ETL with Azure Serverless

Padmaja U.K., Chief Architect, Persistent Systems Limited

Problem Statement

Traditionally, customers who are especially on the Microsoft stack, have been using Microsoft SQL Server Integration Services (SSIS) for their data integration and data transformation solutions. With more and more customers embracing their journey to Microsoft Azure, it makes more sense to adopt a cloud native solution for implementing the Extract-Transform-Load (ETL) process. Though SSIS is indeed a very powerful ETL tool with many out of the box connectors packaged with it, customers migrating their workloads to Microsoft Azure either migrate their SSIS packages to the equivalent Azure Data Factory services or host them on a SQL Server installed on an Azure Virtual Machine. Latter one is a time saver, however, remains as a single point of failure in the application's deployment. Also, supporting additional source systems like Salesforce CRM require the use of 3rd party connectors like SSIS Powerpack from *ZappySys*.

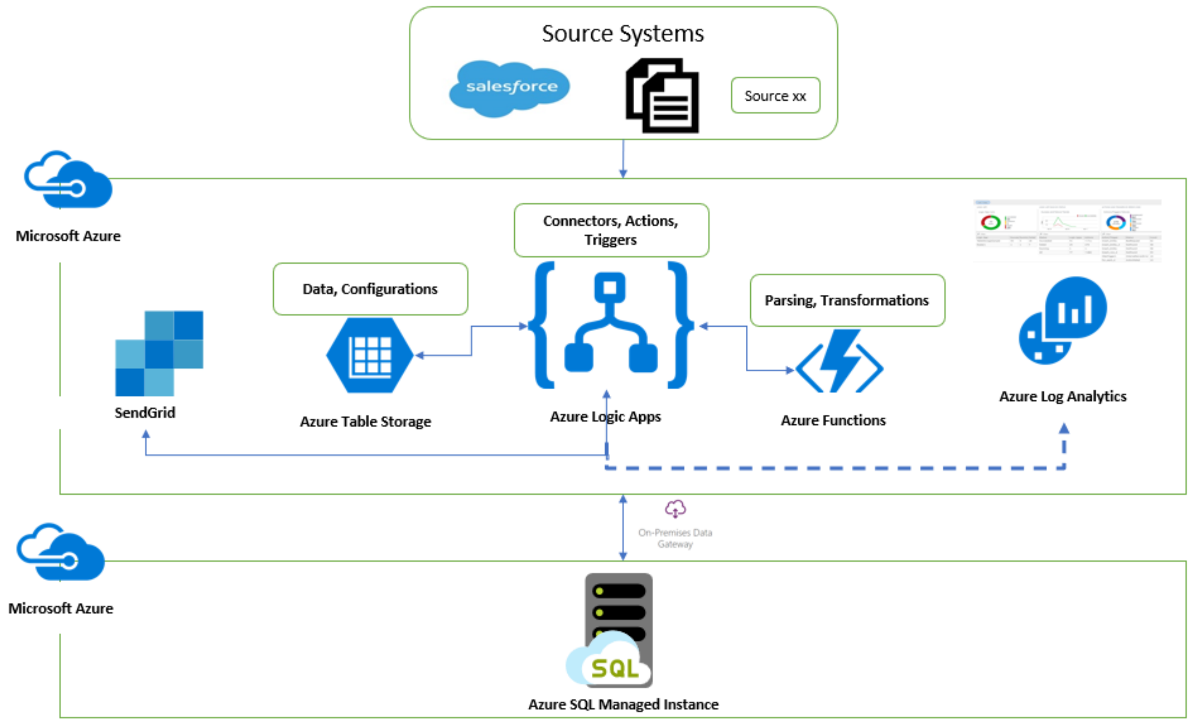
In this blog, we look at how Azure Logic Apps with its ready-made connectors can be leveraged to build an ETL workflow which is easy to deploy, maintain, and monitor. ETL with Logic Apps is not a typical use case for Logic Apps as it is more commonly used as an iPaaS solution.

Solution/Architecture

Solution included the following features:

Out of the box connectors and actions	
Connector	**Action(s)**
Salesforce	Get Record(s)
SFTP	List files in folder, Get file content using path
Azure Table Storage	Get Entity, Get Entities, Insert Entity, Insert or Merge Entity
Azure Blob Storage	Create blob
SQL Server	Execute Stored Procedure
SendGrid	Send Email
Azure Functions	N/A
Workflow capabilities	
Capability	**Usage**
Conditions such as if/else, foreach, until	To control the workflow
Scope	To logically group related actions and to implement exception and error handling at a logical level
Filter Array	To get the context about the actions that failed in a Scope

Variables	To assign and manipulate the values during the workflow
Run after setting	To control the workflow action sequence
Expressions and functions	To manipulate data, e.g., concat(), AND, OR



Technical Details and Implementation of solution

As in any typical ETL workflow, the source data is extracted, cleaned & transformed, and finally loaded to the destination system in this implementation. Each *stage* of the ETL process is defined within a **Scope** in the Logic App. This not only helps in repeating a failed stage but also allows better control around error handling.

Logic app is configured to run at scheduled intervals using *Recurrence* trigger. Every run generates a unique run ID E.g., 08585304237785568400144090857CU73. This ID is captured within the Logic App workflow using the expression: `workflow()['run']['name']` and is recorded in the Table storage for tracking a run.

Configuration data such as information about the source systems, data extraction configurations, emails, source to destination mapping details, and so on is maintained in Azure Table storage and is retrieved using Table storage connector. Raw data extracted from files or Salesforce are also stored in Tables.

Raw data is processed further using Azure functions by reading the Azure Tables and the pre-stage data is prepared, which is then stored in separate Tables. This data is then moved to staging database residing in Azure SQL Managed Instance (MI) which then undergoes business validations and finally gets loaded into the application's database within MI.

Though SQL Managed Instance is an Azure service, Logic App treats it like an on-premise SQL Server, and the connectivity from Logic App is established using an on-premise data gateway. The two systems viz. Logic App and MI, are in a way disconnected; hence a polling strategy is adopted to get the data loading status from the MI.

Any errors encountered during any of the stages would trigger a failure email to the recipients from within the Logic App. Successful processing requires additional information such as number of records received and processed, date and time of completion, business validation errors if any and so on. This information is solely with the MI, hence notification is sent out from MI using database mail capabilities. Unique Run Id of the Logic App run is passed onto the the backend system as well which helps in traceability of the whole run end to end.

Logic Apps management solution is used as the one stop shop for monitoring all the Logic Apps in the Azure subscription

Challenges in implementing the solution

Pagination

Certain connector actions support data retrieval in bulk with a default page size. E.g., Salesforce Connector -> Get records action gives around 2000 records at a time. Such connectors have a Pagination setting which can be used to get the next set of pages using a continuation token handled by the connector itself. However, this wasn't easily understood, as the threshold setting and the records retrieved may vary as described here.

Turning on pagination with correct threshold value to get the right amount of data was challenging.

Batch processing

Getting all the records in a single iteration was difficult to handle from processing point of view as some of the actions including Azure functions timed out due to large amount of data. With batch processing, XML datasets were sent to backend system with a flag to indicate whether all the batches are completed or not.

Parallelism

By default, "For each" iterations run in parallel. This can be controlled by changing the settings where the degree of parallelism can be set to 0 (no concurrency) to a max of 50. During batch processing, it was noticed that concurrency was causing issues with the order in which batches were picked up for processing, and the control had to be changed to “no concurrency” mode.

SQL Connector timeouts

The SQL connector has a stored procedure timeout limit that's less than 2-minutes. This was difficult to identify during the development phase as we dealt with small amounts of data. Design changes were made to make use of SQL Agent job for backend processing which updated the job progress in a SQL table that was polled by the Logic App at regular intervals.

Nesting levels

Actions can be nested as part of the workflow definition, however, the depth is limited to 8 levels.

One of the use cases required creation of a hierarchical data structure from Salesforce object which maintained only parent child relationship in a flat structure. A design change was needed in the workflow to fit within the limit and to meet the use case as the initial design hit the 8-level limit.

Performance

Out of the box connectors, especially Azure Table storage connector and Salesforce connector gave poor performance most of the time, and the queries had to be fine tuned to limit the record set to address the performance problems.

Deployment automation

Integrating Logic App deployment into Azure DevOps pipeline was challenging as the same workflow definition was used across consumers with consumer specific configurations. Most of the values had to be parametrized in the ARM template including the recurrence schedule.

Business Benefit

Configuration driven, multi-tenant, turn-key ETL solution using Logic Apps provided a cost effective, easy to maintain and scalable solution. The design was then re-purposed for various other use cases around data exchange and processing.