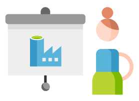
cathrine WILHELMSEN

Copy SQL Server Data in Azure Data Factory

This post is part 16 of 25 in the series Beginner's Guide to Azure Data Factory



In the previous post, we looked at the three different types of integration runtimes. In this post, we will first create a self-hosted integration runtime. Then, we will create a new linked service and dataset using the self-hosted integration runtime. Finally, we will look at some common techniques and design patterns for copying data from and into an on-premises SQL Server.

And when I say "on-premises", I really mean "in a private network". It can either be a SQL Server on-premises on a physical server, or "on-premises" in a virtual machine.

Or, in my case, "on-premises" means a SQL Server 2019 instance running on Linux in a Docker container on my laptop 🤓

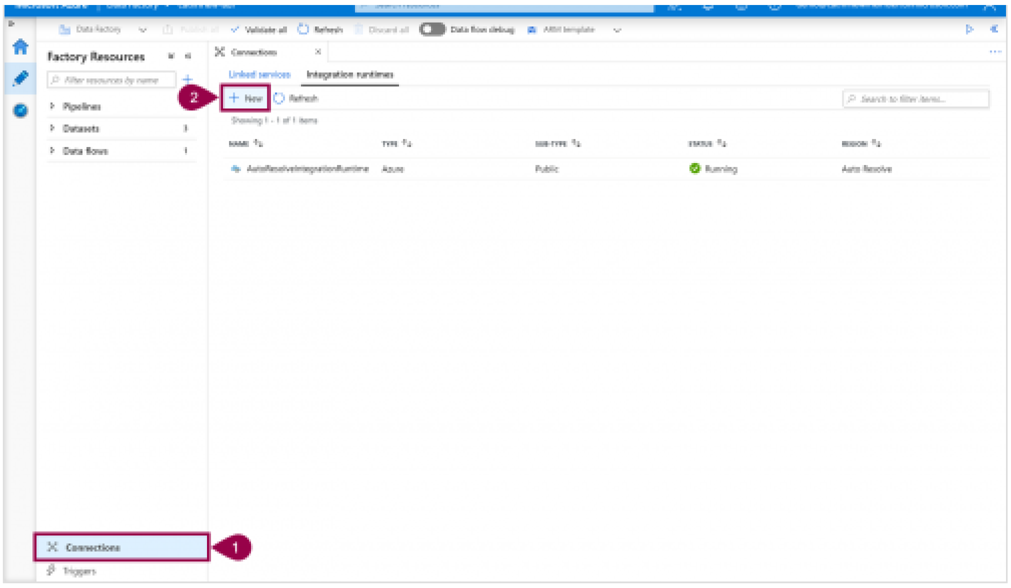
Creating a Self-Hosted Integration Runtime

There are two parts to creating a self-hosted integration runtime. First, you create the integration runtime in Azure Data Factory and download the installation files. Then, you install and configure the integration runtime on a computer in the private network.

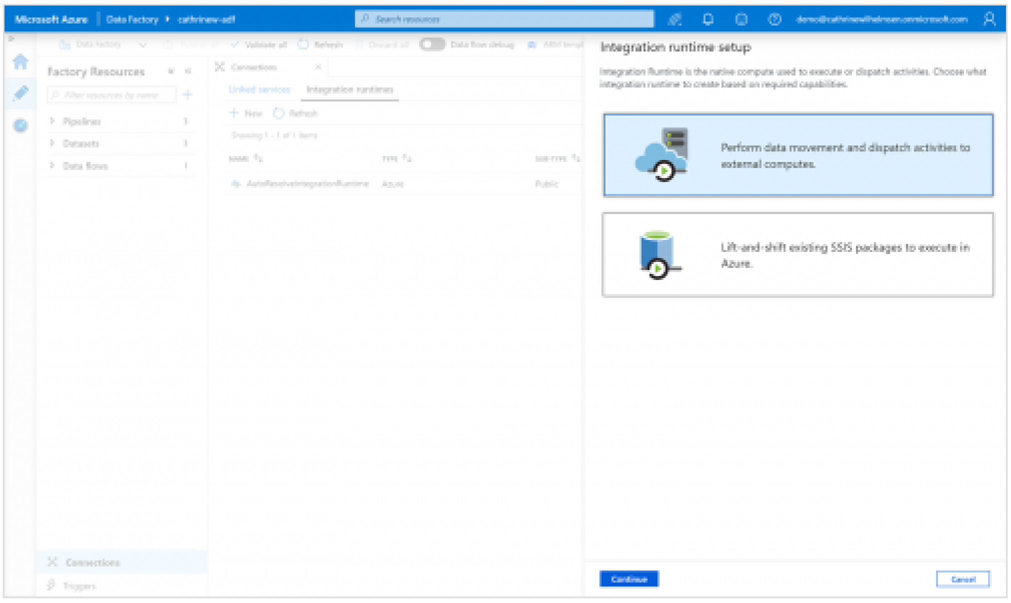
After that, the integration runtime works like a secure gateway so the Azure Data Factory can connect to the SQL Server in the private network.

Create and Download the Self-Hosted Integration Runtime

Open connections, click on integration runtimes, then click **+ new**:



Select “**perform data movement and dispatch activities**“:



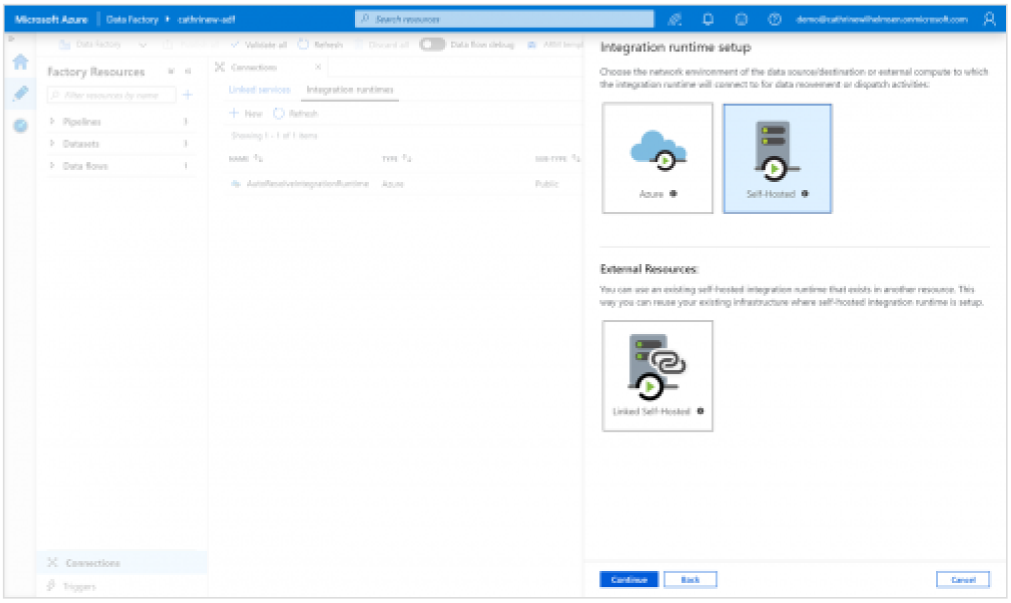
Integration runtime setup

Integration Runtime is the native compute used to execute or dispatch activities. Choose what integration runtime to create based on required capabilities.

Perform data movement and dispatch activities to external computes.

Lift-and-shift existing SSIS packages to execute in Azure.

Select the **self-hosted integration runtime**:



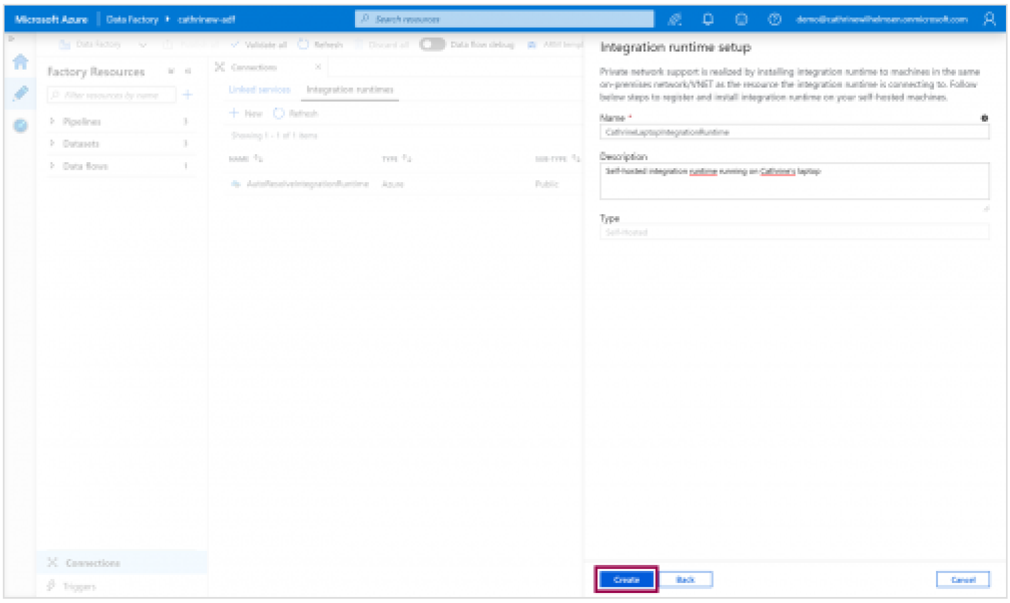
Integration runtime setup

Choose the network environment of the data source/destination or external compute to which the integration runtime will connect to for data movement or dispatch activities:

External Resources:

You can use an existing self-hosted integration runtime that exists in another resource. This way you can reuse your existing infrastructure where self-hosted integration runtime is setup.

Give the new integration runtime a **name** and **description**, then click **create**:



Integration runtime setup

Private network support is realized by installing integration runtime to machines in the same on-premises network/VNET as the resource the integration runtime is connecting to. Follow below steps to register and install integration runtime on your self-hosted machines.

Name *: CathrineLaptopIntegrationRuntime

Description: Self-hosted integration runtime running on Cathrine's laptop

Type: Self-Hosted

The integration runtime is immediately created and saved:

Next, you need to download the integration runtime installation file(s). If you are currently working from the computer you want to install the integration runtime on, you can choose **option 1: express setup**:

Otherwise, you can download the setup files manually from the Microsoft Download Center by choosing **option 2: manual setup**:

After you have downloaded the installation files, you need to install the integration runtime.

Install and Configure the Self-Hosted Integration Runtime

You have two options for installing the integration runtime.

Option 1: Express Setup

The **express setup** is exactly that, an *express* setup. It will download, install, and register the integration runtime in as few steps as possible:

The express setup is fast, but it also means that you can't customize settings like the installation location or the node name. It will automatically use your computer name as the node name:

Option 2: Manual Setup

The **manual setup** is also exactly that, a *manual* setup. You download the installation files, and then click through the installation like with any other software:

If you install the integration runtime on a computer that is configured to enter sleep or hibernate mode, you will get a warning:

(I got this warning because I'm installing the integration runtime on my laptop. Doing that works for simple demos and posts like these, but it's… errr… not exactly best practice for enterprise solutions 😅)

After you have installed the integration runtime, you need to configure it by using one of the keys that were previously displayed in the Azure Data Factory interface:

You can also customize the node name:

The integration runtime will use your chosen name as the node name:

Monitor the Self-Hosted Integration Runtime

Back in the Azure Data Factory interface, under *Nodes* , we can see the new CathrineLaptop node that we just installed:

On the *integration runtimes* page, we can see that our new self-hosted integration runtime is running. Click on the **monitor button** for more details:

On the *Monitor* page, we can see more details, like the available memory and CPU utilization:

Alright, let's create a new linked service and dataset!

Connecting to the Self-Hosted Integration Runtime

We have already covered how to create linked services and datasets, so I'll keep this part brief :)

Create a new **SQL Server linked service** and connect via the new **self-hosted integration runtime**:

Then, create a new **SQL Server dataset** and connect via the new linked service:

Tadaaa! :) Are you ready to copy some data?

I am! :D

Copy From SQL Server

When we went through datasets, we explained how Azure SQL Database datasets can be used *directly* or as a *bridge to the linked service* in the copy data activity. This works exactly the same way when working with SQL Server datasets.

We can choose to use either the **table** defined in the dataset, a **query** that overrides the dataset, or a **stored procedure** that overrides the dataset:

Since we have already covered how this works, I won't repeat it here. Scroll down to "Database Datasets… or Queries?" if you need a refresher :D

Copy Into SQL Server

Copying *into* SQL Server is a new use case, however. Previously, we have used a mapping data flow to copy data into an Azure SQL Database. But since data flows currently only support cloud data stores, they can't be used to copy data into SQL Server. To do that, we need to use the copy data activity.

In this part, we will look at some techniques and design patterns for copying data into SQL Server.

Load (Append New Rows)

The default behavior of the copy data activity is to *append new rows*. You only have to specify the **sink dataset**:

This works well if you are sure that your source only returns new data, so you don't risk inserting duplicate data or get primary key constraint violations.

Truncate and Load (Overwrite All Rows)

If you want to *overwrite all rows*, you have to specify the **sink dataset** and add a **TRUNCATE TABLE** statement in the **pre-copy script**:

This works well if you always need the latest data, and the dataset is small enough that loading times is not an issue.

Load and **Truncate and Load** are simple patterns. The next one… not so much :)

Load Into Stored Procedure (Apply Custom Logic)

If you want to apply any kind of *custom logic* when loading into SQL Server, for example *inserting new rows and updating existing rows (upserting data)*, you have to use a stored procedure.

In this case, instead of loading data directly into a *table*, you load data into a *stored procedure*. The stored procedure then does whatever you tell it to do. In addition to the stored procedure, you also need to create a *table type* to represent the source data.

…yeah. I know. Whaaat…? This took me a while to wrap my head around :)

Let's walk through an example!

To load into a stored procedure, you specify the **sink dataset**, **stored procedure name**, **table type**, and **table type parameter name**:

In this case, the dataset is only used as a *bridge* to the linked service, so we can call the stored procedure:

The stored procedure can look something like this:

```
CREATE PROCEDURE lego.usp_upsert_themes
    @themes lego.themes READONLY
AS
BEGIN

    MERGE lego.themes AS target
    USING @themes AS source
    ON (target.id = source.id)
    WHEN MATCHED THEN
        UPDATE SET
            target.name = source.name
           ,target.parent_id = source.parent_id
    WHEN NOT MATCHED THEN
        INSERT (
            id
           ,name
           ,parent_id
        )
        VALUES
```

```
            (source.id, source.name, source.parent_id);

END;
```

The stored procedure uses a MERGE statement to insert new rows and update existing rows. It has one **parameter** called **@themes**, which takes a **table type** called **lego.themes** as an input.

The table type and the parameter name from the stored procedure are wired up in the copy data activity:

The table type definition can look something like this:

```
CREATE TYPE lego.themes AS TABLE (
    id INT NULL
   ,name NVARCHAR(40) NULL
   ,parent_id INT NULL
);
```

The table type definition **must** have exactly the same columns in the same order as the **source data definition**:

Because here's the magic… The copy data activity reads the **source data**, maps the source data to the **table type**, and passes the table type into the **stored procedure**. The stored procedure then uses the data wrapped in the table type to **insert and update** the rows.

It makes total sense… once you understand how it works :D

Summary

In this post, we first created a self-hosted integration runtime. Then, we looked at some common techniques and design patterns for copying data from and into an on-premises SQL Server.

In the next post, we will get an overview of SSIS in Azure Data Factory.

← Integration Runtimes in Azure Data Factory

Executing SSIS Packages in Azure Data Factory →

Share?

Twitter LinkedIn Email More

Related

Integration Runtimes in Azure Data Factory

Overview of Azure Data Factory Components

Linked Services in Azure Data Factory

Dec 16, 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