azure dev ops

 Summary of Azure Dev Ops

Recall

- What is Azure DevOps?
- What are the services it offers?

Overview

> 📘 Azure Dev Ops is a SAAS platform from microsoft built to implement all of your DevOps processes for projects. It is a tool that gives you features to implement your custom workflow.

DevOps is many things, a combination of concepts and tools. Anything that makes developing and releasing applications fast, automated, and high-quality. It is used in order to achieve an efficient workflow.

The goal is to make the software development lifecycle as simple and efficient as possible through automation.

Azure DevOps aims to provide features for every part of the software development lifecycle

- Azure Boards
- Azure Pipelines
- Azure Repos
- Azure Test Plans
- Azure Artifacts

Boards

> 📘 Planning is the first step of the software development lifecycle.

This is what boards are often used for. Defining what needs to be developed and why. Deciding team roles and workflows. You can customize your board based on what workflow you are going for.

You can create tasks for your application in the form of features, stories, bugs, etc. You can also assign these to team members, create discussions within a task, and track their progress.

Repos

> 📘 The application code that is developed is a major part of the lifecycle.

Azure Repos are used to host that code. Azure repos are a `source control management system` like Github, and has similar features.

- Code Review
- Pull Requests
- Branch Policies

It uses `git`, and the corresponding workflow. The repository branches and pull requests can be linked back to the tasks they correspond to.

Pipelines

Releasing the completed code. In order to do this, the code must be `tested`, and packaged into an `artifact`.

Pipelines script can be written in YAML, and included in your repository.

> 📘 Azure pipelines work with any language, platform , or cloud.

Build Pipelines (CI)

The process of testing and building the project is done by an automated `CI (Continuous Integration)` process. This is what the build pipelines are for.

- Testing code changes and producing an artifact for deployment

> 📘 The smallest and main building block of Azure pipelines are steps.

For example, if you want to test and package an application, you may have steps to:

- run tests
- package applications

- build a (docker) image
- push image to a repository for later deployment

Each step will execute a certain command.

Steps can be written as scripts.

```
trigger:
- master

pool:
  vmImage: 'ubuntu-22.04'

variables:
  buildConfiguration: 'Release'

steps:
- script: dotnet test
  displayName: Run unit tests

- script: dotnet build --configuration $(buildConfiguration)
  displayName: Build application

- script: docker build -t my-image:v1.0 .
  displayName: Build image

- script: docker push my-image:v1.0
  displayName: Push image
```

Steps can also be created as tasks.

A task (in pipeline context) is a configurable way to create commands, rather than writing them yourself. You select the task from the side menu and customize the input fields to fit what you need. Once you add a task it will automatically be added to your YAML code.

- A pre-created script offered for convenience

Pipelines can become complex, spanning different machines or OS. These will require multiple `jobs`. Many times, a job is used to perform the pipeline in a different environment. These are also know as agents.

- An agent is a computing infrastructure that will run the pipeline.
    - They are selected from a pool (windows machines, linux machines, etc.) They define what OS and version the tasks will be run on.

Another common use case for jobs is if there is a set of steps that will run in parallel to save time.

> 📘 A job is a collection of steps.

```
trigger:
- master

variables:
  buildConfiguration: 'Release'

jobs:
  - job: Run on Windows
    pool:
      vmImage: 'windows-latest'
    steps:
    - script: dotnet test
      displayName: Run unit tests

  - job: Run on Linux
    pool:
      vmImage: 'ubuntu-latest'
    steps:
      - script: dotnet
        displayName: Run unit tests
```

Stages in Azure Pipelines

Test → Build image → Push to repo → Deploy

The fully automated process of automatically deploying code changes to production environment. There are two main parts to this, the build stage and the deploy stage. These run one after the other and are reflected in the YAML file. There is also a specific job type for deployment that can be used.

```
stages:
- stage: Build
  jobs:
    - job: <...>
```

```
- stage: Deploy
  jobs:
    - deployment: <...>
```

During deployment, the code is usually deployed to an intermediate environment (DEV) where it is tested extensively while being gradually promoted to the production environment. All of these stages can be reflected in the pipeline YAML.

- Deploy to Developing
- Deploy to Testing
- Deploy to Production

Templates in Azure Pipelines

Templates are used to avoid writing the same pipeline configuration for each application. Instead we can write it once as a template, then reuse it later.

> 📘 Templates are a separate file and can be referenced using a template attribute. These can even be configured with parameters.

You can split your entire pipeline into individual files, which can be stored and managed in a repository. You can have a template for either a stage, a job, or a step.

Environments in Azure Pipelines

When you have multiple environments for multiple applications, it may be hard to keep track of all of your separate environment activities. These environments can be managed and accessed through the Azure Environments Section withing Pipelines. You can target environment names in the Pipeline file, and view the history of each environment. You can also link deployments back to the original task.

Release Pipelines (CD)

`Continuous delivery`

This deployment pipelines can be built as a separate Release Pipeline. Some companies use one pipeline for the entire process, some split it into two pipelines: build

and release. You can select a build pipeline for an artifact as the input for the release pipeline. These release pipelines can only be built using UI, not YAML.

> 📘 It is a better practice to only use one CI/CD pipeline, defined in YAML, rather than two separate.

Artifacts

Depending on the programming language, the artifact produced by the pipeline will be different. For storing these artificats there is a feature called Azure Artifacts. This is used for storing either Maven packages, Nuget packages, or Npm packages.

However in the modern software world, these types of artifacts are often not used. Instead, `docker images` are produced. These types of containers are convenient because they are language/tool independent.

Docker images are stored in a `Container Registry`.

Test Plans

Testing the application before deployment is essential.

In Azure Dev Ops there is an entire section for test-management. Here you can have a centralized view of all test cases.

You can create manual test cases/plans (specifying test steps and predicted outcome), or automated tests (running test within azure pipelines) with viewable test reports.

Azure DevOps Architecture

At the core, there are the Azure Dev Ops services

- Repos, boards, tests, pipelines, planning, etc.

The pipelines themselves run on separate machines called `agents`, which are connected to the services. These agents are connected to microsoft, but you have the option to configure your own agents.

Pricing Model

There is a free tier with certain restrictions.

Paid tiers reduce restrictions.

Service Connections

In the DevOps process we often need to connect and interact with other platforms.

You may want to use other options rather than all of Azure’s services.

- Examples
  - Pushing docker image to a container repo
  - Deploying to a remote server
  - Hosting the project on Github rather than Azure repos

In all these cases, Azure DevOps needs to connect to these other platforms. This is where the service connections feature is used. It is secure, and allows you to create credentials to use when connecting to these outside services.

These can be created in the project settings section.

Comparison to Other Tools

The main difference in comparison with other tools is that Azure DevOps is comprehensive, it strives to cover the entire DevOps process, rather than specializing in a single step of it. This removes the need to integrate multiple tools together.

A direct comparison to Azure DevOps is GitLab. It also covers the DevOps processes. AWS does this as well, but AWS covers many many more services. In AWS there is only one account for all services, while in Azure you need a separate account for the cloud services. They are still connected though, as they are both under Microsoft.

> 📘 Companies who are already using Microsoft tools such as the cloud services are more likely to use Azure DevOps. However, AWS is still the far more popular cloud services tool, and GitLab is the more popular DevOps tool due to it being open source.

It is important to think about what tools you will most likely be working with, when deciding what tools to learn.