

# Continuous Integration and Continuous Delivery with Azure Data Factory and Azure DevOps

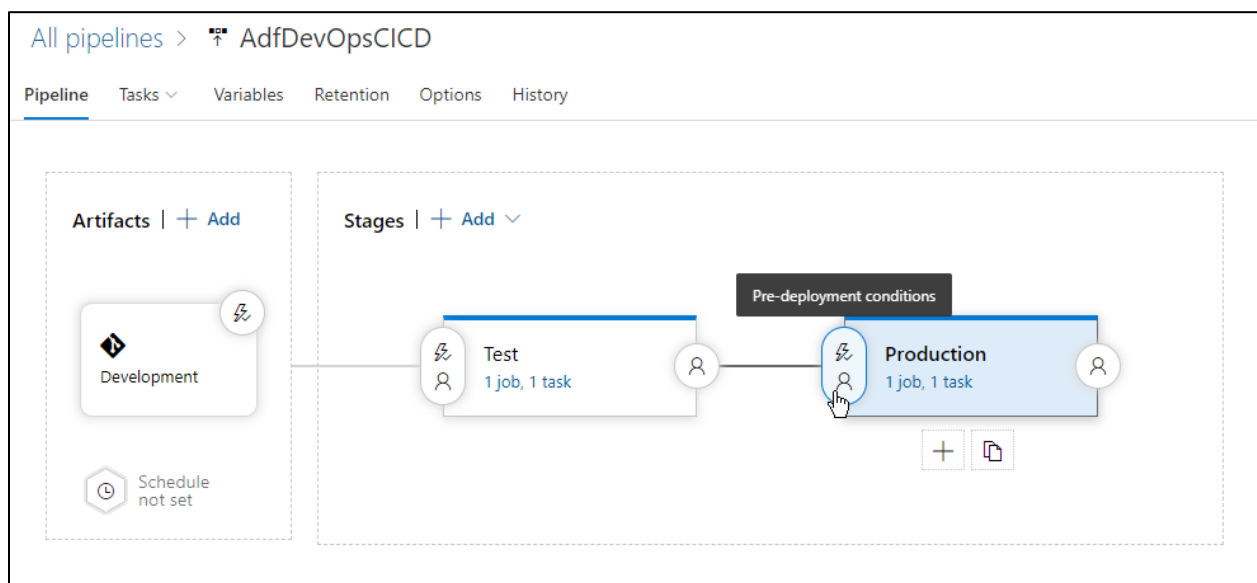
Azure Labs by Roque Daudt (rdaudt@yahoo.com)

## 06 – Continuous Delivery

When we completed the previous lab we had the ARM templates in the **adf\_publish** branch ready to be promoted to higher environments.

In this lab we will build a release pipeline that implements automated delivery from **adf\_publish** to the test and production environments.

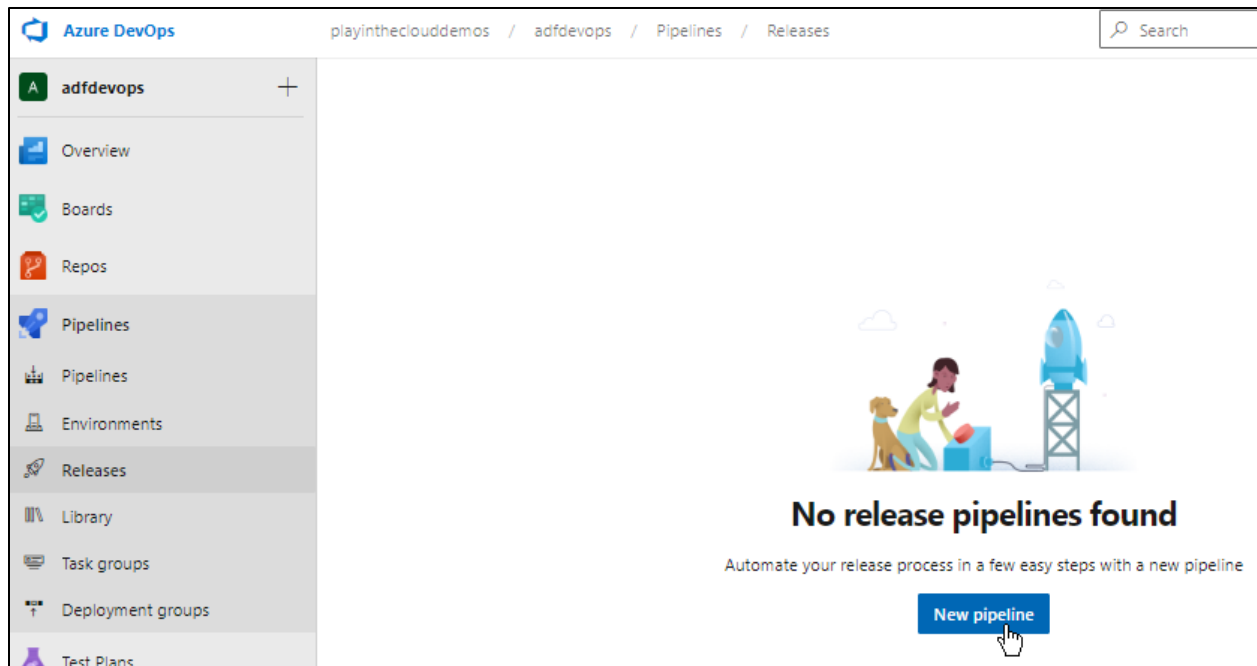
This is a graphical representation of what the pipeline will look like once we are done.



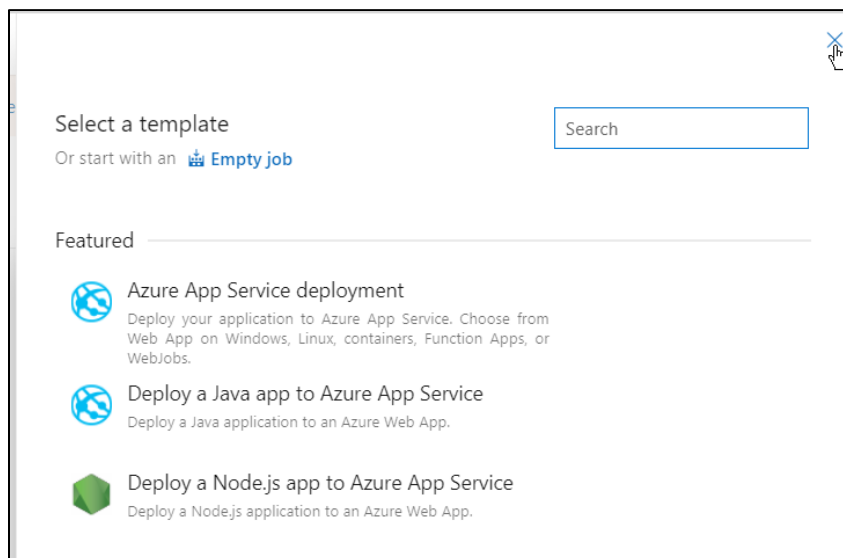
If Azure DevOps is not open now in your browser, make sure to do so now, and let's configure the pipeline.

### Create the pipeline

In Azure DevOps, click **Pipelines / Releases** and then click the **New Pipeline** button.



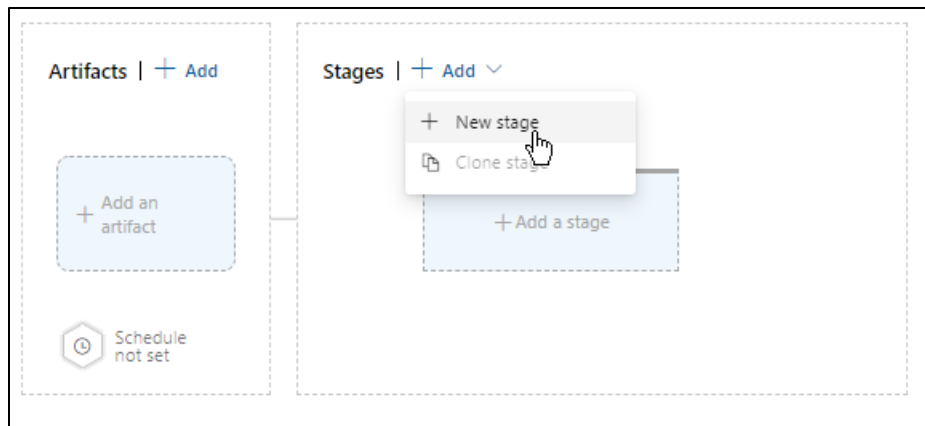
Close the **Select a template** blade open.



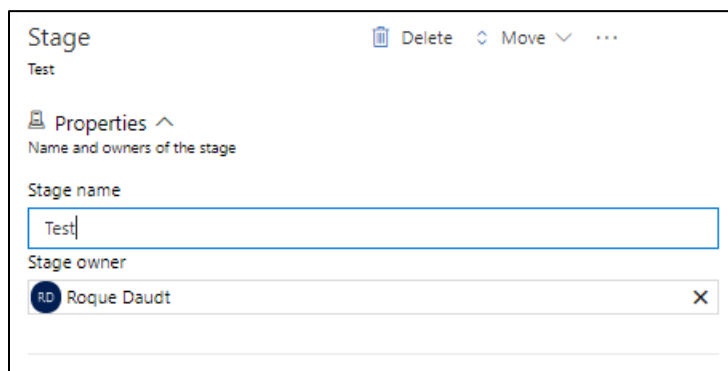
Give the release pipeline better name, if you will.

[All pipelines](#) > **AdfDevOpsCICD**

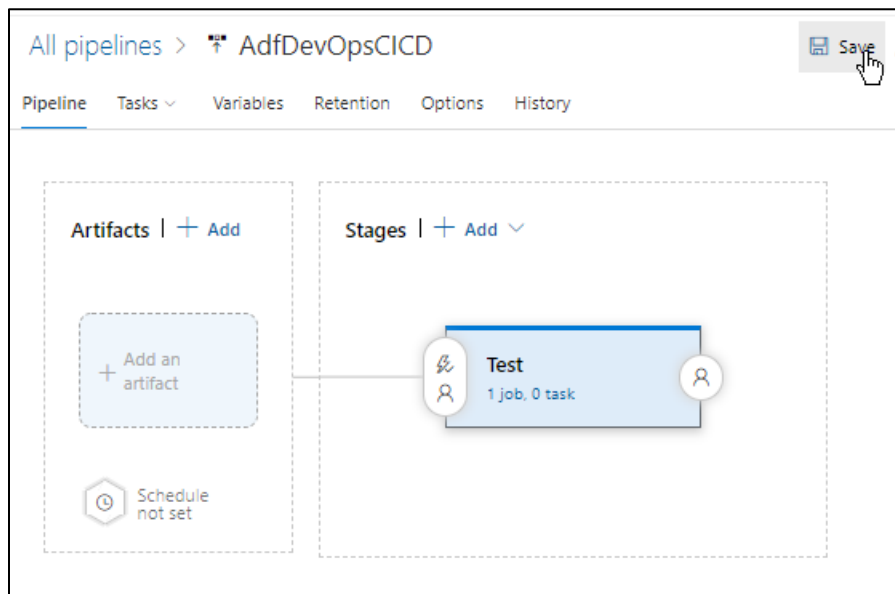
We want the pipeline to first publish to **ADF Tst**. Let's create a new stage then and call it **Test**. Click **New Stage**.



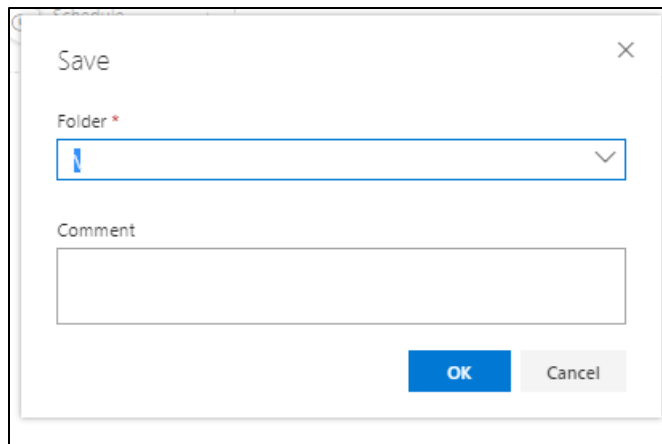
When the **Select a template** blade opens again click **Empty job**. In the next blade, name it **Test**.



Click **Save**.



Click **OK**.



A 'Save' dialog box with a close button (X) in the top right corner. It contains a 'Folder' dropdown menu with a blue arrow icon and a 'Comment' text area. At the bottom are 'OK' and 'Cancel' buttons.

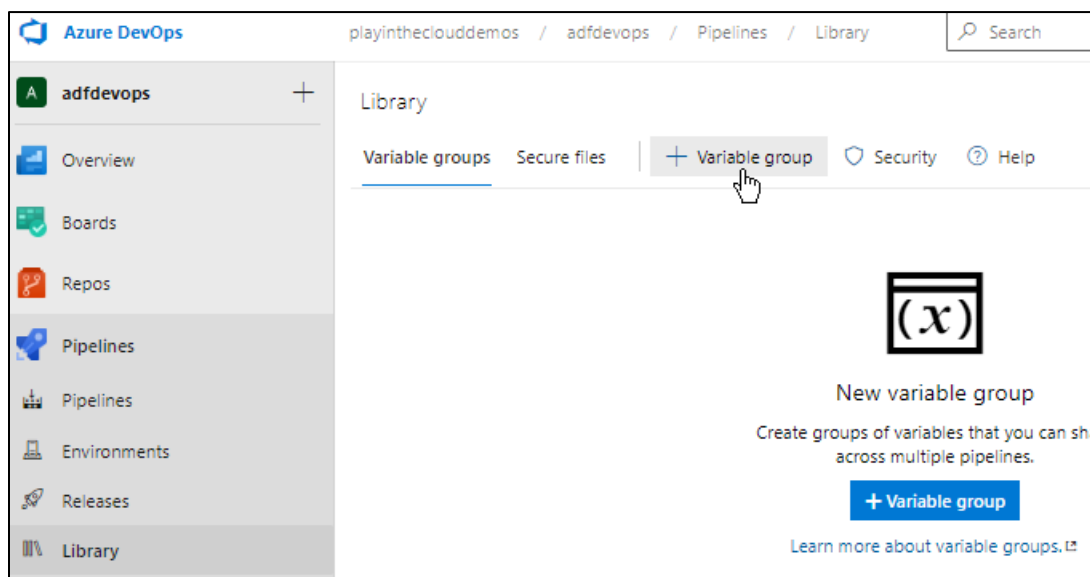
## Configure Group Variables

This DevOps pipeline will promote the ADF pipeline to two different environments: **ADF Tst** and **ADF Prd**. The name of some of the resources used by the ADF pipeline change from environment to environment and we need a strategy to deal with this.

The URI for Azure Key Vault in our development environment, for an instance, is <https://rd2020kvdev.vault.azure.net/>. In test it is <https://rd2020kvtst.vault.azure.net/> and in prod it is <https://rd2020kvprd.vault.azure.net/>. These identifiers need to be set correctly as we promote the ADF artifacts from one environment to another.

One way to do it in Azure DevOps is to rely on the concepts of variables, group variables and linked variables. This process also builds on the fact that we used consistent patterns for the names of the same resources in the different environments.

We are now going to show one way to configure them for our release pipeline. Start by creating variable groups. Click **Library** and then **+ Variable group**.



Name it **TestVarGroup** and click **+ Add** in the **Variables** section.

Library > TestVarGroup\*

Variable group | Save Clone Security Help

### Properties

Variable group name

Description

☒ Allow access to all pipelines

☐ Link secrets from an Azure key vault as variables ⓘ

### Variables

Name ↑	Value
--------	-------

+ Add

Name the variable **Environment**, give it value **"tst"** and click **Save**.

Library > TestVarGroup\*

Variable group | Save Clone Security Help

### Properties

Variable group name

Description

☒ Allow access to all pipelines

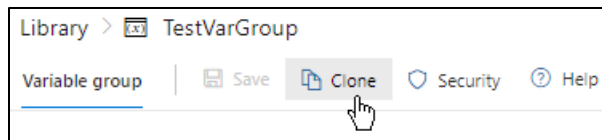
☐ Link secrets from an Azure key vault as variables ⓘ

### Variables

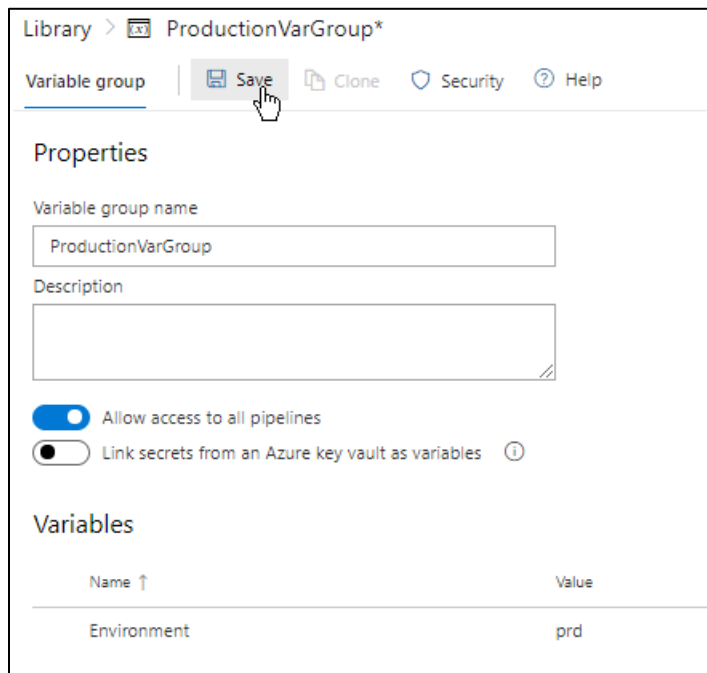
Name ↑	Value
Environment	tst

+ Add

Click **Clone**.

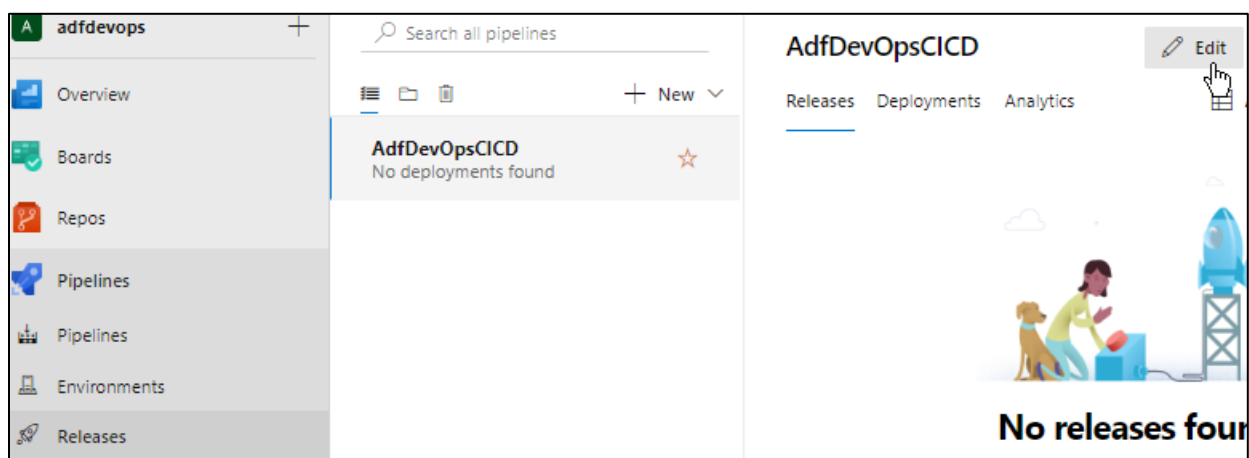


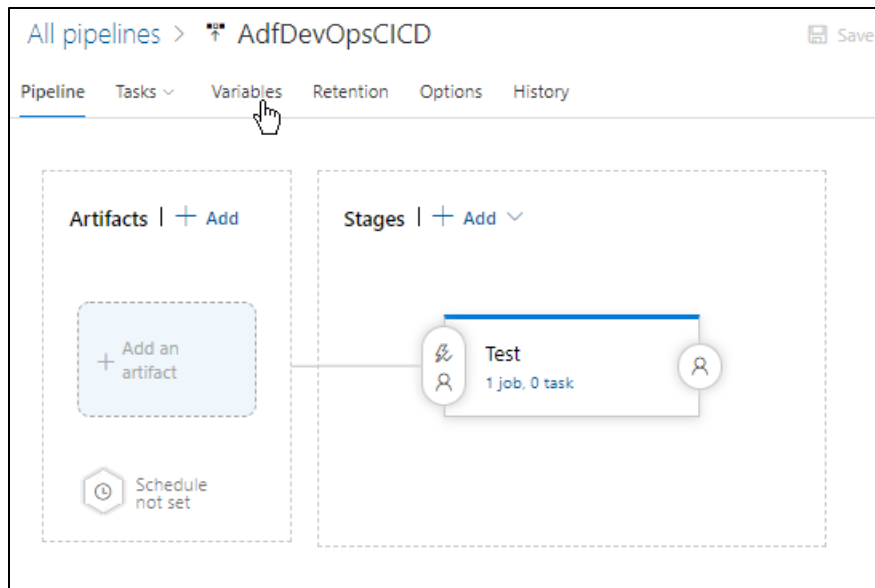
Name the variable group as **ProductionVarGroup**, set value of the Environment variable to “**prd**” and click **Save**.



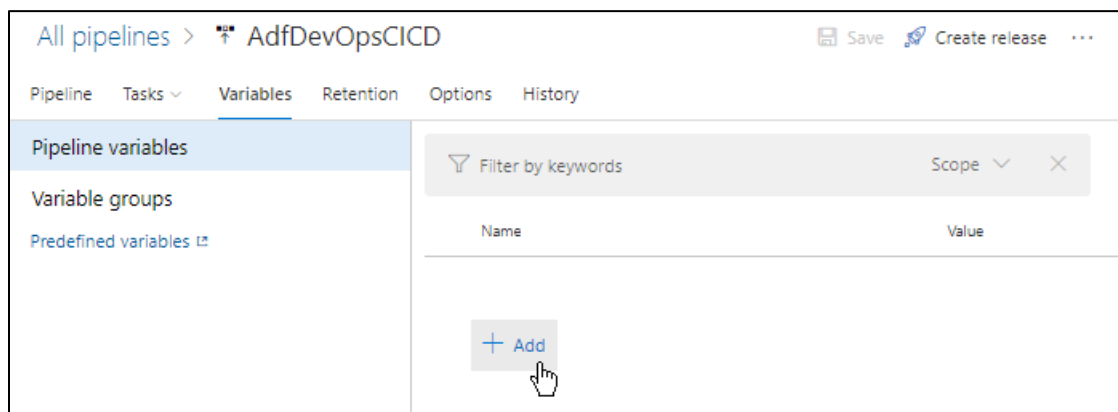
## Configure Release Pipeline Variables

Click **Releases** then click **Edit**.





Make sure that **Pipeline variables** is selected then click **+ Add**.



Create variables with the names shown in the next image.

As per their values, you need to consider the naming convention that you used earlier in Lab 2 to provision the environments. For this demo I used **rd2020** as the prefix for the identifiers. For this reason, I set the value of the **factoryName** variable to **rd2020adf\$(Environment)**.

You entered a different prefix. If you entered a prefix such as **sb202009**, for an instance, then you should set the value of the **factoryName** variable to **sb202009adf\$(Environment)**.

The same applies to the other two variables.

Filter by keywords

Scope

▼

✕

☰

List

Name	Value
factoryName	rd2020adf\$(Environment)
LS_KeyVault_BaseURL	https://rd2020kv\$(Environment).vault.azure.net/
resourceGroup	rd2020rg\$(Environment)

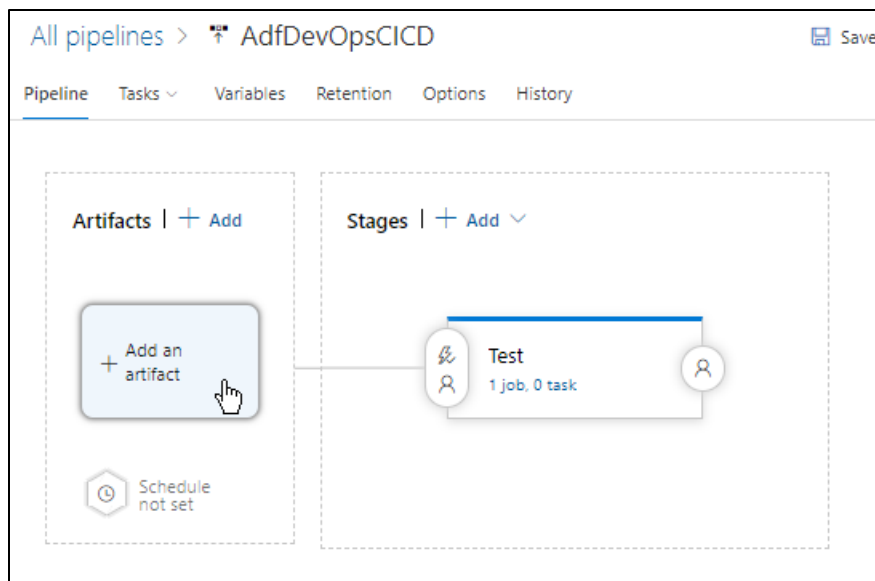
+

Add

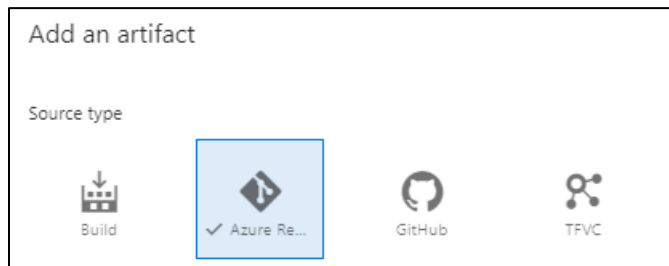
Click **Save**.

## Configure the source for the pipeline

Go back to the **Pipeline** tab and click **+ Add an artifact**.



In the **Add an artifact** blade, select **Azure Repos GIT**.





Fill it up the information as shown below. Basically, we are telling the pipeline that the artifacts should be sourced from the **adf\_publish** branch.





### Add an artifact

Source type

 Build

 ✓ Azure Re...

 GitHub

 TFVC

[5 more artifact types](#) ▾

Project \* ⓘ

adfddevops ▾

Source (repository) \* ⓘ

adfddevops ▾

Default branch \* ⓘ

adf\_publish ▾

Default version \* ⓘ

Latest from the default branch ▾

☐ Checkout submodules ⓘ

☐ Checkout files from LFS ⓘ

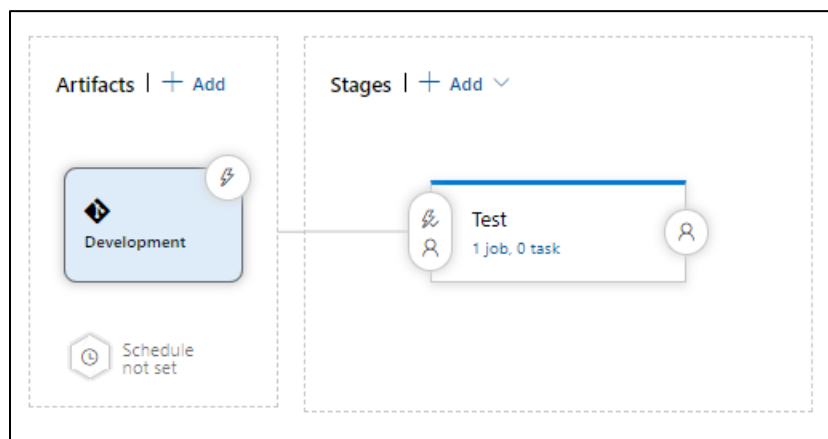
Shallow fetch depth ⓘ

Source alias \* ⓘ

Development

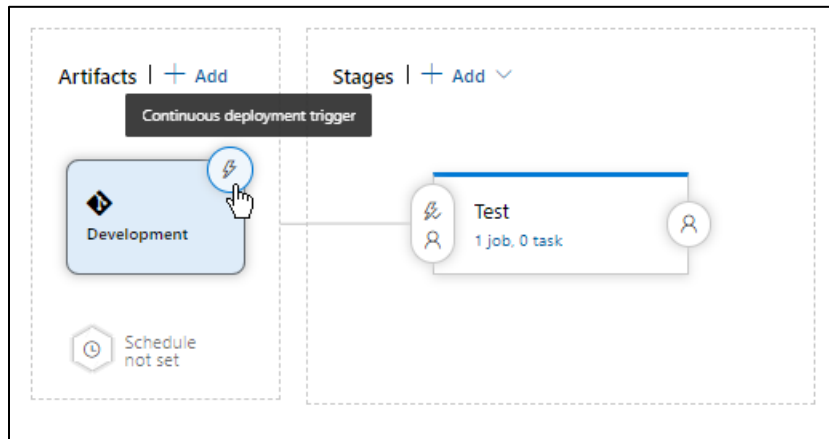
**Add**

Click **Add**. This is what the pipeline looks like now.

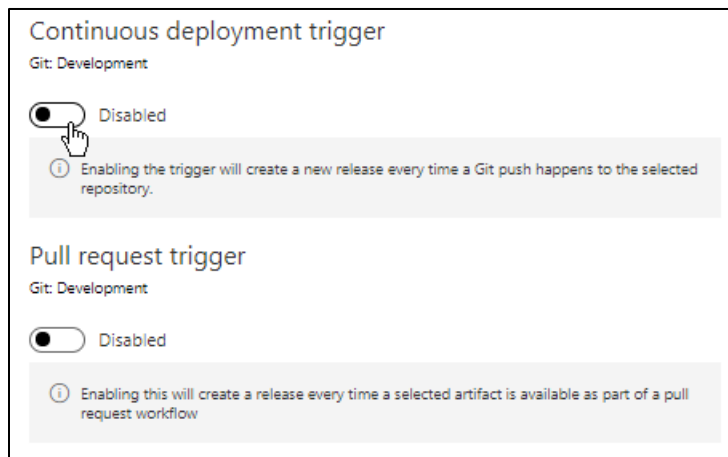


## Configure the Continuous Deployment trigger

We want the release pipeline to fire every time that the **adf\_publish** branch is updated. Click the **Continuous deployment trigger** button.



Enable **Git Development**.



Click **+ Add**.

**Continuous deployment trigger**  
Git: Development

☒ Enabled  
Creates a release every time a Git push occurs in the selected repository.

**Branch filters** ⓘ

No filters added.

[+ Add](#)

**Pull request trigger**  
Git: Development

☐ Disabled

ⓘ Enabling this will create a release every time a selected artifact is available as part of a pull request workflow

Select **adf\_publish**.

**Continuous deployment trigger**  
Git: Development

☒ Enabled  
Creates a release every time a Git push occurs in the selected repository.

**Branch filters** ⓘ

Type	Branch
Include	Select a branch...
ⓘ Specify branch o	<div> <div>Mine All branches</div> <div>Filter my branches</div> <div>adf_publish</div> <div>master</div> </div>

[+ Add](#)

**Pull request**  
Git: Development

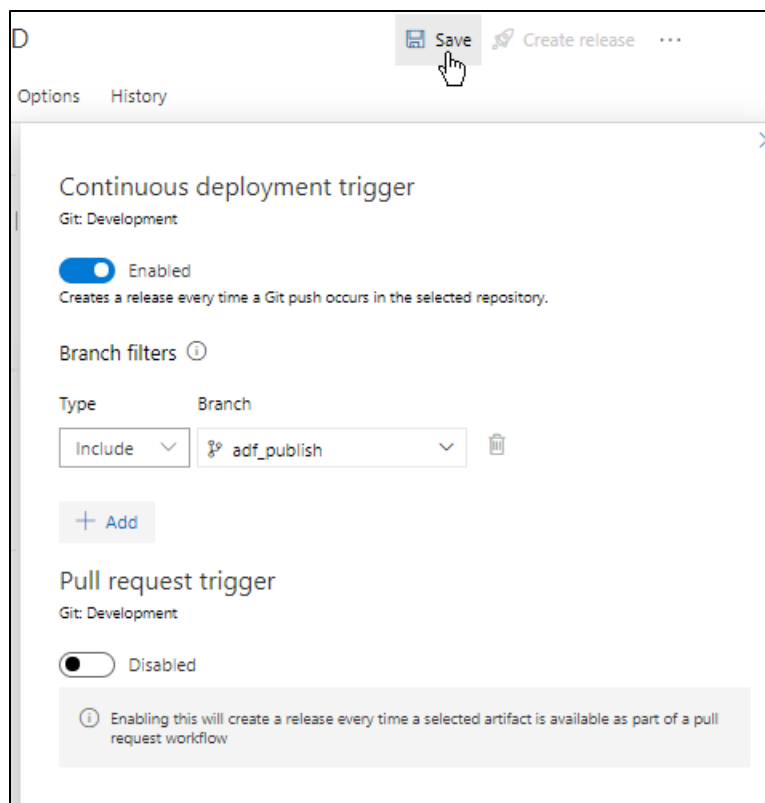
Click **Save**.

**Save** [X]

Comment

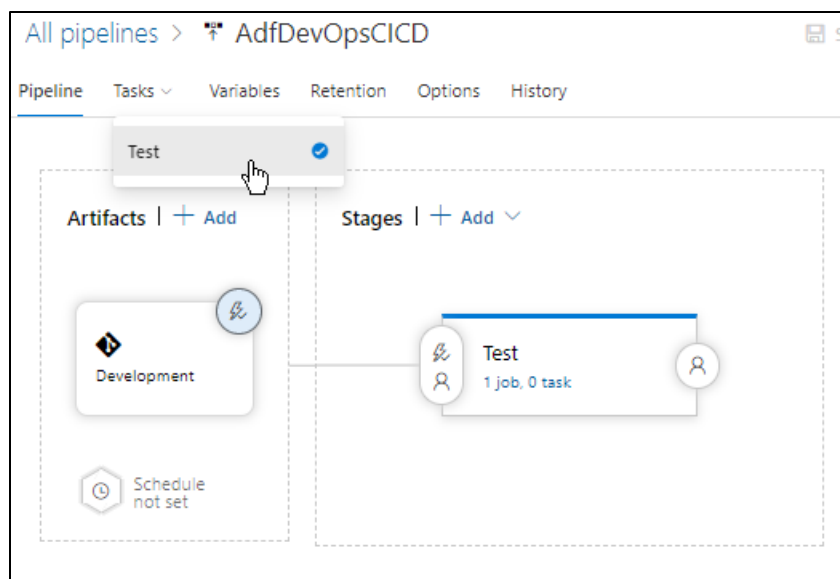
**OK** Cancel

Click **OK** and close the **Continuous deployment trigger** blade.

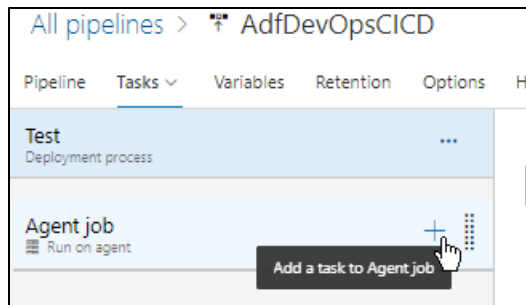


## Configure the Test stage

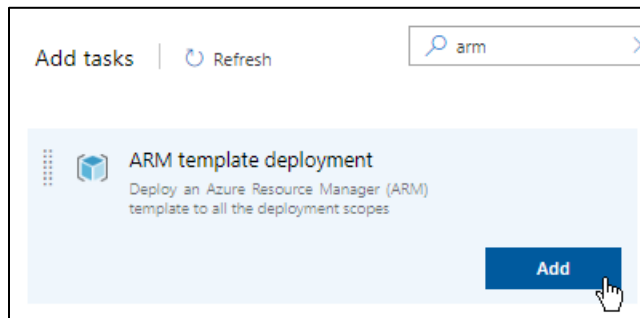
We will now work on the task



Click the + (Add a task to Agent Job) sign shown below.



In the **Add tasks** blade, enter **arm** in the search box and click the **Add** button on the **ARM template deployment** tile.



Back **ARM Template deployment** to configure it.

All pipelines > AdfDevOpsCICD

Save Create release View releases

Pipeline Tasks Variables Retention Options History

Test  
Deployment process

Agent job  
Run on agent

ARM Template deployment: ...  
Some settings need attention

ARM template deployment

Task version 3.\*

Display name \*  
ARM Template deployment: Resource Group scope

Azure Details ^

Deployment scope \*  
Resource Group

Azure Resource Manager connection \*  
This setting is required.

Subscription \*  
This setting is required.

Action \*  
Create or update resource group

Resource group \*  
This setting is required.

Location \*  
This setting is required.

Start entering configuration.

- Leave **Display Name** as it is or change to a more meaningful one
- Leave **Deployment scope** as **Resource Group**
- In the **Azure Resource Manager** connection, select your **Azure subscription**. Once you do that, the **Authorize** button will be enabled. Click it.

ARM template deployment ⓘ

Task version 3,\*

Display name \*

Deploy Data Factory

Azure Details ^

Deployment scope \* ⓘ

Resource Group

Azure Resource Manager connection \* ⓘ | Manage ⚙

Azure subscription 1 [dropdown] Authorize [button] [refresh]

Available Azure subscriptions

Click Authorize to configure an Azure service connection. A new Azure service principal will be created and added to the Contributor role, having access to all resources in the selected subscription. To restrict the scope of the service principal to a specific resource group, see [connect to Microsoft Azure](#) ⚙

Once authorization completes (it takes a few seconds), continue configuration:

- select your subscription again for the **Subscription field**
- select Create or update resource group in the **Action field**
- enter \$(resourceGroup) in the **Resource group field**
- select the location of your choice

Subscription \* ⓘ

Azure subscription 1 [dropdown] [refresh]

Action \* ⓘ

Create or update resource group [dropdown]

Resource group \* ⓘ

\$(resourceGroup) [dropdown] [refresh]

Location \* ⓘ

West US 2 [dropdown] [refresh]

Note that **\$(resourceGroup)** is a reference to the **resourceGroup** pipeline variable that we created earlier in this lab.

Start configuring the **Template** section of the blade. Set **Template location** to **Linked artifact**. Then, click the 3-dot button to the right of the Template control, as shown below.

Template ^

Template location \*

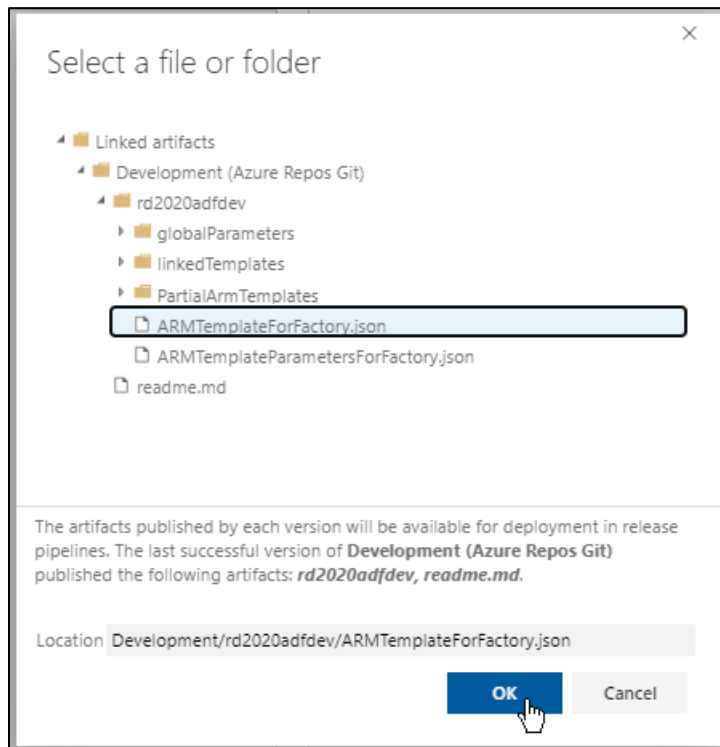
Linked artifact

Template \* ⓘ

Browse Template

This setting is required.

Expand the folders hierarchy, select the **ARMTemplateForFactory.json** file and click **OK**.



Repeat it for the **Template Parameters** field.

Template ^

Template location \*

Linked artifact

Template \* ⓘ

\$(System.DefaultWorkingDirectory)/Development/rd2020adfdev/ARMTemplateForFactory.json

Browse Template

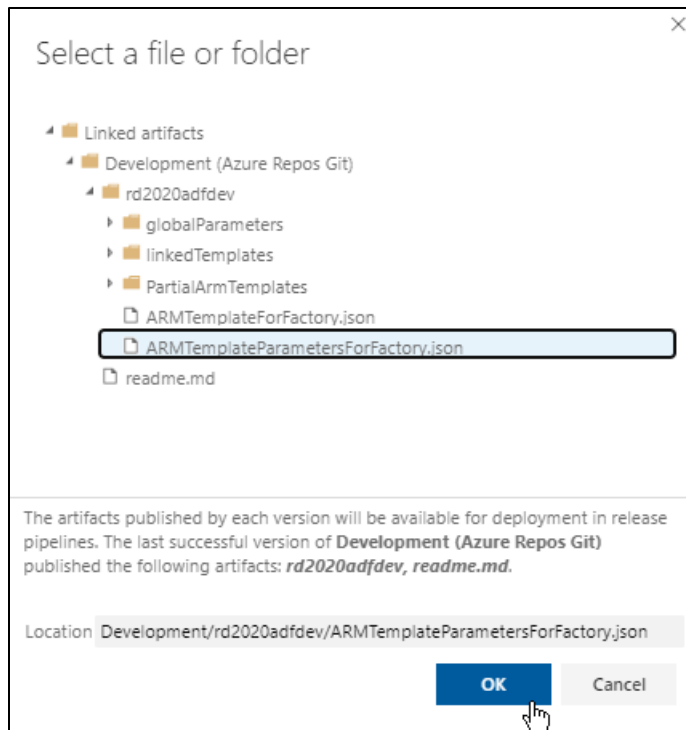
Template parameters ⓘ

Browse Template parameters

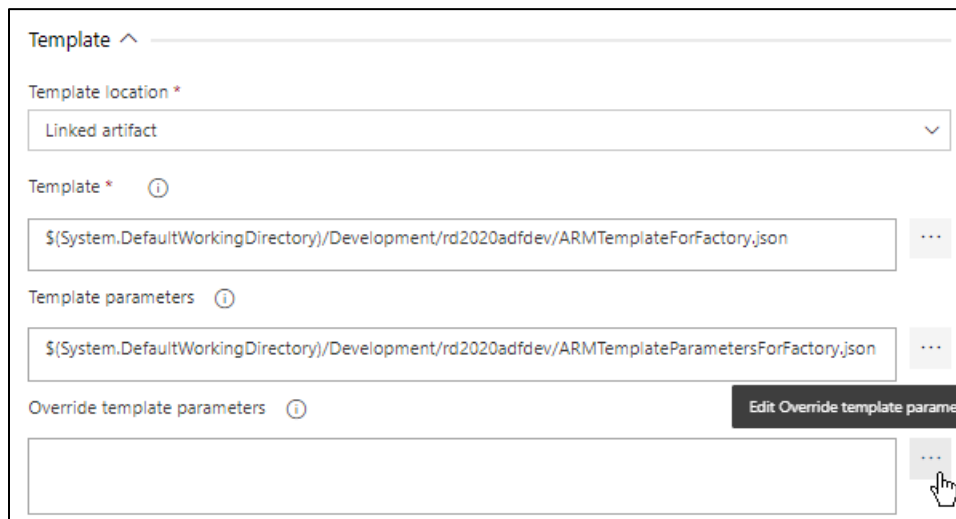
Override template parameters ⓘ

This time select the **ARMTemplateParametersForFactory.json** file. Click **OK**.





In the **Override template parameters** field, click the **Edit Override template parameters** button.



When the **Override template parameters** window opens, we see that it shows the values that currently exist in the **ARMTemplateParametersForFactory.json** file.

Override template parameters

Name	Value
factoryName	"rd2020adfdev"
AzureBlobStorage1_proper...	"connstr"
AzureKeyVault1_properties...	"https://rd2020kvdev.vault...."

OK

Cancel

We have inspected this file in the previous lab. They reflect values that are associated to the development environment because this ARM template was created out of publishing from **master** (DEV) to **adf\_publishing**. I reproduce the file below for your convenience.

adfdevops

rd2020adfdev

globalParameters

rd2020adfdev\_GlobalParameters.json

linkedTemplates

ArmTemplate\_0.json

ArmTemplate\_master.json

ArmTemplateParameters\_master.json

PartialArmTemplates

ArmTemplate\_9\_29\_2020\_10\_12\_20.json

ArmTemplateForFactory.json

ArmTemplateParametersForFactory.json

readme.md

adf\_publish

rd2020adfdev

ARMTemplateParametersForFactory.json

ARMTemplateParametersForFactory.json

Contents

History

Compare

Blame

You updated adf\_publish 19m ago

Create a pull request

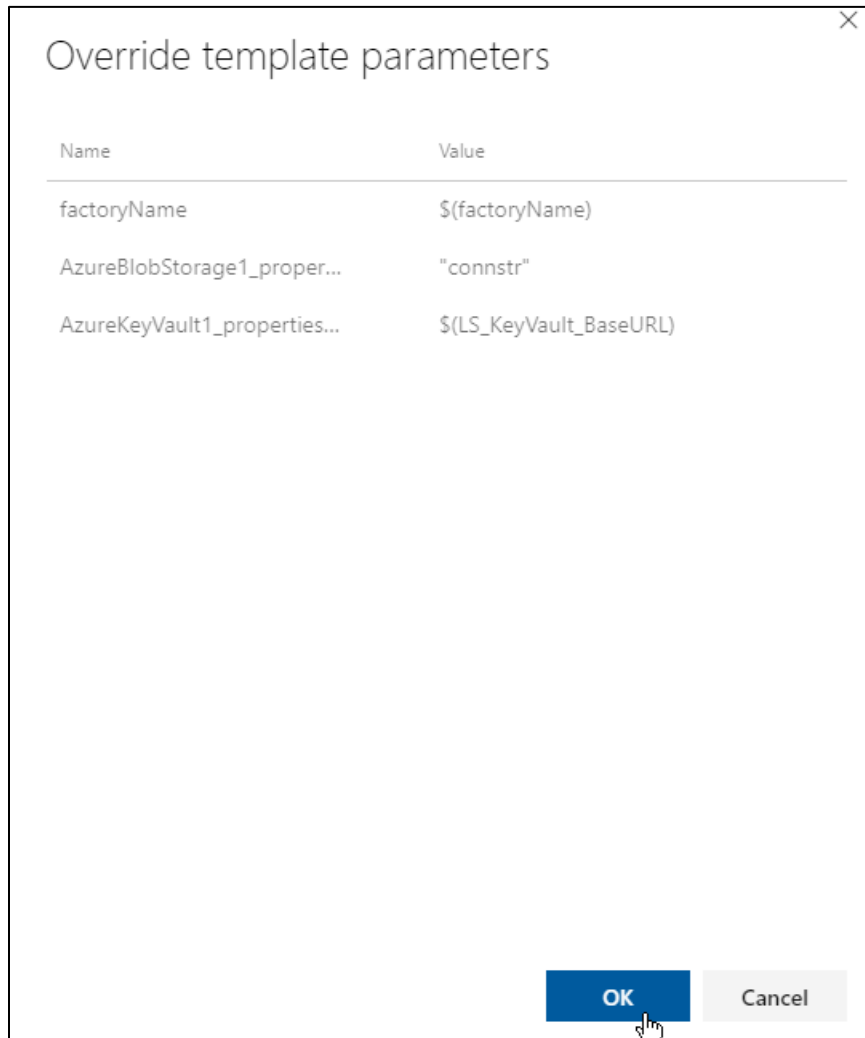
```

1  {
2    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentParameters.json#",
3    "contentVersion": "1.0.0.0",
4    "parameters": {
5      "factoryName": {
6        "value": "rd2020adfdev"
7      },
8      "AzureBlobStorage1_properties_typeProperties_connectionString_secretName": {
9        "value": "connstr"
10     },
11     "AzureKeyVault1_properties_typeProperties_baseUrl": {
12       "value": "https://rd2020kvdev.vault.azure.net/"
13     }
14   }
15 }

```

What we want then is to use variables to configure this action, make it generic and suitable for deployment to test and production. For this reason, we will update the value of two of the parameters in this window, as shown below.

- **factoryName** is set to **\$(factoryName)**
- **AzureKeyVault1\_properties\_typeProperties\_baseUrl** is set to **\$(LS\_KeyVault\_BaseURL)**



Name	Value
factoryName	\$(factoryName)
AzureBlobStorage1_proper...	"connstr"
AzureKeyVault1_properties...	\$(LS_KeyVault_BaseURL)

Click **Ok**. Leave the other fields as they are and click **Save**.

All pipelines > AdfDevOpsCICD

Save Create release ...

Pipeline Tasks Variables Retention Options History

**Test**  
Deployment process

**Agent job**  
Run on agent

**Deploy Data Factory**  
ARM template deployment

\$(resourceGroup)

Location \*  
West US 2

Template ^

Template location \*  
Linked artifact

Template \*  
\$(System.DefaultWorkingDirectory)/Development/rd2020adfdev/ARMTemplateForFactory.json

Template parameters  
\$(System.DefaultWorkingDirectory)/Development/rd2020adfdev/ARMTemplateParametersForFactory.json

Override template parameters  
-factoryName \$(factoryName) -  
AzureBlobStorage1\_properties\_typeProperties\_connectionString\_secretName "connstr" -  
AzureKeyVault1\_properties\_typeProperties\_baseUri \$(LS\_KeyVault\_BaseURL)

Click **OK**.

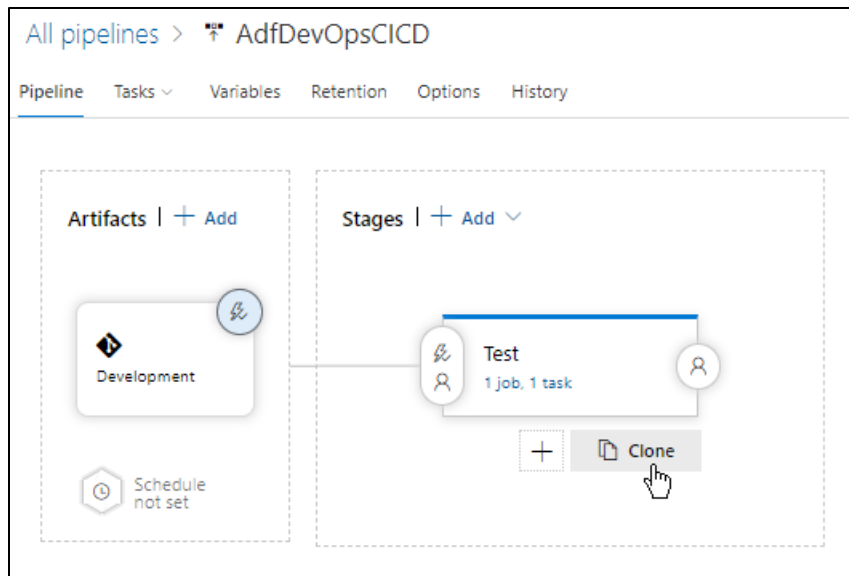
Save

Comment

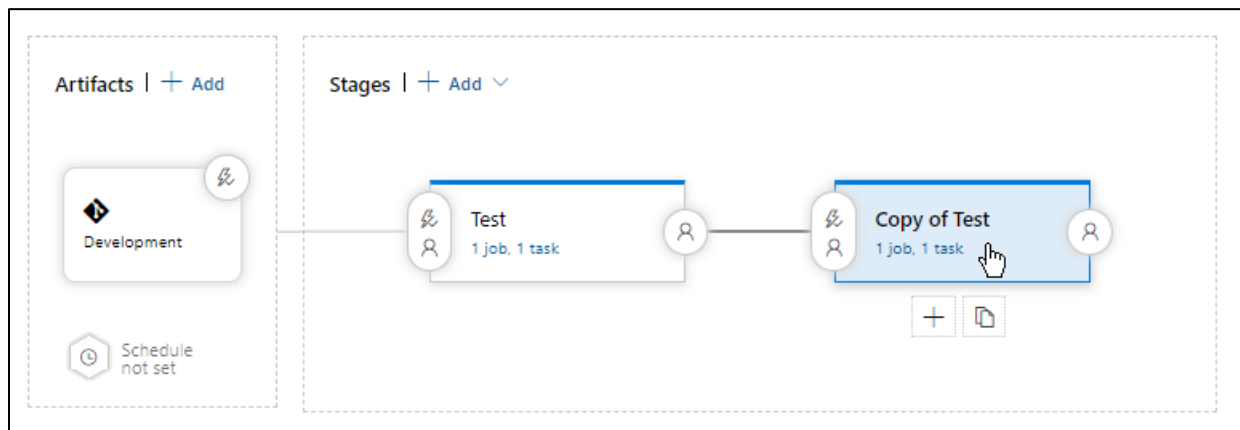
OK Cancel

## Configure the Production stage

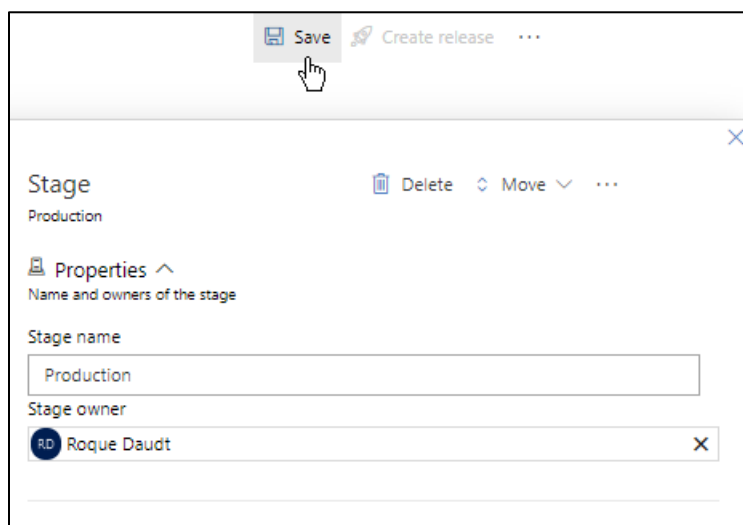
Back to the **Pipeline** tab, click **Clone** as shown below.



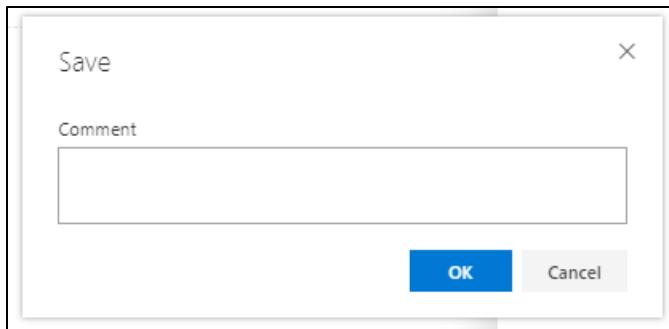
Click the stage box and



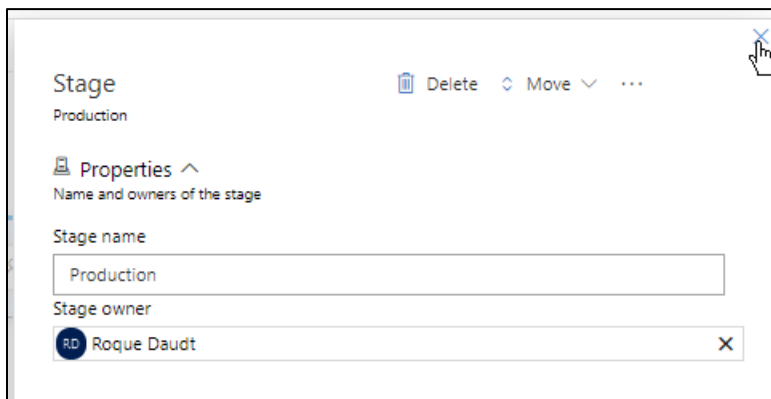
Set **Stage Name** as **Production** and click **Save**.



Click **OK**.

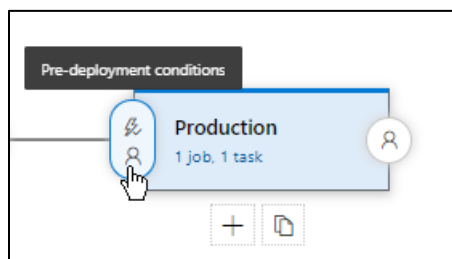
A 'Save' dialog box with a close button (X) in the top right corner. It contains a 'Comment' label above a text input field. At the bottom, there are two buttons: 'OK' (blue) and 'Cancel' (grey).

Close the blade.

A 'Stage' blade titled 'Production'. It has a toolbar with 'Delete', 'Move', and a menu icon. Below the title is a 'Properties' section with the subtitle 'Name and owners of the stage'. It contains a 'Stage name' field with 'Production' and a 'Stage owner' field with 'RD Roque Daudt'. A close button (X) is in the top right corner.

We want now to add pre-deployment approval for production. Once this implemented, someone will need to manually approve the deployment for it to happen.

To do so, we click **Pre-deployment conditions**.



When the **Pre-deployment conditions** blade is shown, enable **Pre-deployment approvals**, as shown below.

## Pre-deployment conditions

Production

### Triggers ^

Define the trigger that will start deployment to this stage

Select trigger ⓘ

After release

After stage

Manual only

### Stages ⓘ

Test

☐ Trigger even when the selected stages partially succeed ⓘ

Artifact filters ⓘ
Disabled

Schedule ⓘ
Disabled

Pull request deployment ⓘ
Disabled

Pre-deployment approvals
Disabled

Select the users who can approve or reject deployments to this stage

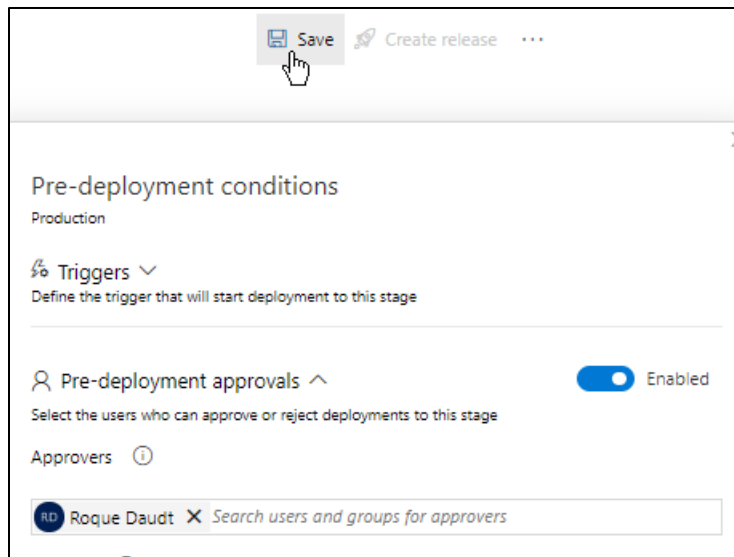
Gates
Disabled

Define gates to evaluate before the deployment.  
[Learn more](#)

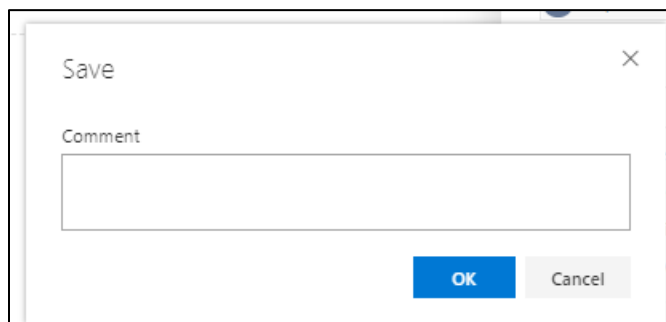
Deployment queue settings v

Define behavior when multiple releases are queued for deployment

Enter the user that should approve deployment (look for yourself) in the **Approvers** control. Click **Save**.



Click **OK**.



Close the **Pre-deployment** conditions blade.

## Linked variables

At this time, both stages are configured to use the content of the pipeline variables for the deployments. However, the content of these variables must change accordingly, depending on which environment its been deployed at each stage. There is a need them for a mechanism to ensure that it happens. This is the role of the **Linked Variable Groups**.

Click the **Variables** tab, select **Variable groups** and click the **Link variable group** button.





In the Link variable group blade, select **ProductionVarGroup (1)**, set **Variable group scope** to **Stages**, select the **Production** stage and click **Link**.

The screenshot shows a 'Link variable group' blade. At the top, there is a title bar with the text 'Link variable group' and an information icon. To the right of the title bar is a search bar with a magnifying glass icon and the word 'Search'. Below the title bar is a list of variable groups. The first item, 'ProductionVarGroup (1)', is selected and highlighted with a blue background. Below it is 'TestVarGroup (1)'. At the bottom of the blade, there is a section titled 'Variable group scope'. It contains two radio buttons: 'Release' (unselected) and 'Stages' (selected). Below the radio buttons is a dropdown menu with 'Production' selected. At the very bottom of the blade is a blue button labeled 'Link', with a mouse cursor pointing at it.

Click the **Link variable group** button again.

All pipelines > AdfDevOpsCICD Save

Pipeline Tasks Variables Retention Options History

Pipeline variables

Variable groups

Predefined variables [?](#)

Name	Value
ProductionVarGroup (1) <span>Scopes: Production</span>	
<a href="#">Link variable group</a>   <a href="#">Manage variable groups</a> <a href="#">?</a>	

Enter the configuration shown below and click **Link**.

Link variable group ⓘ

Search

✓ TestVarGroup (1)

Variable group scope

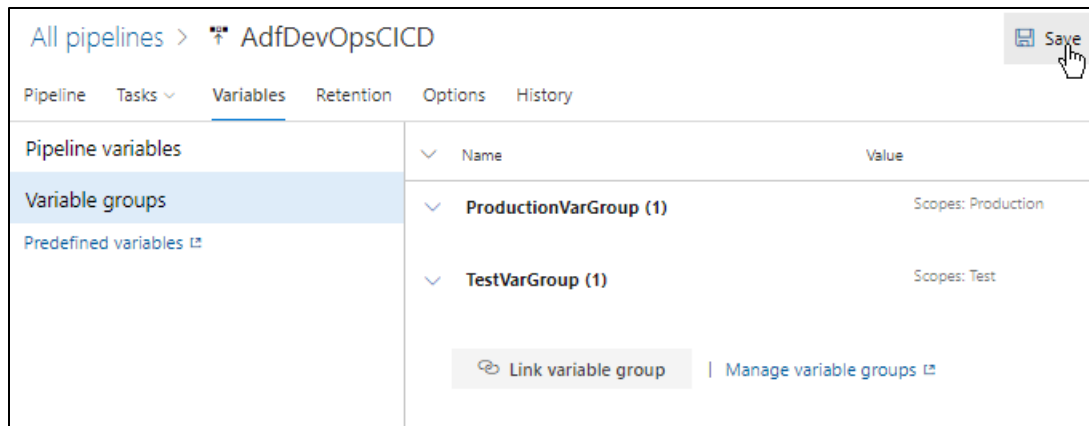
☐ Release

☒ Stages

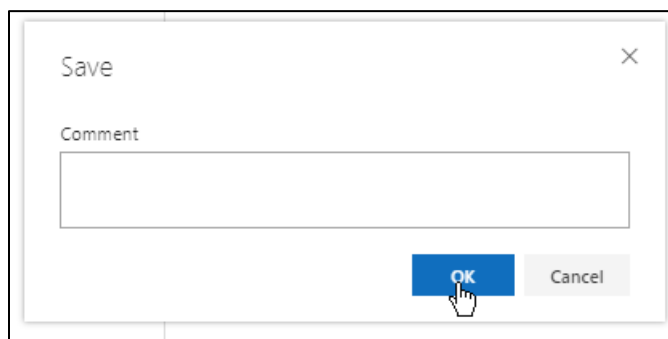
✓ Test ▼

Link

Save the work.



Click **OK**.

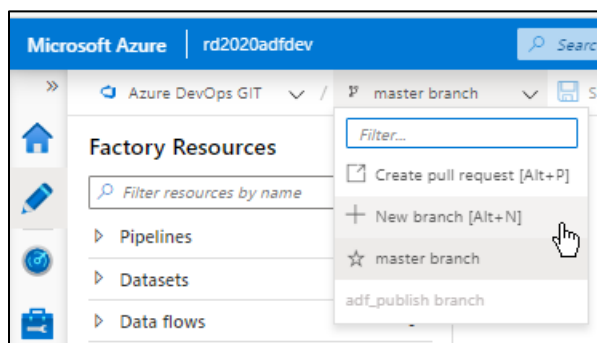


We have completed the pipeline's configuration. Let's run it now.

## Test the release pipeline

To verify that the process is working from end-to-end, we will apply another minor change to the pipeline in ADF Dev. Once we publish it to **adf\_publish**, it should trigger the release pipeline.

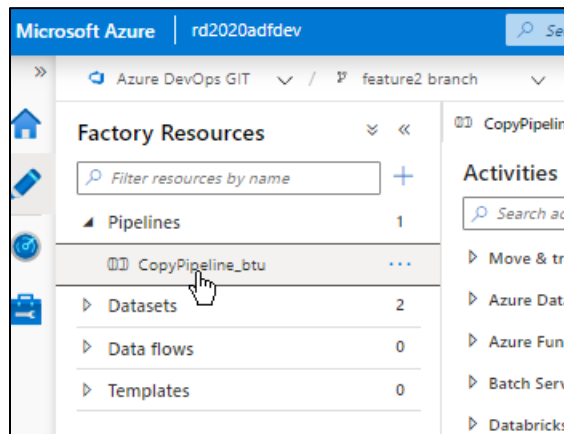
Go to **ADF Dev**, make sure that you are in the **master** branch and create a new feature branch, name it **feature2**.



New branch [Alt+N]

feature2

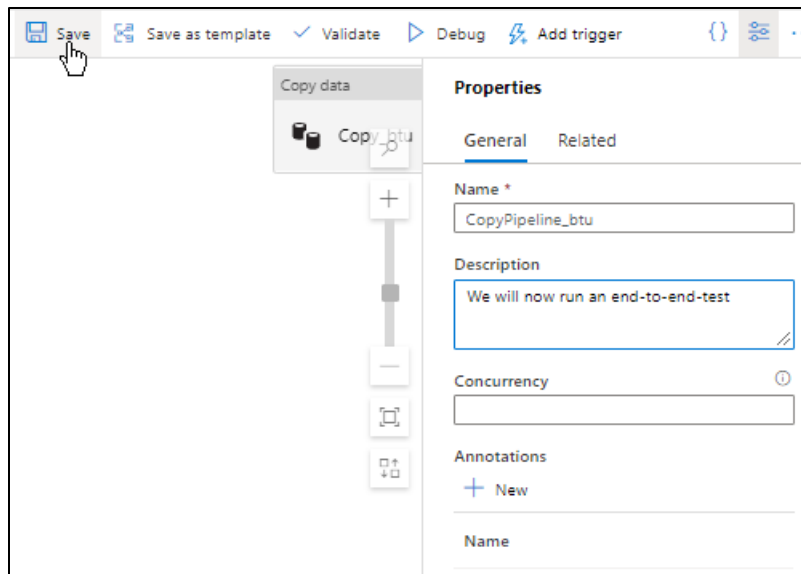
Open pipeline **CopyPipeline\_btu**.



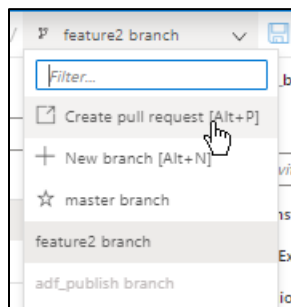
Click the **properties** button.

The screenshot shows the 'Properties' dialog box for the 'CopyPipeline\_btu' pipeline. The 'General' tab is selected. The 'Name' field contains 'CopyPipeline\_btu'. The 'Description' field contains 'Copy file from one container to another. This is the only thing that it does.' The 'Concurrency' field is empty. The 'Annotations' section has a '+ New' button and a 'Name' field.

Change the description to anything else and click **Save**.



Create a new pull request.



Give it a better name and click **Create**.

The screenshot shows the 'New pull request' interface in Azure DevOps. The left sidebar contains navigation links: Overview, Boards, Repos (selected), Files, Commits, Pushes, Branches, Tags, Pull requests, Pipelines, Test Plans, and Artifacts. The main area is titled 'New pull request' and shows the source 'feature2' and target 'master'. The 'Overview' tab is active, displaying fields for Title ('Updating pipeline: end-to-end test') and Description ('Updating pipeline: CopyPipeline\_btu'). Below these are sections for Reviewers, Work items to link, and Tags. A 'Create' button is at the bottom right.

Approve request.



Click **Complete**.



Click **Complete merge**.


Complete pull request

×

Merge type

Merge (no fast forward)

▼



Post-completion options

☒ Complete associated work items after merging

☒ Delete feature2 after merging

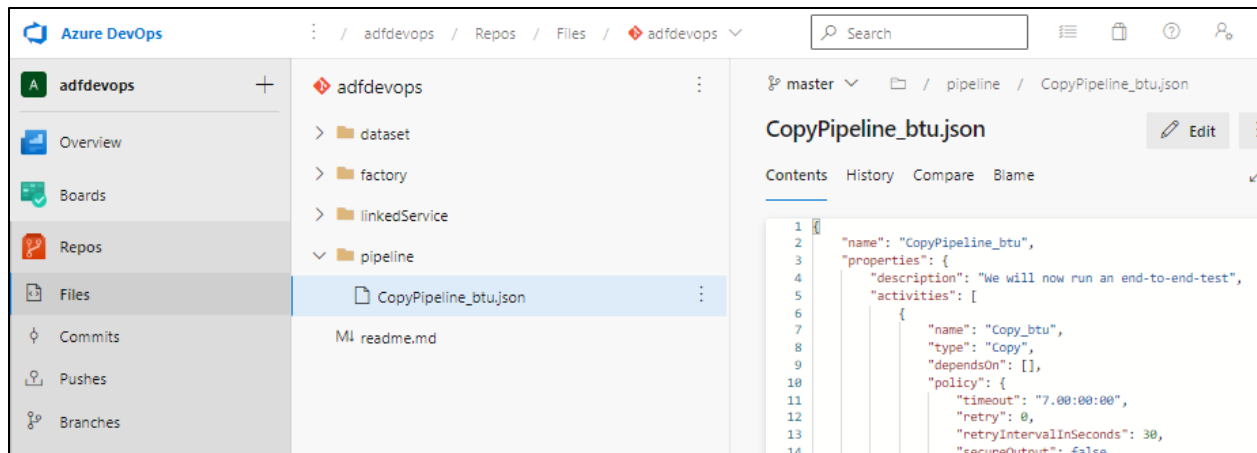
☐ Customize merge commit message

Cancel

Complete merge

In the **master** branch, check the **CopyPipeline\_btu.json** file, verify that the description has been updated.





Go back to **ADF Dev**, indicate that you want to use master.

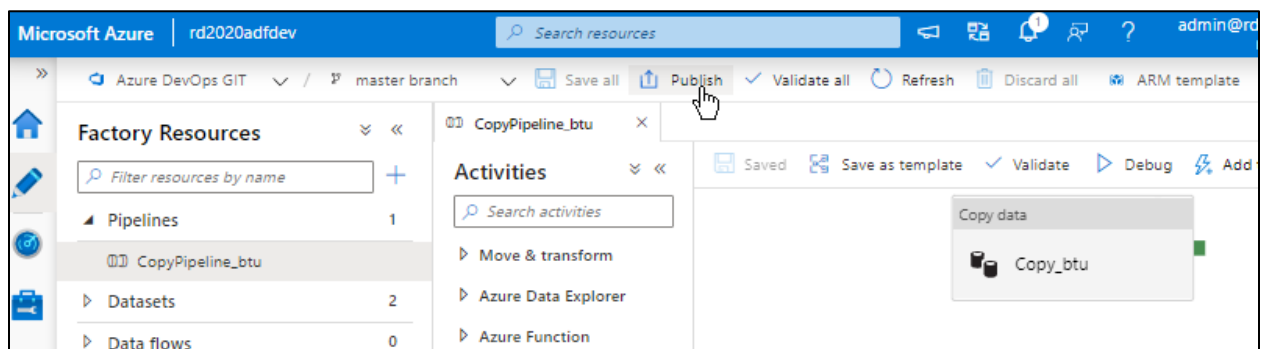
**Select working branch**

Current working branch 'feature2' was deleted, create a new branch.

Working branch ☐ Create new ☒ Use existing

☆ master

Click **Publish**.



Click **OK**.

## Pending changes

Publish branch

The Publish branch is the branch in your repository where publishing related ARM templates are stored and updated.



### ▲ Data Factory properties

### ▲ Pipelines

CopyPipeline\_btu (Edited) CopyPipeline\_btu

### ▲ Datasets

### ▲ Data flows

### ▲ Integration runtimes

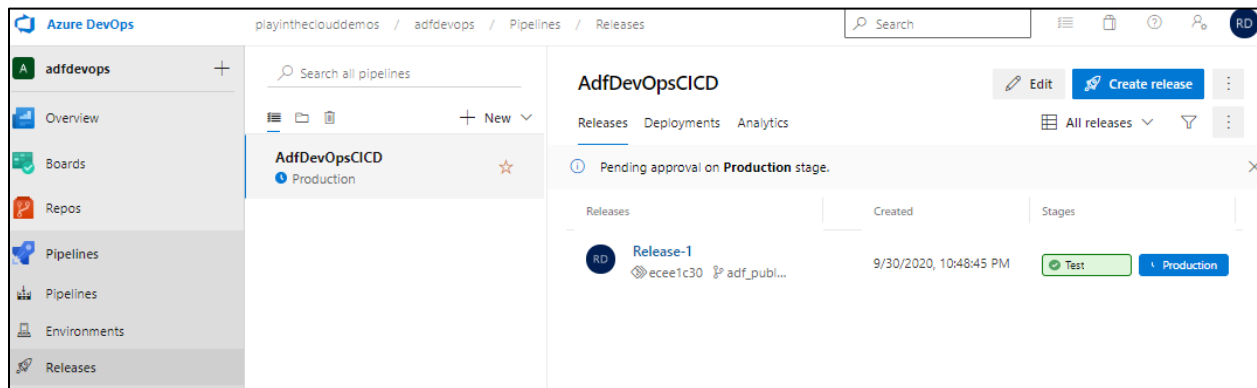
### ▲ Linked services

### ▲ Triggers

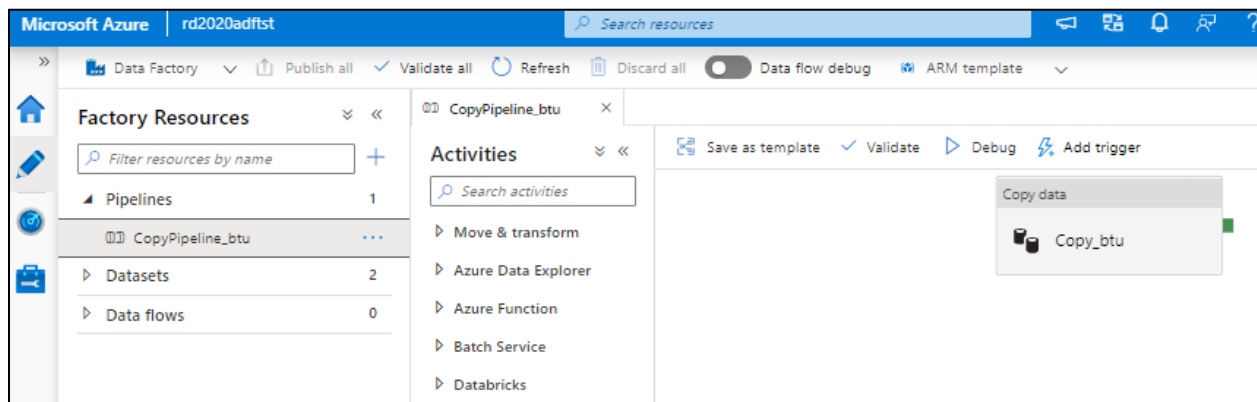
OK

Cancel

After a minute or so, go to Azure DevOps, select **Pipelines / Releases**. Check the pipeline. Note that it succeeded in Test and it is waiting for approval for Production.



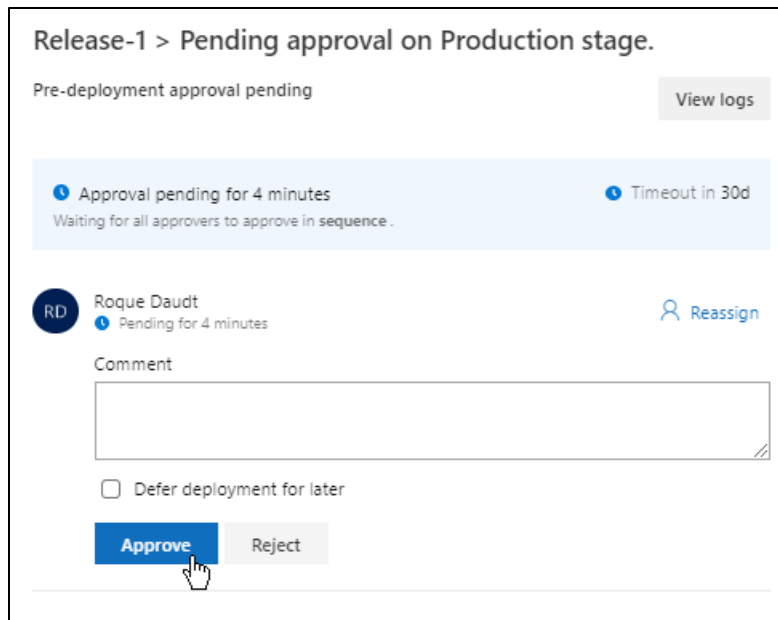
Open **ADF Tst** and confirm that the ADF pipeline has been created.



Go back to Azure DevOps. Click **Production**.



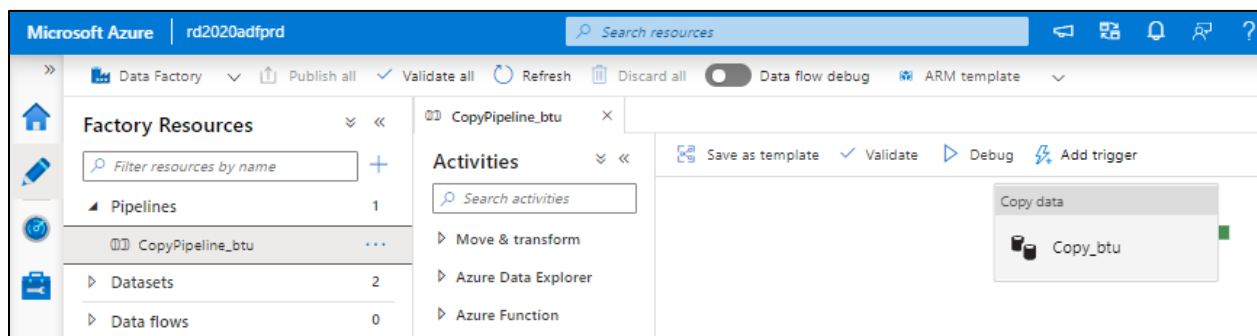
Click **Approve** and close the blade.



Wait a bit more and re-check the status of the release pipeline run. Verify that it is successful (tile is green)



Open **ADF Prd** and confirm that the pipeline has been created there.

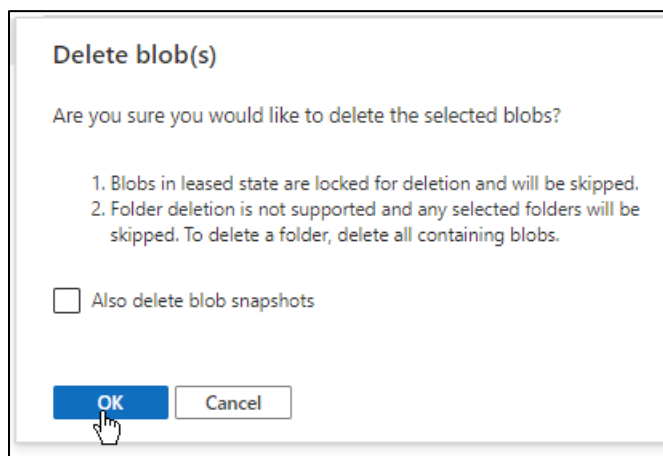
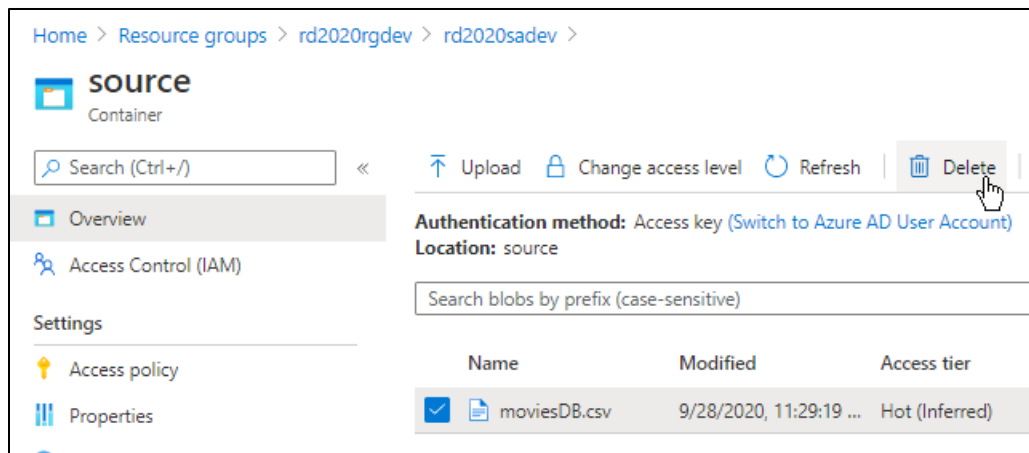


## Test the pipeline in ADF Test

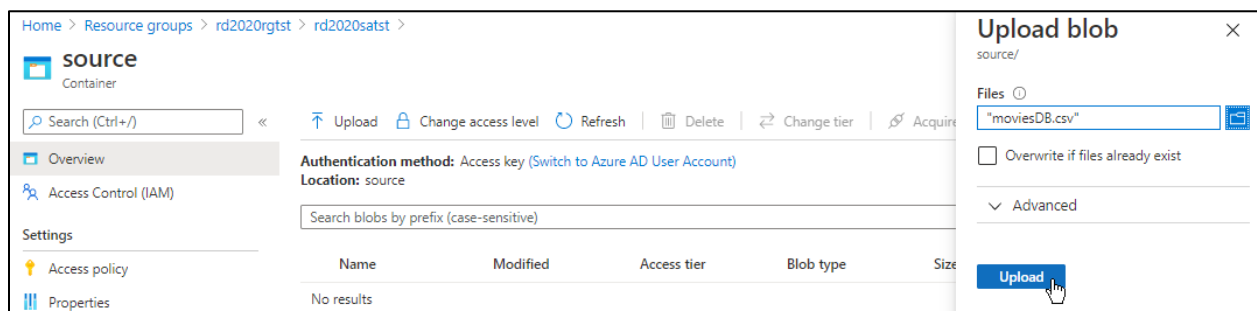
Wrapping up this lab, we want to run the ADF pipeline in Test, ensure that it works the same way as it does in ADF Dev. In order to prepare the test we will do two things

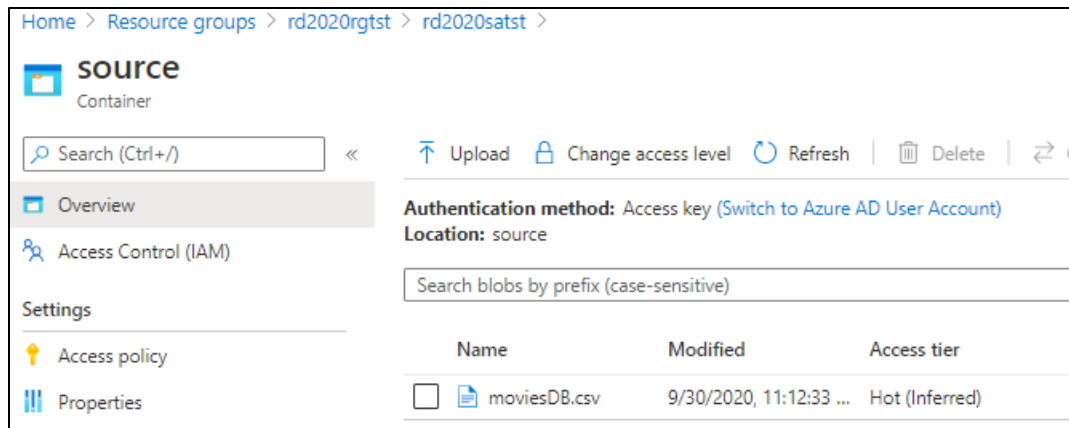
- Upload **movieDB.csv** file to the storage account in Test
- Remove **movieDB.csv** file from the storage account in Dev, to make sure that each environment is using its own resources.

Open the storage account in dev, open the default container, go to the source folder and delete the file.

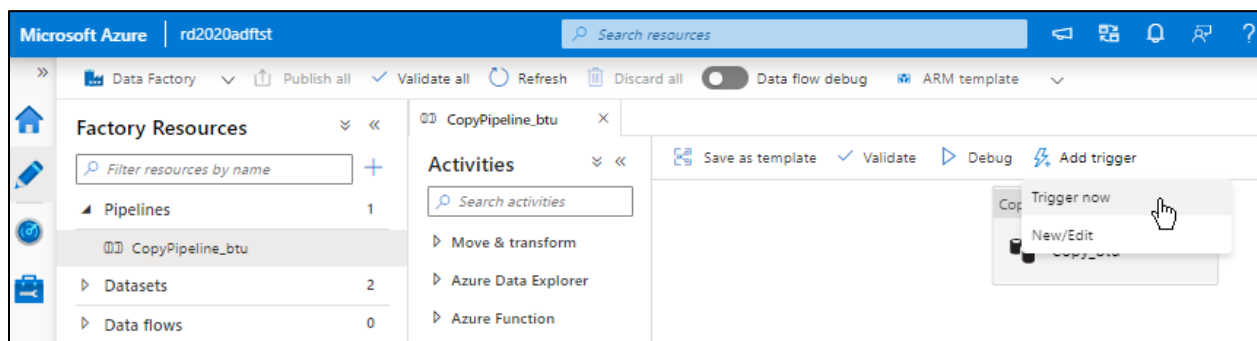


Go to the storage account in test and upload the file.

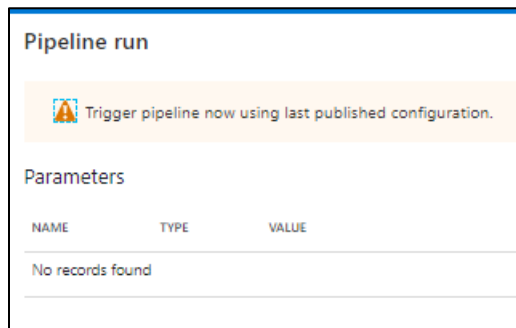




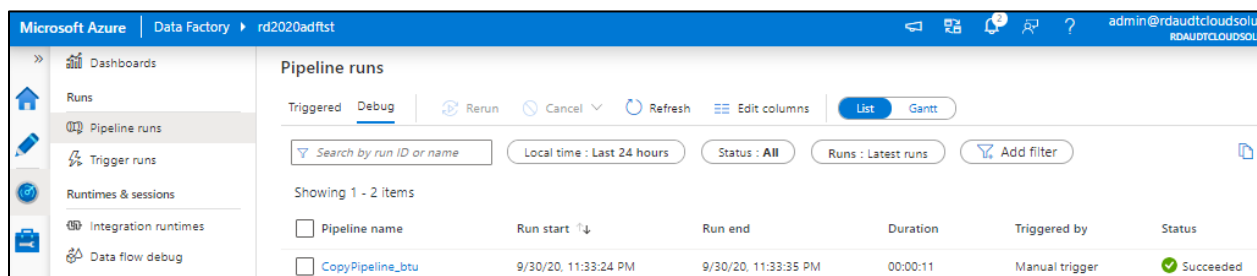
Go to **ADF Test**, open the pipeline and click **Trigger now**.



Click **OK** in the **Pipeline run** blade.




Monitor the progress, verify that it succeeded.



Check the target folder in the test storage account. Verify that the file was successfully copied.

Home > Resource groups > rd2020rgtst > rd2020satst >

 **target**  
Container


« [Upload](#) [Change access level](#) [Refresh](#)

[Overview](#)  
[Access Control \(IAM\)](#)

Settings

- [Access policy](#)
- [Properties](#)
- [Metadata](#)

Authentication method: Access key ([Switch to Azure AD U](#))  
Location: target

	Name	Modified
<input type="checkbox"/>	 moviesDB.csv	9/30/2020, 11:33:33 ...

## Next

This lab concludes the review of Microsoft’s “official, or “classic” way of deployment. Next, we will explore an alternative approach: deployment from the json files, instead of relying on ARM templates.

Go to **07 – Deployment from JSON files**.