DevOps

Week 04

Murtaza Munawar Fazal

Pipeline

- Business demands continuous delivery of value. Value is created only when a product is delivered to a satisfied customer.
- It's not created when one silo in the process is completed.
- The core idea is to create a repeatable, reliable, and incrementally-improving process for taking software from concept to customer.
- The goal is to enable a constant flow of changes into production via an automated software production line.
- Think of it as a pipeline. The pipeline breaks down the software delivery process into stages.
- Each stage aims to verify the quality of new features from a different angle to validate the new functionality and prevent errors from affecting your users.
- The pipeline should provide feedback to the team. Also, visibility into the changes flows to everyone involved in delivering the new feature(s).

Pipeline

- A delivery pipeline enables the flow of more minor changes more frequently, with a focus on flow.
- Your teams can concentrate on optimizing the delivery of changes that bring quantifiable value to the business.
- This approach leads teams to continuously monitor and learn where they're finding obstacles, resolve those issues, and gradually improve the pipeline's flow.
- As the process continues, the feedback loop provides new insights into new issues and barriers to be resolved.
- The pipeline is the focus of your continuous improvement loop.
- A typical pipeline will include the following stages:
  - build automation and continuous integration,
  - test automation and
  - deployment automation.

Build automation and continuous integration

- The pipeline starts by building the binaries to create the deliverables passed to the following stages. New features implemented by the developers are integrated into the central code base, built, and unit tested. It's the most direct feedback cycle that informs the development team about the health of their application code.

Test automation

- The new version of an application is rigorously tested throughout this stage to ensure that it meets all wished system qualities. It's crucial that all relevant aspects—whether functionality, security, performance, or compliance—are verified by the pipeline. The stage may involve different types of automated or (initially, at least) manual activities.

Deployment Automation

- A deployment is required every time the application is installed in an environment for testing, but the most critical moment for deployment automation is rollout time.
- Since the preceding stages have verified the overall quality of the system, It's a low-risk step.
- The deployment can be staged, with the new version being initially released to a subset of the production environment and monitored before being rolled out.
- The deployment is automated, allowing for the reliable delivery of new functionality to users within minutes if needed.

Azure Pipeline

- Azure Pipelines is a cloud service that automatically builds and tests your code project and makes it available to other users. It works with just about any language or project type.
- Azure Pipelines combines continuous integration (CI) and continuous delivery (CD) to test and build your code and ship it to any target constantly and consistently.

Azure Pipeline

- Languages
  - You can use many languages with Azure Pipelines, such as Python, Java, PHP, Ruby, C#, and Go.
- Version Control Systems
  - Azure Pipelines integrates with GitHub, GitLab, Azure Repos, Bitbucket, and Subversion.
- Application Types
  - You can use Azure Pipelines with most application types, such as Java, JavaScript, Python, .NET, PHP, Go, XCode, and C++.
- Deployment Targets
  - Container registries.
  - Virtual machines.
  - Azure services, or any on-premises or cloud target such:
    - Microsoft Azure.
    - Google Cloud.
    - Amazon Web Services (AWS).

CI / CD

- Continuous integration is used to automate tests and builds for your project. CI helps to catch bugs or issues early in the development cycle when they're easier and faster to fix. Items known as artifacts are produced from CI systems. The continuous delivery release pipelines use them to drive automatic deployments.
- Continuous delivery is used to automatically deploy and test code in multiple stages to help drive quality. Continuous integration systems produce deployable artifacts, which include infrastructure and apps. Automated release pipelines consume these artifacts to release new versions and fixes to the target of your choice.

Azure Pipelines Key Terms

- Agent
  - When your build or deployment runs, the system begins one or more jobs. An agent is installable software that runs a build or deployment job.
- Artifact
  - An artifact is a collection of files or packages published by a build. Artifacts are made available for the tasks, such as distribution or deployment.
- Build
  - A build represents one execution of a pipeline. It collects the logs associated with running the steps and the test results.
- Continuous delivery
  - Continuous delivery (CD) (also known as Continuous Deployment) is a process by which code is built, tested, and deployed to one or more test and production stages. Deploying and testing in multiple stages helps drive quality. Continuous integration systems produce deployable artifacts, which include infrastructure and apps. Automated release pipelines consume these artifacts to release new versions and fix existing systems. Monitoring and alerting systems constantly run to drive visibility into the entire CD process. This process ensures that errors are caught often and early.
- Continuous Integration
  - Continuous integration (CI) is the practice used by development teams to simplify the testing and building of code. CI helps to catch bugs or problems early in the development cycle, making them more accessible and faster to fix. Automated tests and builds are run as part of the CI process. The process can run on a schedule, whenever code is pushed, or both. Items known as artifacts are produced from CI systems. The continuous delivery release pipelines use them to drive automatic deployments.
- Deployment Targets
  - A deployment target is a virtual machine, container, web app, or any service used to host the developed application. A pipeline might deploy the app to one or more deployment targets after the build is completed and tests are run.

Azure Pipelines Key Terms

- Job
  - A build contains one or more jobs. Most jobs run on an agent. A job represents an execution boundary of a set of steps. All the steps run together on the same agent. For example, you might build two configurations - x86 and x64. In this case, you have one build and two jobs.
- Pipeline
  - A pipeline defines the continuous integration and deployment process for your app. It's made up of steps called tasks. It can be thought of as a script that describes how your test, build, and deployment steps are run.
- Release
  - When you use the visual designer, you can create a release or a build pipeline. A release is a term used to describe one execution of a release pipeline. It's made up of deployments to multiple stages.
- Stage
  - Stages are the primary divisions in a pipeline: "build the app," "run integration tests," and "deploy to user acceptance testing" are good examples of stages.
- Task
  - A task is the building block of a pipeline. For example, a build pipeline might consist of build and test tasks. A release pipeline consists of deployment tasks. Each task runs a specific job in the pipeline.
- Trigger
  - A trigger is set up to tell the pipeline when to run. You can configure a pipeline to run upon a push to a repository at scheduled times or upon completing another build. All these actions are known as triggers.

Continuous Integration (CI)

- Continuous integration (CI) is the process of automating the build and testing of code every time a team member commits changes to version control.
- CI encourages developers to share their code and unit tests by merging their changes into a shared version control repository after every small task completion.
- Committing code triggers an automated build system to grab the latest code from the shared repository and build, test, and validate the entire main branch (also known as the trunk or main).

Continuous Integration (CI)

- The idea is to minimize the cost of integration by making it an early consideration.
- Developers can discover conflicts at the boundaries between new and existing code early, while conflicts are still relatively easy to reconcile.
- Once the conflict is resolved, work can continue with confidence that the new code honors the requirements of the existing codebase.
- Integrating code frequently doesn't offer any guarantees about the quality of the new code or functionality.
- In many organizations, integration is costly because manual processes ensure that the code meets standards, introduces bugs, and breaks existing functionality.

Continuous Integration (CI)

- In practice, continuous integration relies on robust test suites and an automated system to run those tests to address this friction within the integration process.
- When a developer merges code into the main repository, automated processes kick off a build of the new code.
- Afterward, test suites are run against the new build to check whether any integration problems were introduced.
- If either the build or the test phase fails, the team is alerted to work to fix the build.
- The end goal of continuous integration is to make integration a simple, repeatable process part of the everyday development workflow to reduce integration costs and respond to early defects.
- Working to make sure the system is robust, automated, and fast while cultivating a team culture that encourages frequent iteration and responsiveness to build issues is fundamental to the strategy's success.

Four pillars of continuous integration

- Version Control System
  - Git
  - Apache Subversion
  - Team Foundation Version Control
- A package management system is used to install, uninstall, and manage software packages.
  - NuGet
  - Node Package Manager (NPM)
  - Chocolatey
  - HomeBrew
  - RPM
- A continuous integration system merges all developer working copies to shared mainline several times a day.
  - Azure DevOps
  - TeamCity
  - Jenkins
- An automated build process creates a software build, including compiling, packaging, and running automated tests.
  - Apache Ant
  - NAnt
  - Gradle

Demo

AB Agile Planning and Por...

Overview

Summary

Dashboards

Wiki

Boards

Repos

Pipelines

Test Plans

Artifacts

Project settings

AB Agile Planning and Portfolio Management with Azure Boards

Private | Invite

About this project

Like 0

Generated by Azure DevOps Demo Generator

Languages

JavaScript CSS

Project stats

Period: Last 7 days

Boards

0 Work items created

0 Work items completed

Repos

0 Pull requests opened

2 Commits by 1 authors

Pipelines

100% Builds succeeded

0% Deployments succeeded

Members 1

Source control types supported by Azure Pipelines

Repository type	Azure Pipelines (YAML)	Azure Pipelines (classic editor)
Azure Repos Git	Yes	Yes
Azure Repos TFVC	No	Yes
Bitbucket Cloud	Yes	Yes
Other Git (generic)	No	Yes
GitHub	Yes	Yes
GitHub Enterprise Server	Yes	Yes
Subversion		Yes

Microsoft Hosted Agent

- If your pipelines are in Azure Pipelines, then you've got a convenient option to build and deploy using a Microsoft-hosted agent.
- With a Microsoft-hosted agent, maintenance and upgrades are automatically done.
- Each time a pipeline is run, a new virtual machine (instance) is provided. The virtual machine is discarded after one use.
- For many teams, this is the simplest way to build and deploy.
- You can try it first and see if it works for your build or deployment. If not, you can use a self-hosted agent.
- A Microsoft-hosted agent has job time limits.

Self Hosted Agents

- An agent that you set up and manage on your own to run build and deployment jobs is a self-hosted agent.
- You can use a self-hosted agent in Azure Pipelines. A self-hosted agent gives you more control to install dependent software needed for your builds and deployments.
- You can install the agent on:
  - Linux.
  - macOS.
  - Windows.
  - Linux Docker containers.
- After you've installed the agent on a machine, you can install any other software on that machine as required by your build or deployment jobs.
- A self-hosted agent doesn't have job time limits.

Job Types

- In Azure DevOps, there are four types of jobs available:
  - Agent pool jobs.
  - Container jobs.
  - Deployment group jobs.
  - Agentless jobs.

Agent Pools

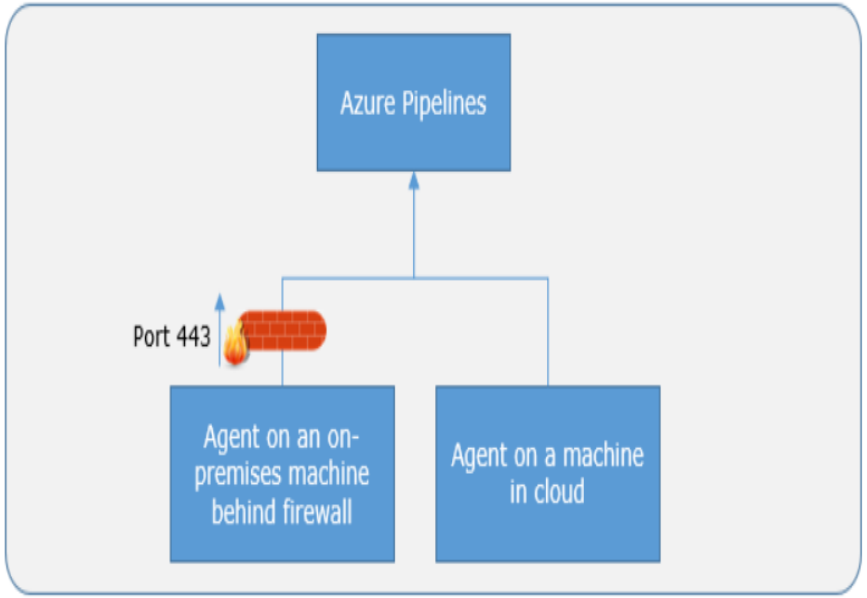
- Instead of managing each agent individually, you organize agents into agent pools. An agent pool defines the sharing boundary for all agents in that pool.
- In Azure Pipelines, pools are scoped to the entire organization so that you can share the agent machines across projects.
- If you create an Agent pool for a specific project, only that project can use the pool until you add the project pool into another project.
- When creating a build or release pipeline, you can specify which pool it uses, organization, or project scope.
- Pools scoped to a project can only use them across build and release pipelines within a project.
- To share an agent pool with multiple projects, use an organization scope agent pool and add them in each of those projects, add an existing agent pool, and choose the organization agent pool. If you create a new agent pool, you can automatically grant access permission to all pipelines.

Predefined Agent Pool

- Azure Pipelines provides a pre-defined agent pool-named Azure Pipelines with Microsoft-hosted agents.
- It will often be an easy way to run jobs without configuring build infrastructure.
- The following virtual machine images are provided by default:
  - Windows Server 2022 with Visual Studio 2022.
  - Windows Server 2019 with Visual Studio 2019.
  - Ubuntu 20.04.
  - Ubuntu 18.04.
  - macOS 11 Big Sur.
  - macOS X Catalina 10.15.

Communicate with Azure Pipelines

- The agent communicates with Azure Pipelines to determine which job it needs to run and report the logs and job status.
- The agent always starts this communication. All the messages from the agent to Azure Pipelines over HTTPS, depending on how you configure the agent.
- This pull model allows the agent to be configured in different topologies



Communicate with Azure Pipelines

- The user registers an agent with Azure Pipelines by adding it to an agent pool. You've to be an agent pool administrator to register an agent in that pool. The identity of the agent pool administrator is needed only at the time of registration.
- Once the registration is complete, the agent downloads a listener OAuth token and uses it to listen to the job queue.
- Periodically, the agent checks to see if a new job request has been posted in the job queue in Azure Pipelines. The agent downloads the job and a job-specific OAuth token when a job is available. This token is generated by Azure Pipelines for the scoped identity specified in the pipeline. That token is short-lived and is used by the agent to access resources (for example, source code) or modify resources (for example, upload test results) on Azure Pipelines within that job.
- Once the job is completed, the agent discards the job-specific OAuth token and checks if there's a new job request using the listener OAuth token.
- The payload of the messages exchanged between the agent, and Azure Pipelines are secured using asymmetric encryption. Each agent has a public-private key pair, and the public key is exchanged with the server during registration.
- The server uses the public key to encrypt the job's payload before sending it to the agent. The agent decrypts the job content using its private key. Secrets stored in build pipelines, release pipelines, or variable groups are secured when exchanged with the agent.

Demo

https://murtaza-fazal.visualstudio.com/Agile%20Planning%20and%20Portfolio%20Management%20with%20Azure%20Boards

Perseus | Personal | MCT | Perseus | Other favorites

Azure DevOps murtaza-fazal / Agile Planning and Portfolio... / Overview / Summary

Search

MF

AB Agile Planning and Por...

- Overview
- Summary
- Dashboards
- Wiki
- Boards
- Repos
- Pipelines
- Test Plans
- Artifacts

Project settings

AB Agile Planning and Portfolio Management with Azure Boards

Private | Invite

About this project

Like 0

Generated by Azure DevOps Demo Generator

Languages

JavaScript CSS

Pipelines
- Pipelines
- Environments
- Releases
- Library
- Task groups
- Deployment groups

Project stats

Period: Last 7 days

Boards

0 Work items created

0 Work items completed

Repos

0 Pull requests opened

3 Commits by 1 authors

Pipelines

100% Builds succeeded

0% Deployments succeeded

Members 1

https://murtaza-fazal.visualstudio.com/Agile Planning and Portfolio Management with Azure Boards/_build

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

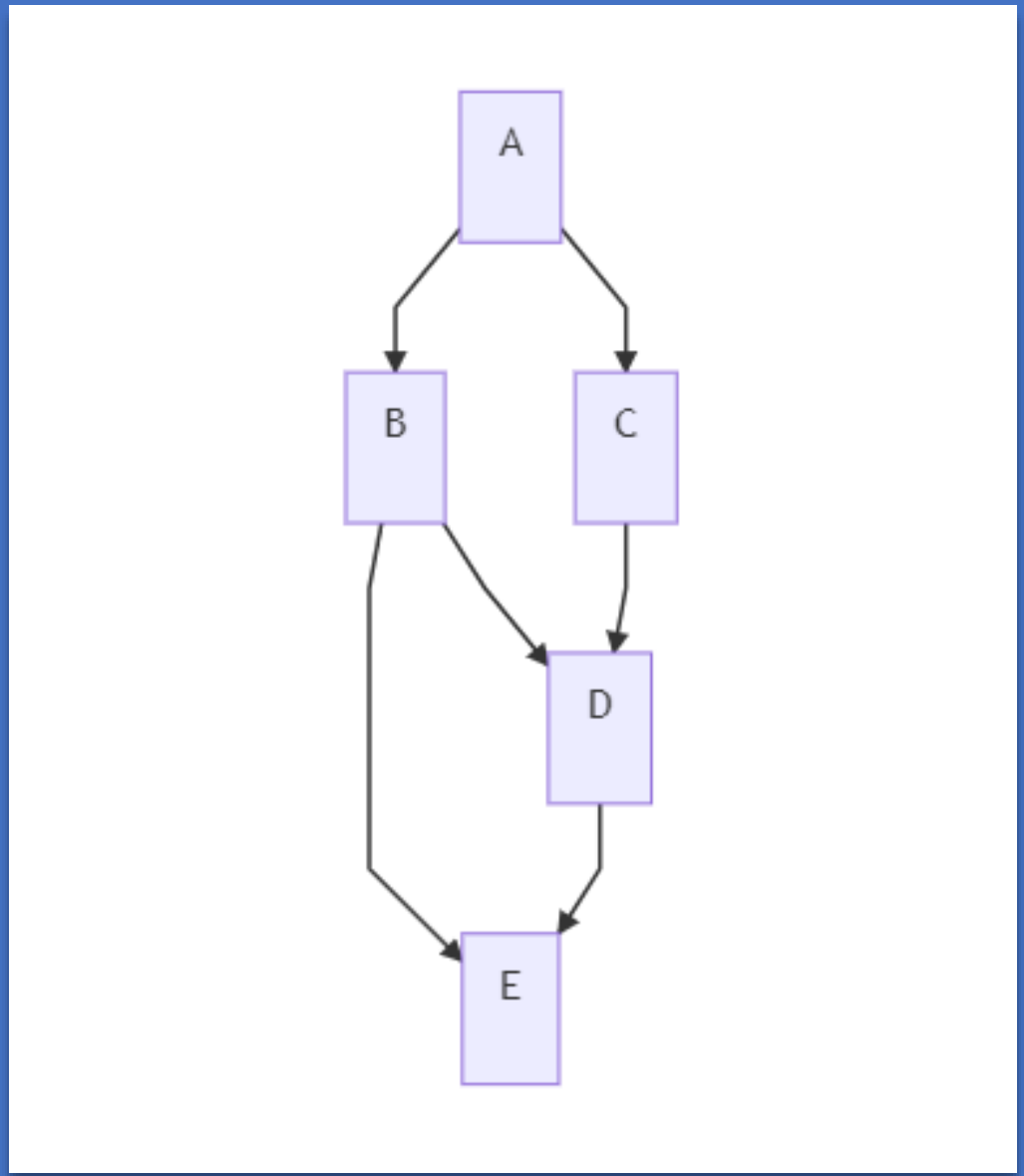
```
jobs:
- job: A
  steps:
  - script: echo' job A.'

- 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)

Demo

Implementing GitHub Actions for CI/CD

https://github.com/MicrosoftLearning/eShopOnWeb

Perseus | Personal | MCT | Perseus | Other favorite

Search or jump to... / Pull requests Issues Codespaces Marketplace Explore

MicrosoftLearning / eShopOnWeb Public

generated from MicrosoftLearning/INF99X-SampleCourse

Watch 3 | Fork 77 | Star 33

Code | Issues | Pull requests 3 | Actions | Projects | Security | Insights

main | 2 branches | 0 tags | Go to file | Add file | Code

LuizMacedo Merge pull request #51 from peterschoellhorn/patch-1 ... 94113b9 3 weeks ago 56 commits

.ado	Update eshoponweb-ci-dockercompose.yml	last month
.azure/bicep	add and update files to support docker webapp deployment	2 months ago
.devcontainer	initial commit	4 months ago
.github/workflows	fix: typo in RESOURCE GROUP env var	3 weeks ago
.images	initial commit	4 months ago
.vscode	initial commit	4 months ago
src	Revert "Bump Microsoft.AspNetCore.Components.WebAssembly.DevSer...	3 months ago
tests	Revert "Bump Microsoft.EntityFrameworkCore.InMemory from 6.0.7 to 7....	3 months ago
.dockerignore	initial commit	4 months ago
.editorconfig	initial commit	4 months ago
.gitignore	initial commit	4 months ago
CodeCoverage.runsettings	initial commit	4 months ago
LICENSE	initial commit	4 months ago
MTT-Notes.md	Update MTT-Notes.md	4 months ago
README.md	initial commit	4 months ago
docker-compose-webapp.yml	initial commit	4 months ago

About

Repository maintained by AZ-400 course and Learn content community. Project used for AZ-400 Labs. Forked from: https://github.com/dotnet-architecture/eShopOnWeb Sample - ASP.NET Core 6.0 reference application, powered by Microsoft, demonstrating a layered application architecture with monolithic deployment model.

github devops pipelines github-actions bicep az-400

Readme

MIT license

33 stars

3 watching

77 forks

Releases

No releases published

Packages

No packages published