Azure DevOps YAML CI/CD pipelines – real experiences

Jan Vilímek
Backend Architecture Lead @ Oriflame
jan@vilimek.cz
@vilimekJan

Motivation

- In Oriflame, we have been using Microsoft Azure DevOps for CI/CD (continuous integration/deployment) for several years for projects where about 200 engineers cooperate.
- We would like to share our experiences working with YAML pipelines that cover most of our needs.
- Why you should love them?
- What to rather avoid?

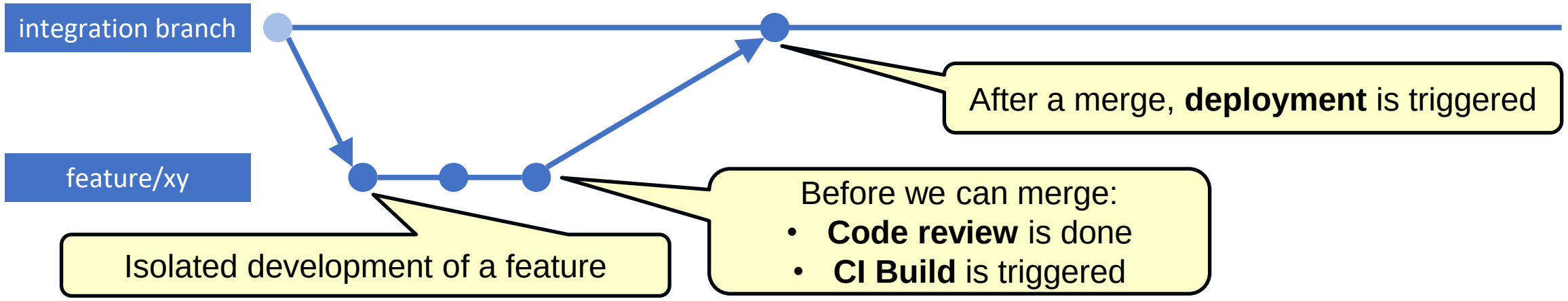
Agenda

1. Brief introduction to CI/CD pipelines
2. Current situation in Oriflame
3. Let’s talk about YAML pipelines
4. What worked well for us

Brief introduction to CI/CD pipelines

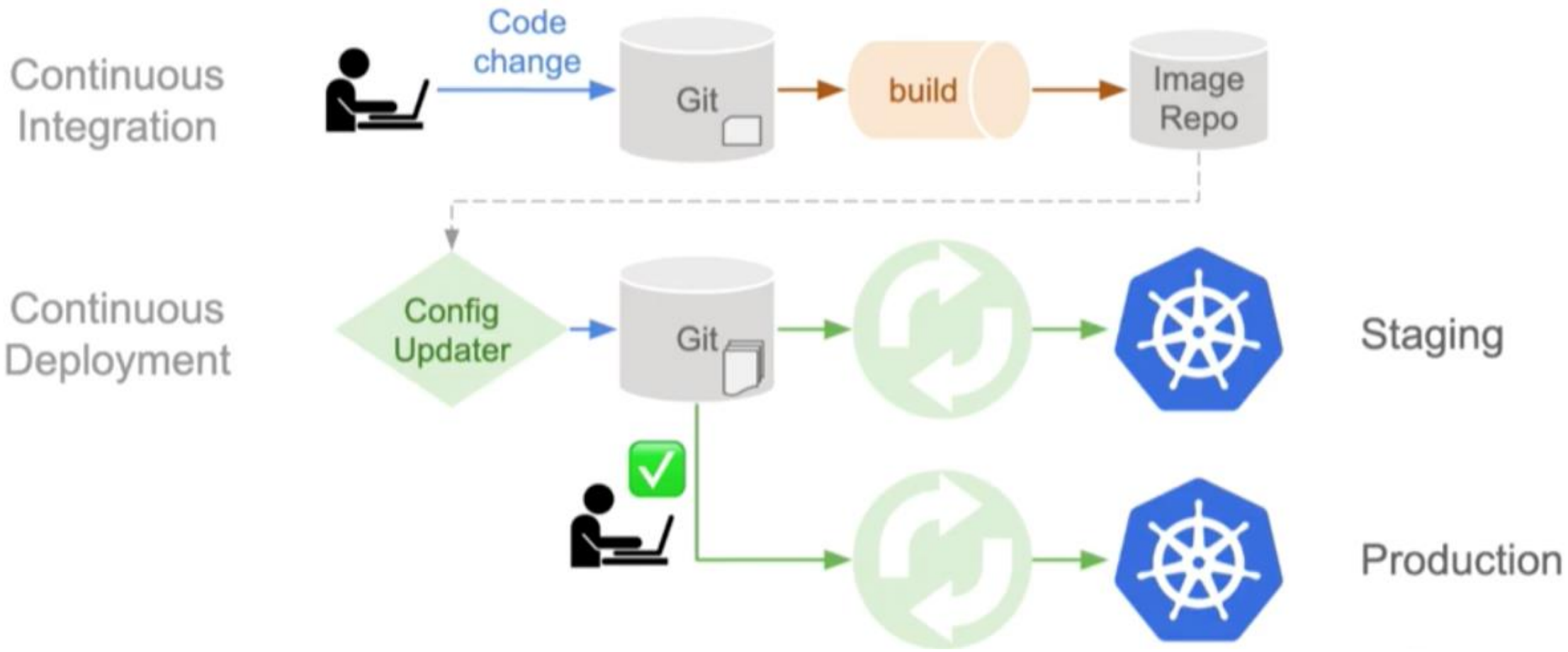
About CI/CD pipelines

- CI/CD: continuous **integration** and **deployment**



- Needs to be fully automated
  - Triggered via pull request, push
- CI is about build, unit tests
- CD is about deployment (pushing/deploying package)

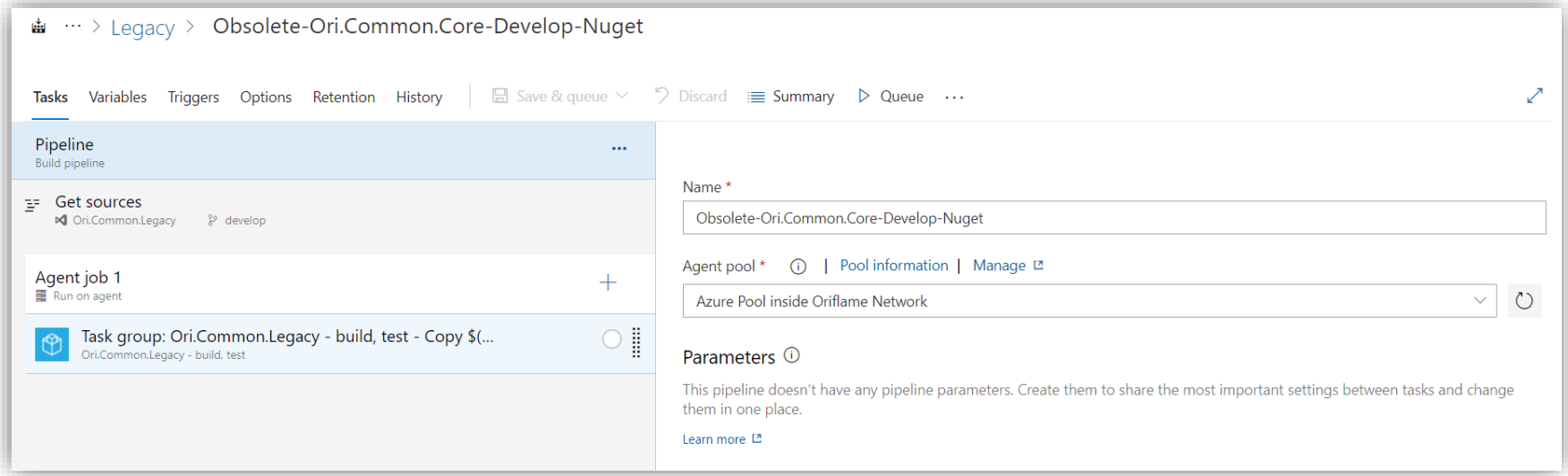
Example of CI/CD flow



Current situation in Oriflame

In Oriflame, we are using YAML pipelines

- We **were** using „classic“ Azure DevOps pipelines
  - Nice presentation by Tomáš Herceg (Azure DevOps Automatické Buildy a releasy, 2020/02/11 WUG)
- These were **not good enough** for us
- Why? Let me explain on the following slides



Current projects

- eCommerce monolith (.NET + SQL Azure backend + Storage +…)
  - 12 teams
  - DEV, UAT, STG, PAT, LIVE environments
  - 8 regions x 4 roles (IaaS) ~ 50 servers per environment
- ~40 microservices
  - .NET core (most of it)
  - mix of Azure Kubernetes (**AKS**) and App Services
  - SQL Azure, Cosmos DB, Azure Storage, REDIS,…
- ~50 single page application & components
  - Typescript; mix of tools including webpack, parcel, babel, …
- ~100 npm & NuGet packages

Current tools

- Azure DevOps
  - Project per domain / service (~90)
  - Repository per service / package (at least 1 pipeline)
  - In total **~200 pipelines**
- 99% repositories in Git
- Package published to Azure Artifacts + Container Registry
- Pipeline Agents
  - Both hosted + self-hosted

Requirements for CI/CD pipelines

- How to share it?
- How to test it?
- How to review changes?
- How to track changes / history?
- How to implement it?
- How to use it?

Let’s talk about YAML pipelines

YAML pipelines important features

- Next generation of “classic” pipelines since 2017, “release” 2020/04
- Pipeline part of the repository
- Versioning/history/branching/code review
- Sharing between projects (templates)
- Testable
- Combination of build & release (stages, jobs, steps)
- Easy to use samples (copy & paste)
- Parameters & Variables

Next generation of “classic” pipelines

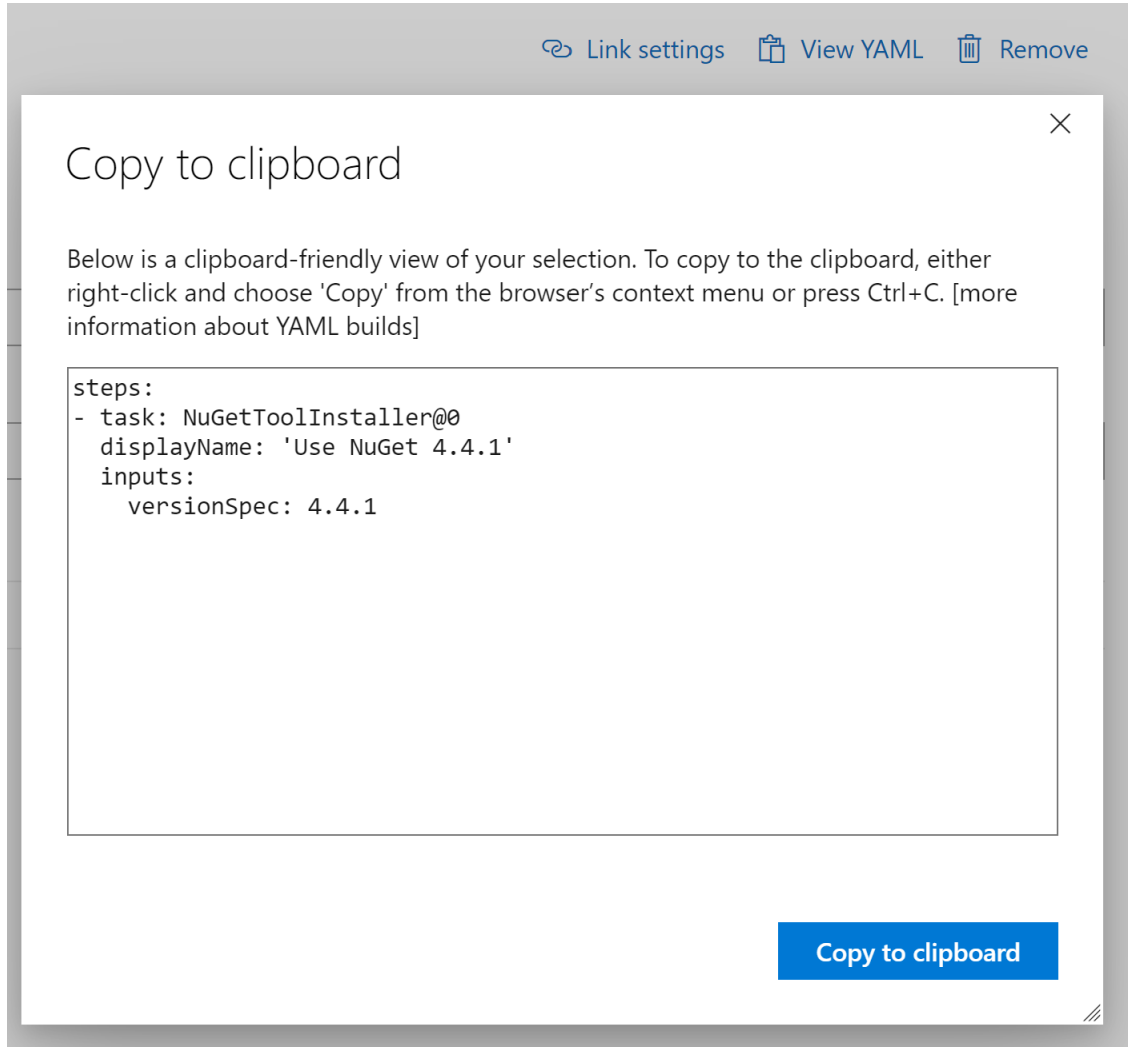
- YAML pipelines share basic logic
  - Evolving over time
- Can use the same steps
- Can not run on TFVC
- Similar, but not quite the same as GitHub Actions

Azure Pipelines

```yaml
jobs:
  - job: scripts
    pool:
      vmImage: 'windows-latest'
    steps:
      - script: echo "This step runs in the default shell"
      - bash: echo "This step runs in bash"
      - pwsh: Write-Host "This step runs in PowerShell Core"
      - task: PowerShell@2
        inputs:
          script: Write-Host "This step runs in PowerShell"
```

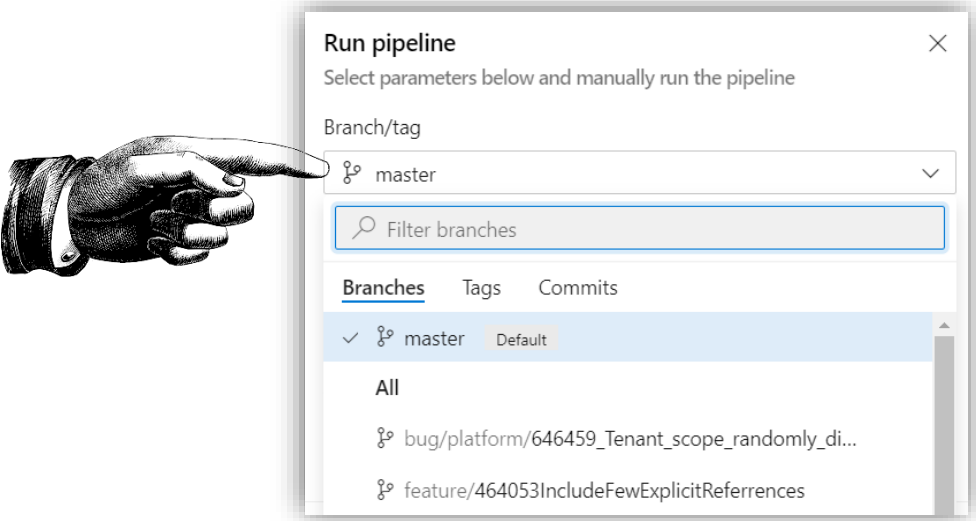
GitHub Actions

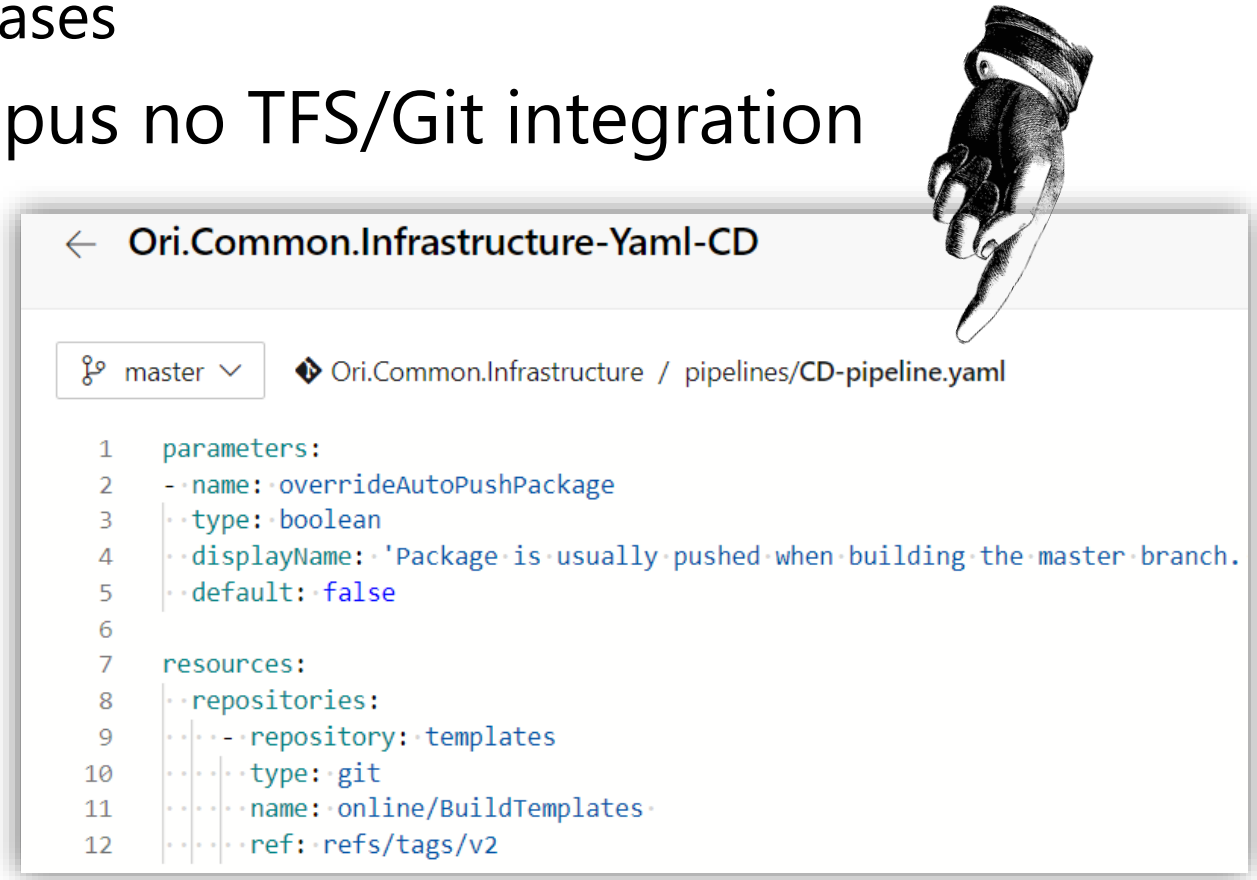
```yaml
jobs:
  scripts:
    runs-on: windows-latest
    steps:
      - run: echo "This step runs in the default shell
      - run: echo "This step runs in bash"
        shell: bash
      - run: Write-Host "This step runs in PowerShell
        shell: pwsh
      - run: Write-Host "This step runs in PowerShell"
        shell: powershell
```



Pipeline part of the repository

- A little history:
  - TFS, VSO/VSTS -> XAML, a single build pipeline
  - VSTS/DevOps: new pipelines & releases
- 3rd party orchestrators, e.g. Octopus no TFS/Git integration
- Now it is part of source repo





Versioning/history/branching/code review

- Since it is part of source code…
- …you see versions of the pipeline across history
- …you can isolate your work in branches (PR build validation)
- …you can do code review during pull request
- This will protect us from breaking branches not with sync with pipeline changes, e.g. release build

Sharing between projects (1/2)

- Before YAML, sharing was done via “task groups”
  - Not between projects (only export/import possible)
- Now you can use templates from other repositories (Git/GitHub)
- You can even choose specific version (e.g. branch, tag…)
- Examples:

```yaml
resources:
  repositories:
  - repository: repository_identifier
    type: git
    name: project/repository_name #if in the same repo
    ref: refs/heads/main

jobs:
- template: relative-repo-path/pipeline-definition.yml@repository_identifier
```

```yaml
resources:
  repositories:
    - repository: TemplatesRepo
      type: git
      name: Pipelines/OriCommon
      ref: refs/tags/v2

extends:
  template: templates/stages/buildPackage.yml@TemplatesRepo
```

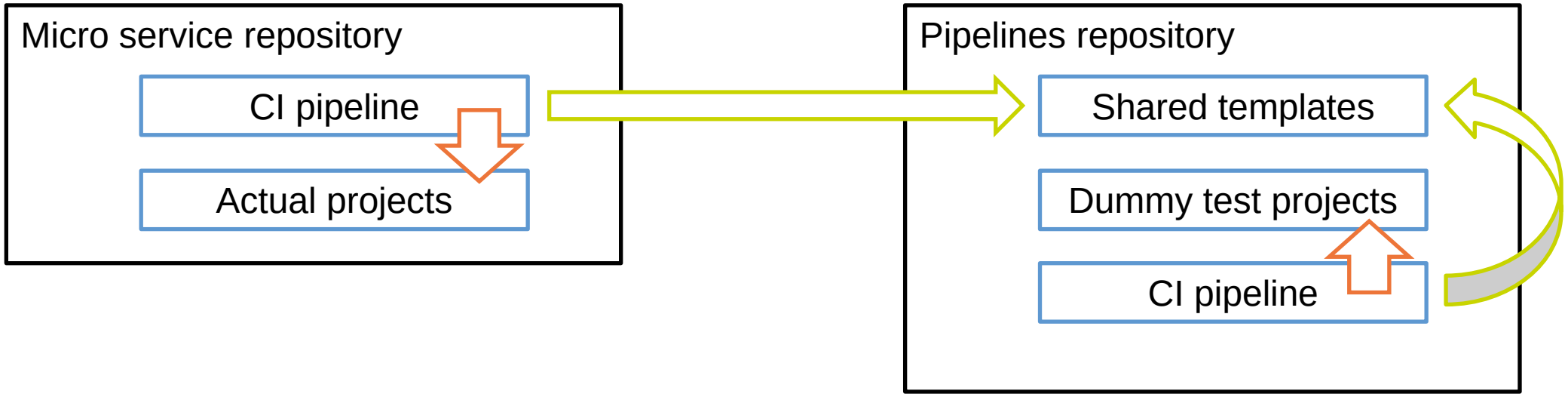
Sharing between projects (2/2)

- Another approach is to do a **multi-checkout**
- Useful when you have also resources (e.g. scripts) in shared repo
- Example (More info: multi-repo-checkout):

```yaml
resources:
  repositories:
  - repository: repository_identifier
    type: git
    name: project/repository_name #if in the same repo can be just name of the repository
    ref: refs/heads/main

jobs:
- job: job_name
  pool:
    vmImage: ubuntu-latest
  steps:
    - checkout: self  # will checkout current repository
      path: this-repo # relative path where to check out source code, will be '$(Agent.BuildDirectory)/this-repo'

    - checkout: repository_identifier
      path: other-repo

    - task: PowerShell@2
      inputs:
        filePath: '$(Agent.BuildDirectory)/other-repo/relative-path-in-repo/script.ps1'
        failOnStderr: true
        showWarnings: true
        pwsh: true
        workingDirectory: '$(Agent.BuildDirectory)/backstage-catalog-components/pipelines'
```

Testable

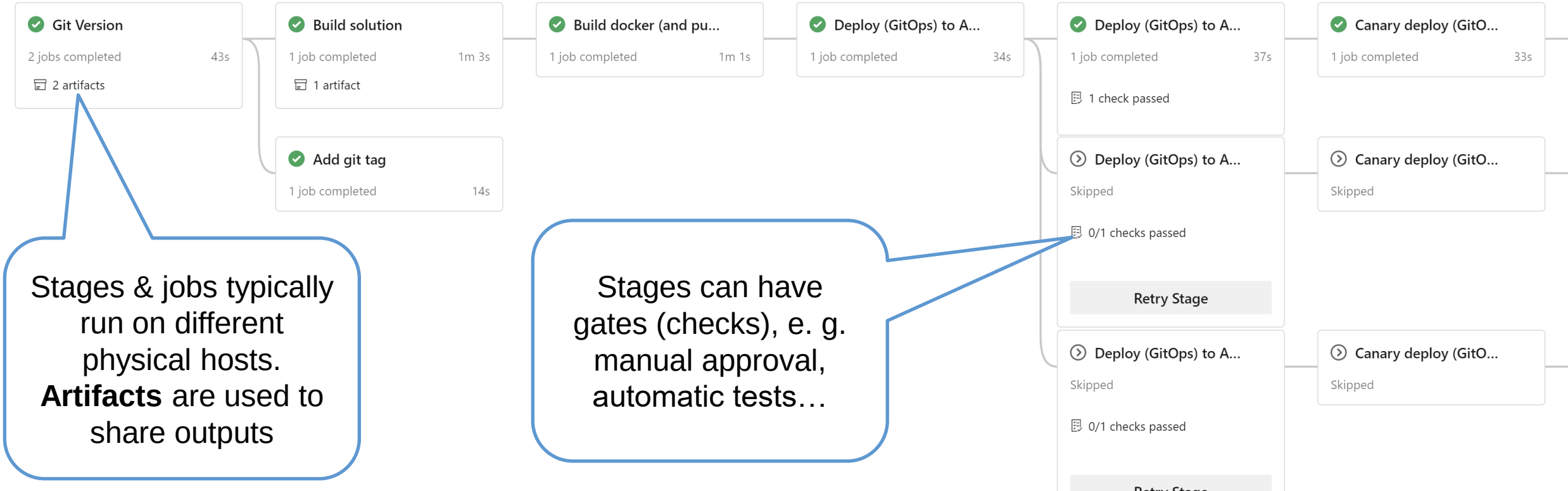
- You know your pipeline works ok during pull request
- How can you tell the same about **shared templates**?
- Answer is simple: CI builds on shared pipelines repository



Sample CI build for templates: https://github.com/Oriflame/devops/blob/develop/ci-build.yaml

Combination of build & release (stages, jobs, steps)

- There were pipelines & releases separated
- Now there is only YAML ☺
- Pipeline anatomy: Stages -> Jobs -> Tasks



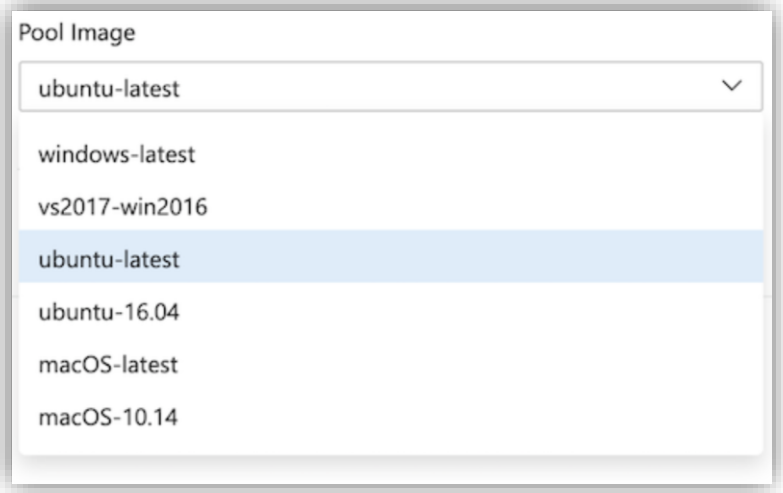
Stages & jobs typically run on different physical hosts. **Artifacts** are used to share outputs

Stages can have gates (checks), e. g. manual approval, automatic tests…

Easy to use samples

- Microsoft samples
- Simple example by Jordan Lee
- In Oriflame, we use **boilerplates**:
  - It's kind of a "sample" repository
  - All you need: simple project, build&deployment pipelines, documentation, …
  - Maturity check (both boilerplate and your service)
  - Guidance & scripts how to setup branches (e.g. policies)
- A team can easily start new service/SPA/NuGet/…
- And they still have the possibility to change anything

Parameters & Variables



- Parameters:
  - Provided at runtime (start of pipeline)
  - Can be type-safe (types, ranges, defaults…)
  - Evaluated at start -> “dynamic” pipeline (should I **include** step A?)
- Variables:
  - Provided by pipeline definition or agent (e.g. System.AccessToken)
  - Can be modified (or added) via steps
  - Can be used in conditions (should I **execute** step A?)
- Documentation: variables, parameters

Have we now answers for CI/CD pipelines?

- How to share it?
- How to test it?
- How to review changes?
- How to track changes / history?
- How to implement it?
- How to use it?

What worked well for us

Shared YAML templates

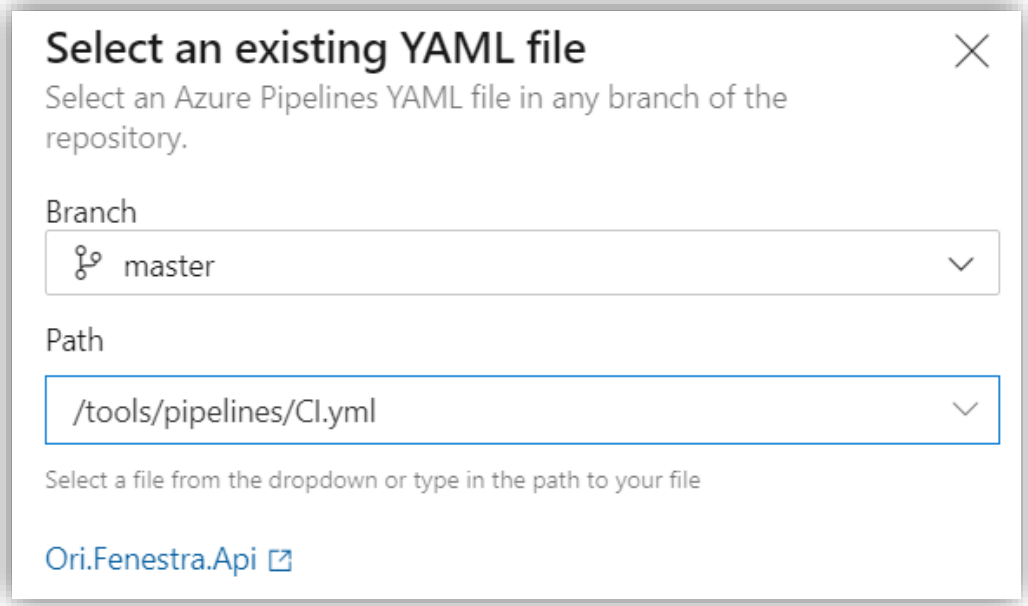
- Templates in “Pipelines” project
- Supported tags
  - latest: always the latest build & deployment
  - v1, v2, …: latest stable major version
  - 1.0.0: exact version
- Multistage/jobs: by default, the whole pipeline referenced
  - But you can choose what to run, override stages
- Linux and .NET core builds preferred (twice as faster)
- Using e.g. Gitversion task with the ability to override configuration
- Don’t overcomplicate parameters (instead use e.g. stages, more pipelines)

Integration to services

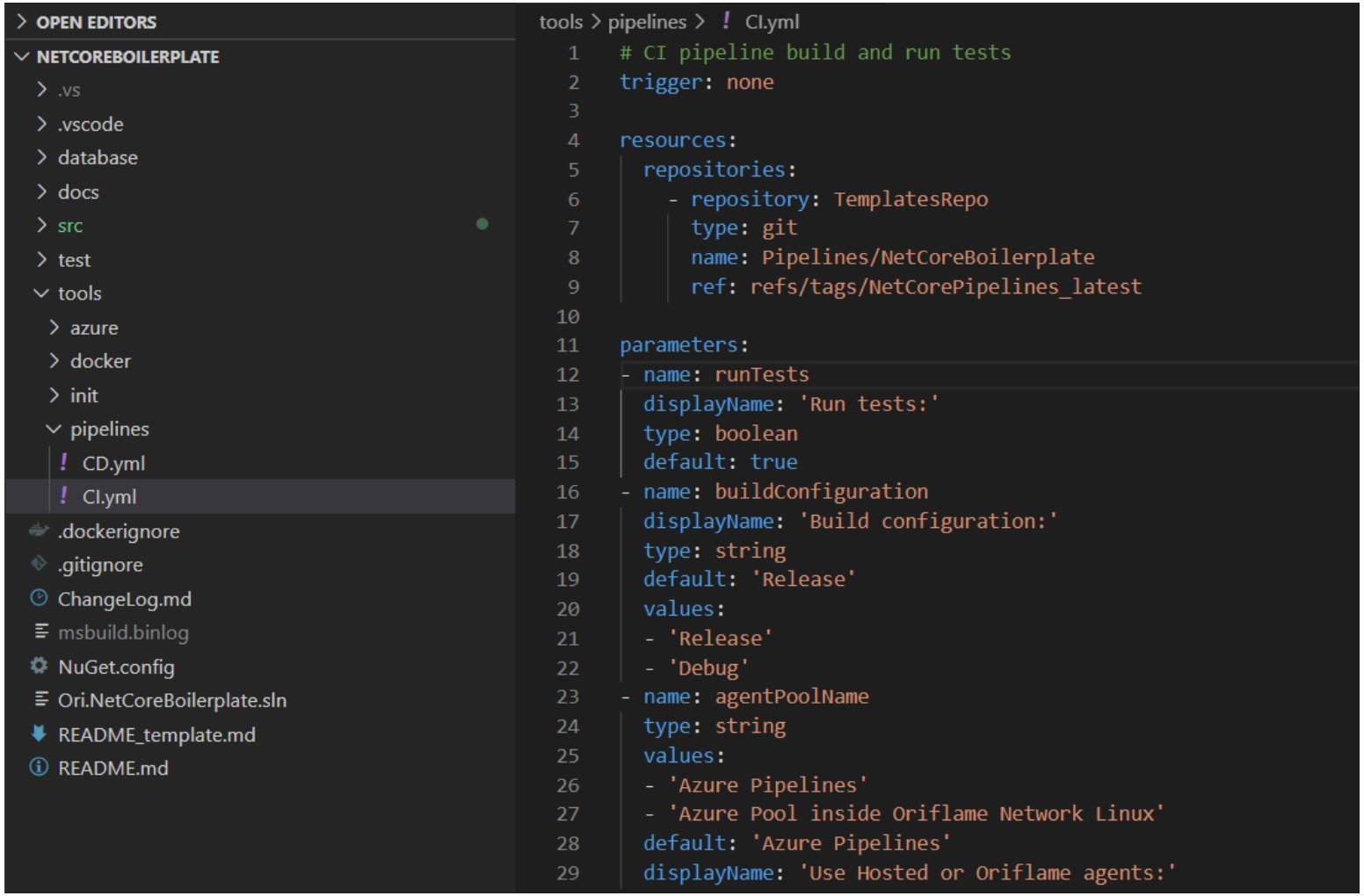
- Needs to be simple as this:

```yaml
trigger: none
resources:
  repositories:
    - repository: pipelines
      type: git
      name: Pipelines/NetCoreBoilerplate
      ref: refs/tags/v2
stages:
- template: templates/stages/buildStages.yml@pipelines
```

- And one manual action: select path to it while adding new pipeline from Azure DevOps portal



Of course the reality is a bit more complicated



DEMO

Setup repository policies

- Integration/main branch protection via PR
- Code review of pipeline
- CI builds for shared pipelines
- Split pipelines (stages) based on concerns
  - CI build
  - CD for NuGet package
  - CD for Docker image
  - CD for AKS (GitOps) release
  - Security checks
  - Integrations

Caveats

- YAML…SPACES & TABS…oh my!
  - Setup the IDE correctly or you will cry…like a lot!
- Anyone can modify the pipeline
  - Can be “fixed” via a PR policy
- Easily can become complex
  - Combination of template vs. runtime parameters & variables is hell
  - E.g. Variable-init issue

That’s all folks, thank you!

Jan Vilímek
Backend Architecture Lead @ Oriflame
jan@vilimek.cz
@vilimekJan