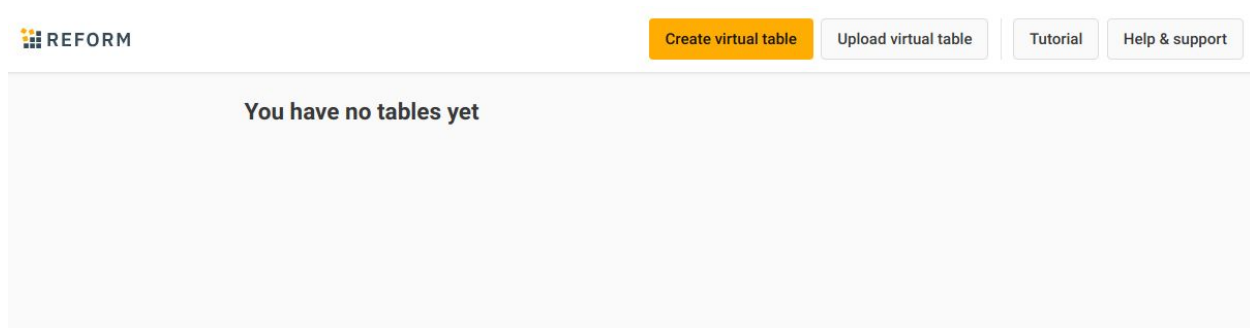


Ingesting JSON as analytic ready tables into SQL Server

The REFORM platform enables data analysts and engineers to access complex JSON data as tables. In many cases we want those tables to be accessible via Microsoft SQL Server. This guide will show you how this process can be easily automated. We start with a JSON API and via REFORM and Azure Data Factory we transform this data into a table in SQL Server.

Creating a virtual table in REFORM

REFORM uses a concept called *Virtual Tables* to transform JSON-formatted data into tables. To create a virtual table we click the *Create virtual table* button on the REFORM homepage.



First step of virtual table creation is data source definition. REFORM provides connectors for cloud storage services such as S3, Wasabi or Azure. Additionally you can also load data from MongoDB or a public URL. In this case we want to ingest information about SpaceX launches and parts reuse for further analysis. This data is accessible via an API on a public URL: <https://api.spacexdata.com/v3/launches>.

To create a data source we click the plus icon in the *Data Sources* column. In the configuration windows we provide a unique name, select *URL* as the connection type and make sure that the data format is set to *Array wrapped*. REFORM also supports LDJSON. We enter the API endpoint URL into the *URLS* box and click the *Add* button.

Add new datasource

NAME

Launches



CONNECTION TYPE

URL

DATA FORMAT

Array wrapped

COMPRESSION SCHEME

None

MAXIMUM NUMBER OF REDIRECTS

5

URLS

<https://api.spacexdata.com/v3/launches>

HEADERS

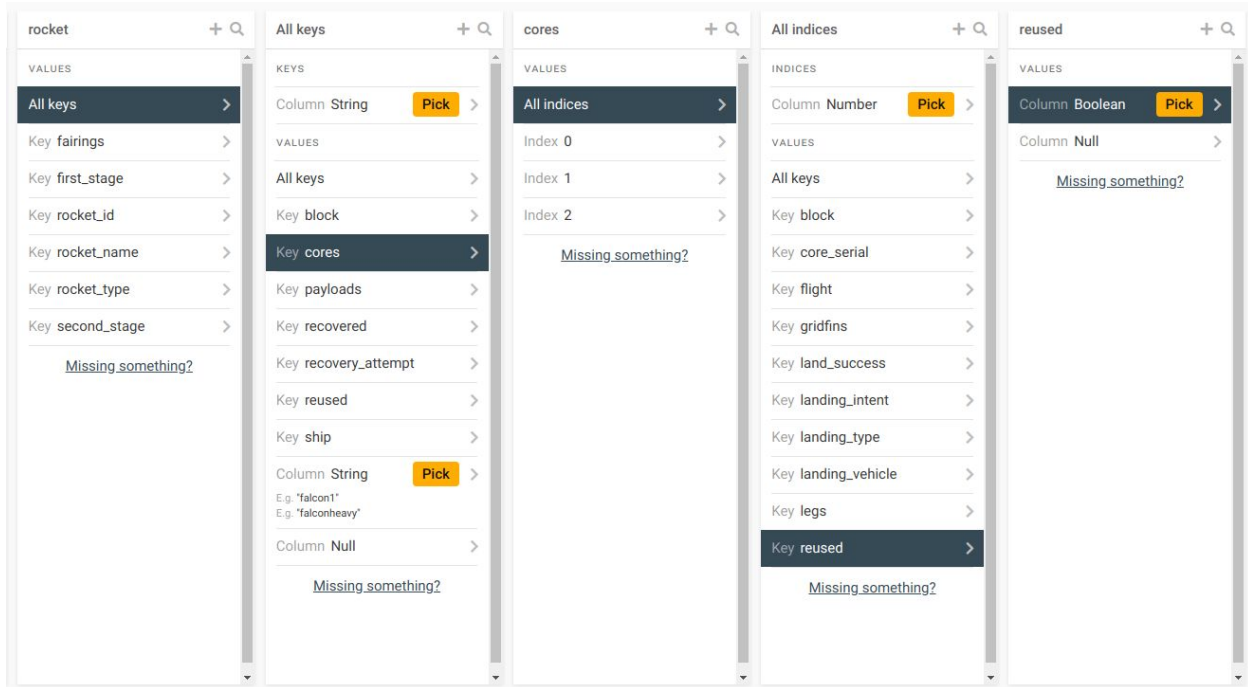
Name

Value

Cancel

Add

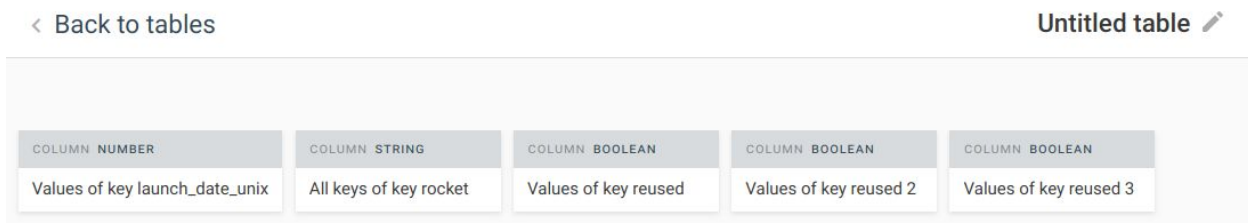
We can now browse the JSON data simply by clicking on desired keys or array indices. By clicking on the *Pick* button we can then add required values to the virtual table as columns.



For our table, we want to pick the following:

- launches -> launch_date_unix -> Number (time of the launch as UNIX timestamp)
- launches -> rocket -> All keys -> String (all rocket component names present as JSON keys)
- launches -> rocket -> All keys -> cores -> All indices -> reused -> Boolean -> As Number -> Number (This gives us 1 or 0 depending on whether the core was reused or not)
- launches -> rocket -> All keys -> payloads -> All indices -> reused -> Boolean -> As Number -> Number (Was the payload reused?)
- launches -> rocket -> All keys -> reused -> Boolean -> As Number -> Number (Was any part of the vehicle reused?)

After the selection we can see the table structure at the bottom of the screen.



The *Rename* button opens a dialog allowing us to change the column names to something more descriptive. After we change the column names we click the *Done* button, that takes us back to the virtual table editor.

Rename columns Cancel Done

Provide new names for columns and then press "Done".

COLUMN NUMBER	COLUMN STRING	COLUMN BOOLEAN	COLUMN BOOLEAN	COLUMN BOOLEAN
Date	Part	Rocket Reused	Payload Reused	Cores Reused

Finally we can change the name of the table itself and confirm the table definition by clicking on *Done*.

< Back to tables **Rocket Reuse** Done

Exclude Remove Detach Merge Filter Rename

COLUMN NUMBER	COLUMN STRING	COLUMN NUMBER	COLUMN NUMBER	COLUMN NUMBER
Date	Part	Cores Reused	Payload Reused	Part Reused

Once the virtual table is defined, we can access the live results as CSV using a sharing URL provided by REFORM.

Rocket Reuse

Download Edit as copy Edit table Archive

LIVE ACCESS

Anyone with the following streaming CSV link can access this table's live results

<https://aws-mp.qa.internal.slamdata.com/api/table/861047c3-c730-4498-a8c8-130bcf855dd3/live/dataset>

Creating a MSSQL table

Before we can insert the data into SQL Server, we need to create a suitable table. We do this by running the following SQL command:

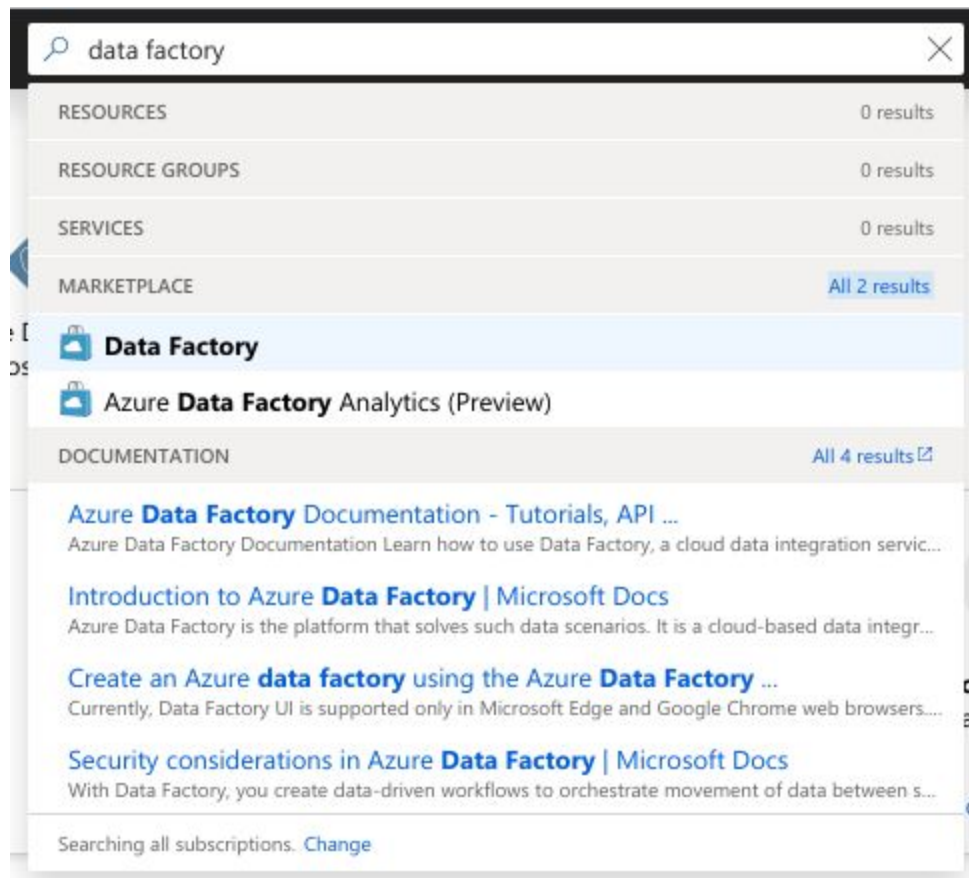
```
CREATE TABLE spacex (  
    date int,  
    part nvarchar(100),  
    cores_reused bit,  
    part_reused bit,  
    payload_reused bit  
)
```

Ingesting data using Azure Data Factory

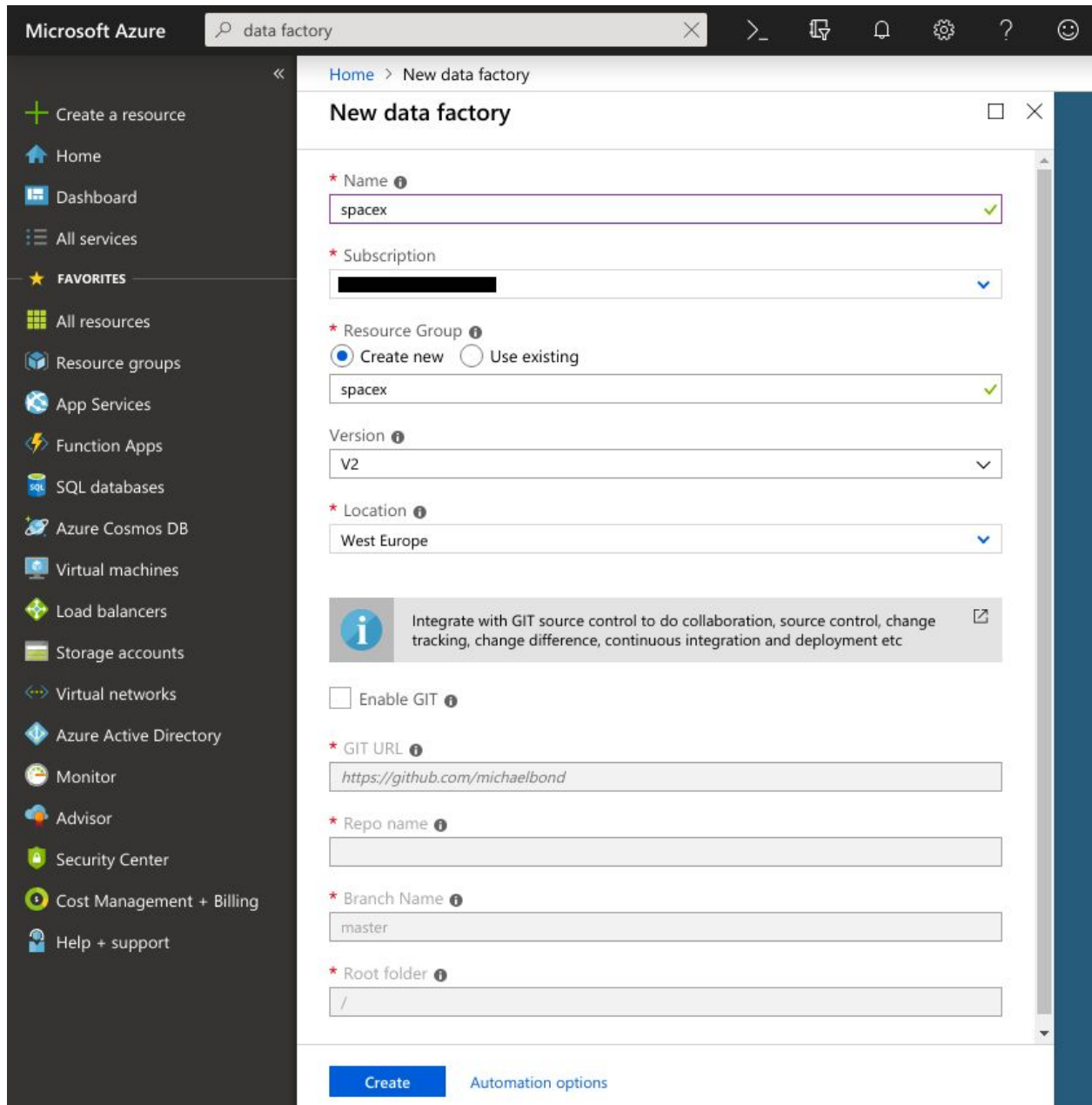
Azure Data Factory is a data integration tool developed by Microsoft. Besides many other great features, it provides a user-friendly way of moving data between multiple data sources. Although it is a part of Microsoft's Azure cloud, it is not limited to services hosted on the platform. You can use it easily with your on-premise services as well.

In this demo, we are storing our data in a Microsoft SQL Server database. But Azure Data Factory is not limited to this service. It also provides connectors for MySQL, PostgreSQL and many other popular solutions.

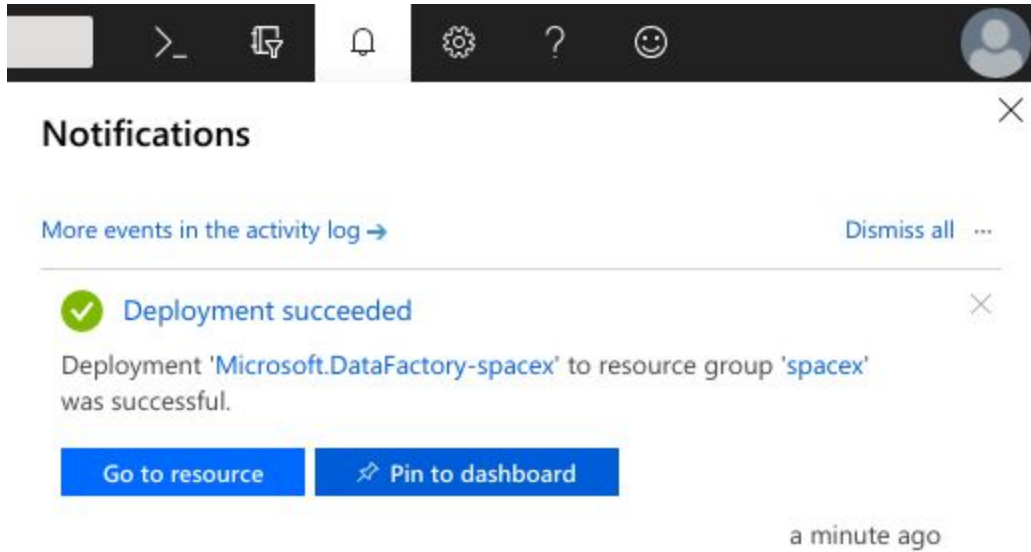
First we need to create a new data factory instance. To do this we log in into the [Azure Portal](#) and type "data factory" into the search bar. Then we select *Data Factory* from the list:



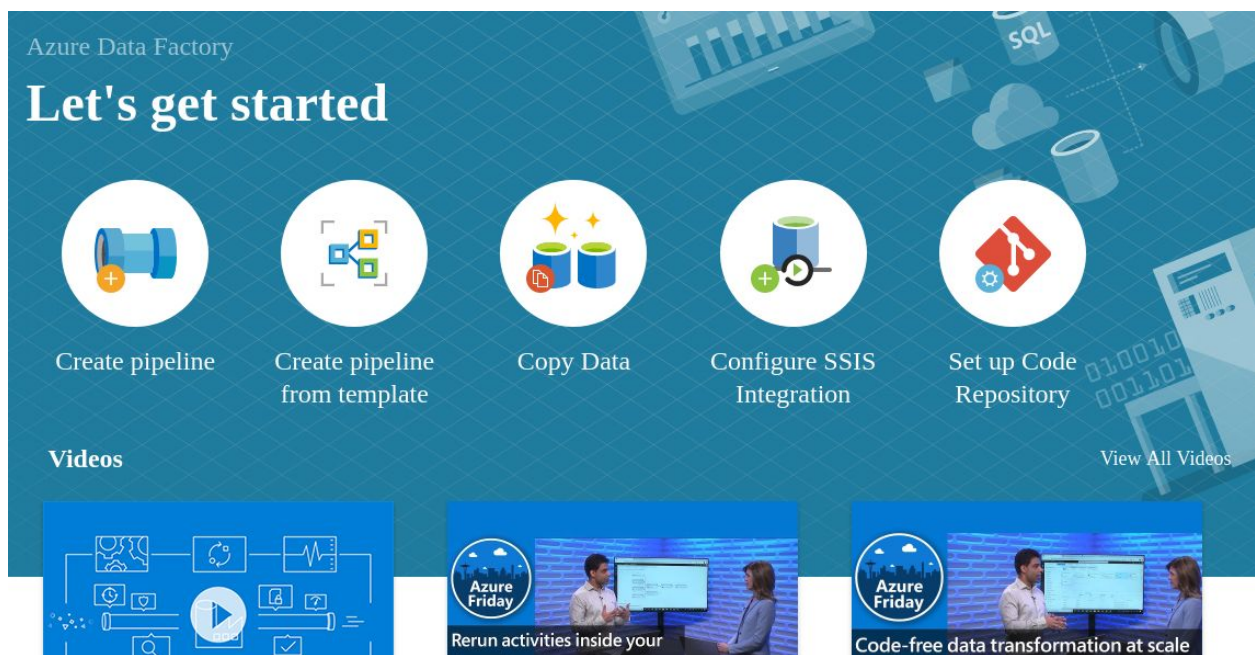
A form for creating new data factory shows up. We provide a name for our instance, select a suitable [resource group](#) and [subscription](#) plan. We are going to use the newer V2 version and choose a geographical location depending on our needs. GIT support is not required for our project. Finally, we click on the *Create* button to initialize our data factory.



We are redirected to the dashboard and after a few seconds a notification about a successful deployment appears. To access the data factory resource, we can click the *Go to resource* button.



On the data factory screen we click the *Author and Monitor* button, which takes us to the data factory web interface. After the application loads, we choose the *Copy Data* option from the *Let's get started* screen.



On the first screen of the *Copy Data* wizard we need to provide a unique name for the task. Optionally, we can set up a schedule. This will keep your SQL Server up to date with the latest data from the API. Once the schedule is set up in Data Factory REFORM will automatically provide most recent data when the process is executed.

1 Properties

2 Source

- Connection
- Dataset

3 Destination

- Connection
- Dataset

4 Settings

5 Summary

6 Deployment

Properties

Enter name and description for the copy data task.

Task name *

Task description

Task cadence or Task schedule

Run once now Run regularly on schedule

Trigger type *

Schedule Tumbling Window

Start Date (UTC) *

Recurrence *

Every

End *

No End On Date

Delay

Max Concurrency *

Retry Policy: Count

Retry Policy: Interval in seconds

For the purposes of the guide, we are going to run the task just once, so we select the *Run once now* option and click *Next*.








Next, we need to define the data source. On the *Source Data Store* screen we click the + *Create New Connection* option. In the dialog, we switch to the *File* tab and then select the *HTTP* option.

New Linked Service



Search

All Azure Database **File** Generic protocol NoSQL Services and apps

 Amazon S3	 FTP	 File System
 Google Cloud Storage (S3 API)	 HDFS	 HTTP
 SFTP		

Cancel Continue

In the configuration we provide a unique name for the data source and most importantly enter the URL. If you are certain that you will be copying only one table, you can simply enter the whole URL as provided by Reform. Otherwise, it is a good idea to enter only the part that stays the same for all tables: [https://\[reform-host\].com/api/table/](https://[reform-host].com/api/table/).

We also change the authentication type to *Anonymous* and click the *Finish* button.

← New Linked Service (HTTP) ×

Name *

Description

Connect via integration runtime * ⓘ

Base URL *

Server Certificate Validation ⓘ
 Enable Disable

Authentication type *

Annotations

▶ **Advanced** ⓘ

After our data source is defined, we can move on to the *Specify HTTP dataset properties* screen. Here we provide the relative URL. If you entered the whole table URL in the previous step, you can leave this field empty. However, if you followed our recommendation and entered only the unchanging part, now is the time to enter the table specific one. No other changes are needed on this screen, so we can just click *Next*.

Specify HTTP dataset properties

Relative Url

Request Method

Additional Headers

Binary copy

 ⓘ

Compression type

Request timeout

[Previous](#)[Next](#)

We continue by setting up the file format. The default *JSON format* needs to be changed to *Text format*. The first row of the table contains column names, so we also check the *Column names in the first row* checkbox. Afterwards we should be able to see the data preview at the bottom of the screen.

File format settings

File format ?

Text format ▼

Detect Text Format

Column delimiter

Comma (,) ▼

Use custom delimiter

Row delimiter

Carriage Return + Line feed (\r\n) ▼

Use custom delimiter

Skip line count ?

0

Column names in the first row

▶ Advanced

Preview	Schema	
Date	Part	Cores Reused
1143239400	rocket_id	
1143239400	rocket_name	
1143239400	rocket_type	
1143239400	first_stage	0
1143239400	second_stage	
1143239400	fairings	
1174439400	rocket_id	

Previous

Next

Now we switch to the *Schema* tab. Here we can define the data types of the columns. In our case, we perform just a few modifications - change the *flight_number* column to *Int16* and *launch_success* and *is_tentative* to *Boolean*. Then we continue to the next step by clicking on the *Next* button.

Preview		Schema	
+ New column		↻ Clear	🗑 Delete
<input type="checkbox"/>	COLUMN NAME	TYPE	
☰	Date	123 Int32	▼
☰	Part	abc String	▼
☰	Cores Reused	✓ Boolean	▼
☰	Payload Reused	✓ Boolean	▼
☰	Part Reused	✓ Boolean	▼






Now we need to define the destination connection. Similarly to the source data store, we start by clicking on the + *Create New Connection* button. In this case we select the *SQL server* option under the *Database* tab.

New Linked Service



Search

All Azure Database File Generic protocol Services and apps

 Azure Data Explorer (Kusto)	 Informix	 Microsoft Access
 Oracle	 SQL Server	

Next, we enter all the information needed to create a connection: server name (IP address or domain name), database name, username and password. Just to be sure, we can verify the configuration by clicking on the *Test connection button*. If everything works, we click the *Finish button*.

← New Linked Service (SQL Server)



Name *
cool_company_server

Description

Connect via integration runtime *
AutoResolveIntegrationRuntime

Connection String Azure Key Vault

Server name *
13.79.16.201

Database name *
slamdata

Authentication type
SQL Authentication

User name *
sa

Password Azure Key Vault

Password *
.....

✔ Connection successful

Cancel

Test connection

Finish

On the next screen we select the target table and click *Next*.

Table mapping

For each table you have selected to copy in the source data store, select a corresponding table in the destination data store or specify the stored procedure to run at the destination.

Source	Destination
HTTP file	→ [dbo].[spacex] ↕

Skip column mapping for all tables

[Previous](#) [Next](#)

Next, we define how the CSV columns are mapped to the database table. For each database column we simply select the corresponding CSV column from the list. If the column names are the same for both, Data Factory is able to map everything automatically. After the mapping is defined, we can move on to the next step.

Column mappings

<input checked="" type="checkbox"/> HTTP file		[dbo].[spacex]
<input checked="" type="checkbox"/>	Date (Int32)	→ date (Int32)
<input checked="" type="checkbox"/>	Part (String)	→ part (String)
<input checked="" type="checkbox"/>	Cores Reused (Boolean)	→ cores_reused (Boolean)
<input checked="" type="checkbox"/>	Part Reused (Boolean)	→ part_reused (Boolean)
<input checked="" type="checkbox"/>	Payload Reused (Boolean)	→ payload_reused (Boolean)

On the *Settings* screen we choose how to deal with incorrect rows. For our project, we want to *skip incompatible rows*.

Settings

More options for data movement

▲ Fault tolerance settings

Fault tolerance ⓘ

▲ Performance settings

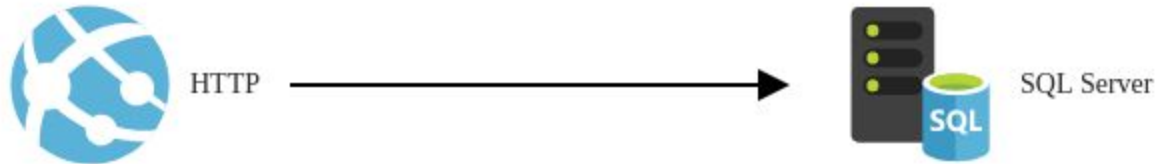
Enable staging ⓘ

▶ Advanced settings

Finally, on the *Summary* screen we can review all the settings and deploy the pipeline by clicking on the *Next* button.

Summary

You are running pipeline to copy data from HTTP to SQL Server.



Properties

Task name `copy_spacex_data_1`

Task description

Source

Connection name `spacex_reform`

Dataset name `SourceDataset_nr8`

Relative Url `ac9ec4a6-f1f5-480c-93dd-a4eeff5ab977/live/dataset`

Request Method `GET`

Destination

Connection name `cool_company_server`

Dataset name `DestinationDataset_nr8`

Table name `[dbo].[spacex]`

Copy settings

Timeout `7.00:00:00`

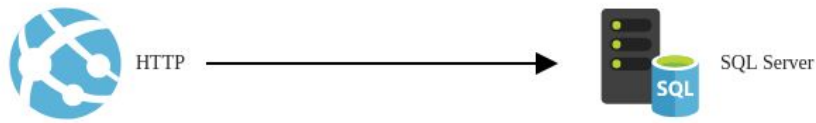
Retry `0`

Retry interval `30`

Secure output `false`

Secure input `false`

In our case, the pipeline executes immediately. We can monitor the progress by clicking on the *Monitor* button.



Deployment complete

- ▶ Creating Datasets ✓
- ▶ Creating Pipelines ✓
- ▶ Running Pipelines ✓

Datasets and pipelines have been created. You can now monitor and edit the copy pipelines or click finish to close the copy wizard.

[Edit Pipeline](#) [Monitor](#)

📅 Last 24 Hours 04/11/2019 9:19 AM - 04/12/2019 9:19 AM ▼

🌐 Time Zone (UTC+01:00) Belgrade, Bratislava, ... ▼

🔍 View All Rerun History

🔍 Filter

All Succeeded In Progress Queued Failed Cancelled

□ Pipeline Name ▼	Actions	Run Start ↕	Duration	Triggered By	Status	Parameters	A
copy_spacex_data_1	 	04/12/2019, 9:18:58 AM	00:00:06	Manual trigger	 In Progress		

After the process finishes, we can verify that the data is indeed stored in our database as a table, for example with Azure Data Studio:

