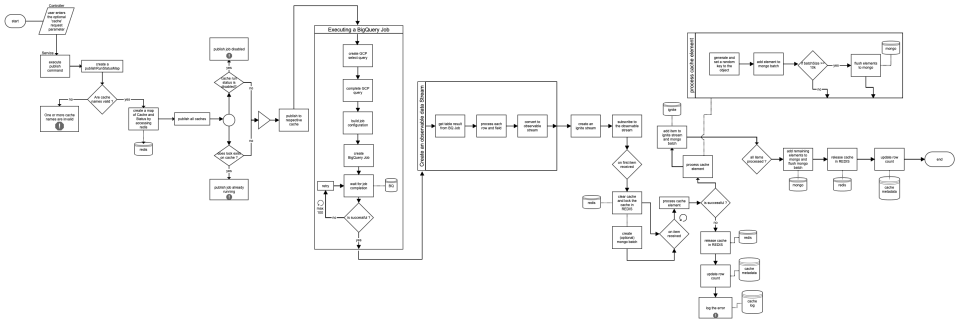
Publish Job Flow



This API is for copying data from a BigQuery table into its respective ignite cache.
It clears the entire cache and starts inserting records from the beginning.

Controller

SchedulerCommandController.java

The user sends a request parameter called ‘cache’ along with their request, this parameter can have 3 types of values

1. Empty
2. List of comma separated cache names
3. A single cache name

Service

PublishCommandImpl.java

Mono<Boolean> execute()

- The execute command first creates a publishRunStatusMap (Map<String, Boolean> -> Map<CacheName, PublishJobStatus>) by calling the getPublishRunStatusMap(String cacheName) which takes in the cacheName (list, empty or single) as a parameter
- It then calls publishAllCaches(Map statusMap) that takes the above created statusMap, it calls it via CompletableFuture.runAsync and uses a ThreadPoolExecutor

Map<String, Boolean> getPublishRunStatusMap(String cacheName)

- This method first gets all the available caches in ignite via the ApacheIgniteConstant.getCacheList()
- Here REDIS is used to store the status of publish job for each cache
    - TRUE -> the publish job is active for that specific cache
    - FALSE -> not active
- cacheName :
    - If the cacheName parameter is empty
    - Then, it iterates through all the available cache list and creates a map entry i.e String, Boolean, the boolean value is obtained from REDIS
    - If cacheName is a single cache or a list of caches
    - Then, it iterates through the entire list, sets all the caches to true (meaning that a publish job is ongoing, hence locking it), it only updates the caches inside the cacheName parameter by obtaining the status from REDIS.
    - It also checks if the caches inside the cacheName parameter is valid or not by calling another method.

void publishAllCaches(Map statusMap)

- Each cache has its own cacheProcessor class that extends IgniteCacheProcessor.java
- All these cacheProcessors have a build_$cacheName$_Cache(Map statusMap) method
- The above method takes in the generated statusMap, it checks for the status of its cache in it,
    - If it is true, it logs a message saying that a publish job for that cache already exists
    - If it is false, It calls the publishToCache() method of that cacheProcessor.

CacheProcessors

$CacheName$CacheProcessor.java

void publishToCache()

- Every CacheProcessor has this method.
- This method is for Querying the data from BigQuery into the respective ignite cache
- To query all the data from BigQuery, buildGCPQuery() -> buildGCPSelectQuery() of the CacheProcessor class and BigQueryProcessor class are called to create a query that returns all the records of that table.
- The buildGCPSelectQuery(Class TargetClass, String table) builds a select query String with all the fields of the table, the fields are obtained from the TargetClass parameter
- A BigQueryInventoryProcessor object is used to process the query.
- BigQueryInventoryProcessor extends the BigQueryProcessor which has a method called executeQuery()

BigQueryInventoryProcessor.java

CompletableFuture<Observable<T>> execute(String query, final Class<T> targetClass, final List<String> fields)

- This method calls two methods from its parent class , executeQuery() and toObservable()
- The result of executeQuery() is passed on to toObservable()
- It returns a Future object.

BigQueryProcessor.java

Job executeQuery(BigQuery queryClient, String query)

- Here the JobInfo configuration is done
- The configuration takes in the query to be performed along with other properties and a new QueryJobConfiguration object is created.
- It then calls the processCloudPlatform() which returns a Job object.

Job processCloudPlatformQuery(BigQuery queryClient, String query, QueryJobConfiguration jobConfiguration)

- Here, the actual processing of the query is done.
- BigQuery processes queries by creating a job, it accepts a JobInfo
- The JobInfo object is built using the QueryJobConfiguration object and a random Id is set to it.
- The result is a Job object and there can be multiple retries as well.
- The resulting Job object has the TableResult and other data.

<T> Observable<T> toObservable(Job response, Class<T> clazz, final List<String> fields)

- This method helps to supply data in the form of individual records to be inserted into the cache.
- It takes the response from the BigQuery client, the target class and a list of fields(columns for the table).
- As the TargetClass represents one record in the table, the method iterates through the FieldValueList, transforms the object by setting the values to their respective fields and returns it back.
- toTransform() helps to transform all the rows of the table……..
- ——————————————————————xxxxxxxxxxx————————————————————————————————

void publishToCache(Observable<T> targetCache)

- This method is responsible for publishing each row in the form of a cache object into its corresponding cache.
- A cache streamer is used to achieve this.
- When it receives the first object, it locks the cache by setting the publish job status key of that cache to true in REDIS.
- It also clears the entire cache i.e. ignite and mongo caches.
- Not all the details are stored in both ignite and mongo cache
- For example,
    - For the vendor table, the VendorDetails objects are stored in mongo cache whereas the Vendor objects are stored in ignite.
    - After processing all the objects, doOnComplete() is called, in case of a mongo cache It sends the remaining elements in the batch to the cache and clears the batch
    - It then unlocks the cache and make it open (lockRelease) to handle publish jobs again, This is done by setting the status key to false in REDIS.
    - The final step is to updates the row count in the cache metadata.