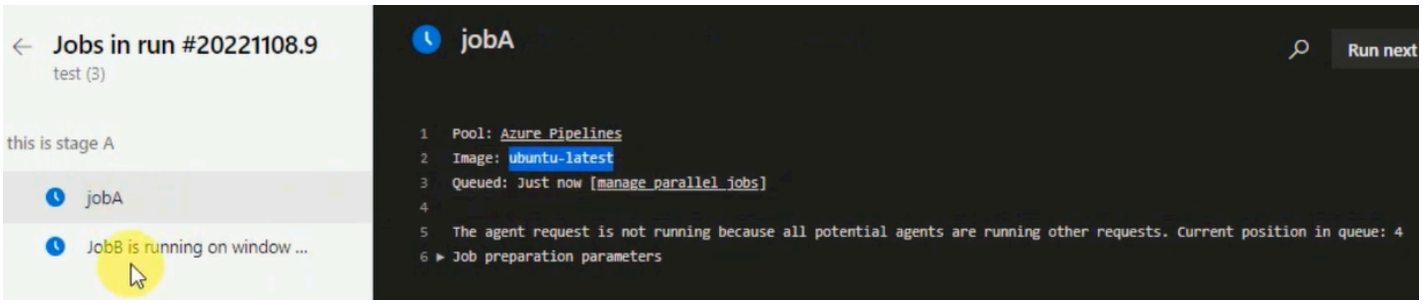
```yaml
stages:
- stage: A
  jobs:
  - job: A1
  - job: A2

- stage: B
  jobs:
  - job: B1
  - job: B2
```

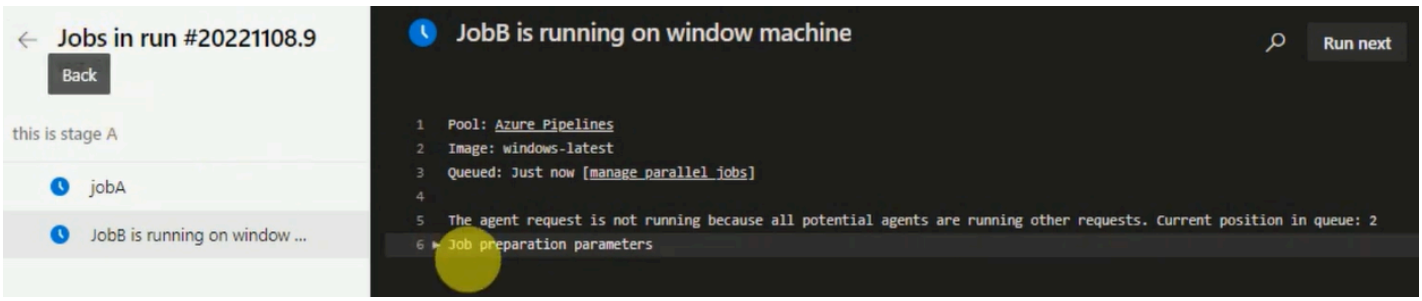
If you choose to specify a pool at the stage level, then all jobs defined in that stage will use that pool unless otherwise specified at the job-level.

```yaml
stages:
- stage: A
  pool: StageAPool
  jobs:
  - job: A1 # will run on "StageAPool" pool based on the pool defined
  - job: A2 # will run on "JobPool" pool
    pool: JobPool
```

```yaml

pool:
  vmImage: ubuntu-latest

stages:
  - stage: stageA
    displayName: "this is stage A"
    jobs:
      - job: jobA

  - stage: stageB
    displayName: "this is stage B"
    jobs:
      - job: jobB

  - stage: stageC
    displayName: "this is stage C"
    jobs:
      - job: jobC

  - stage: stageD
    displayName: "this is stage D"
    jobs:
      - job: jobD


```

```yaml
stages:
  - stage: stageA
    pool:
      vmImage: ubuntu-latest
    displayName: "this is stage A"
    jobs:
      - job: jobA
      - job: JobB
        displayName: "JobB is running on window machine"
        pool:
          vmImage: windows-latest
```





How to run stages and jobs parallel

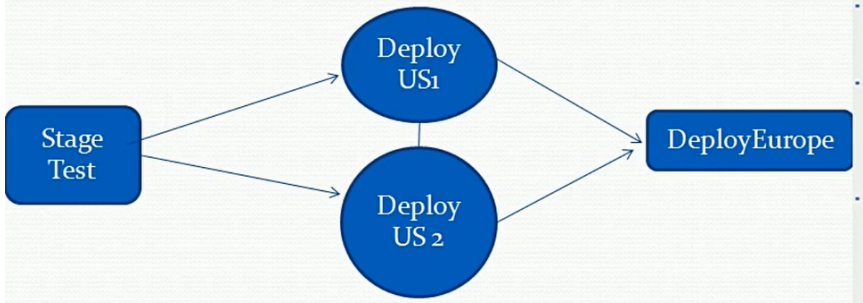
Example stages that run parallel:

Method 1- with dependsOn

YAML Copy

```yaml
stages:
- stage: FunctionalTest
  jobs:
  - job:
    ...

- stage: AcceptanceTest
  dependsOn: []    # this removes the implicit dependency on previous stage
  jobs:
  - job:
    ...
```

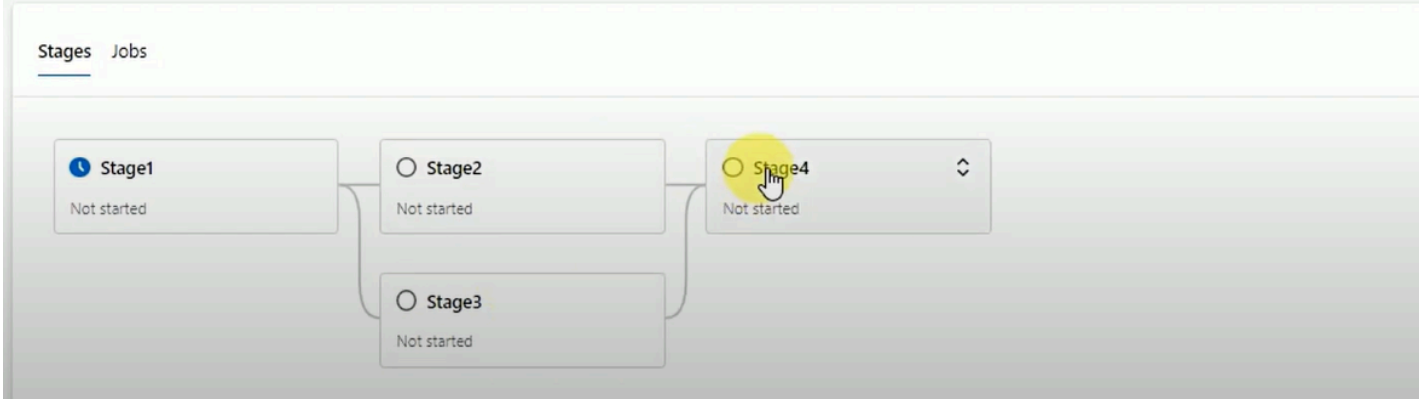
Example 2



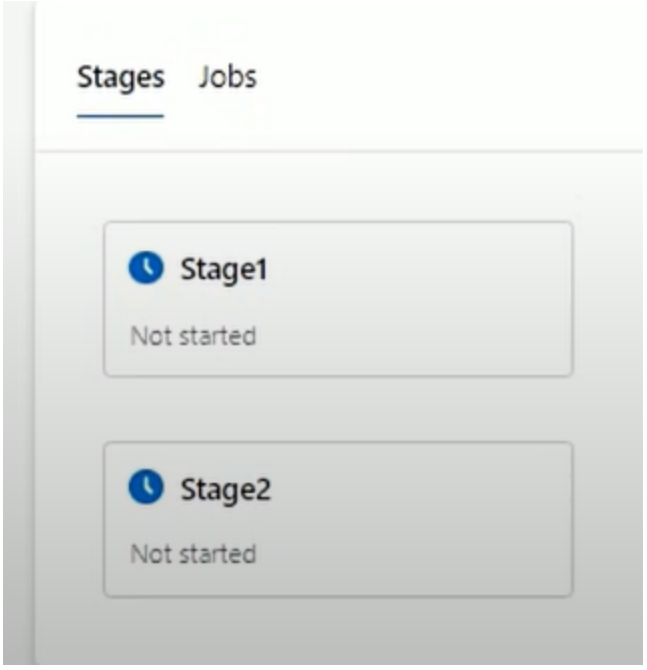
YAML Copy

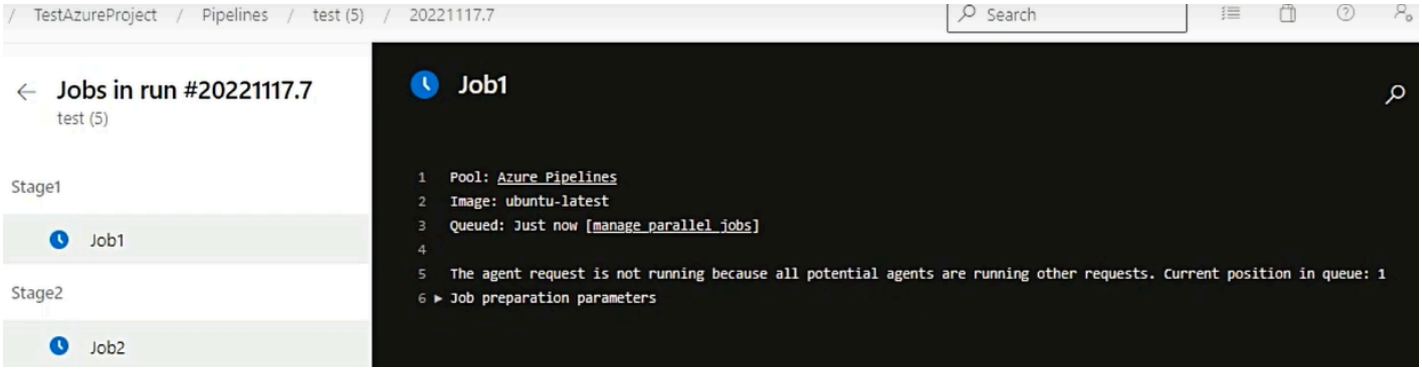
```yaml
stages:
- stage: Test

- stage: DeployUS1
  dependsOn: Test    # this stage runs after Test

- stage: DeployUS2
  dependsOn: Test    # this stage runs in parallel with DeployUS1, after Test

- stage: DeployEurope
  dependsOn:         # this stage runs after DeployUS1 and DeployUS2
  - DeployUS1
  - DeployUS2
```

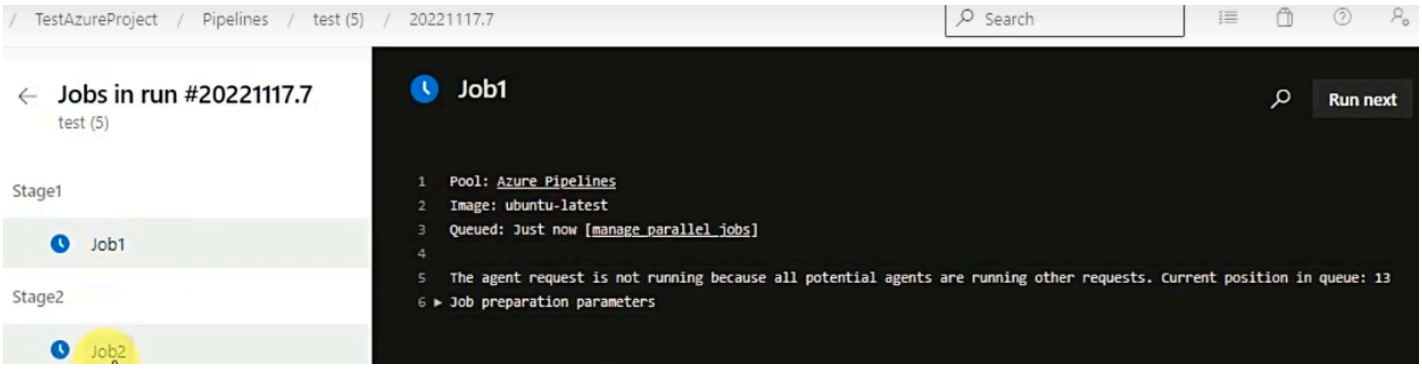
```yaml
pool:
  vmImage: ubuntu-latest

stages:
  - stage: Stage1
    jobs:
      - job: Job1

  - stage: Stage2
    dependsOn: Stage1
    jobs:
      - job: Job2

  - stage: Stage3
    dependsOn: Stage1
    jobs:
      - job: Job3

  - stage: Stage4
    dependsOn:
      - Stage2
      - Stage3
    jobs:
      - job: Job4
```




```yaml
stages:
  - stage: Stage1
    pool:
      vmImage: ubuntu-latest
    jobs:
      - job: Job1

  - stage: Stage2
    dependsOn: []
    pool:
      vmImage: windows-latest
    jobs:
      - job: Job2
```


Stages
Jobs
Stage1
Not started
Stage2
Not started


TestAzureProject / Pipelines / test (5) / 20221117.7
Search
Jobs in run #20221117.7
test (5)
Stage1
Job1
Stage2
Job2
Job1
1 Pool: Azure Pipelines
2 Image: ubuntu-latest
3 Queued: Just now [manage parallel jobs]
4
5 The agent request is not running because all potential agents are running other requests. Current position in queue: 1
6 ▶ Job preparation parameters


TestAzureProject / Pipelines / test (5) / 20221117.7
Search
Jobs in run #20221117.7
test (5)
Stage1
Job1
Stage2
Job2
Job1
Run next
1 Pool: Azure Pipelines
2 Image: ubuntu-latest
3 Queued: Just now [manage parallel jobs]
4
5 The agent request is not running because all potential agents are running other requests. Current position in queue: 13
6 ▶ Job preparation parameters

How to add Conditions and dependency between stages

You can customize this behaviour by forcing a stage to run even if a previous stage fails or by specifying a custom condition.

You can specify the conditions under which each stage runs.

Example to run a stage based upon the status of running a previous stage:

YAML

```yaml
stages:
- stage: A

# stage B runs if A fails
- stage: B
  condition: failed()

# stage C runs if B succeeds
- stage: C
  dependsOn:
  - A
  - B
  condition: succeeded('B')
```

Job and Types of jobs

Jobs can be of different types, depending on where they run.

Agent pool jobs run on an agent in an agent pool.

Server jobs run on the Azure DevOps Server.

Container jobs run in a container on an agent in an agent pool.

DEFINITION

A job is a series of steps that run sequentially as a unit. In other words, a job is the smallest unit of work that can be scheduled to run.

This YAML file has a job that runs on a Microsoft-hosted agent and outputs Hello world.

YAML

```yaml
pool:
  vmImage: 'ubuntu-latest'
steps:
- bash: echo "Hello world"
```

Pipeline Single Jobs

A pipeline has a single job

You may want to specify additional properties on that job. In that case, you can use the job keyword.

```yaml
jobs:
- job: myJob
  timeoutInMinutes: 10
  pool:
    vmImage: 'ubuntu-latest'
  steps:
  - bash: echo "Hello world"
```

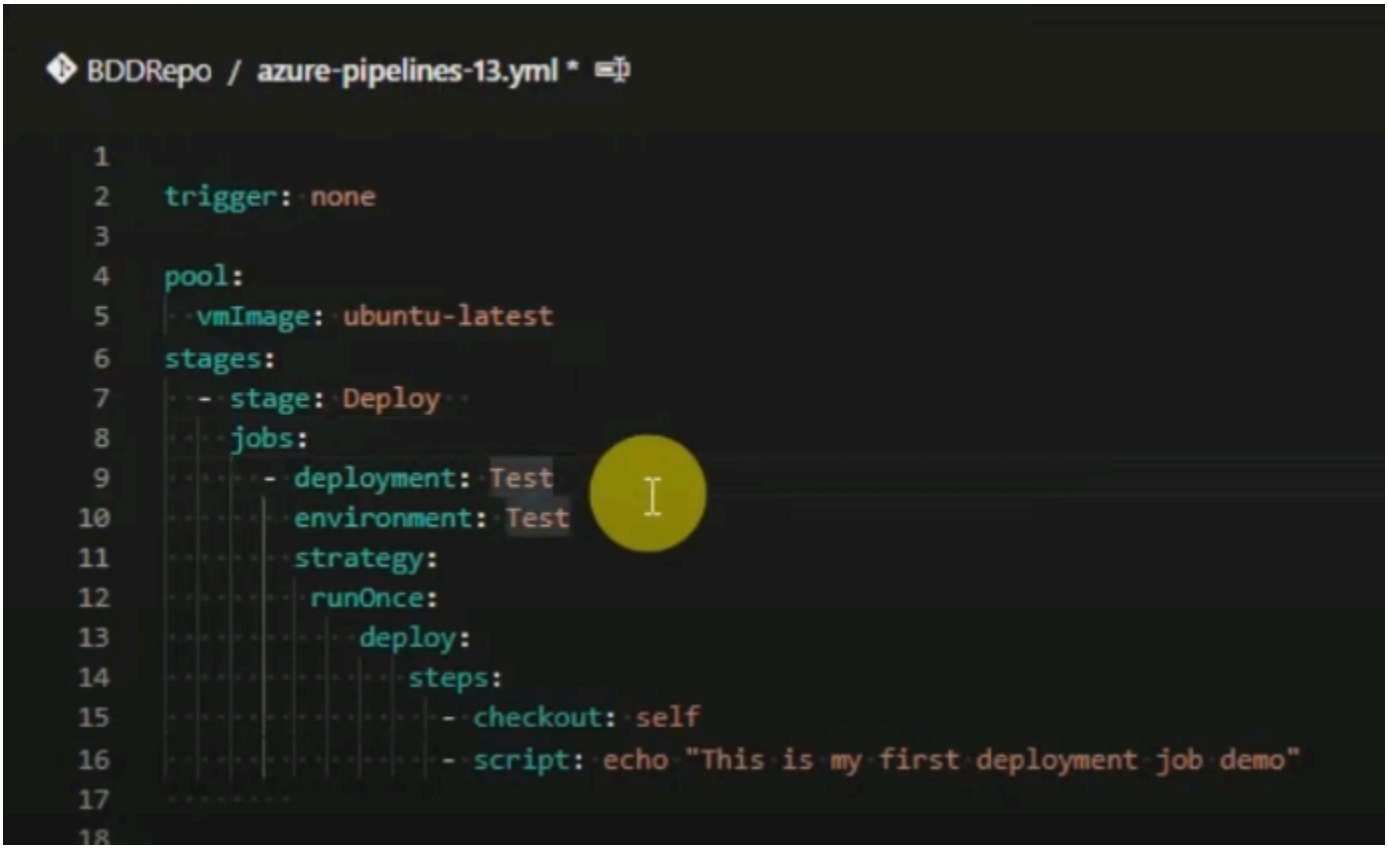
YAML pipeline multi jobs

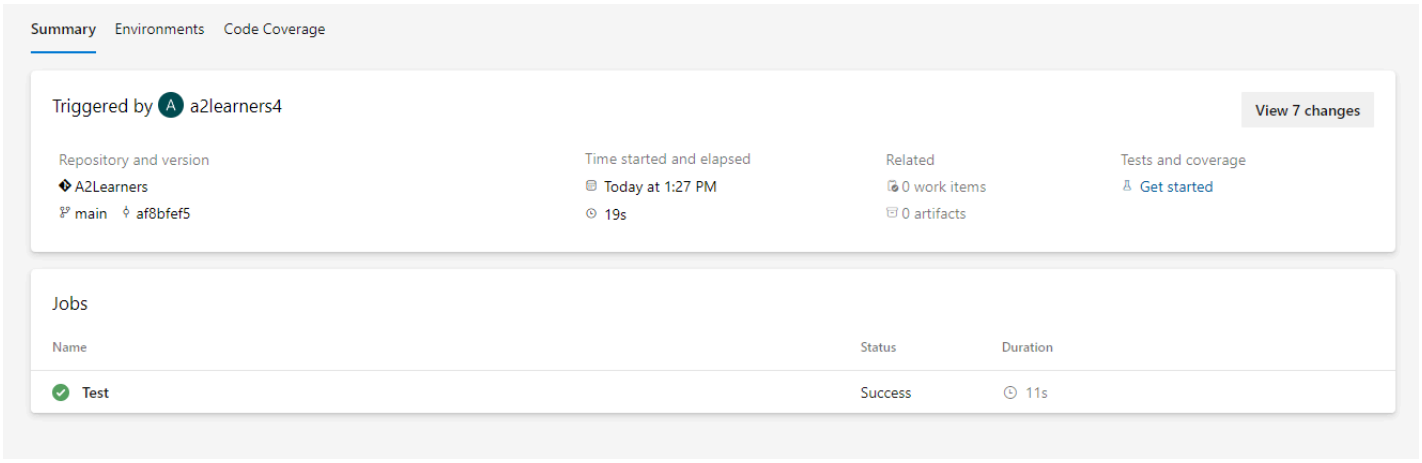
You can directly specify the jobs in your YAML file.

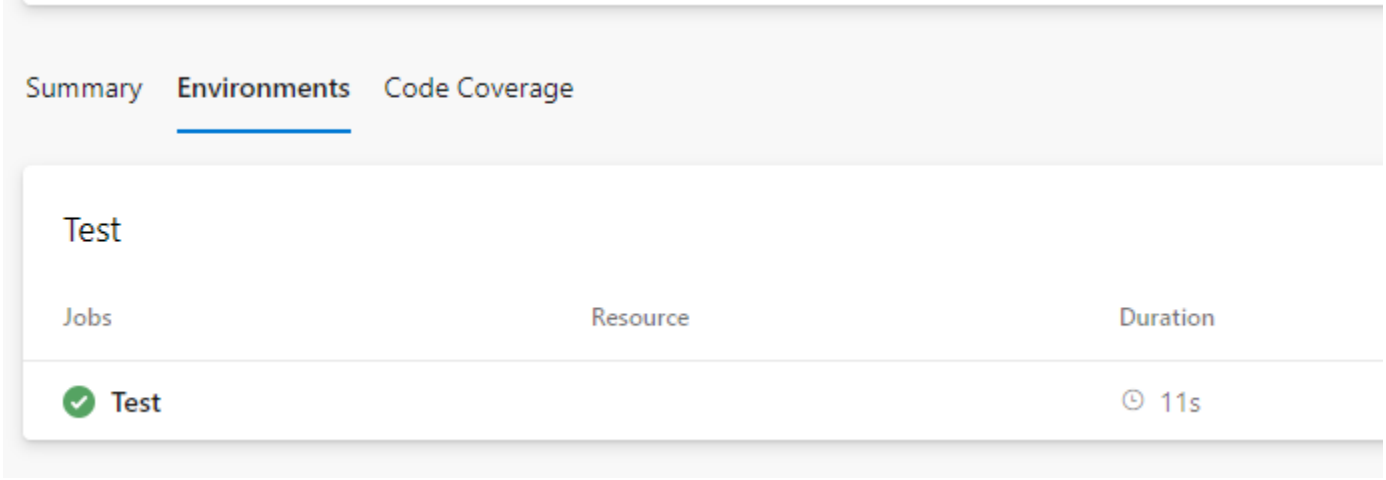
Your pipeline may have multiple jobs. In that case, use the jobs keyword.

```yaml
# this pipeline has one implicit stage
jobs:
- job: A
  steps:
  - bash: echo "A"

- job: B
  steps:
  - bash: echo "B"
```

How to create and run First deployment job using YAML pipeline in Azure DevOps |




Summary
Environments
Code Coverage
Triggered by a2learners4
View 7 changes
Repository and version
A2Learners
main
af8bfef5
Time started and elapsed
Today at 1:27 PM
19s
Related
0 work items
0 artifacts
Tests and coverage
Get started
Jobs
Name
Status
Duration
Test
Success
11s


Summary
Environments
Code Coverage
Test
Jobs
Resource
Duration
Test
11s

Pipeline Task types

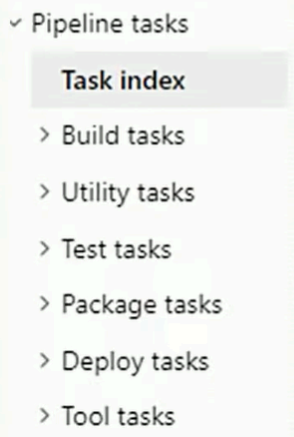
A task performs an action in a pipeline. A task is simply a procedure with a set of inputs.

Custom and Built in tasks

In YAML, you specify the major version using @ in the task name.
For example, version 2 of the PublishTestResults task is define like below

YAML

```
steps:
- task: PublishTestResults@2
```



Task control options

Control options are available as keys on the `task` section.

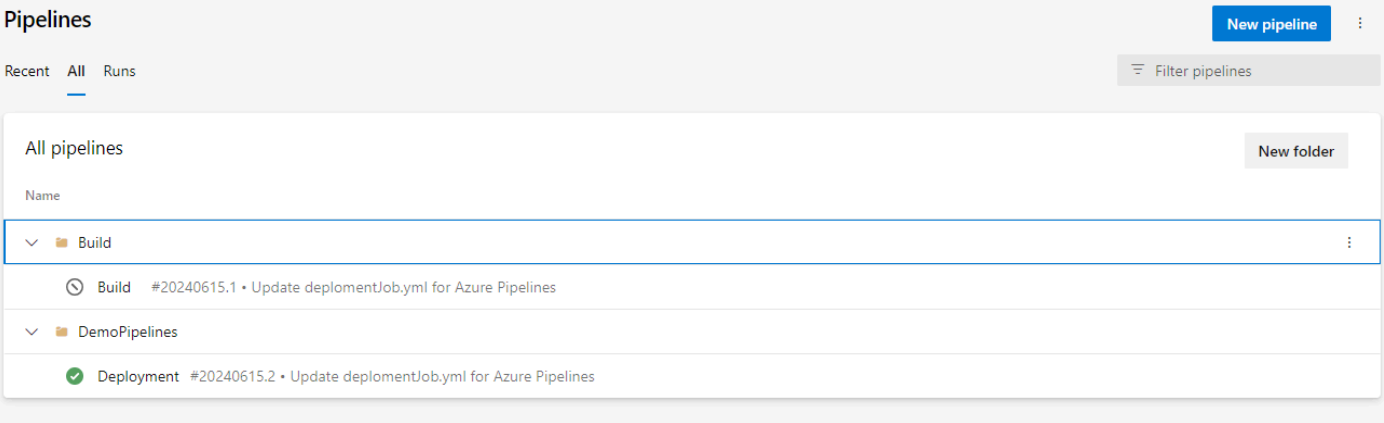
YAML Copy

```
- task: string  # reference to a task and version, e.g. "VSBuild@1"
  condition: expression     # see below
  continueOnError: boolean  # 'true' if future steps should run even if this
  enabled: boolean          # whether or not to run this step; defaults to 't
  retryCountOnTaskFailure: number # Max number of retries; default is zero
  timeoutInMinutes: number  # how long to wait before timing out the task
  target: string            # 'host' or the name of a container resource to t
```

In this YAML, `PublishTestResults@2` will run even if the previous step fails because of the succeededOrFailed() condition.

YAML Copy

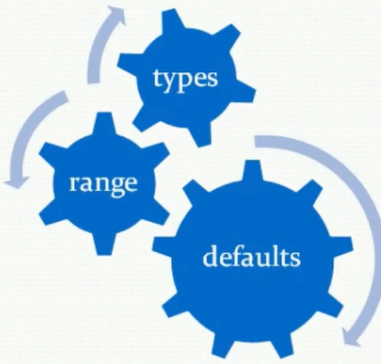
```
steps:
- task: UsePythonVersion@0
  inputs:
    versionSpec: '3.7'
    architecture: 'x64'
- task: PublishTestResults@2
  inputs:
    testResultsFiles: "**/TEST-*.xml"
  condition: succeededOrFailed()
```

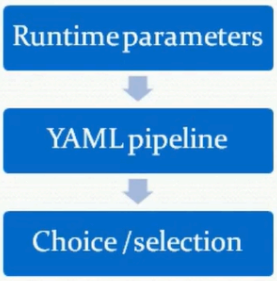


Runtime parameters in YAML pipeline

With runtime parameters :

1. Supply different values to scripts and tasks at runtime
2. Control parameter types, ranges allowed, and defaults
3. Dynamically select jobs and stages with template expressions
4. Runtime parameters does not apply to classic pipelines





Define parameters in YAML pipeline

How to Define parameters :

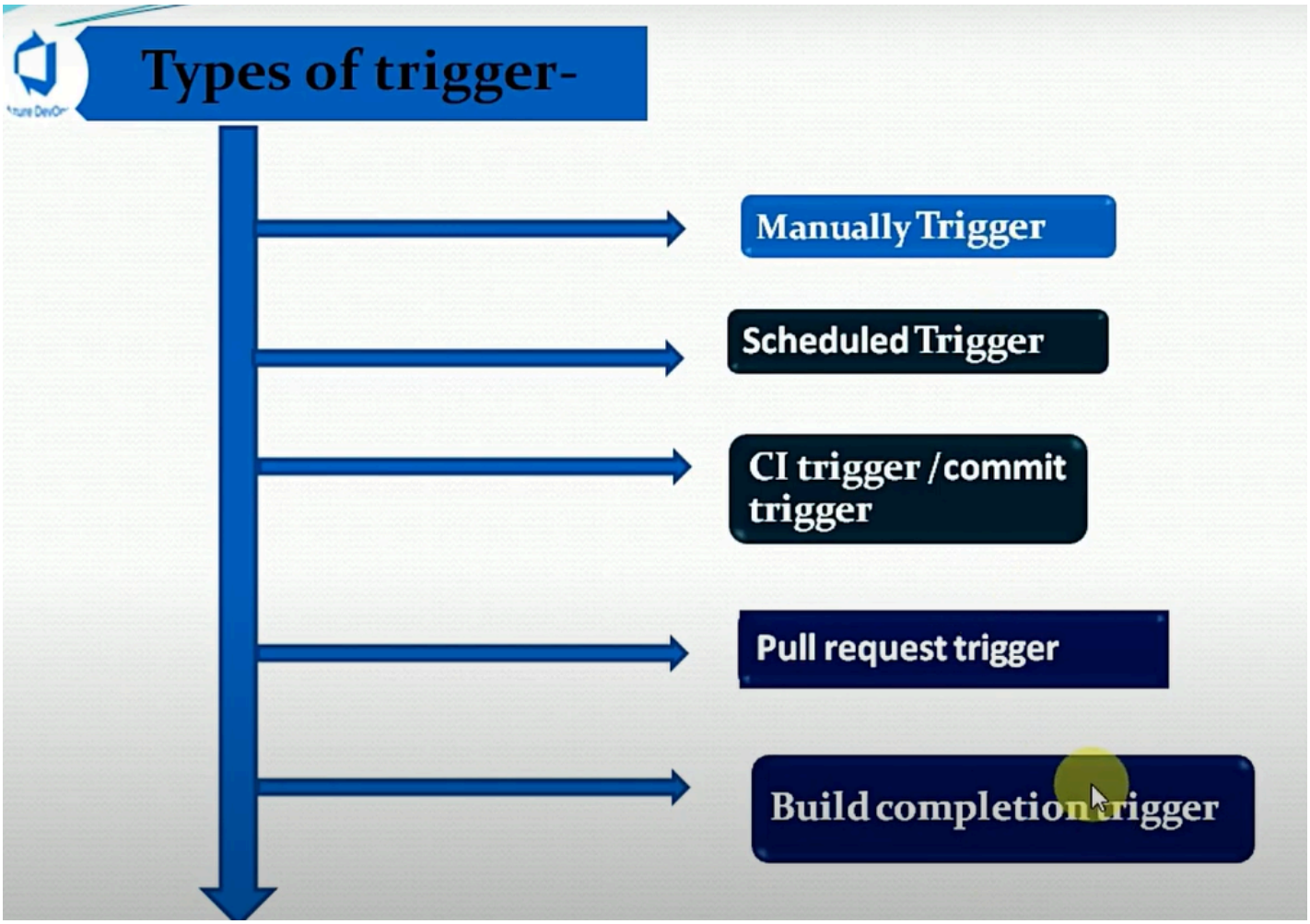
1. Parameters must contain a name and data type.
2. Parameters cannot be optional.
3. A default value needs to be assigned in YAML file or when we run pipeline. If we do not assign a default value or set default to false, the first available value will be used.
4. The parameters section in a YAML defines what parameters are available.

How to access parameters :

1. Parameters are access just before the pipeline runs so that **values** surrounded by **${{ }}** are replaced with parameter values.

```
${{parameters.image}}  #Template expression aat runtime
```

```yaml
parameters:
  - name: image
    displayName: Poolimage
    default: macOS-latest
    type: string
    values:
      - windows-latest
      - ubuntu-latest
```


Types of trigger-
Manually Trigger
Scheduled Trigger
CI trigger /commit trigger
Pull request trigger
Build completion trigger