Source Control in Azure Data Factory

This post is part 18 of 25 in the series Beginner's Guide to Azure Data Factory

Raise your hand if you have wondered why you can only *publish* and not *save* anything in Azure Data Factory 🙋 Wouldn't it be nice if you could save work in progress? Well, you can. You just need to set up source control first! In this post, we will look at why you should use source control, how to set it up, and how to use it inside Azure Data Factory.

And yeah, I usually recommend that you set up source control *early* in your project, and not on day 18... However, it does require some external configuration, and in this series I wanted to get through the Azure Data Factory basics first. But by now, you should know enough to decide whether or not to commit to Azure Data Factory as your data integration tool of choice.

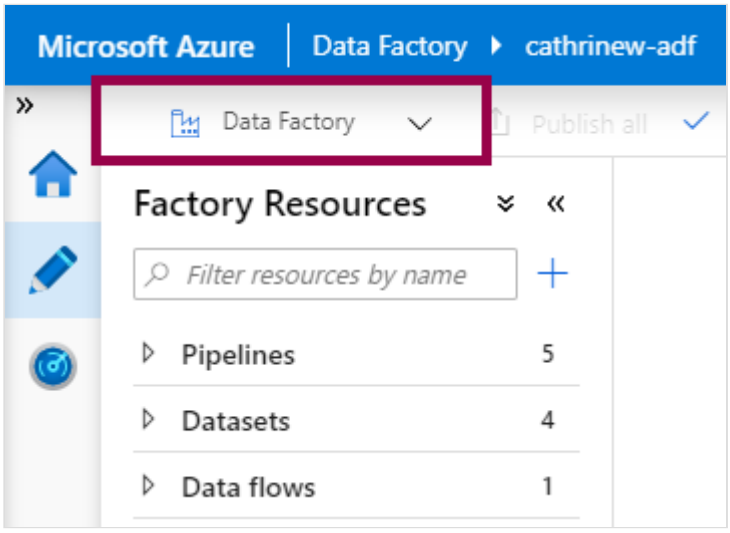
Get it? Commit to Azure Data Factory? Source Control? Commit? 🤓

Ok, that was *terrible*, I know. But hey, I've been writing these posts for 18 days straight now, let me have a few minutes of fun with Wil Wheaton 😂

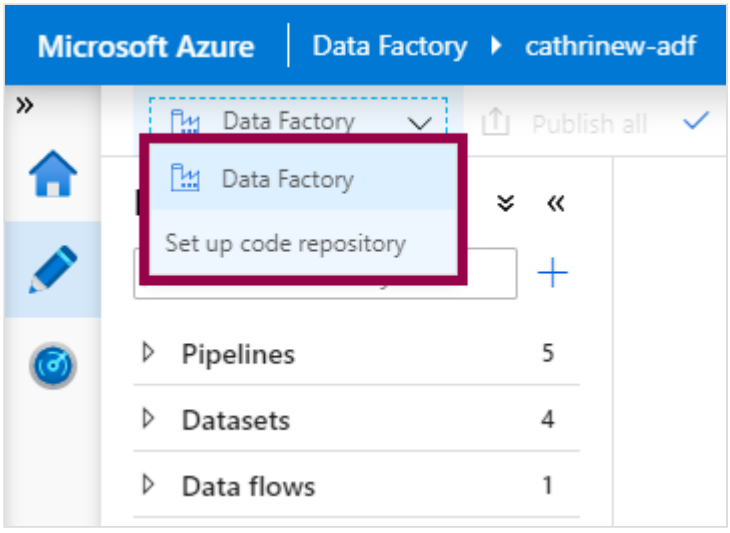
Aaaaanyway!

Authoring Modes in Azure Data Factory

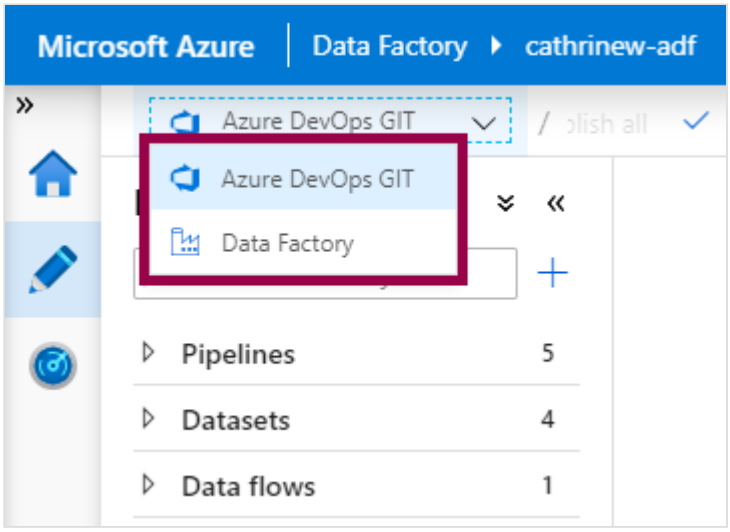
So far, we've been working in the Azure Data Factory mode:



If we haven't set up source control yet, we can do that from the authoring mode menu:



But once we *have* set up source control, we can switch between the Azure Data Factory mode and the Source Control mode:



But what's the difference between these two modes?

Azure Data Factory Mode

When I compare the two authoring modes, I usually refer to the **Azure Data Factory mode** as the "*production mode*". In this mode, you have to **publish to save**, and that requires everything to validate first. That's because when you publish, you deploy your solution from the user interface to the Azure Data Factory service. Or the way I think about it, you deploy "*into production*".

Source Control Mode

Just like I refer to the Azure Data Factory mode as "production mode", I refer to the **source control mode** as "*development mode*". In this mode, you add an additional step to your development process. First, you **save** your changes in the source control repository, and then you **publish** from the source control repository to the Azure Data Factory service.

Saving vs. Publishing

We can illustrate saving and publishing using the Azure Data Factory mode and the source control mode like this:

By using source control in Azure Data Factory, you get the option to save your work in progress. This is because all you're really doing is saving the JSON files behind the scenes to the code repository :)

Source Control Options

If you click the **set up code repository** button, the **repository settings** pane will open, and you can choose the **repository type**:

You can choose either **Azure DevOps Git** or **GitHub**. From here, I will assume that you already have one of these accounts and have the rights to create new projects and repositories :)

Azure DevOps

> First, let's go through how to set up an Azure DevOps code repository, connect our Azure Data Factory to it, then create and save and publish a new dataset. I'm assuming that your user has access to both Azure Data Factory and Azure DevOps.

Warning! There be screenshots. Many, many, many screenshots 🤓

Creating an Azure DevOps Code Repository

First, log into Azure DevOps and choose the organization. I have one called **cathrinew-devops**. Create a **new project**:

Go to **repos** -> **files**:

A Git code repository must always contain at least one file. Create (**initialize**) the code repository by adding a README file to it:

We now have our **empty code repository**, ready to go! (*I'm going to ignore the friendly TODO instructions on what to add to my README file for now. But in a real project, I would totally listen to the smart advice and add helpful explanations and descriptions* 😉)

Connecting to an Azure DevOps Code Repository

Back in Azure Data Factory, click through the settings and specify the **Azure DevOps account**, **project name**, and **git repository name**. I always use **master** as the collaboration branch, and keep **/** as the root folder. Then, add the existing pipelines, datasets, and so on to the code repository by checking **import existing Azure Data Factory resources** to the **collaboration** branch:

From now on, whenever you open Azure Data Factory, you will have to choose a **branch** to work in. Notice the new **save all** button, YAY!

If we switch back to Azure DevOps and refresh the code repository, we will see that all the **imported Azure Data Factory resources**:

We don’t want to work directly in the master branch, though. Let’s create a **new branch**:

I like to **name** my branches after the feature I'm working on. In this example, I want to create a dataset for the sets.csv file:

After we have created our new dataset, we can **save all** or **save** the dataset:

Woop woop! Saved!

But if we try to **publish**, we will be told that we can only **publish from master**, from the collaboration branch. This is a good thing thing! This ensures that everything has to be working in master before we can publish to the Azure Data Factory service:

Creating an Azure DevOps Pull Request

When you click on merge the changes to master, you will be taken back to Azure DevOps, where you can create a new **pull request**. This will merge the changes from the *sets* branch into *master*:

Once the pull request has been created, you can **complete** it. Ideally, you want someone else to review and complete it, but let's just pretend you're a coworker for now :)

If you are done with developing the feature, you can also choose to **delete the branch**:

Tadaaa! We have **completed** our first pull request:

When we switch back to Azure Data Factory, we will be asked to choose a **working branch**, since the *sets* branch was deleted. Let's choose *master*:

Publishing from Master

Now, we can **publish**:

But! And this is cool :) Instead of just publishing, we can now see **what** is getting published, and whether it's new, edited, or deleted:

Tadaaa! We have **published** from the collaboration branch:

But… what if we prefer working with GitHub? Or what if we want to change the code repository? We can easily do that :)

GitHub

> Next, let's go through how to set up a GitHub code repository, connect our Azure Data Factory to it, then create and save and publish another dataset. I'm assuming that your user already has a GitHub account.

Creating a GitHub Code Repository

First, log into GitHub and create a new **code repository**:

We now have our **empty code repository**, ready to go!

Disconnecting from an Existing Code Repository

Next, we need to disconnect from the Azure DevOps code repository. If you are starting from scratch with GitHub, you can skip this part :) Go to the *Home* page and click on **Git repo settings**:

Click **remove Git**:

Always read the warnings! :) Your Azure DevOps code repository will *not* be deleted, but you should *publish* all changes from it before disconnecting. Type the name of your Azure Data Factory and click **Confirm**:

Connecting to a GitHub Code Repository

Click **set up code repository**:

Choose **GitHub** and then log into your GitHub account:

Specify the **GitHub account** and **git repository name**. I always use **master** as the collaboration branch, and keep **/** as the root folder. Then, add the existing pipelines, datasets, and so on to the code repository by checking **import existing Azure Data Factory resources** to the **collaboration** branch:

We are now connected to the GitHub code repository, woohoo!

If we switch back to GitHub and refresh the code repository, we will see that all the **imported Azure Data Factory resources**:

Let’s create another **new branch**:

After we have created the new dataset, we can create a **pull request**:

Creating a GitHub Pull Request

When you create a new pull request, you will be taken back to GitHub. This will merge the changes from the *colors* branch into *master*. Compare the changes, and click **create pull request**:

Review the pull request, and click **create pull request**:

Once the pull request has been created, you can **merge** it. Ideally, you want someone else to review it first, but let's just pretend you're a coworker for now :)

You can **delete** the branch as well:

Back in Azure Data Factory, you can do the whole publishing loop again :)

Summary

In this post, we looked at why you should use source control, how to set up source control using Azure DevOps and GitHub, and how to use it inside Azure Data Factory.

When we set up source control, you may have noticed another new thing pop up in the interface…

Guess what we will look at in the next post? Yep! Templates!

← Executing SSIS Packages in Azure Data Factory

Templates in Azure Data Factory →

Share?

Related

Executing SSIS Packages in Azure Data Factory

Copy Data Activity in Azure Data Factory

Templates in Azure Data Factory

Dec 18, 2019 Data Platform Azure Data Factory

About the Author

Cathrine Wilhelmsen is a Microsoft Data Platform MVP, BimlHero Certified Expert, Microsoft Certified Solutions Expert, international speaker, author, blogger, and chronic volunteer who loves teaching and sharing knowledge. She works as a Senior Business

Intelligence Consultant at Inmeta, focusing on Azure Data and the Microsoft Data Platform. She loves sci-fi, chocolate, coffee, craft beers, ciders, cat gifs and smilies :)

Find me!

Subscribe to new posts?

E-mail

Yes, subscribe me!

Recent Posts

Sneaking back in… (2020 edition)

Keyboard shortcuts for moving text lines and windows (T-SQL Tuesday #123)

Speaking at NIC 2020

Azure Data Factory Training Day at SQLBits 2020

Personal Highlights from 2019

Popular Posts

Table Partitioning in SQL Server - The Basics

Preparing for and Taking Microsoft Exam DP-200 (Implementing an Azure Data Solution)

Parameters in Azure Data Factory

Table Partitioning in SQL Server - Partition Switching

Custom Power BI Themes: Page Background Images

Top Tags

Azure Data Factory Biml Certifications Don't Repeat Yourself Microsoft Ignite Notepad++ PASS Summit Personal Precon Speaking SQLBits SQLFamily SQLHangout SQLSatOslo SQLSaturday SQL Server SSIS T-SQL Tuesday Volunteering Webinar

All Categories

Select Category

Full Archive

Select Month