ETL Accelerators: How to avoid doing the boring work!

Michael Alkema

A data engineer who is moving from client to client encounters different levels of maturity in the design and execution of a data warehouse and its accompanying data pipelines. A company can still reside in a greenfield, whereby a project is new and starting up. But other clients are more brownfield, so they already have established data procedures and a certain way of working. With both green- and brownfield clients, a data engineer can spend a lot of time developing manual processes.

Considering greenfield projects, a data engineer has to put in the effort to build new ETL processes from scratch even though he might have done this multiple times already at other projects. It is incredibly tedious to repeat the work that you have done before or face the same issues that you already solved in a previous assignment. Brownfield projects come with a different set of challenges. You might have to work with existing pipelines that include some flaws that the company never had time to fix. Moreover, how can you, as a data engineer, bring added value and innovation to such a client?

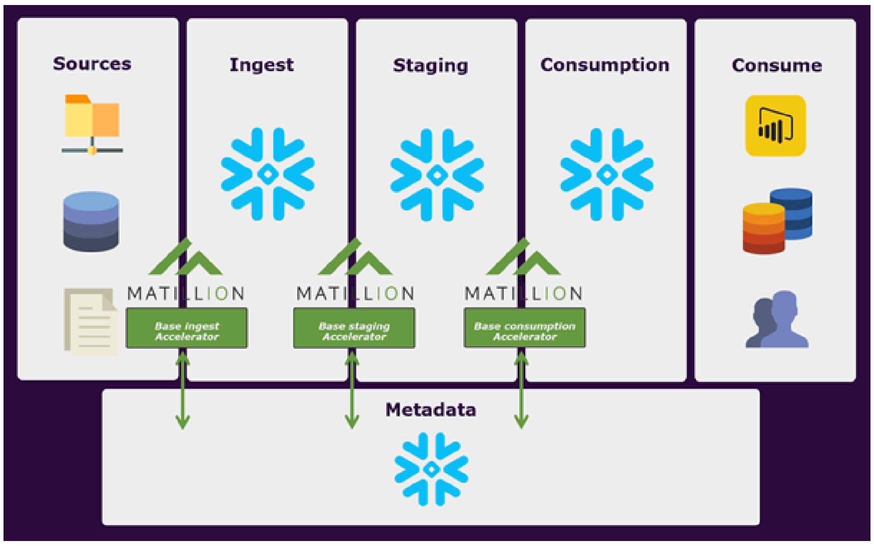
Would it not be great if we could generate or reuse existing work for these projects rather than build everything by hand? This is exactly what an accelerator does. With such a tool it is possible to simply import existing processes rather than code them manually, which safes a lot of time. In this blog I will show you how accelerators can play an incredible role in speeding up your engineering. We will discuss the benefits of an accelerator and demonstrate this with

The value of an accelerator

Before discussing the workings of the accelerators and the technical makeup according to best practices, let us consider the actual value of using an accelerator. As mentioned in the introduction, an accelerator works by reusing- or generating code and standardize way of working. This is useful for a multitude of reasons. First of all, working with the output of a generator helps speed up a project at greenfield clients. A data engineer does not have to spend time working on the elementary parts of a project such as creating a first ingest of sources or setting up a basic ETL flow. Secondly, since this accelerator is community developed, you can be certain that the output adheres to a vendor and community best practices. Any oversights that might have made their way in the code of the accelerator can be easily found and patched by a member of the community. This transparency of the accelerator, whereby anyone can see and improve the code is an incredible advantage over having to build a process yourself. For greenfield projects this means that by using an accelerator one also avoids running into problems that you could have faced when building a process from scratch. This transparency can also aid clients working in a brownfield situation whereby the accelerator can help to easily fix or replace flawed processes that the companies' data team could not fix before. Lastly, a good accelerator brings value in its way of reusability. Regardless of the data maturity of a client, previous done work can easily be reapplied to new pipelines that need to be build.

The accelerator in the data warehouse architecture

A typical data warehouse exists out of a number of layers or data stages. For our accelerators we will be using a separate Snowflake schema for Ingest, Staging and Consumption. We have data coming in from a multitude of different sources, such as a batch load from a SQL server database or a streaming data from Axual. Using our first accelerator in Matillion, we will bring the 'raw' data from these aforementioned sources into our data platform and load them in the Ingest layer. After the ingestion of our data, we will move the data to our staging layer with the second accelerator. From the staging layer we can model, integrate and prepare the data to be ready for consumption in whatever tools our client will be using. Lastly, a metadata schema in Snowflake is used to store the tables that control our Matillion jobs and the tables that store our logs.



Overview used architecture

When building an accelerator, the developed jobs need to be dynamic. If there are hardcoded values in the Matillion job, then this job will not run in other projects without having changed all these values by hand. What we want to do instead is have our Matillion jobs use variables and be controlled by metadata. For this accelerator we will be using two different self-made tables in Snowflake; JOB_MANAGEMENT and TABLE_DETAIL.

The JOB_MANAGEMENT table is the one that defines the different jobs that need to run in our accelerators . The STAGE column refers to the target schema (e.g. ingest, stage). The SOURCE column defines from which source the data is coming from, for ingest this can be sources such as BLOB or SQL_DATABASE. For staging the source is the ingest schema. The columns SOURCE- and TARGET_TABLE define the name of the tables to be loaded. Whereby the column LOAD_TYPE describes whether a table is loaded fully or incrementally. Lastly, the ACTIVE column determines if a job should run or not and the LAST_LOADED column tracks the last time the table has been loaded.

In TABLE_DETAIL the metadata of the loaded tables are stored. Columns here describe both the source- and target database, schema, table name, column name, data type and ordinal position. Naturally, this table can be expanded with other useful data such as maximum character length. Exactly how these tables are used will be shown later on.

Ingest- and Staging accelerators in Matillion

When starting a new project within Matillion, you are welcomed by an empty space for you to design your ETL flows in. There are different ways to get started with a new project and depending on the experience of the data engineer getting things up and running can take a while. It is this situation where using an accelerator is powerful. Just download a community developed- and ready to use pipeline from Github.

Matillion jobs are stored in JSON format, anyone is able to up- and download these project files. Starting out with a project, you can upload the base ingest accelerator from our Github repo. In this blog, csv files are loaded from our Blob storage into Snowflake, so it is required to upload the Azure Blob Storage component into our Matillion project. After inserting the base ingest accelerator and Blob Storage component, you also needs to look at the ‘need-to-set’ environment variables, whereby an example is also provided on the Github repo. By using environment variables in Matillion, it is easy to rename components such as the schema name without having to manually adjust every individual component.

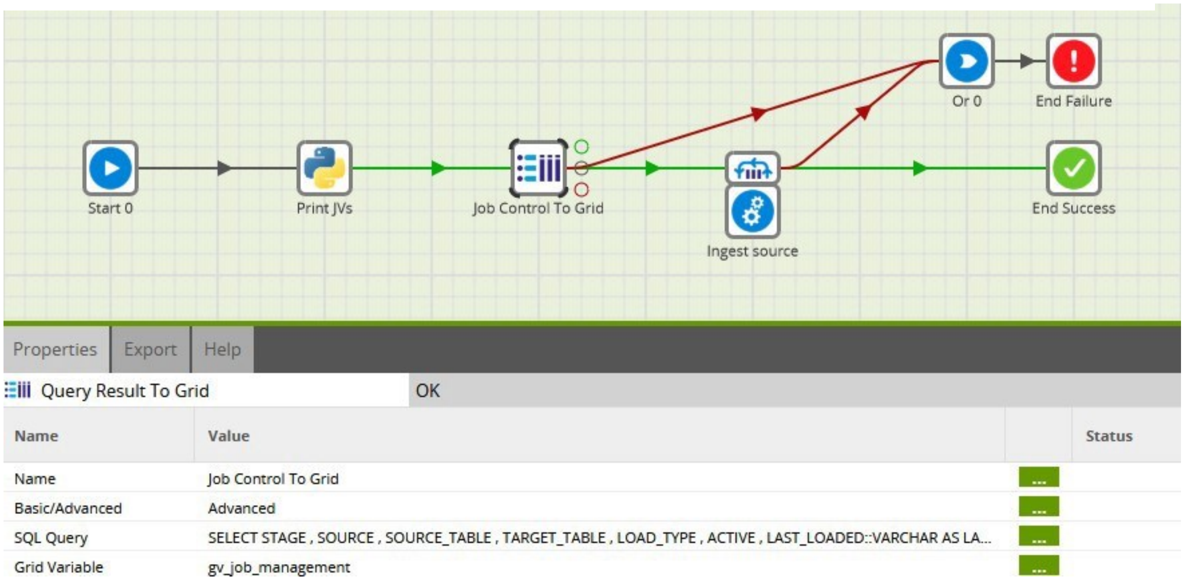
Having setup the details from our blob storage into our metadata tables, it is now already possible to run the ingest job and have the data from the Blob storage inserted into our Snowflake based data platform. Now in the case that we also have data stored in a SQL database, you can easily adjust our Matillion project by uploading the SQL database component from Github in our project and make an adjustment in our job control tables. The same goes for including an API ingestion job, however, you need to configure an API profile in Matillion first before being able to run the component.

When moving on to engineering a data pipeline for your staging- or even modeling layer, the exact same principles apply with the other accelerators as that of setting up the ingest stage. Again, you only need to pull the accelerator and additional components from the Github repository and import the JSON files in the Matillion project. Before running the job, you need to populate the required metadata tables and the jobs are good to run.

Technical aspects of the accelerator

So how do the Matillion accelerators run? As mentioned before, the jobs in the accelerators are built to be dynamic by using different Matillion variables. These variables are determined using the aforementioned metadata tables. When looking at the first orchestration job in the ingest accelerator, you can see how this works. A so-called 'Query result to grid' component is used to retrieve all the relevant data from the JOB_MANAGEMENT table and store these in a Matillion grid variable . The next component is going to iterate over every table, whereby the previous queried columns are stored as job variable.

```
SELECT STAGE
     , SOURCE
     , SOURCE_TABLE
     , TARGET_TABLE
     , LOAD_TYPE
     , ACTIVE
     , LAST_LOADED::VARCHAR AS LAST_LOADED
     , QUERY
FROM "${ev_database}"."${ev_sch00_metadata}"."JOB_MANAGEMENT"
WHERE STAGE = 'INGEST'
  AND IS_ACTIVE = 1
```

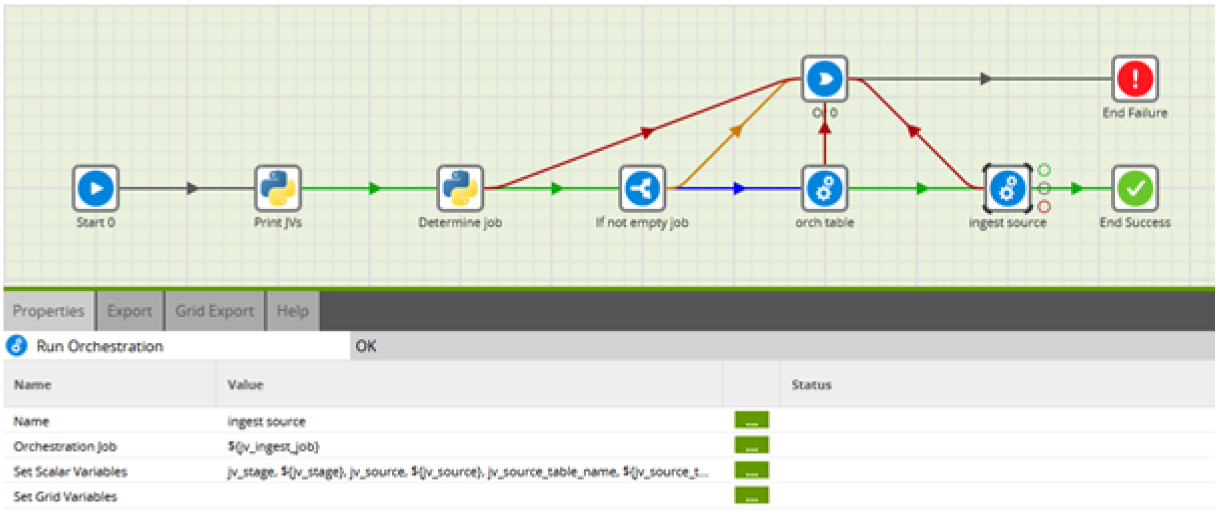


Query Result To Grid OK

Name	Value	Status
Name	Job Control To Grid	
Basic/Advanced	Advanced	
SQL Query	SELECT STAGE , SOURCE , SOURCE_TABLE , TARGET_TABLE , LOAD_TYPE , ACTIVE , LAST_LOADED::VARCHAR AS LA...	
Grid Variable	gv_job_management	

'Query result to grid component in the accelerator'

These job variables are then used to control the job flow. For example, the job variable 'jv_ingest_job' is established by the SOURCE column from the JOB_MANAGEMENT, this job variable is than used in the 'determine job' component to select the loading module. You can read more about grid variables on the Matillion website through this link.



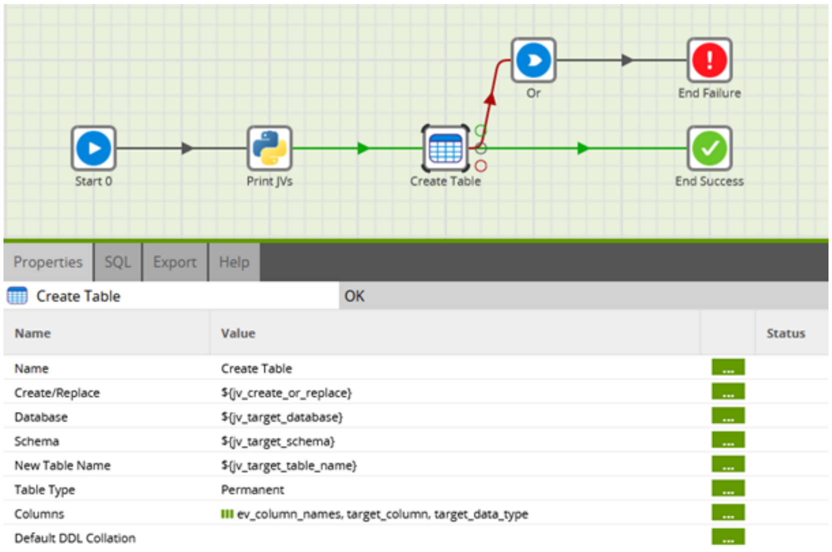
Run Orchestration OK

Name	Value		Status
Name	ingest source	...	
Orchestration Job	${jv_ingest_job}	...	
Set Scalar Variables	jv_stage, ${jv_stage}, jv_source, ${jv_source}, jv_source_table_name, ${jv_source_t...	...	
Set Grid Variables		...	

'Query result to grid component in the accelerator'

A second grid variable is populated later on in the job by querying the TABLE_DETAIL table. By creating a grid variable of all the relevant column data, we can use this to dynamically create, update or select tables in Matillion by simply inserting the grid variable as the required column list.

```sql
SELECT "TARGET_COLUMN", "TARGET_DATA_TYPE"
FROM "${ev_database}"."${ev_sch00_metadata}"."TABLE_DETAIL"
WHERE "TARGET_TABLE" = '${jv_target_table_name}'
  AND "STAGE" = '${jv_stage}'
ORDER BY TARGET_POSITION asc
```



Create Table OK

Name	Value		Status
Name	Create Table	...	
Create/Replace	${jv_create_or_replace}	...	
Database	${jv_target_database}	...	
Schema	${jv_target_schema}	...	
New Table Name	${jv_target_table_name}	...	
Table Type	Permanent	...	
Columns	ev_column_names, target_column, target_data_type	...	
Default DDL Collation		...	

Insert the list of columns as grid variable to create a table

Our accelerators are scalable by design. Matillion projects have the tendency to grow, so you need to build a flexible job in order to have this be effective on a large scale with a big engineering team. There are multiple steps in achieving this goal. The first one is to make use of a consistent variable naming scheme. This naming scheme makes it clear for all team members what kind of variable you are working with. In the accelerators we use the prefix jv_ for job variables, gv_ for grid variables and ev_ for environment variables. Furthermore, at the start of every job, a list of all job- and grid variables are printed in a Python component. This is ideal for troubleshooting since it helps your team to understand the current state of your data when running the tasks.

Logging accelerator in Matillion

When running multiple Matillion processes, chances are that you want to log your processes. This can be for various reasons such as sending an alert on errors, troubleshooting and auditing. We also have a logging accelerator that that supports just this out of the box. This job is simply plug-and-play, you only have to import the jobs and call on it to have it write the output. In our case we are writing our logging outputs to two tables in our metadata schema in Snowflake. The logging data is retrieved by making two API calls on Matillion itself: logging summary and detail. Unlike the API component discussed in our ingestion job, the API profile for Matillion is already included within the program so one does not have to configure this profile separately. After making the two API calls, the retrieved data is combined with each other in a transformation job to make one big table. From here on out we split the data to make two relevant tables; one for displaying all the jobs that have run with an error and another table containing details about all the tasks that have run.

Dealing with platform limitations in Matillion accelerators

You also have to keep the limitations of the platforms in mind when designing these accelerators. In the current form of Matillion, you need to be mindful of the resources of the Virtual Machine running underneath. This means that Python jobs should be quick and lightweight and should not run too many lines in parallel. The general advice when using Matillion is to prefer transformation jobs over orchestration jobs, since transformations are push-down SQL on Snowflake. This means, that there are less performance limitations.

The thing to keep in mind for the Snowflake side, is the preference for creating your own TABLE_DETAIL table rather than using the similar one available in INFORMATION_SCHEMA. The reason here is that although the INFORMATION_SCHEMA can be quite detailed, it also becomes slower to query upon as your data warehouse keeps growing . This slow query time can definitely become a big bottleneck in your Matillion job performance. Furthermore, with a large amount of data movement it can be useful to dynamically increase and decrease the size of the Snowflake warehouse. This does not help with a faster performance, but due to the way in which costs are calculated, using a larger warehouse can even end up being cheaper than a smaller one.

So, looking back; why is using accelerators so valuable? Let’s take a look at the key takeaways:

1. **Transparency**: everyone is able to see the code and improve upon it
2. **Reusability**: variables make it possible to run a job in multiple environments
3. **Metadata driven**: for controlling your job and logging your data
4. **Scalability:** designed with the limitations of your resources in mind

Firstly, you can rely on the expertise of your fellow coworkers and make use of a community-developed accelerator. You do not have to reinvent the wheel again, the accelerators are meant to be a great starting point in getting the first data pipelines and thus valuable business use cases running as quickly as possible. When expanding the accelerators or building a new job, you need to take in mind to create dynamic jobs. Not hardcoding values, makes it easier to reuse your developed jobs. Furthermore, using metadata tables makes it easier to have one central place to control your pipelines. Any adjustments are easily made in these tables. Metadata can also be used for logging your flows and gaining insight on these processes. Lastly, you need to keep scalability in mind. So be cautious when using resource intensive components in larger projects such as Python in Matillion or the INFORMATION_SCHEMA in Snowflake. Keeping these key takeaways in mind, you should be able to have a flying start with your next ETL project using an accelerator.

Happy engineering!

More from the Sogeti Data Experts | Netherlands

The Sogeti Data Community loves to share its experiences with each other and with you. Please feel free to connect. You can join our SoConnect Meetup Group where we discuss a wide range of topics, problems and real-life data solutions. Of course all related to Data services, Data platforming, Data management and Data science topics.

If you are looking for inspiration and learning on a day-to-day basis, check out WerkenBijSogeti. We are always looking for great minds to join us.