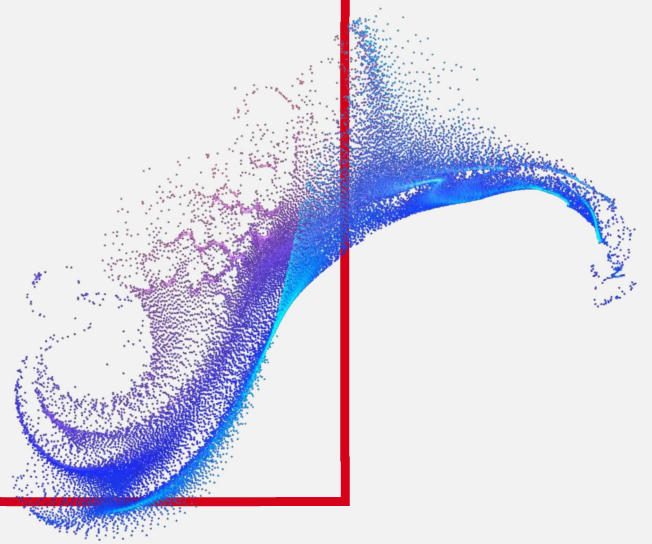


# Azure Pipelines



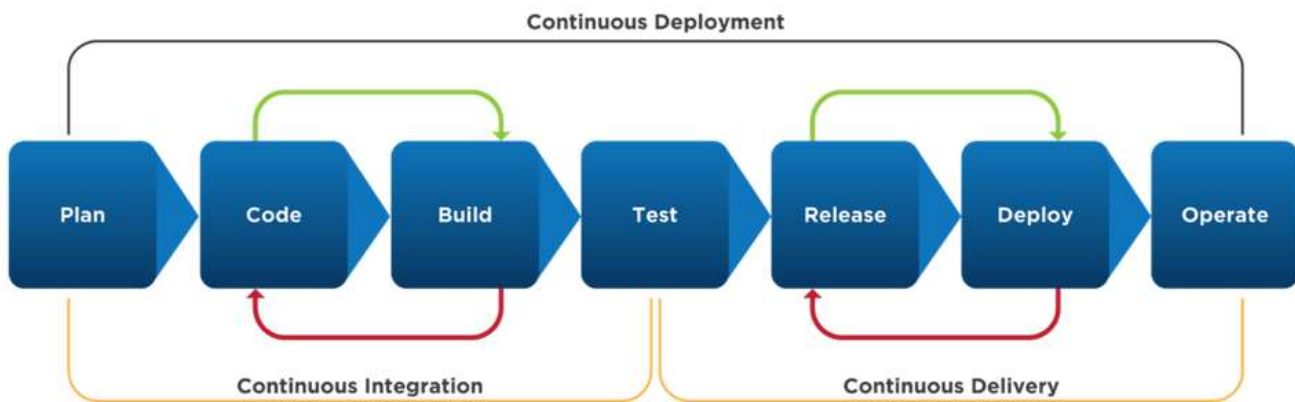
# Agenda

1. **CI/CD tool – Azure Pipelines**
  - What is CI/CD?
  - Get started
2. **Key concepts**
3. **Pipeline building block**
  - Repositories
  - Triggers
  - Tasks & templates
  - Jobs & stages
  - Library, variables, secure files
  - Approvals, checks & gates
4. **Build apps**
  - Multiple branches
  - Publish artifacts
  - Build numbers
5. **Run unit test & integration test**
6. **Integrate to Sonar**
7. **Trivy**
  - What's trivy?
  - How to install and use Trivy?
8. **Introduce common pipeline tasks**
9. **Deploy apps to AKS**
10. **Q&A**
11. **Demo**
12. **How to separate CI/CD repo with Application repo**
13. **Practice to write a full pipeline for a service**

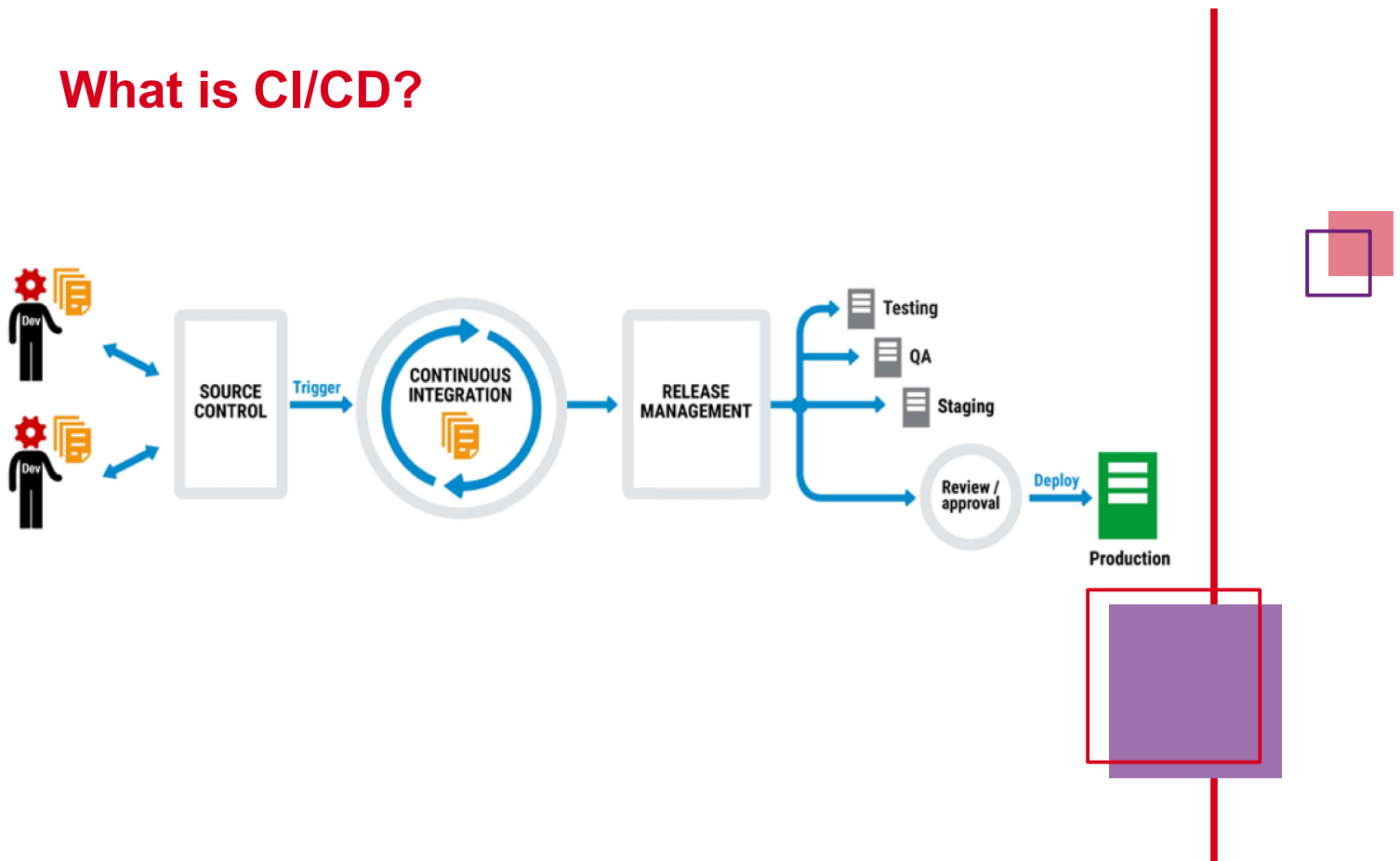
## **CI/CD tool – Azure Pipelines**



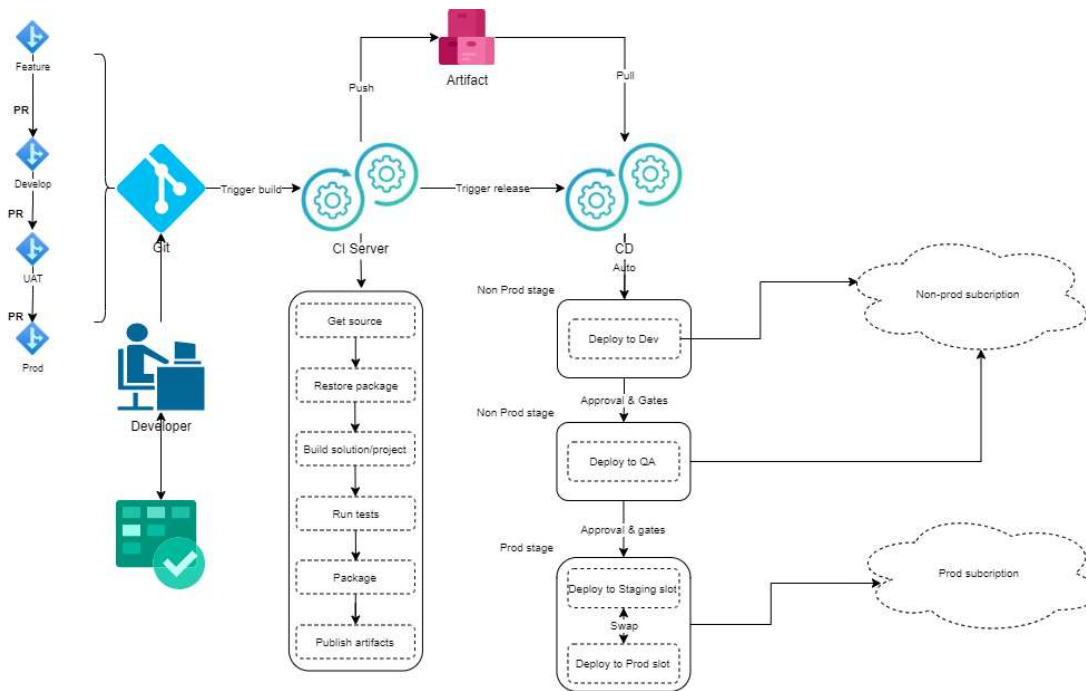
## What is CI/CD?



## What is CI/CD?



# Get started



# Get started

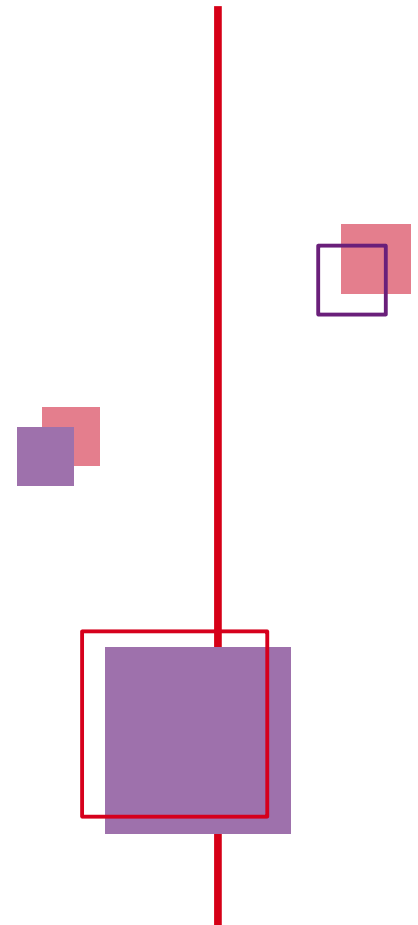
## ■ Prerequisites

- Create a Microsoft Account: [Create Your Azure Free Account Today | Microsoft Azure](#)
- Create a project: <https://dev.azure.com/>
- Invite team members

## 1. *Create your first pipeline*

## 2. *Customize your pipeline*

- *Build across multiple platforms*
- *Build using multiple versions*
- *Customize CI triggers*

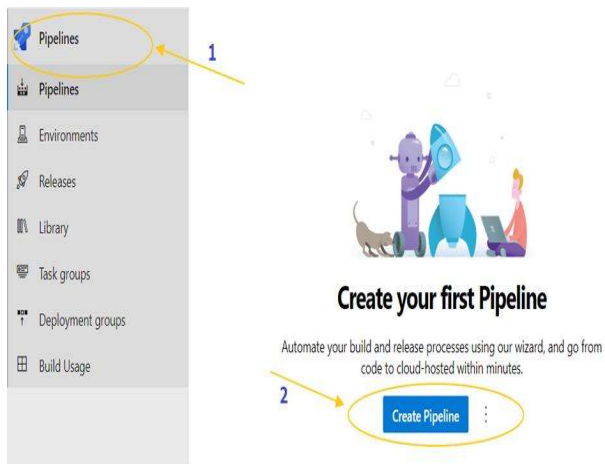


# Create your first pipeline

## ■ Prerequisites

- A Microsoft account
- An Azure DevOps organization
- A repository that has container source code

## Create your first pipeline (Demo)



```
1 # azure-pipelines.yml
2 variables:
3   # Agent VM image name
4   imageName: 'ubuntu-latest'
5
6 pool:
7   vmImage: $(imageName)
8
9 steps:
10  - script: |
11      echo "Hello Azure DevOps"
12      displayName: echo
13
```



# Customize your pipeline

- Build across multiple platforms

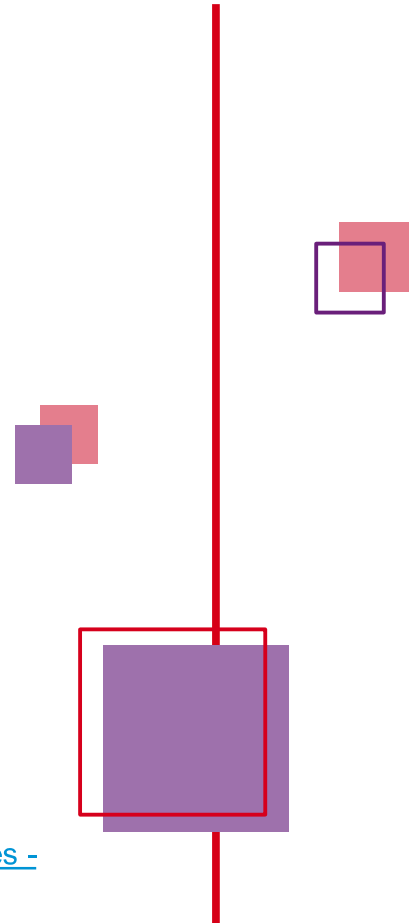
```
1 # azure-pipelines.yml
2 variables:
3   # Agent VM image name
4   imageName: 'ubuntu-latest'
5
6 pool:
7   vmImage: $(imageName)
8
9 steps:
10  - script: |
11    | echo "Hello Azure DevOps"
12    | displayName: echo
13
```



```
## azure-pipelines.yml
strategy:
  matrix:
    linux:
      imageName: 'ubuntu-latest'
    mac:
      imageName: 'macOS-latest'
    windows:
      imageName: 'windows-latest'
  maxParallel: 3

pool:
  vmImage: $(imageName)

steps:
  - script: |
    | echo "Hello Azure DevOps"
```



[Customize your pipeline - Azure Pipelines | Microsoft Docs](#)

[Microsoft-hosted agents for Azure Pipelines - Azure Pipelines | Microsoft Docs](#)

# Customize your pipeline

- Build using multiple versions

```
trigger:
- main

strategy:
  matrix:
    node18_linux:
      imageName: "ubuntu-latest"
      nodeVersion: "18.x"
    node20_windows:
      imageName: "windows-latest"
      nodeVersion: "20.x"
  maxParallel: 2

pool:
  vmImage: $(imageName)

steps:
- task: NodeTool@0
  inputs:
    versionSource: 'spec'
    versionSpec: '$(nodeVersion)'
    checkLatest: true
    displayName: "Install Node"

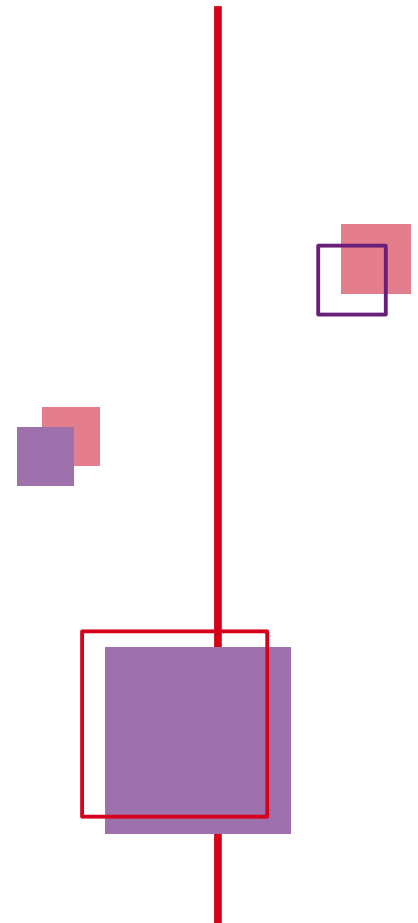
- script: |
  node --version
  displayName: "Check node version"
```

- Customize CI triggers

```
trigger:
  branches:
    include:
      - main
      - develop

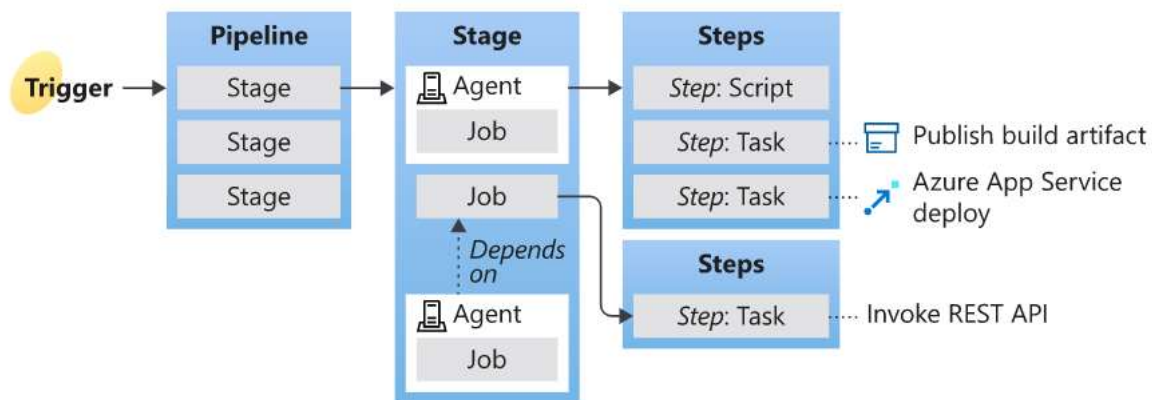
pr:
  branches:
    include:
      - main
      - develop

pool:
  vmImage: ubuntu-latest
```



# Key concepts for new Azure Pipelines users

## Key concepts overview



Refer to [Azure Pipelines New User Guide - Key concepts - Azure Pipelines | Microsoft Docs](#)

## Key concepts



# Pipeline building block

- Repository
- Trigger
- Tasks & templates
- Jobs & stages
- Library, variables, secure files
- Approvals, checks & gates



# Repository

Repository type	Azure Pipelines (YAML)	Azure Pipelines (classic editor)	Azure DevOps Server 2019, TFS 2018, TFS 2017, TFS 2015.4	TFS 2015 RTM
Azure Repos Git	Yes	Yes	Yes	Yes
Azure Repos TFVC	No	Yes	Yes	Yes
GitHub	Yes	Yes	No	No
GitHub Enterprise Server	Yes	Yes	TFS 2018.2 and higher	No
Bitbucket Cloud	Yes	Yes	No	No
Bitbucket Server	No	Yes	Yes	Yes
Subversion	No	Yes	Yes	No

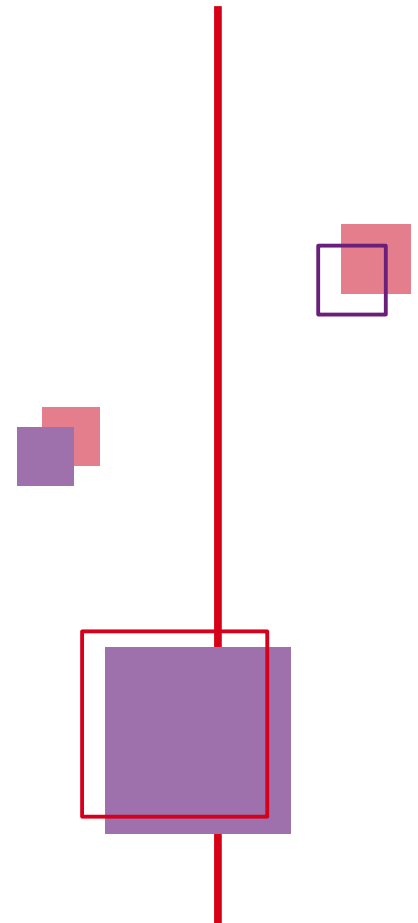
# Trigger

- Types of triggers: branches, pr, paths

```
##### Trigger #####
# specific branch build
# specific path build
trigger:
  branches:
    include:
      - main
      - develop/*
      - feature/*
      - bug/*
    exclude:
      - develop/*-bk*
  paths:
    include:
      - '**'
    exclude:
      - .ci/*
      - .cd/*
pr:
  branches:
    include:
      - main
      - develop
pool:
  vmImage: ubuntu-latest
```

- Scheduled triggers

```
schedules:
- cron: '00 03 * * *'
  displayName: Daily midnight build
  branches:
    include:
      - develop
    exclude:
      - develop/*-bk
- cron: '0 12 * * 0'
  displayName: Weekly Sunday build
  branches:
    include:
      - main
  always: true
pool:
  vmImage: ubuntu-latest
steps:
- task: NodeTool@0
  inputs:
    versionSource: 'spec'
    versionSpec: '20.x'
    checkLatest: true
    displayName: "Install Node"
- script: |
  node --version
  displayName: "Check node version"
```



# Tasks & Templates

## Tasks

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 step fails; defaults to 'false'
  enabled: boolean # whether or not to run this step; defaults to 'true'
  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 target
```

YAML

Copy

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



# Tasks & Templates

## Templates

### Passing parameters

YAML

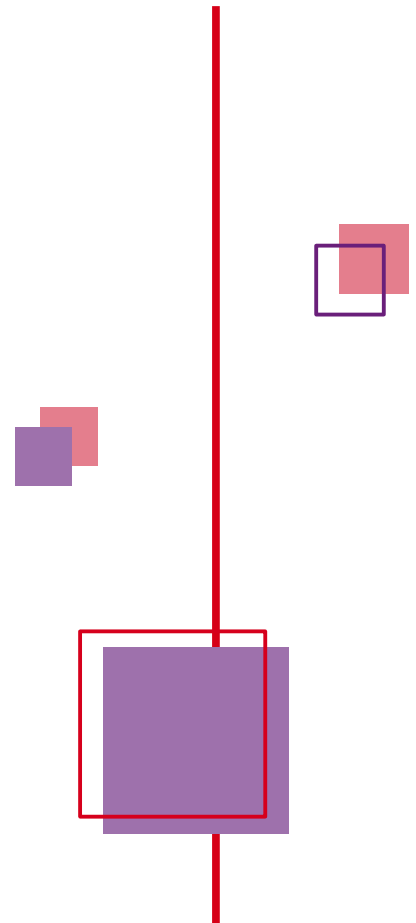
```
# File: simple-param.yml
parameters:
- name: yesNo # name of the parameter; required
  type: boolean # data type of the parameter; required
  default: false

steps:
- script: echo ${ parameters.yesNo }
```

YAML

```
# File: azure-pipelines.yml
trigger:
- main

extends:
  template: simple-param.yml
  parameters:
    yesNo: false # set to a non-boolean value to have the build fail
```



# Tasks & Templates

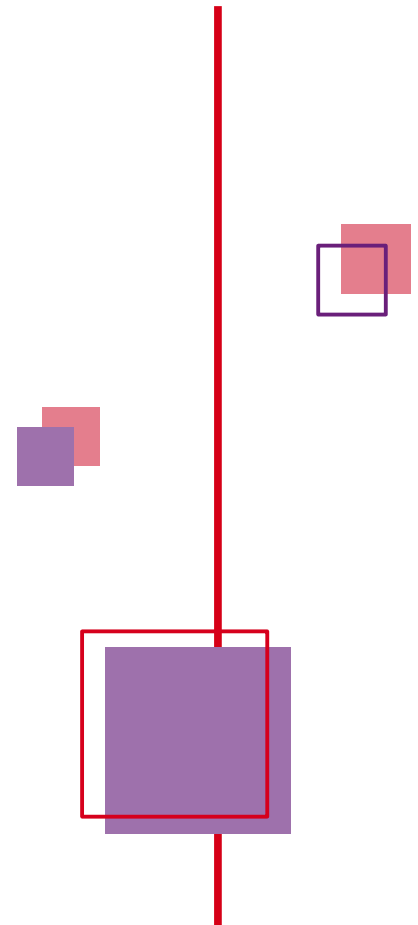
## Templates

### Parameters to select a template at runtime

yml

```
#azure-pipeline.yml
parameters:
- name: experimentalTemplate
  displayName: 'Use experimental build process?'
  type: boolean
  default: false

steps:
- ${{ if eq(parameters.experimentalTemplate, true) }}:
  - template: experimental.yml
- ${{ if not(eq(parameters.experimentalTemplate, true)) }}:
  - template: stable.yml
```



# Jobs & Stages

## Jobs

YAML

```
jobs:
- job: A
  steps:
  - bash: echo "A"

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

```
# Jobs
pool:
  vmImage: 'ubuntu-latest'

jobs:
- job: myJob
  steps:
  - bash: echo "Hello world"
```

## Stages

YAML

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

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

```
# stages
pool:
  vmImage: 'ubuntu-latest'

stages:
- stage: Demo1
  jobs:
  - job: A1
    steps:
    - script: |
      echo "Hello A1"
  - job: A2
    steps:
    - script: |
      echo "Hello A2"

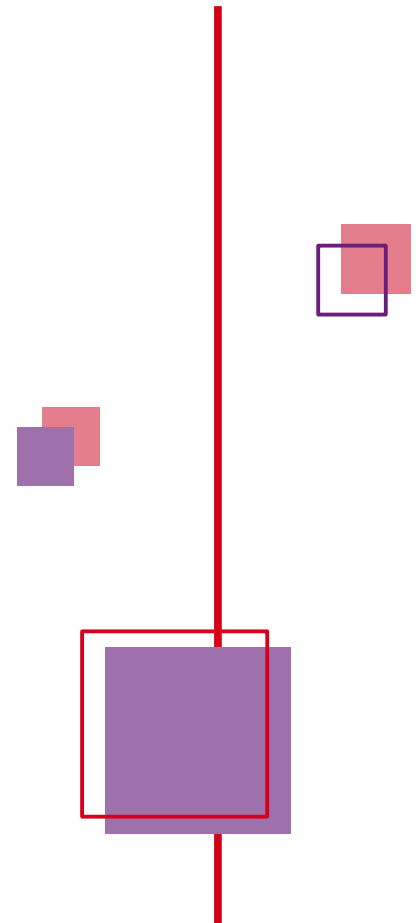
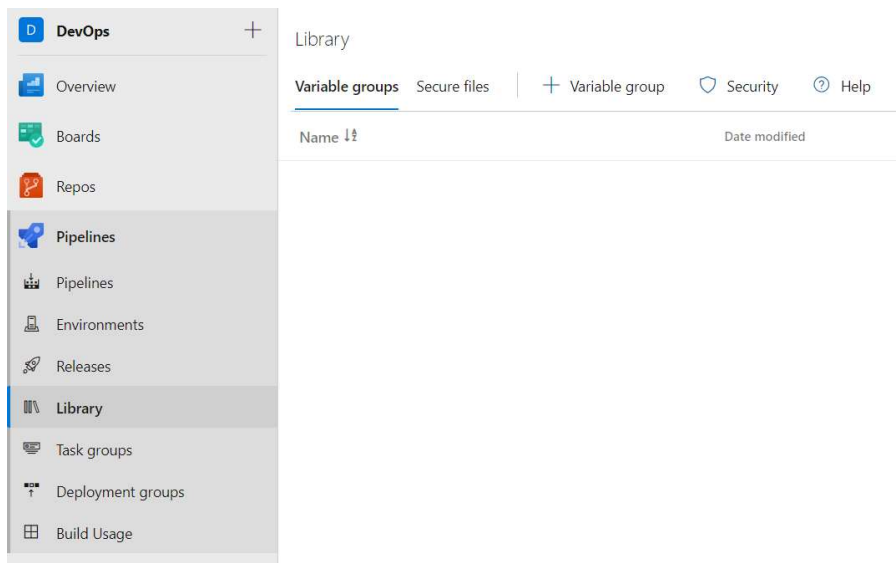
- stage: Demo2
  jobs:
  - job: B1
    steps:
    - script: |
      echo "Hello B1"
  - job: B2
    steps:
    - script: |
      echo "Hello B2"
```



# Library, variables, secure files

- **Library of assets**

- The library contains two types of assets: [variable groups](#) and [secure files](#).



# Variables

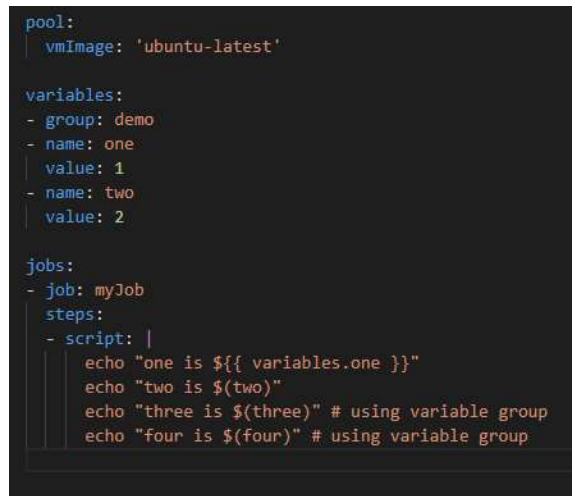
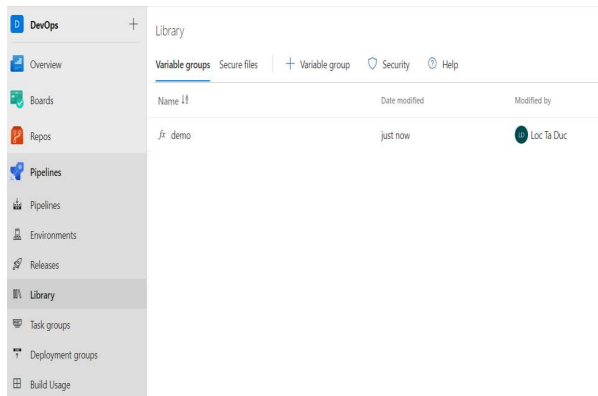
## ▪ Variables syntax

- *Template expression variables:* `${{ variables.one }}`
- *Macro syntax variables:* `$(one)`
- *Runtime expression syntax:* `${variables.var}`

Syntax	Example	When is it processed?	Where does it expand in a pipeline definition?	How does it render when not found?
macro	<code>\$(var)</code>	runtime before a task executes	value (right side)	prints <code>\$(var)</code>
template expression	<code>\${{ variables.var }}</code>	compile time	key or value (left or right side)	empty string
runtime expression	<code>\${variables.var}</code>	runtime	value (right side)	empty string

# Variables

- Use a variables in variable group



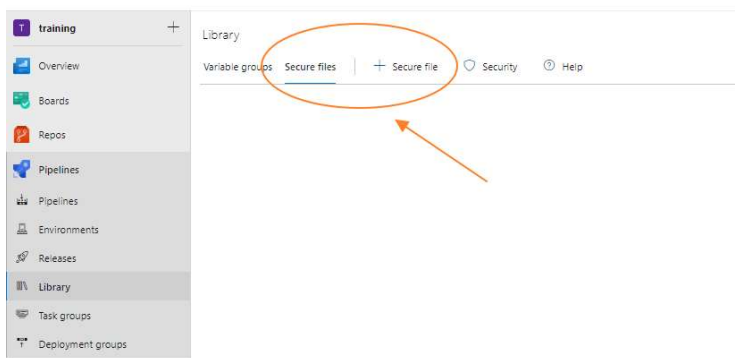
[Define variables - Azure Pipelines | Microsoft Docs](#)

## Secure files

These files can be stored on the server without having to commit them to your repository.

- signing certificates
- Apple Provisioning Profiles
- Android Keystore files
- SSH keys

The size limit for each secure file is 10 MB.



# Secure files

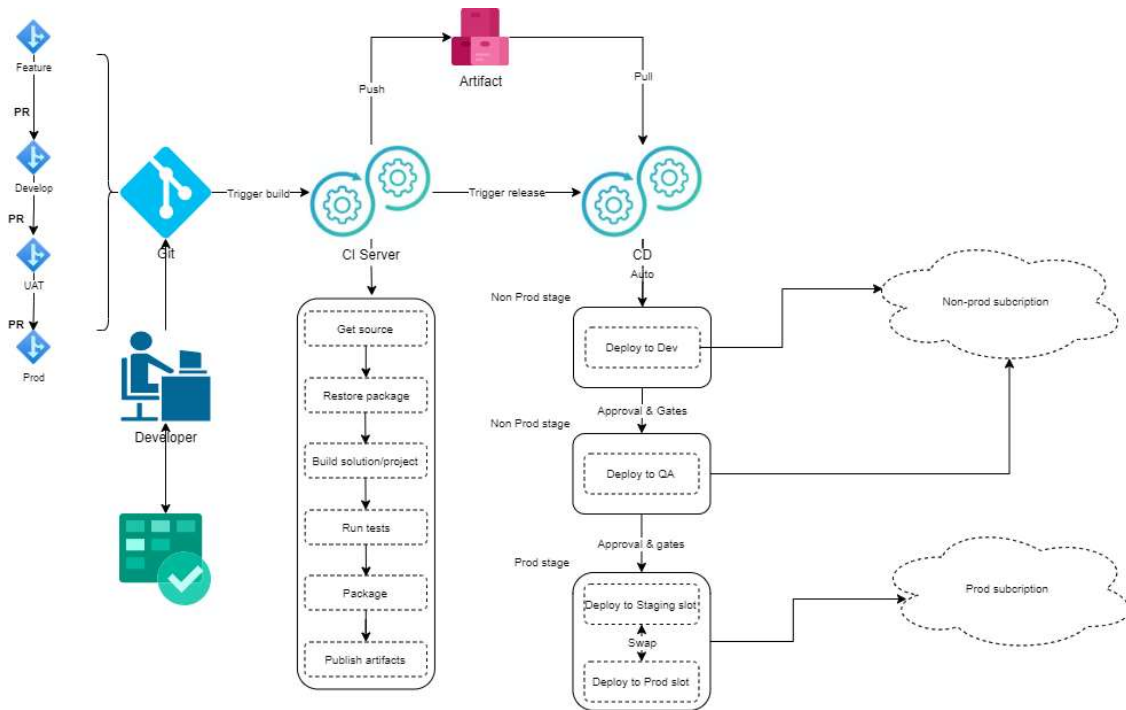
How to consume a secure file in a pipeline?

```
2
3 - task: DownloadSecureFile@1
4   name: caCertificate
5   displayName: 'Download CA certificate'
6   inputs:
7     secureFile: 'demo.pem'
8
9 - script: |
10   echo Installing $(caCertificate.secureFilePath) to the trusted CA directory...
11   sudo chown root:root $(caCertificate.secureFilePath)
12   sudo chmod a+r $(caCertificate.secureFilePath)
13   sudo ln -s -t /etc/ssl/certs/ $(caCertificate.secureFilePath)
```

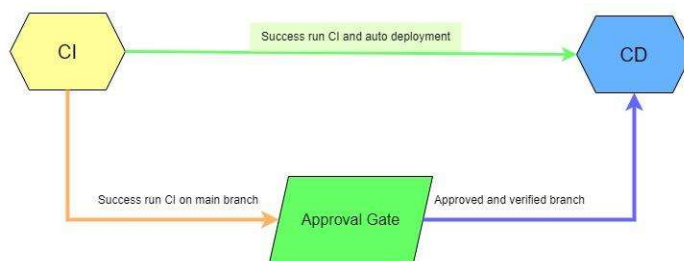
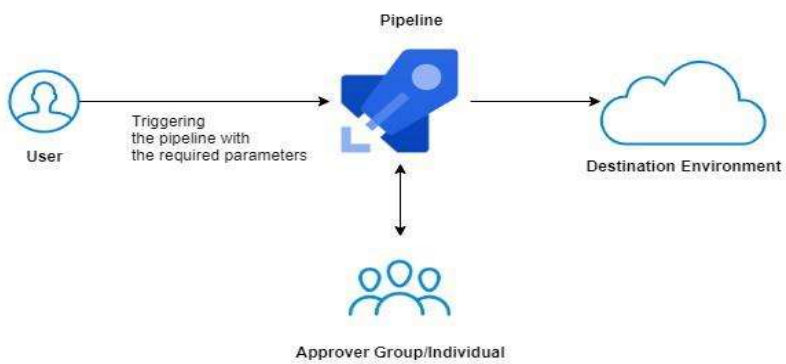
[Secure files for Azure Pipelines - Azure Pipelines | Microsoft Docs](#)



## Approvals, checks & gates

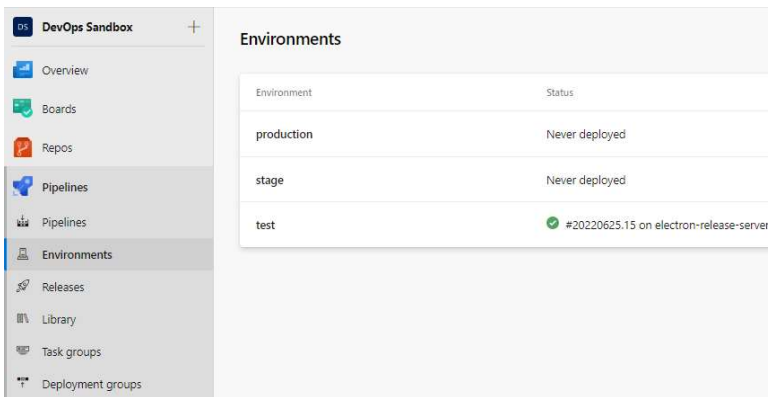


## Approvals, checks & gates

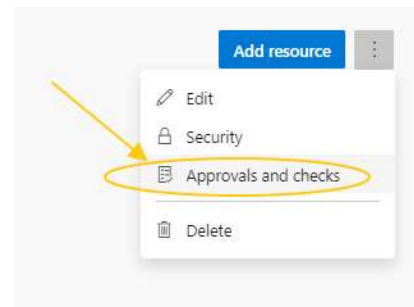


# Approvals, checks & gates

- To use approvals and check you need to define environment first.  
Once you have an environment you can define approvals and checks



Environment	Status
production	Never deployed
stage	Never deployed
test	✓ #20220625.15 on electron-release-server



**Build apps**



# Build multiple branches

- Set up a CI trigger for a topic branch

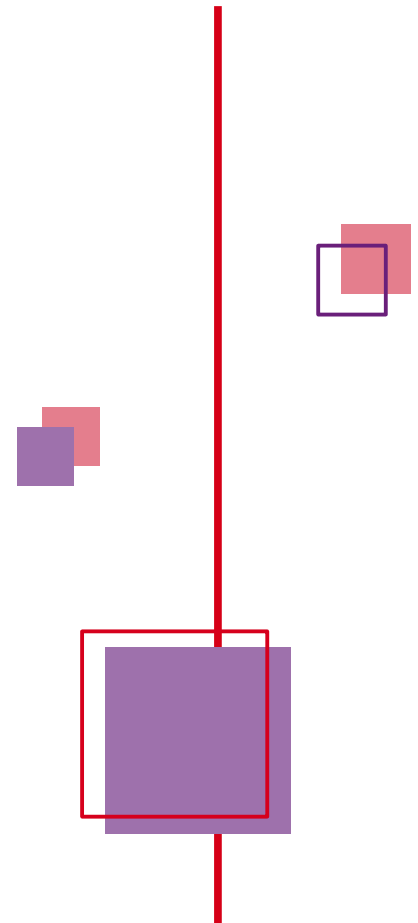
```
trigger:  
  branches:  
    include:  
      - main  
      - develop  
      - feature/*  
      - bug/*
```

- Exclude or include tasks for builds based on the branch being built

```
# Jobs  
pool:  
  vmImage: 'ubuntu-latest'  
  
jobs:  
  - job: myJob  
    steps:  
      - bash: echo "Hello world"  
  
  - task: PublishBuildArtifacts@1  
    condition: and(succeeded(), eq(variables['Build.SourceBranch'], 'refs/heads/main'))
```

- Validate pull requests

Use policies to protect your branches by requiring successful builds before merging pull requests.



# Publish Pipeline Artifacts

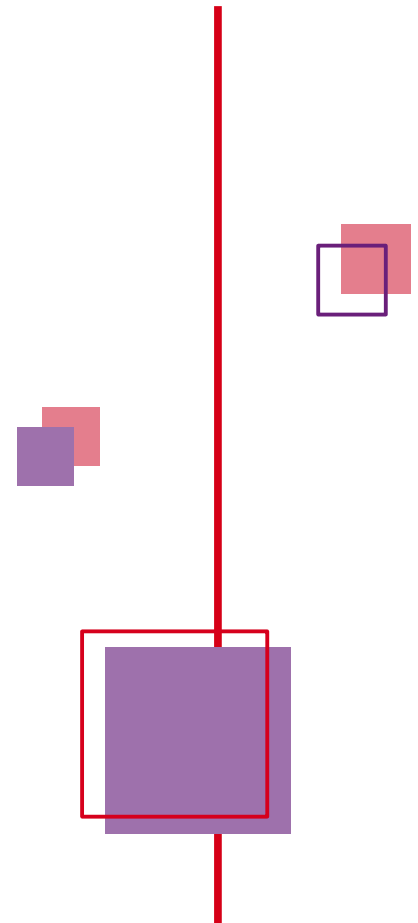
## Publish Artifacts

You can publish your Artifacts at any stage of your pipeline using YAML or the classic editor.

```
# Publish Artifact
steps:
- task: PublishPipelineArtifact@1
  inputs:
    targetPath: '${Pipeline.Workspace}'
    artifactType: 'pipeline'
    artifactName: 'drop'
```

## Publish Artifacts from the command line

```
# Publish Artifact by az cli
az pipelines runs artifact upload \
  --artifact-name your_artifact_name \
  --path your_path_to_publish \
  --run-id '<artifact_run_id>'
```



# Build Numbers

In YAML, this property is called name and is at the root level of a pipeline. If not specified, your run is given a unique integer as its name. You can give runs much more useful names that are meaningful to your team. You can use a combination of tokens, variables, and underscore characters.

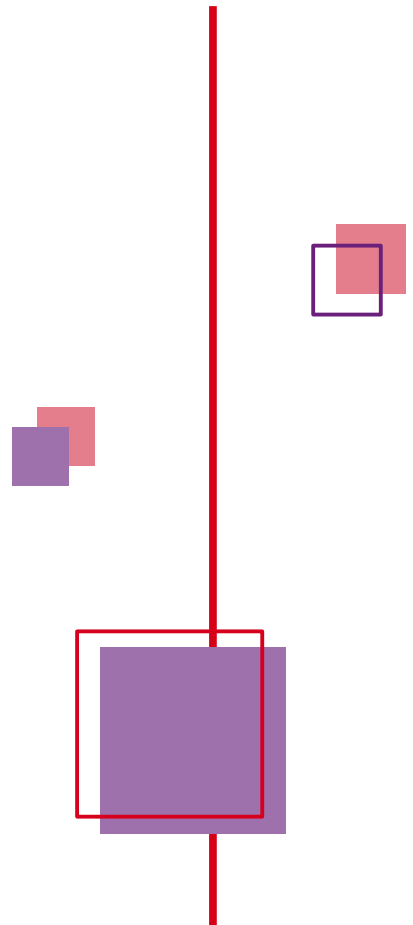
```
# azure-pipelines.yaml
trigger:
  branches:
    include:
      - main
      - develop
      - feature/*
      - bug/*
  batch: true

pool:
  vmImage: 'ubuntu-latest'

name: $(TeamProject)_$(Build.DefinitionName)_$(SourceBranchName)_$(Date:yyyyMMdd)$(Rev:.r)

steps:
  - script: echo '$(Build.BuildNumber)'
```

[Predefined variables - Azure Pipelines | Microsoft Docs](#)



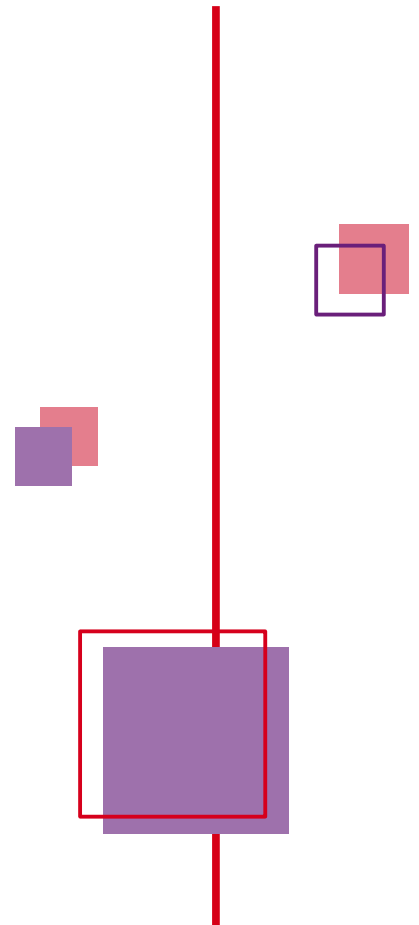
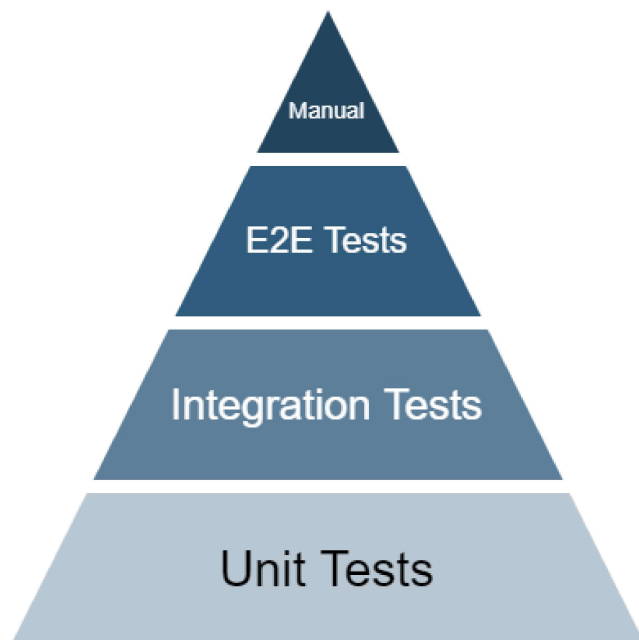
**Run unit test & integration  
test**





## Unit test & Integration test

Unit testing and integration testing make up the pyramid's two base layers, emphasizing their importance in a comprehensive testing strategy:



# Unit Test

## What is Unit Testing?

- Unit Testing is a type of software testing where individual units or components of a software are tested

## Why do Unit Testing?

- developers sometimes try saving time doing minimal unit testing
- Here, are the key reasons to perform unit testing:
  - help to fix bugs early
  - help to understand the testing code base and enables them to make changes quickly
  - Good unit tests serve as project documentation
  - help with code re-use

## How to run Unit Testing in Azure Pipelines?

```
# Unit Test
steps:
# ...
# do this after other tasks such as building
- task: DotNetCoreCLI@2
  inputs:
    command: test
    projects: '**/*UnitTests/*.csproj'
    arguments: '--configuration $(buildConfiguration)'
```

# Integration Test

## What is integration testing?

- Integration testing is defined as a type of testing where software modules are integrated logically and tested as a group

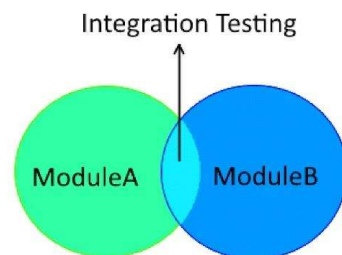
## Why do Integration Testing?

Although each software module is unit tested, defects still exist for various reasons like

- A Module, in general, is designed by an individual software developer whose understanding and programming logic may differ from other programmers
- At the time of module development, there are wide chances of change in requirements by the clients
- Interfaces of the software modules with the database could be erroneous
- External Hardware interfaces, if any, could be erroneous
- Inadequate exception handling could cause issues

## How to run Integration Testing in Azure Pipelines?

```
# Integration test
steps:
# ...
# do this after other tasks such as building
- task: DotNetCoreCLI@2
  inputs:
    command: test
    projects: '**/*IntegrationTests/*.csproj'
    arguments: '--configuration $(buildConfiguration)'
```



**Integrate to Sonar**



# Integration to SonarQube

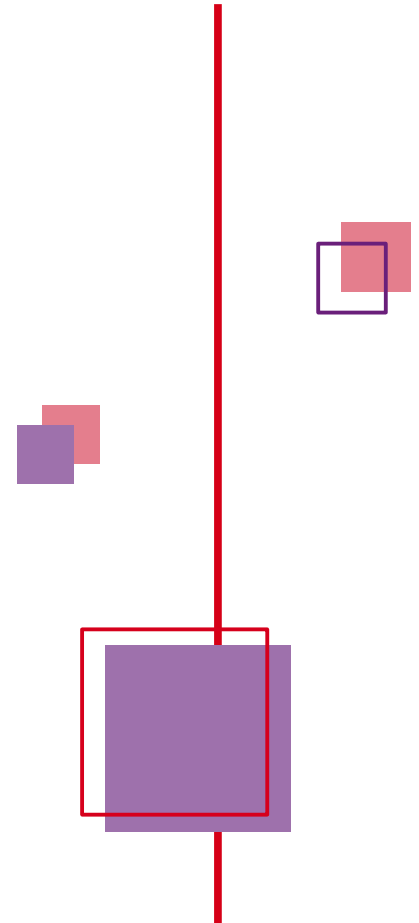
## What is SonarQube?

- SonarQube an open source platform for continuous inspection of code quality to perform automatic reviews with static analysis of code

## Why do you should use SonarQube?

## How to integration SonarQube?

- Demo



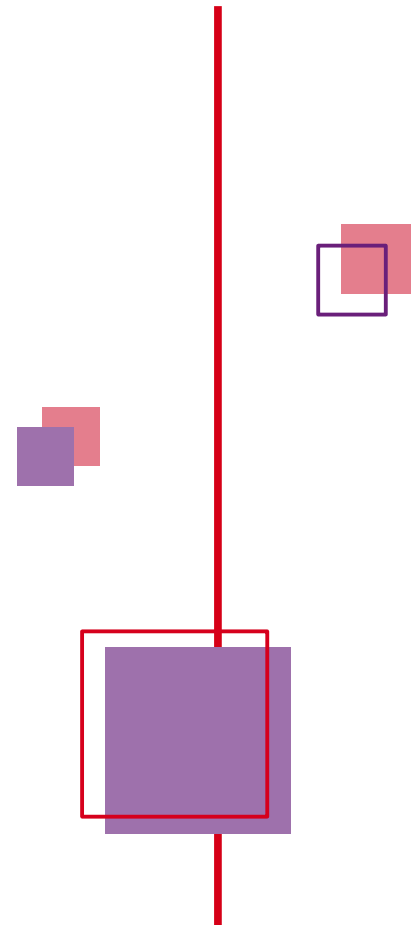
**Trivy**



# What's trivy?

## What is trivy?

- Trivy is a tool scanner for vulnerabilities in container images, file systems, and Git repositories, as well as for configuration issues.
- Trivy detects vulnerabilities of OS packages (Alpine, RHEL, CentOS, etc.) and language-specific packages (Bundler, Composer, npm, yarn, etc.).
- Trivy scans Infrastructure as Code (IaC) files such as Terraform, Dockerfile and Kubernetes, to detect potential configuration issues that expose your deployments to the risk of attack
- Trivy also scans hardcoded secrets like passwords, API keys and tokens
- Target:
  - Container Image
  - Filesystem
  - Git repository (remote)
  - Kubernetes cluster or resource
- Scanners:
  - OS packages and software dependencies in use (SBOM)
  - Known vulnerabilities (CVEs)
  - IaC misconfigurations
  - Sensitive information and secrets



# How to install and use Trivy?

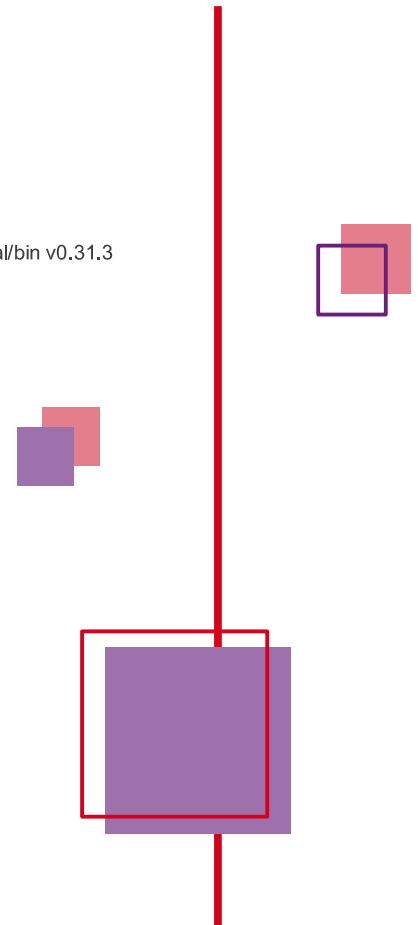
## Install

- Ubuntu:
  - `curl -sL https://raw.githubusercontent.com/aquasecurity/trivy/main/contrib/install.sh | sh -s -- -b /usr/local/bin v0.31.3`
- <https://aquasecurity.github.io/trivy/v0.18.3/installation/>

## General usage

- `trivy image --severity HIGH,CRITICAL your_docker_image`
- `trivy fs --security-checks vuln,secret,config myproject/`
- `trivy k8s --report summary cluster`

– Refer: <https://github.com/aquasecurity/trivy>





**Deploy apps to AKS**



# Using Environments – AKS

## Using Environments – AKS

To deploy an application to AKS

- Create a resource group
- Create an azure container registry
- Create an AKS
- Create a Docker registry service connection
- Create a Kubernetes service connection
- Create a new environment
- Demo

## Build and Push Docker image

```
5 trigger:
6 - main
7
8 resources:
9 - repo: self
10
11 variables:
12 # Container registry service connection established during pipeline creation
13 dockerRegistryServiceConnection: 'acr'
14 kubernetesServiceConnection: 'aks'
15 imageRepository: 'mydockerimage'
16 containerRegistry: 'mydockerimage.azurecr.io'
17 dockerfilePath: '$(Build.SourcesDirectory)/app/Dockerfile'
18 tag: '$(Build.BuildId)'
19 imagePullSecret: 'acrsecret'
20 k8sNamespace: 'test'
21 environment: 'test-cluster'
22 appName: 'demonodejs'
23
24 # Agent VM image name
25 vmImageName: 'ubuntu-latest'
26
27 stages:
28 - stage: Build
29   displayName: Build and push stage
30   jobs:
31   - job: Build
32     displayName: Build
33     pool:
34       vmImage: '$(vmImageName)'
35     steps:
36
37     - task: Docker@2
38       displayName: Build and push an image to container registry
39       inputs:
40         command: buildAndPush
41         repository: '$(imageRepository)'
42         dockerfile: '$(dockerfilePath)'
43         containerRegistry: '$(dockerRegistryServiceConnection)'
44         tags: |
45           $(tag)
46
47     - task: PublishPipelineArtifact@1
48       inputs:
49         artifactName: 'manifests'
50         path: 'manifests'
51
```

# Using Environments – AKS

## Build and Push Docker image

```
5 trigger:
6   - main
7
8 resources:
9   - repo: self
10
11 variables:
12   # Container registry service connection established during pipeline creation
13   dockerRegistryServiceConnection: 'acr'
14   kubernetesServiceConnection: 'aks'
15   imageRepository: 'mydocimage1'
16   containerRegistry: 'nashtechdemoacr.azurecr.io'
17   dockerfilePath: '$(Build.SourcesDirectory)/app/Dockerfile'
18   tag: '$(Build.BuildId)'
19   imagePullSecret: 'acrsecret'
20   k8sNamespace: 'test'
21   environment: 'test-cluster'
22   appname: 'demonodejs'
23
24 # Agent VM image name
25 vmImageName: 'ubuntu-latest'
26
27 stages:
28   - stage: Build
29     displayName: Build and push stage
30     jobs:
31       - job: Build
32         displayName: Build
33         pool:
34           vmImage: $(vmImageName)
35         steps:
36
37       - task: Docker@2
38         displayName: Build and push an image to container registry
39         inputs:
40           command: buildAndPush
41           repository: $(imageRepository)
42           dockerfile: $(dockerfilePath)
43           containerRegistry: $(dockerRegistryServiceConnection)
44           tags: |
45             $(tag)
46
47       - task: PublishPipelineArtifact@1
48         inputs:
49           artifactName: 'manifests'
50           path: 'manifests'
51
```

## Deploy app to AKS

```
✓ - stage: Deploy
  displayName: Deploy stage
  dependsOn: Build
  jobs:
    - deployment: Deploy
      displayName: Deploy job
      pool:
        vmImage: $(vmImageName)
      environment: '$(environment)' #customize with your environment
      strategy:
        runOnce:
          deploy:
            steps:
              - task: DownloadPipelineArtifact@2
                inputs:
                  artifactName: 'manifests'
                  downloadPath: '$(System.ArtifactsDirectory)/manifests'
              - task: KubernetesManifest@0
                displayName: Create imagePullSecret
                inputs:
                  action: createSecret
                  kubernetesServiceConnection: '$(kubernetesServiceConnection)'
                  secretName: $(imagePullSecret)
                  namespace: $(k8sNamespace)
                  dockerRegistryEndpoint: $(dockerRegistryServiceConnection)
              - task: KubernetesManifest@0
                displayName: Deploy to Kubernetes cluster
                inputs:
                  action: deploy
                  kubernetesServiceConnection: '$(kubernetesServiceConnection)'
                  namespace: $(k8sNamespace)
                  manifests: |
                    $(System.ArtifactsDirectory)/manifests/deployment.yml
                    $(System.ArtifactsDirectory)/manifests/service.yml
                  imagePullSecrets: |
                    $(imagePullSecret)
                  containers: |
                    $(containerRegistry)/$(imageRepository):$(tag) | Loc Ta Duc, 2 days ago
```

## Demo

[Deploy to Azure Kubernetes Service with Azure Pipelines - Azure Kubernetes Service | Microsoft Docs](#)

**Introduce common pipeline  
tasks**



# Introduce common pipeline tasks

```
- task: NodeTool@0
  inputs:
    versionSpec: '16.x'
  displayName: Install Nodejs

- task: Yarn@3
  inputs:
    projectDirectory: '$(Build.SourcesDirectory)'
    arguments: '--frozen-lockfile'
  displayName: 'Yarn (install)'

- task: Yarn@3
  inputs:
    projectDirectory: '$(Build.SourcesDirectory)'
    arguments: 'build'
  displayName: 'Yarn (build)'

- script: |
  echo "training"
  displayName: "Training"

- task: Bash@3
  inputs:
    filePath: 'demo.sh'
    arguments: '--name $(projectname)'
  displayName: "Demo"
```

```
- task: Docker@2
  displayName: Login to Docker Hub
  inputs:
    command: login
    containerRegistry: DockerHub

- task: Docker@2
  displayName: Build NodeJs Project
  inputs:
    containerRegistry: 'dockerhub'
    repository: 'training/nodejs'
    arguments: '--rm --squash'
    command: 'build'
    buildContext: '.'
    dockerfile: '$(Build.SourcesDirectory)/Dockerfile'
    tags: |
      latest
      $(Build.BuildNumber)

- task: Docker@2
  inputs:
    containerRegistry: 'dockerhub'
    repository: 'training/nodejs'
    command: 'push'
    tags: |
      latest
      $(Build.BuildNumber)
    addPipelineData: false
    addBaseImageData: false
  displayName: Push Image to DockerHub
```

**Q&A**



**Demo**



# Agenda

1. Practice to write a full pipeline for a service
2. Best practices
  - Guide how to separate DevOps repo with Application repo
  - Create template from DevOps repo and using it from DevOps repo
  - Versioning
3. Q&A



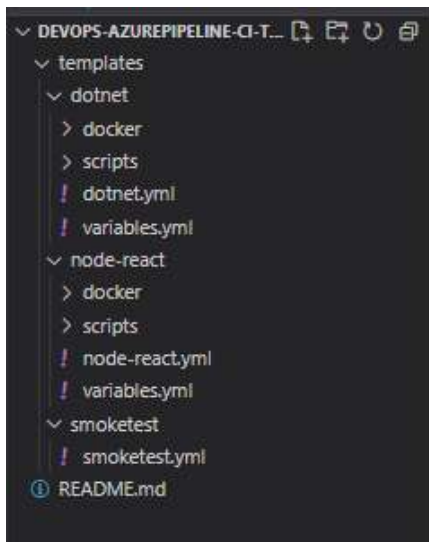
## Best practices

- Guide how to separate DevOps repo with Application repo
- Create template from DevOps repo and using it from DevOps repo
- Versioning

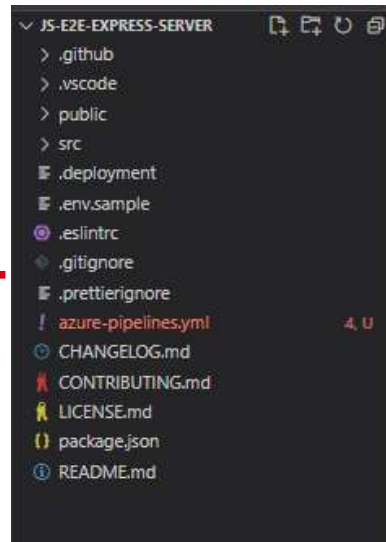


## how to separate DevOps repo with Application repo

Devops repository



Application repository



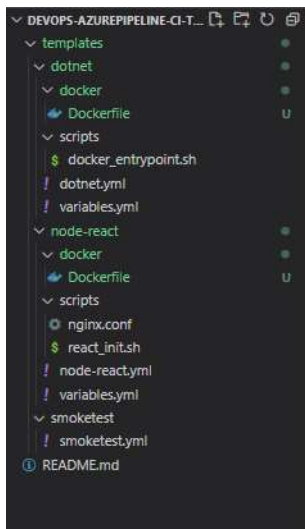
Use template from  
devops repository



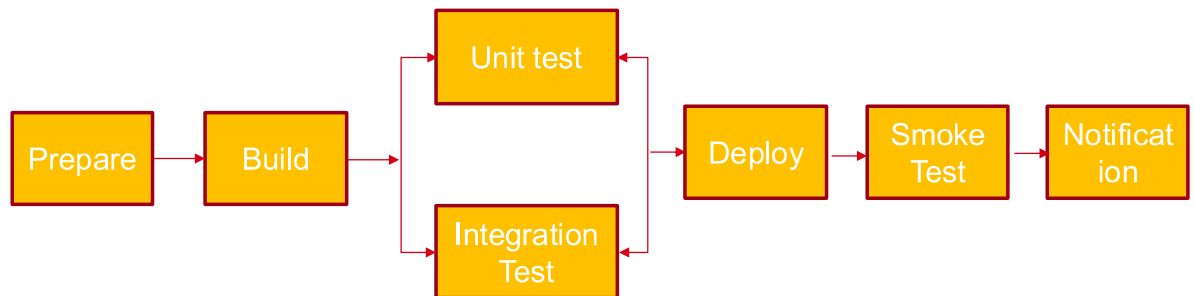
# Create template from DevOps repo

Devops repository

Directory template structure

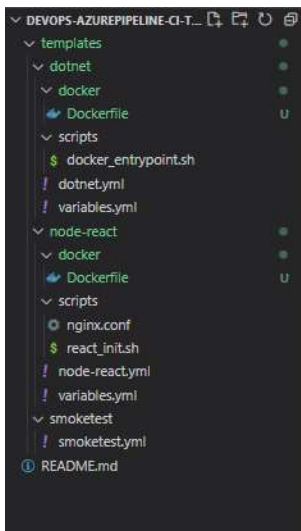


- Create a directory template structure
- Define CI pipelines base on per application
- Define CI/CD flow
  - For Ex: CI/CD flow



# How to using templates from DevOps repo

## cicd repository



## Application repository

```
resources:
  repositories:
  - repository: devops
    endpoint: azurerepo
    type: git
    name: DevOps/devops-azurepipeline-ci-template
    ref: refs/heads/feature/civ5
```

```
variables:
  - template: templates/dotnet/variables.yml@devops
```

```
stages:
  - template: templates/dotnet/dotnet.yml@devops
  - template: templates/smoketest/smoketest.yml@devops
```

```
15 pool:
16   vmImage: 'ubuntu-latest'
17
18 resources:
19   repositories:
20     - repository: devops
21       endpoint: azurerepo
22       type: git
23       name: DevOps/devops-azurepipeline-ci-template
24       ref: refs/heads/feature/civ5
25
26 parameters:
27   - name: skipBuild
28     displayName: Skip Build
29     type: boolean
30     default: false
31   - name: runUnitInt
32     displayName: Run Unit and Integration Test
33     type: boolean
34     default: false
35   - name: createRelease
36     displayName: Create Release
37     type: boolean
38     default: false
39   - name: runSmoke
40     displayName: Run Smoke Test
41     type: boolean
42     default: false
43
44 variables:
45   - template: templates/dotnet/variables.yml@devops
46   parameters:
47     runUnitInt: ${{ parameters.runUnitInt }}
48     skipBuild: ${{ parameters.skipBuild }}
49     createRelease: ${{ parameters.createRelease }}
50     smoke: ${{ parameters.runSmoke }}
51
52 stages:
53   - template: templates/dotnet/dotnet.yml@devops
54   - template: templates/smoketest/smoketest.yml@devops
```

**Practice to write a full  
pipeline for a service**



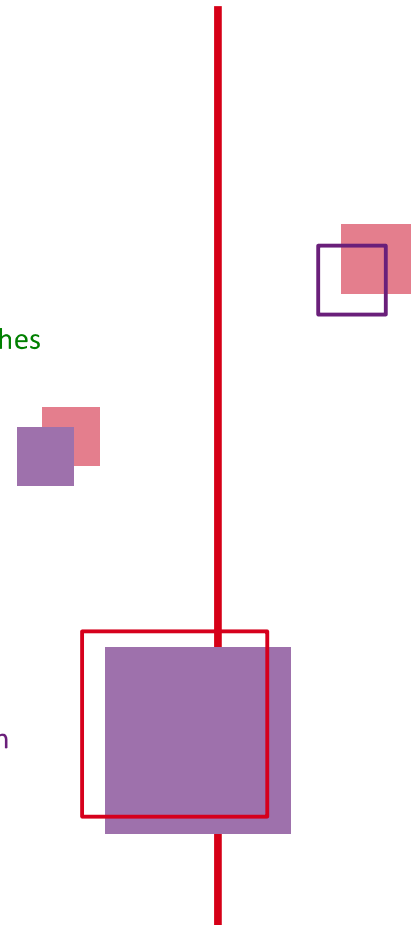
# Practice to write a full pipeline for a service

Write a pipeline with

- Only trigger when having a code change from main, release, and develop branches
- Install and build an app
- Run the unit test and integration test
- Build and Push Docker image to a registry
- Deploy the app to aks
- Notification build pipelines job to MS teams or Slack

Note:

- With PR, only trigger and run the install, build, and unit test
- With develop branch, run full pipelines
- With the release and main branches, run full pipelines except for the integration test
- Approval is only used when triggered from the main branch



**Q&A**



**Thank you**