**Business Central CI/CD Pipeline Setup Guide**

**Table of Contents**

1. Overview
2. Prerequisites
3. Phase 1: Azure Setup
4. Phase 2: Azure DevOps Artifacts Setup
5. Phase 3: Self-Hosted Agent Configuration
6. Phase 4: Build Pipeline Creation
7. Phase 5: Release Pipeline Setup
8. Phase 6: Deployment and Testing
9. Phase 7: Release Pipeline for BC Docker

**Overview**

This guide outlines the complete process for setting up a CI/CD pipeline for Business Central applications using Azure DevOps. The pipeline automatically builds .app files from AL source code and deploys them to Business Central cloud environments and Docker.

**Why CI/CD for Business Central?**

* **Automation**: Eliminates manual build and deployment processes
* **Consistency**: Ensures identical deployment process across environments
* **Traceability**: Maintains deployment history and rollback capabilities
* **Team Collaboration**: Supports multiple developers working on the same project

**Prerequisites**

Before starting, ensure you have:

**Access Requirements**

* **Azure DevOps Organization** with project creation permissions
* **Business Central Admin Center** access with environment management rights
* **Azure Active Directory** permissions to create app registrations
* **Virtual Machine or Local Machine** with admin rights for agent installation

**Software Requirements**

* **Visual Studio Code** with AL Language extension
* **Business Central Development** environment
* **PowerShell 5.1 or later**
* **Azure CLI** (will be installed during setup)

**Phase 1: Azure Setup**

**1.1 Create OAuth Application Registration**

**Purpose**: OAuth registration enables secure authentication between Azure DevOps and Business Central services without storing credentials in code.

**Steps:**

1. Navigate to **Azure Portal** → **Azure Active Directory** → **App registrations**
2. Click **"New registration"**
3. Configure the application:
   * **Name**: BC-DevOps-Integration
   * **Supported account types**: Single tenant
   * **Redirect URI**: Leave blank for now
4. After creation, note down:
   * **Application (Client) ID**
   * **Directory (Tenant) ID**
5. Go to **Certificates & secrets** → **New client secret**
6. Create secret and **immediately copy the value** (it won't be shown again)

**Why This Matters**: This OAuth app acts as the identity for your DevOps pipeline when it connects to Business Central APIs. Without proper authentication, deployments will fail.

**1.2 Configure API Permissions**

**Steps:**

1. In the app registration, go to **API permissions**
2. Add the following permissions:
   * **Dynamics 365 Business Central**: Financials.ReadWrite.All
   * **Microsoft Graph**: User.Read
3. Grant admin consent for your organization  
     
   A screenshot of a computer

   AI-generated content may be incorrect.  
   A screenshot of a computer

   AI-generated content may be incorrect.  
   4. Create a entra application using the same Client ID and give D365 full access and also ADLSE permission if Data Lake is installed.  
   A screenshot of a computer

   AI-generated content may be incorrect.

**Phase 2: Azure DevOps Artifacts Setup**

**2.1 Why Use Azure Artifacts?**

Azure Artifacts serves as a centralized repository for:

* **AL Compiler binaries**: Required to compile AL source code into .app files
* **AL Package dependencies**: Standard BC libraries and custom extensions your app depends on

**Benefits:**

* Ensures consistent compiler versions across all builds
* Eliminates "works on my machine" issues
* Provides version control for build tools

**2.2 Locate AL Compiler Files**

**Default Location:**

C:\Users\<USERNAME>\.vscode\extensions\ms-dynamics-smb.al-<VERSION>\bin\win32

**Example:**

C:\Users\vmadmin\.vscode\extensions\ms-dynamics-smb.al-15.2.1630495\bin\win32

**Steps:**

1. Open File Explorer and navigate to the path above
2. Copy the entire bin\win32 folder to C:\AL\ALCompiler
3. Locate your AL packages (usually in the same extension folder under al-packages)
4. Copy AL packages to C:\AL\ALPackages  
   A screenshot of a computer

   AI-generated content may be incorrect.  
   A screenshot of a computer

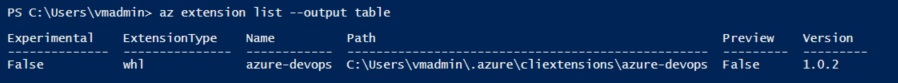
   AI-generated content may be incorrect.

**2.3 Install and Configure Azure CLI**In my case the files are present in VM so I have installed it in the VM itself

**Purpose**: Azure CLI allows PowerShell scripts to interact with Azure DevOps services programmatically.

**Steps:**

1. Download Azure CLI from: https://aka.ms/installazurecliwindowsx64
2. Install using default settings
3. Open PowerShell as Administrator and run:
4. az login
5. Verify login:
6. az account show
7. Install DevOps extension:
8. az extension add --name azure-devops
9. Verify installation:
10. az extension list --output table  
    A computer screen shot of a blue screen

    AI-generated content may be incorrect.  
    

**2.4 Create Azure DevOps Feeds**

**Purpose**: Feeds store and version your build dependencies, making them available to build agents.

**Steps:**

1. In Azure DevOps, navigate to **Artifacts**
2. Create two feeds:
   * **Feed Name**: TOBC-ALCompiler
   * **Feed Name**: TOBC-ALPackages
3. Set visibility to **"Organization"** for team access
4. Connect to feed using Universal Package: because we will be bringing our own package  
   A screenshot of a computer

   AI-generated content may be incorrect.  
   A screenshot of a computer

   AI-generated content may be incorrect.

**2.5 Upload Packages to Feeds**

**Upload AL Compiler:**

az artifacts universal publish \

--organization https://dev.azure.com/YourOrganization/ \

--project="YourProject" \

--scope project \

--feed TOBC-ALCompiler \

--name ALCompiler \

--version 0.0.1 \

--description "Business Central AL Compiler" \

--path "C:\AL\ALCompiler"

**Upload AL Packages:**

az artifacts universal publish \

--organization https://dev.azure.com/YourOrganization/ \

--project="YourProject" \

--scope project \

--feed TOBC-ALPackages \

--name ALPackages \

--version 0.0.1 \

--description "Business Central AL Packages" \

--path "C:\AL\ALPackages"

**Phase 3: Self-Hosted Agent Configuration**

**3.1 Why Self-Hosted Agents?**

**Microsoft-hosted agents** don't include Business Central development tools by default. **Self-hosted agents** provide:

* **Custom tool installation**: BC PowerShell modules, custom scripts
* **Better performance**: Dedicated resources, persistent caching
* **Cost control**: No per-minute charges for build time

**3.2 Create Agent Pool**

**Steps:**

1. In Azure DevOps, go to **Project Settings** → **Agent pools**
2. Click **"Add pool"**
3. Configure:
   * **Pool type**: Self-hosted
   * **Name**: BC-Build-Agents
   * **Grant access permission to all pipelines**: ✅  
     A screenshot of a computer

     AI-generated content may be incorrect.  
     A screenshot of a computer

     AI-generated content may be incorrect.

**3.3 Download and Configure Agent**

**Steps:**

1. In the agent pool, click **"New agent"**
2. Download the agent package (Windows x64)  
   A screenshot of a computer

   AI-generated content may be incorrect.
3. Extract to the folder(in my case I have created agent in my VM)
4. Open PowerShell as Administrator in the agent directory
5. Run configuration:
6. .\config.cmd  
   A screenshot of a computer

   AI-generated content may be incorrect.
7. Provide configuration details:
   * **Server URL**: <https://dev.azure.com/YourOrganization>

For e**.**g https://dev.azure.com/TravelOperations

* + **Authentication**: PAT (Personal Access Token)
  + A screenshot of a computer

    AI-generated content may be incorrect.
  + **Agent pool**: give the agent pool name which we entered ealier
  + **Agent name**: BC-Agent-01
  + **Work folder**: \_work
  + **Run as service**: Y
  + **Service account**: Use default (Network Service)  
    A screenshot of a computer

    AI-generated content may be incorrect.

**3.4 Install Business Central PowerShell Module**

**Purpose**: Required for deploying apps to BC cloud environments.

**Steps:**

1. On the agent machine, open PowerShell as Administrator
2. Install the module:
3. Install-Module BcContainerHelper -ForceInstall-Module CloudBcManager -Force

**3.5 Verify Agent Status**

**Steps:**

1. Check Windows Services for **"Azure Pipelines Agent (BC-Agent-01)"**
2. Verify status is **"Running"**
3. In Azure DevOps agent pool, confirm agent shows as **"Online"**

**Troubleshooting**: If agent won't start, change service **Log On** to **"Local System"**A screenshot of a phone

AI-generated content may be incorrect.A computer screen shot of a computer screen

AI-generated content may be incorrect.

**Phase 4: Build Pipeline Creation**

**4.1 Pipeline Purpose and Flow**

**Purpose**: Automatically compile AL source code into deployable .app files whenever code changes are committed or pipeline is run.

**Flow:**

Trigger → Download Dependencies → Compile → Generate Artifacts → Store for Release

**4.2 Create Build Pipeline**

**Steps:**

1. In Azure DevOps, go to **Pipelines** → **New pipeline**
2. Select your source repository (Azure Repos Git, GitHub, etc.)
3. Choose **"Classic editor"**
4. Configure pipeline settings:
   * **Agent pool**: BC-Build-Agents
   * **Trigger**: Disable continuous integration (manual builds initially)  
     A screenshot of a computer

     AI-generated content may be incorrect.  
     A screenshot of a computer

     AI-generated content may be incorrect.  
     A screenshot of a computer

     AI-generated content may be incorrect.

**4.3 Add Universal Package Download Tasks**

**Purpose**: Download AL compiler and packages from artifacts feeds to build agent.

**Task 1: Download AL Compiler**

* **Task**: Universal packages → Download
* **Feed**: TOBC-ALCompiler
* **Package**: ALCompiler
* **Version**: 0.0.1
* **Download directory**: $(System.ArtifactsDirectory)/ALCompiler  
  A screenshot of a computer

  AI-generated content may be incorrect.  
  A screenshot of a computer

  AI-generated content may be incorrect.

**Task 2: Download AL Packages**

* **Task**: Universal packages → Download
* **Feed**: TOBC-ALPackages
* **Package**: ALPackages
* **Version**: 0.0.1
* **Download directory**: $(System.ArtifactsDirectory)/ALPackages  
  A screenshot of a computer

  AI-generated content may be incorrect.

**4.4 Add PowerShell Build Script**

**Purpose**: Read app.json metadata, generate dynamic file names, and compile the application.

**Task Configuration:**

* **Task**: PowerShell
* **Type**: Inline
* **Script**:

# Read app.json to get app details

$appJsonPath = "$(System.DefaultWorkingDirectory)/app.json"

$appJson = Get-Content $appJsonPath | ConvertFrom-Json

# Extract app details

$appName = $appJson.name -replace '\s', '' # Remove spaces

$appPublisher = $appJson.publisher -replace '\s', '' # Remove spaces

$appVersion = $appJson.version

# Generate dynamic app file name

$appFileName = "${appPublisher}\_${appName}\_${appVersion}.app"

Write-Host "App Name: $($appJson.name)"

Write-Host "Publisher: $appVersion"

Write-Host "Version: $appVersion"

Write-Host "Generated App File Name: $appFileName"

# Create the App directory (this automatically cleans any existing content)

Remove-Item -Path "$(Pipeline.Workspace)/App" -Recurse -Force -ErrorAction SilentlyContinue

New-Item -ItemType Directory -Force -Path "$(Pipeline.Workspace)/App"

# Build the app with dynamic name

$(System.ArtifactsDirectory)/ALCompiler/alc.exe /project:$(System.DefaultWorkingDirectory) /out:$(Pipeline.Workspace)/App/$appFileName /packagecachepath:$(System.ArtifactsDirectory)/ALPackages/

# Ensure only our current app file name is stored (overwrite any existing file)

$appFileName | Out-File -FilePath "$(Pipeline.Workspace)/App/appfilename.txt" -Force

Write-Host "Build completed successfully. App file: $appFileName"  
  
A screenshot of a computer

AI-generated content may be incorrect.

**Why Dynamic Naming?**: This ensures each app version has a unique filename, preventing deployment conflicts and enabling proper version tracking.

**4.5 Add Publish Artifacts Task**

**Purpose**: Make the compiled .app file available to release pipelines.

**Task Configuration:**

* **Task**: Publish Pipeline Artifacts
* **File or directory path**: $(Pipeline.Workspace)/App
* **Artifact name**: AppFile  
  A screenshot of a computer

  AI-generated content may be incorrect.
* Pipeline setup is done, save it and run the pipeline  
  A screenshot of a computer

  AI-generated content may be incorrect.

**Phase 5: Release Pipeline Setup**

**5.1 Why Separate Build and Release Pipelines?**

**Benefits of Separation:**

* **Reusability**: One build can deploy to multiple environments
* **Control**: Manual approval gates for production deployments
* **Flexibility**: Different deployment strategies per environment
* **Auditing**: Clear separation between compilation and deployment logs

**5.2 Create Release Pipeline**

**Steps:**

1. Go to **Pipelines** → **Releases** → **New pipeline**
2. Start with **"Empty job"**
3. Name the pipeline: BC App Deployment  
   A screenshot of a computer

   AI-generated content may be incorrect.  
   A screenshot of a computer

   AI-generated content may be incorrect.

**5.3 Add Build Artifact**

**Purpose**: Link the release pipeline to your build pipeline's outputs.

**Configuration:**

* **Source type**: Build
* **Project**: Your project
* **Source (build pipeline)**: Your build pipeline name
* **Default version**: Latest
* **Source alias**: \_BuildArtifact
* **Continuous deployment trigger**: Disabled (manual releases initially)  
  A screenshot of a computer

  AI-generated content may be incorrect.  
  A screenshot of a computer

  AI-generated content may be incorrect.  
  The same for Pipeline, we will disable the CI in pipeline so it wont create the app file whenever we commit into the Branch  
  A screenshot of a computer

  AI-generated content may be incorrect.

**5.4 Create Deployment Stage**

**Steps:**

1. Click **"Add a stage"**
2. Choose **"Empty job"**
3. Name: Deploy to Development
4. Configure stage:
   * **Agent pool**: BC-Build-Agents  
     A screenshot of a computer

     AI-generated content may be incorrect.

**5.5 Add Deployment Tasks**

**Task 1: Read App File Name**

# Read the app file name from the specific build artifact

$appFileNamePath = "$(System.DefaultWorkingDirectory)/\_BuildArtifact/AppFile/appfilename.txt"

if (Test-Path $appFileNamePath) {

$appFileName = Get-Content $appFileNamePath

Write-Host "App File Name from build: $appFileName"

Write-Host "##vso[task.setvariable variable=app\_file\_name]$appFileName"

} else {

Write-Error "appfilename.txt not found in build artifacts"

exit 1

}

**Task 2: Deploy to Business Central**

# Use Azure DevOps variables (replace these with your actual variable names)

$clientId = "$(client\_id)"

$clientSecret = "$(client\_secret)"

$tenantId = "$(tenant\_id)"

$environment = "$(env\_name)"

# Construct the full path to the app file

$appFile = "$(System.DefaultWorkingDirectory)/\_BuildArtifact/AppFile/$(app\_file\_name)"

# Verify the app file exists

if (-not (Test-Path $appFile)) {

Write-Error "App file not found at path: $appFile"

exit 1

}

Write-Host "Using the following parameters:"

Write-Host "Client ID: $clientId"

Write-Host "Tenant ID: $tenantId"

Write-Host "Environment: $environment"

Write-Host "App File Path: $appFile"

try {

# Create authentication context

Write-Host "Creating authentication context..."

$authContext = New-BcAuthContext -clientID $clientId -clientSecret $clientSecret -tenantID $tenantId

# Publish the Business Central app

Write-Host "Publishing Business Central app..."

Publish-BcContainerApp `

-bcAuthContext $authContext `

-environment $environment `

-appFile $appFile `

-Verbose

Write-Host "App deployment completed successfully!"

} catch {

Write-Error "Error during app deployment: $($\_.Exception.Message)"

Write-Error "Full error details: $\_"

exit 1

}  
A screenshot of a computer

AI-generated content may be incorrect.  
A screenshot of a computer

AI-generated content may be incorrect.

**5.6 Configure Release Variables**

**Purpose**: Store sensitive information and environment-specific settings securely.

**Variables to Create:**

* client\_id: Your OAuth application client ID
* client\_secret: Your OAuth application client secret (mark as secret)
* tenant\_id: Your Azure AD tenant ID
* env\_name: Business Central environment name (e.g., "Production", "Sandbox")

**Steps:**

1. In release pipeline, go to **Variables** tab
2. Add each variable with appropriate values
3. Mark client\_secret as **secret** (lock icon)  
   A screenshot of a computer

   AI-generated content may be incorrect.

**Phase 6: Deployment and Testing**

**6.1 Execute First Build**

**Steps:**

1. Go to your build pipeline
2. Click **"Run pipeline"**
3. Monitor build progress and logs
4. Verify artifacts are published successfully

**6.2 Execute First Release**

**Steps:**

1. Go to your release pipeline
2. Click **"Create release"**
3. Select the latest build artifact
4. Monitor deployment progress
5. Check Business Central environment for deployed app  
   A screenshot of a computer

   AI-generated content may be incorrect.

**6.3 Verify Deployment**

**In Business Central:**

1. Open Business Central environment
2. Go to **Extension Management**
3. Verify your app appears in the installed extensions list
4. Test app functionality

**7 Release Pipeline for BC Docker**

Purpose: Deploy .app to local/containerized BC environment.

**7.1 Same artifact from CI pipeline.**

A screenshot of a computer

AI-generated content may be incorrect.

**7.2 Add Powershell tasks**Read .app filename and version.

Deploy using BCContainerHelper:  
- Uninstall previous version  
 - Unpublish old versions  
 - Publish, sync, install new version  
 - Verify deployment success.  
A screenshot of a computer

AI-generated content may be incorrect.

Script   
Script to read file name   
# Read the app file name from the build artifact

$appFileNamePath = "$(System.DefaultWorkingDirectory)/\_TOBC-CI-PPM/AppFile/appfilename.txt"

Write-Host "Looking for appfilename.txt at: $appFileNamePath"

if (Test-Path $appFileNamePath) {

    $appFileName = Get-Content $appFileNamePath

    Write-Host "✅ App File Name from build: $appFileName"

    # Extract version from filename (e.g., TravelOperations\_TravelOperationsBusiness\_1.31.260.26.app)

    $versionMatch = [regex]::Match($appFileName, '\_(\d+\.\d+\.\d+\.\d+)\.app$')

    if ($versionMatch.Success) {

        $appVersion = $versionMatch.Groups[1].Value

        Write-Host "✅ Extracted App Version: $appVersion"

        Write-Host "##vso[task.setvariable variable=appVersion]$appVersion"

        Write-Host "##vso[task.setvariable variable=appFileName]$appFileName"

    } else {

        Write-Error "Could not extract version from filename: $appFileName"

        exit 1

    }

} else {

    Write-Host "❌ appfilename.txt not found at expected path"

    Write-Host "Debugging artifact structure..."

    # Debug: Show what's actually in the artifacts

    Write-Host "Contents of $(System.DefaultWorkingDirectory):"

    Get-ChildItem -Path "$(System.DefaultWorkingDirectory)" -Recurse | ForEach-Object {

        Write-Host "  - $($\_.FullName)"

    }

    exit 1

}

Script 2   
To deploy into Container   
try {

    # App configuration

    $AppId = "$(appId)"

    $AppName = "$(appName)"

    $AppVersion = "$(appVersion)"

    $ContainerName = "$(containerName)"

    $TenantId = "$(tenantId)"

    $Username = "$(username)"

    $Password = "$(bcPassword)"

    Write-Host "Deploying $AppName v$AppVersion to $ContainerName"

    # Get app file from build artifacts - CORRECTED PATH

    $artifactPath = "$(System.DefaultWorkingDirectory)/\_TOBC-CI-PPM/AppFile"

    if (-not (Test-Path $artifactPath)) {

        throw "Artifact directory not found: $artifactPath"

    }

    # Find the app file

    $appFileName = "$(appFileName)"

    $AppFilePath = Join-Path $artifactPath $appFileName

    if (-not (Test-Path $AppFilePath)) {

        # Debug and fallback

        Write-Host "App file not found at: $AppFilePath"

        Write-Host "Available files in artifact directory:"

        Get-ChildItem -Path $artifactPath | ForEach-Object {

            Write-Host "  - $($\_.Name)"

        }

        # Fallback: find any .app file

        $appFile = Get-ChildItem -Path $artifactPath -Filter "\*.app" | Sort-Object LastWriteTime -Descending | Select-Object -First 1

        if ($appFile) {

            $AppFilePath = $appFile.FullName

            Write-Host "✅ Using fallback app file: $($appFile.Name)"

        } else {

            throw "No app file found in artifacts"

        }

    } else {

        Write-Host "✅ Using app file: $appFileName"

    }

    # Import BCContainerHelper module

    Write-Host "Importing BCContainerHelper module..."

    if (-not (Get-Module -ListAvailable -Name BCContainerHelper)) {

        Write-Host "Installing BCContainerHelper module..."

        Install-Module -Name BCContainerHelper -Force -AllowClobber -Scope CurrentUser

    }

    Import-Module BCContainerHelper -Force

    # Setup credentials

    $SecurePassword = ConvertTo-SecureString $Password -AsPlainText -Force

    $Credential = New-Object PSCredential ($Username, $SecurePassword)

    # Remove existing installations

    Write-Host "Removing existing installations..."

    try {

        $InstalledApps = Get-BcContainerAppInfo -containerName $ContainerName -tenant $TenantId -installedOnly

        foreach ($App in $InstalledApps | Where-Object { $\_.AppId -eq $AppId }) {

            Write-Host "Uninstalling app: $($App.Name) v$($App.Version)"

            UnInstall-BcContainerApp -containerName $ContainerName -name $App.Name -version $App.Version -tenant $TenantId -force

        }

    }

    catch {

        Write-Warning "Could not remove existing installations: $($\_.Exception.Message)"

    }

    # Unpublish previous versions

    Write-Host "Unpublishing previous versions..."

    try {

        $PublishedApps = Get-BcContainerAppInfo -containerName $ContainerName | Where-Object { $\_.AppId -eq $AppId }

        foreach ($App in $PublishedApps) {

            Write-Host "Unpublishing app: $($App.Name) v$($App.Version)"

            UnPublish-BcContainerApp -containerName $ContainerName -name $App.Name -version $App.Version

        }

    }

    catch {

        Write-Warning "Could not unpublish previous versions: $($\_.Exception.Message)"

    }

    # Deploy new version

    Write-Host "Publishing new version..."

    Publish-BcContainerApp -containerName $ContainerName -appFile $AppFilePath -credential $Credential -SkipVerification

    Write-Host "Syncing and installing..."

    Sync-BcContainerApp -containerName $ContainerName -appName $AppName -appVersion $AppVersion -tenant $TenantId -force

    Start-BcContainerAppDataUpgrade -containerName $ContainerName -appName $AppName -appVersion $AppVersion -tenant $TenantId

    Install-BcContainerApp -containerName $ContainerName -appName $AppName -appVersion $AppVersion -tenant $TenantId -force

    # Verify deployment

    Write-Host "Verifying deployment..."

    $FinalApp = Get-BcContainerAppInfo -containerName $ContainerName -tenant $TenantId -installedOnly | Where-Object {

        $\_.AppId -eq $AppId -and $\_.Version -eq $AppVersion

    }

    if ($FinalApp) {

        Write-Host "✅ Deployment successful: $($FinalApp.Name) v$($FinalApp.Version)"

    } else {

        throw "❌ Deployment verification failed"

    }

}

catch {

    Write-Error "❌ Deployment failed: $($\_.Exception.Message)"

    Write-Host "Exception details: $($\_.Exception | Format-List | Out-String)"

    exit 1

}

finally {

    Write-Host "Deployment process completed."

}

**7.3 Add Required variables**

A screenshot of a computer

AI-generated content may be incorrect.

**Troubleshooting**

**Common Build Issues**

**Problem**: AL Compiler not found **Solution**: Verify ALCompiler artifact download task and file paths

**Problem**: Package dependencies missing **Solution**: Ensure ALPackages artifact contains all required dependencies

**Common Deployment Issues**

**Problem**: Authentication failed **Solution**: Verify OAuth client credentials and API permissions

**Problem**: Environment not found **Solution**: Check environment name spelling and case sensitivity

**Problem**: App installation failed **Solution**: Review dependency requirements and app manifest

**Monitoring and Logs**

**Build Logs**: Available in build pipeline run history   
**Release Logs**: Available in release deployment history

**Best Practices**

**Process**

* Version your artifacts consistently

**Maintenance**

* Keep AL compiler versions updated
* Regular cleanup of old artifacts

**Conclusion**

This CI/CD pipeline provides automated building and deployment of Business Central applications, improving development efficiency and reducing deployment errors. The separation of concerns between building and releasing enables flexible deployment strategies while maintaining security and auditability.

For questions or issues not covered in this guide, consult the Azure DevOps documentation or contact your DevOps administrator.