Making the most of your DevOps Artifacts

Matthew Sheehan

Planet DDS

https://github.com/MattSheehanDev/codemash2022-artifacts

What is an artifact?

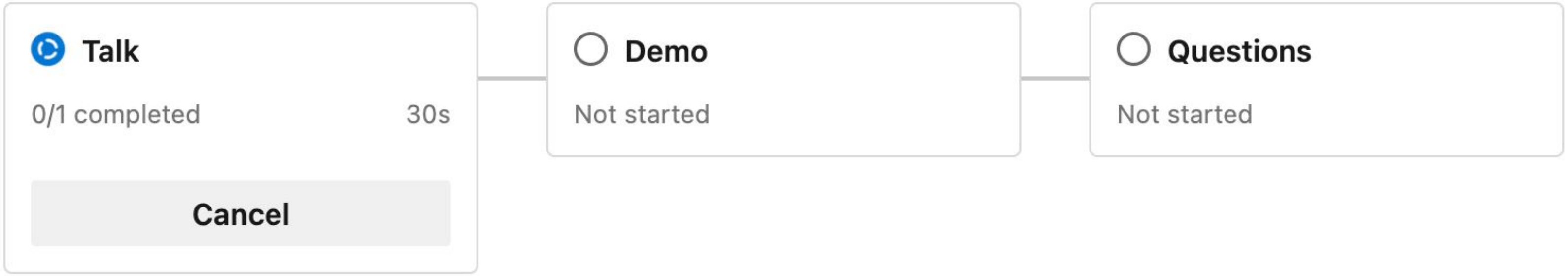
- ~~An artifact is really just the build output of some sort.~~
- The by-product of some process applied to a source code repository.
  - Build process - .exe, .dll, .lib
  - Packaging process - Nuget, NPM
  - Automated testing process
  - Code analysis process - Dependency Check
- Can be recreated given the same environment, process, and inputs.
- Usually stored separate from source code and have separate lifetimes.

Artifacts in DevOps

- Artifacts are produced as part of the CI/CD tooling.
  - Azure Pipelines
  - Github Actions
  - Gitlab CI/CD
  - Jenkins
- Different types of artifacts:
  - Deployment artifacts
  - Library artifacts
  - Testing artifacts
  - Pipeline / intermediate artifacts

Artifacts in DevOps

- Artifacts are produced as part of the CI/CD tooling.
  - **Azure Pipelines**
  - Github Actions
  - Gitlab CI/CD
  - Jenkins
- Different types of artifacts:
  - Deployment artifacts
  - Library artifacts
  - Testing artifacts
  - Pipeline / intermediate artifacts


Talk
0/1 completed
30s
Cancel
Demo
Not started
Questions
Not started

Azure Pipeline Agent Environments

- Microsoft-hosted agents
  - Run in VM or container.
  - Windows Server, Linux (Ubuntu), macOS.
  - 2 CPU Cores, 7 GB RAM, 14 GB SSD with ~10GB allocated to pipeline builds.
    - Standard_DS2_v2 general purpose virtual machines.
  - Located in the same region as your DevOps organization.
- Self-hosted agents
  - Deploy as a VM or a Docker container.
  - Can drop artifacts onto a file share.
  - Can pre-deploy with any additional software needed.

* MS Hosted Agent environment: https://github.com/actions/virtual-environments

Microsoft-hosted Agent VMs

```yaml
jobs:
- job: Linux
  pool:
    vmImage: 'ubuntu-latest'
  steps:
  - script: echo hello from Linux
- job: macOS
  pool:
    vmImage: 'macOS-latest'
  steps:
  - script: echo hello from macOS
- job: Windows
  pool:
    vmImage: 'windows-latest'
  steps:
  - script: echo hello from Windows
```

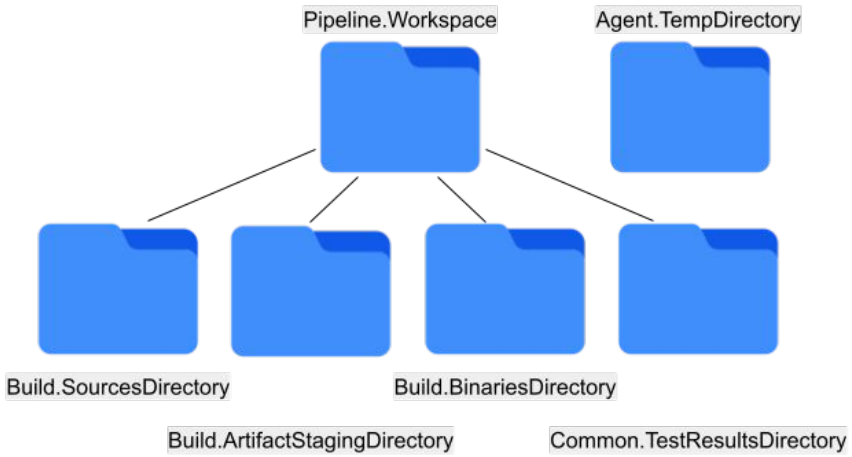
```
ubuntu-latest / ubuntu-20.04 (default)
ubuntu-18.04
```

```
macOS-latest / macOS-11
macOS-10.15
```

```
windows-2022
windows-latest / windows-2019
```

Pipeline Workspace Folders

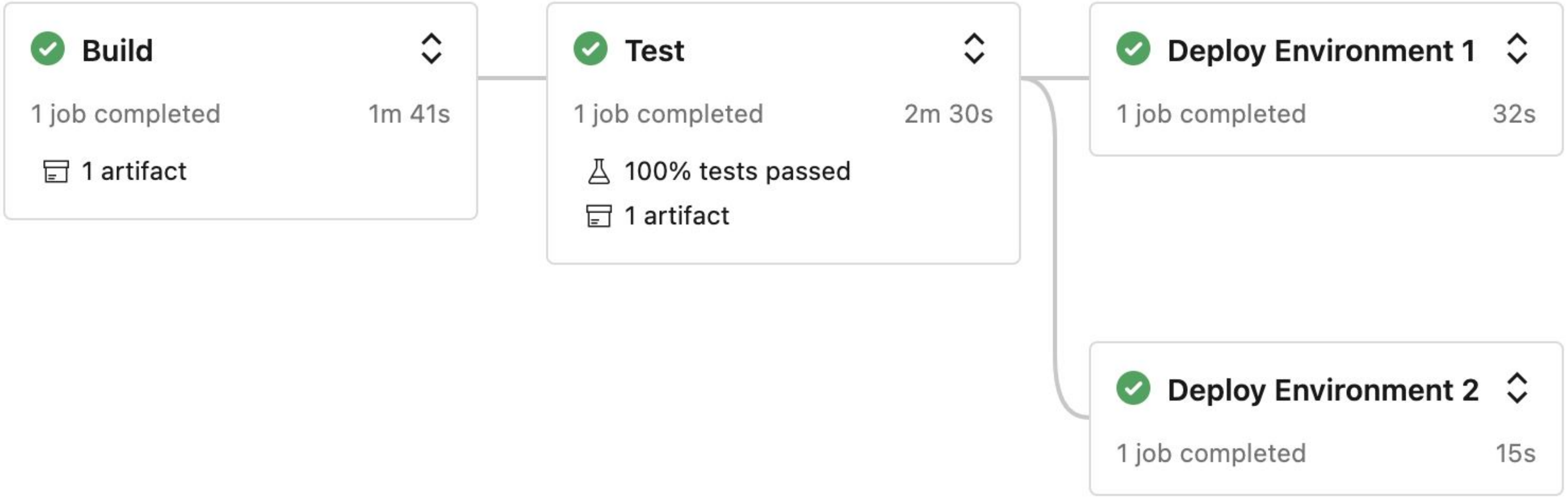
- Workspace Directory - Ex. D:/agent/1
  - `$(Pipeline.Workspace)`
  - `$(Agent.BuildDirectory)`
- Sources Directory - Ex. $(Pipeline.Workspace)/s
  - `$(Build.SourcesDirectory)`
  - `$(System.DefaultWorkingDirectory)`
- Artifact Directory - Ex. $(Pipeline.Workspace)/a
  - `$(Build.ArtifactStagingDirectory)`
  - `$(Build.StagingDirectory)`
- Binaries Directory - Ex. $(Pipeline.Workspace)/b
  - `$(Build.BinariesDirectory)`
- Test Results Directory - Ex. $(Pipeline.Workspace)/TestResults
  - `$(Common.TestResultsDirectory)`



Pipeline Workspace Folder Cleanup

- For Microsoft-hosted agents
  - Each job gets either a clean VM or container during every pipeline run.
- For self-hosted agents
  - Only $(Build.ArtifactStagingDirectory) and $(Common.TestResultsDirectory) are cleaned every run.
  - Enables incremental builds and better caching.
  - Persistent partial builds are not always desired…

```yaml
- job: Job1
  workspace:
    clean: all  # outputs | resources | all
  steps:
    ...
```


Build
1 job completed 1m 41s
1 artifact

Test
1 job completed 2m 30s
100% tests passed
1 artifact

Deploy Environment 1
1 job completed 32s

Deploy Environment 2
1 job completed 15s

Publish Pipeline Artifacts

```
- task: PublishPipelineArtifact@1
  inputs:
    targetPath: $(Build.ArtifactStagingDirectory)
    artifactName: artifactName
```

```
# shortcut for publish task
- publish: $(Build.ArtifactStagingDirectory)
  artifact: artifactName
```

- The publish task uploads a folder or file as a pipeline artifact.
- Artifacts will be stored with their pipeline run.
- Specifying an artifact name is technically optional but should be considered a best practice.
  - Cannot be a reserved ASP.NET folder name (Bin, App_Data, etc).
- The publish keyword is a shortcut for the publish task.

Publish Pipeline Artifacts

```yaml
- task: DotNetCoreCLI@2
  displayName: Build
  inputs:
    command: 'build'
    projects: 'project.csproj'

- task: CopyFiles@2
  displayName: 'Copy To: $(Build.ArtifactStagingDirectory)'
  inputs:
    SourceFolder: '$(Build.SourcesDirectory)'
    TargetFolder: '$(Build.ArtifactStagingDirectory)'
    Contents: |
      **/bin/**/*.exe
      **/bin/**/*.dll

- publish: '$(Build.ArtifactStagingDirectory)'
  artifact: api
```

- What if you only want to publish some files?
  - The publish tasks only accepts a fully-qualified path without wildcard support.
- `CopyFiles@2` is commonly used to copy files from a source to target location.

.artifactignore

- Same syntax as .gitignore and can be checked into version control.
- The .git folder is ignored by default.
- Can be used to avoid copying files to a staging directory to reduce execution time.
- Must be in the same directory from which you upload your artifacts.

```
# ignore all files
**/*
# except those in the release dir
!src/app/bin/Release/**.*
```

Publish Pipeline Artifacts

- When are pipeline artifacts deleted?
  - Artifacts are deleted when the pipeline run is deleted.
  - Pipeline runs can be manually deleted or deleted according to the project retention policy (default 30 days).
  - All pipeline and build artifacts, logs, test results, symbols, binaries, run metadata are deleted.
- Can you modify published pipeline artifacts?
  - Pipeline Artifacts are immutable. Once an artifact is published, that version is preserved for its lifetime.
  - Running a job again from the same pipeline run will result in the job failing if the same artifact has already been published.

Publish Pipeline Artifacts

- `PublishBuildArtifacts@1` task is deprecated in favor of `PublishPipelineArtifact@1`.
  - `PublishPipelineArtifact@1` provides significantly faster performance when uploading artifacts.
- Pro Tip: Setting the pipeline variable `System.Debug` to `true` will enable detailed logs for your pipeline runs.

Download Pipeline Artifacts

```yaml
- task: DownloadPipelineArtifact@2
  inputs:
    source: current
    artifact: api
    # optional
    path: $(Pipeline.Workspace)
    patterns: **
    project:
    pipeline:
    runVersion: 'latest'
    runBranch: 'main'
    runId:

# shortcut for download task
- download: current
  artifact: api
```

- The download task consumes artifacts from a previous job.
- Artifacts can be downloaded from the current pipeline run or from a specified pipeline.
- `DownloadBuildArtifacts@0` deprecated in favor of `DownloadPipelineArtifact@2`.
- The download keyword is a shortcut for the download task.

Download Pipeline Artifacts

- Can multiple artifacts be downloaded?
  - Excluding the artifact name from the task will download all artifacts that exist from the current (or specified) pipeline run.
- Where are artifacts downloaded?
  - When a single artifact is downloaded, the default location is `$(Pipeline.Workspace)`.
  - If multiple artifacts are downloaded, the default location is `$(Pipeline.Workspace)/<artifact name>`.
- Can artifacts be downloaded from other pipelines?
  - By setting `source: specific` and specifying the pipeline name and optionally the pipeline run.

Pipeline Resources

- Pipelines can be added as an additional resource.
- Pipelines that publish artifacts, can be consumed in other pipelines when set as a pipeline resource.

pipelines
builds
repositories
containers
packages
webhooks

```yaml
resources:
  pipelines:
    # symbolic name, can be anything
    - pipeline: pipelineSymName
      # name of the pipeline as it appears in DevOps
      source: Azure Pipeline Name
      version: # specific pipeline run
      branch: # branch to get the artifact from
      tags:   # fetch artifacts from pipeline with tags.
      # pipeline triggers can be set
      trigger:
        branches:
          include:
            - master

stages:
...
```

Pipeline Resources

- How does pipeline triggering work with pipelines in separate repos?
  - Pipelines in the same repository will trigger on the same branch and commit.
  - A pipeline resource in a separate repository will trigger the current pipeline on the default repo branch (usually main/master).
- What happens when the 2nd pipeline is triggered manually?
  - If a specific pipeline version is specified, artifacts from that pipeline run will be selected.
  - If a branch is specified, get the latest artifacts from that pipeline on that branch.
  - If pipeline tags are specified, get the latest artifacts from that pipeline with those tags.
  - Otherwise, get the latest artifacts from the most recent pipeline run.

Pipeline Resources vs. Download Tasks

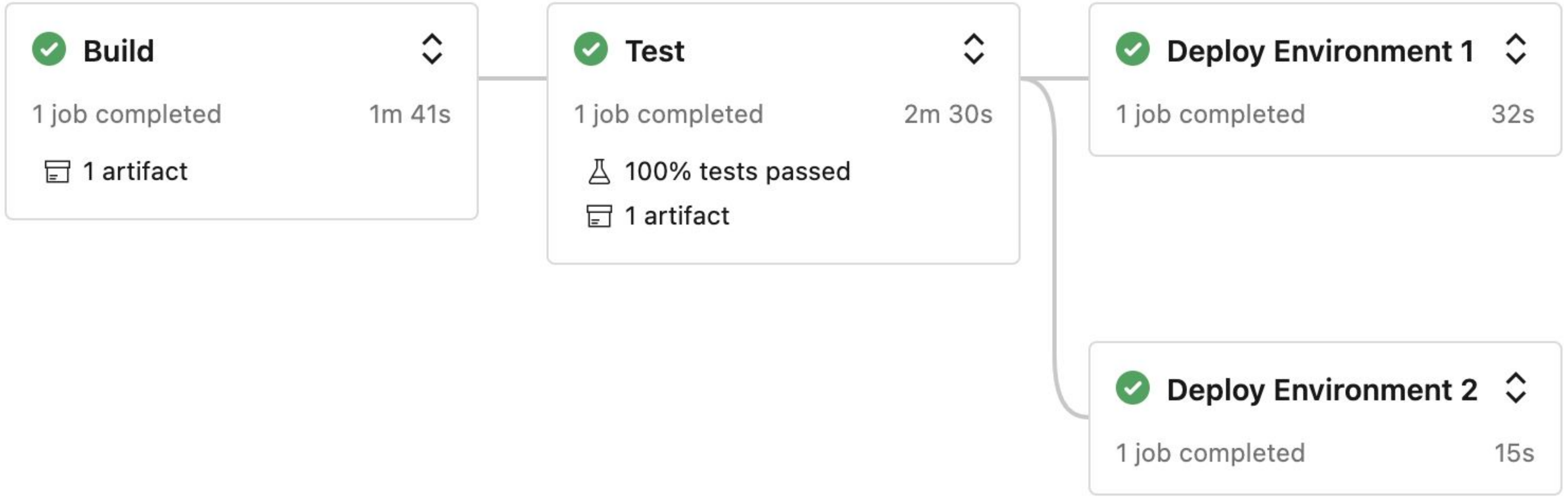
- Pipeline resources are the preferred way to consume artifacts across pipelines.
- Resources linked to a pipeline have increased traceability.
  - What resource/pipeline was responsible for the trigger.
  - What artifact version was consumed.
- Pipeline resources allow you to consume an artifact and also configure triggers.
- Download tasks can be used to avoid a direct dependency.

Pipeline Caching

- Can be used to reduce build times by reusing files in later runs.
- Stored in Azure blob storage.
- No enforced cache limit size or number of caches.
- Caches expire after 7 days of inactivity.
- Use pipeline caching only when the files not existing in cache will not affect the jobs ability to run.

```yaml
- task: Cache@2
  displayName: Cache Yarn packages
  inputs:
    path: $(Pipeline.Workspace)/.yarn
    key: '"yarn" | "$(Agent.OS)" | yarn.lock'
    restoreKeys: |
      yarn | "$(Agent.OS)"
      yarn
```

Deploy Pipeline



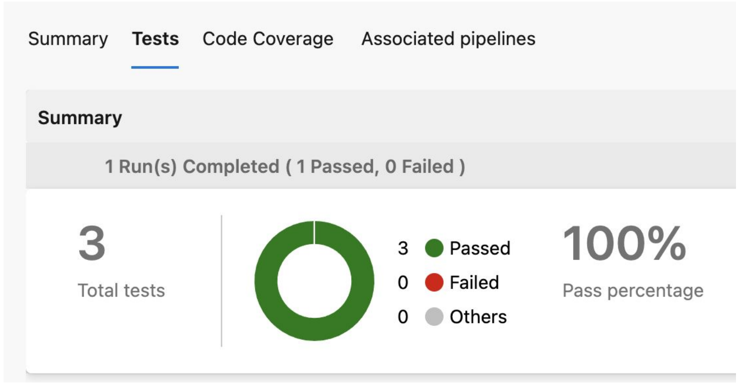
Deployment jobs

- Runs against a set environment that track the deployment history.
- Can define a deployment strategy.
- Does not automatically clone the source repo.
  - `- checkout: self`
- **The deploy hook automatically downloads artifacts from the current pipeline and related pipeline resources.**
  - `- download: none`

```
- deployment: DeploymentJob
  environment: env-name
  strategy:
    runOnce:  # runOnce | rolling | canary
      preDeploy:
        steps:
          ...
      deploy:
        steps:
          ...
```

Publish Unit Test Results

- Test results and code coverage are linked to the pipeline run.
  - Tests can also be linked to project dashboard widgets and Test Plans.



Summary Tests Code Coverage Associated pipelines

Summary

1 Run(s) Completed (1 Passed, 0 Failed)

3
Total tests

3 🟩 Passed
0 🟥 Failed
0 ● Others

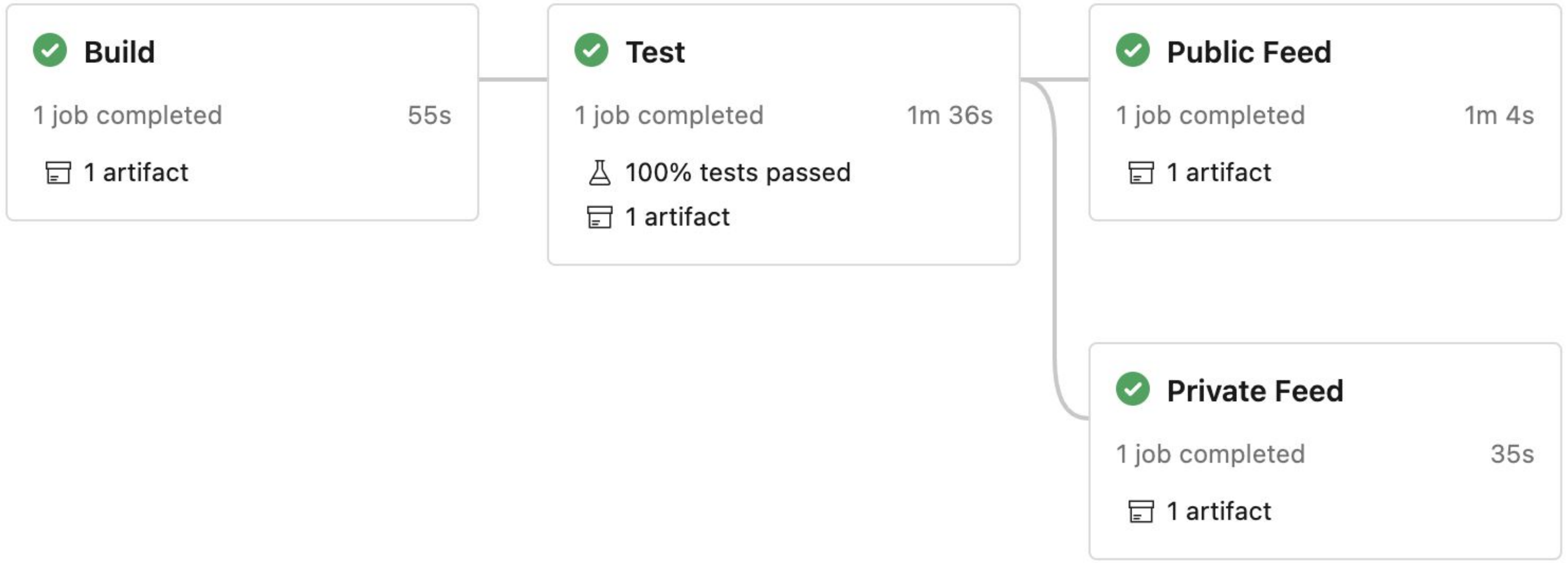
100%
Pass percentage

- Integrating code coverage helps establish a baseline and to know when code coverage drops.

```yaml
- task: PublishTestResults@2
 displayName: Publish Test Results
 inputs:
   testResultsFormat: 'VSTest'
   testResultsFiles: '$(Common.TestResultsDirectory)/**/*.trx'
```

```yaml
- task: PublishCodeCoverageResults@1
  displayName: Publish Code Coverage Results
  inputs:
    codeCoverageTool: Cobertura
    summaryFileLocation: '$(Common.TestResultsDirectory)/coverage.xml'
```

Publish to Package Feed



Pipeline artifacts vs Azure Artifacts

- Pipeline artifacts are tied to the pipeline that created them.
    - Can be downloaded for as long as the pipeline run is retained.
    - Not meant to be consumed outside of the pipeline.
- Azure Artifacts is a service where package feeds (Nuget, NPM) can be created and shared.
    - Feeds can be public (similar to nuget.org or npmjs.com) or private to your organization.
    - Possible pipeline artifact publish destination.

Publish to Azure Artifacts

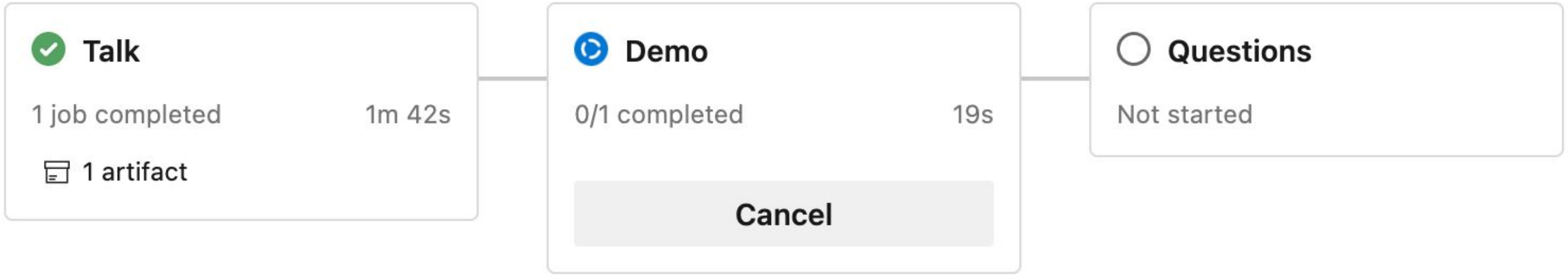
- Publish Nuget packages, NPM packages, universal packages (etc.) to one or multiple feeds.
    - Feeds are package-type independent.
    - Package versions are immutable.
- Azure Artifacts includes a symbol server.
    - Symbols still have the same retention policy as the build that generated them.

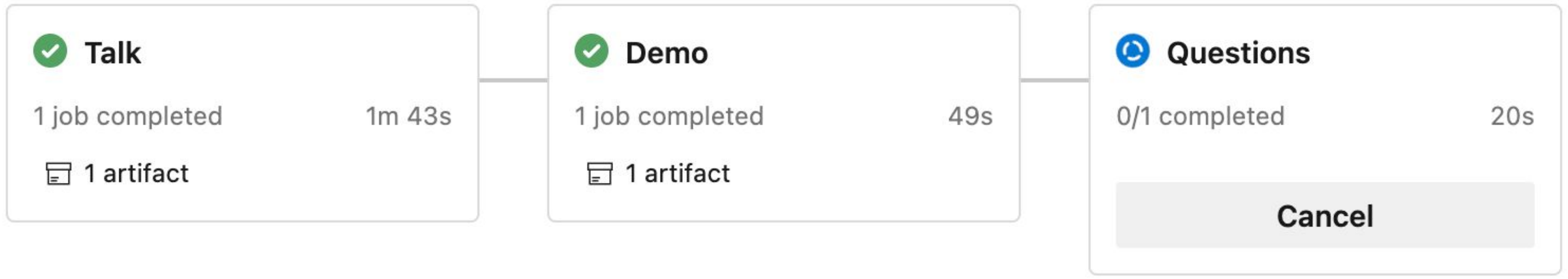
```yaml
 # Publish NPM pkg
- task: Npm@1
  inputs:
    command: publish
    publishRegistry: useFeed
    publishFeed: Project/Feed

# Publish NuGet pkg
- task: DotNetCoreCLI@2
  displayName: Publish Package
  inputs:
    command: 'push'
    packagesToPush: '$(Agent.BuildDirectory)/**/*.nupkg'
    nuGetFeedType: 'internal'
    feedPublish: 'Project/Feed'

 # Publish package symbols
- task: PublishSymbols@2
  displayName: Publish Numbers Symbols
  inputs:
    symbolsFolder: '$(Agent.BuildDirectory)'
    searchPattern: '**/bin/**/*.pdb'
    symbolServerType: 'TeamServices'
```

Universal Packages

- Universal Packages are a type of package that can store any set of files.
    - Models and textures
    - ML training data and models
- Optimized for upload and download of very large packages.
- A place to store artifacts with a lifetime separate of the pipeline build that produced them.
    - Preferable to keeping long living retention leases on old pipeline runs if those artifacts are shared.

```
- task: UniversalPackages@0
  displayName: 'Universal download'
  inputs:
    command: download
    vstsFeed: 'Project/Feed'
    vstsFeedPackage: 'packageName'
    vstsPackageVersion: 'packageVersion'
    downloadDirectory: '$(System.DefaultWorkingDirectory)'
```


Talk
1 job completed
1m 42s
1 artifact
Demo
0/1 completed
19s
Cancel
Questions
Not started



✓ **Talk**
1 job completed 1m 43s
1 artifact

✓ **Demo**
1 job completed 49s
1 artifact

Questions
0/1 completed 20s
Cancel