Building Reliable Dataload Processes through Reltio REST APIs

Cloud-based distributed systems require different techniques from traditional programming models to meet the higher expectations of users on commodity cloud-based hardware. Building ***resilient systems*** requires designing applications to cope with and recover from inevitable system failures without any data loss.

In order to have the best performance and reliability during dataloads, it's critical to make sure that the customer's ETL follows the guidelines described below.

Optimized Data Model

In order to achieve the best performance of loading data into Reltio it is critical to follow Reltio's best practices for data modeling - e.g. make sure immutable reference attributes, good-performing match rules.

The Durability of Data Sources

In order to make sure that there is no loss of data before loading it into Reltio, it is critical that all objects that need to be loaded into Reltio are stored in durable storage: disk, AWS S3, GCP GCS, Azure Blob Storage, stream-processing platforms. An object can be marked as successfully loaded into Reltio (e.g. acknowledge an event in a stream-processing platform) only after getting a response from Reltio with 200 HTTP status code.

Loading Data into Reltio Through REST APIs

Reltio offers various REST APIs to load data to Reltio. All of the REST APIs are synchronous and may include multiple objects within the payload of a POST request. REST API requests can be executed in parallel with the recommended number of parallel simultaneous requests to have the optimal performance when loading data to Reltio.

Synchronous Requests

Reltio REST APIs are *synchronous*. *Synchronous calls* must keep the network connection open until a response is returned from the service, unlike asynchronous calls which close the connection as soon as the request is submitted.

Connections Pool

Opening and closing connections is an expensive process so keep a pool of open connections that get consumed and then returned to the pool rather than opening and closing for each record.

Requests Sizes and Number of Simultaneous Requests

Each tenant and dataset loaded will exhibit different behavior with respect to performance. Below is a chart with suggested numbers of objects per post and thread counts based on the size of the message body and attribute count.

Size	**Object Size**	**Approx. Attributes Per Object Size**	**Records Per POST Request**	**Threads**
Small	0Kb - 15Kb	0 - 300	50 - 100	15 - 20
Medium	15Kb - 70Kb	300+	30-60	10-15
Large	70Kb+	300+	10-30	5-10

NOTE:

- If the limit of the number of simultaneous requests for a tenant is reached, Reltio may return a response with a 503 or a 429 HTTP error code, which indicates the client must slow down requests using an exponential backoff algorithm
- It is strongly recommended to avoid updating the same objects from different parallel threads.

Retries

A request should be retried only if an error code (non 200 HTTP status code) is received in the response. A retry should be performed in the following manner.

- Using the same pool of requests with the recommended number of parallel requests.
    - It is strongly recommended that you do not create new connections to Reltio for failed records as it will increase the number of simultaneous requests.
- Use exponential backoff: https://en.wikipedia.org/wiki/Exponential_backoff.
- A retry should be done only for failed records and not for all of the records. For example, for daily dataloads, it is not recommended to reload all data for

that day if only 10 records had errors. Only the 10 records resulting in errors should be retried.

Recording Failed Records

If a record failed to be loaded into Reltio even after retries, it is critical to save those records in a file or a dead letter queue to make sure those records can be investigated and reloaded once any issues are addressed.

It is always a good idea to record the reason for failure with the original response details that were obtained from the platform.

Monitoring and Logging

It is critical to make sure that the integration code has the proper level of logging and monitoring. It is useful to have real-time status through logging system or monitoring custom metrics about the progress of the current dataload:

- how many records have been loaded
- how many records have been failed
- current operations per second

Sample Code for Loading Data to Reltio

The following is a sample code for handling HTTP response codes where it is recommended that the client application retry the request.
Java:
https://bitbucket.org/reltio-ondemand/util-dataload-processor/src/master/