Take control of your deployments with Azure Pipelines YAML

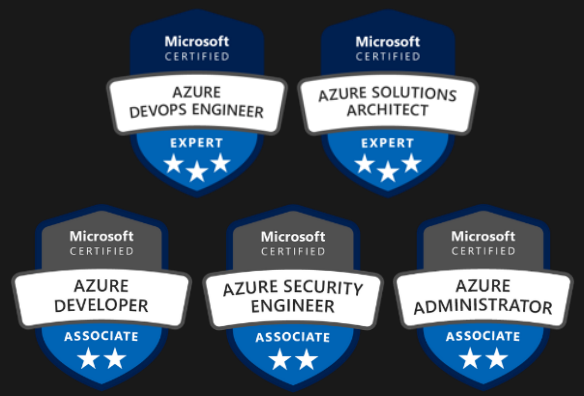
Pasi Huuhka

Pasi Huuhka

DevOps Architect

pasi.huuhka@zure.com

- DevOps expert & Developer from Finland
- Working on Azure since 2014
- Helped to develop & automate applications on Azure for 20+ customers from startups to enterprises



- Twitter: @DrBushyTop
- Blog: huuhka.net
- zure.ly/pasi-huuhka

- zure.ly/faug
- zure.ly/gdbc-2020

100%
Azure since 2011

52 / 55
experts

14,2
experience avg.

4,6 / 5
customer satisfaction

4
Azure MVPs

2
Offices

Microsoft Partner
Microsoft

Gold Application Development
Gold Cloud Platform
Gold Data Analytics
Gold Data Platform
Gold DevOps

MVP
Microsoft® Most Valuable Professional

Microsoft CERTIFIED Trainer

Partner Seller
Microsoft

Microsoft Partner
2019 Partner of the Year Finalist
Application Innovation Award

Overview of the session

- You will learn how Azure Pipelines function, and how YAML pipelines improve on the Classic approach
- You will also get to know the benefits of pipelines as code, and understand how to take advantage of them
- We will take a look at how Azure Pipelines YAML helps you control the deployments of an application and the whole Azure DevOps organization
- Best practices, demos, resources to get started.

Pipeline Basics

Introducing Azure DevOps

Azure Boards

Plan, track, and discuss work across teams, deliver value to your users faster.

Azure Repos

Unlimited cloud-hosted private Git repos. Collaborative pull requests, advanced file management, and more.

Azure Pipelines

CI/CD that works with any language, platform, and cloud. Connect to GitHub or any Git provider and deploy continuously to any cloud.

Azure Test Plans

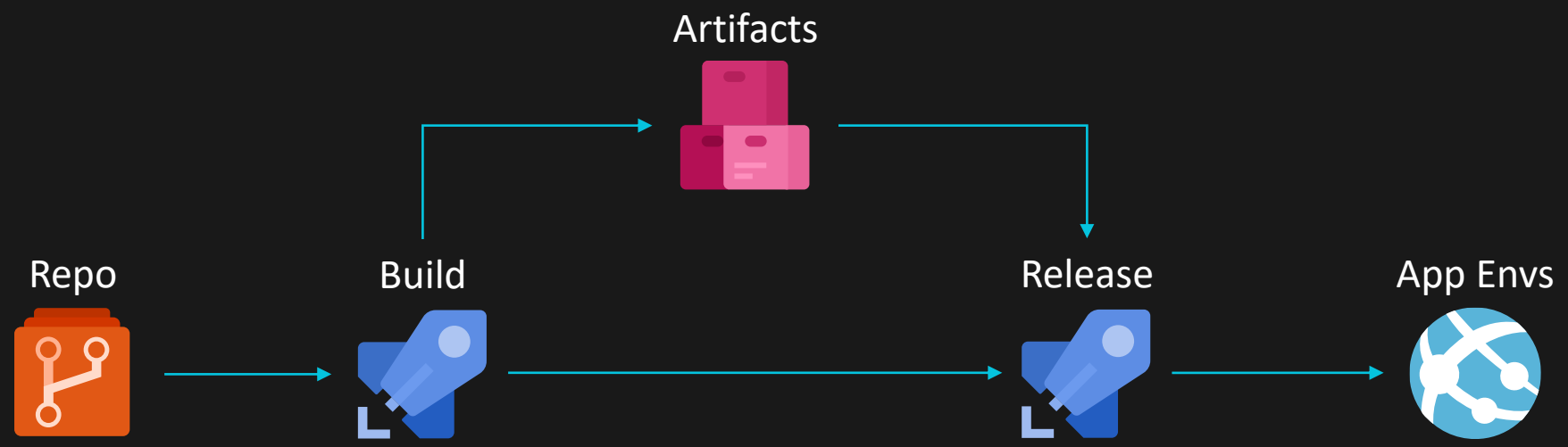
The test management and exploratory testing toolkit that lets you ship with confidence.

Azure Artifacts

Create, host, and share packages. Easily add artifacts to CI/CD pipelines.

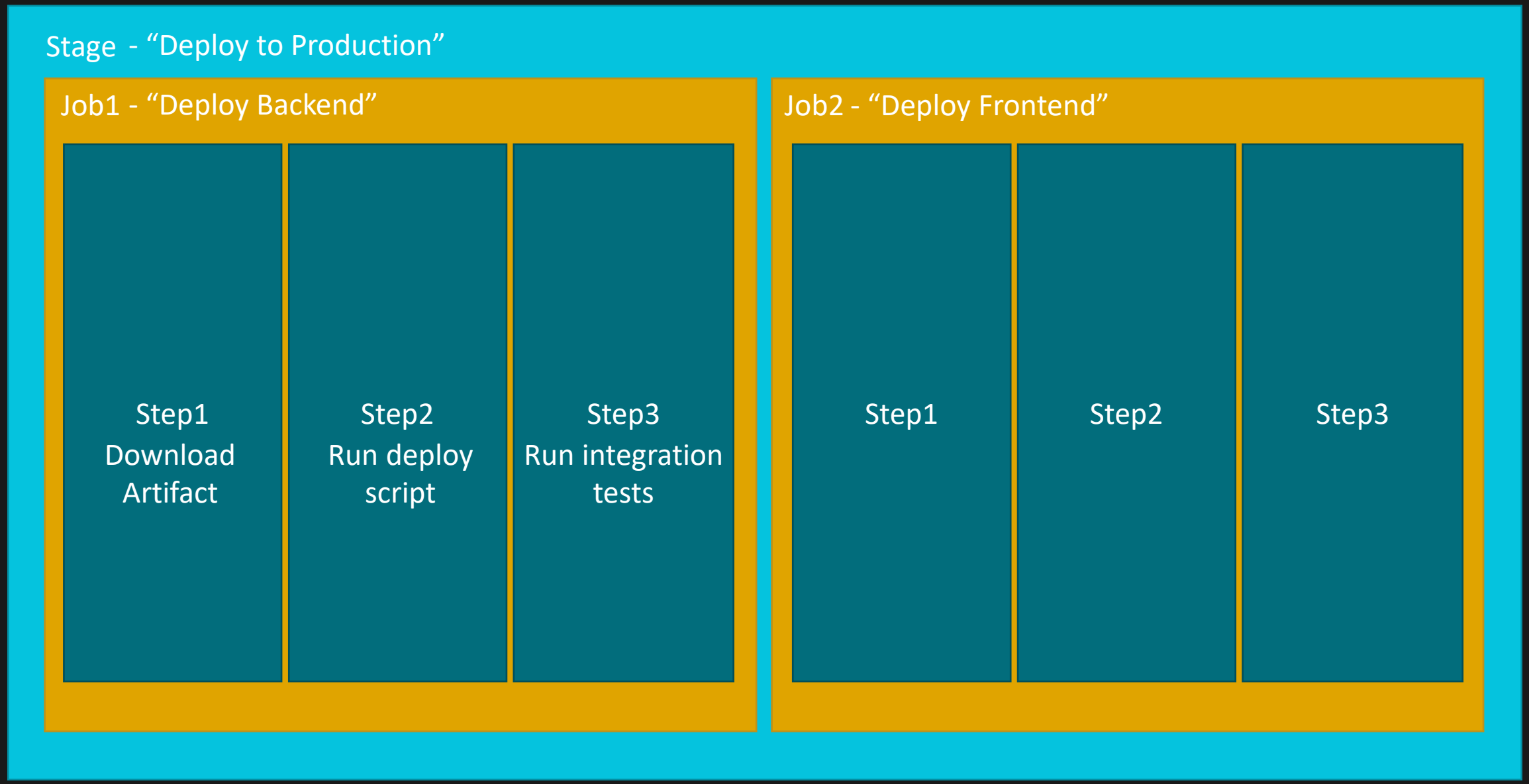
Azure Pipelines

Pipeline Concepts



- Get Source
- Install Tools
- Build Solution
- Run Tests
- Package, Version & Publish Artifacts

- Deploy to environments in stages
- Check Approvals & Gates in between
- Update Environment Status
- Update Work Item status


Stage - “Deploy to Production”
Job1 - “Deploy Backend”
Step1
Download
Artifact
Step2
Run deploy
script
Step3
Run integration
tests
Job2 - “Deploy Frontend”
Step1
Step2
Step3

Classic Pipeline Demo

Problems with Classic approach

- Repeated work
- Timing changes is hard (depending on your release strategy)
  - You can only make changes to production stages when the release branch has required binaries in it. For example a new function app.
- No ability to test something without breaking the pipeline for everyone else
  - Cloning is a possibility, but you still need to do changes twice if you do this.
- Parallelization requires compromises in other aspects
  - Either you run tons of stages, or spend more time running your deployments
  - Builds: All or Nothing in Parallel
- No process to approve changes before committing
  - If you have permission to edit, you can do it without anyone knowing.

YAML to the rescue

But what is that?

- Pipeline definition using YAML (Yet Another Markup Language)
- More importantly, it is Azure DevOps' implementation of **Pipelines as Code**
- Initially in public preview at the end of 2017
- Most of new features are coming to YAML first
- Some seem to not be implemented in the Classic pipelines at all.

basic.yaml

```yaml
trigger:
- master

stages:
- stage: stage1
  jobs:
  - job: job1
    pool:
      vmImage: 'windows-latest'
    steps:
    - task: NuGetToolInstaller@1
    - task: NuGetCommand@2
      inputs:
        restoreSolution: 'mysolution.sln'
    - script: echo Hello, world!
      displayName: 'Run a one-line script'

- stage: stage2
  jobs:
  - job: importantjob
    pool:
      vmImage: 'windows-latest'
    steps:
      - pwsh: 'write-output "I do nothing"'
```

Pipelines as Code?

"Infrastructure as code is the approach to defining computing and network infrastructure through source code that can then be treated just like any software system."

- Martin Fowler

”Pipelines as code is the approach to defining continuous integration and deployment through source code that can then be treated just like any software system.”

- Albert Einstein, probably

Benefits?

- Addition of version control
- Ability to run automated tests
- Less stress from rolling out changes
- Reduced recovery time
- Logic is often self documenting
- Code can be reused

Addition of version control

- Faster and more controlled development flow
- Code and pipeline logic stays in sync
- Portability of the whole product

Code can be reused

- Support of templates and parameters
- Standardized company pipelines through repositories as resouces
- Separation of concerns possible

Template stru

template-caller.yaml

Unsaved changes (cannot determine recent change or a

```yaml
trigger:
- master

stages:
- template: template-content.yaml # Lo
  parameters:
    project: 'UsefulApp'
    vmImage: 'windows-latest'
    buildConfiguration: 'Release'
```

template-content.yaml

```yaml
parameters:
  project: ''
  vmImage: 'windows-latest' # Default value if not given in
  buildConfiguration: ''

stages:
  - stage: 'dotnet_build'
    displayName: 'Build ${{ parameters.project }}'
    pool:
      name: 'Azure Pipelines'
      vmImage: ${{ parameters.vmImage }}

    jobs:
    - job: 'dotnet_build'
      steps:
        - task: DotNetCoreCLI@2
          displayName: 'dotnet restore'
          inputs:
            command: restore
            projects: |
              ${{ parameters.projectName }}.Api/${{ parameters.projectName }}.Api.csproj
              ${{ parameters.projectName }}.Tests/${{ parameters.projectName }}.Tests.csproj

        - task: DotNetCoreCLI@2
          displayName: 'dotnet build'
          inputs:
            projects: |
              ${{ parameters.projectName }}.Api/${{ parameters.projectName }}.Api.csproj
              ${{ parameters.projectName }}.Tests/${{ parameters.projectName }}.Tests.csproj
            arguments: '--no-restore -c $(buildConfiguration)'

#### Rest of the steps...
```

template-extend-skeleton.yaml

You, a few seconds ago | 1 author (You)

```yaml
parameters:
- name: buildSteps # the name of the parameter is buildSteps
  type: stepList # data type is StepList
  default: [] # default value of buildSteps

stages:
- stage: secure_buildstage
  pool:
    name: Azure Pipelines
    vmImage: 'ubuntu-latest'
  jobs:
  - job: secure_buildjob
    steps:

    - script: echo This happens before code
      displayName: 'Base: Pre-build'
    - script: echo Building
      displayName: 'Base: Build'

    - ${{ each step in parameters.buildSteps }}:
      - ${{ each pair in step }}:
          ${{ if ne(pair.key, 'script') }}:
            ${{ pair.key }}: ${{ pair.value }}
          ${{ if eq(pair.key, 'script') }}: # checks for buildStep with script
            'Rejecting Script: ${{ pair.value }}': error # rejects buildStep when script is found

    - script: echo This happens after code
      displayName: 'Base: Signing'
```

template-extend-caller.yaml

Yaml_Control.Deployment > Pipelines > template-extend-caller.yaml > [ ] trigger

```yaml
trigger:
- master

extends:
  template: template-extend-skeleton.yml
  parameters:
    buildSteps:
      - bash: echo Test #Passes
        displayName: Test - Will Pass
      - bash: echo "Test"
        displayName: Test 2 - Will Pass
      - script: echo "Script Test" # Comment out to successfully pass
        displayName: Test 3 - Will Fail
```

Variable templat

common-variables.yaml

Unsaved changes (cannot determine recent change or autho

```yaml
variables:
  # These are defaults for most runs
  - name: project
    value: UsefulApp
  - name: vmImage
    value: windows-latest
  - name: buildConfiguration
    value: Debug
  - name: NotifyStakeholders
    value: false

  - ${{ if startswith(variables['Build.SourceBranch'], 'refs/heads/release') }}:
    - name: buildConfiguration
      value: Release
    - name: NotifyStakeholders
      value: true
```

template-caller-vars.yaml

You, 3 minutes ago | 1 author (You)

```yaml
trigger:
- master
- release/*

variables:
- template: common-variables.yaml

stages:
- template: template-content.yaml
  parameters:
    project: ${{variables.project}}
    vmImage: $(vmImage)
    buildConfiguration: $(buildConfiguration)

- ${{ if eq(variables['NotifyStakeholders'], true) }}:
  - stage: handle_emails
```

Pipeline from another repository

```yaml
resources:
  repositories:
    - repository: templates # Identifier to refer to the repository in pipeline
      type: git # git (azure repos) / github / bitbucket
      name: Tooling/BuildTemplates # Projectname/Reponame, or just Reponame if in same project
      ref: refs/heads/master # specify branch / tag to use, defaults to master

jobs:
- template: build/common.yml@templates  # Template reference -> Path in repo@repoid
```

More things can run in parallel

- Classic mode only allows for Stage-level parallelisation. YAML allows for Jobs to run in parallel.
- YAML saves a lot of time in both builds and releases, but you still need multiple agents.

Remember our problems?

- Repeated work?
  - No more with template structure!
- Timing changes is hard?
  - Your pipelines move with the code, no longer an issue!
- No ability to test something without breaking the pipeline for everyone else
  - Git version control, Branching!
- Parallelization requires compromises in other aspects
  - Parallelize jobs to your hearts content, while keeping your status views clean!
- No process to approve changes before committing
  - Pull requests provide this functionality with a familiar process!

Working with YAML

Decorator

Variables | Save

master | Decorator / azure-pipelines.yml *

Show assistant

```yaml
trigger:
- master

pool:
  vmImage: 'windows-latest'

steps:
- script: npm i
- script: npm i -g tfx-cli
- script: tfx extension create --manifest-globs vss-extension.json --rev-version --output-path "$(Pipeline.Workspace)/Decorator"
```

```yaml
# ASP.NET Core (.NET Framework)
# Build and test ASP.NET Core projects targeting the full .NET Framework.
# Add steps that publish symbols, save build artifacts, and more:
# https://docs.microsoft.com/azure/devops/pipelines/languages/dotnet-core

trigger:
- master

name: $(Date:yyyyMMdd)$(Rev:.r)-$(SourceBranchName)

pool:
  name: Azure Pipelines
  vmImage: vs2017-win2016
  demands:
  - msbuild
  - visualstudio

variables:
  buildPlatform: 'Any CPU'
  buildConfiguration: 'Release'

steps:
- task: NuGetToolInstaller@1

- task: DotNetCoreCLI@2
  displayName: 'dotnet restore'
  inputs:
    command: restore
    projects: |
      RandomCustomer.Simulation.API/RandomCustomer.Simulation.API.csproj
      RandomCustomer.Simulation.Functions.BatchPoller/RandomCustomer.Simulation.Functions.BatchPoller.csproj

- task: DotNetCoreCLI@2
  displayName: 'dotnet build'
  inputs:
    projects: |
      RandomCustomer.Simulation.API/RandomCustomer.Simulation.API.csproj
```

Decorator-Test

Tasks | Variables | Triggers | Options | Retention | History | Save & queue | Discard | Summary | Queue | ...

Pipeline
Build pipeline

Get sources
Decorator | master

Agent job 1
Run on agent

Hello World 1
PowerShell

Publish Pipeline Artifact
Publish Pipeline Artifacts

PowerShell

Link settings | View YAML | Remove

Task version 2.*

Display name *
Hello World 1

Type

File Path | Inline

Script *

```
Write-Host "Hello World"
```

ErrorActionPreference

Stop

Advanced

How to get started?

- Convert old pipelines to YAML
- Play around in the portal
- Get VS code extension
- Read the docs, search github etc. for examples
- Create snippets or just utilize templates

Advanced uses

Demos

- Yaml Pipeline Demo
  - Template Structure
  - Variable Groups
  - Pipeline resources
  - Deployment Jobs - Specialized Job type to handle the lifecycle of a deployment
- Pipeline Decorator Demo
  - Add custom logic to every build pipeline in the organization

YAML Pipeline Demo

Pipeline Decorators Demo

Cons?

- Tooling still needs some work (No local testing!)
- Not everything is yet supported out of the box (Deployment Gates etc.) -> Build it yourself?
- Something you take for granted in Classic might be tough to implement with YAML (Specific artifact selection during runtime etc.)
- Documentation is not yet all there
- There is a learning curve, but it gets easier after the first hump

Takeaways & best practices

- YAML is the way forward in Azure DevOps, Use it!
- Use templating when possible, evaluate the need for centralized repo
- Parallelize everything you can to speed up runs (still need agents!)
- Utilize Deployment Jobs, keep an eye out for further development
- Try out the VS code extension (Azure Pipelines)

Resources & Links

- Official Documentation
- Azure DevOps Blog
- dotnet/arcade
- My blog, of course! huuhka.net

Questions?

Thank you!

ZURE