Implement and Manage Azure Pipelines Infrastructure

UNDERSTANDING AZURE PIPELINES AGENTS
Overview

Understanding Azure Pipelines Agents
Microsoft Hosted vs Self-Hosted Agents
Implementing Self-Hosted Agents
Leveraging Docker in Azure Pipelines
Overview

Understanding Pipeline Jobs
Running Pipeline Jobs
Developing Azure Pipeline Jobs
Exploring Azure Pipeline Jobs
Integrating Third-Party Platforms
Understanding Pipeline Jobs
What are Pipeline Jobs?

The smallest unit of organisation in a pipeline

Consists of a series of steps & can be combined in to stages

Can be run across a range of different compute platforms

https://docs.microsoft.com/en-us/azure/devops/pipelines/process/phases
job: Sample_Job

timeoutInMinutes: 10

pool:
  vmImage: 'ubuntu-16.04'

steps:
- bash: echo "Hello world"

- Use `job:` when you want to provide additional properties like `timeoutInMinutes`.

- `pool` and `vmImage` are needed when you want to run the job against a Hosted Agent.

- `steps:` consist of multiple discrete actions, like processing a Bash script on the agent which is running the job.
Running Pipeline Jobs
Agent Pool Jobs

Run on a dedicated or assigned system contained within a pool

The capabilities of the system determine the jobs which can be run

Jobs can only be run if the pool has an agent available

https://docs.microsoft.com/en-us/azure/devops/pipelines/process/phases#agent-pool-jobs
Server Jobs

Jobs are executed directly on the Azure DevOps (or TFS) server

Jobs are executed without an agent, so range of jobs are limited

Use ‘pool: server’ or ‘server: true’ to use server jobs

https://docs.microsoft.com/en-us/azure/devops/pipelines/process/phases#server-jobs
Using Agent Demands

- Demands can be asserted manually or automatically.
- Multiple demands can be specified for each job.
- Linked to operating system, applications and versions.
- Specifies what capabilities the agent must have.

[Link to Azure DevOps documentation](https://docs.microsoft.com/en-us/azure/devops/pipelines/process/demands)
pool:

  name: privatePool

demands:
  - agent.os -equals Linux
  - python3 -equals /usr/bin/python3

steps:
  - task: PythonScript@0

inputs:
  scriptSource: inline
  script: print("Hello, World!")

- Specify the name of the private pool
- Specify multiple demands (if the task does not automatically assert demands)
- Create a job which utilizes the asserted demands
Container Jobs

- Jobs can run inside a Docker container on Windows and Linux agents.
- Provides more control over the job execution environment.
- Images can be retrieved from Docker Hub or private registries.

https://docs.microsoft.com/en-us/azure/devops/pipelines/process/container-phases
Developing Azure Pipeline Jobs
Build/Release Pipelines
Manual job addition Useful for learning Underlying YAML exposed

Using the Classic UI

Agent job 1
- Run on agent

Use NuGet 4.4.1
- NuGet Tool Installer

NuGet restore
- NuGet

Build solution **\*.sln
- Visual Studio Build

VsTest - testAssemblies
- Visual Studio Test

Azure App Service Deploy:
- Some settings need attention

Publish symbols path
- Index Sources & Publish Symbols

Publish Artifact: drop
- Publish Build Artifacts
Unified CI/CD Pipelines
Targeted at more modern platforms
UI offers drag-and-drop plus IntelliSense

Using YAML Pipelines

```yaml
master

# ASP.NET Core
# Build and test ASP.NET Core projects targeting .NET Core.
# Add steps that run tests, create a NuGet package, deploy, and more:
# https://docs.microsoft.com/azure/devops/pipelines/languages/dotnet-core

trigger:
  - master

pool:
  - vmImage: 'ubuntu-16.04'

variables:
  - buildConfiguration: 'Release'

steps:
  - task: DotNetCoreInstall@0
    inputs: {}
    version: '2.1.300'

  - task: DotNetCoreCLI2@2
    inputs: {}
    command: restore
    projects: '**/*.csproj'
    feed.azureDevOps: config
    nugetConfiguration: NuGet.config -- Relative to root of the repository
    externalFeedCredentials: <Name of the NuGet service connection>

  - task: DotNetCoreCLI2@2
    displayName: Build
    inputs: {}
    command: build
    projects: '**/*.csproj
    arguments: '--configuration Release'
```

Classic UI vs YAML Pipelines

**Classic UI**

- Build and Release Pipelines are separate
- Release pipelines require build artifacts
- Suitable for more mature platforms
- Cannot be managed via source control
- Does not support container jobs
- Will slowly be phased out

**YAML Pipelines**

- Multi-stage Pipelines enable unified CI/CD
- Build artifacts are not necessary
- Suitable for more modern platforms
- Managed via source control
- Only way to run container jobs
- Will slowly become the only approach
Demo

Explore pipelines using the Classic UI
Explore pipelines using the YAML editor
Integrating Third-party Platforms
Extending Azure Pipelines Functionality

Azure DevOps is extensible via the Visual Studio Marketplace

Allows for integration with external and pre-existing platforms

Enables Azure DevOps to be part of an integrated CI/CD framework

# Deploy to Chef environments by editing environment attributes

- task: Chef@1

  inputs:

  connectedServiceName: "
  environment: 'dev'
  attributes: 'something'
  chefWaitTime: '30'

- Uses the standard task syntax

- Name of the connected service endpoint

- Task inputs which are only meaningful to the remote service
Summary

Understanding Pipeline Jobs
Running Pipeline Jobs
Developing Azure Pipeline Jobs
Exploring Azure Pipeline Jobs
Integrating Third-party Platforms
Coming next:
Microsoft Hosted vs. Self-hosted Agents
Continuous Delivery and DevOps with Azure DevOps: Pipelines
Outline

- Release management in the context of continuous delivery
- Steps of a release
- Release management concepts
- Release management infrastructure
- Deploying to on-premise or cloud
Release Management in the Context of Continuous Delivery
In continuous delivery we strive to separate a deployment from a release.

This provides better stability to the deployment, better validation of the deployment and makes releasing a feature a functional operation, preferably done by the “business” at the moment they prefer.
Separating Deployment from Release

Deploy your product
See if it operates in a stable fashion
Enable your feature for
  - Segment of users
  - Random selected percentage
  - All in one
Watch how the system behaves
This is a safe and stable way of deploying. It separates deployments from revealing a feature; you can now deploy any time a day!
Steps of a Release
Steps of a Release

First you build the software
- And validate product quality

Then you deploy the software
- And validate runtime stability

Then you release the feature
- And validate feature usage

Clean separation in the different stages of delivery
Azure DevOps Release Pipelines

Distinction between build and release

Build produces artifacts
  - Mix and match with other build software

A release pipeline picks these up and deploys them in an environment

End-to-end traceable process
Release Management Concepts
How a Release Is Set Up

- Release Pipeline Definition
- Release Agent
Artifacts, Stages, and Gates

Artifacts:
- WebSiteBinaries
- DockerImages
- Schedule: not set

Stages:
- Initial Stage: 1 job, 1 task
- Parallel Stage: 1 job, 0 task
- Validation Stage: 1 job, 0 task
Jobs and Tasks

All pipelines > New release pipeline

SmokeTest
Deployment process

Agent job
Run on agent

Add tasks
Search

.NET Core
Build, test, package, or publish a dotnet application, or run a custom dotnet command. For package commands, supports NuGet.org and authenticated feeds like Package Management and MyGet.

.NET Core SDK Installer
Acquires a specific version of the .NET Core SDK from internet or the local cache and adds it to the PATH. Use this task to change the version of .NET Core used in subsequent tasks.

Android Signing
Sign and align Android APK files

Ant
Build with Apache Ant
Release Variables

Custom Variables
$(variablename)
Build In Variables
Secrets
Environment Variable
Continuous Deployment Trigger

Build, (Azure DevOps) Git,
Team Foundation Version Control
GitHub
Jenkins
Azure DevOps Artifact Management
Container Registry
Docker Hub
Release Management Infrastructure
Agents and Pipelines

- Hosted Agent
- Pipelines
- Custom Agent
Hybrid Release Management
Deploying to On-premise or Cloud
Deployment Groups

Provisioned per project or for multiple teams
- Deployment Pool
- Deployment Group

Requires Agent install per machine that is part of the pool/group

Agent runs as system service

Primarily used for on-premises hosting
Create a release for an ASP.NET web application
Demo

Deploy to a deployment group
Demo

Set up a custom agent
Summary

Release management in the context of continuous delivery

Steps of a release

Release management concepts

Release management infrastructure

Deploying to on-premise or cloud
Infrastructure as Code
Outline

- Infrastructure as code
- Provisioning on demand
- Using Containers
- Yaml based pipelines
Infrastructure as Code
Infrastructure as Code

- infrastructure defined in text files checked in your version control system
- Provision on demand
- From environments to quality gates
- Improves traceability of changes
- Improves repeatability
- Improves cost efficiency
Configuration and Secrets

Have admin define secrets in variables
Use the release to replace secrets
Use transform tasks
Provisioning on Demand
Provisioning

Prepare the environment on which we can deploy our new version of the software

Azure ARM
Template
Parameters
Create new appservice as part of the release
Test, Validation, and Approval

- We deploy to the new provisioned resource
- We run various tests
- We report the results
- We de-provision resources
- We wait on approval
Demo

Deploy to the provisioned appservice and test the product using UI tests
Using Containers
Container Workflow

Build “bakes” the container(s)

Deployment involves only command to target machine

Target machine takes care of the work. Often a cluster, e.g. Kubernetes
Deploy our website to a kubernetes cluster
Yaml Based Pipelines

- Build pipeline
- Release pipeline
- Combined pipeline experience
- Improved traceability
- Environments
Demo

Integrated Yaml pipelines and environments
Summary

Infrastructure as code
Provisioning on demand
Using Containers
Yaml based pipelines
Security, Approval, and Audit Trails
Outline

- Release pipeline security
- Audit trails and logs
- Four eyes principle
Release Pipeline Security
Securable Parts of a Release

- Release Definition
- Release Stages
- Agents & Queues
- Logs & Audit Trails
Demo

Set up release security
Audit Trails and Logs
What Information Gets Logged?

Release runs
Every step in a release is logged
Every approval is logged
Changes to release definition
Access to the Azure DevOps services
Four Eyes Principle
The four eyes principle is a requirement that two individuals approve some action before it can be taken.

http://whatis.techtarget.com/definition/four-eyes-principle
Demo

Implement 4 eyes principle
Leveraging Docker in Azure Pipelines
Overview

Understanding Docker with Azure Pipelines
How to use Docker within Azure Pipelines
Implement Self-hosted Docker agent
Deploy a container-based solution
Understanding Docker with Azure Pipelines
Why Use Docker?

Provides more control over where and how jobs are run

Enables a strategy of testing builds across different OS versions

Decouples application dependencies from supporting host system
Ways of Using Docker

Docker tasks
Agent executes docker binary to run Pipeline jobs

Container jobs
Job is executed within nominated container image

Docker agent
Agent runs in a container on an agentless system
Understanding Docker Tasks

Agent acts as the Docker host (Hosted Linux and VS2017)

Functionality exposed by native tasks or docker binary (script)

Cached containers are not persisted on Microsoft-hosted agents

https://docs.microsoft.com/en-us/azure/devops/pipelines/languages/docker
Understanding Container Jobs

Agent acts as the Docker host (Hosted Linux and VS2017)

Functionality exposed by native tasks or docker binary (script)

Cached containers are not persisted on Microsoft-hosted agents

https://docs.microsoft.com/en-us/azure/devops/pipelines/process/container-phases
Understanding the Docker Agent

- Agent runs within the container, Docker host is agentless
- Agent runs on Windows Server Core or Ubuntu container images
- Containers run on self-hosted system or Azure Container Instances (ACI)

https://docs.microsoft.com/en-us/azure/devops/pipelines/agents/docker
Use Docker with Azure Pipelines
Implement a Self-hosted Docker agent
Verify agent functionality
Demo

Create a container-based solution
Deploy and verify solution
Summary

Understanding Docker with Azure Pipelines
How to use Docker within Azure Pipelines
Implement Self-hosted Docker agent
Deploy a container-based solution
Overview

Understanding Azure Pipelines Agents
Microsoft Hosted vs. Self-hosted Agents
Implementing Self-hosted Agents
Leveraging Docker in Azure Pipelines