Certainly! Here's an updated Azure Pipeline definition that includes the specified requirements:

```yaml
trigger:
  branches:
    include:
      - master

variables:
  name: 'my-go-package'
  artifactName: '$(name)'
  artifactDocsName: '$(name)_docs'
  package: 'my-go-package'
  vstsFeedName: '<your_vsts_feed_name>'

jobs:
- job: TestAndBuild
  displayName: 'Test and Build'
  pool:
    vmImage: 'windows-latest'

  steps:
  - task: UseGoVersion@0
    inputs:
      version: '1.x'
    displayName: 'Use Go version'

  - task: Cache@2
    displayName: 'Cache Go modules'
    inputs:
      key: 'go-modules | "$(Agent.OS)" | go.sum'
      restoreKeys: 'go-modules | "$(Agent.OS)"'

  - script: go mod download
    displayName: 'Download Go modules'

  # Additional tasks for linting, validating, security checks, etc.

  - script: |
      go vet ./...
      go test -v -coverprofile=coverage.out ./...
    displayName: 'Lint, Validate, and Test'

  - task: PublishTestResults@2
    displayName: 'Publish Test Results'
    inputs:
      testResultsFormat: 'JUnit'
      testResultsFiles: '**/TEST-*.xml'
```

```
  - task: PublishCodeCoverageResults@1
    displayName: 'Publish Code Coverage'
    inputs:
      codeCoverageTool: 'Cobertura'
      summaryFileLocation: '$(System.DefaultWorkingDirectory)/coverage.out'

  - script: go build -o $(Build.ArtifactStagingDirectory)/$(artifactName)
    displayName: 'Build Application'

  - task: PublishPipelineArtifact@1
    displayName: 'Publish Artifact'
    inputs:
      targetPath: '$(Build.ArtifactStagingDirectory)/$(artifactName)'
      artifact: '$(artifactName)'
      publishLocation: 'pipeline'

- job: PullAndPublish
  displayName: 'Pull and Publish'
  dependsOn: TestAndBuild
  condition: eq(variables['Build.SourceBranch'], 'refs/heads/master')

  steps:
  - task: DownloadPipelineArtifact@2
    displayName: 'Download Artifact'
    inputs:
      patterns: '$(artifactName)'

  - task: UniversalPackages@0
    displayName: 'Publish Universal Package'
    inputs:
      command: 'publish'
      publishDirectory: '$(Pipeline.Workspace)'
      feedsToUsePublish: 'internal'
      vstsFeedPublish: '$(vstsFeedName)'
      packagePublishDescription: 'My Golang Package'
      versionOption: 'patch'
      packagePublishName: '$(package)'

  - script: go doc -all > $(Build.ArtifactStagingDirectory)/$(artifactDocsName).md
    displayName: 'Build Documentation'

  - task: PublishPipelineArtifact@1
    displayName: 'Publish Documentation'
    inputs:
      targetPath: '$(Build.ArtifactStagingDirectory)/$(artifactDocsName).md'
      artifact: '$(artifactDocsName)'
      publishLocation: 'pipeline'
```

To visualize the workflow using Mermaid.js, you can use the following Markdown code:

```
## Workflow

```mermaid
graph LR
    A[Test and Build] --> B((Pull and Publish))
    B --> C{Check branch}
    C -- 'is master' --> D(Publish Artifact)
    C -- 'is not master' --> E{Stop}
    A --> F(Build Documentation)
    F --> D
    style A fill:#FFCC66
    style

 B fill:#66CC99
    style C fill:#999999
    style D fill:#66CCFF
    style E fill:#CCCCCC
```

# Pipeline Documentation

## Test and Build

The first job in the pipeline is responsible for testing and building the Go package. It runs on a Windows agent (vmImage windows-latest). The following tasks are performed:

1. **Use Go version**: The appropriate Go version (1.x) is configured for the job.
2. **Cache Go modules**: The Go modules are cached to speed up subsequent pipeline runs.
3. **Download Go modules**: The Go modules are downloaded using the `go mod download` command.
4. **Lint, Validate, and Test**: Linting, validation, and testing are performed on the Go code. This includes running the `go vet` command for static analysis, and executing the tests with coverage calculation using the `go test` command. The results are published as pipeline artifacts.
5. **Publish Test Results**: The test results in JUnit format are published for visibility and analysis.
6. **Publish Code Coverage**: The code coverage results in Cobertura format are published to track code coverage trends.
7. **Build Application**: The Go code is compiled, and the resulting application is published as a pipeline artifact.
```

### Pull and Publish

The second job in the pipeline is triggered only if the previous job (Test and Build) succeeds and if the branch is the master branch. The following steps are executed:

1. **Download Artifact**: The artifact published in the previous job (Test and Build) is downloaded for further processing.
2. **Publish Universal Package**: The downloaded artifact is published as a Universal Package. The package is published to the specified VSTS feed, which is configured using global variables.
3. **Build Documentation**: The documentation for the Go package is generated from the source code using the `go doc -all` command. The generated documentation is saved as a Markdown file.
4. **Publish Documentation**: The generated documentation is published as a pipeline artifact for easy access and reference.

Please note that the graphical representation of the workflow may not be visible in the text-based format, but you can use Mermaid.js to render it in a Markdown viewer or online Markdown editors that support Mermaid.js.