

***Azure DevOps Developers Handbook***

**<Version No. 1.1>**

|  |  |  |  |
| --- | --- | --- | --- |
| Document Revision History: | | | |
| Document Version Number | Document Revision Date | Written By | Change Summary  (Reference section[s] changed) |
| V1.0 | 6/24/2020 | Durga Navaneethan |  |
| V1.1 | 7/14/2020 | Durga Navaneethan | Updated document based on review comments |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Contents

[**Scope and Purpose –** 2](#_Toc45633087)

[Scope 2](#_Toc45633088)

[Purpose 3](#_Toc45633089)

[**Pre-requisites** 3](#_Toc45633090)

[**Process Flow -** 5](#_Toc45633091)

[Stage 1: Clone feature Branch 5](#_Toc45633092)

[Stage 2: Build pipeline 8](#_Toc45633093)

[Stage 3 : Release Pipeline 15](#_Toc45633094)

[Stage 4: Create PR to merge code to develop branch 22](#_Toc45633095)

[Stage 5: QA deploy 25](#_Toc45633096)

[Stage 6: Create PR to merge code to Release branch 27](#_Toc45633097)

[Stage 7: Prod Deployment 27](#_Toc45633098)

[Stage 8: Merge code to Master 28](#_Toc45633099)

[Stage 9: Cherry-Picking 28](#_Toc45633100)

[Stage 10: Roll-Back 30](#_Toc45633101)

[Stage 11: Best Practices 30](#_Toc45633102)

## **Scope and Purpose –**

## Scope

Automating entire end-to-end release process.so that developers can build fast, deploy fast, test fast, fail fast and thus developers get to see the problems before applications go to productions and reduces manual effort.

## Purpose

This document will detail the steps for onboarding into Azure devops

## 

## **Pre-requisites**

* Ensure GIT tool is installed by every user in their local machine. Please use below link for GIT tool installation

“https://abbvie.service-now.com/self\_service?id=evg\_sc\_cat\_item&sys\_id=9a491dab37d56a002e9383dcb3990e65”

* During azure onboarding phase,drop an email to the Admin team to get access to Azure portal
* Ensure that every user of the project team has access to azure devops URL and added as part of corresponding team as below

“<https://dev.azure.com/abbvie-devops-lab/Abbvie%20BTS>”

|  |  |
| --- | --- |
| **Team Name** | **Details** |
| CDL-BTS-ADMIN | Admin level access |
| CDL-BTS-contributors | Developers who will have contribute access to repo and create access to build/release pipeline |
| CDL-BTS-approvers | Operations team/Project leads for any code review before merging to release branch |
| CDL-BTS-app-support | Operations team for QA deployment approval |

* Ensure below branches are in place for the repo’s

**Note: Admin team is responsible for creating any branches, Developers will not have access to create branches**

|  |  |  |
| --- | --- | --- |
| **Branch Name** | **comments** | **Naming Standard** |
| Feature | 1) Branched off of the Develop branch 2) Used by developers to develop new code for a current of future release and deploy to Dev | <Projectname\_resource>  Ex: EndoSHA\_configuration |
| Develop | 1) Long lived 2) An integration branch for changes you are preparing to move to higher environments like QA deploy | develop |
| Release Branches | 1) Branched off of the Master branch when you are ready to cut a release, Using PR, merge the latest code to release branch 2) Merge back into develop AND master when a release is completed 3) Used to create a chunk of code that is ready to go to Production | Release1.0\_Projectname  Ex: Release1.0\_EndoSHA |
| Master | 1) Long lived 2) Copy of code currently deployed to Production | master |
| Hotfix | 1) Used to add a change directly into production | Hotfix\_ Release1.0\_Projectname  Ex: Hotfix\_Release1.0\_EndoSHA |

* Below CDL Repo’s are available in Azure

|  |  |
| --- | --- |
| **System Name** | **Repo name** |
| CDL - Platform | CommercialDatalake-platform |
| CDLS | CommercialDatalake-syndicated |
| CDLM | CommercialDatalake-marketing |
| CDLO | CommercialDatalake-operations |
| CDLMA | CommercialDatalake-market-access |
| CDLH | CommercialDatalake-hcp360-operations |
| CDLHO | CommercialDatalake-hcp360 |
| CDL - Platform | CommercialDatalake-framework |

**Access provisioning**

Before getting on boarded into azure devops, developers should have access to the below items

* Team level access
* Repo level access
* Build pipeline access (To build Artifacts)
* Release Pipeline access( To deploy build artifacts into Dev,QA,Prod env)
* Deployment group access

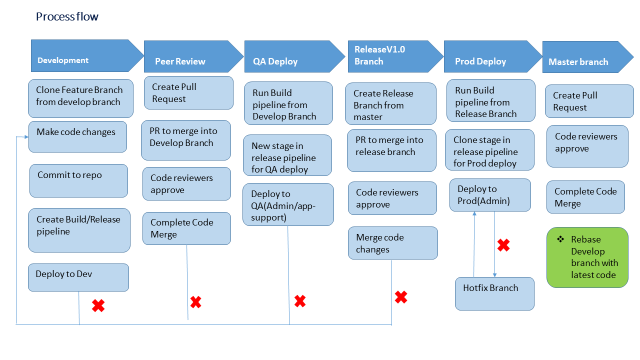
**For any Naming Standard – Refer the below excel sheet**



For Folder structure details which includes build, release pipeline &deployment group refer below sheet



## **Process Flow -**



### 

### Stage 1: Clone feature Branch

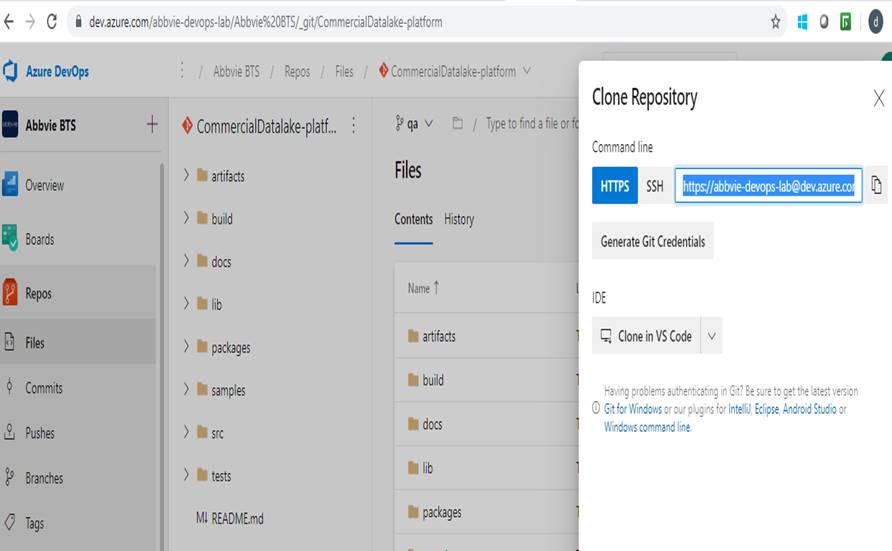
Only Admin team will have access to create a feature branch it’s per project per application

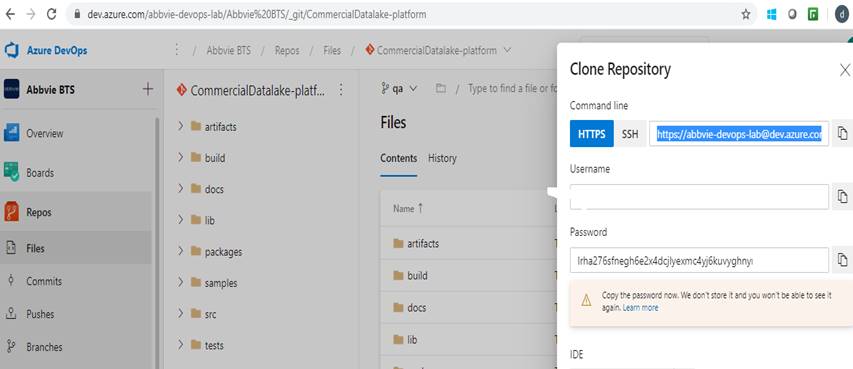
Feature Branch Naming Standard : <Projectname\_resource>

Clone a feature branch

Steps to be followed:

1. To clone feature branch repository, select feature branch of the project repo and Click the **Copy to clipboard** button next to the repo clone URL. You can plug this URL into any Git-compatible tool to get a copy of the codebase.
2. Click on the Generate Git Credentials
3. Copy the Git password

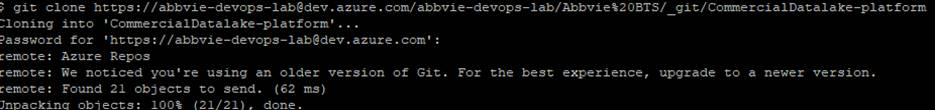




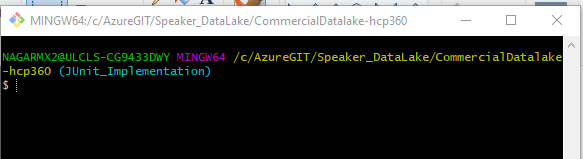
1. Login to local Machine where you would like to clone the Azure repo
2. Execute Git clone command

$ git clone <https://abbvie-devops-lab@dev.azure.com/abbvie-devops-lab/Abbvie%20BTS/_git/CommercialDatalake-platform>

Provide the password copied from generate git credentials



1. Navigate to the folder once repository is cloned in your local machine and right click on the folder and select **Git Bash Here**



Note : Developers will work on cloned copy .

1. Make code changes only in respective folder
2. After making all the changes .Use git add command only for the folder path where changes are made

Cmd : "git add src/application/readme.txt"

1. commit the code using "git commit -m "code change "
2. git push to feature branch using below command

Cmd : "git push --set-upstream origin <Feature branch>"

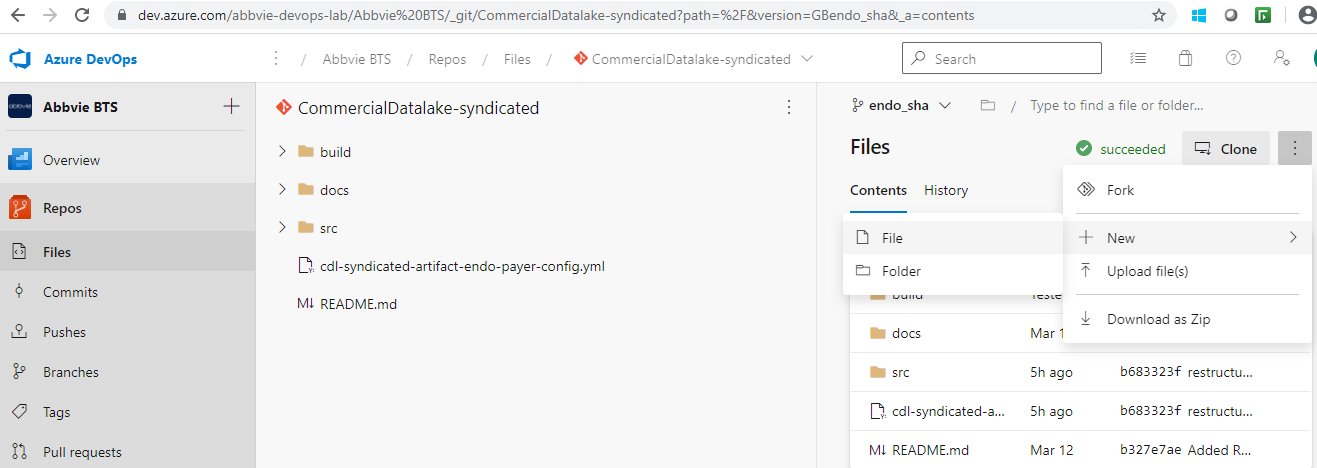
### 

### Stage 2: Build pipeline

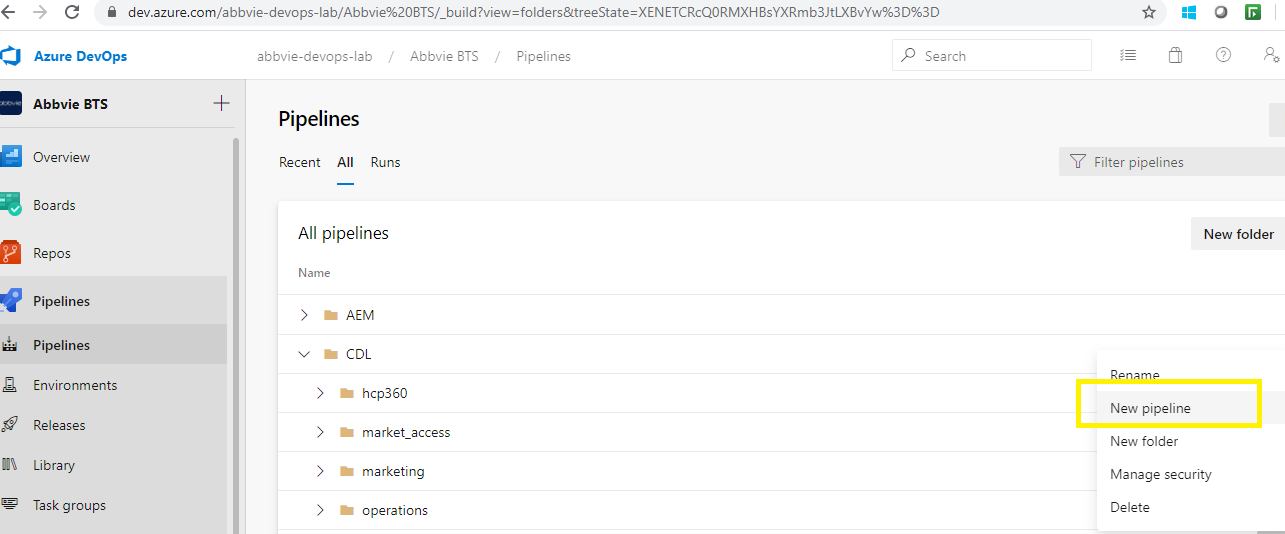
Build Pipeline Naming standard: “cdl-<SystemName>-build-<project name>-<Resource>”

Steps to create Build pipeline:

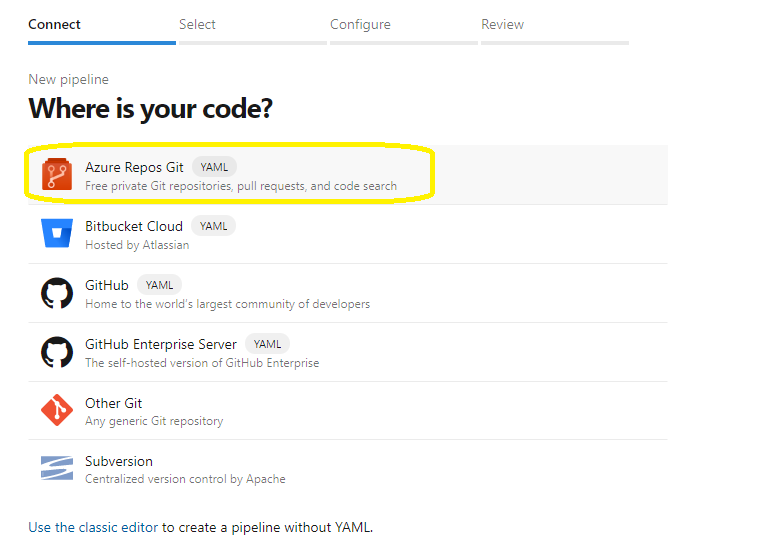
1. In a browser tab, navigate to abbvie-devops-lab -> Abbvie BTS on Azure devops
2. Goto repo->feature branch->click on the three dots to create new file

****

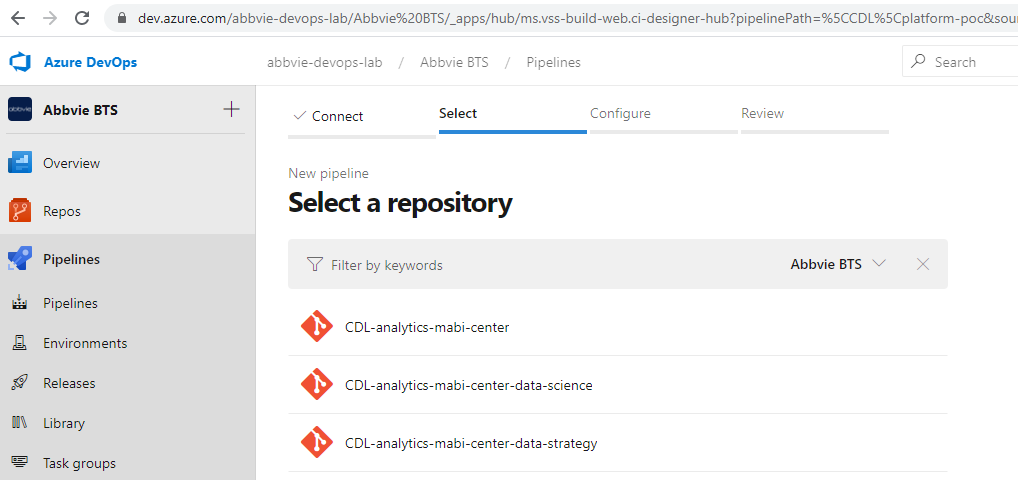
1. Create new yml file and save it
2. Select pipelines->pipelines on Abbvie BTS
3. Navigate to All under pipelines
4. Select CDL ->systemname(eg : syndicated)->project1>,under which select proper folder/resource for which new pipeline has to be created



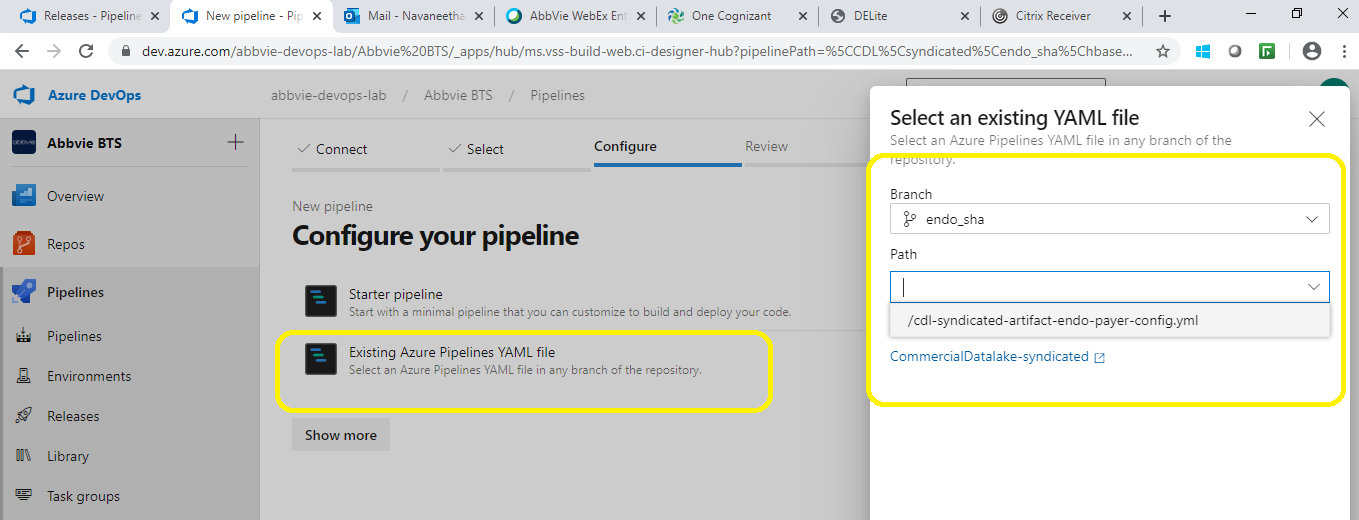
1. Click on newpipeline and select “Azure Repos Git



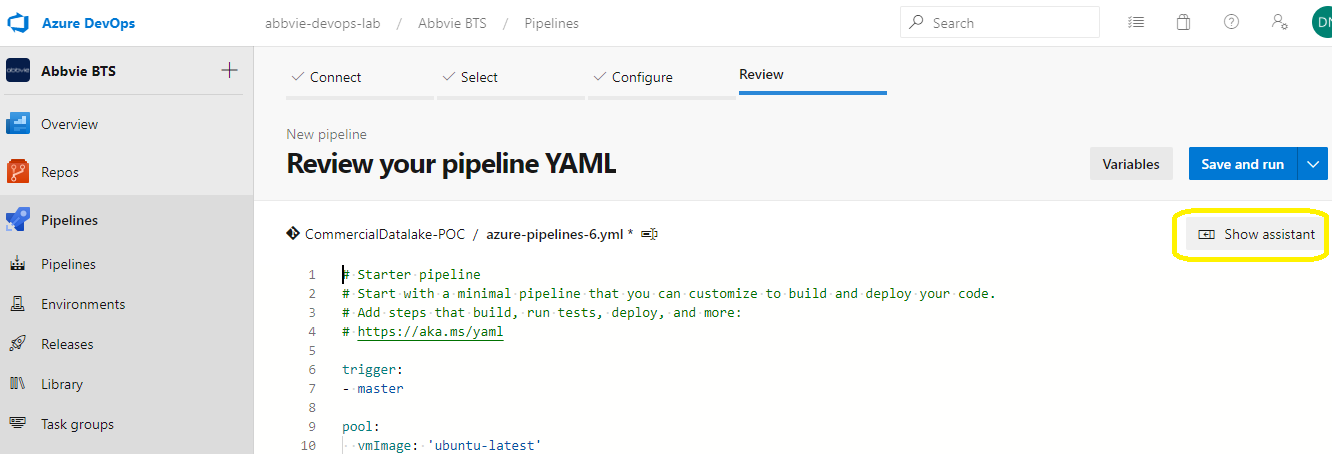
1. Choose proper repository and resource for which Build pipeline needs to be created ,then select continue

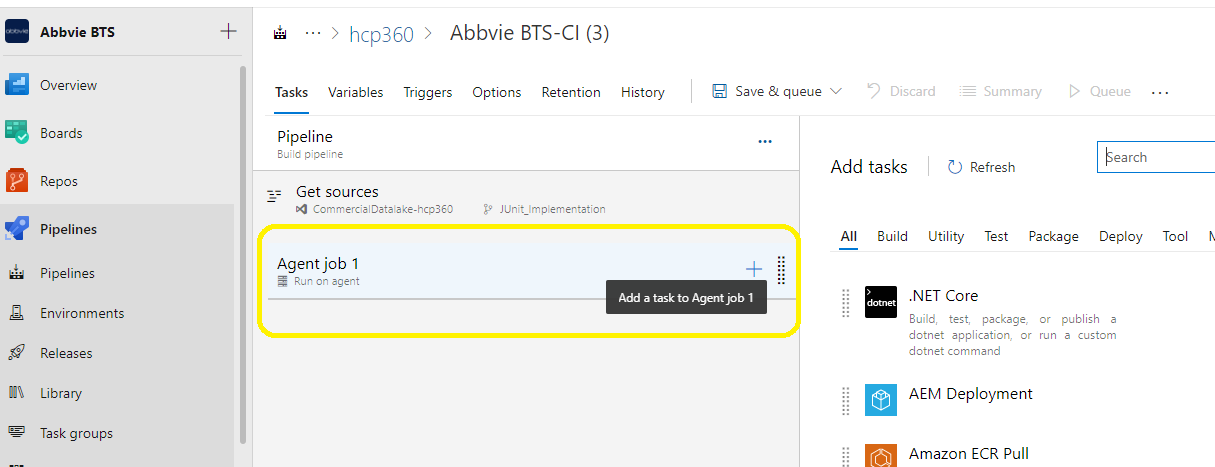


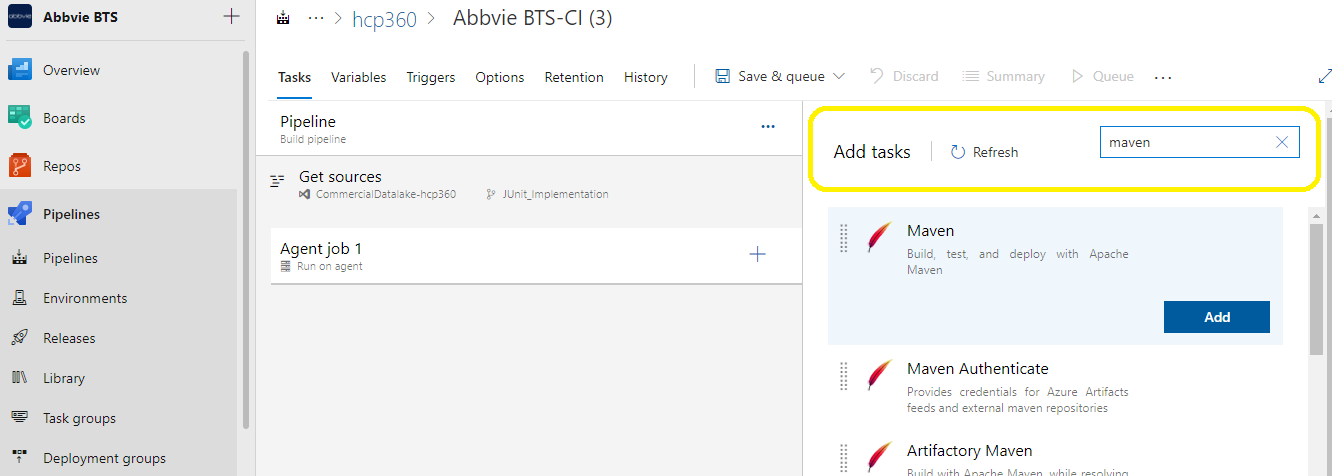
1. Select existing pipeline and edit the yaml



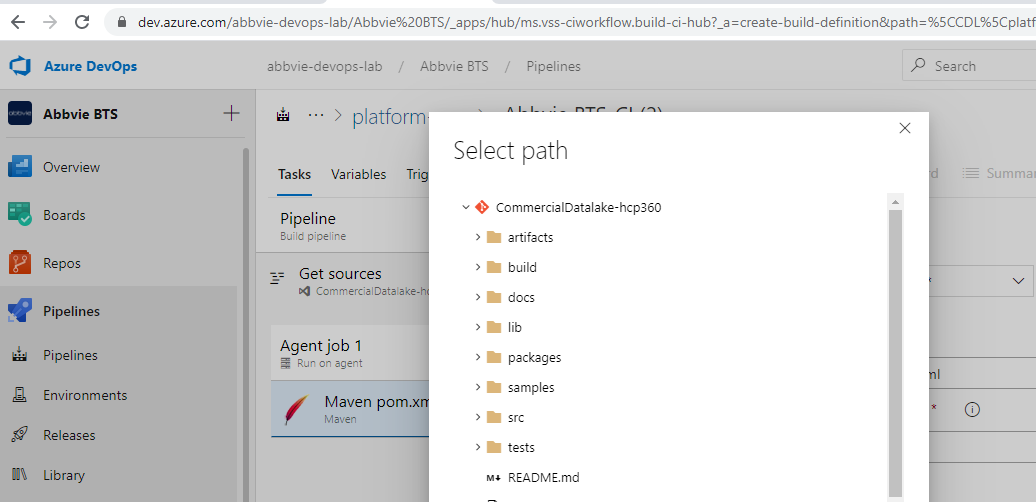
1. Now click on show assistant and add task to agent job by clicking the + symbol

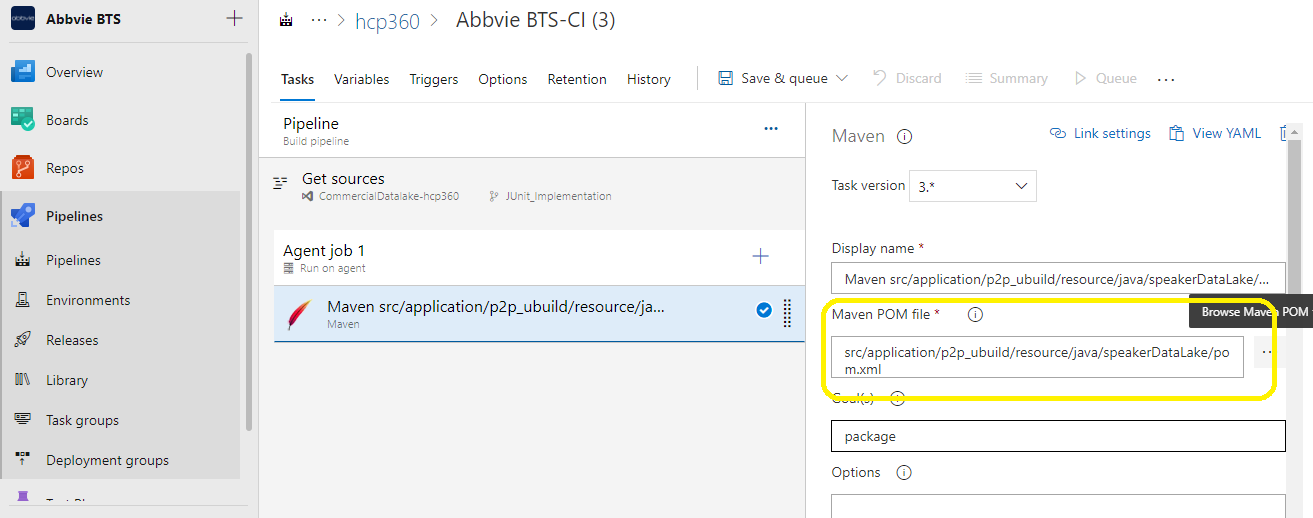


Search for Maven and add if maven build task needs to be added to build a jar/war file(For just a file copy maven build task is not required )  
  
  


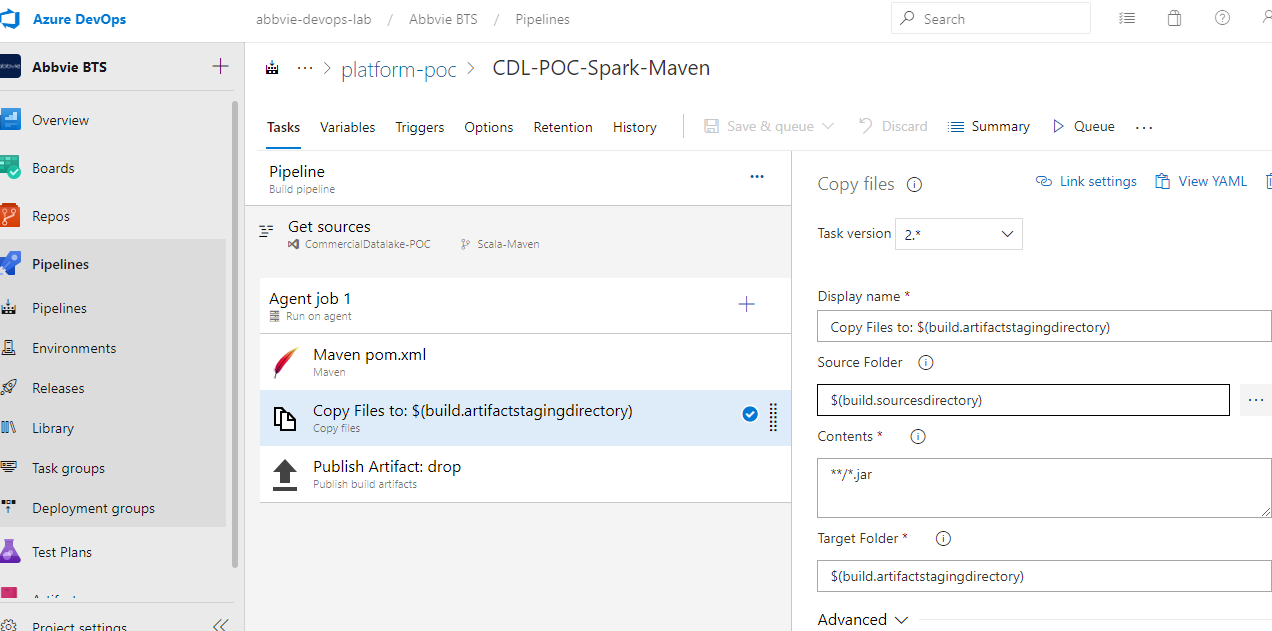


1. Select the appropriate pom file from repo folder which needs to be used

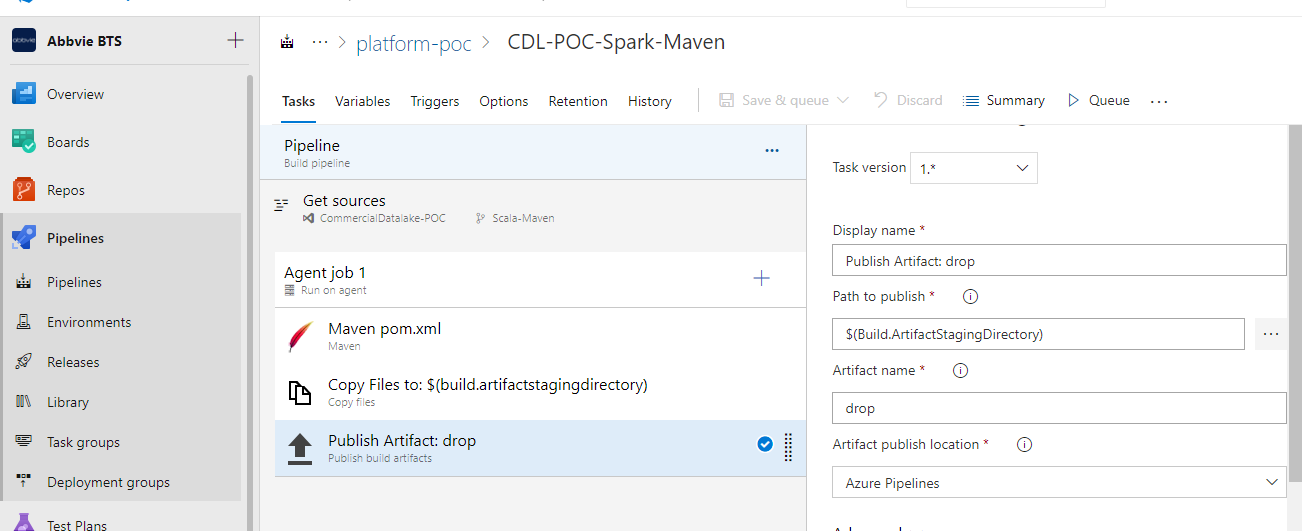




1. To copy the jar files add another task and look for copy files
2. Add copy files with the below parameters as below snapshot



1. To publish the Artifact add another task
2. Search and select publish artifact
3. Provide parameters as same as below snapshot



1. Yaml file will look as below for reference change yaml file name as cdl-<SystemName>-build-<project name>-<Resource>.yml

Example: yaml for config files build

# Starter pipeline

# Start with a minimal pipeline that you can customize to build and deploy your code.

# Add steps that build, run tests, deploy, and more:

# https://aka.ms/yaml

trigger:

- none

pool:

  vmImage: 'ubuntu-latest'

steps:

- task: CopyFiles@2

  inputs:

    SourceFolder: 'src/endo\_payor\_reports/configuration'

    Contents: '\*\*/\*'

    TargetFolder: '$(build.artifactstagingdirectory)'

- task: PublishBuildArtifacts@1

  inputs:

    PathtoPublish: '$(Build.ArtifactStagingDirectory)'

    ArtifactName: 'cdl-syndicated-artifact-endo-payer-config'

    publishLocation: 'Container'

Example : Yaml file for Maven build

# Maven

# Build your Java project and run tests with Apache Maven.

# Add steps that analyze code, save build artifacts, deploy, and more:

# https://docs.microsoft.com/azure/devops/pipelines/languages/java

trigger:

   -none

  paths:

    include: ['src/common/resource/python']

pool:

  vmImage: 'ubuntu-latest'

steps:

- task: Maven@3

  inputs:

    mavenPomFile: 'src/pom.xml'

    mavenOptions: '-Xmx3072m'

    javaHomeOption: 'JDKVersion'

    jdkVersionOption: '1.8'

    jdkArchitectureOption: 'x64'

    publishJUnitResults: true

    testResultsFiles: '\*\*/surefire-reports/TEST-\*.xml'

    goals: 'package'

- task: CopyFiles@2

  inputs:

    SourceFolder: '$(build.sourcesdirectory)'

    Contents: '\*\*/\*.jar'

    TargetFolder: '$(build.artifactstagingdirectory)'

- task: PublishBuildArtifacts@1

  inputs:

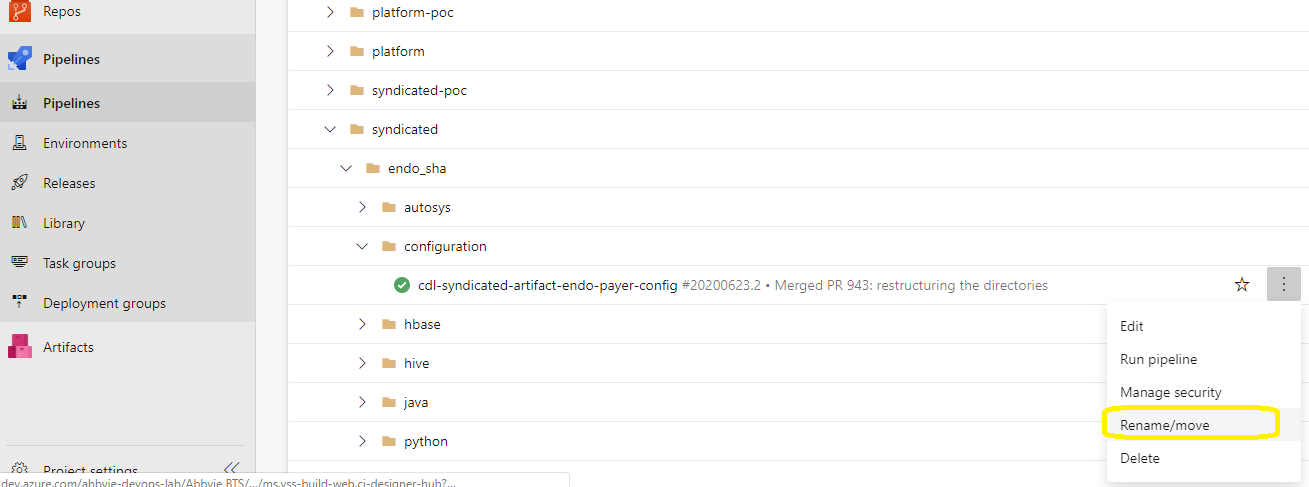
    PathtoPublish: '$(Build.ArtifactStagingDirectory)'

    ArtifactName: 'drop'

    publishLocation: 'Container'

Finally Save & run the job for execution

Note : Change the pipeline name as “cdl-<SystemName>-build-<project name>-<Resource>”



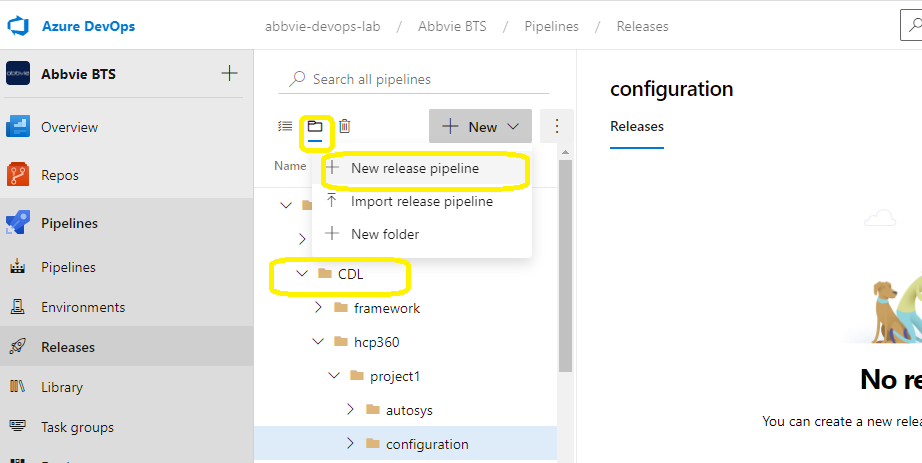
### Stage 3 : Release Pipeline

Release pipeline Naming standard: “cdl-<SystemName>-release-common-<Resource>”

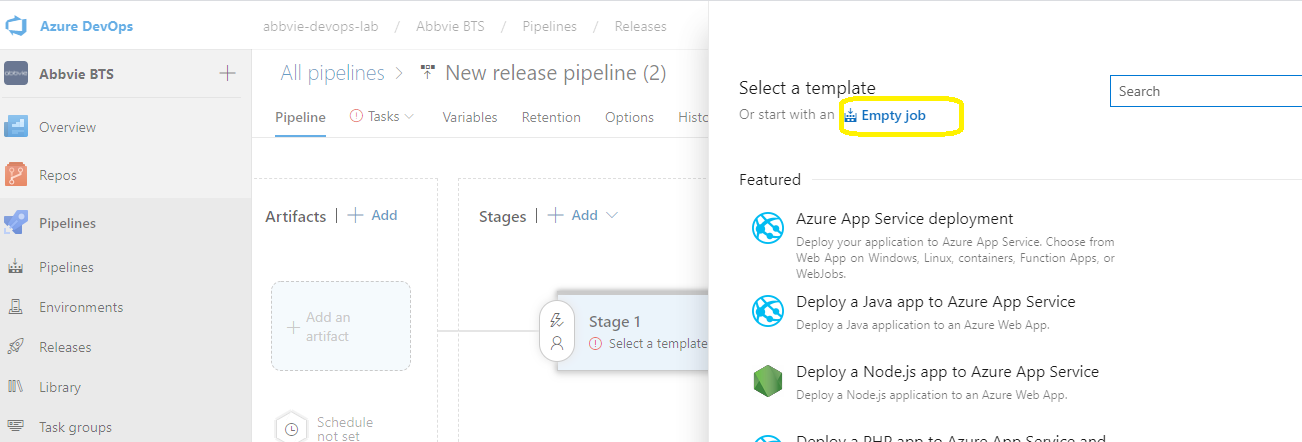
Artifact Naming Standard: “cdl-<SystemName>-artifact-<project name>-<Resource>”

Source alias Naming Standard: “release”

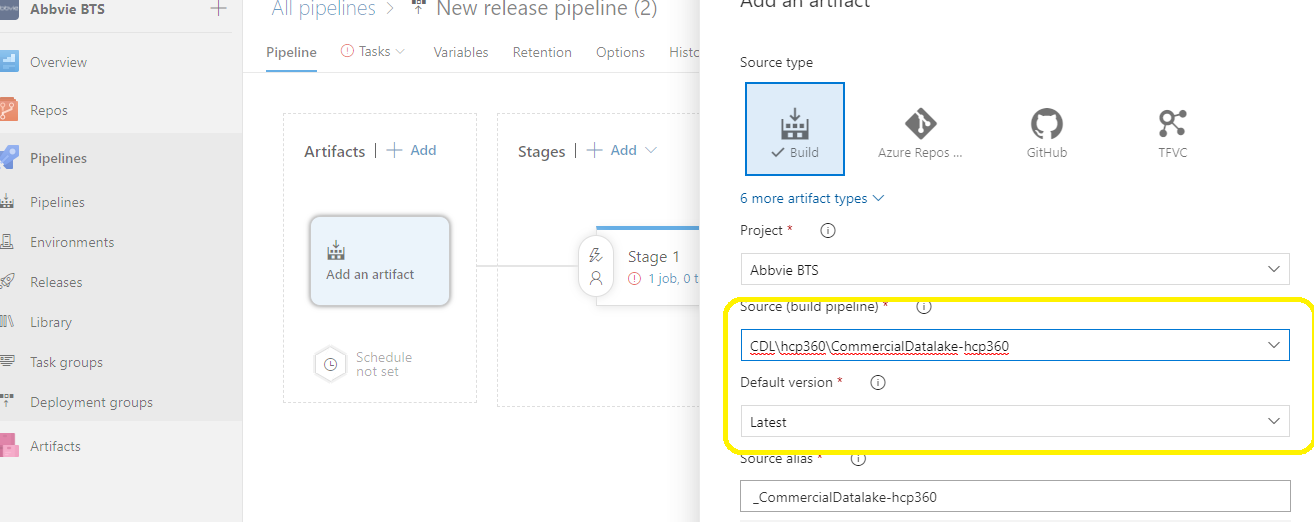
* Select pipelines->releases->folder->CDL-><System>-
* Click on + New and select new release pipeline



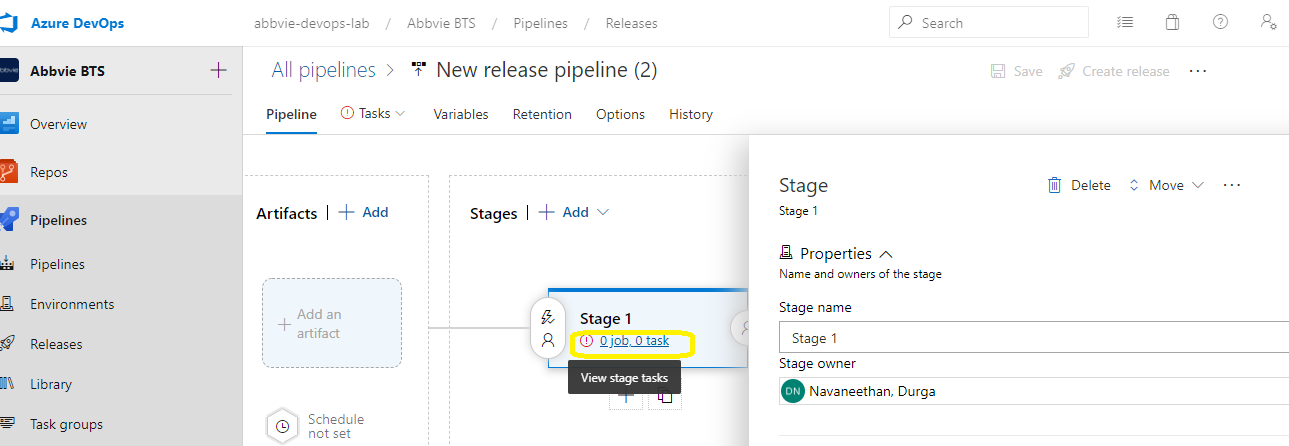
1. Select Empty job



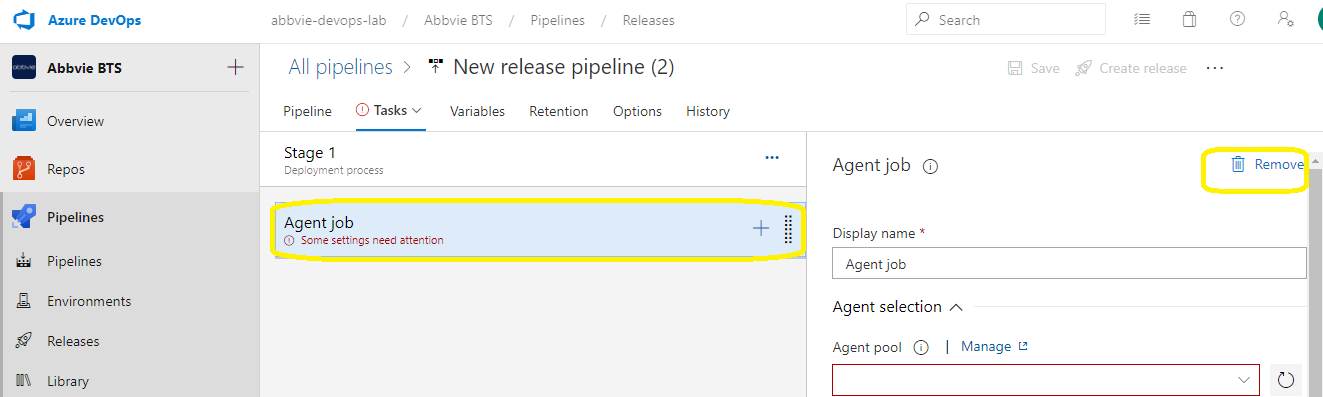
1. Select Add an artifact and choose the project repo path from where the artifacts need to be pulled



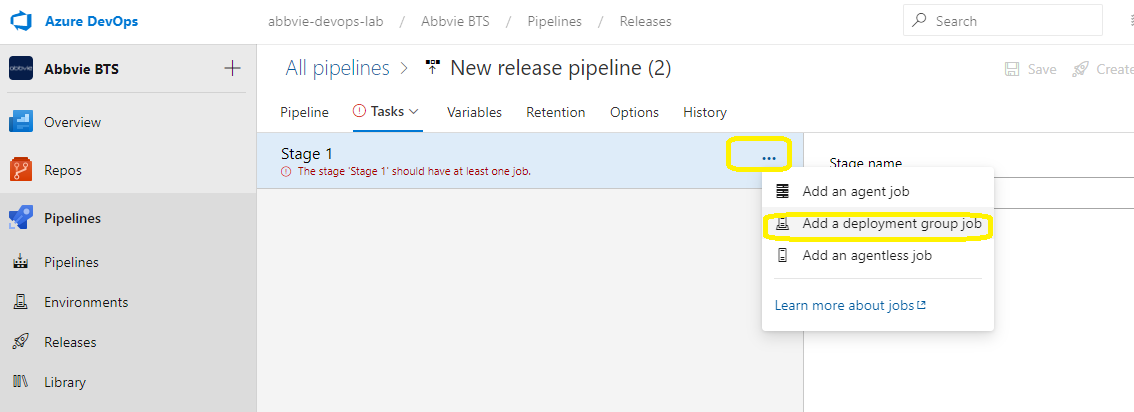
1. Select job



1. Select agent Job and click remove icon on the right side

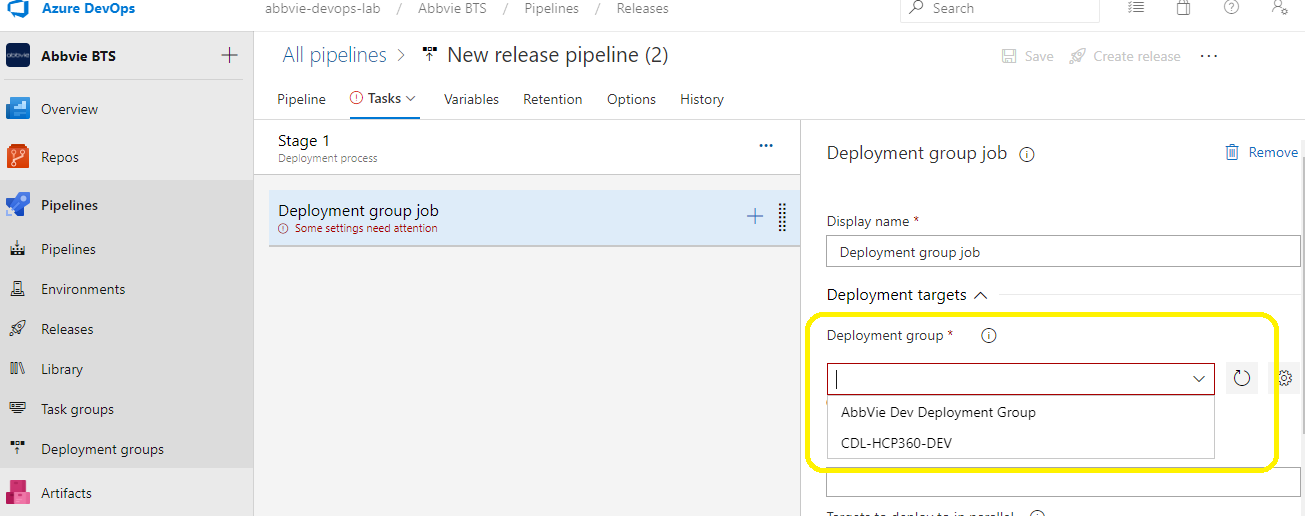


1. Click on the three dots of stage 1 and select “add a deployment group job “



1. In the deployment group drop down select the appropriate environment (Dev,QA,Prod )where we need to deploy the code

Note: Developers will have access to select only dev deployment group . If it’s not added in deployment group you will not able to see env details in drop down list



g) Adding Task Group:

Task group can be added using the below following methods

1. Add a bash task group and have all the commands/steps to deploy
2. Or use the predefined task group “CDL\_Release”

To add bash task

* Click on + and search for bash script
* Select Add
* Select inline
* add all the deployments steps commands as below

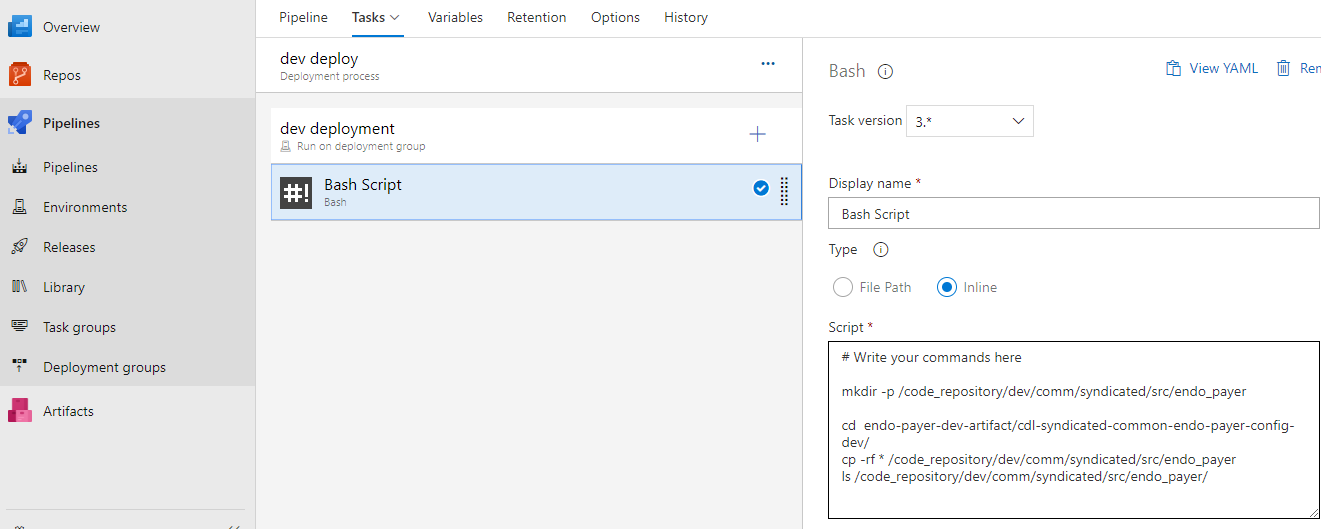
“# Write your commands here

mkdir -p /code\_repository/dev/comm/syndicated/src/endo\_payer

cd endo-payer-config-artifact/cdl-syndicated-common-endo-payer-config/

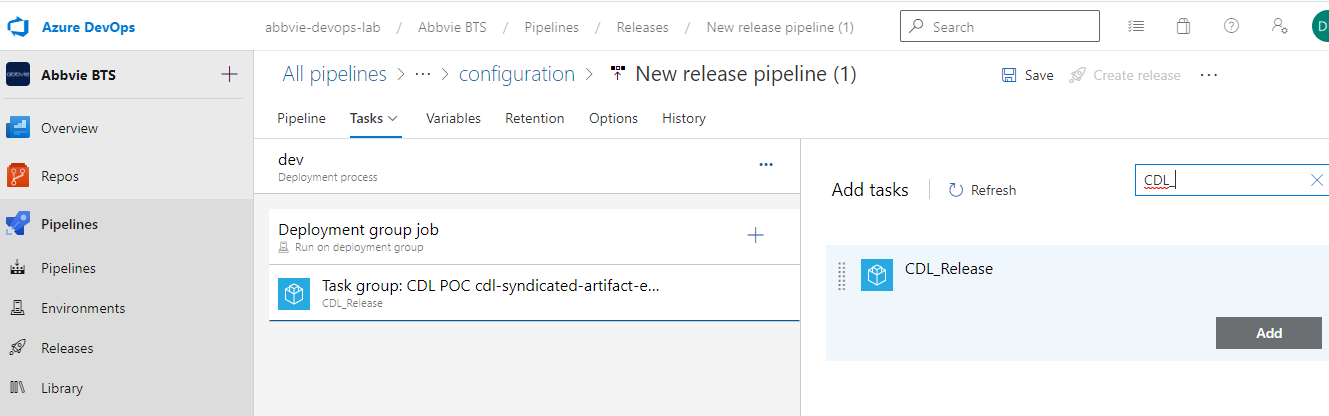
cp -rf \* /code\_repository/dev/comm/syndicated/src/endo\_payer

ls /code\_repository/dev/comm/syndicated/src/endo\_payer/”



To add predefined task group

* Click on + and search for CDL\_Release

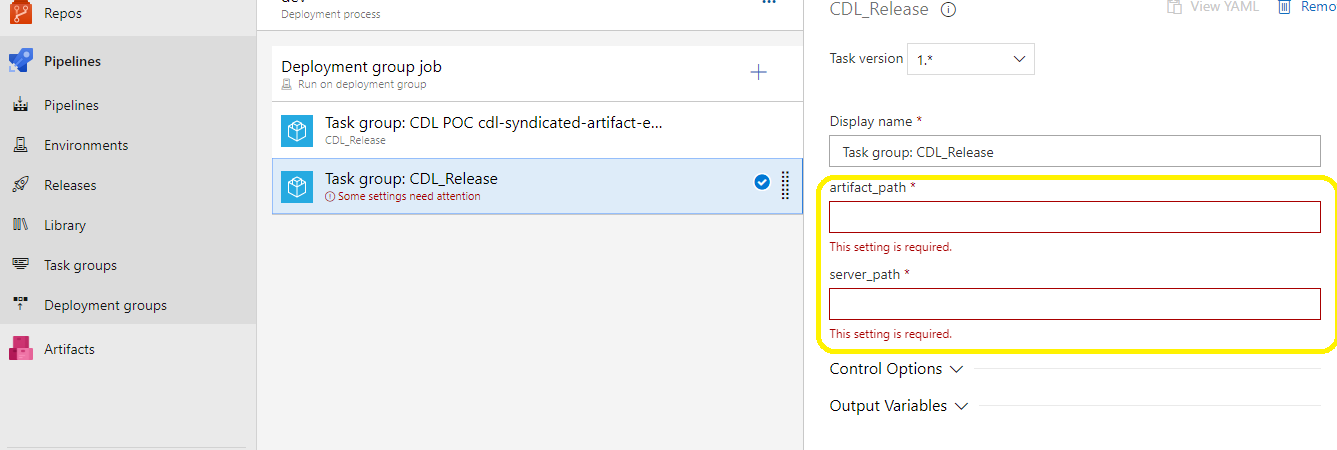


* Click on Add
* Provide artifact path as “endo-payer-config-artifact/cdl-syndicated-common-endo-payer-config/”

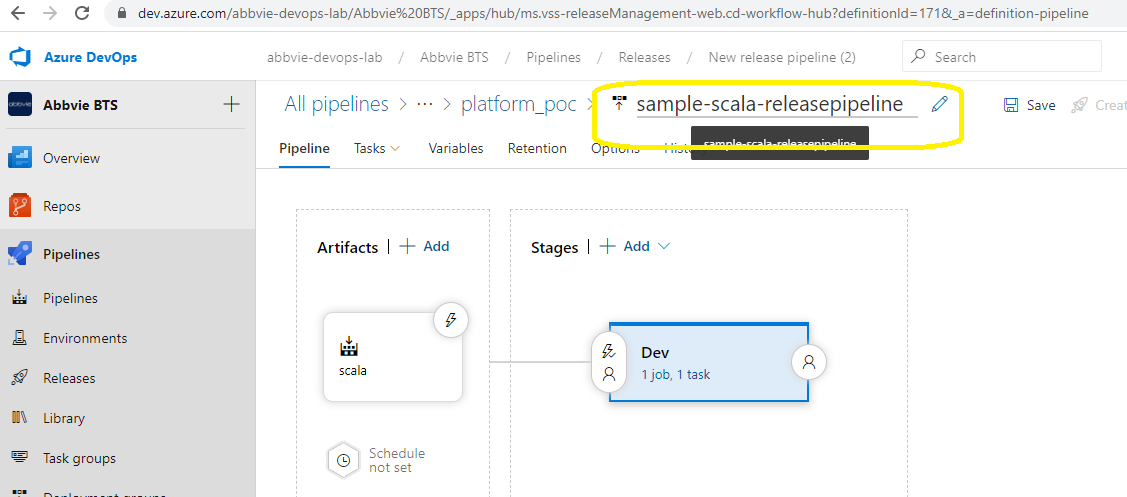
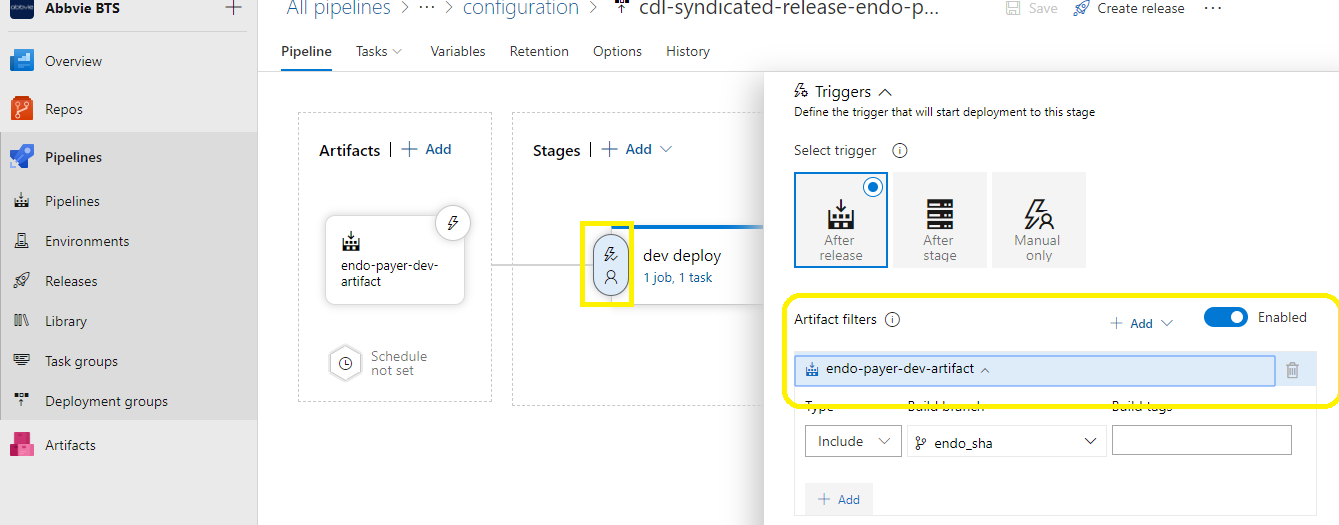
<source alias name /artifact name>

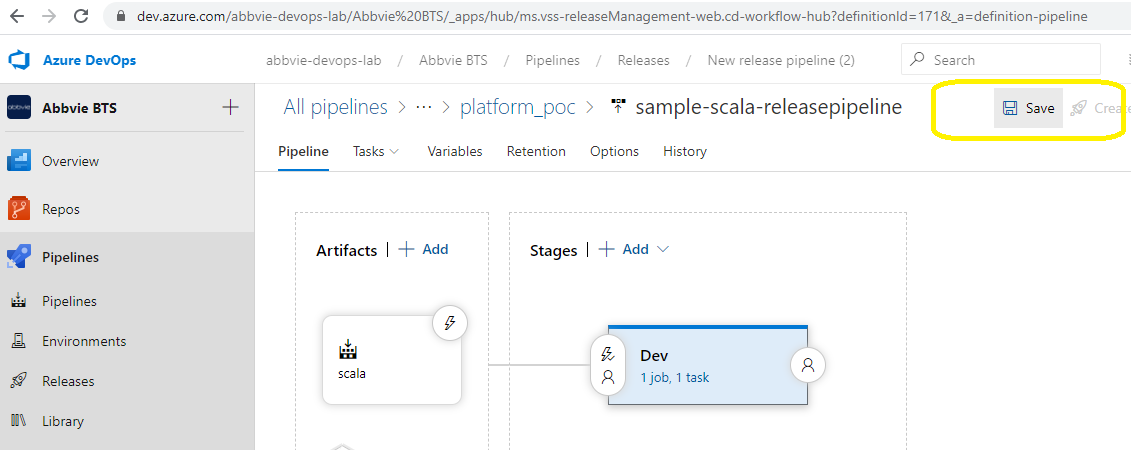
* Server path as your edge node path where the code has to be deployed

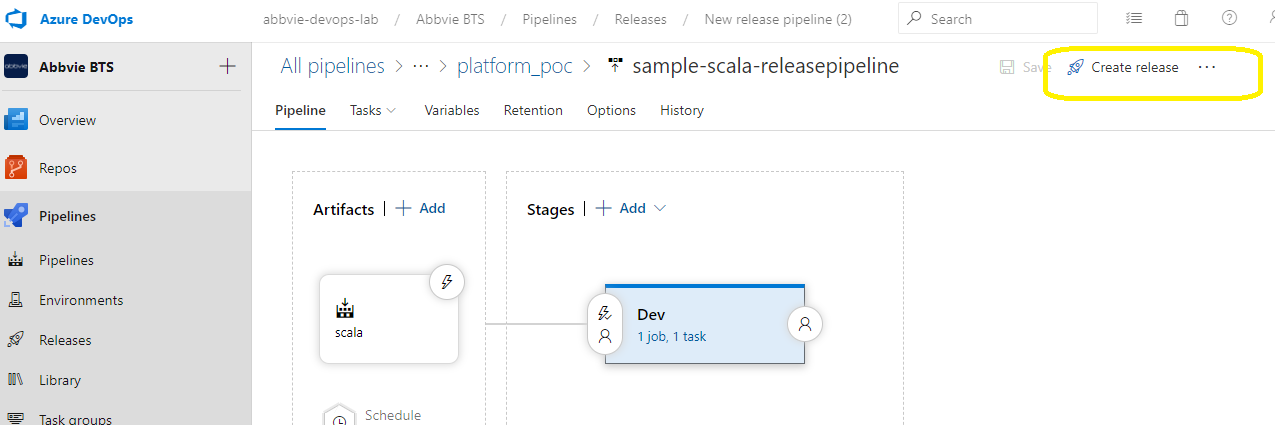
EX : /code\_repository/dev/comm/syndicated/src/endo\_payer

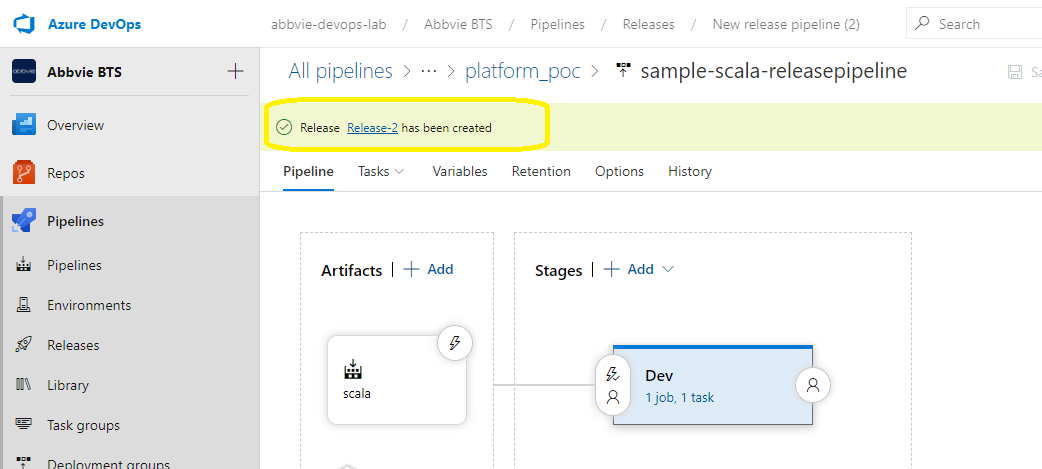


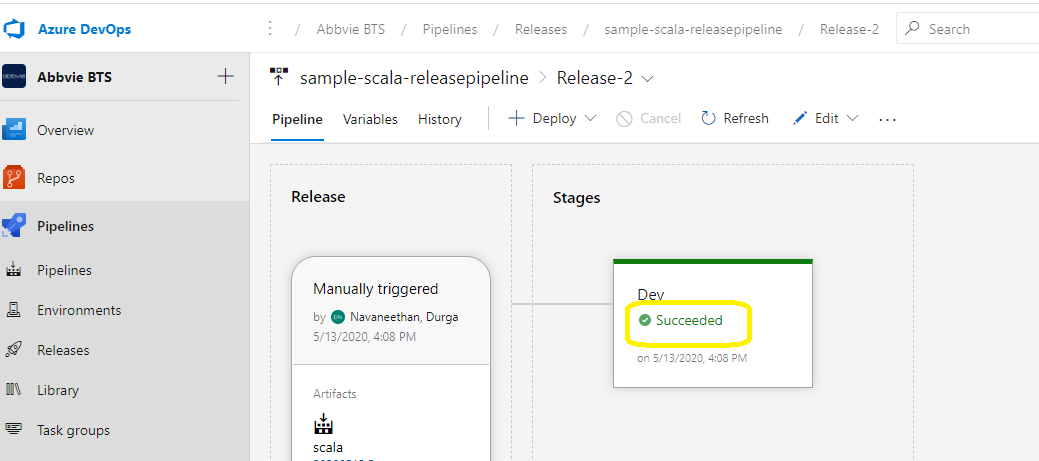
1. Rename the release pipeline as per the standards ”cdl-<SystemName>-release-common-<Resource>”

  
  
i) Select pre-deployment conditions and enable artifact filter .Ensure we restrict the artifact from feature branch only for any dev deployment  
  
  
  
  
j) Save and click on create release



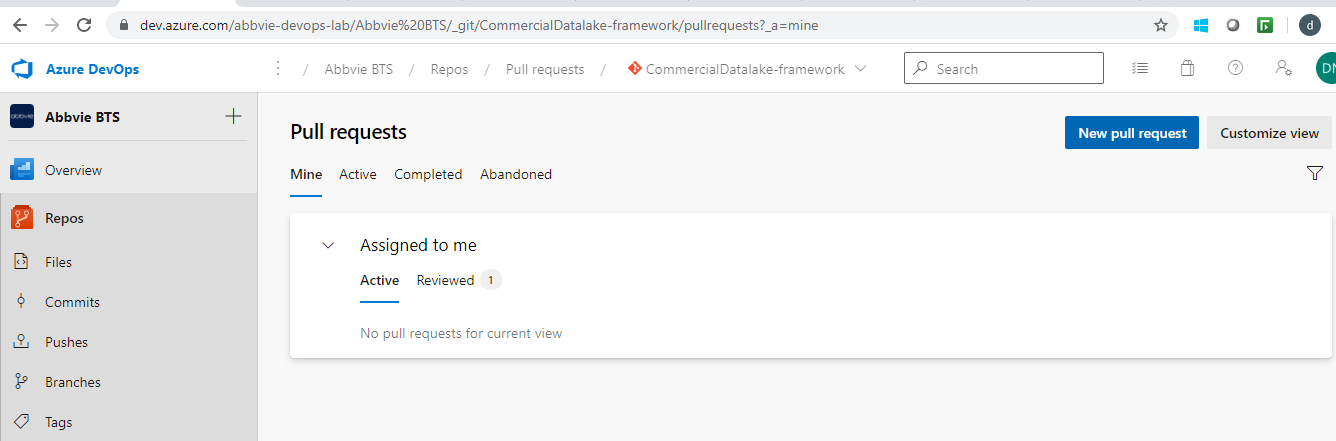


Navigate to release pipeline where we can find details/logs about the release  
  
  


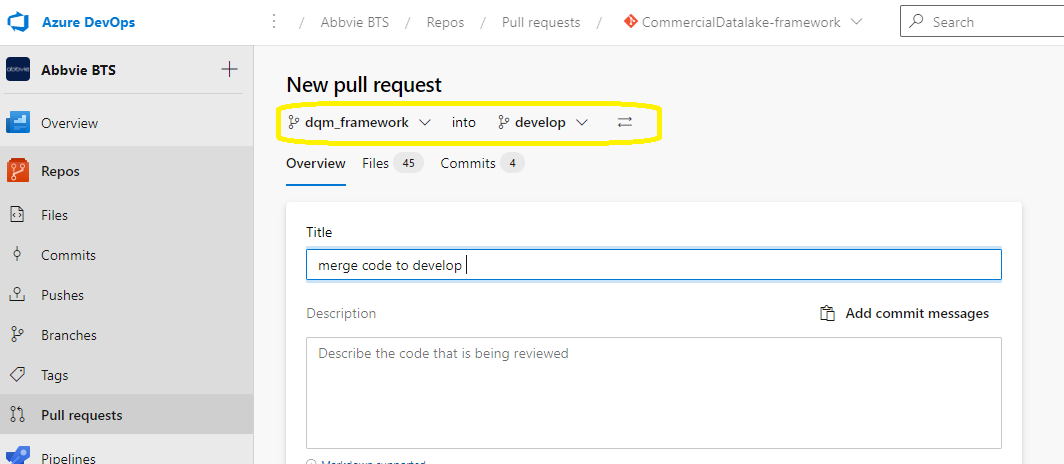


### Stage 4: Create PR to merge code to develop branch

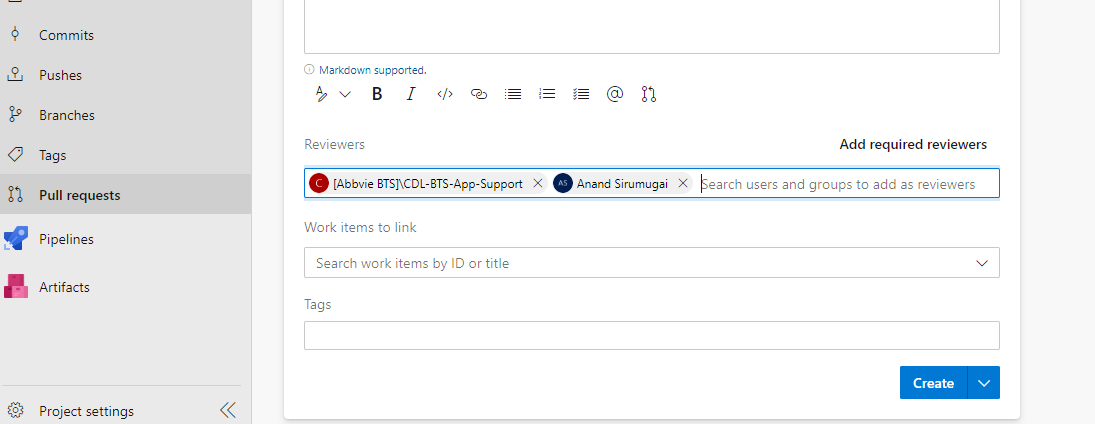
* Goto repo
* Select pull request->new pull request



* Select the feature branch into develop branch



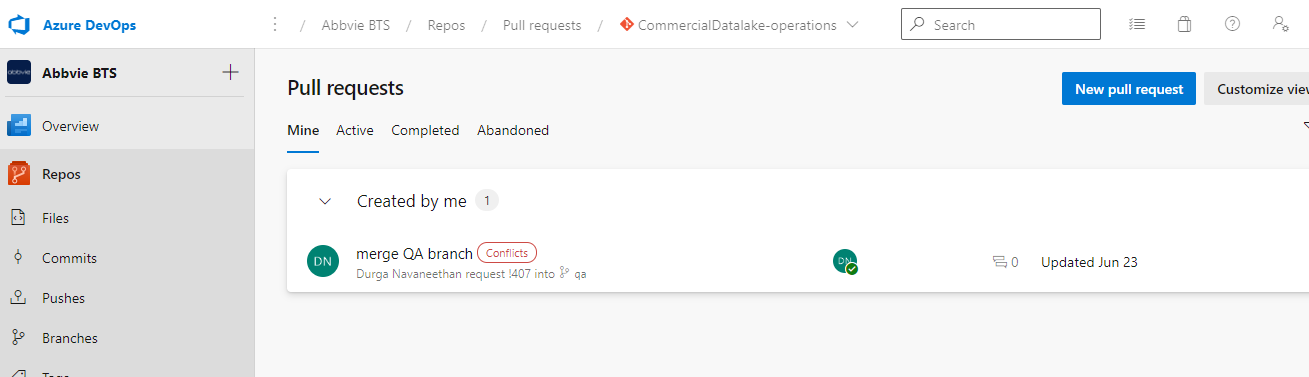
* Add reviewers details for code review before merging into Develop branch and click on create



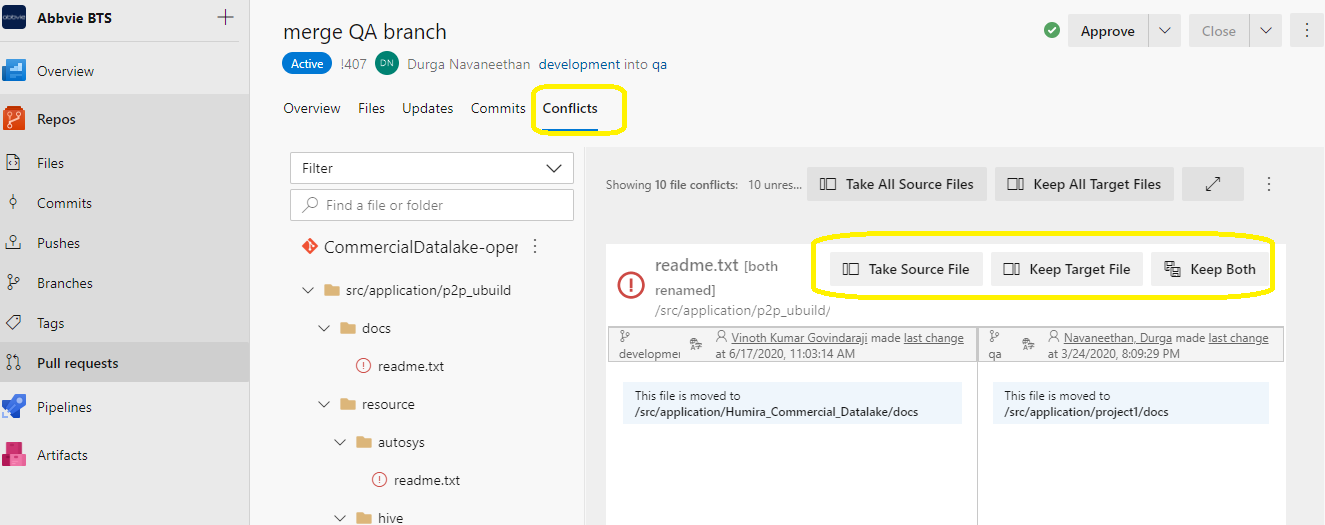
* If same file is edited by different developers conflict will arise while creating a pull request.

Developers should resolve the conflict before merging the code into develop branch

Ex : sample snapshot for conflits



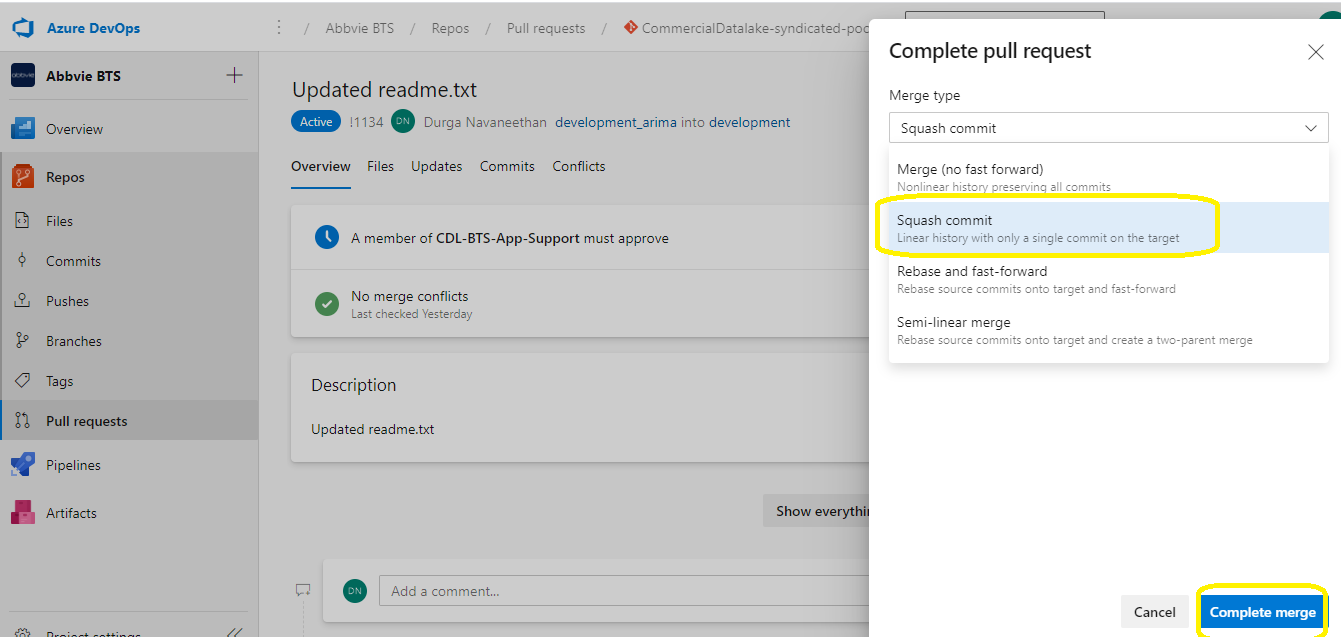
Select conflicts tab as below, which will show the code changes which created conflicts and developers needs to take a call on whether they want to “take source file /keep target file or keep both “ and create completing pull request



* Once pull request is reviewed and approved. click on complete



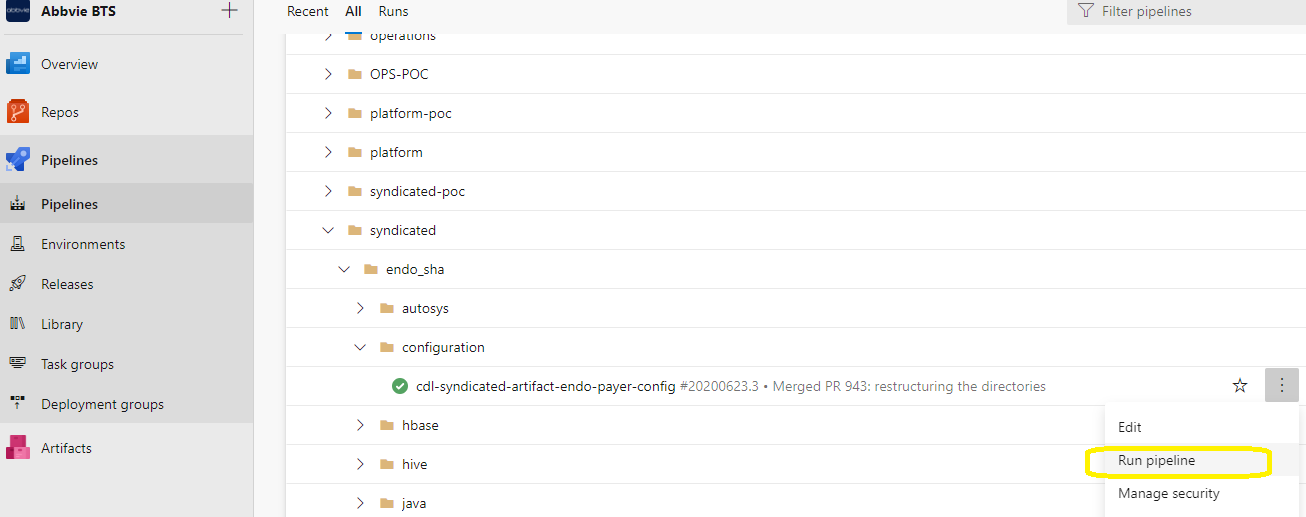
* As a best practice always select squash commit to maintain linear history of commits and complete merge



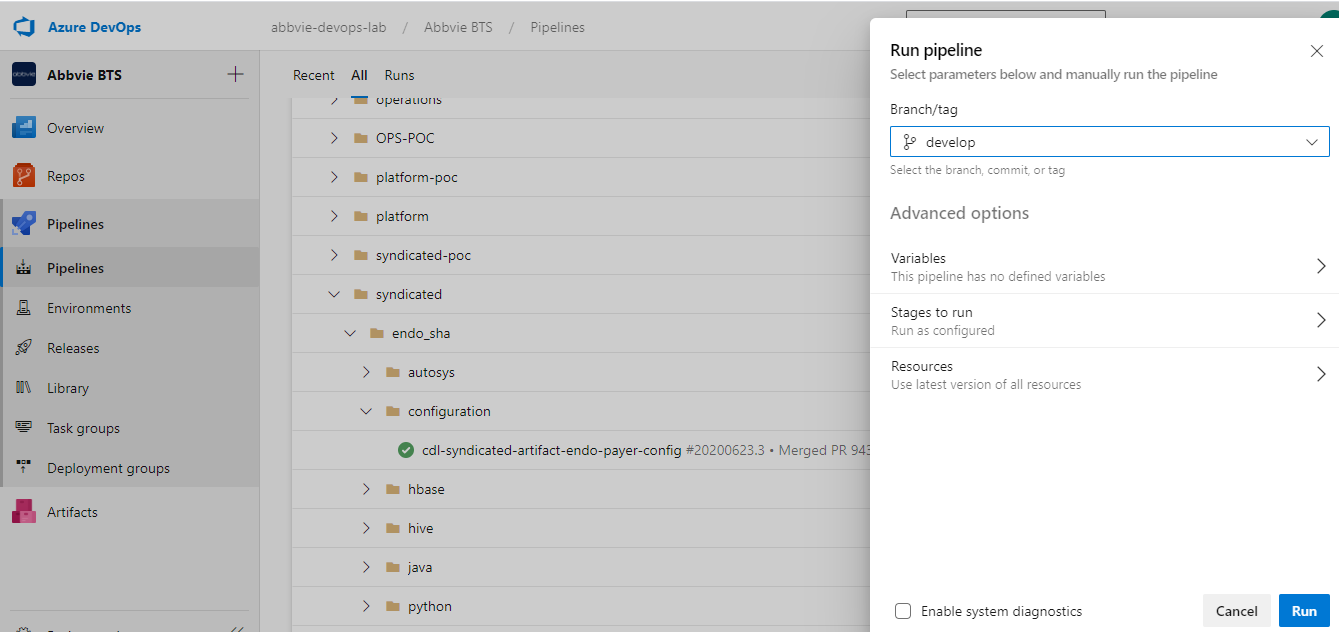
Note: Always ensure to pull the latest code from develop and merge the code

### Stage 5: QA deploy

* Once the code is merged to develop branch
* Use the same build pipeline and run the pipeline from develop branch for QA deploy instead of feature branch

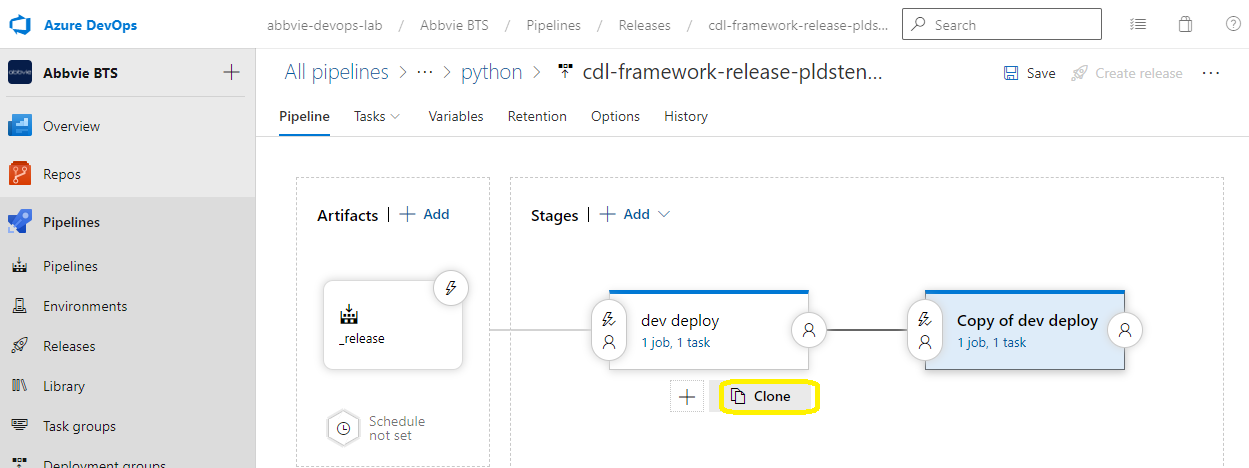


* Select develop branch and run

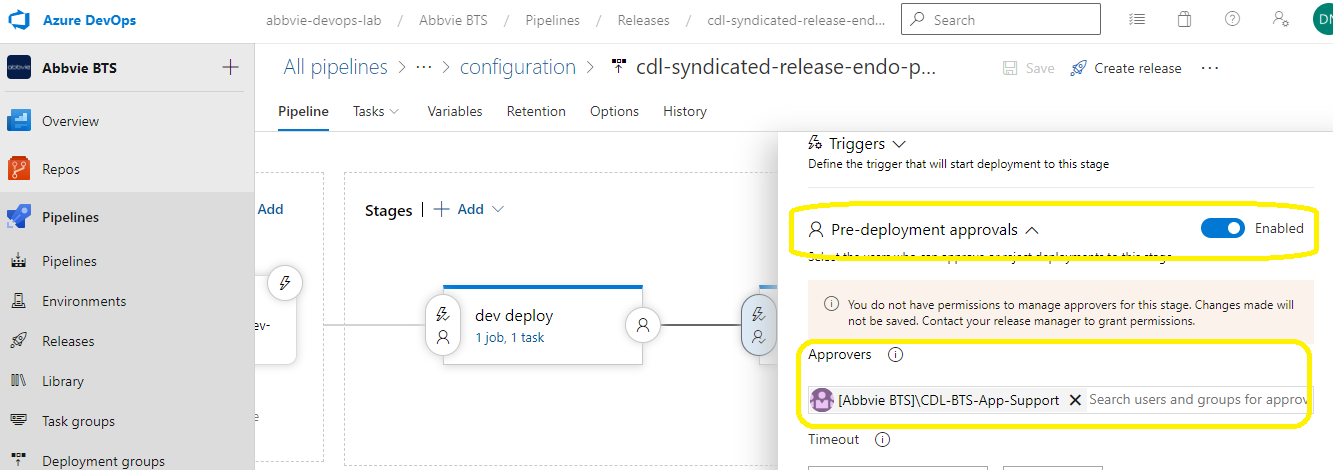


Release pipeline

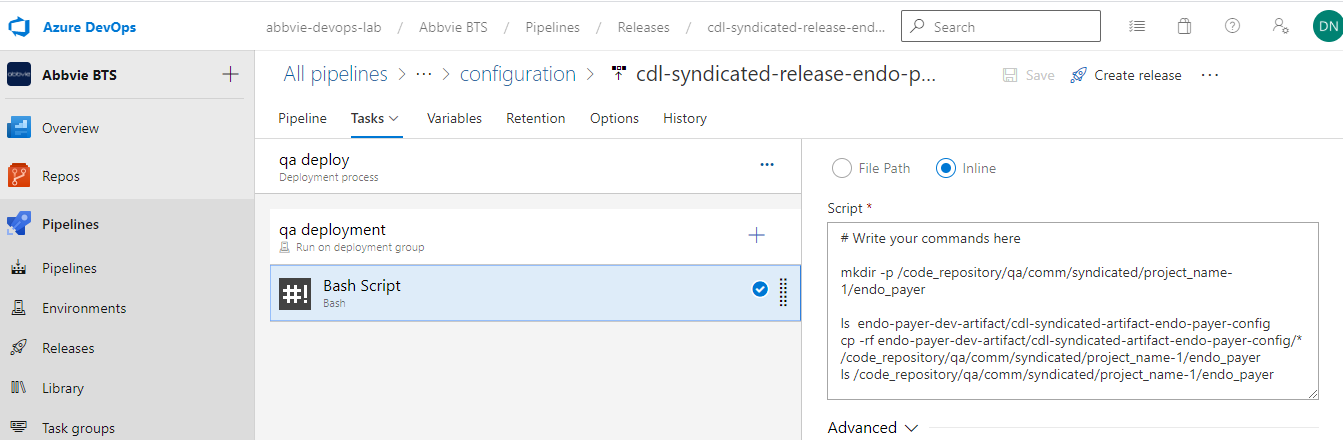
* Reuse the same release pipeline created for dev deploy
* Add another stage for QA deploy by cloning the dev stage



* In pre deployment condition enable artifact filter and restrict the artifact from develop branch only for QA deployment
* Enable pre-deployment approvals also and add CDL-BTS-app-support to approve the deployment



* Change the deployment path in bash script to QA path



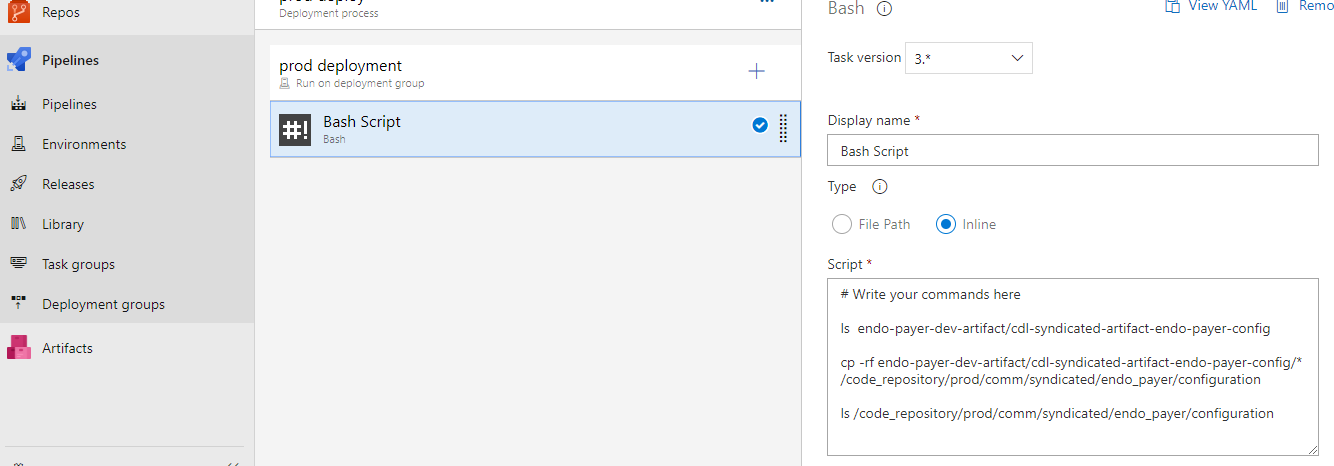
* Select dev deployment group only as developers will not have access to QA and prod
* Admin/Ops team will change the deployment group by the time they trigger QA deployment
* Save all the changes and have SIQ for QA deploy

### Stage 6: Create PR to merge code to Release branch

* Release branch will be created by Admin
* Refer section Stage 4 to create a pull request for merging code from develop into Release branch
* Cherry – pick the commits if needed which needs to go to production

### Stage 7: Prod Deployment

* Use existing build pipeline and run the pipeline from Release branch
* Use existing release pipeline and clone another stage for prod deployment
* In pre deployment condition enable artifact filter and restrict the artifact from release branch only for prod deployment
* Enable pre-deployment approvals also and add CDL-BTS-app-support to approve the prod deployment
* Change the deployment path in bash script to prod path



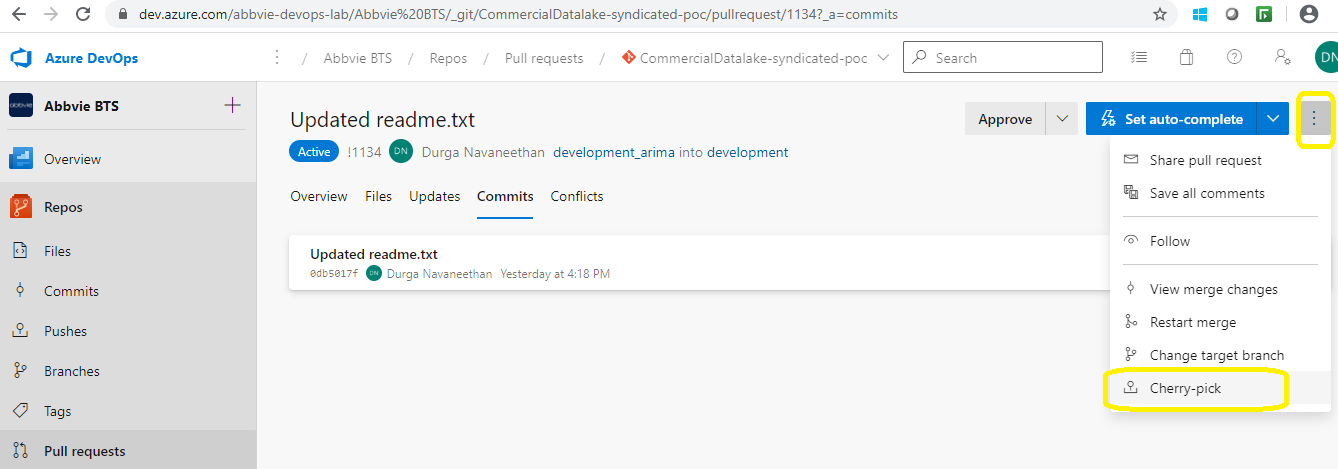
* Select dev deployment group only as developers will not have access to QA and prod
* Admin team will change the deployment group by the time they trigger Prod deployment

### Stage 8: Merge code to Master

* Once prod deployment is successful merge the code to master
* In case of failure have the code fix done in Hotfix branch and redeploy the code
* In case of Rollback ,use the previous release branch version of code to deploy into prod
* After successful prod deployment ,Rebase the develop branch using master as master branch will have the production like copy

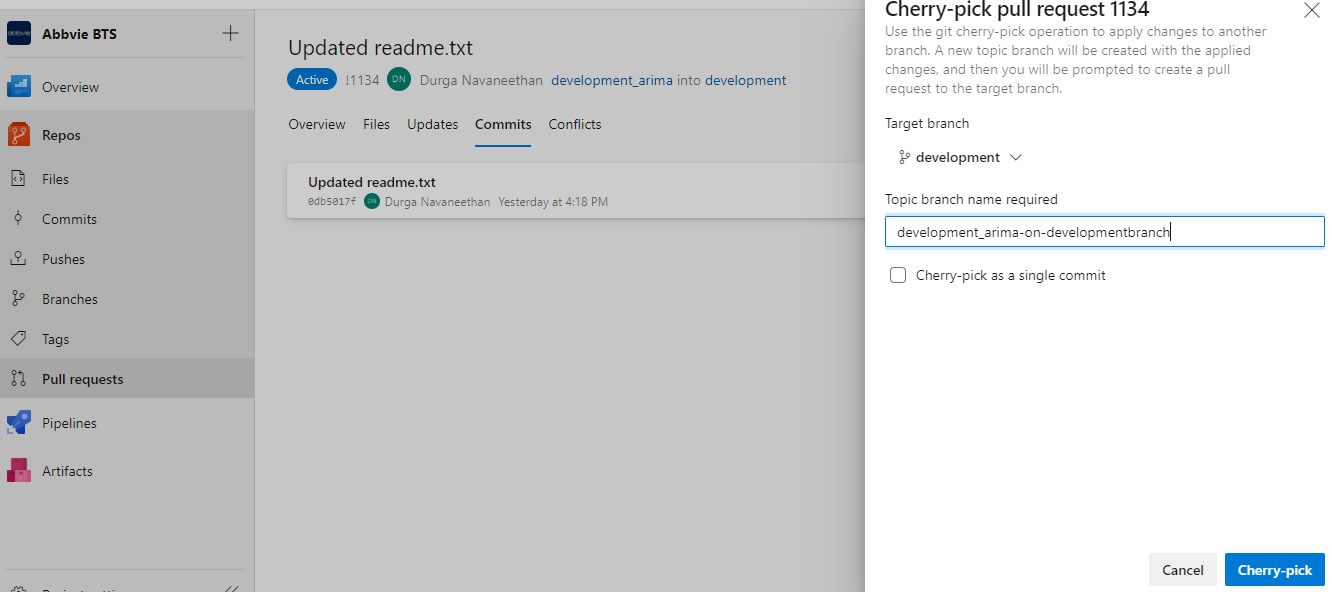
### Stage 9: Cherry-Picking

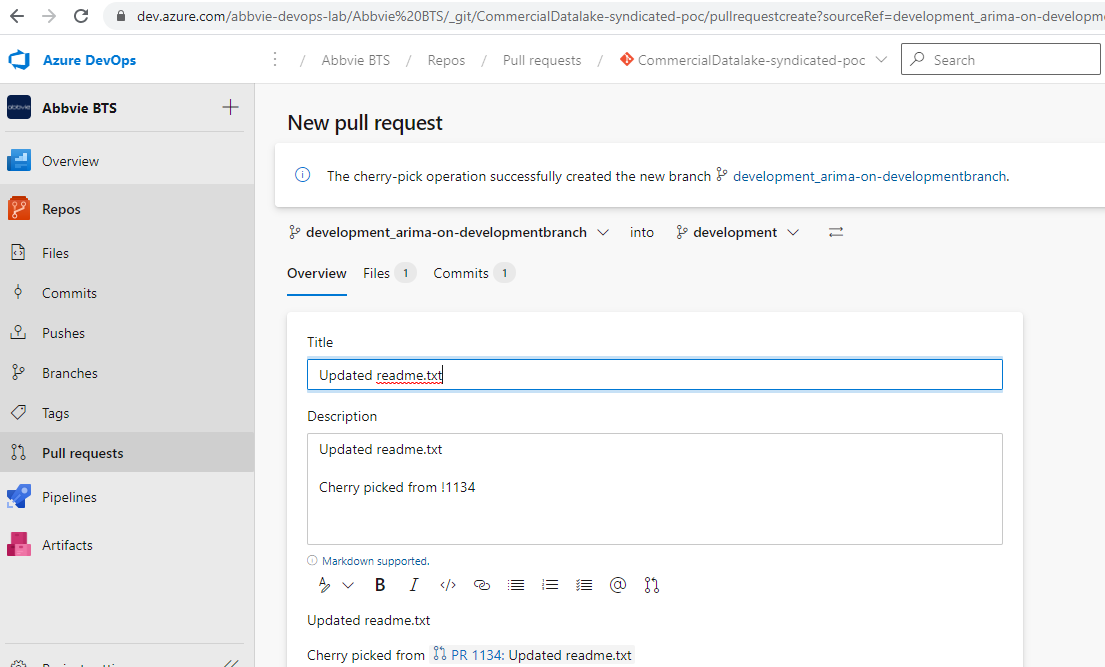
* To cherry pick a latest commit and merge it to any branch cherry-picking can be used
* Create a pull request and click on the three dots
* Select cherry-pick option



* Provide the target branch where cherry-pick changes to be merged

Note : Cherry-pick option will always create new branch and then new PR needs to be raised to merge back the changes to target branch



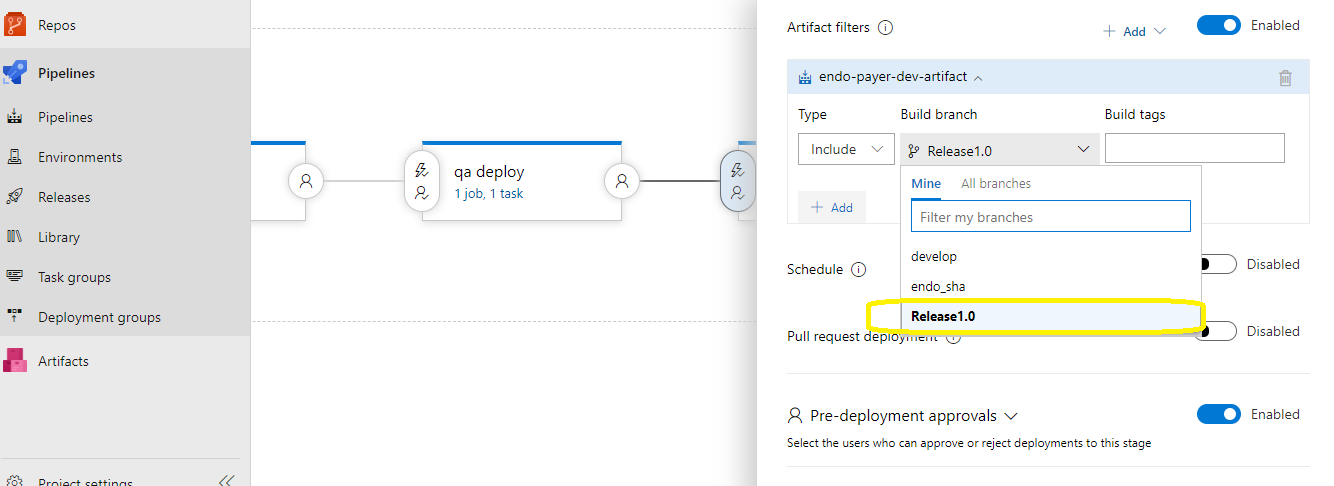


### Stage 10: Roll-Back

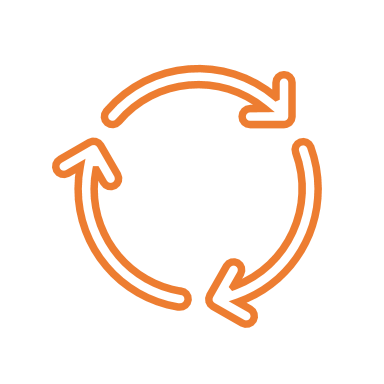
* In case of any failure Rollback the previous release branch version of code to deploy into prod

For Example:

* 1. If there is a failure with Release2.0
  2. Go to release pipeline->Prod Deploy stage
  3. In pre deployment condition artifact filter select the Release1.0 branch and trigger create release to deploy the older version to prod



### Stage 11: Best Practices

**Best Practices

* Push the changes by end of the day to local branch
* Always ensure that developers checkout latest code from feature branch, make code changes and merge the changes back to feature branch
* Don’t check-in directly into develop,Master branch
* Always, Set Code Reviewer while pull request before merge the code into develop or master branch
* Always specify the Code Check-in Comments while check in and add comments while review
* Compare the branch ALWAYS before merging 2 branches
* Release branches are ideal for major releases, consolidated features.   
  Ex: release/OriX, release/endo\_sha
* Create a Pull request from develop branch and merge the code to release branch which is approved to get into production
* Do [Cherry Pick](https://docs.microsoft.com/en-us/azure/devops/repos/git/pull-requests?view=azure-devops#cherry-pick-a-pull-request) a pull request if you want to move selected check ins / features to release branch
* Add Code reviewer and Submit for Code review
* If there are no code changes, merge Code with master branch with proper comments once code is stabilized is prod / verified / tested.
* Every new feature must be created in new branch.
* Rebase develop branch from master after successful prod deployment (Owner : Admin team)

What is NOT ALLOWED?

* CheckIn the code directly into develop/master
* Push code to develop/master without Pull Request
* Merge code to Master without Code review
* Deployment to QA and Prod env without approval in place