



# Runbook for CI/CD with Azure DevOps for Cora SeQUENCE projects

For Cora SeQUENCE V9.x

December 2021

## Contents

What is CI/CD with Azure DevOps? .....	4
Want to learn more?.....	5
Application life cycle: from the developer's machine to the production environment.....	5
The CI/CD process at a very high level .....	7
CI/CD for Cora SeSequence .....	7
Setup prerequisites .....	8
Update workflows or customize applications .....	9
Update workflows.....	10
Executing the pipelines.....	17
Commit to Azure DevOps.....	18
Deployment methods .....	21
Best practices.....	22
Execute the Unit Package pipeline .....	23
Execute the Processes Release pipeline .....	26
Customize Cora SeSequence .....	33

## Notice

All rights reserved. No part of this document (including the text, images, graphics or the selection or arrangement of content, which forms an original compilation), may be reproduced or transmitted in any material form or by any means, electronic or mechanical, including photocopying or recording in any medium (whether or not transiently) without the written permission of the copyright holder. Such written permission must also be obtained before any part of this publication is stored in a retrieval system of any nature. The same shall apply to the export of this publication from India, and a violation of this condition will lead to civil and criminal prosecution.

## What is CI/CD with Azure DevOps?

CI/CD with Azure DevOps is an automated platform on the Azure cloud that enables you to continuously integrate developed applications (projects) with the Azure cloud and continuously deploy the developed application to any Cora SeQUENCE environment like UAT or production (Live) seamlessly via pipelines. After all the pipelines are created and set up properly, the whole process can run automatically without any manual intervention.

Continuous Integration (CI) is the practice used by development teams to automate the merging and testing of code. Implementing CI helps to catch bugs early in the development cycle, which makes it less expensive to fix. Automated tests run as part of the CI process to ensure quality. Artifacts are produced from CI systems and fed to release processes to drive frequent deployments. The Build service in the Azure DevOps Server helps you set up and manage CI for your applications.

Continuous Delivery (CD) is a process by which code is built, tested, and deployed to one or more test and production environments. Deploying and testing in multiple environments drives quality. CI systems produce the deployable artifacts including infrastructure and apps. Automated release processes, CD processes, consume these artifacts to release new versions and fixes to existing systems. Monitoring and alerting systems run continually to drive visibility into the entire CD process. The Release service in the Azure DevOps Server helps you set up and manage CD for your applications.

To configure CI and CD, you create pipeline definitions. A pipeline definition is a representation of the automation process that you want to run to build and test your application. The automation process is defined as a collection of tasks. Azure DevOps pipelines have a few inbuilt tasks to help you build and test your application. You can further customize your pipelines by adding your own command lines, PowerShell commands, or Shell scripts in your automation.

## Want to learn more?

[What is CI/CD?](#)

[What is Azure DevOps?](#)

[CI/CD with Azure DevOps](#)

## Application life cycle: from the developer's machine to the production environment

1. The developer (with permissions) creates a project in his/her local machine.
2. The CI/CD System Admin creates a repository (repos) in the local machine using GIT commands (Source control system).

A repository, also called repos for short, is a folder with version control tools that you can use to manage your code.

Version control systems are software that help you track changes you make in your code over time. As you edit your code, you tell the version control system to take a snapshot of your files. The version control system saves that snapshot permanently so you can recall it later if you need it.

Use version control to save your work and coordinate code changes across your team.

3. The developer creates an Azure repo in Azure DevOps (cloud server).

Note that the Azure repo in Azure DevOps is the implementation of GIT in the Azure cloud.

4. The developer then creates a CI pipeline in the Azure portal to push the project from the developer's machine repo into the Azure DevOps repos. The result is an artifact that is stored in the Azure cloud.

The pipeline contains build tasks that are composed of tasks like getting the source code, building the solution, running tests, packaging artifacts, and finally publishing the artifacts.

A pipeline is similar to a script with a list of command tasks that are executed in sequence. They could include commands such copy file, delete file, execute a PowerShell command, and others. The artifact is the packaged software that results after the pipeline completely executes all its commands and tasks.

5. The CI pipeline (that is, the build job and tasks) is run by an agent machine. Azure usually provides a VM (Microsoft hosted machine) that is used as the agent machine. However, the developer can use his/her own hosted machine as the agent machine.
6. The developer creates a CD pipeline (consisting of a list of tasks) that is used for deploying and testing the artifact into multiple environments, like Dev, QA, or Production.

Note that the CD pipeline is also run by the agent machine.

7. The developer can add approvals at certain stages so that projects can be approved by QA. For example, before the project gets deployed into production.

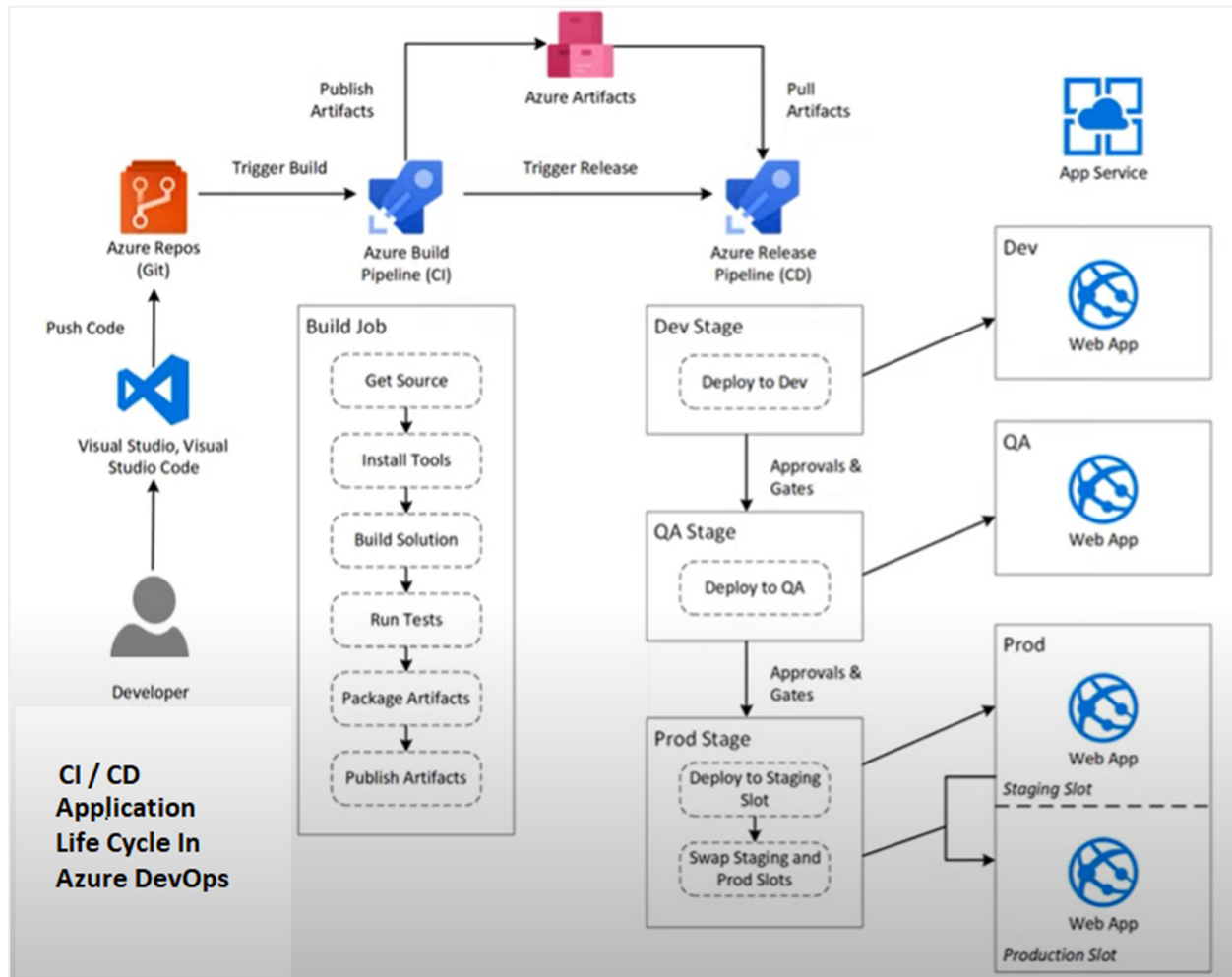
The CI/CD process can be fully or partially automated.

In a fully automated CI/CD process, there's no user intervention. As soon as the user pushes the source code in the GIT repo of the local machine, the whole CD/CD process gets executed and all the environments (UAT, Production, and others) are updated as per the configured CI/CD process.

In a partially automated CI/CD process, some of the processes, like copying the files to the Azure repo, are done manually. Or the pipelines are triggered manually by the user.

## The CI/CD process at a very high level

Deploying an application to different environments using the CI/CD process.



## CI/CD for Cora SeSequence

In order to set up the pipelines, the **DevOps System Admin** needs relevant project details, such as server name, database username and password, Administration folder name, and SeSequence Administration URLs.

These details are usually provided by the **Tech Lead** or **Project Manager** who created the Cora SeSequence project on the Azure DevOps platform.

## Setup prerequisites

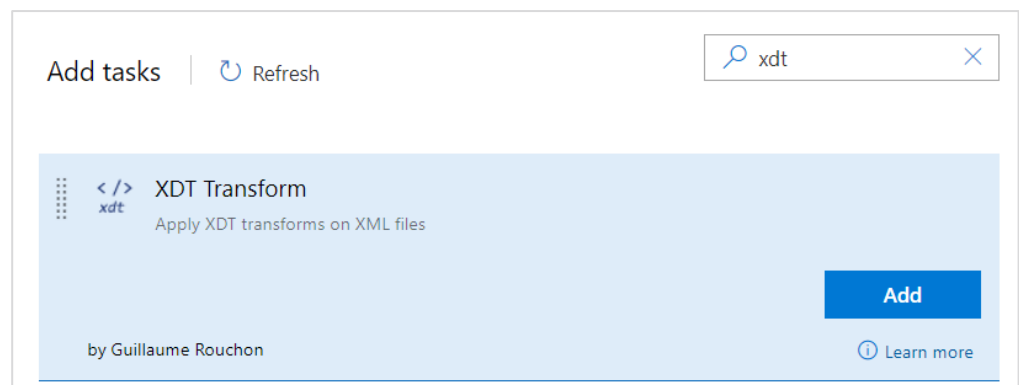
The **Tech Lead** or **Project Manager** needs to set up the following components:

- **Servers:** The DEV, TEST, and PROD servers need to be setup.
- **Users:** The Tech Lead or Project Manager needs to create users in the Azure DevOps platform for whoever needs to access the Azure portal. Because all users need access to the Repos section, they need to have at least the basic license (not the Stakeholders license).
- **Network:** There needs to be physical network connectivity between the servers and the Azure DevOps application. Several URLs need to be whitelisted to ensure a seamless experience with Azure DevOps.

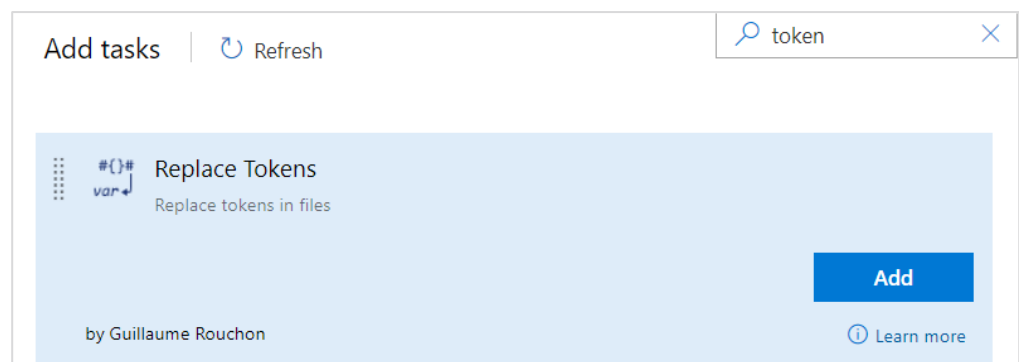
For more information, see this article: [Allowed IP addresses and domain URLs](#)

- **Additional tasks:** The Tech Lead or Project Manager needs to add the following tasks to the Azure DevOps platform through the Marketplace:

- XDT Transform

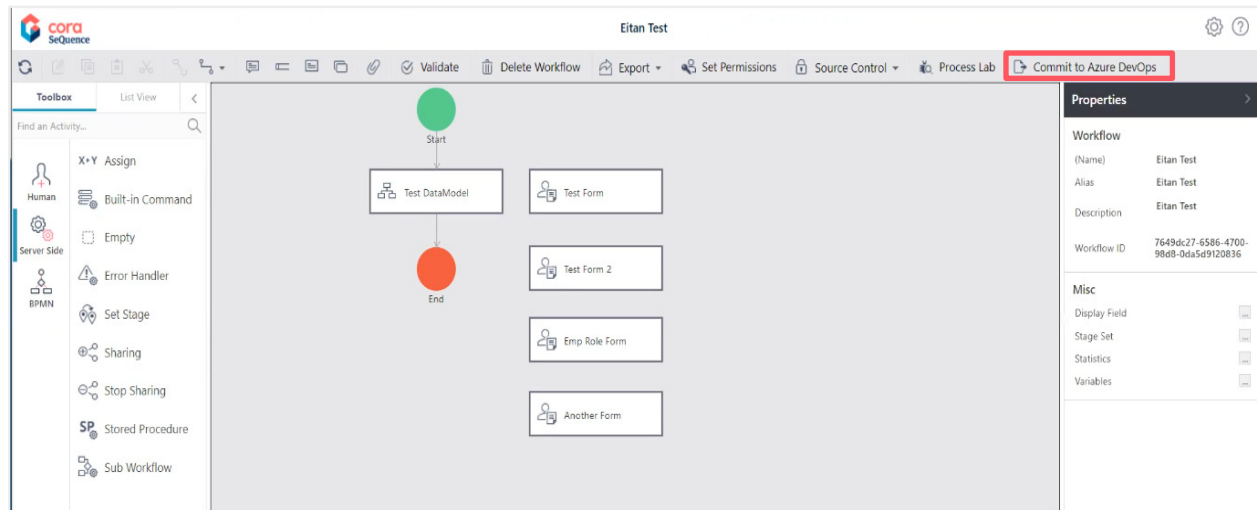


- Replace Tokens





After the project is configured, the **DevOps System Admin** can set up the pipelines in the Azure DevOps platform. The setup includes installing the CI/CD tool on each server and making the **Commit to Azure DevOps** button available in the Administration site.

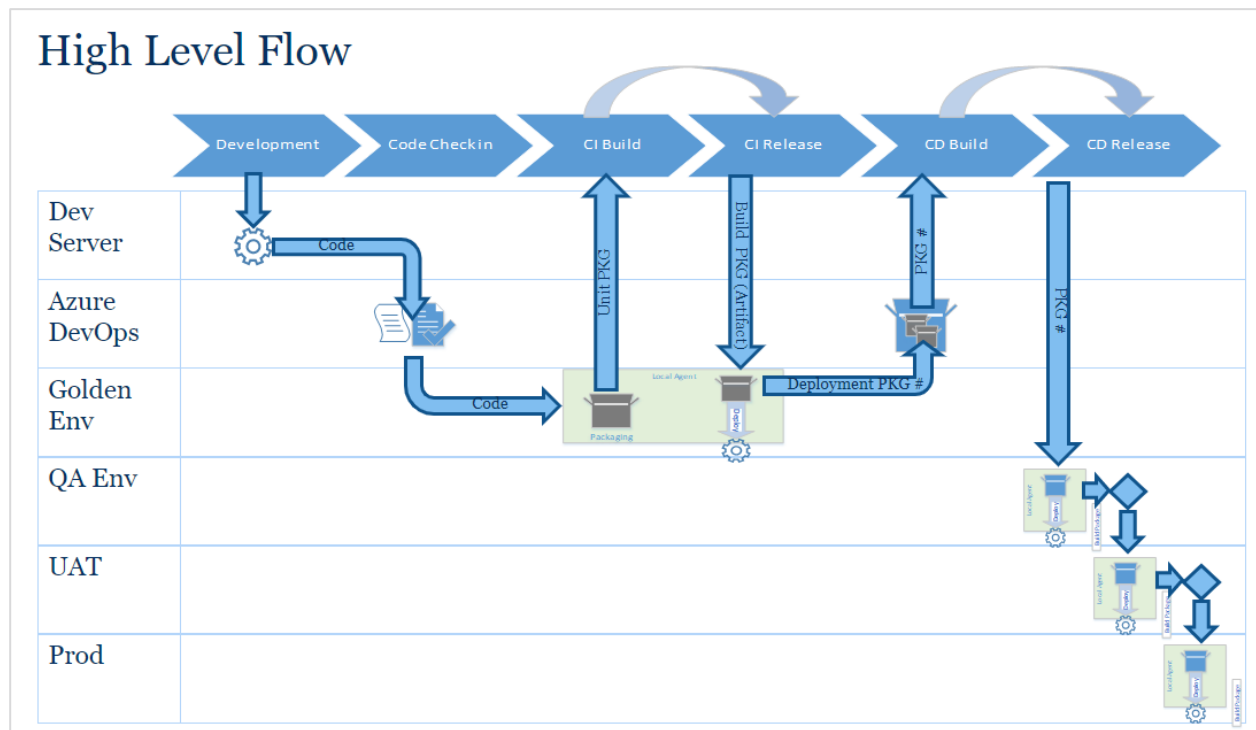


## Update workflows or customize applications

You use CI/CD pipelines for two main purposes:

- Update workflows or workflow-related applications.
- Customize Cora SeSequence features.

The following diagram provides a very high-level overview of the CI/CD process for Cora SeSequence projects and applications.



## Note

The **Golden environment** is where the artifact (software package) is stored after running the CI pipelines. The artifact is used by the CD pipeline to propagate the software to other environments.

## Update workflows

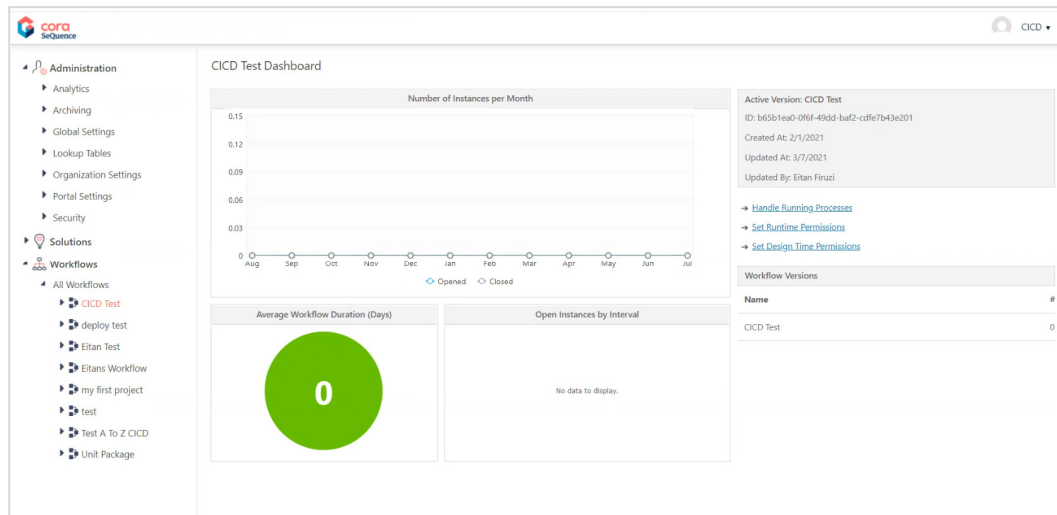
You can use two pipelines to deploy workflows and workflow elements on different environments.

- **Unit Package (CI Pipeline):** packages workflows that have been committed to Azure and creates an artifact in the project's repo.
- **Processes Release (CD Pipeline):** takes the artifact from the Azure repo and deploys it to the target environment as specified in the pipeline

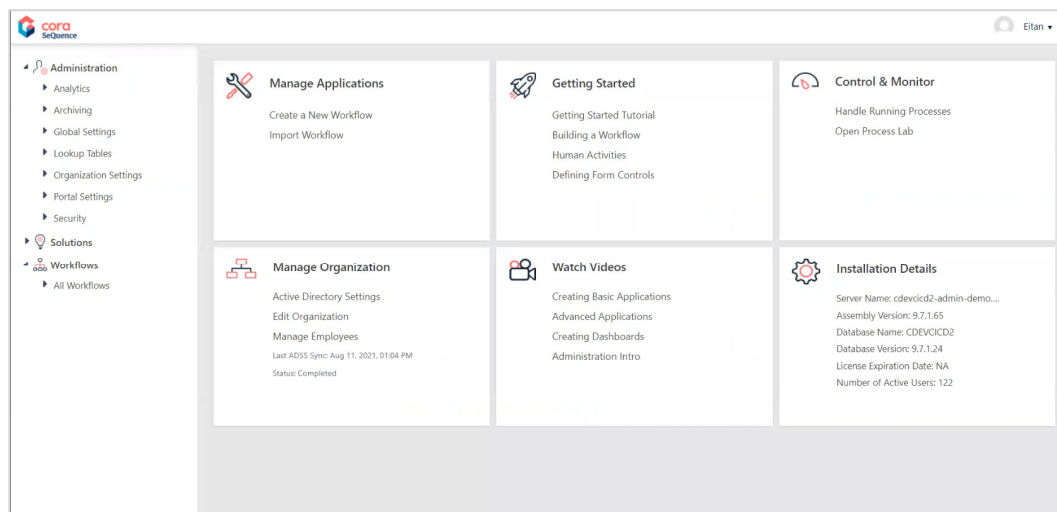
We are going to use an **example** to demonstrate how to use the two pipelines.

Assume you already have two Cora SeSequence environments set up:

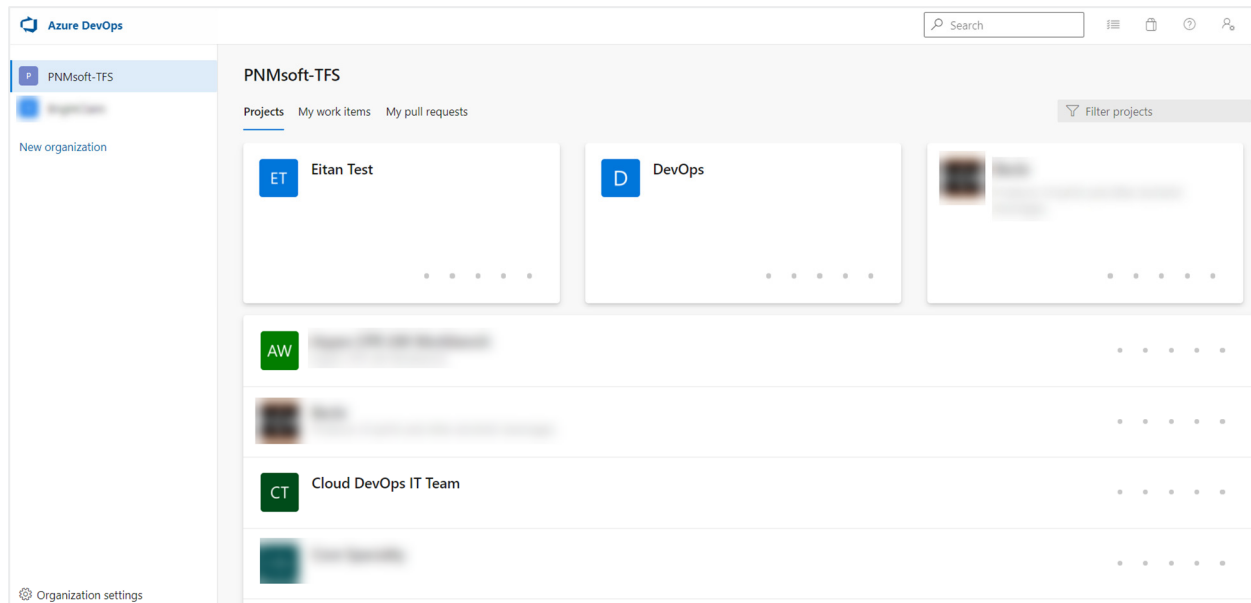
- Environment 1: CDEVICD1 (Dev server)



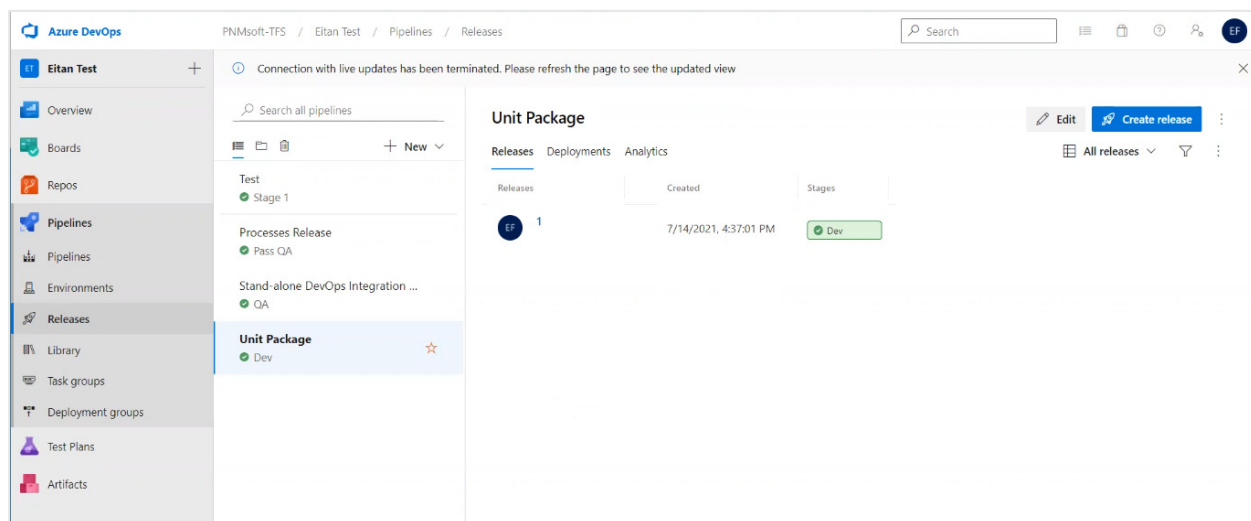
- Environment 2: CDEVICID2 (Test server)



Let's have a look at the pipelines in the Azure DevOps Portal.



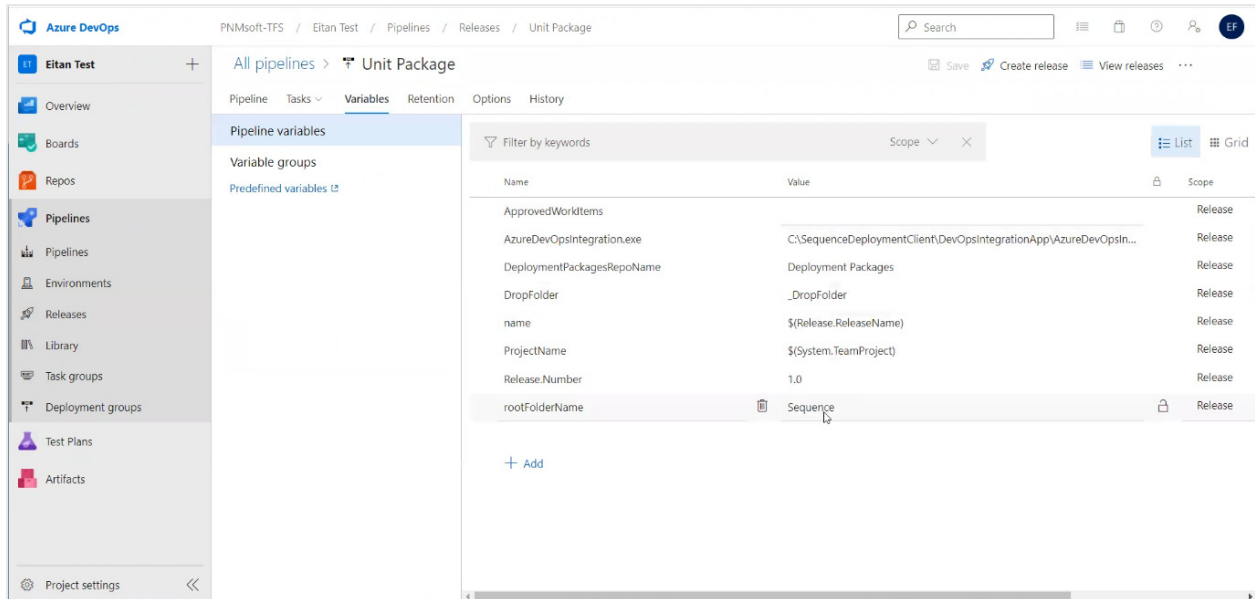
We have an Organization, PNMsoft-TFS, and under that Organization, we have the project Eitan Test.



Under the Eitan Test project, there are a few pipelines, including Unit Package (CI Pipeline) and Processes Release (CD Pipeline).

## The Unit Package pipeline

Variables available in the Unit Package (CI) pipeline.



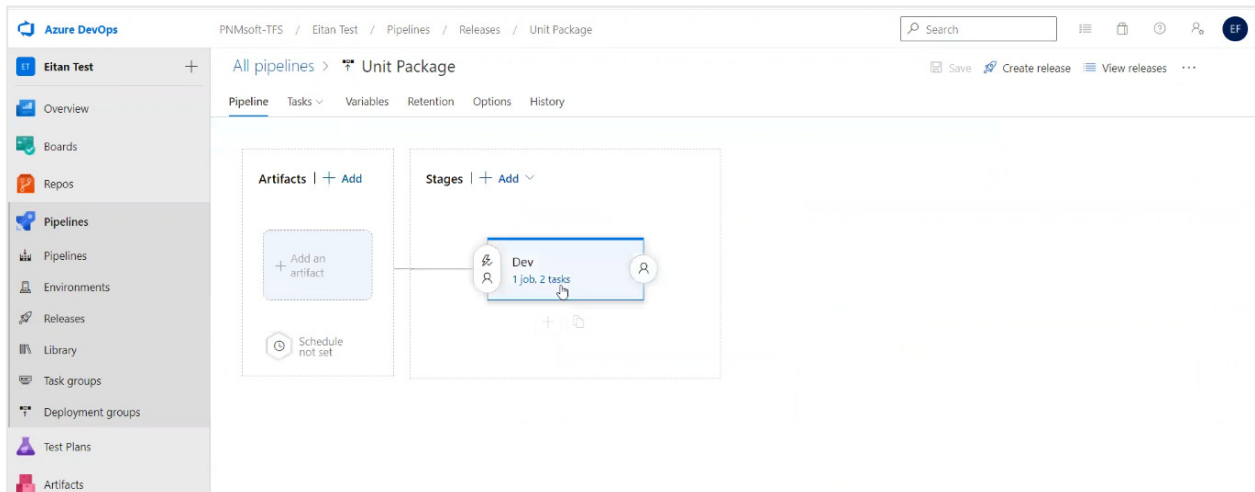
The screenshot shows the Azure DevOps interface for the 'Unit Package' pipeline. The left sidebar contains navigation options like Overview, Boards, Repos, Pipelines, Environments, Releases, Library, Task groups, Deployment groups, Test Plans, and Artifacts. The main area displays the 'Variables' tab for the 'Unit Package' pipeline. A table lists the pipeline variables:

Name	Value	Scope
ApprovedWorkItems		Release
AzureDevOpsIntegration.exe	C:\SequenceDeploymentClient\DevOpsIntegrationApp\AzureDevOpsIn...	Release
DeploymentPackagesRepoName	Deployment Packages	Release
DropFolder	..DropFolder	Release
name	\$(Release.ReleaseName)	Release
ProjectName	\$(System.TeamProject)	Release
Release.Number	1.0	Release
rootFolderName	Sequence	Release

Below the table is a '+ Add' button.

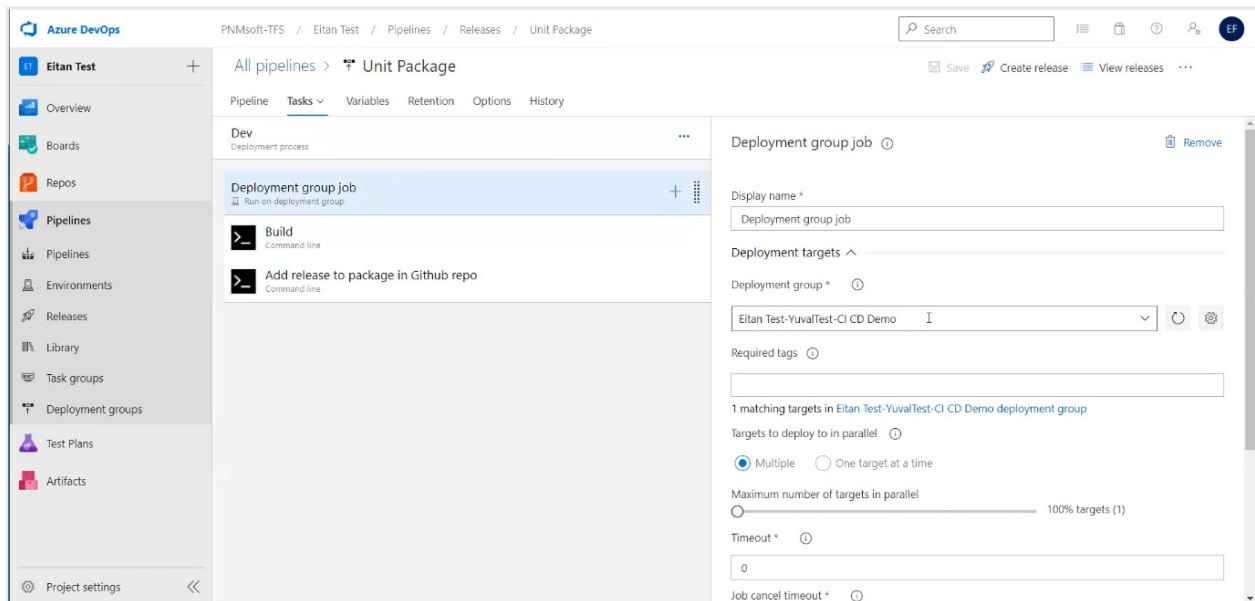
We use the variables to specify the repos of the source project and destination repos of the artifact and other properties needed for the CI process. This configuration is currently done by the PS CI/CD System Admin.

Graphic view of the Unit Package pipeline



The screenshot shows the 'Pipeline' tab for the 'Unit Package' pipeline. The main area displays a visual representation of the pipeline. On the left, under 'Artifacts', there is a button '+ Add an artifact'. On the right, under 'Stages', there is a stage named 'Dev' with '1 job, 2 tasks'. A 'Schedule not set' icon is visible at the bottom left.

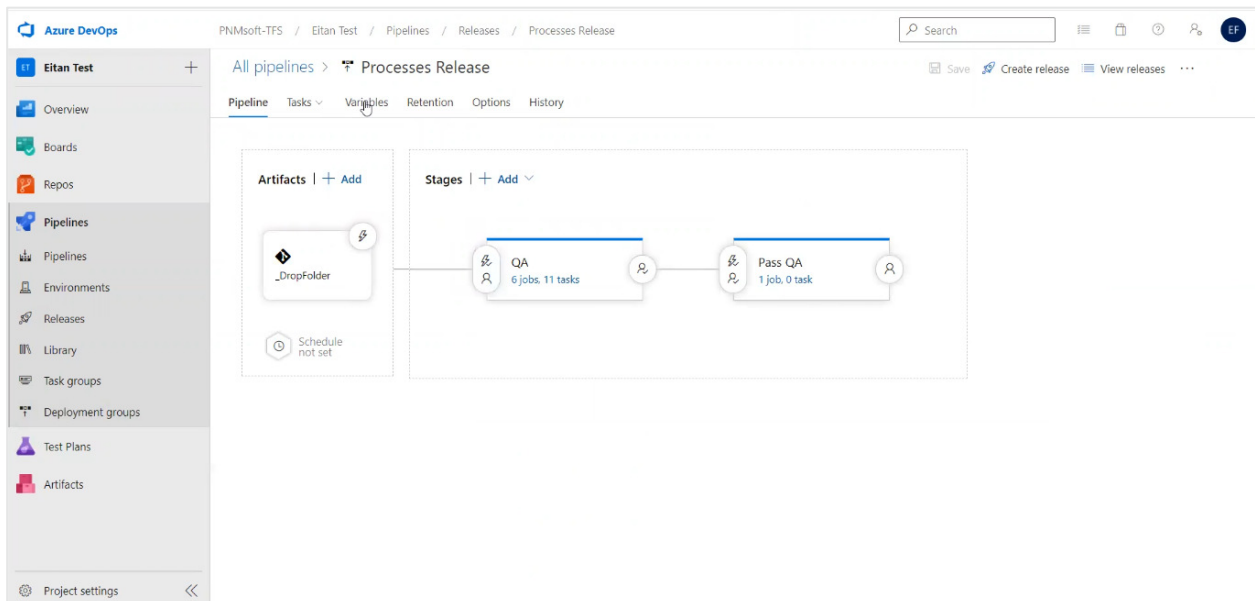
## Unit Package pipeline tasks



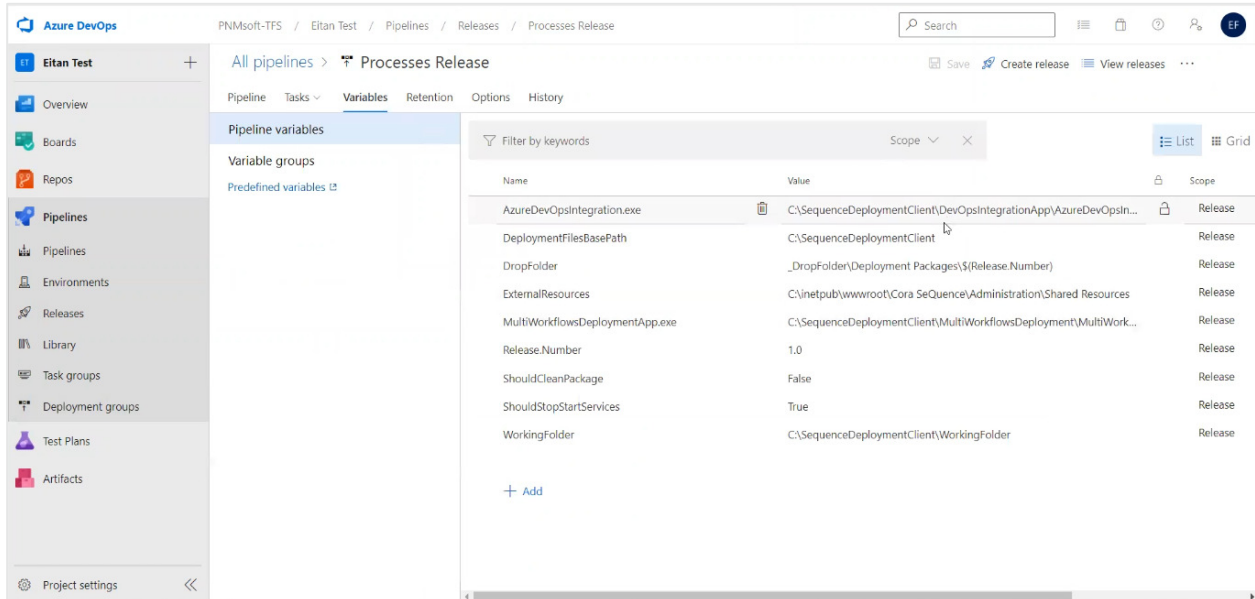
The screenshot shows the Azure DevOps interface for the 'Unit Package' pipeline. The left sidebar lists various project components, with 'Pipelines' selected. The main area displays the 'Dev' deployment process, which includes two tasks: 'Build' (Command line) and 'Add release to package in Github repo' (Command line). The right panel shows the configuration for the 'Deployment group job', including a display name, deployment targets, required tags, and deployment options.

## The Processes Release pipeline

Graphic view of the Process Release (CD) pipeline. There are two stages in the pipeline: QA and Pass QA.



## Variables available in the Process Release (CD) pipeline.

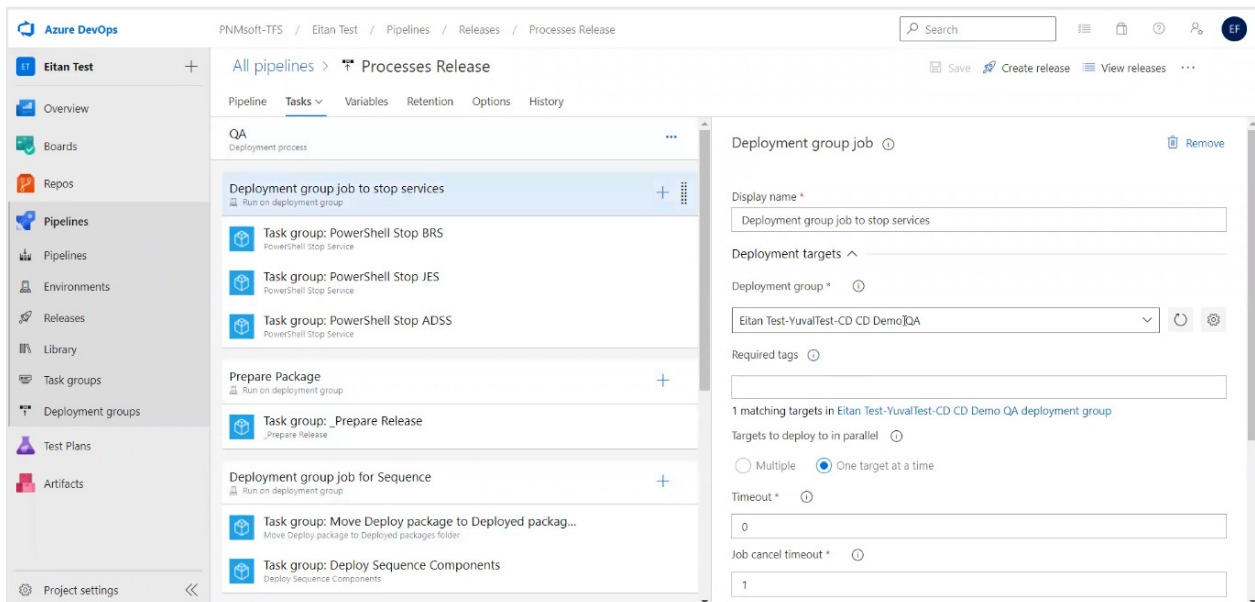


The screenshot shows the Azure DevOps interface for the 'Processes Release' pipeline. The 'Variables' tab is selected, displaying a list of pipeline variables. The variables are as follows:

Name	Value	Scope
AzureDevOpsIntegration.exe	C:\SequenceDeploymentClient\DevOpsIntegrationApp\AzureDevOpsIn...	Release
DeploymentFilesBasePath	C:\SequenceDeploymentClient	Release
DropFolder	..DropFolder\Deployment Packages\\$(Release.Number)	Release
ExternalResources	C:\inetpub\wwwroot\Cora SeSequence\Administration\Shared Resources	Release
MultiWorkflowsDeploymentApp.exe	C:\SequenceDeploymentClient\MultiWorkflowsDeployment\MultiWork...	Release
Release.Number	1.0	Release
ShouldCleanPackage	False	Release
ShouldStopStartServices	True	Release
WorkingFolder	C:\SequenceDeploymentClient\WorkingFolder	Release

These variables specify the artifact repos and the destination repos where the applications will be deployed, and additional properties needed for the CD process.

## Process Release (CD) pipeline tasks



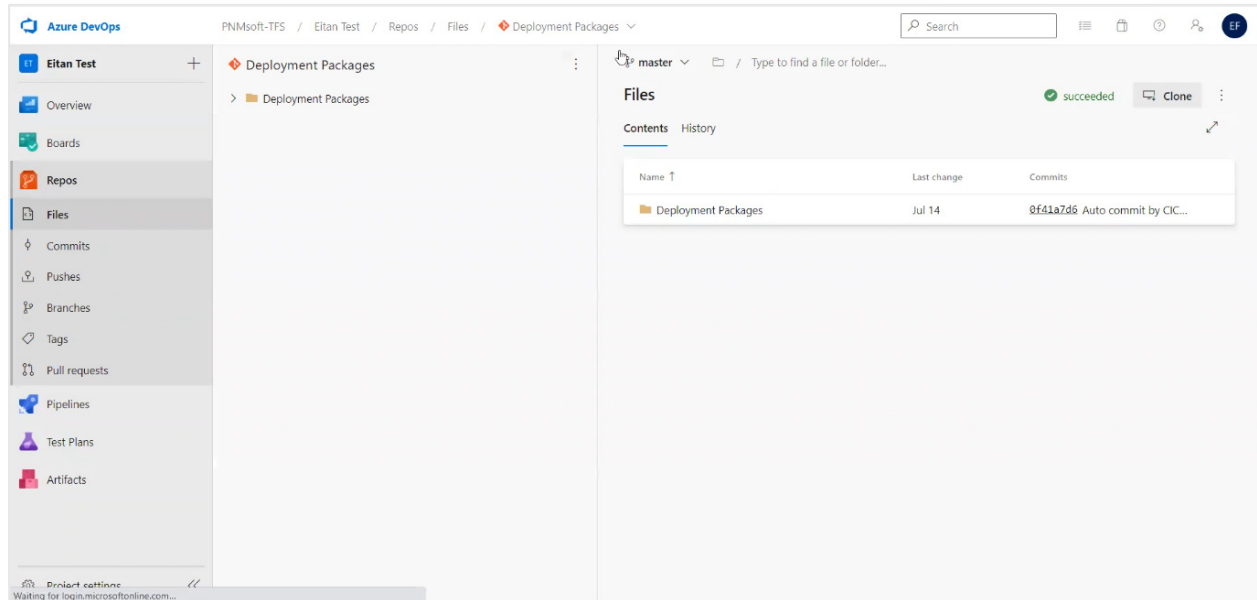
The screenshot shows the Azure DevOps interface for the 'Processes Release' pipeline, specifically the 'Tasks' tab. The pipeline is named 'QA' and is a 'Deployment process'. The tasks are organized into three main sections:

- Deployment group job to stop services** (Run on deployment group):
  - Task group: PowerShell Stop BRS (PowerShell Stop Service)
  - Task group: PowerShell Stop JES (PowerShell Stop Service)
  - Task group: PowerShell Stop ADSS (PowerShell Stop Service)
- Prepare Package** (Run on deployment group):
  - Task group: \_Prepare Release (\_Prepare Release)
- Deployment group job for Sequence** (Run on deployment group):
  - Task group: Move Deploy package to Deployed package... (Move Deploy package to Deployed packages folder)
  - Task group: Deploy Sequence Components (Deploy Sequence Components)

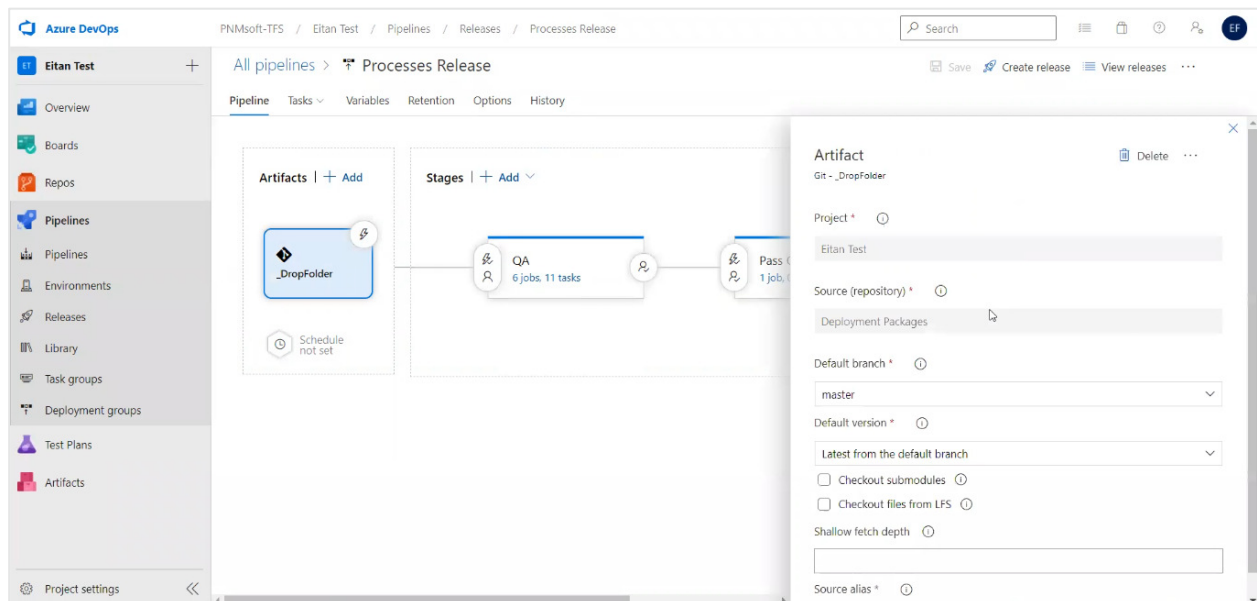
The right-hand pane shows the configuration for the 'Deployment group job'.

- Display name \***: Deployment group job to stop services
- Deployment targets ^**:
  - Deployment group \***: Eitan Test-YuvalTest-CD Demo QA
  - Required tags**: (Empty)
  - 1 matching targets in Eitan Test-YuvalTest-CD Demo QA deployment group**
  - Targets to deploy to in parallel**:
    - ☐ Multiple
    - ☒ One target at a time
  - Timeout \***: 0
  - Job cancel timeout \***: 1

This is the repo of the Azure DevOps where the CI pipeline stores the project after processing. That is, the Deployment packages folder.



The Artifact source points to the repo configured in the previous screenshot. That is, the Deployment Packages folder.

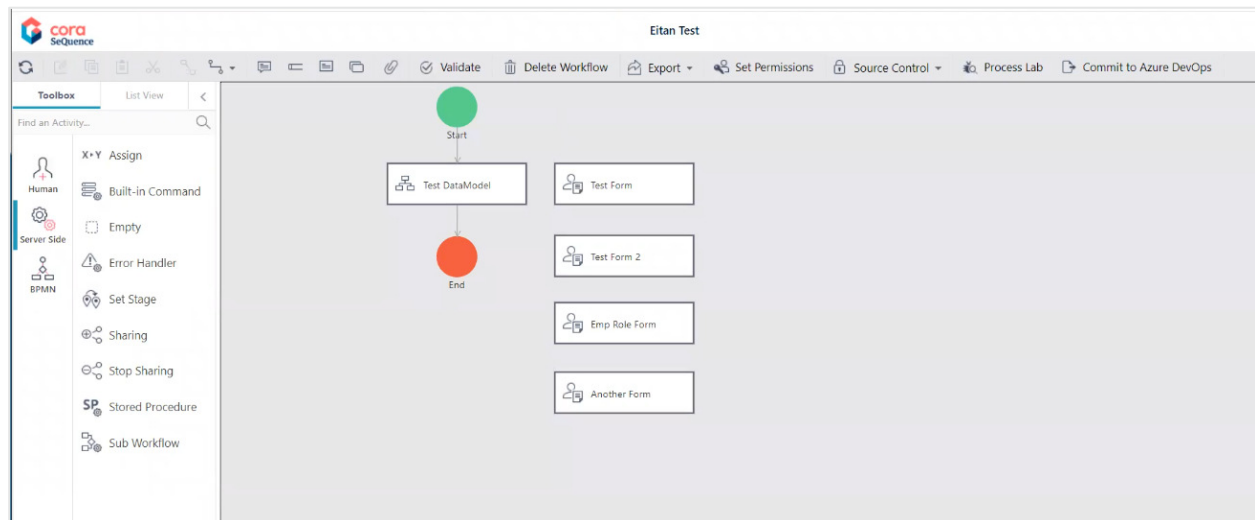




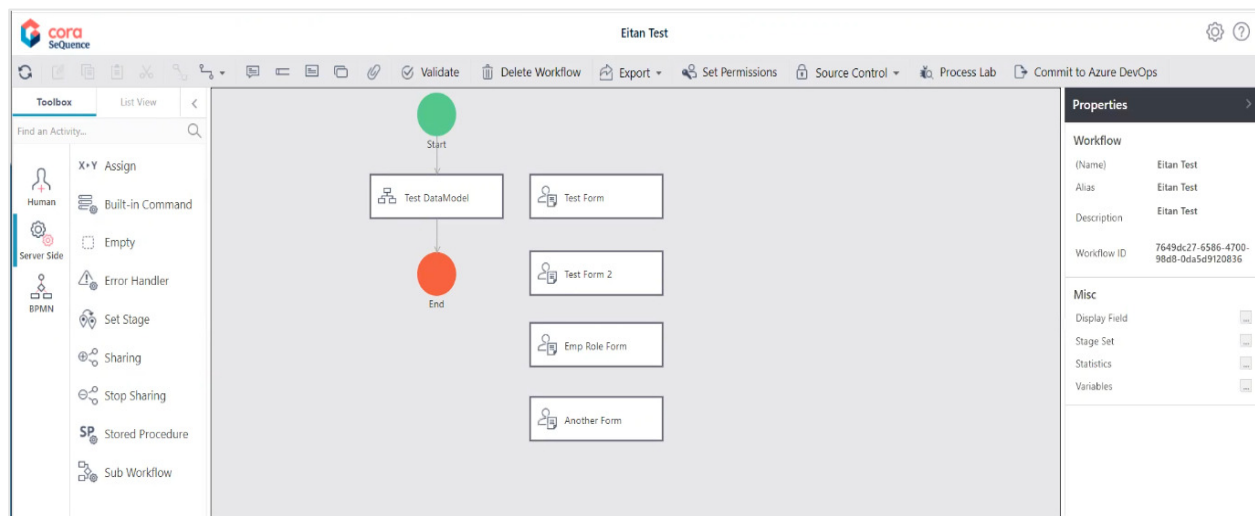
## Executing the pipelines

In this example, we're going to change the Development server, CDEVICID1, and then propagate the change to the Test server, CDEVICID2.

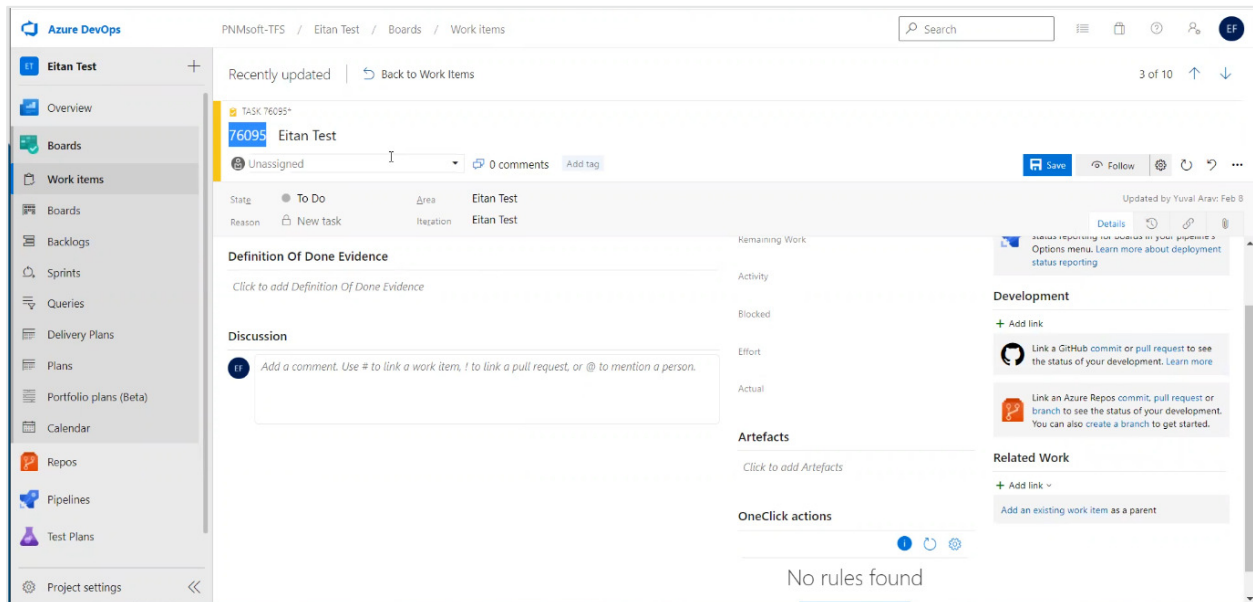
### Development server CDEVICID1



### Test server CDEVICID2



You need to connect the change to a work item on Azure DevOps. The work item includes detailed instructions about the change and it can also include related documentation. In this example, we use the work item 76095.

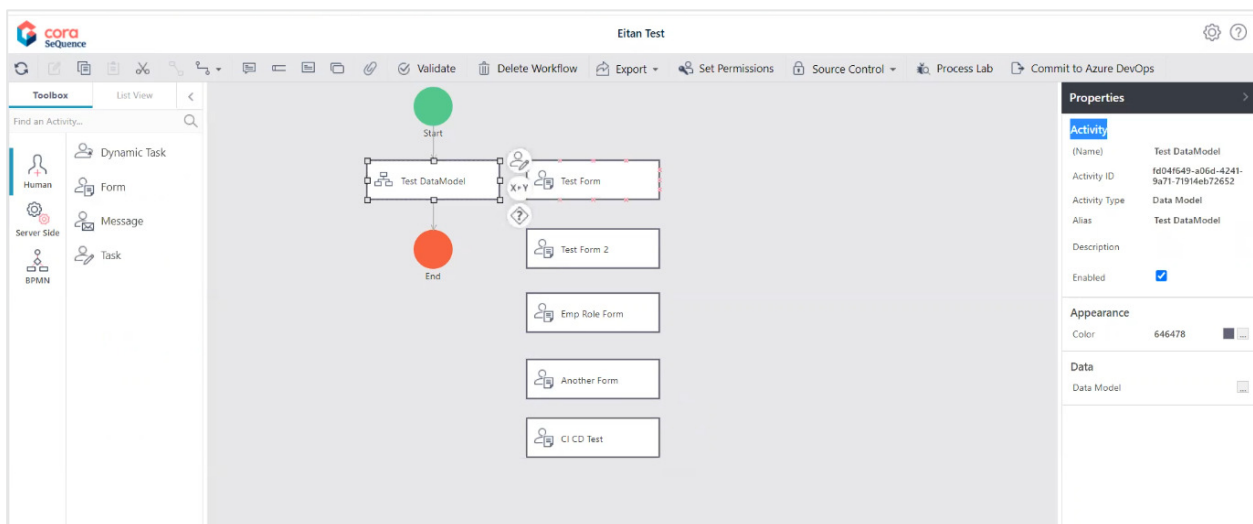


## Note

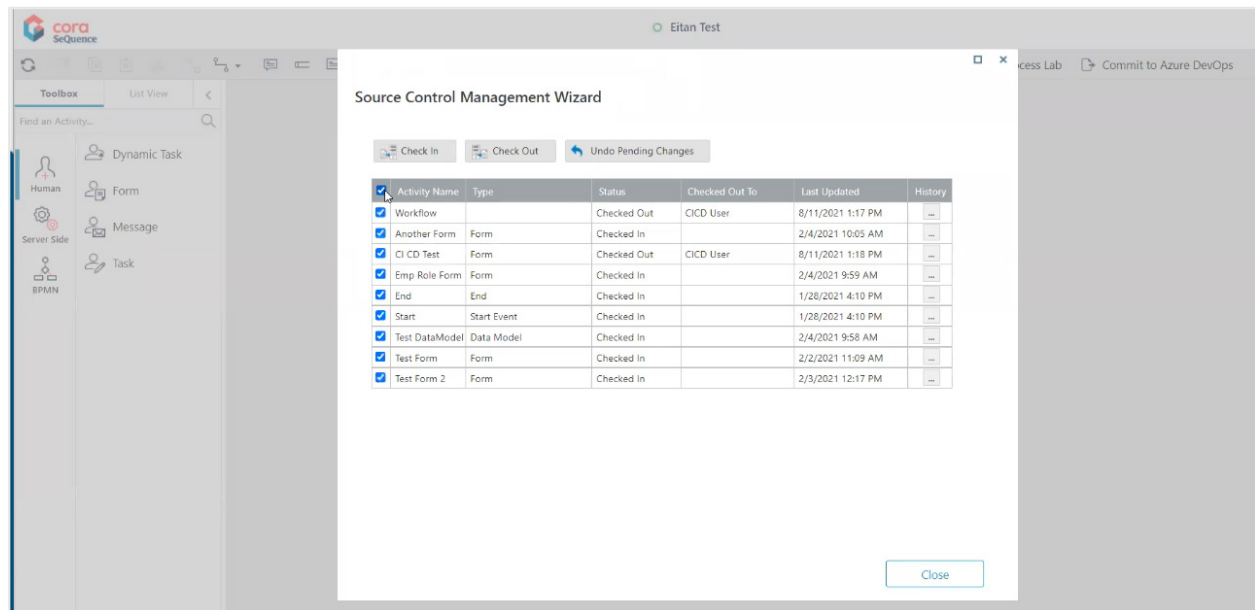
If a work item doesn't exist, you can create it in Azure DevOps.

## Commit to Azure DevOps

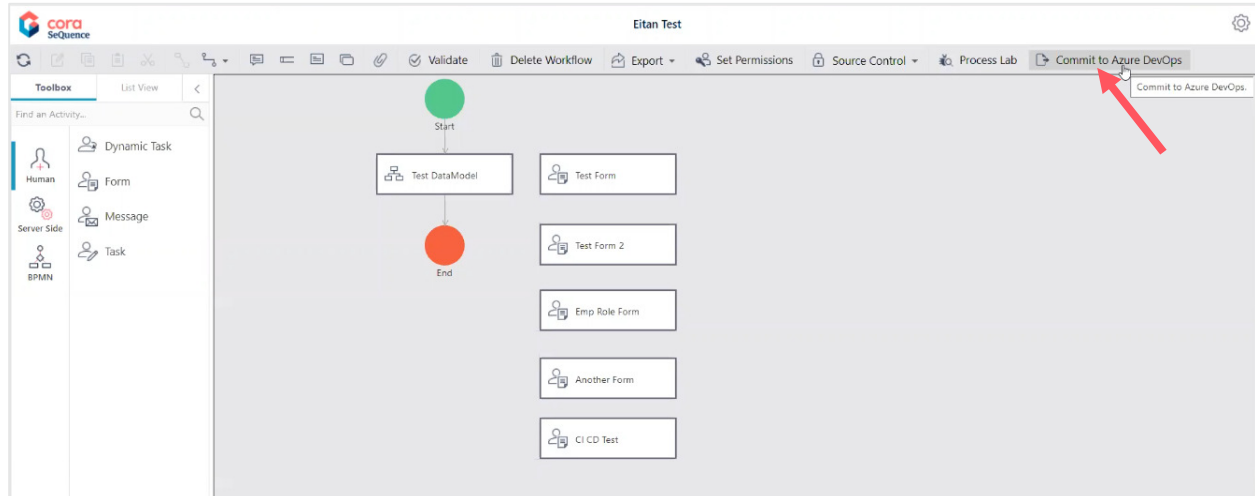
In the following example, we've added a Form activity called **CI/CD Test** to the workflow Eitan Test in the Development server.



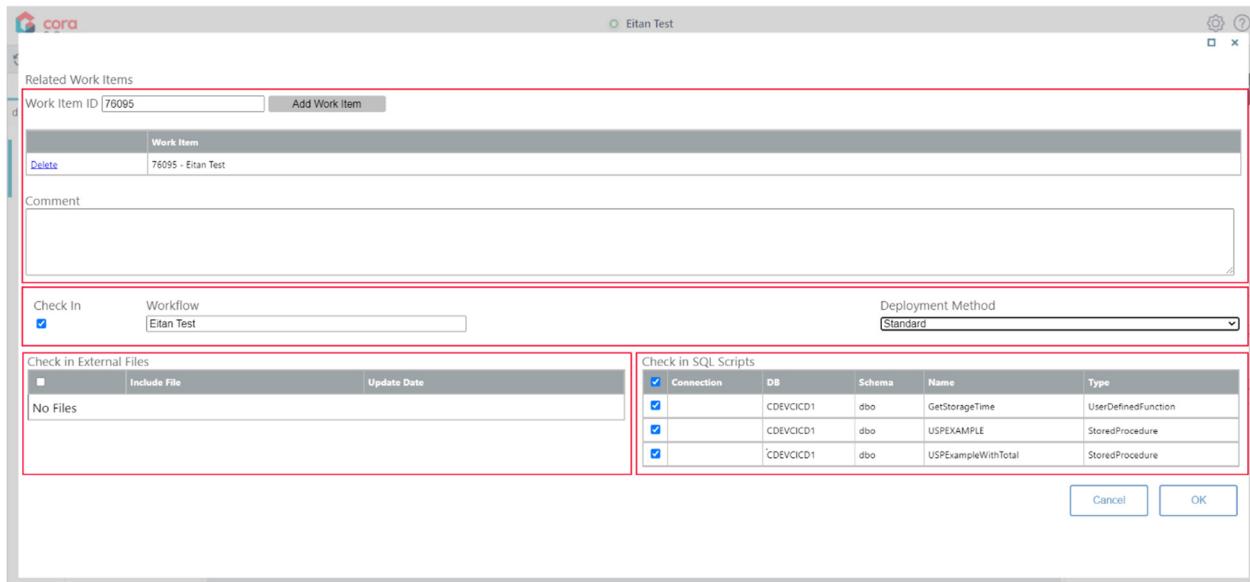
After adding the new form, check in the workflow.



After check-in, commit the workflow to Azure DevOps. Click the **Commit to Azure DevOps** button.



After you click **Commit to Azure DevOps**, the following screen displays:



It includes the following main configuration areas:

## 1. Work Item

- Click **Add Work Item**.

We need to connect every change to a work item in Azure DevOps.

You can link multiple changes to the same work item.

In this example, the work item ID is 76095.

## 2. Workflow

- Select the **Check In** checkbox.
- Verify that the displayed workflow name is the correct one.  
In this example, it is **Eitan Test**.
- Select the relevant deployment method.  
For more information about deployment methods, see Deployment methods.

## 3. External files

We need to indicate all the files that are relevant to the change that we made.

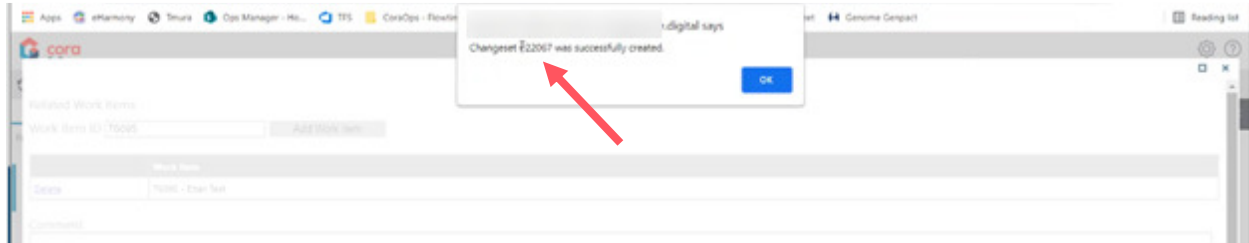
- Under Check In External Files, select the relevant files, if required.  
Our example doesn't include external files.

## 4. SQL scripts

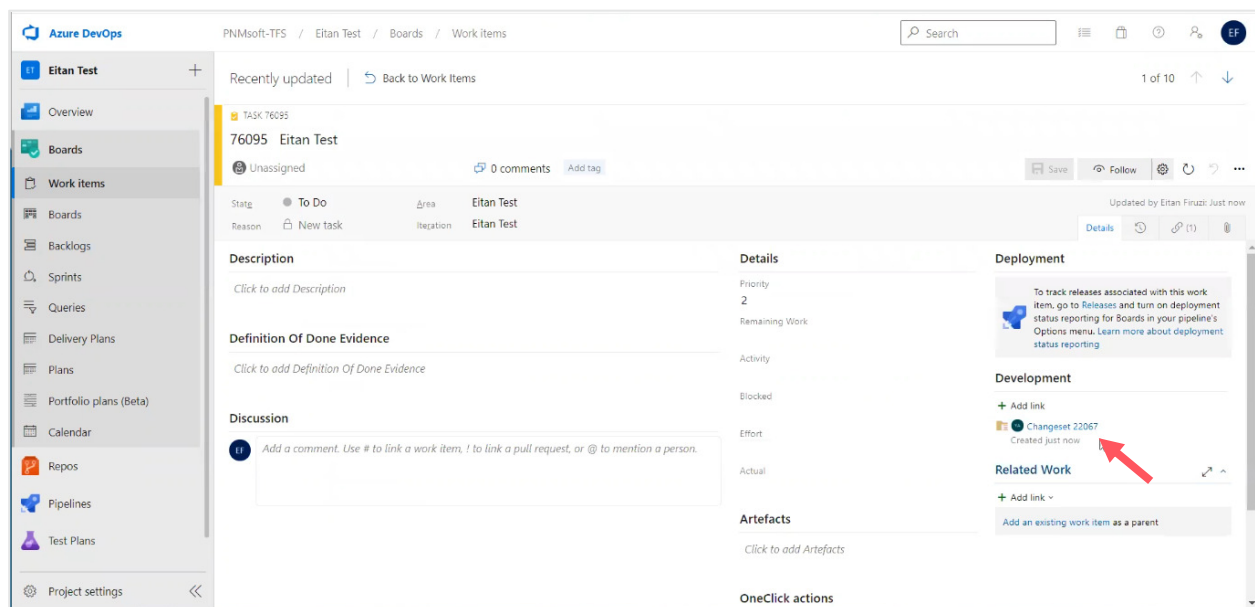
We need to indicate all the *database objects* that are relevant to the change that we made.

- Under Check In SQL Scripts, select the relevant database objects.  
In our example, all the stored procedures are required.

After the commit is successful, we get a changeset number.



You can also view the changeset in Azure DevOps under the work item number 76095.



## Deployment methods

Related only to the workflow that we are deploying and not to files and database objects.

Method	Description
<b>Standard</b>	<p>Deploys the workflow, including all its activities, permissions, and Cora SeSequence standard tables within the workflows.</p> <p>This method doesn't deploy stored procedures, views, reference data, and all the end points definition within the workflow (system objects elements).</p>

Method	Description
<b>End Point</b>	<p>Deploys the workflows and its end points. For example, the web consumer's definition and URLs, without the referenced data within the workflow.</p> <p>Doesn't deploy the permissions.</p>
<b>Data Incremental</b>	<p>Deploys data within the workflow. For example, lookup tables, in an incremental method.</p> <p>Doesn't deploy the permissions.</p>
<b>Data Overwrite</b>	<p>Deploys data within the workflow. For example, lookup tables, in an overwrite method – overwrite the exiting data in the target environment.</p> <p>Doesn't deploy the permissions.</p>

## Best practices

### Dos

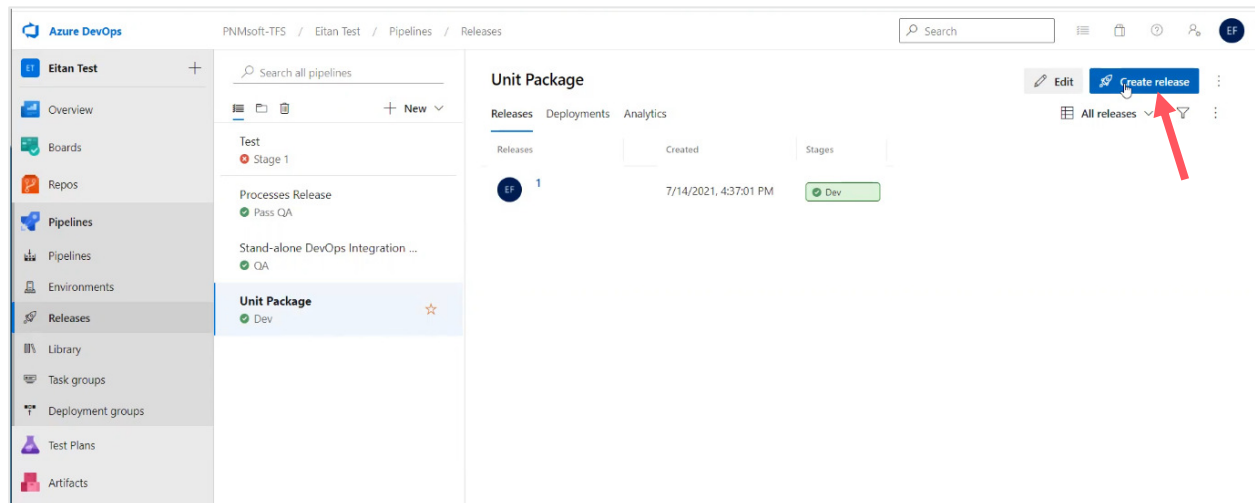
- Have a separate workflow that holds only the end points.  
Deploy the workflow using the End Point deployment method.
- Have a separate workflow with the data model that holds only the data objects.  
For example, tables and stored procedures.  
Deploy the workflow using the relevant data deployment method (Incremental or Overwrite).

### Don'ts

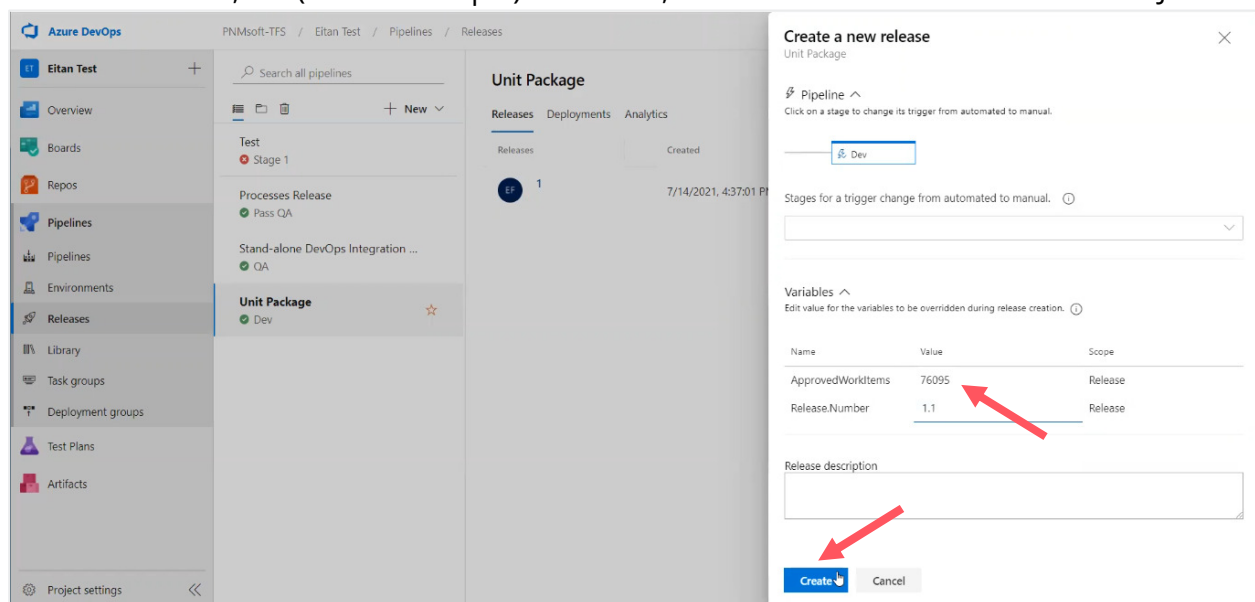
- Don't deploy the same workflow with several deployment methods to create End Points/Data.  
If required, create a separate workflow for deploying end points and another one, for deploying data.

## Execute the Unit Package pipeline

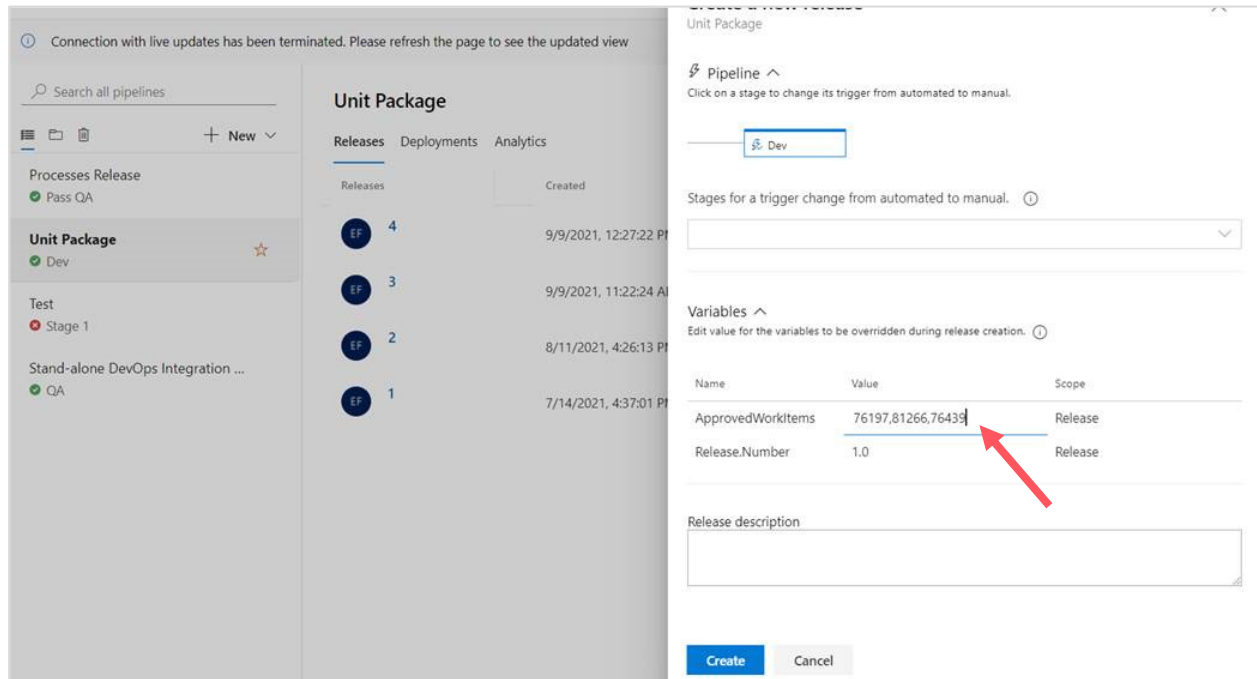
After the commit has been made, we need to execute the CI pipeline Unit Package in Azure DevOps by clicking on the **Create Release** button.



We add the work item number in the **ApprovedWorkItem** text box, 76095, and the Release Number, 1.1 (in this example). And then, click the **Create** button to start the Job.



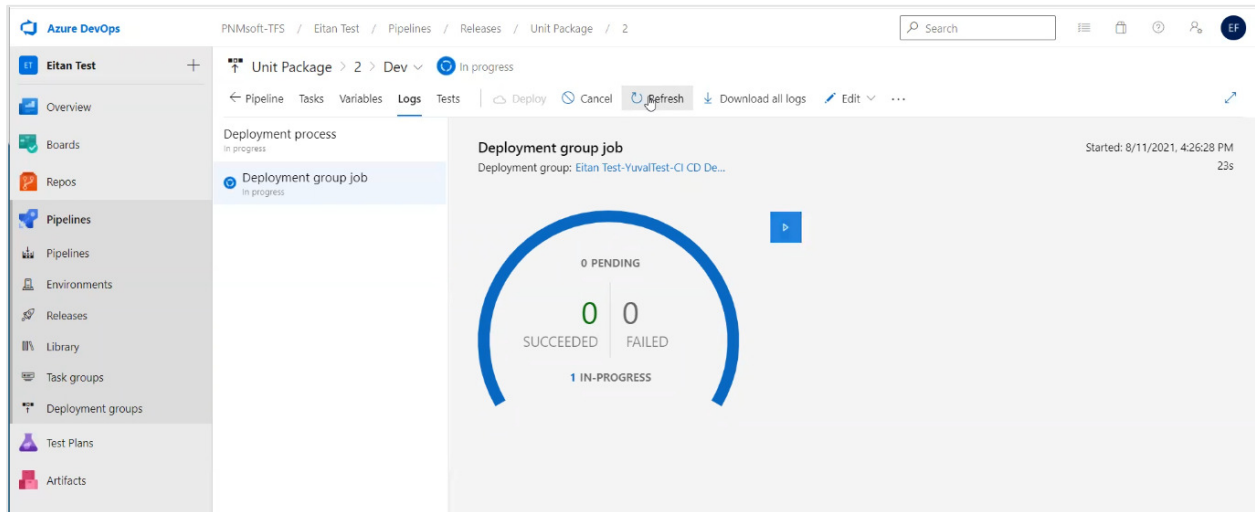
In case we want to deploy multiple work items in one go (that is, multiple changesets) in the Unit Package, we can add multiple work items by separating them with a comma without any space (see screenshot below).



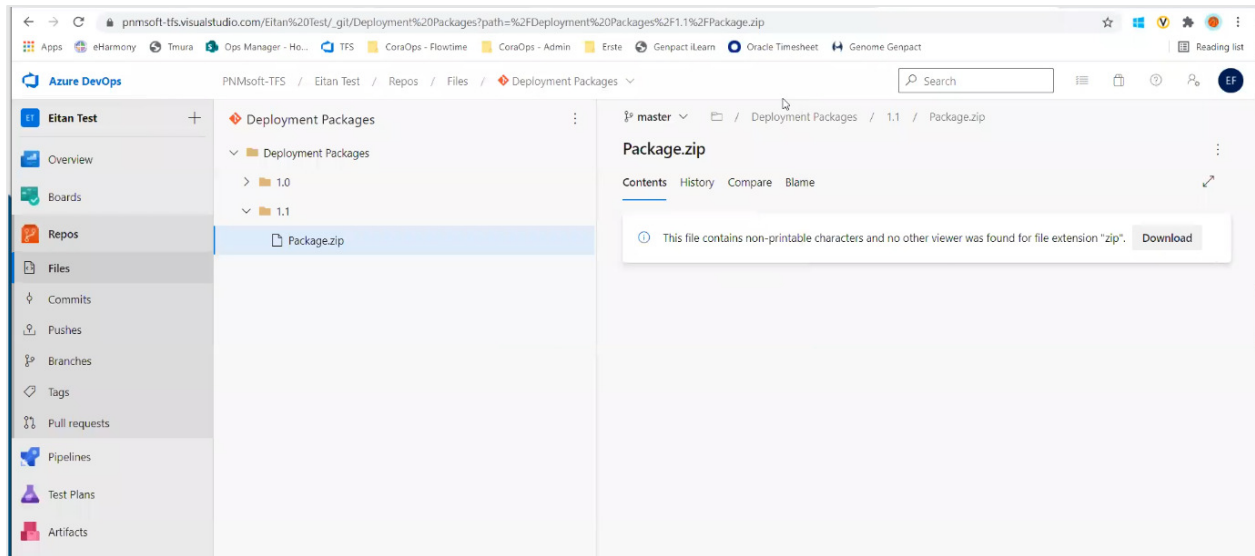
Deploying several changesets saves time and provides the option to package a few workflows from different work items to the same package.



After the CI pipeline job has started, we can track its progress on the following screen:

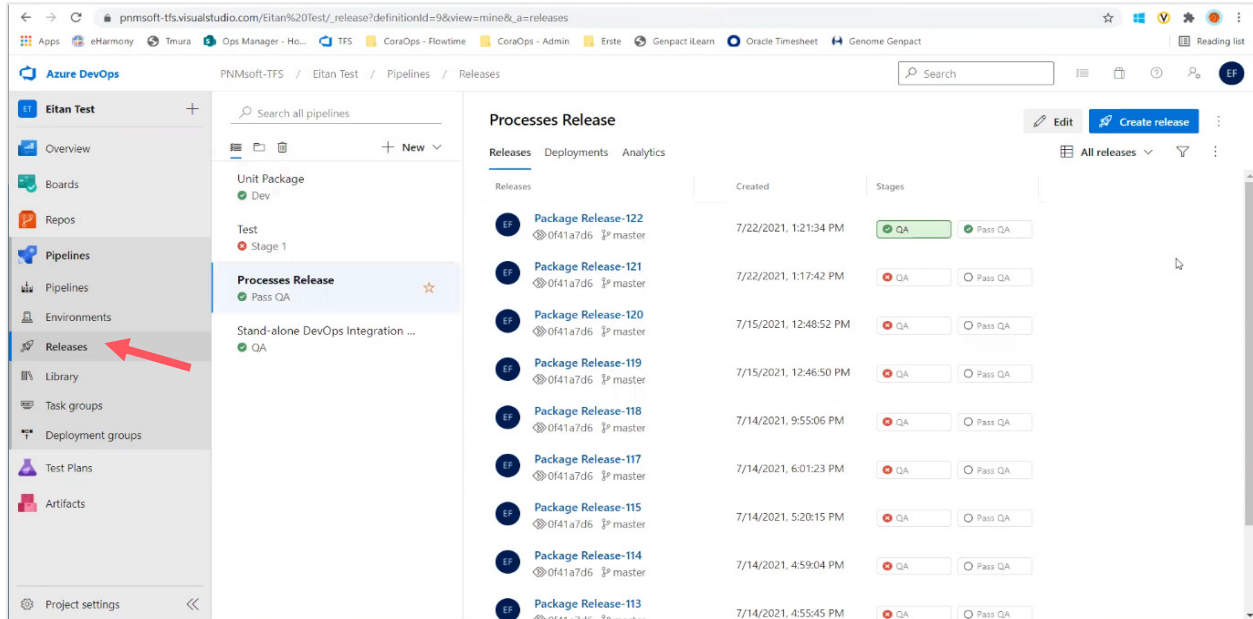


After the CI pipeline finishes running, the **Package.zip** is created as an artifact in the deployment package folder.



## Execute the Processes Release pipeline

After the artifact Package.zip has been created, we can now execute the CD pipeline (Processes Release pipeline).

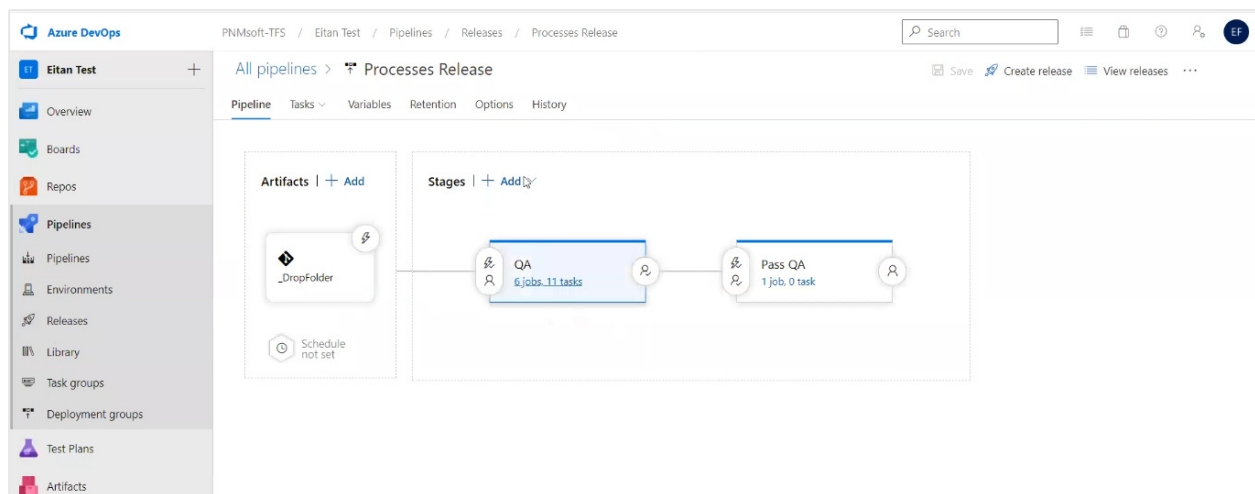


The screenshot shows the Azure DevOps portal interface. In the left-hand navigation pane, the 'Releases' option is highlighted with a red arrow. The main content area displays the 'Processes Release' pipeline, showing a list of releases with columns for 'Releases', 'Created', and 'Stages'. The releases are listed in descending order of creation time, starting from 7/22/2021, 1:21:34 PM.

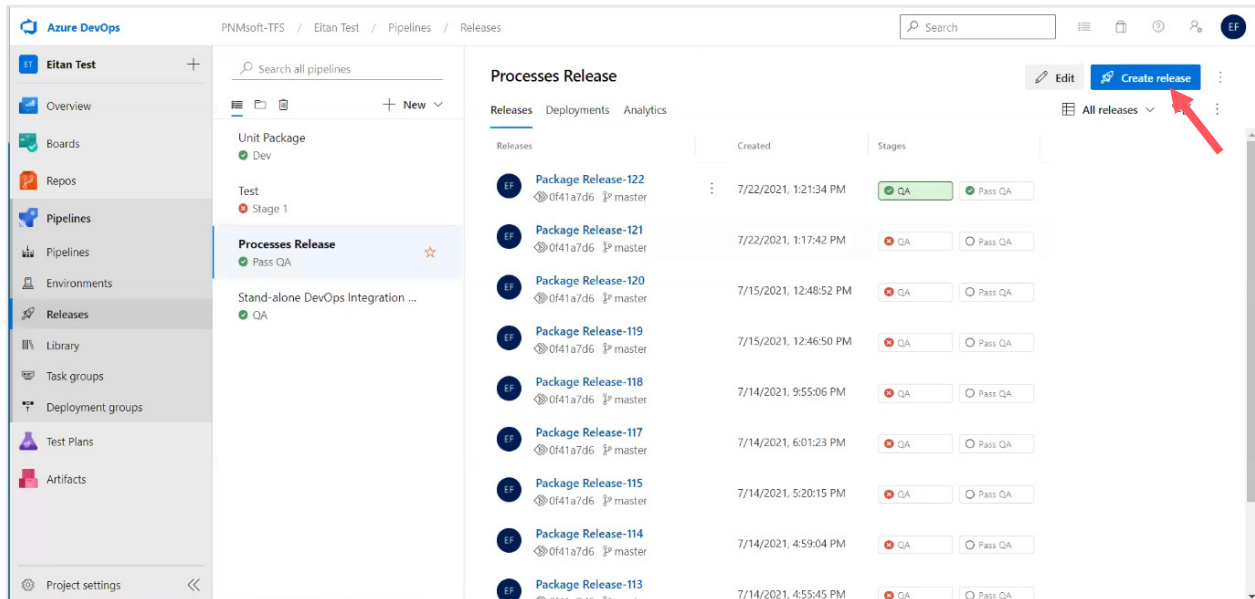
Release Name	Created	Stages
Package Release-122	7/22/2021, 1:21:34 PM	QA (Pass)
Package Release-121	7/22/2021, 1:17:42 PM	QA (Fail)
Package Release-120	7/15/2021, 12:48:52 PM	QA (Fail)
Package Release-119	7/15/2021, 12:46:50 PM	QA (Fail)
Package Release-118	7/14/2021, 9:55:06 PM	QA (Fail)
Package Release-117	7/14/2021, 6:01:23 PM	QA (Fail)
Package Release-115	7/14/2021, 5:20:15 PM	QA (Fail)
Package Release-114	7/14/2021, 4:59:04 PM	QA (Fail)
Package Release-113	7/14/2021, 4:55:45 PM	QA (Fail)

To view a list of the pipelines, in the Azure DevOps portal, in the left-panel menu, click **Releases**, and then select **Processes Release**.

Graphical view of the Processes Release CD pipeline

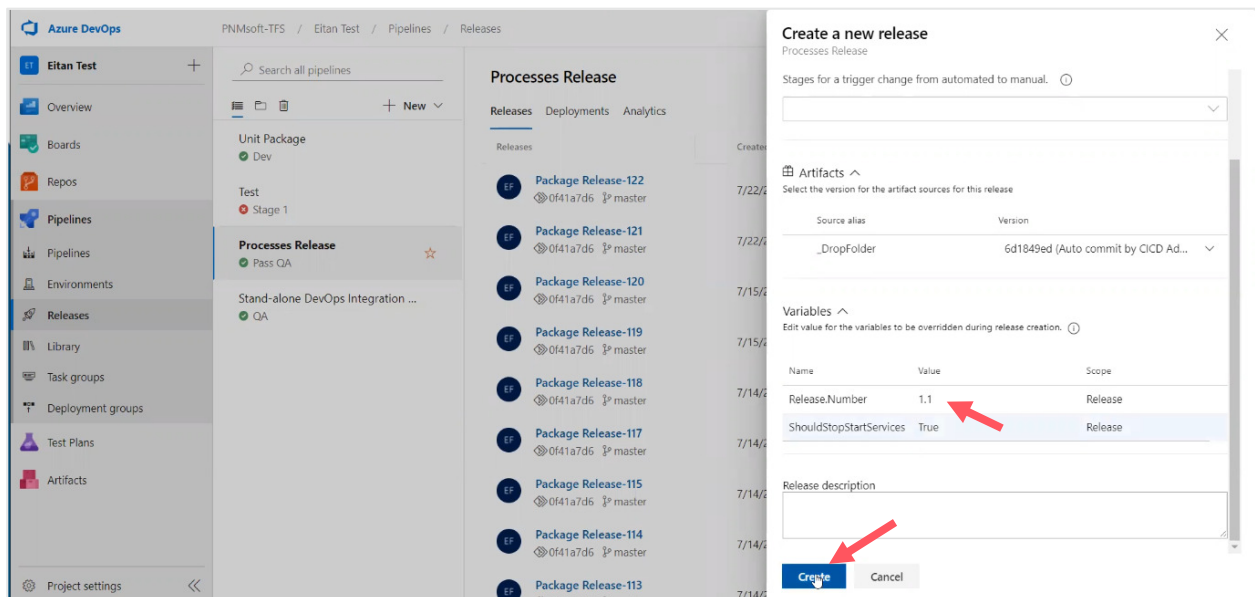


To execute the pipeline, click the **Create release** button.

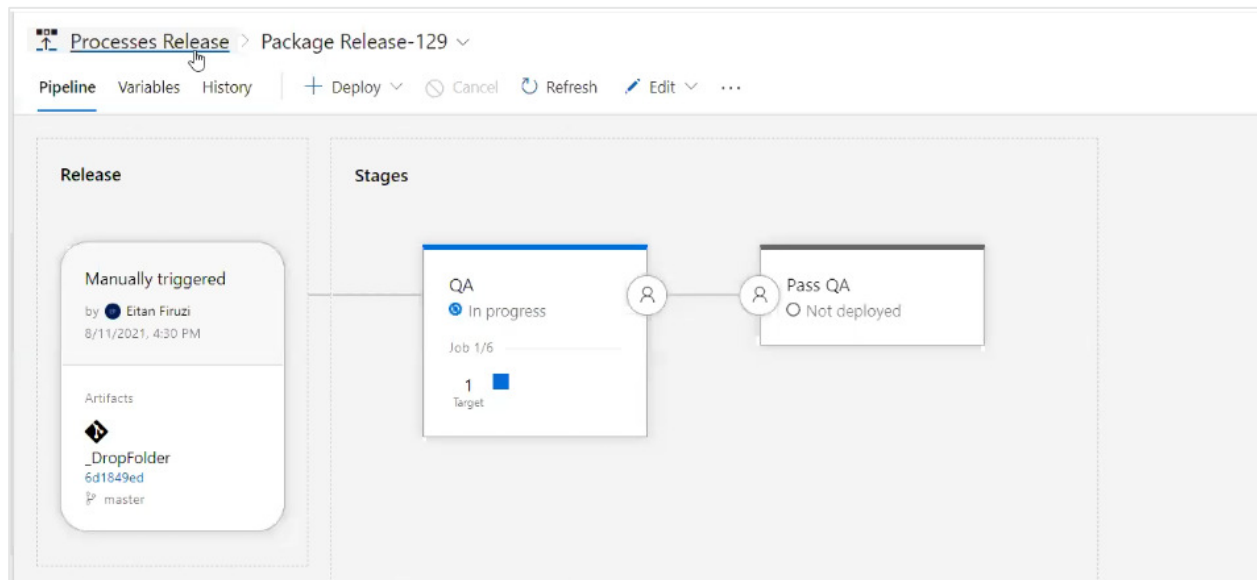


Enter values for the **ReleaseNumber** (give a version number to the release) and **ShouldStopStartServices** fields (specify if IIS should be stopped and restarted after the packages are deployed).

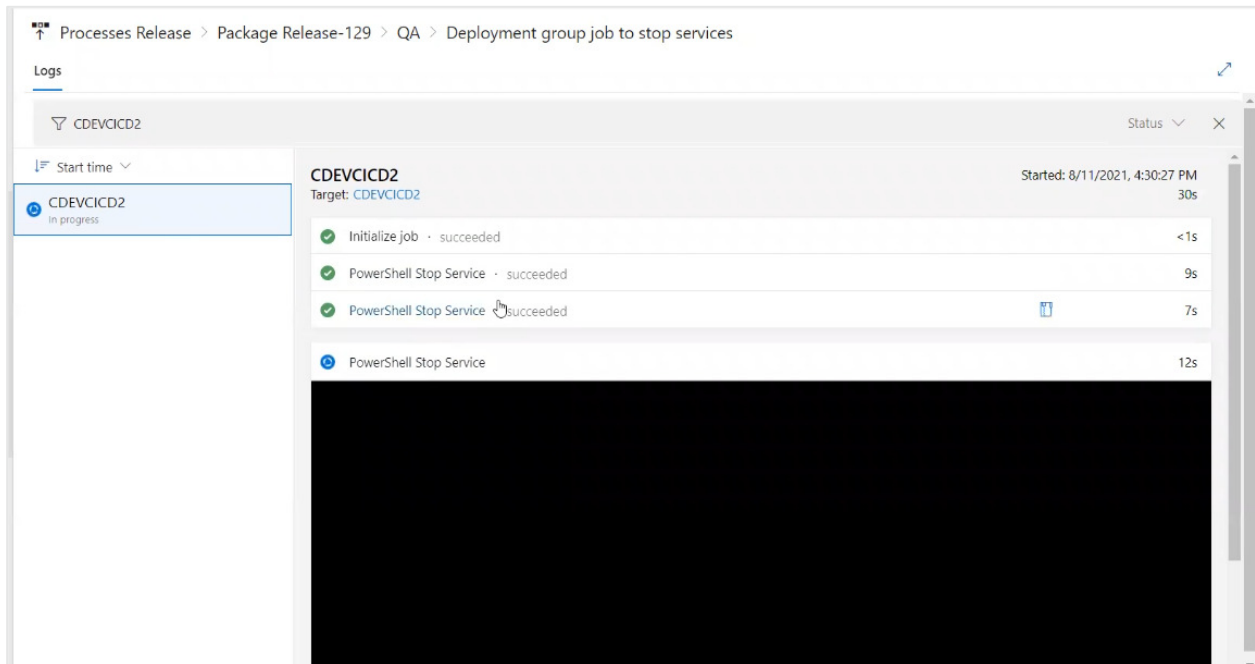
To begin execution, click **Create**.



You can track the progress of the execution (see below).



You can also follow the process in the logs view.

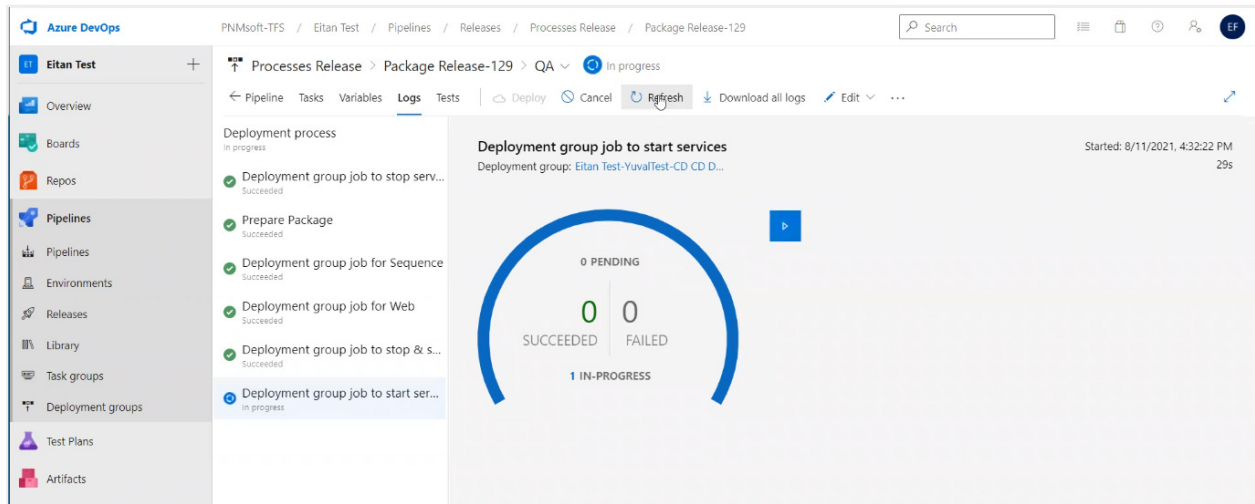


The screenshot shows the 'Logs' view for the 'QA' stage, specifically for the 'Deployment group job to stop services'. The logs are filtered by 'CDEVICID2'. The log entries are as follows:

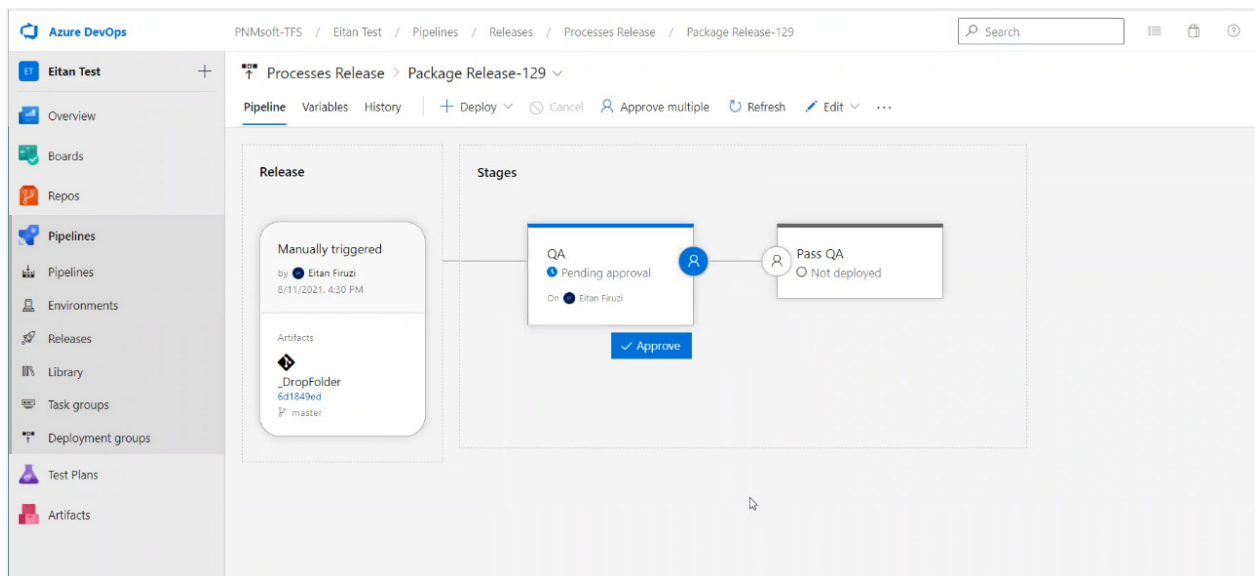
Log Entry	Duration
Initialize job · succeeded	<1s
PowerShell Stop Service · succeeded	9s
PowerShell Stop Service · succeeded	7s
PowerShell Stop Service	12s

The log for 'PowerShell Stop Service' is currently expanded, showing a black screen, likely representing a video recording of the command execution.

In the above screenshot, the process is in the last stage.

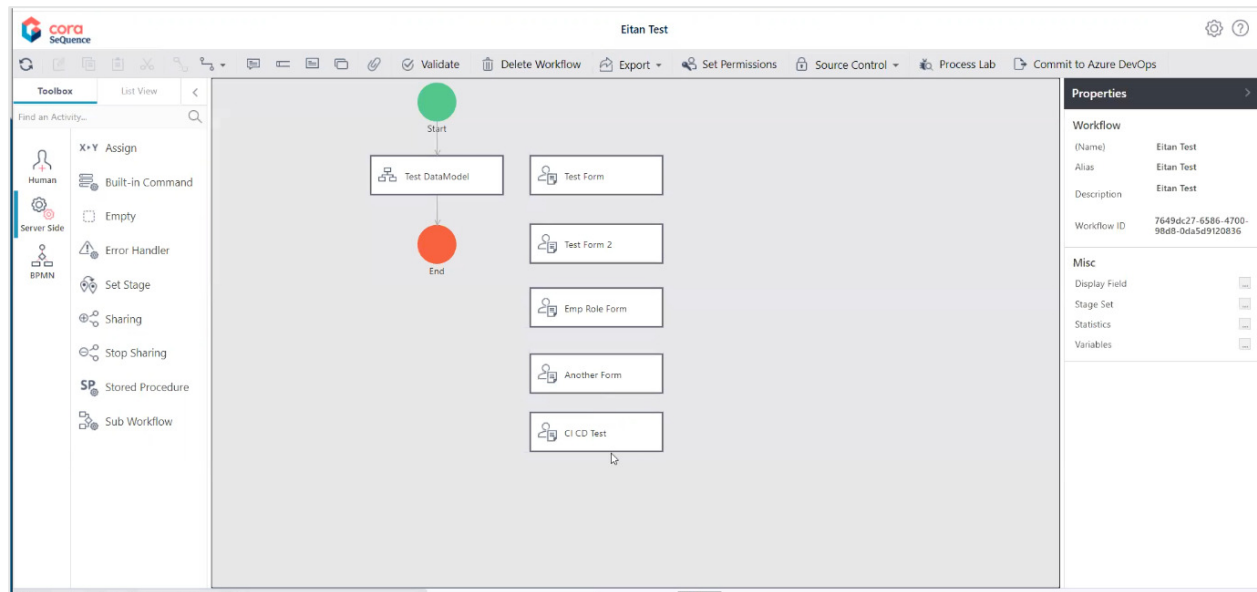


The Processes release pipeline has two stages: QA and Pass QA.



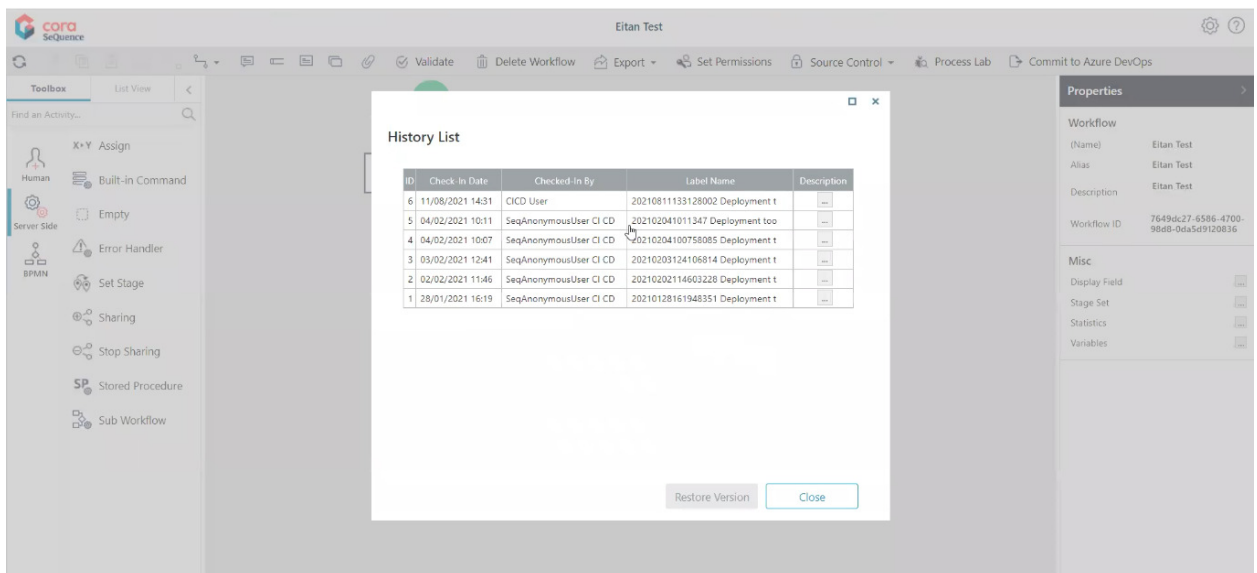
In the QA stage, the administrator needs to check and confirm if the deployment was done correctly and then select **Approve**.

Let's check the workflow in the Test server to see if the deployment has been successful.



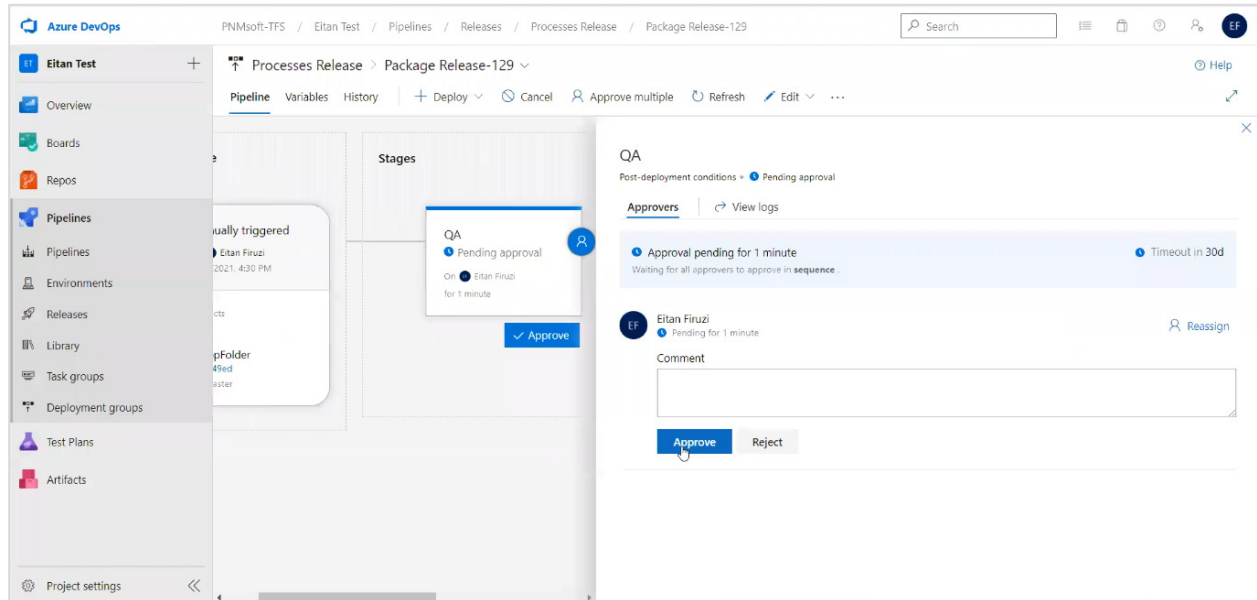
The deployment was successful. The CI/CD form has been added to the workflow.

If we check the Source control history, we can see that the last user who checked in the workflow was the CI/CD user.



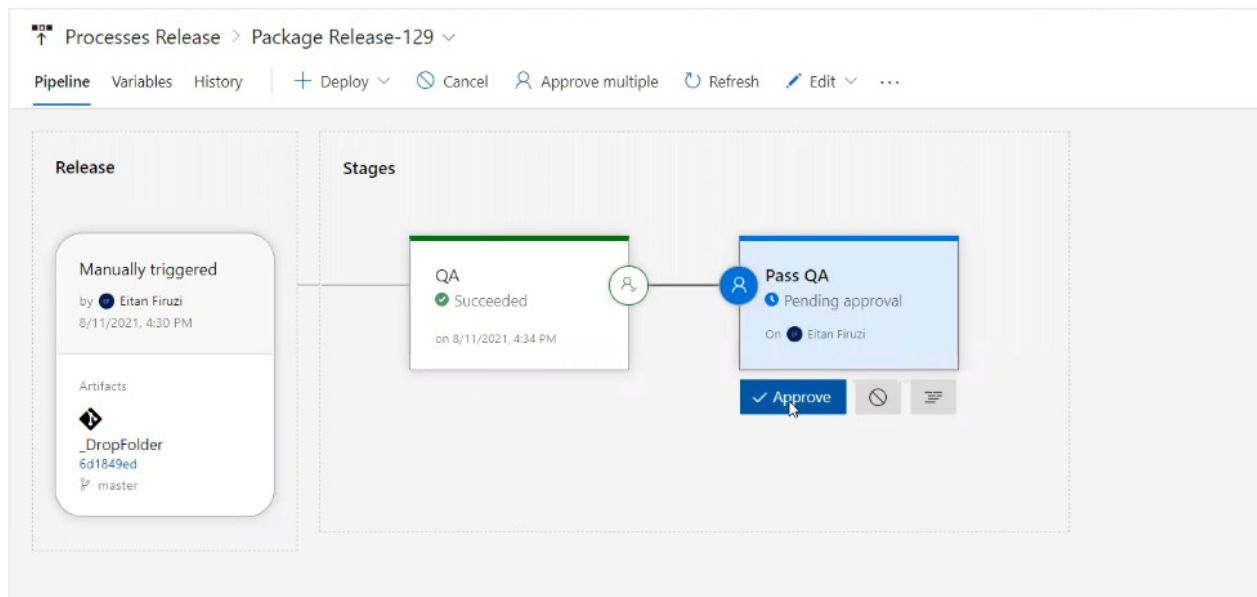
ID	Check-In Date	Checked-In By	Label Name	Description
6	11/08/2021 14:31	CI/CD User	20210811133128002 Deployment t	
5	04/02/2021 10:11	SeqAnonymousUser CI CD	202102041011347 Deployment too	
4	04/02/2021 10:07	SeqAnonymousUser CI CD	20210204100758085 Deployment t	
3	03/02/2021 12:41	SeqAnonymousUser CI CD	20210203124106814 Deployment t	
2	02/02/2021 11:46	SeqAnonymousUser CI CD	20210202114603228 Deployment t	
1	28/01/2021 16:19	SeqAnonymousUser CI CD	20210128161948351 Deployment t	

The first approval is done by the user who executes the pipeline. This user needs to verify that the implementation completed successfully, based on its purpose.



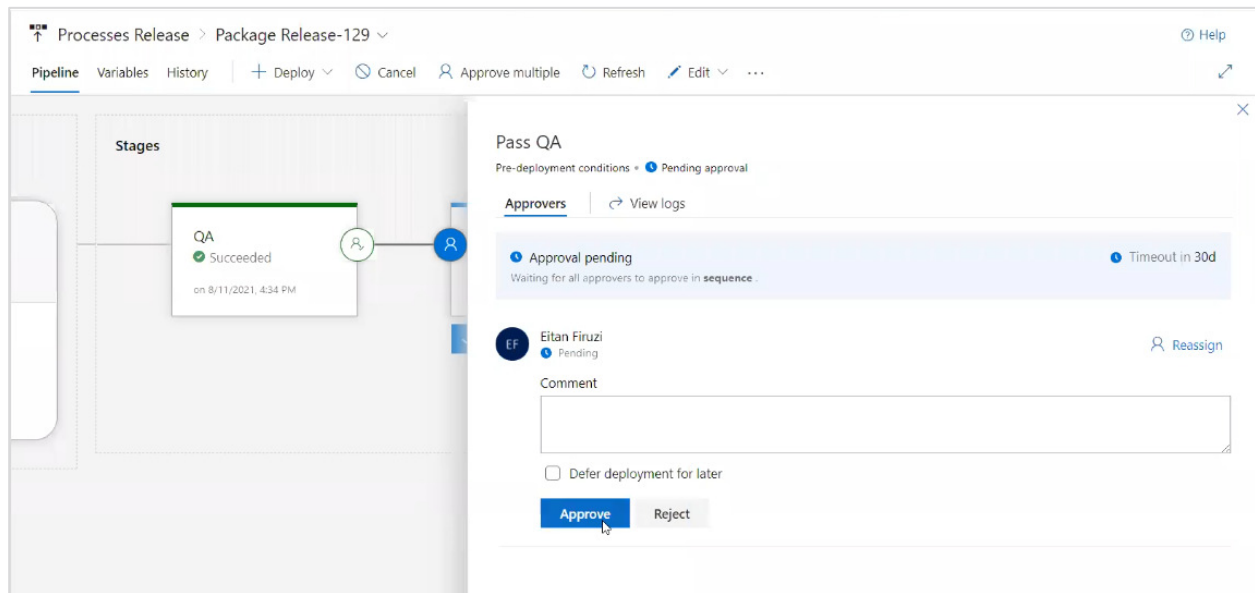
After approval, the pipeline moves to the Pass QA stage.

After this stage is completed, the QA person needs to approve the changes to complete the process. This approval cycle is not mandatory, but it's recommended. The QA user should make sure that the workflows have been deployed and that they are working properly.

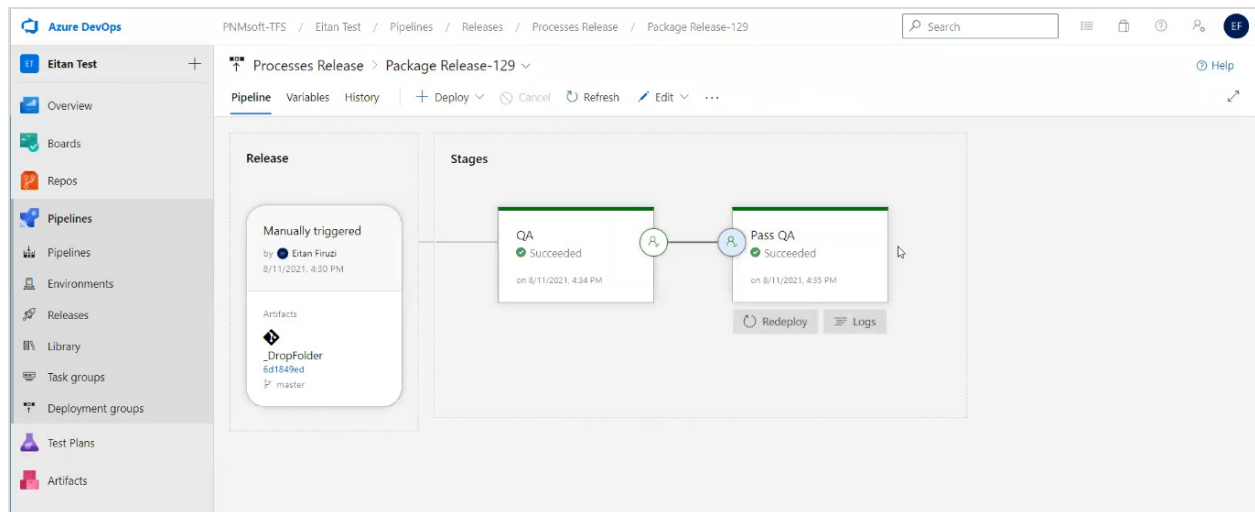




Pass QA stage approval screen.



After Pass QA is approved, the CD process is completed.



After QA is approved, the deployment continues to the next environment, usually Production, but it depends on the specific project.

### Important

When deploying to Production, or Production-like environments, make sure that the package that you deploy is approved and final.



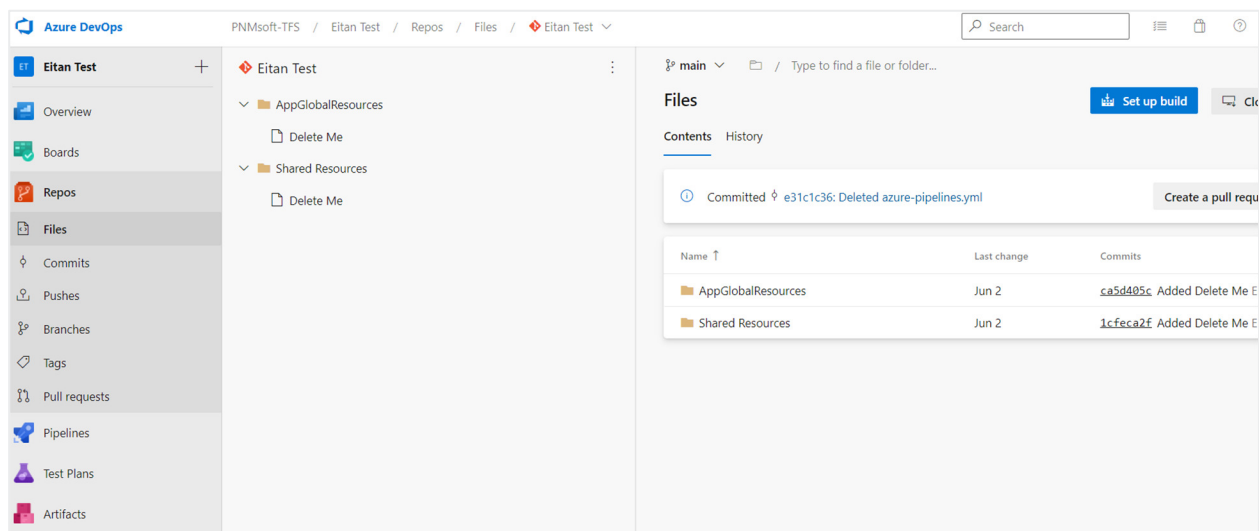
## Customize Cora SeSequence

The pipeline available for Cora SeSequence customizations is the **Release Shared Resources** pipeline.

The Release Shared Resources pipeline reads files from a specified repo and propagates them in the required Cora SeSequence folders of the target server.

We need to create a repo to be used by the customization pipeline:

- Repo type: GIT
- Repo name: Same name as the Azure DevOps project name.



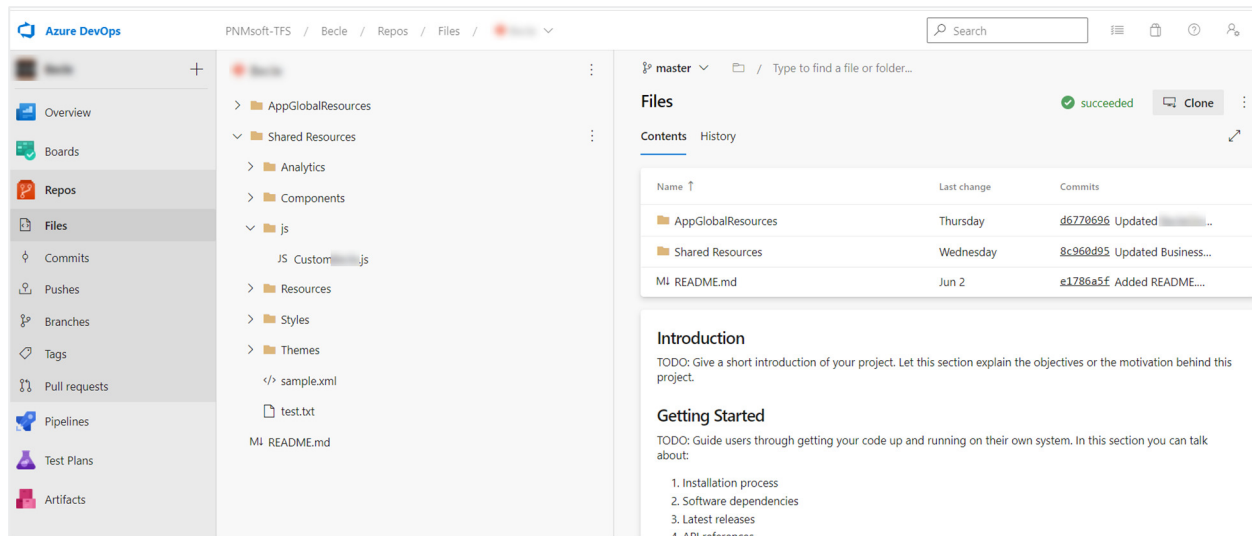
### Note

It is recommended to create a separate repo for each customer.  
The naming convention is **<CustomerName+CI CD>**.

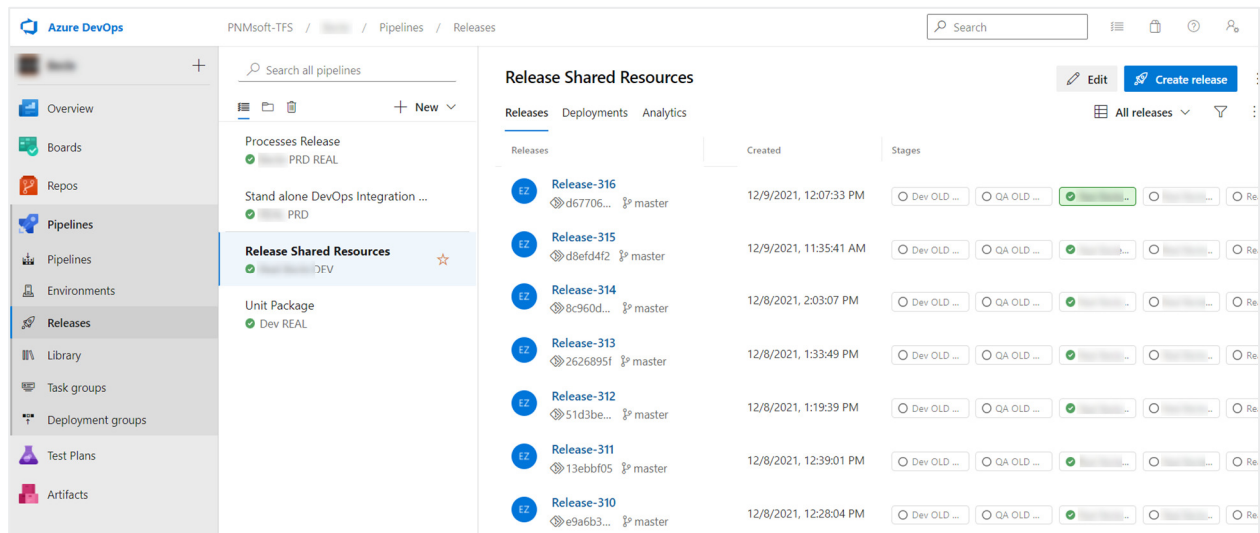
The developers upload their files directly into the repos that have been created for them in the appropriate folder.

For example, if developers have made changes to javascript files that are used by forms, all they need to do is to upload the updated files to the appropriate folder in the repo. In our example, it's the folder **shared resources\js**. See screenshot below.

Example of repo (the customer name has been hidden).



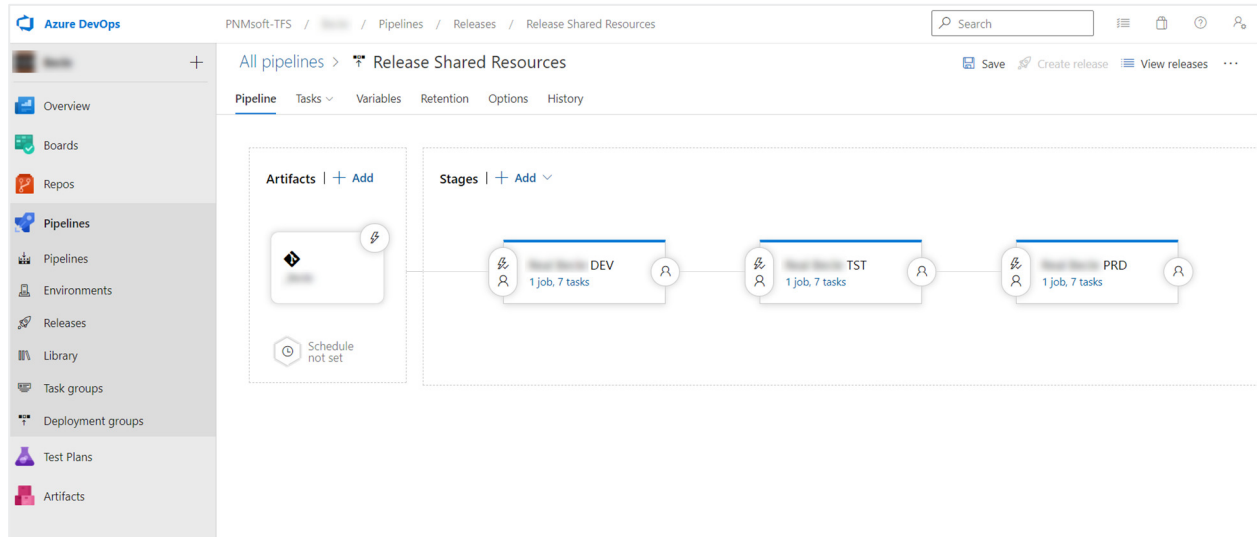
The **Release Shared Resources** pipeline is used for customizations. It's the main pipeline used for customer deployments.



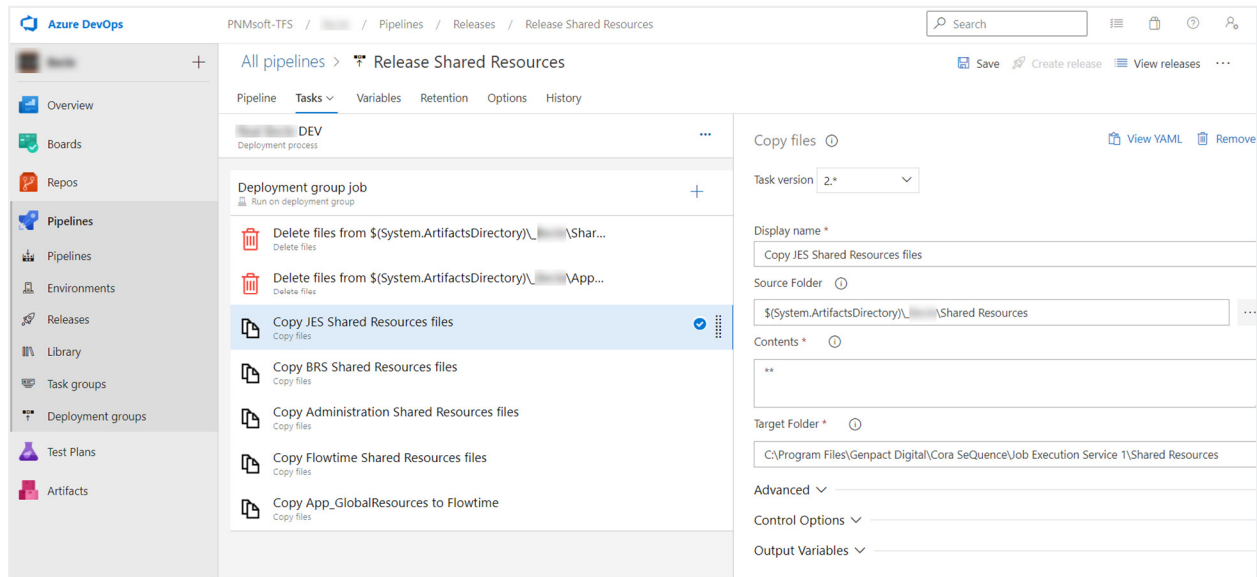
## Important

The DevOps platform includes additional pipelines, which are used for the initial Cora SeSequence installation. These pipelines are not covered as they are used only by CI/CD System Admins.

The Release Shared Resources pipeline takes files from the repo (artifacts) and updates the Dev and QA environments.



Most of the tasks in the Release Shared Resources pipeline are to **delete** files and **copy** files, which replace them with the ones from the repo.



The pipeline executes each task in the list in the specified order.

The successful completion of the pipeline indicates that the customization has been deployed to all the configured environments (commonly in this order DEV > TEST > PROD).

