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

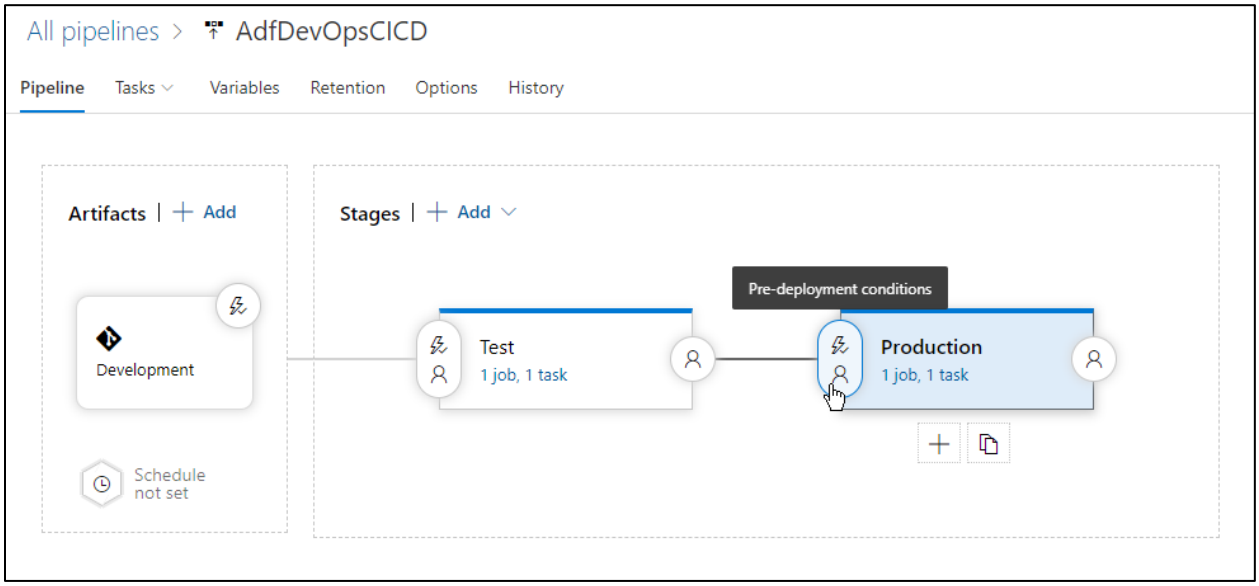
Azure Labs by Roque Daudt (rdaudt@yahoo.com)

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.

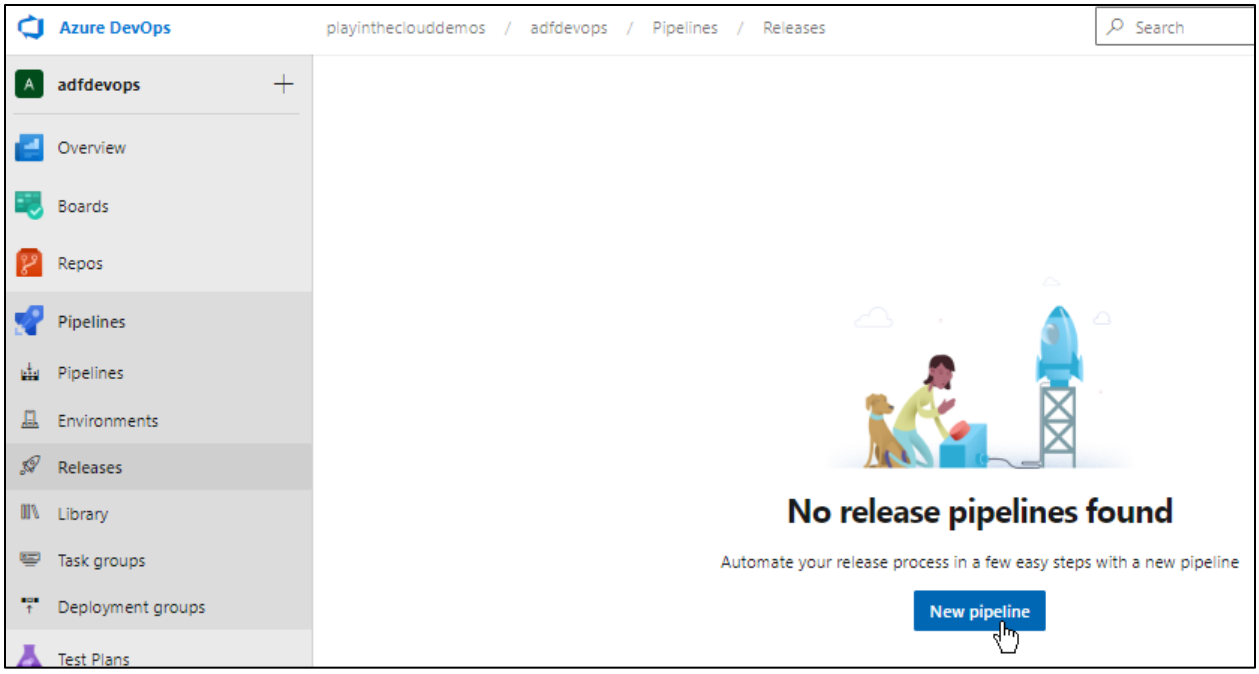
This is a graphical representation of what the pipeline will look like once we are done.



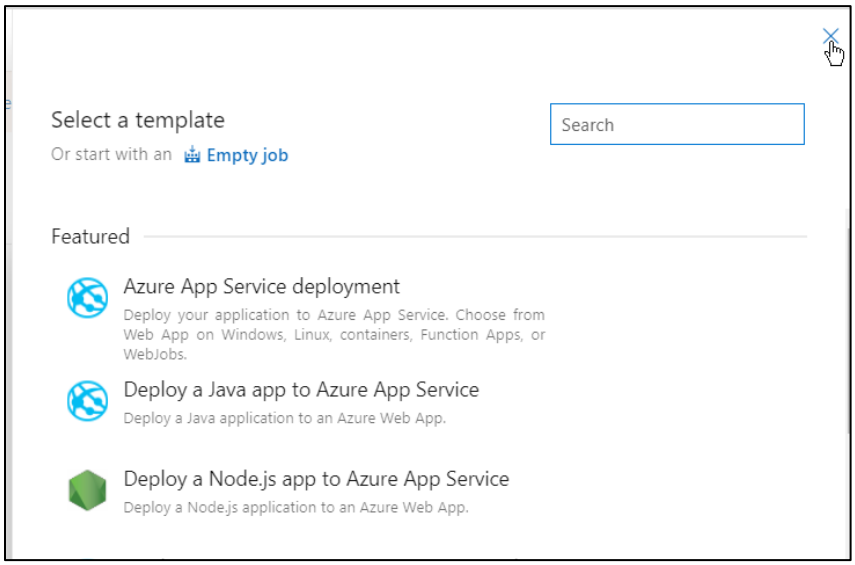
If Azure DevOps is not open now in your browser, make sure to do so now, and let's configure the pipeline.

Create the pipeline

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

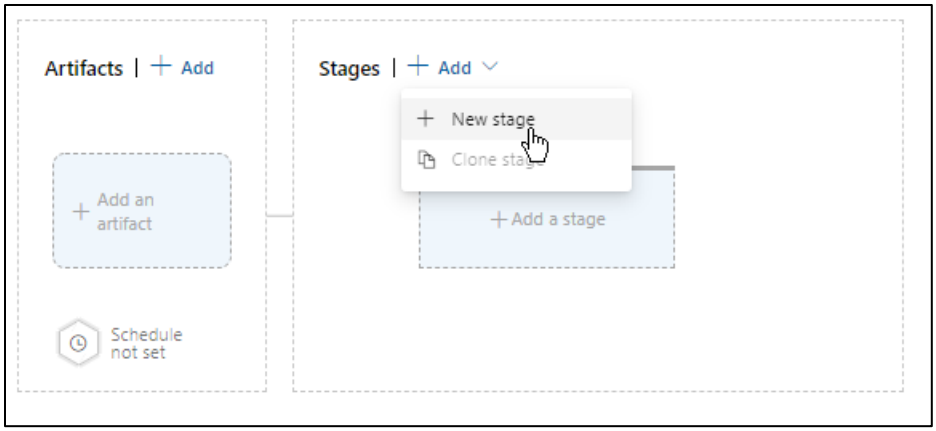


Close the **Select a template** blade open.

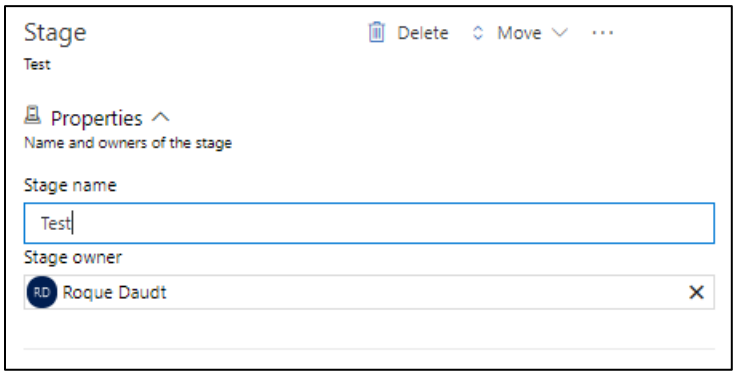


Give the release pipeline better name, if you will.

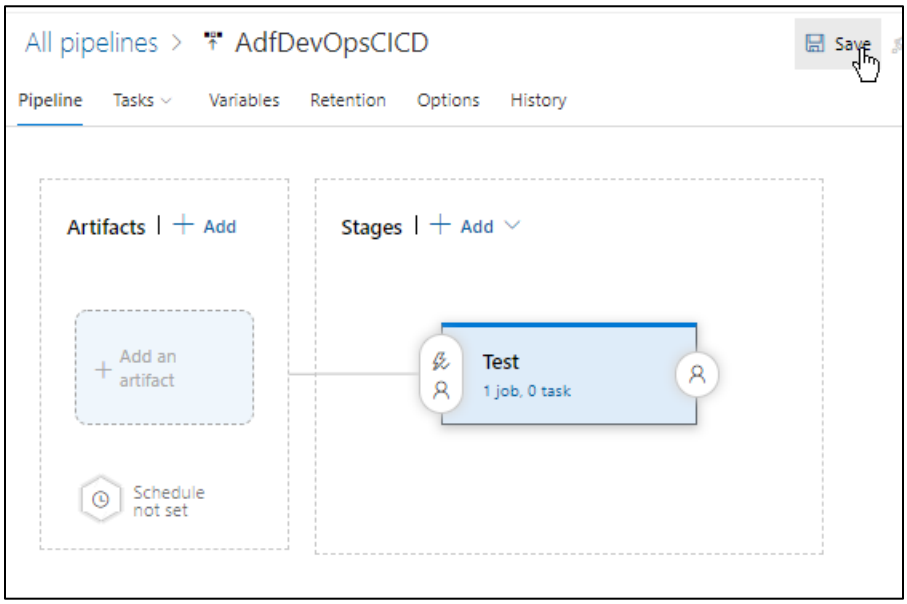
We want the pipeline to first publish to **ADF Tst**. Let's create a new stage then and call it **Test**. Click **New Stage**.



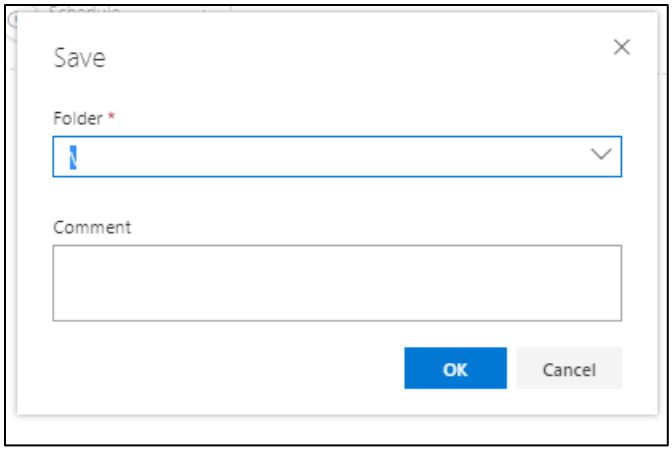
When the **Select a template** blade opens again click **Empty job**. In the next blade, name it **Test**.



Click **Save**.



Click **OK**.



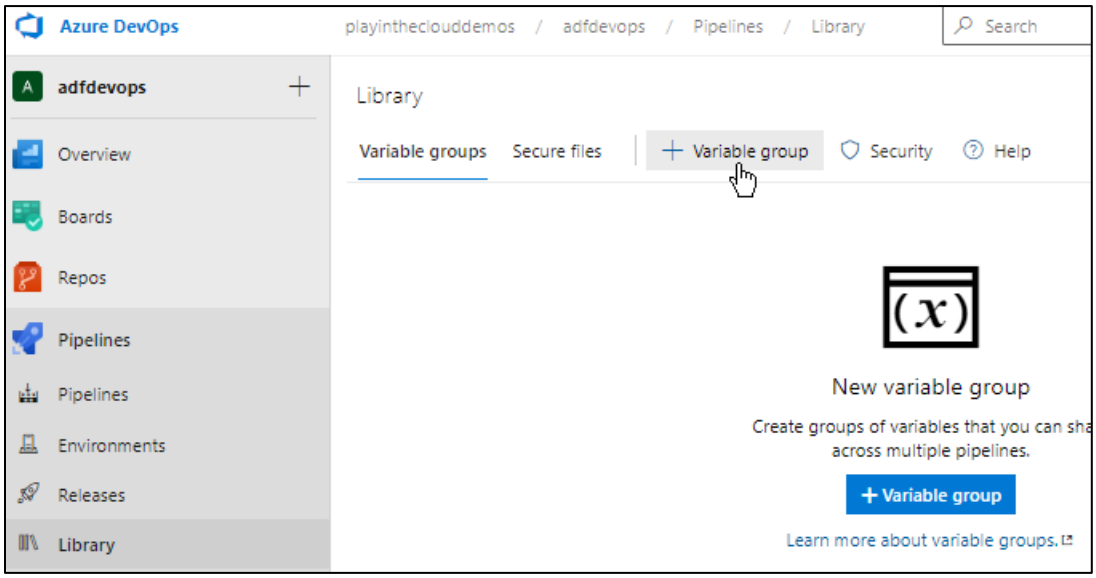
Configure Group Variables

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.

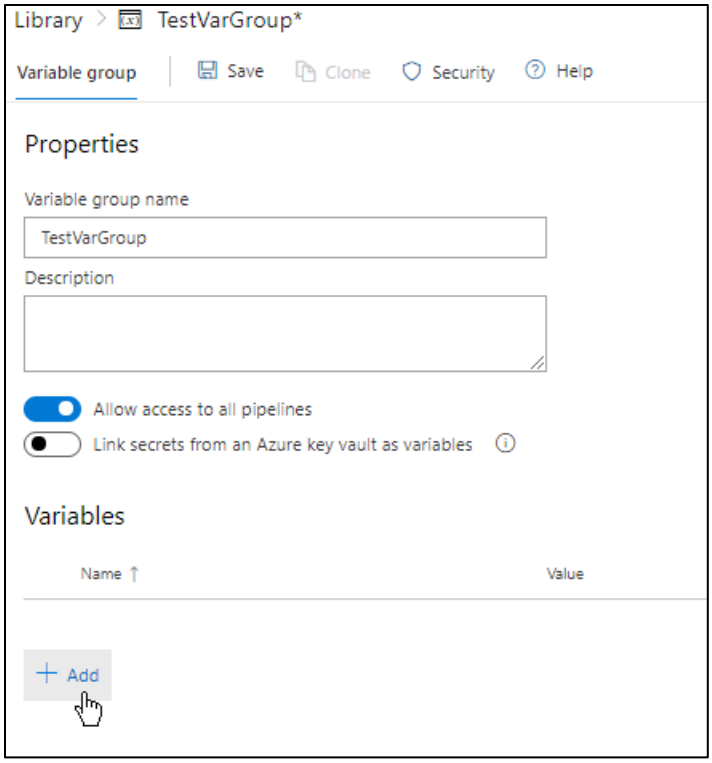
The URI for Azure Key Vault in our development environment, for an instance, is https://rd2020kv**dev**.vault.azure.net/. In test it is https://rd2020kv**tst**.vault.azure.net/ and in prod it is https://rd2020kv**prd**.vault.azure.net/. These identifiers need to be set correctly as we promote the ADF 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.

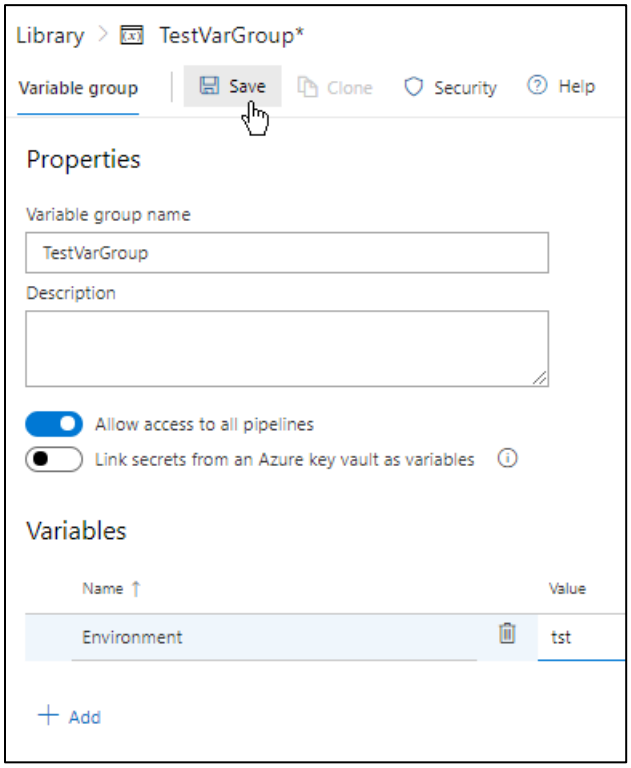
We are now going to show one way to configure them for our release pipeline. Start by creating variable groups. Click **Library** and then **+ Variable group**.



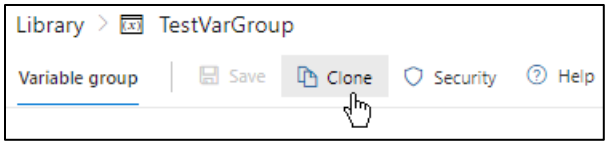
Name it **TestVarGroup** and click **+ Add** in the **Variables** section.



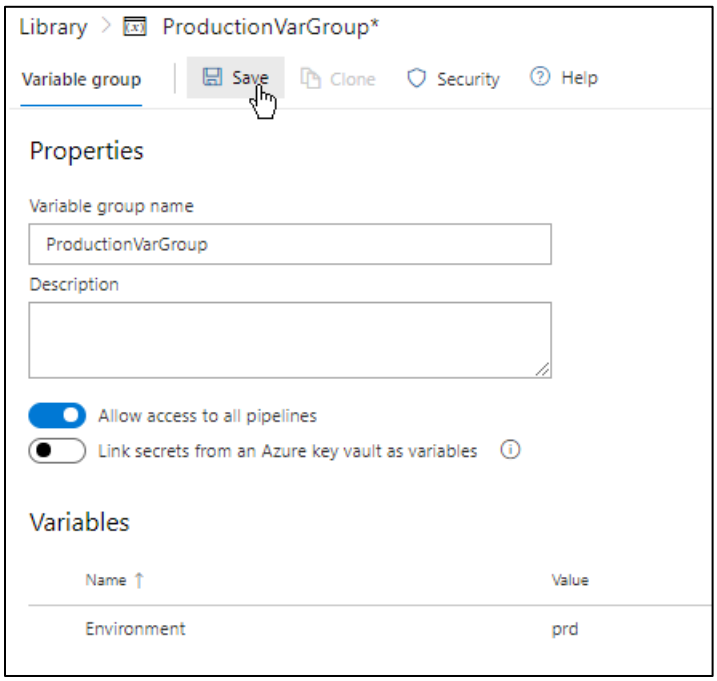
Name the variable **Environment,** give it value "**tst**" and click **Save**.



Click **Clone.**

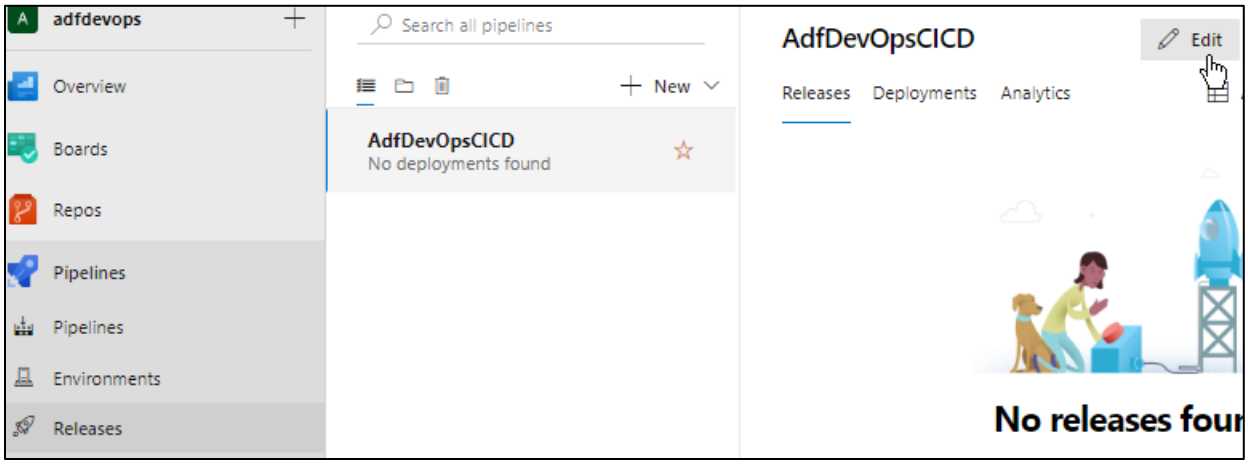


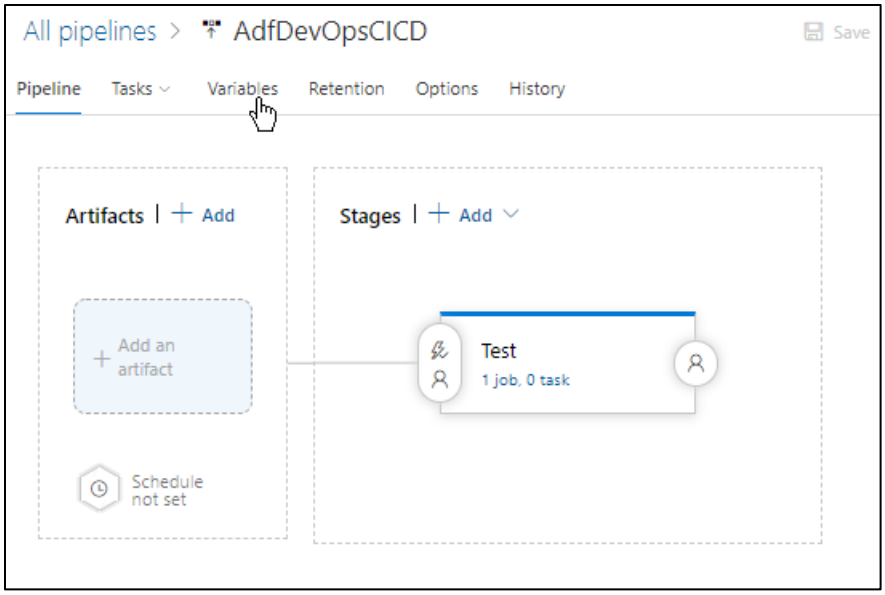
Name the variable group as **ProductionVarGroup**, set value of the Environment variable to "**prd**" and click **Save**.



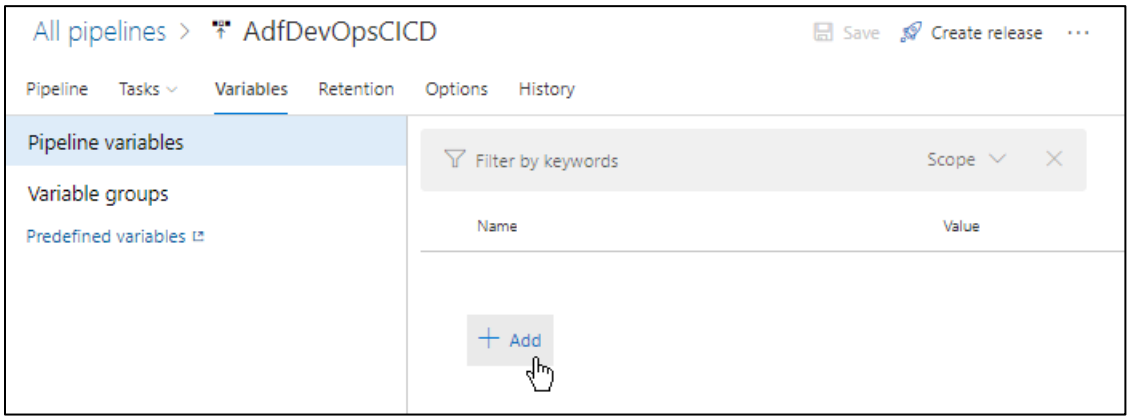
Configure Release Pipeline Variables

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





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

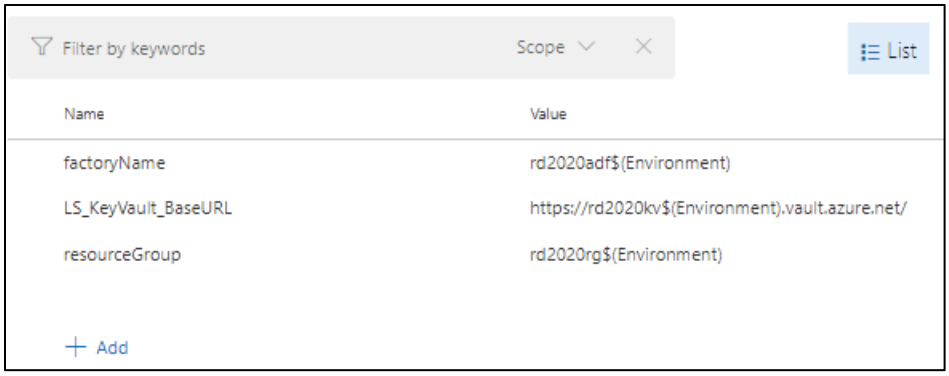


Create variables with the names shown in the next image.

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 **rd2020**adf$(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 **sb202009**adf$(Environment).

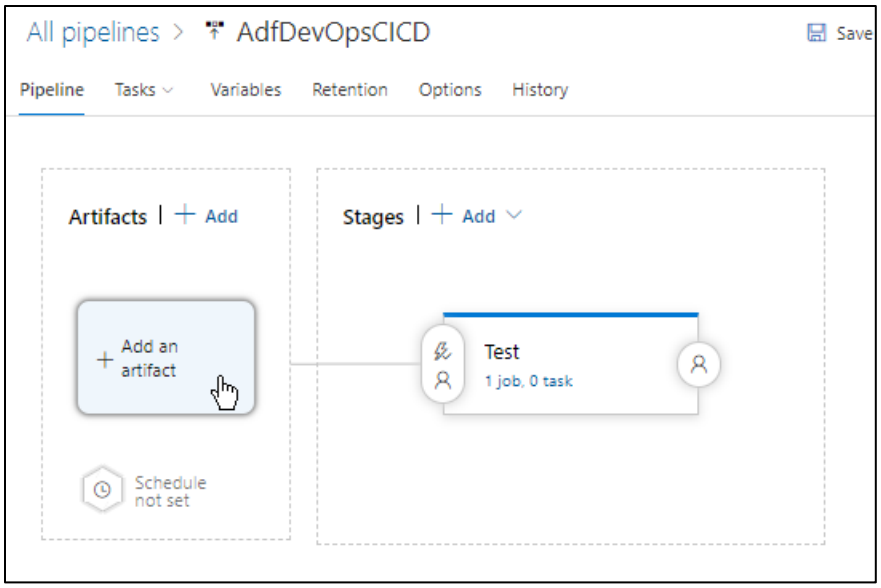
The same applies to the other two variables.



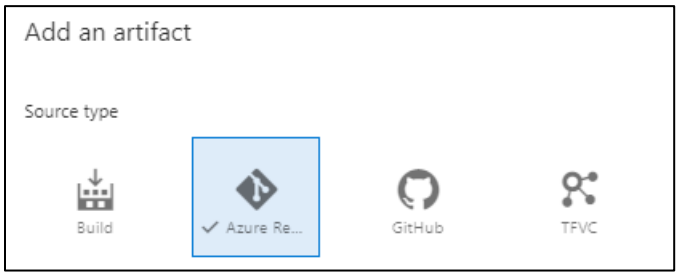
Click **Save**.

Configure the source for the pipeline

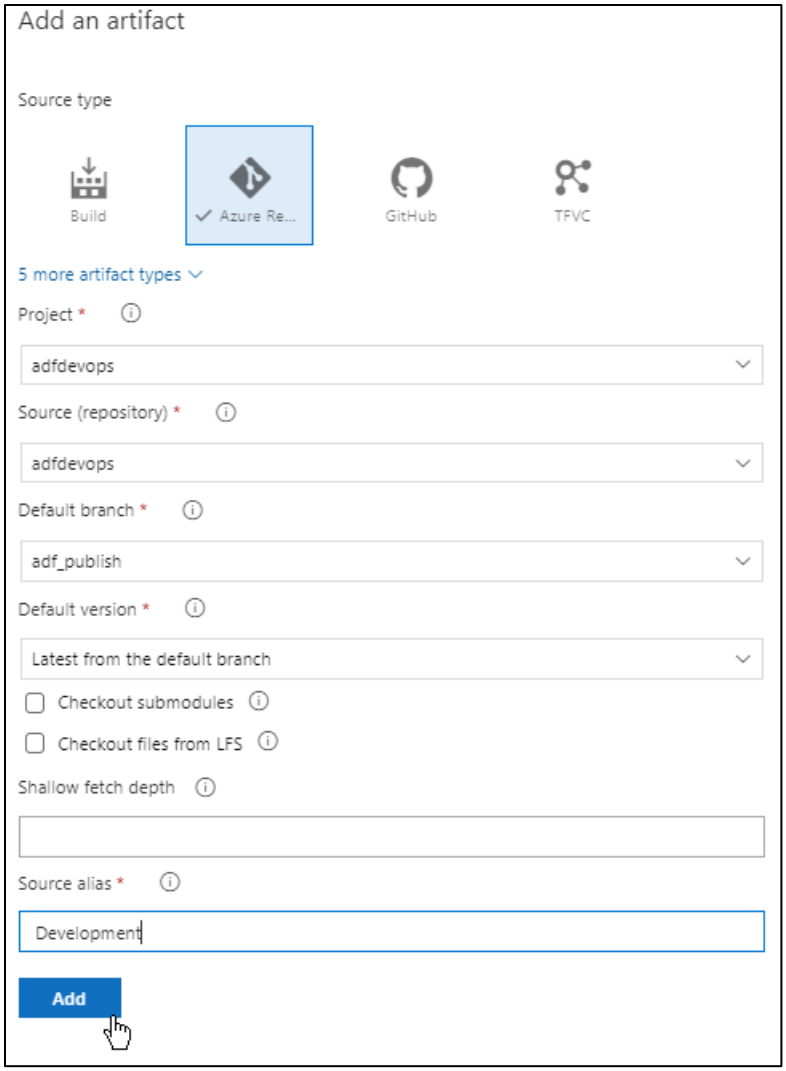
Go back to the **Pipeline** tab and click **+ Add an artifact**.



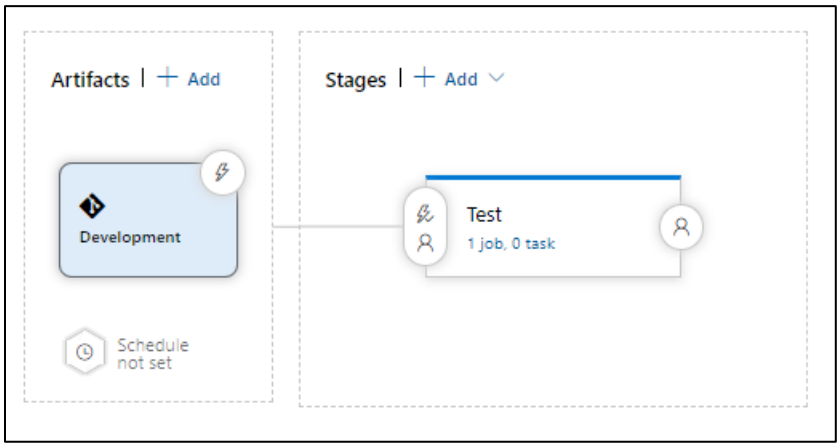
In the **Add an artifact** blade, select **Azure Repos GIT.**



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.

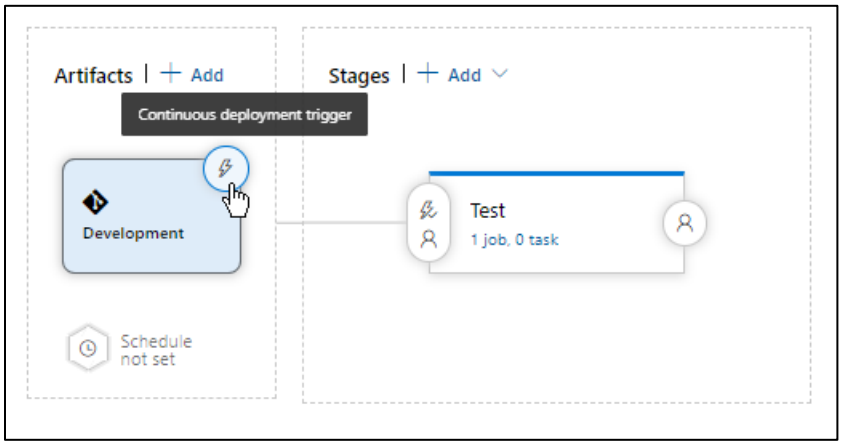


Click **Add**. This is what the pipeline looks like now.

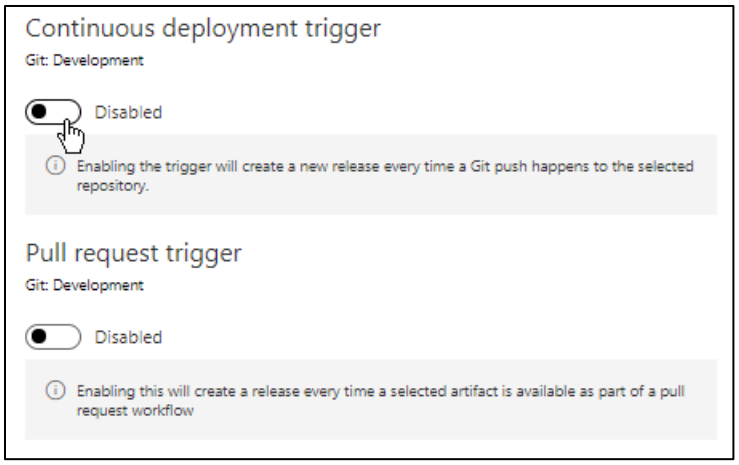


Configure the Continuous Deployment trigger

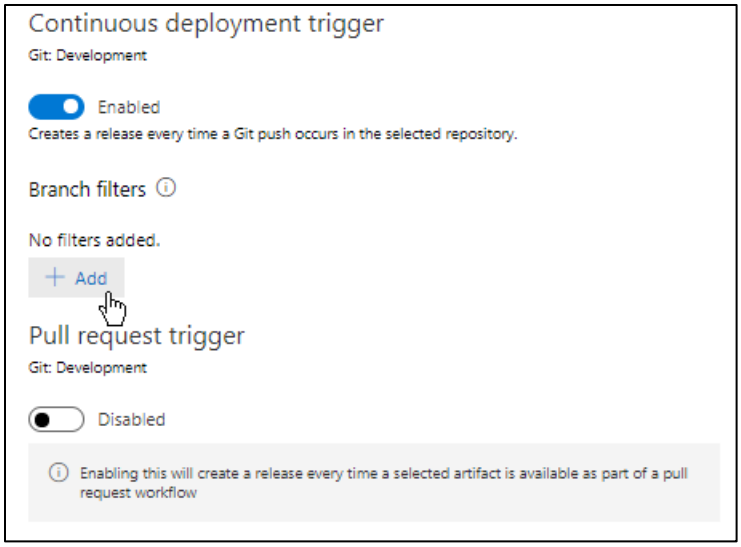
We want the release pipeline to fire every time that the **adf_publish** branch is updated. Click the **Continuous deployment trigger** button.



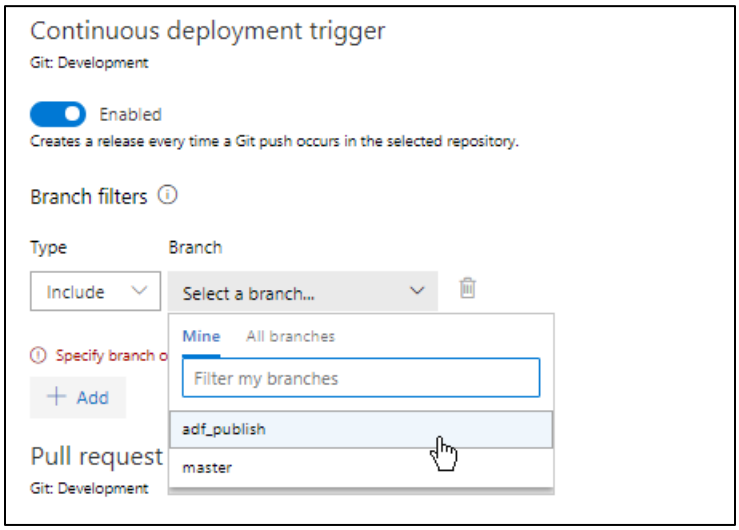
Enable **Git Development**.



Click **+ Add**.

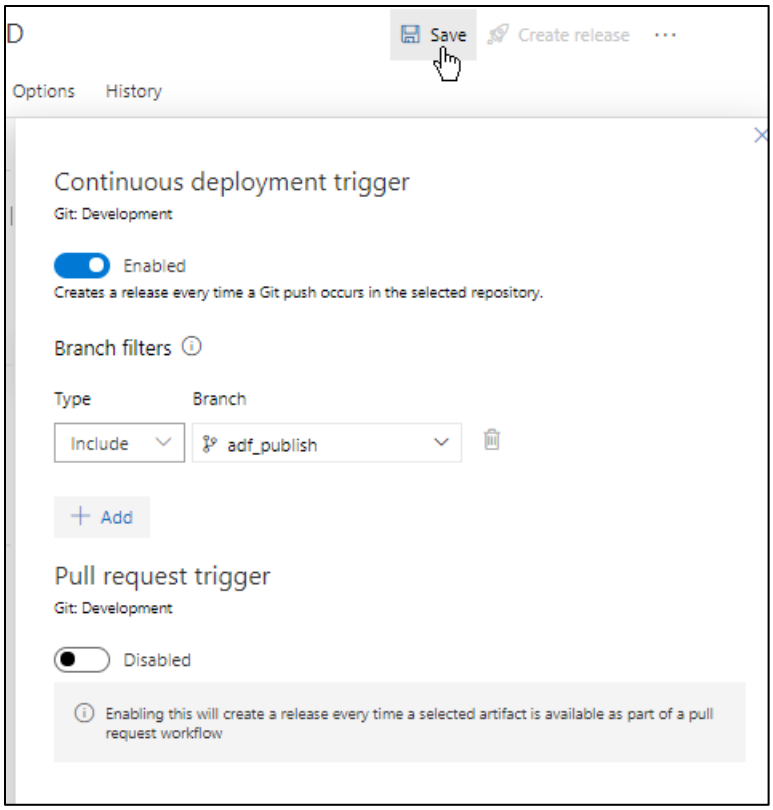


Select **adf_publish**.



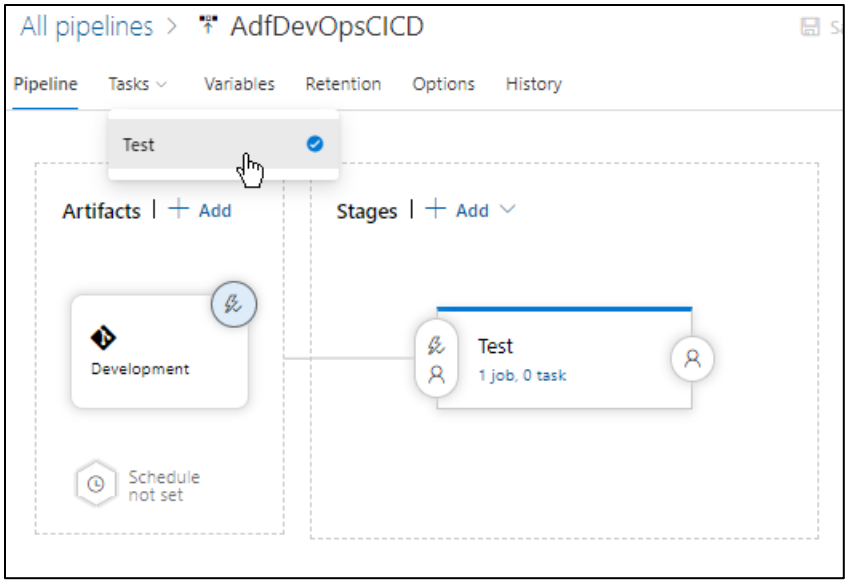
Click **Save**.

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

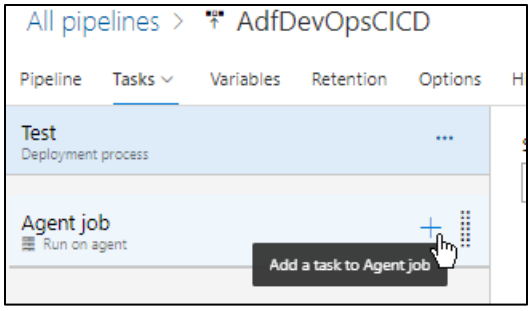


Configure the Test stage

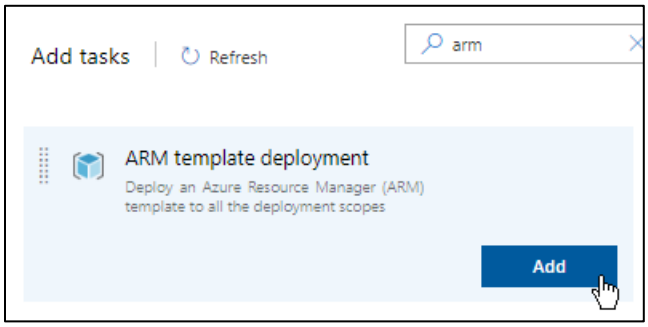
We will now work on the task



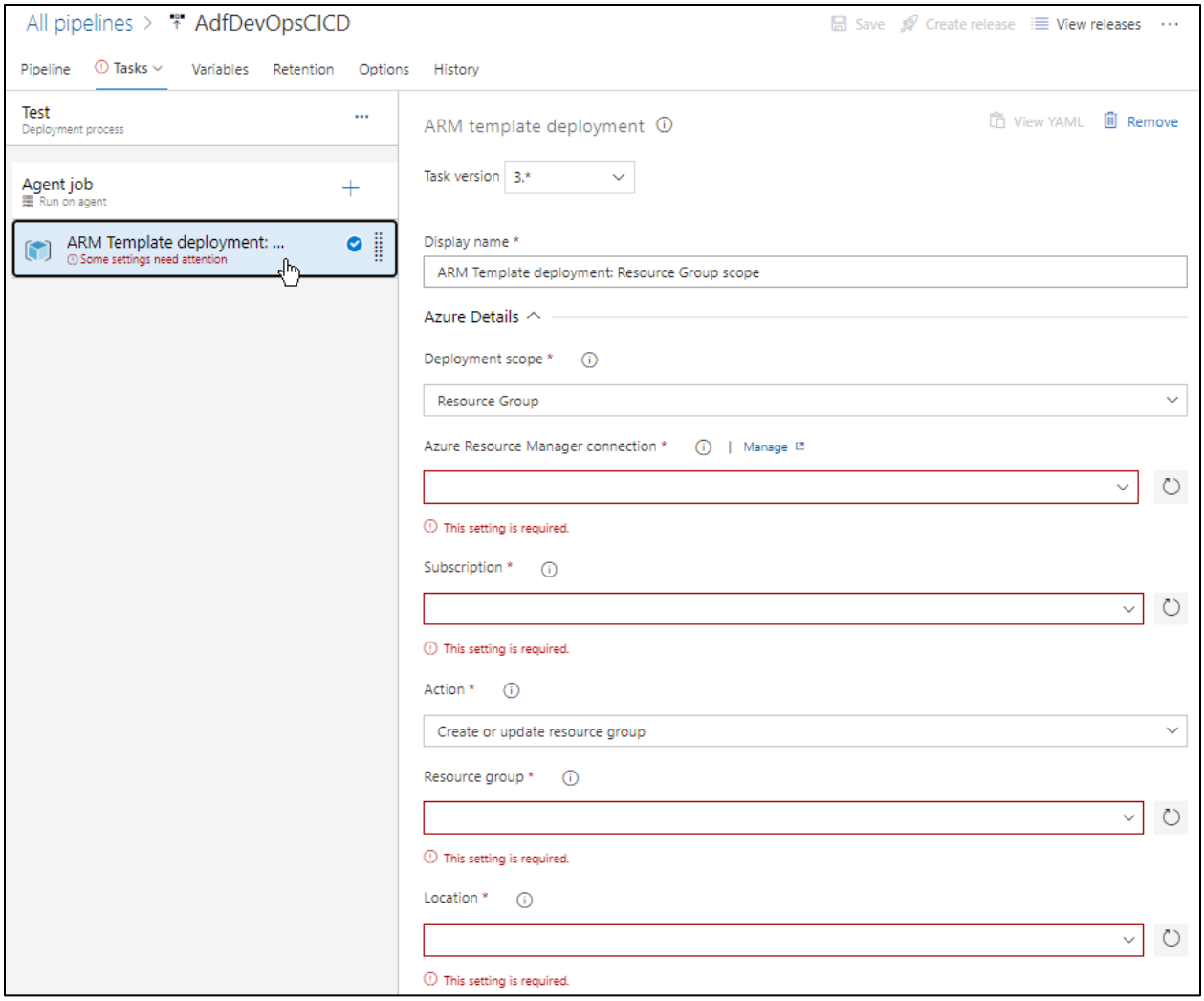
Click the **+ (Add a task to Agent Job)** sign shown below.



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

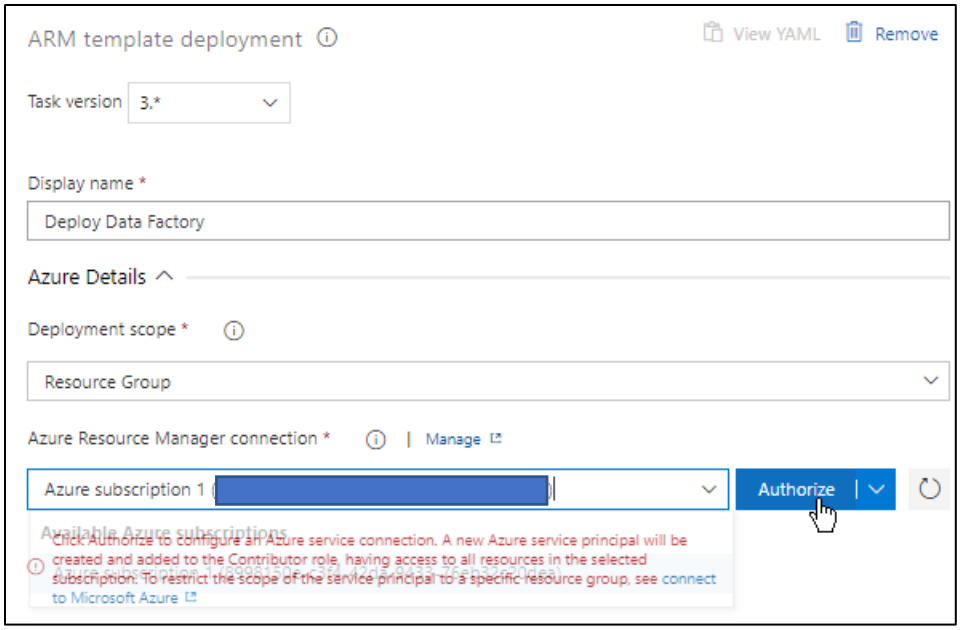


Back **ARM Template deployment** to configure it.



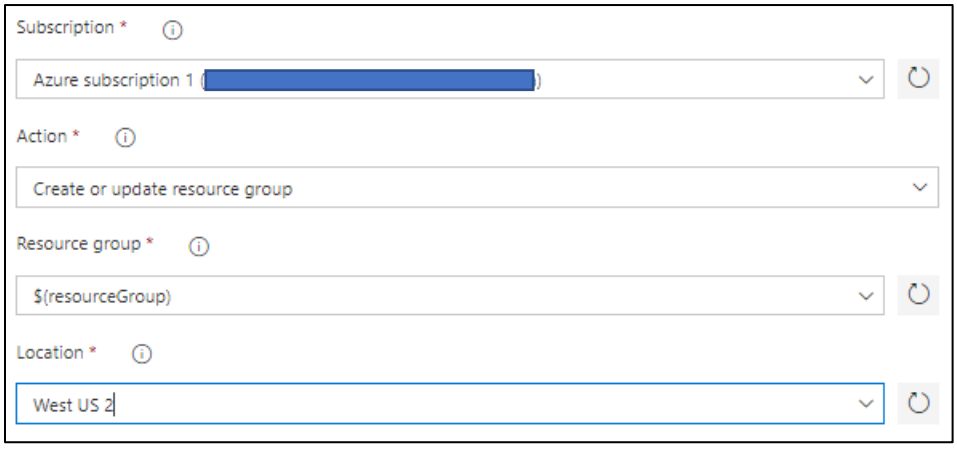
Start entering configuration.

- 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.



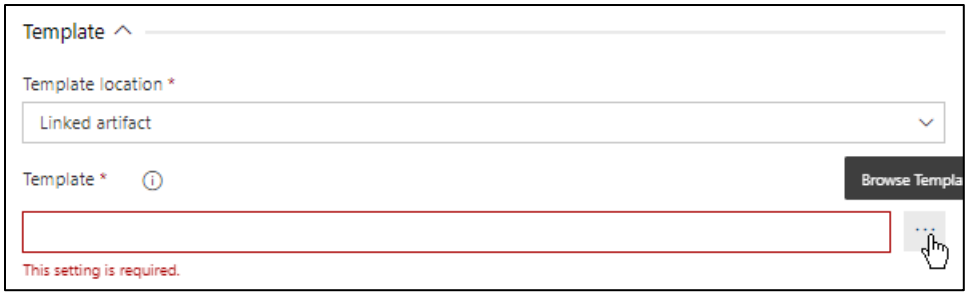
Once authorization completes (it takes a few seconds), continue configuration:

- 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

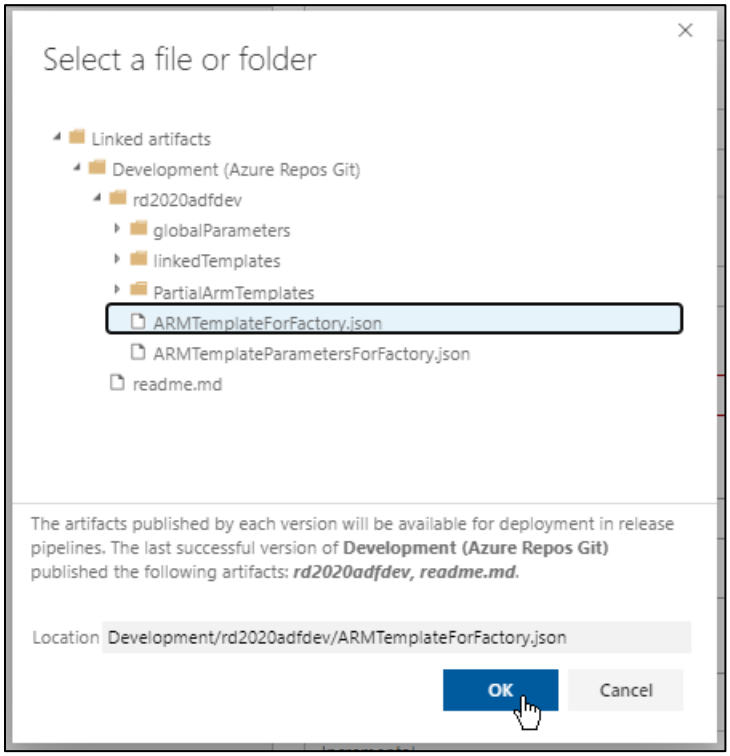


Note that **$(resourceGroup)** is a reference to the **resourceGroup** pipeline variable that we created earlier in this lab.

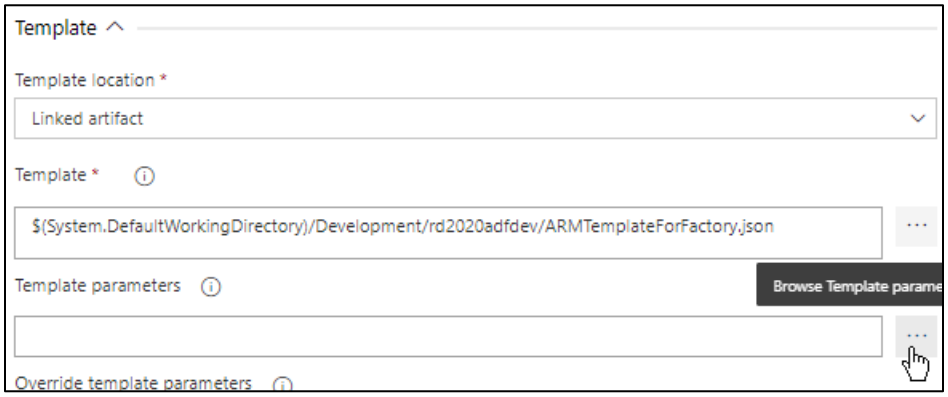
Start configuring the **Template** section of the blade. Set **Template location** to **Linked artifact.** Then, click the 3-dot button to the right of the Template control, as shown below.



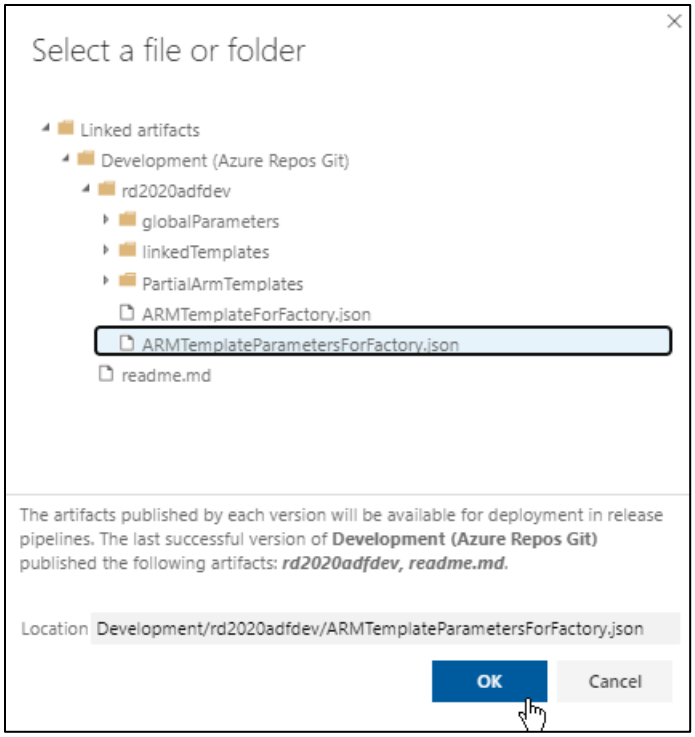
Expand the folders hierarchy, select the **ARMTemplateForFactory.json** file and click **OK**.



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

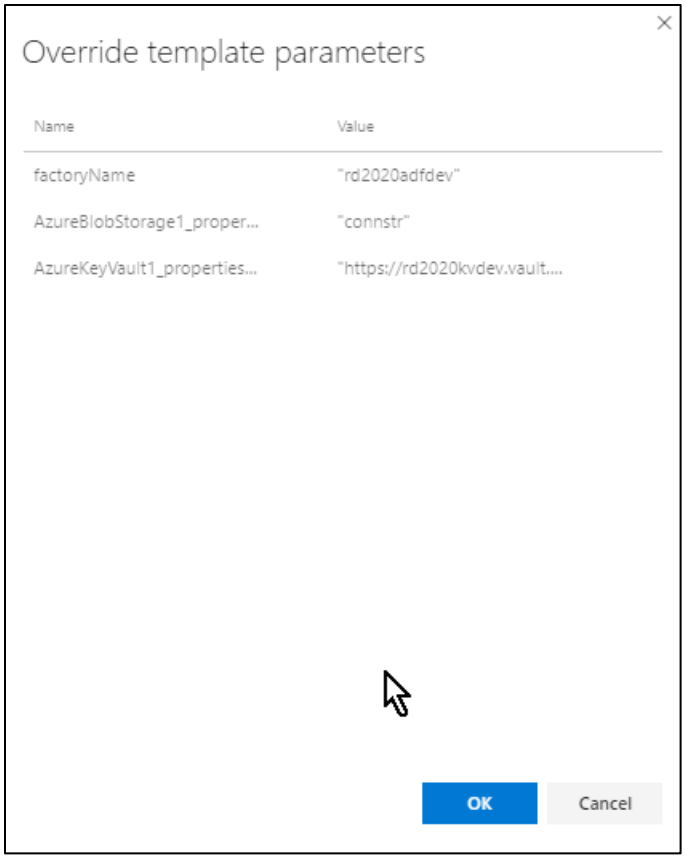


This time select the **ARMTemplateParametersForFactory.json** file. Click **OK**.

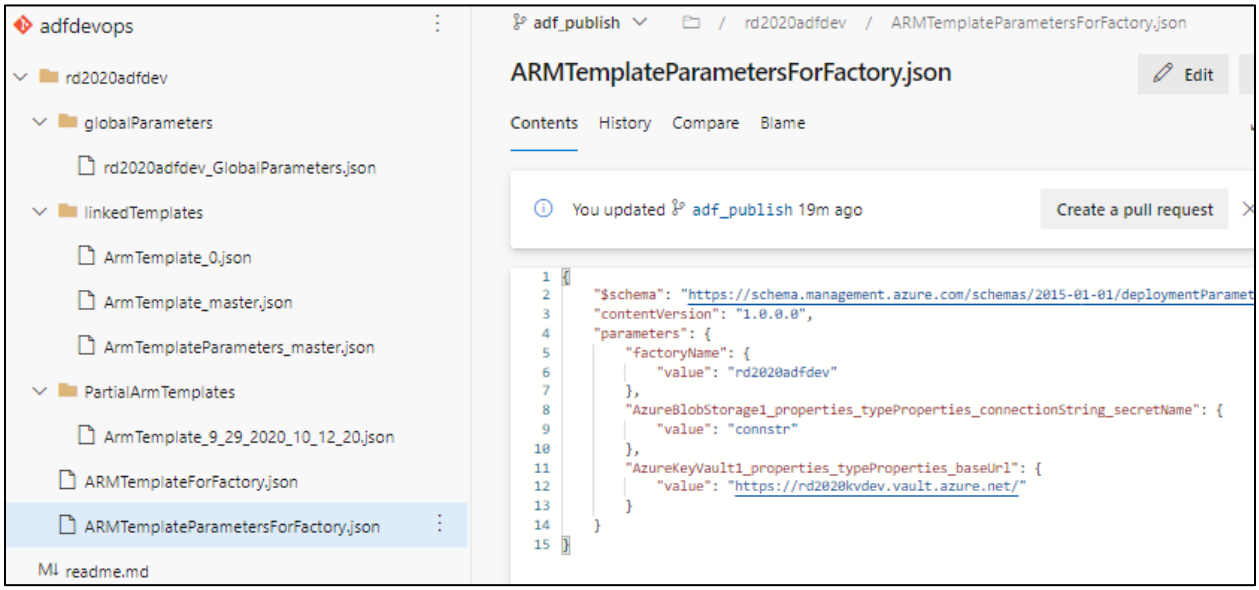


In the **Override template parameters** field, click the **Edit Override template parameters** button.

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

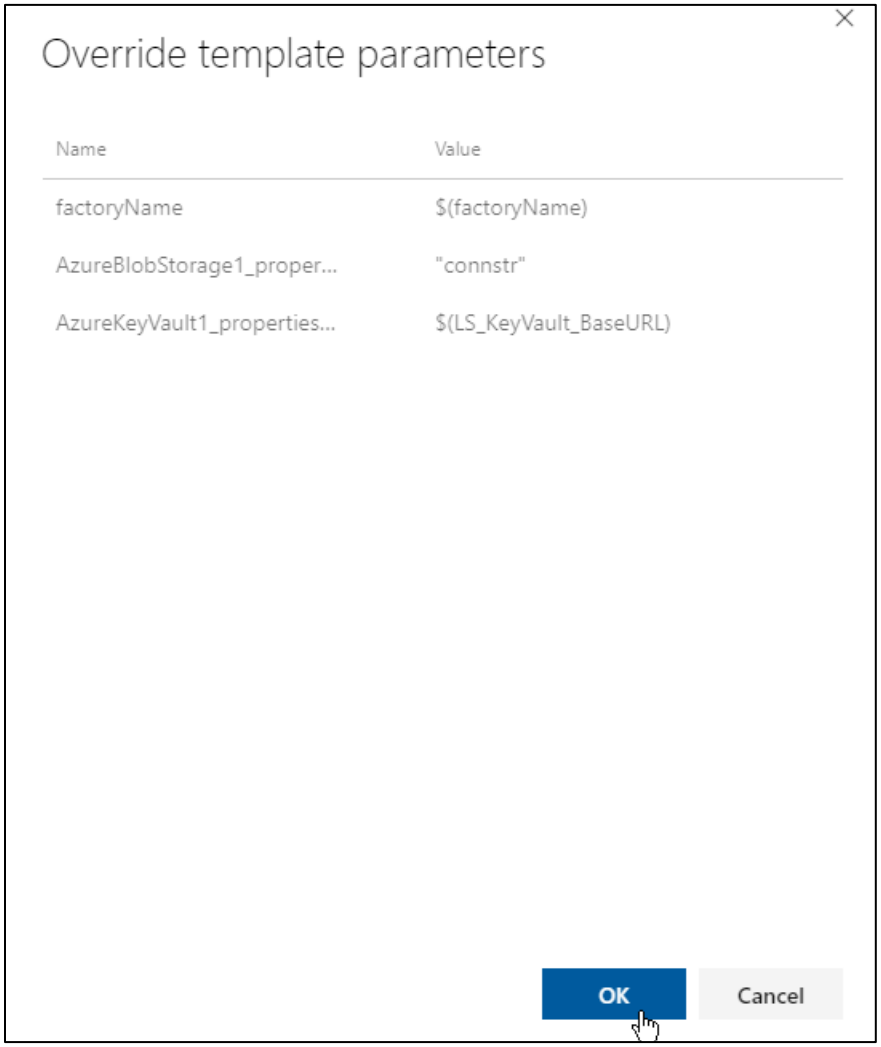


We have inspected this file in the previous lab. They reflect values that are associated to the development environment because this ARM template was created out of publishing from **master** (DEV) to **adf_publishing**. I reproduce the file below for your convenience.

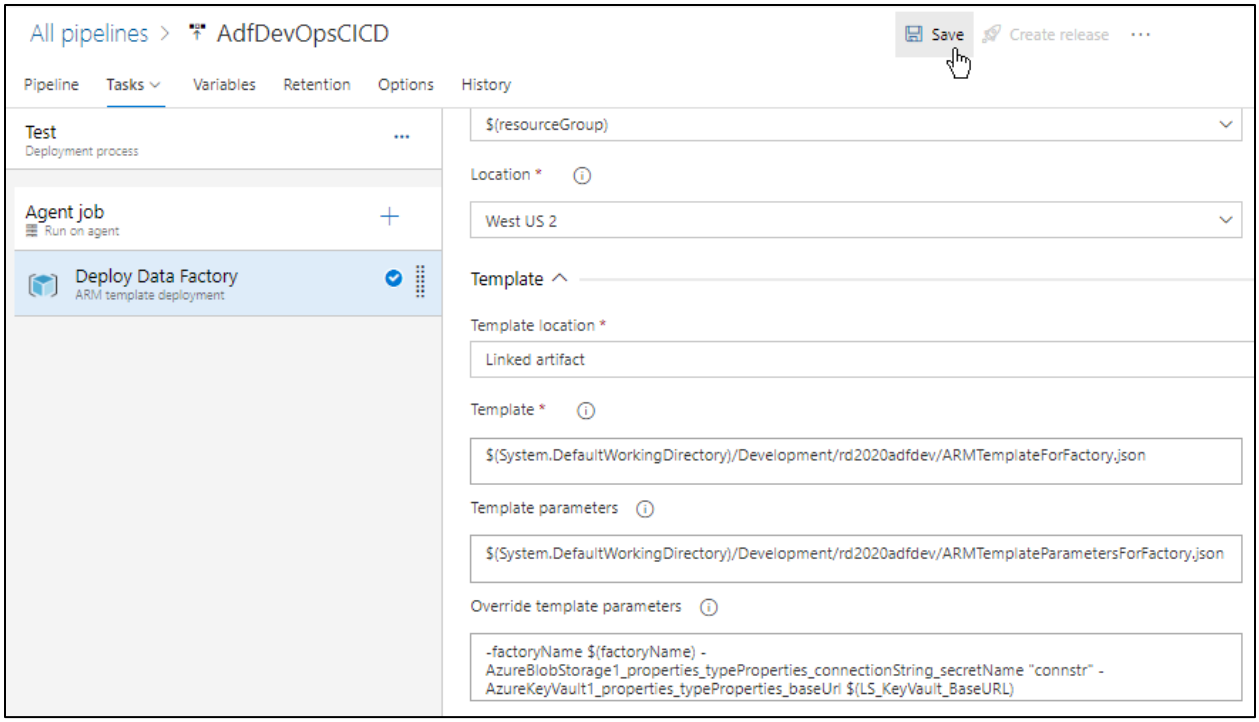


What we want then is to use variables to configure this action, make it generic and suitable for deployment to test and production. For this reason, we will update the value of two of the parameters in this window, as shown below.

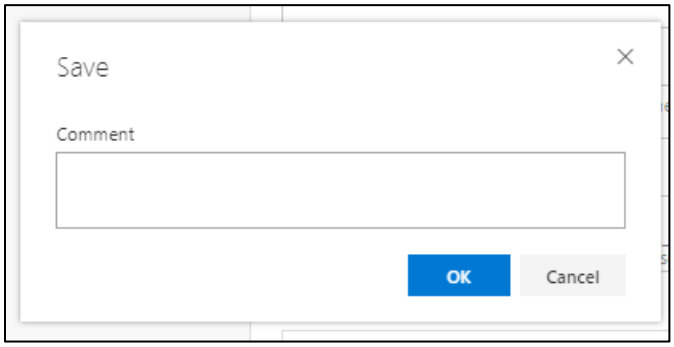
- **factoryName** is set to **$(factoryName)**
- **AzureKeyVault1_properties_typeProperties_baseUrl** is set to **$(LS_KeyVault_BaseURL)**



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

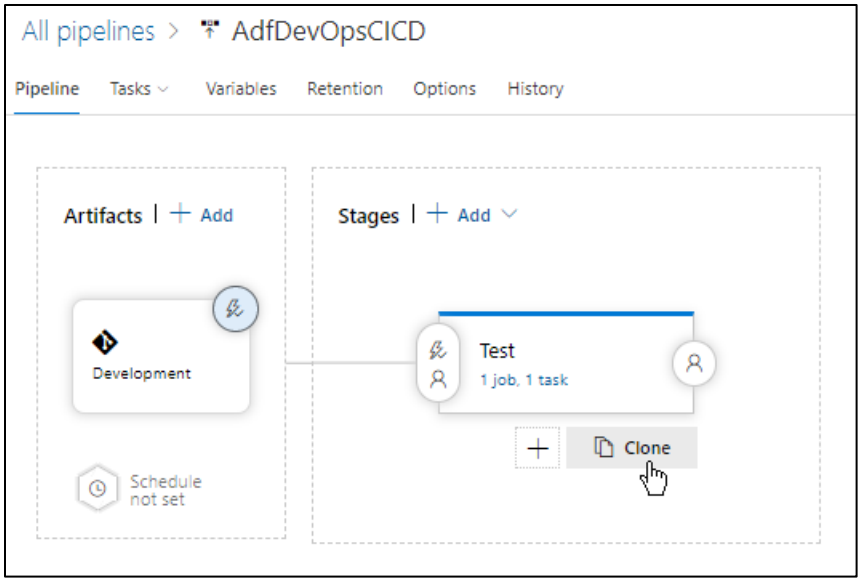


Click **OK**.

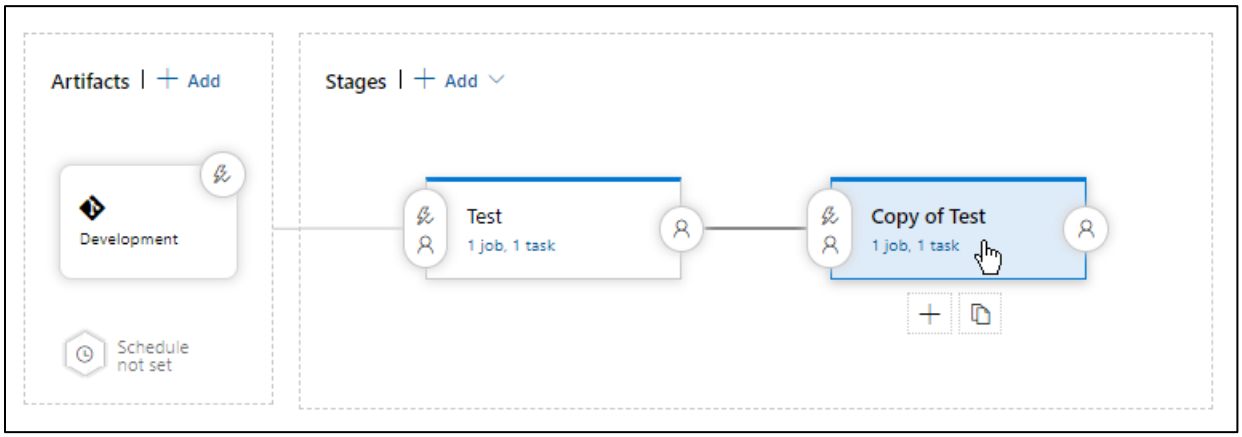


Configure the Production stage

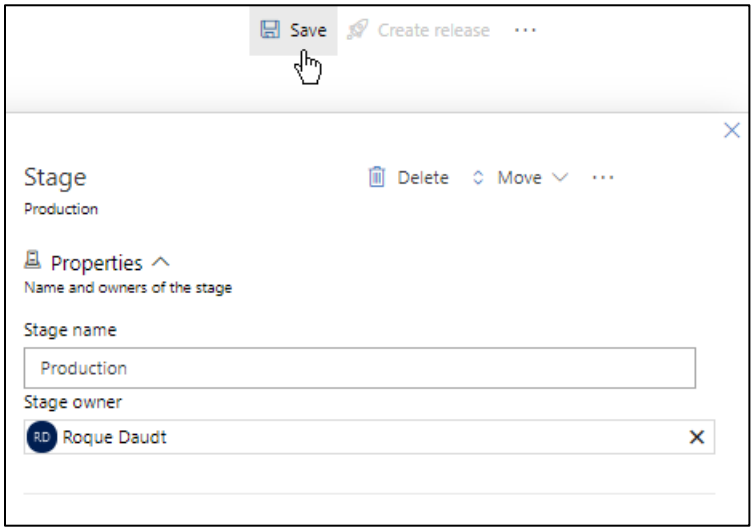
Back to the **Pipeline** tab, click **Clone** as shown below.



Click the stage box and

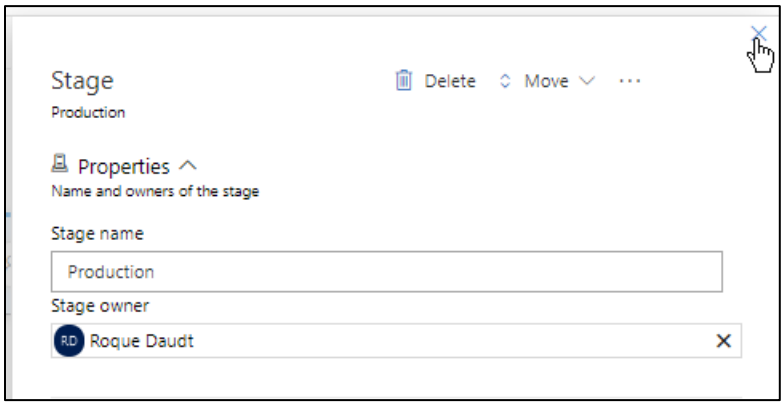


Set **Stage Name** as **Production** and click **Save**.



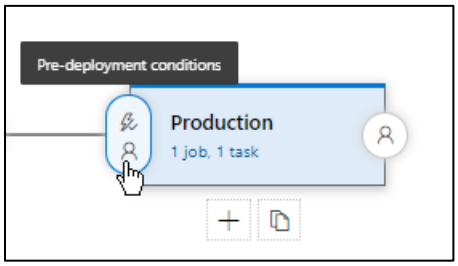
Click **OK**.

Close the blade.

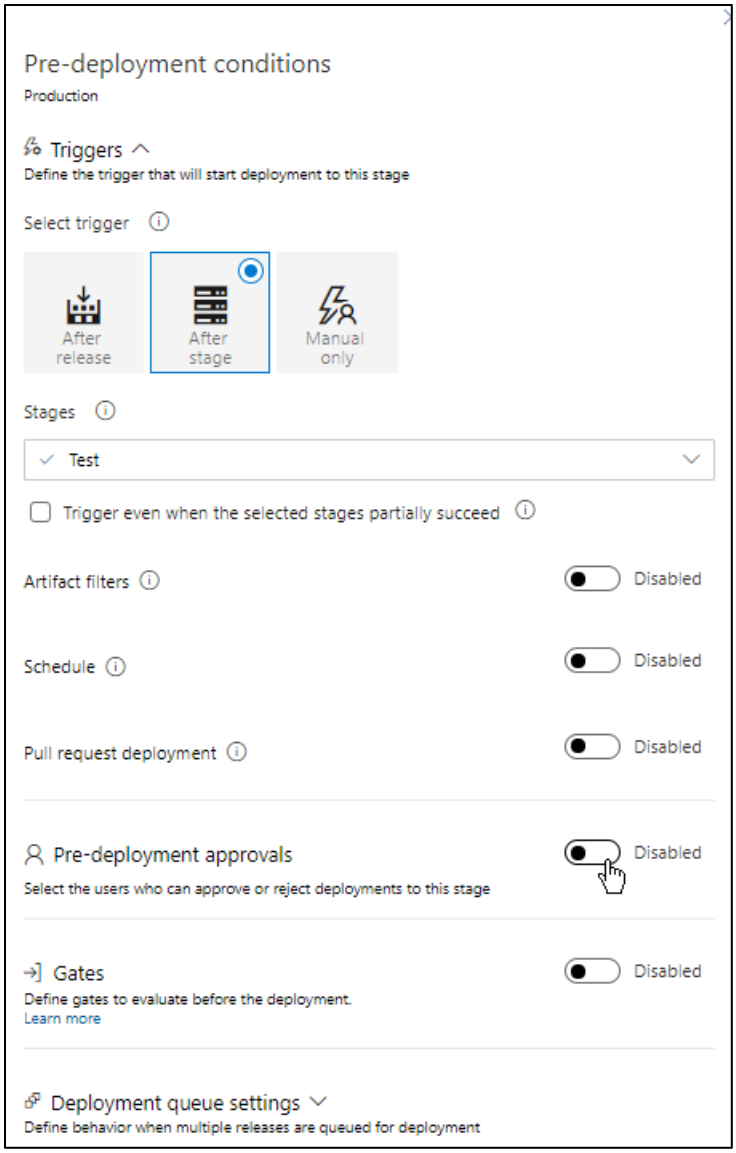


We want now to add pre-deployment approval for production. Once this implemented, someone will need to manually approve the deployment for it to happen.

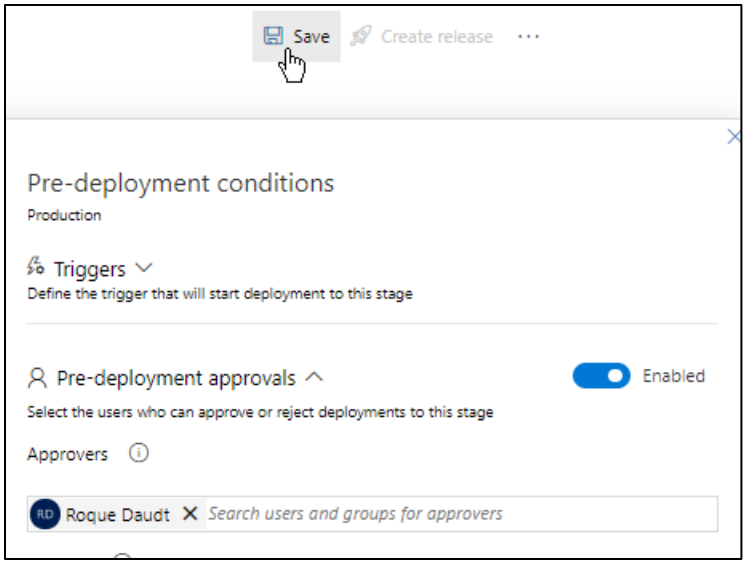
To do so, we click **Pre-deployment conditions**.



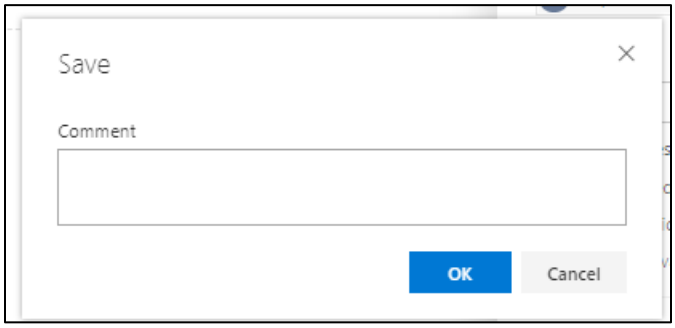
When the **Pre-deployment conditions** blade is shown, enable **Pre-deployment approvals**, as shown below.



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



Click **OK**.



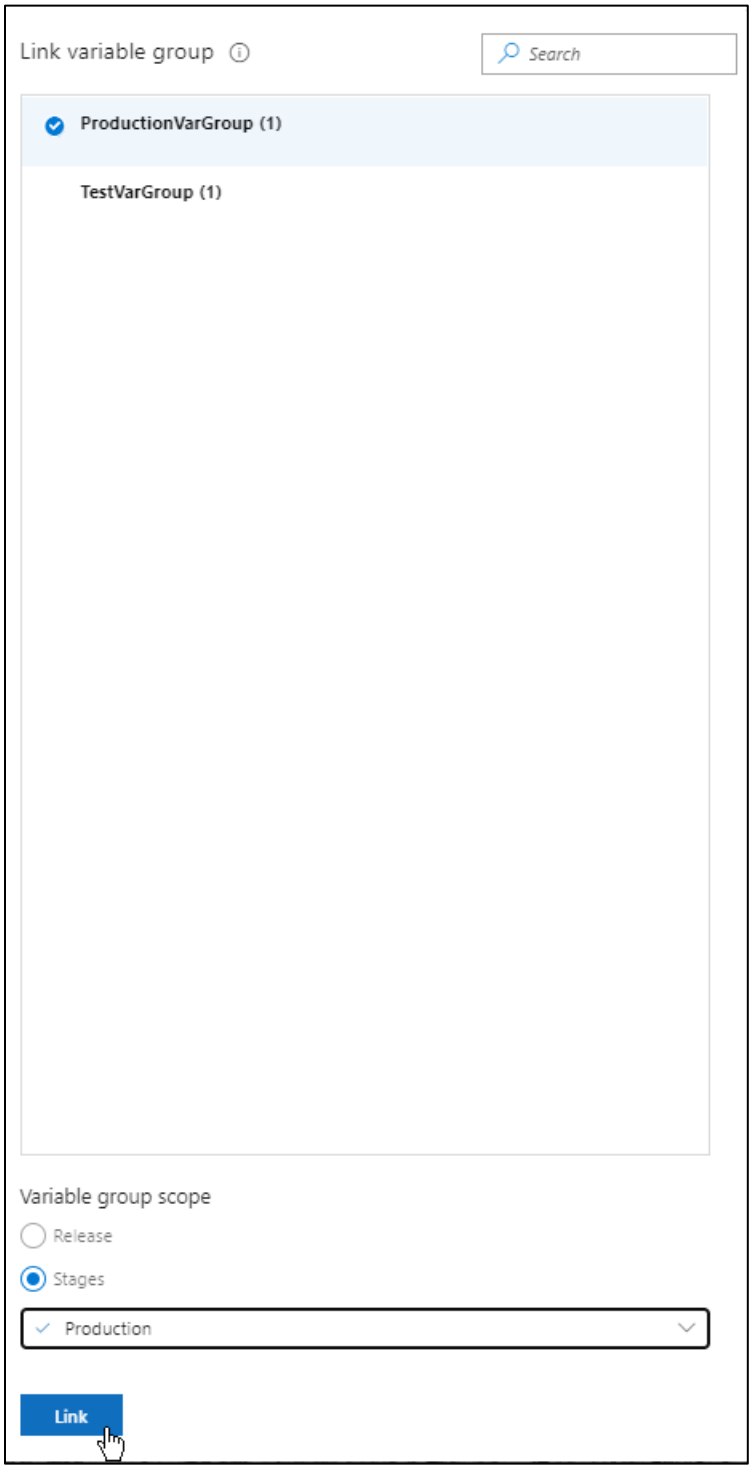
Close the **Pre-deployment** conditions blade.

Linked variables

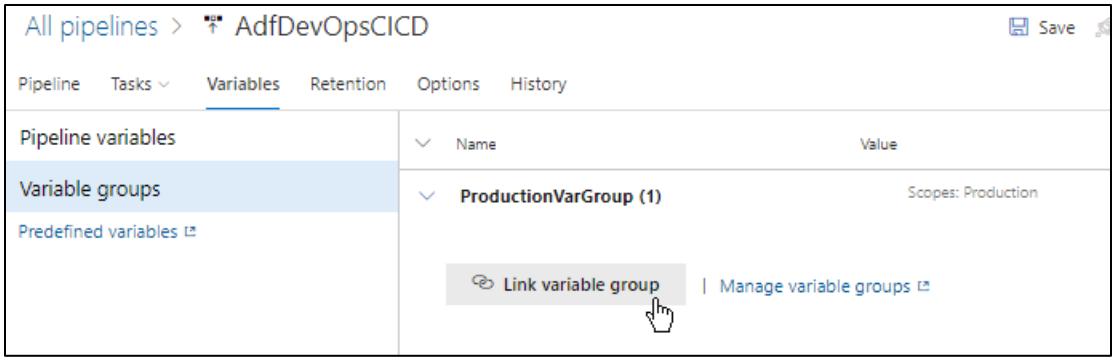
At this time, both stages are configured to use the content of the pipeline variables for the deployments. However, the content of these variables must change accordingly, depending on which environment its been deployed at each stage. There is a need them for a mechanism to ensure that it happens. This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

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

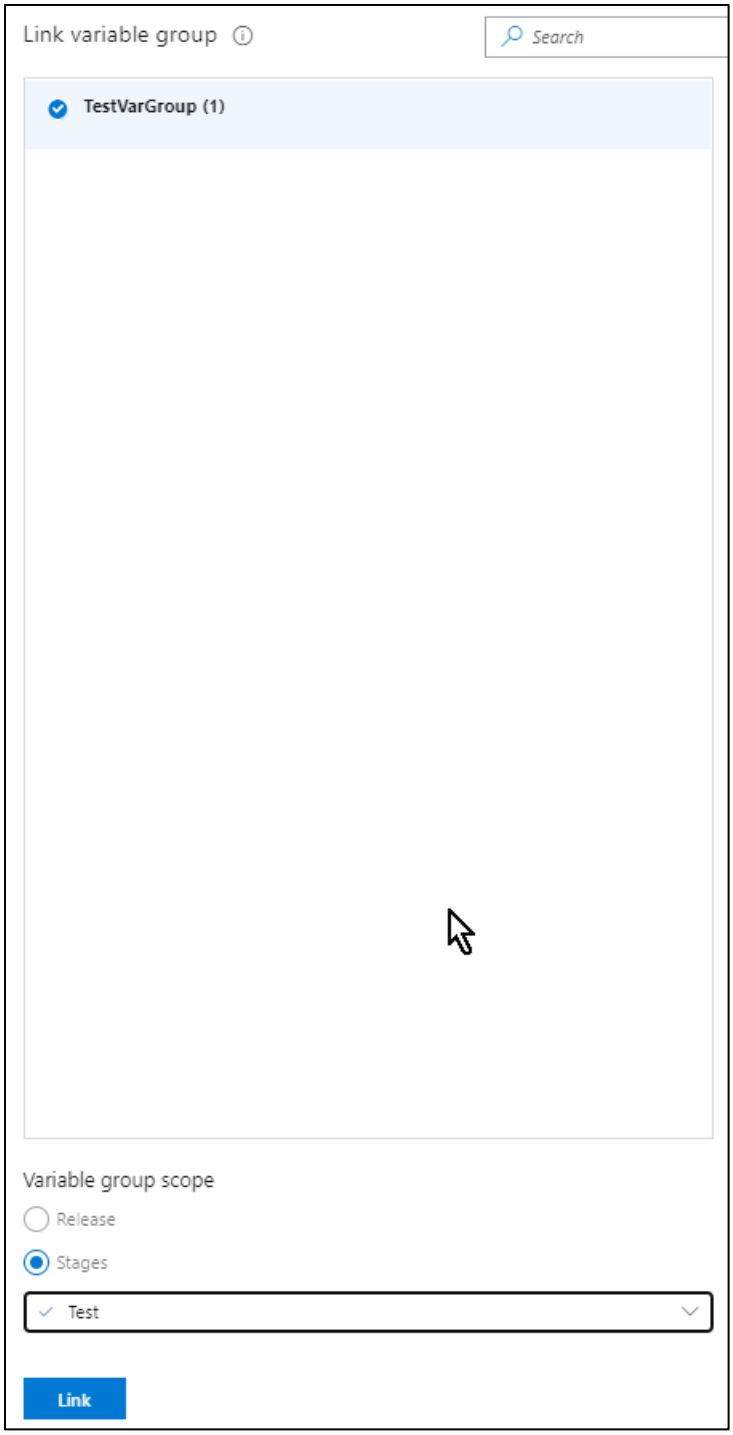
In the Link variable group blade, select **ProductionVarGroup (1),** set **Variable group scope** to **Stages** , select the **Production** stage and click **Link**.



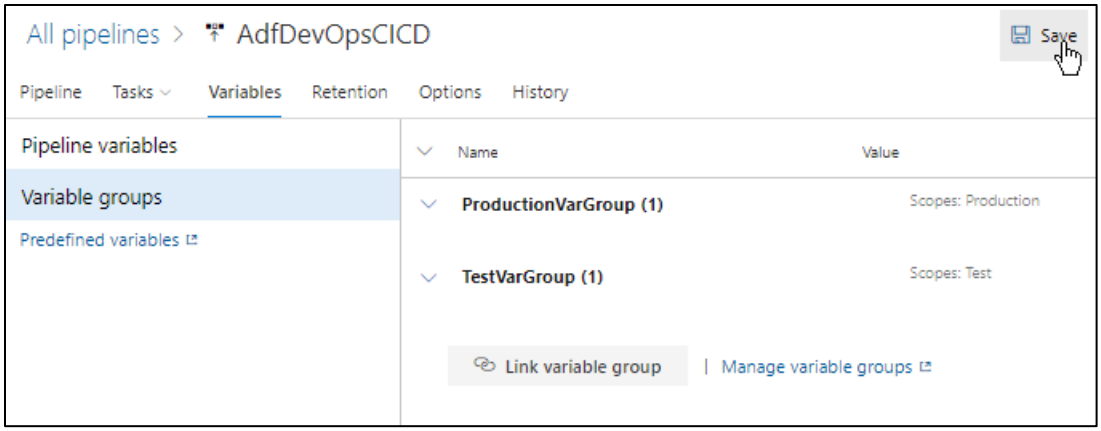
Click the **Link variable group** button again.



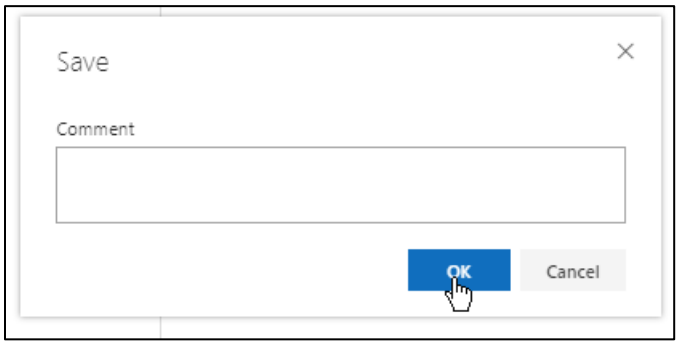
Enter the configuration shown below and click **Link**.



Save the work.



Click **OK**.

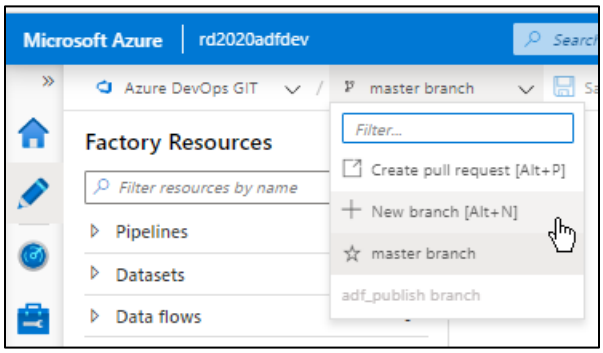


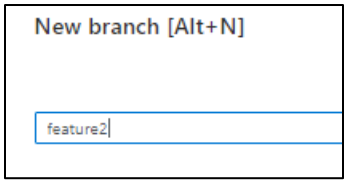
We have completed the pipeline's configuration. Let's run it now.

Test the release pipeline

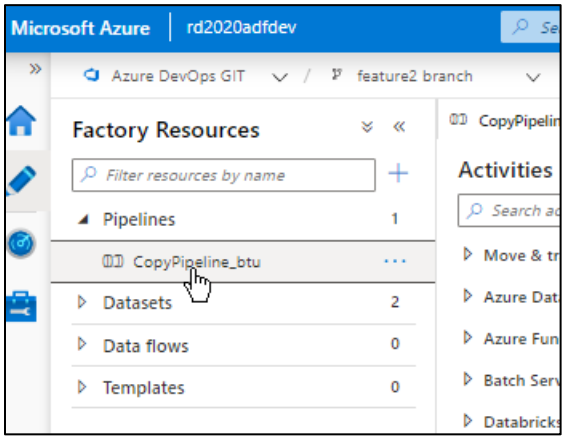
To verify that the process is working from end-to-end, we will apply another minor change to the pipeline in ADF Dev. Once we publish it to **adf_publish**, it should trigger the release pipeline.

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

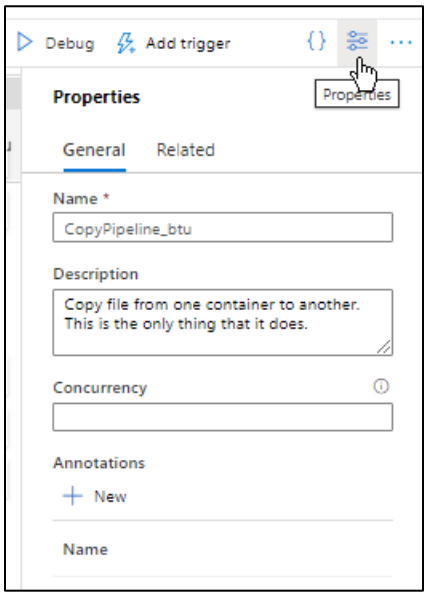




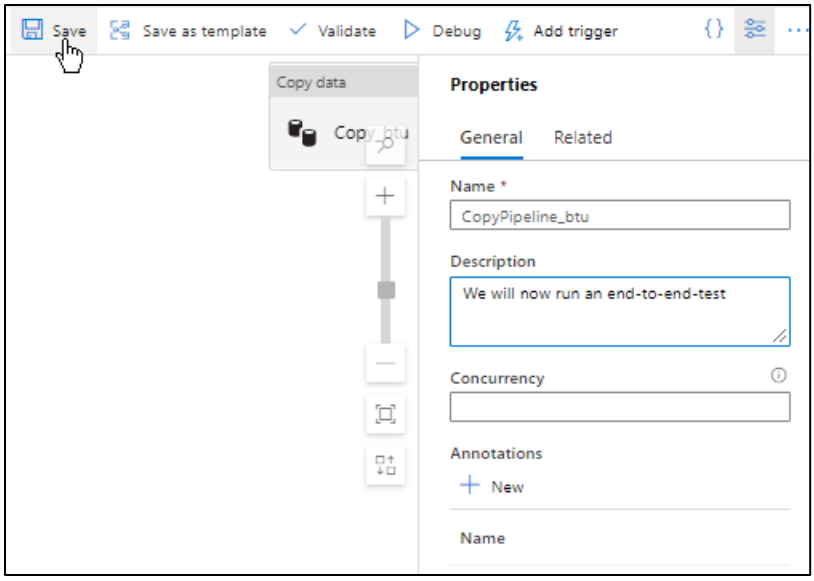
Open pipeline **CopyPipeline_btu**.



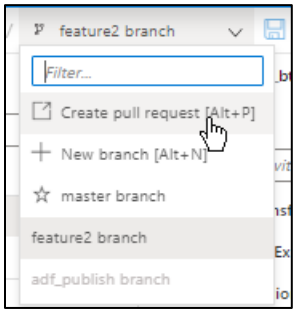
Click the **properties** button.



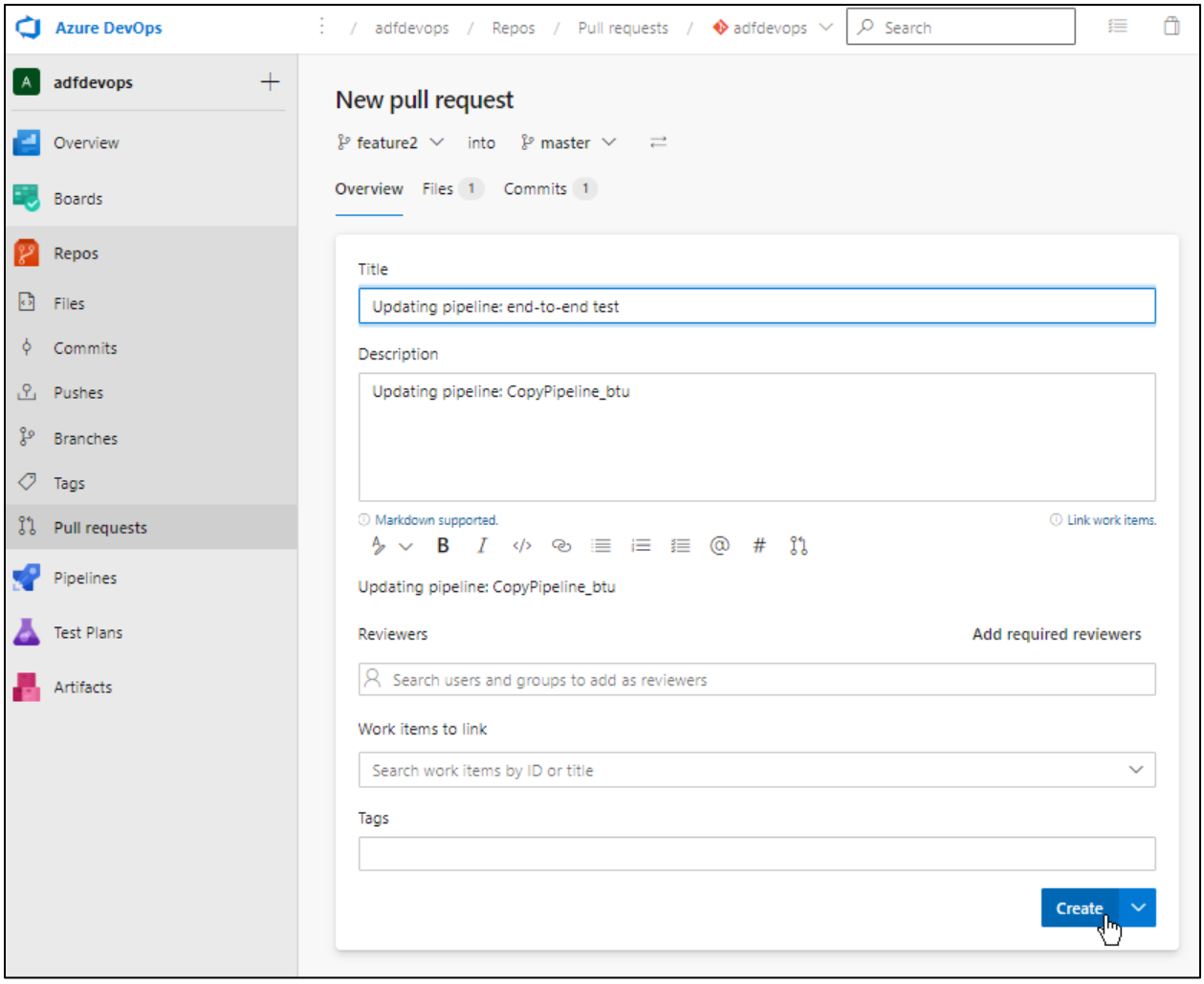
Change the description to anything else and click **Save**.



Create a new pull request.



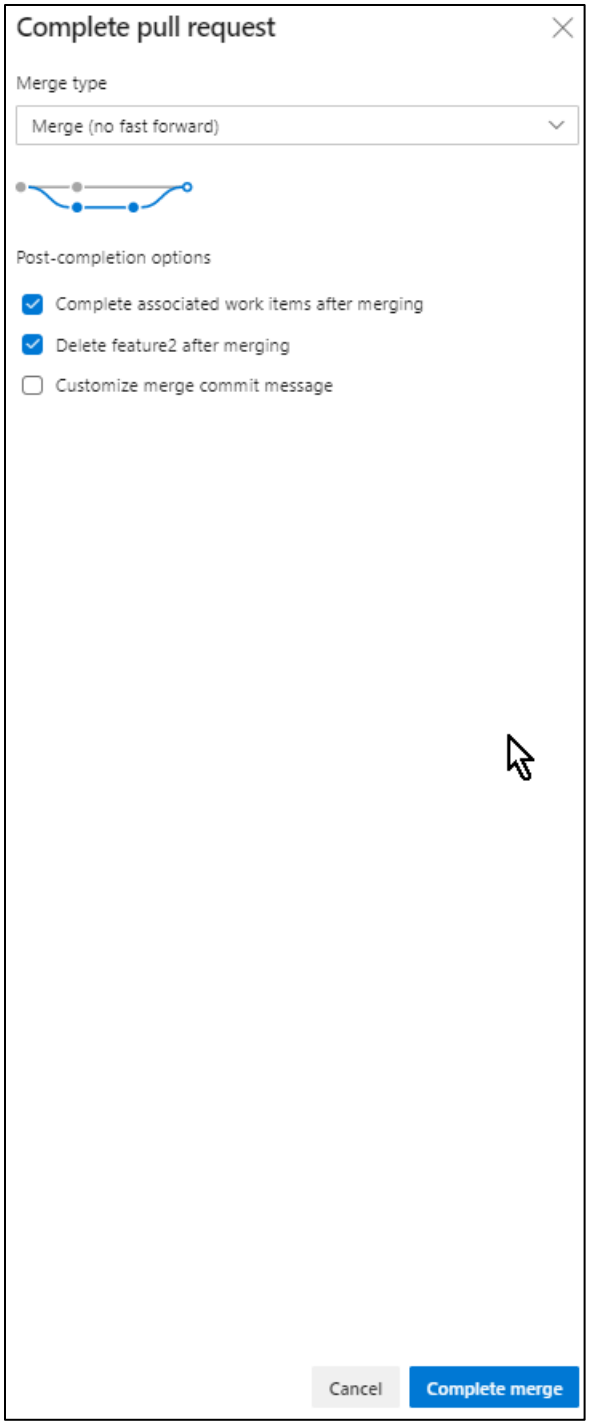
Give it a better name and click **Create**.



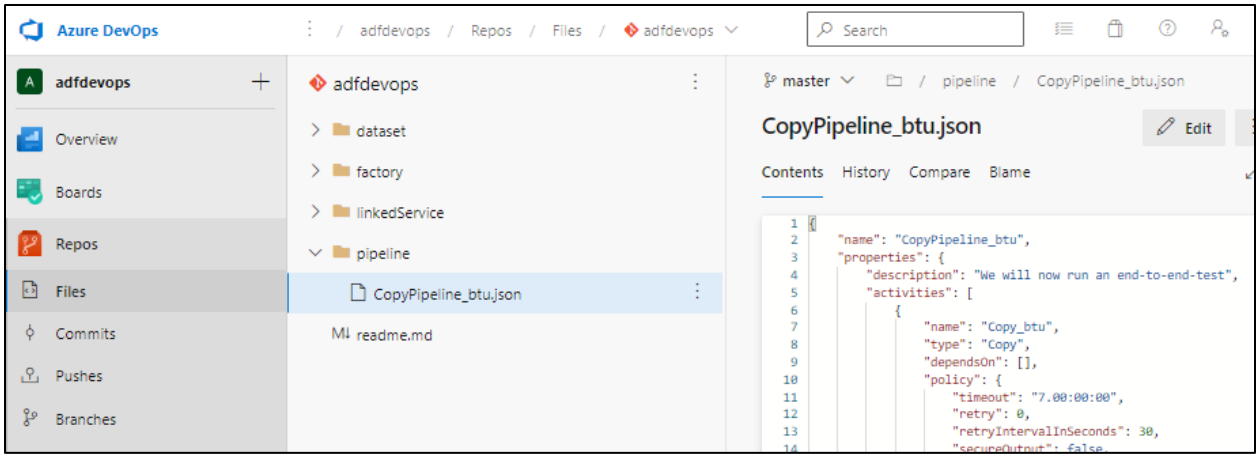
Approve request.

Click **Complete**.

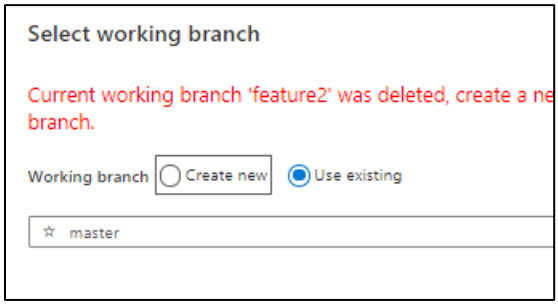
Click **Complete merge**.



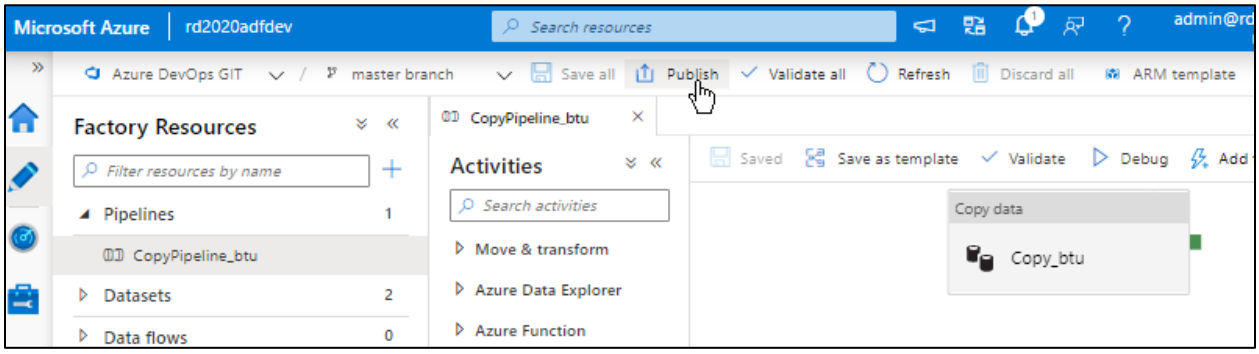
In the **master** branch, check the **CopyPipeline_btu.json** file, verify that the description has been updated.



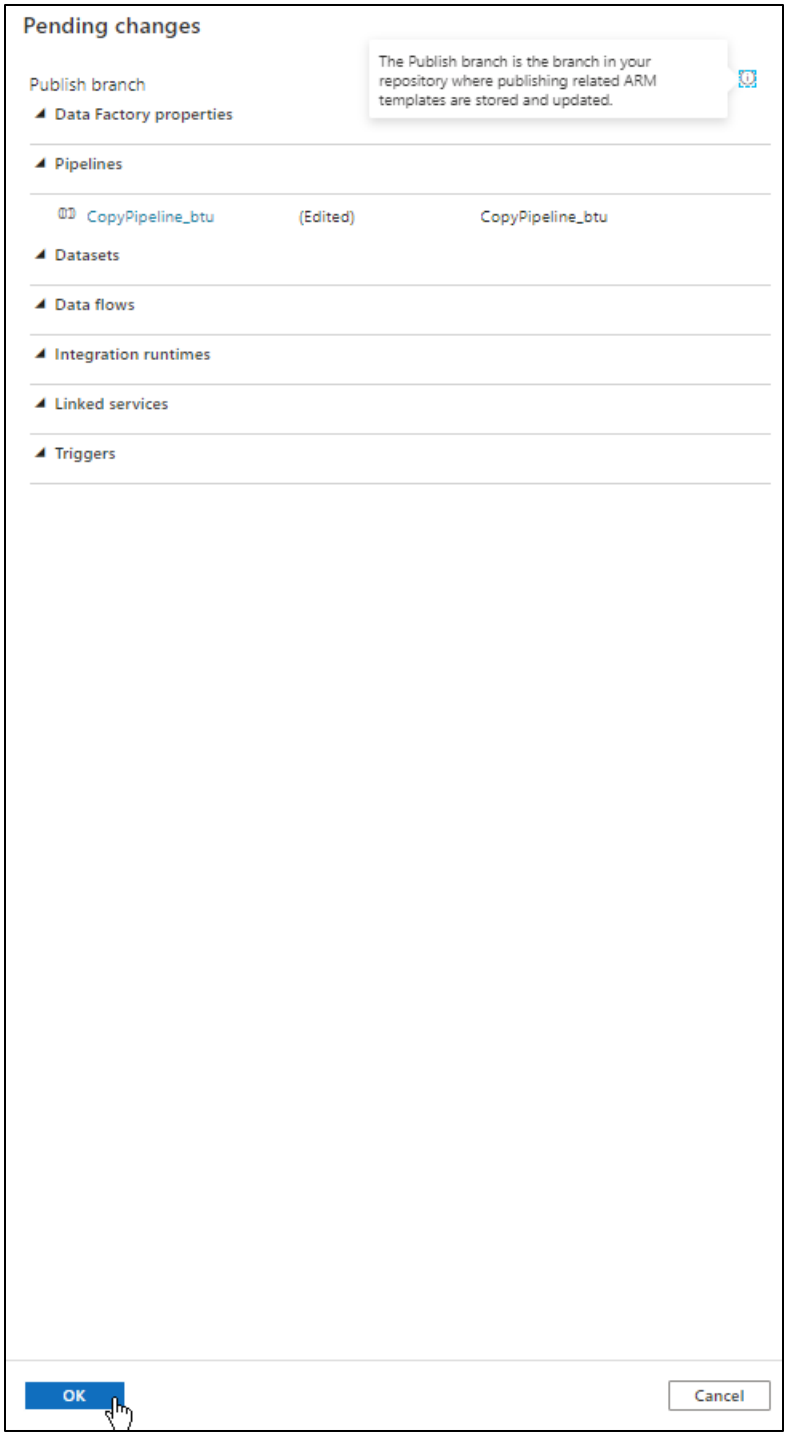
Go back to **ADF Dev**, indicate that you want to use master.



Click **Publish**.

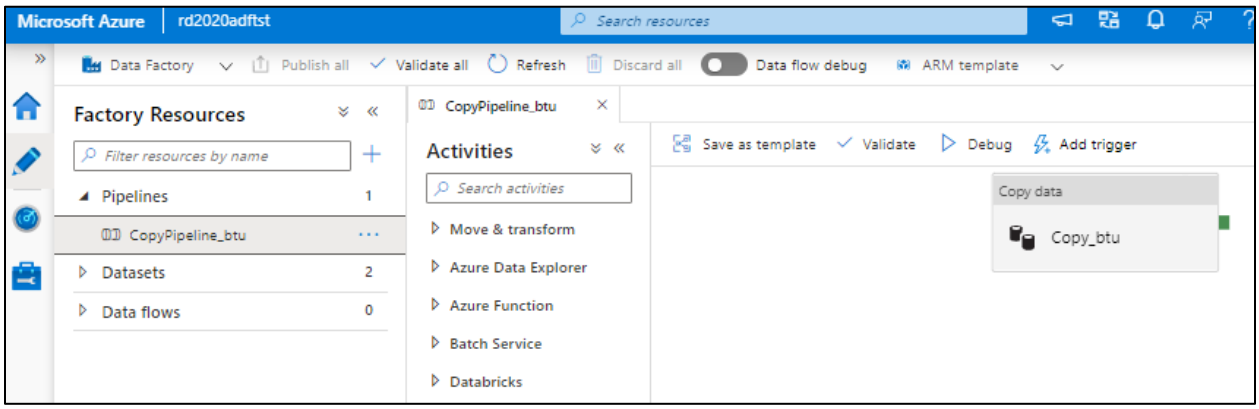


Click **OK**.


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

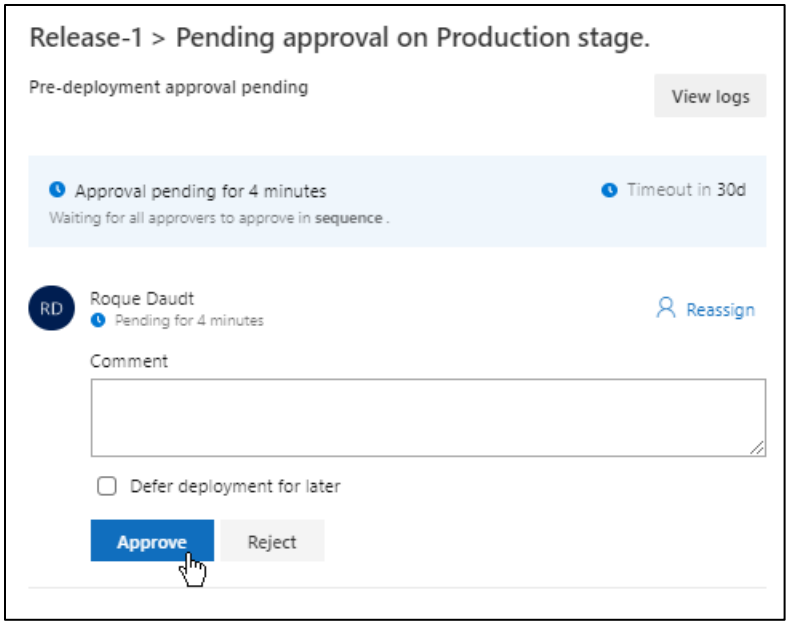
After a minute or so, go to Azure DevOps, select **Pipelines / Releases**. Check the pipeline. Note that it succeeded in Test and it is waiting for approval for Production.

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



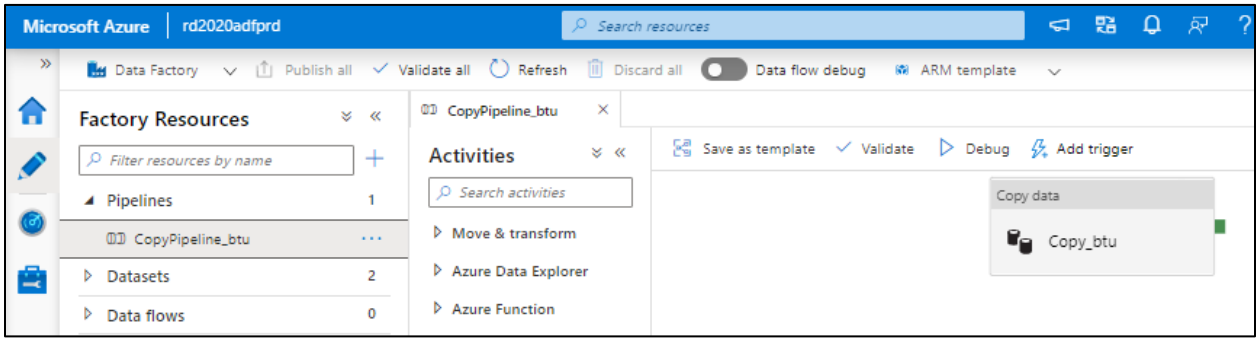
Go back to Azure DevOps. Click **Production**.

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.

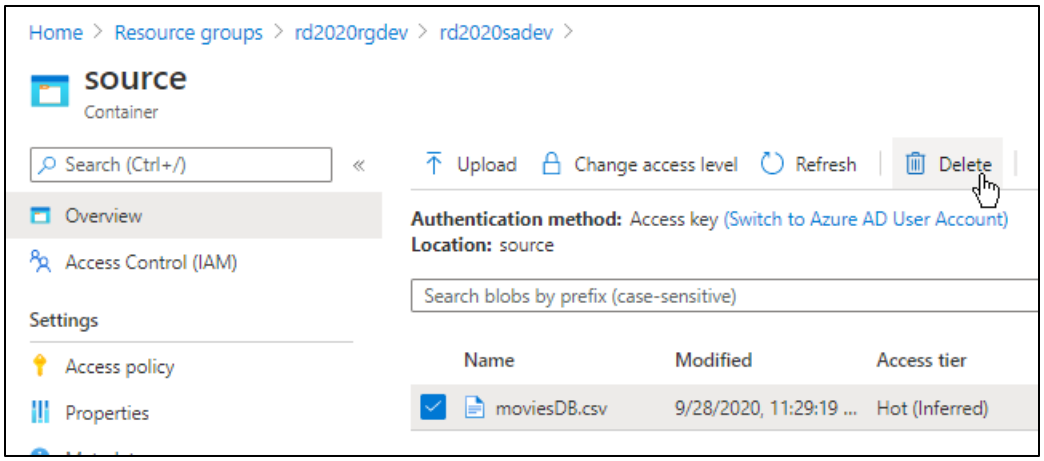


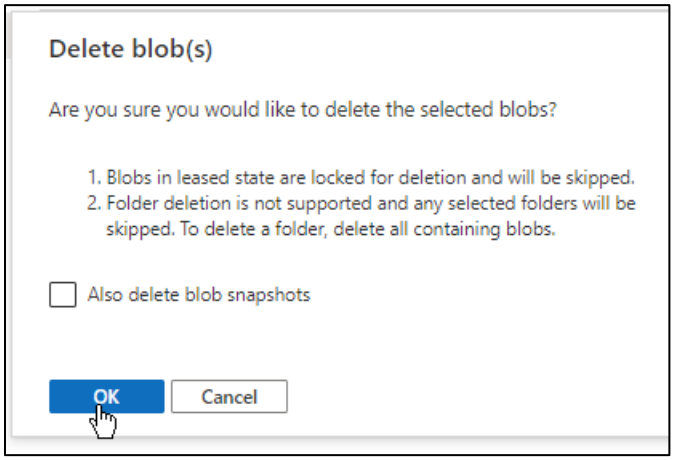
Test the pipeline in ADF Test

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

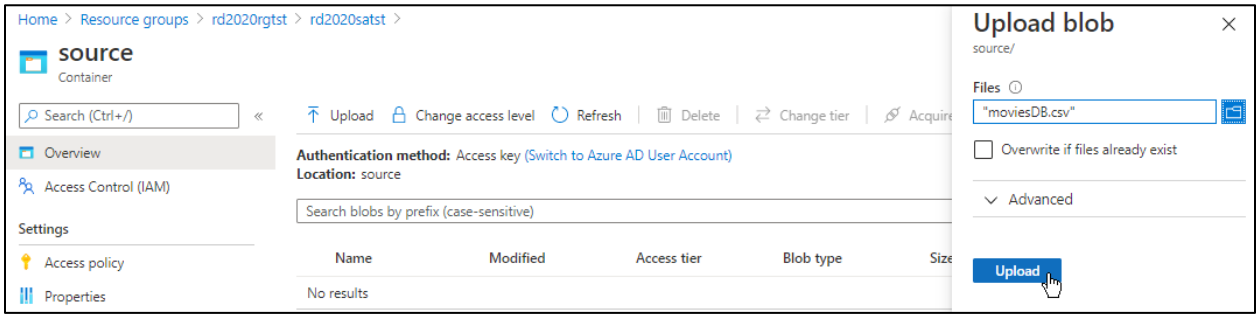
- Upload **movieDB.csv** file to the storage account in Test
- Remove **movideDB.csv** file from the storage account in Dev, to make sure that each environment is using its own resources.

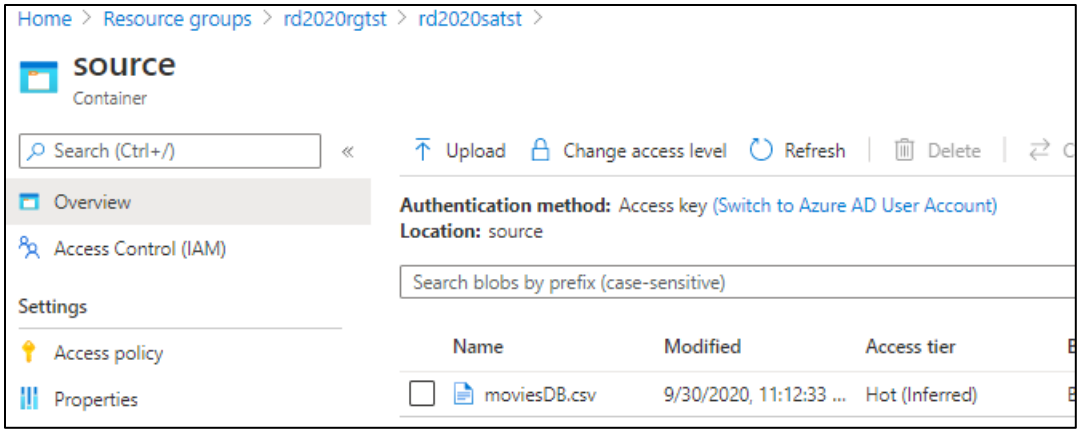
Open the storage account in dev, open the default container, go to the source folder and delete the file.



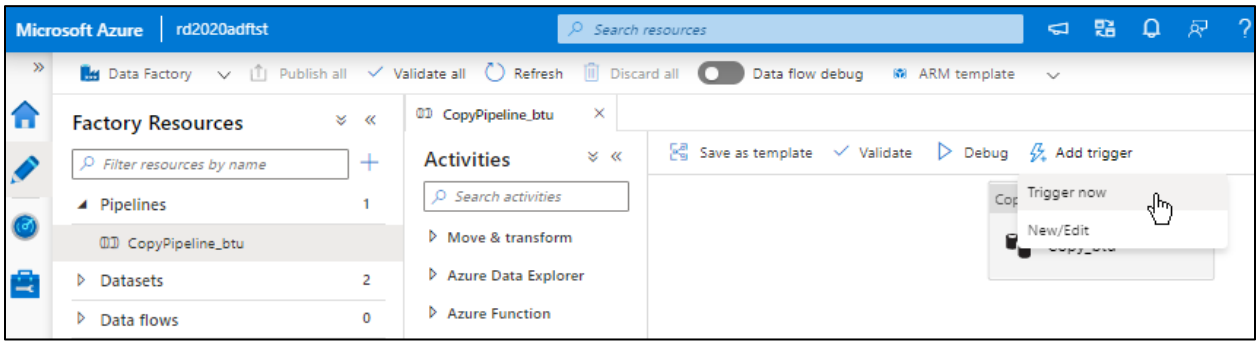


Go to the storage account in test and upload the file.



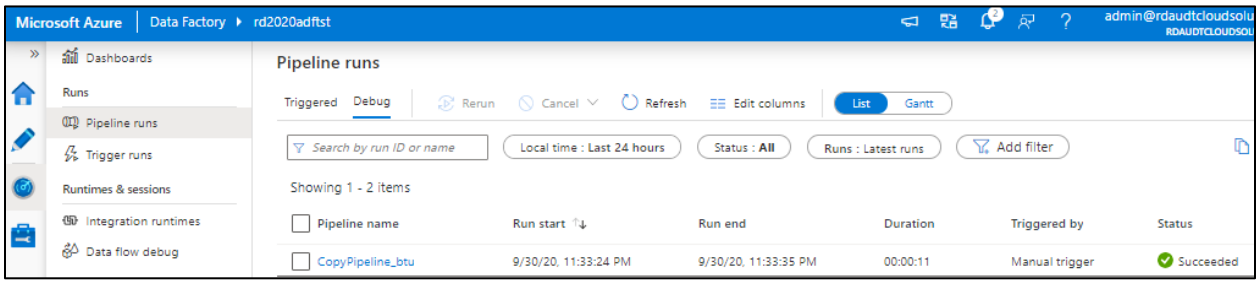


Go to **ADF Test**, open the pipeline and click **Trigger now**.

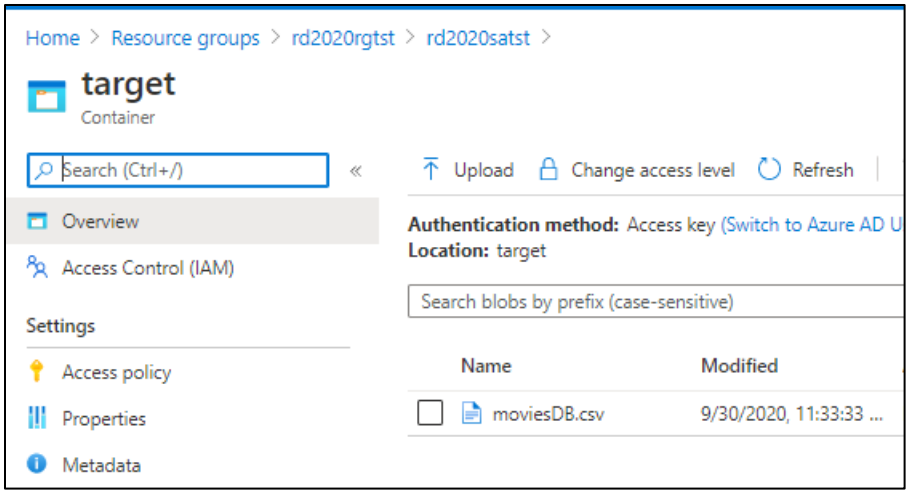


Click **OK** in the **Pipeline run** blade.

Monitor the progress, verify that it succeeded.



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



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**.