Contents

[Recital 3](#_Toc509400641)

[Existing setup 3](#_Toc509400642)

[Bird’s eyeview 3](#_Toc509400643)

[Detailed view 3](#_Toc509400644)

[Technology stack 5](#_Toc509400645)

[Limitations 5](#_Toc509400646)

[Purpose of new CarrierTracker project 6](#_Toc509400647)

[Scope 6](#_Toc509400648)

[Technology stack 6](#_Toc509400649)

[Proposed design 7](#_Toc509400650)

[Program Flow 7](#_Toc509400651)

[Artifacts of new design 7](#_Toc509400652)

[Prototypes / tasks 7](#_Toc509400653)

[Challenges 8](#_Toc509400654)

[QA scope and test scripts 8](#_Toc509400655)

[Research 8](#_Toc509400656)

[Benefits of proposed design 9](#_Toc509400657)

Revision history

|  |  |  |  |
| --- | --- | --- | --- |
| Index | Date | Person | Remarks |
| 1 | 14th Mar 2018 | Sushant | Created the document |
| 2 | 15th mar 2018 | Sushant | Modified |
| 3 | 16th Mar 2018 | Sushant | Explored Azure serverless functions |
| 4 | 19th Mar 2018 | Sushant | Explored present design |
| 5 | 20th Mar 2018 | Sushant | Documented present design in the document |
| 6 | 21st mar 2018 | Sushant | Incorporated proposed design in the document |
|  |  |  |  |

# Recital

* Wineshipping, a logistics entity for the wine industry, provides direct-to-consumer fulfillment services. The company ships over 50-60,000 shipments daily during the peak with over 2 million shipments in a year
* The packages are shipped through common carriers (UPS, Fedex, GSO and potentially others)
* Shipping statuses of different packages are tracked and published / updated to a portal through an inhouse carrier tracker software. The respective customers can see status of delivery on the portal.
* A requirement is felt to upgrade / rearchitect the backend existing carrier tracker application to cater to increased workload using efficient methodologies. The application polls the carrier databases for status updates and then updates the new status in the tracking DB. This process of update needs to become efficient and leverage API or other newer technologies.
* The new project is named as **CarrierTracker**
* Business / Functional Requirements - ?

# Existing setup

## Bird’s eyeview

* The client uses Microsoft Dynamics AX ERP solutions as tool for running its day-to-day business. Primary business of client is to ship several thousand wine packages to various locations across the country through reputed carriers. The entire ship-to information along with carrier details is captured in Dynamics AX database.
* By using custom code and a bunch of stored procedures data is imported from Dynamics AX and stored in a database named as **Tracking** of Sql Server. By using custom code the tables in Tracking database are further processed for proper formatting.
* Tracking database contains information of various carriers. The api’s of these carriers are queried by passing them the wine package information and status of those wine packages are obtained. The final status data of wine packages at various stages of shipment is stored / updated back in Tracking database.
* Customer gets the latest status information of wine packages being shipped by visiting a portal which makes use of Tracking database.

## Detailed view

* **Tracking database**
  + **Stored procedures**
    - ***ImportPackages***

Parameters are passed to this SP. These parameters are inserted in **Wineshipping$Package Info** table.

* + - ***GetPackages***

Gets data from **Wineshipping$Package Line** table.

* + - ***GetPackagesToTrack***

Gets data from **Wineshipping$Package Info** table

* + - **InsertLog**

Accepts parameters and inserts them to Logs table

* + - **UpdateTrackingStatus**

Using the input parameters updates the table Wineshipping$package Info table with latest status information of wine package

* + Tables
    - **Wineshipping$Package Info**

Contains Wine package header information imported from Dynamics AX

* + - **Winshipping$Package Line**

Contains Wine packages line items information imported from Dynamics AX

* + - **PackagesToTrack**

Contains Wine packages header information along with Tread no’s created by TrackPackageThread project. This table is emptied at the end of processing. In the beginning the number of records in PackagesToTrack is same as that of **WineShipping$Package Info** table.

* Custom code
  + **ImportPackage**: Custom project in C#
    - Imports data from Dynamics AX using ODATA. Makes use of custom utility ODataUtility. Data is available in a large JSON object.
    - The JSON object is parsed and parameters are passed to ImportPackages stored procedure which in turn updates / inserts data in **Wineshipping$Package Info** table.
    - Data is also updated / inserted in table **Winshipping$Package Line** table by using inline sql command.
    - ImportPackage is run on a scheduled basis.
  + **TrackPackageThread**: Custom project in VB.net

In below the word thread is a misnomer. Thread below is merely a number in database table and is not related to actual **thread** in a .net environment.

The purpose of this project is to create thread numbers for each 2500 records and store data in PackagesToTrack table and then call azure web job for each thread no.

Pseudo code:

* + - Get records from Wineshipping$Package Info table into a dataset ds.
    - Iterate through all records in ds
      * Create a new thread for every 2500 records and insert a new record along with the new incremented thread no into table PackagesToTrack
      * At the end of above iteration, the no of records in Wineshipping$Package Info table = no of records in PackagesToTrack table.

Say the variable n stores total number of threads.

* Iterate n times

Call an Azure web job and pass it the thread no which is iteration count. Thus, if n threads are there, the web job is called n times each time passing it the running number.

N.B: There is some dangling code in this project which is neither commented nor used.

* + **PackageTracker**: Custom project in VB.net
    - Receives thread no as argument
    - Read all records from **PackagesToTrack** table into a dataset ds against the thread no obtained as argument. PackagesToTrack table was populated by previous custom project TrackPackageThread. Each row of the table maps to a single wine package which is to be tracked.
    - Iterate through ds
      * Connect to each carrier through calling its api from http request and get the response which contains the tracking status of package.
      * Using the above response update the PackageTrackStaging table with latest tracking status.
      * Repeat this till all rows in the ds are exhausted
    - Using the subroutine FinalDBUpdate update Wineshipping$Package info table with latest tracking status information from PackageTrackStaging table.
    - Truncates two tables PackagesToTrack and PackageTrackStaging.
    - Send mail to concerned person with information of status update

## Technology stack

* Azure web jobs / schedulers: To schedule the execution of carrier tracker application
* C#.net: To import information from Dynamics AX in asynchronous call
* VB.net: To update the carrier shipping status at customer portal database
* Azure Sql server: To store intermediate and final status data of carrier tracking

## Limitations

* It appears that existing codebase has evolved in 5-6 years, thus contains the old baggage. Certain code snippets appeared to have been used earlier which was later discarded but the codebase was never cleaned up and hence there is a lot of dangling code which is never reached.
* Database connections are unnecessarily opened and closed several times which may lead to extra processing time and consumption of resources.
* Database stored procedures are made unnecessarily complex. There is no proper rule as to when and why stored procedures are to be used in comparison to inline sql code.
* Multiple web job calls are made in parallel using heavy consumption of resources. Each web job results in a new thread.
* Around 2500 API calls are made in sequence in synchronous manner by each web job. Minimum time taken to run entire package is time to execute 2500 http api calls in sequence. If single api call takes 3 seconds to execute then entire time taken might be 3\* 2500 = 125 min’s = 2 hours approx. It appears that entire package might take 2Hrs to 2Hrs 30 mins to execute. This time might be reduced to less than 10 minutes.
* The codebase is non-maintainable. Different programming styles are incorporated as and when thought of.
* The present design relies upon web job. The author of codebase has probably found a magic number 2500 through heat and trial way. This magic number is the number of calls to carrier API in synchronous manner. If a carrier api call is delayed by one second, then there may be an additional delay of 2500 seconds to execute entire code.
* If no of rows in file Wineshipping$Package Info becomes 1,000,000 then there will be 1,000,000/2500 = 400 threads running in parallel. This will significantly add to resources since each thread in .net consumes 1MB RAM. Hence there may be chances that 400 MB of ram is consumed by the threads only. Furthermore, a high number of uncontrolled threads can crash the code.

The code is susceptible to crash in certain situations.

# Purpose of new CarrierTracker project

* Implement more efficient way to track carrier information.
* Cope up with increased workload with a view of further increment of workload in future.
* Easily maintain the code otherwise and when carrier upgrades its api or introduces new api.
* Provide easy steps for adding new carriers.
* Come out of limitations of existing tracking application using modern techniques.
* Reduce complexity of existing setup.
* Reduce consumption of resources, package execution time and instances of crash.

# Scope

* CarrierTracker project seeks to provide functionality provided by existing **PackageTracker** and **TrackPackageThread** projects**.** Import of shipping and carrier information data from Dynamics AX to database currently provided by projects **ImportPackage** and **ODataUtility** is beyond the scope of the new project.
* CarrierTracker assumes that updated shipping and carrier information data is already made available to sql server database after successful import through existing ImportPackage custom code.
* CarrierTracker updates carrier status information for wine packages in the database.
* The sql server database is used by a different web portal application to display the latest carrier status information. The functionality of web portal application is beyond the scope of this project.

# Technology stack

* Azure functions running JavaScript as underlying programming language.
* Azure sql server

# Proposed design

## Program Flow

* ImportPackage custom code imports latest data from Dynamics AX and updates two database tables Winshipping$Package Info and Wineshipping$Package Line as before.
* An Azure **Serverless Function App** is created with name CarrierTracker. This function will behave like node.js endpoint. All the sought for functionalities are encapsulated in this single serverless function.
* CarrierTracker is scheduled to run every six hours or as per requirement. The schedule time can be altered by user by a simple configuration of crone job.
* Functionality of CarrierTracker
  + Connect the sql server and get Wineshipping$Package Info table in JSON array. The JSON array such received is an array of objects. Each object in the JSON array corresponds to each row of table Wineshipping$Package Info.
  + Iterate through each object of JSON array.
    - While iterating through each object fire an http get request to respective carrier passing in the package information in asynchronous manner.
    - In asynchronous response of each request update the Tracking database table Wineshipping$Package Info table.
    - Fire the respective contextual email to concerned person.

## Artifacts of new design

* All configuration data like database connection info and API keys etc. are stored in a single configuration file. Whenever any credential changes it is sufficient to make changes in configuration file only. There is no need of any recompilation.
* The CarrierTracker fires all requests in parallel in one go and waits for asynchronous response. There is no sequence of http calls to the api’s in synchronous manner. As such expected time of completion entirely depends on how fast the carriers can withstand a volley of requests from the same source. In case any carrier is slow then some delay might be artificially introduced.
* Main advantage of new design is its asynchronous nature. This makes the code simple and maintainable at the same time faster.
* The new design does not use any intermediate tables such as **PackagesToTrack** and **PackageTrackStaging.**
* There is no need to manage multiple threads. Entire work is done in a single thread.
* New design is simple but tricky.

## Prototypes / tasks

* Create a JavaScript container project with minimum functionalities as Serverless function app and deploy it in Azure.
* Using node.js Sql server driver connect and get the JSON array corresponding to table WineShipping$Package Info.
* Separately thoroughly understand and test api for each carrier from node.js.
* Test carrier api’s through a volley of http calls and check the asynchronous response. If there is any sort of failure, then identify remedy.
* With node.js minimum application as Azure function test the complete cycle of entire functionality with around 10 records only.
* Create a test data of say 100,000 records to process and test with asynchronous http calls to carrier api.
* Check if any api supports for bulk query.

## Challenges

* Scarcity of node.js programmers.
* The app does not use any intermediator database tables. Entire process is completed in computer memory itself. If number of records to process is heavy, then computer memory will be a challenge. If one record takes 200 characters, say 200 bytes then 100,000 records take around 20 MB spaces and 1,000,000 records take around 200MB. The actual scenario needs to be evaluated.
* Asynchronous calls need to be carefully coded with high level of expertise.
* Understanding and implementation of api’s for different carriers which are at present 4-5 in numbers will be challenging. This is where most of time will go experimenting and researching. There are some providers like “**TrackingMore**” at <https://www.trackingmore.com/api-nodejs.html> who claim to provide Tracking API for all major vendors. This is yet to be decided whether to implement individual carrier api services or use a vendor specific provider.
* Some carriers might not have yet implemented REST enabled API’s. It will be challenging to connect SOAP based web services from node.js code.
* There might be limitations for making queries in terms of frequencies from various carriers. These limitations must be carefully investigated before implementation.
* After implementation of the project there should be a testing phase for 8 weeks and the results needs to be carefully studied and discrepancies need to be rectified. That means a few hours say 8 hours a week for 8 weeks of a developer need to be deployed after implementation to production.

## QA scope and test scripts

* Create a test data table of 100,000 records. These records may have several repetitions, but data need to be real for package and carrier information.
* Create test script to iterate 100,000 records and make http API calls to carriers asynchronously and capture the response. Check the behavior.
* Track and log errors. Errors need to be minimized.

## Research

* A thorough research in carrier api’s is to be done. It is to be found for which carrier api’s the customer has subscribed and what are the limitations of query to such api’s.
* A research in the memory model of node.js is to be performed by creating prototypes as explained above. Since whole processing is done in memory it needs to be ascertained how many Wine Packages / records can be handled.

## Benefits of proposed design

* It recommends in memory processing in contrast with persistence of intermediate data in database tables as being done presently. This may improve throughput and reduce consumption of resources minimizing the recurring cost.
* It recommends asynchronous operations in comparison to synchronous operation which increases speed and efficiency consuming less resources.
* It envisages keeping configurable data like database connection, api keys, email details and various other parameters in configuration files. Presently some of this information is maintained in code.
* It proposes keeping all SQl code in separate SQl file. This will ease up maintainability and support.
* It does not require recompilation of code and hence any changes made to the code will be immediately effective after restart.
* Some carriers provide bulk query features. This can be used to increase the speed.

New Aspects

* CarrierTracker will persist historical data in a TrackHistory table. This table will be used to draw reports which will be part of another project X.
* Project X will provide To-the-point analytical reports / pivot reports / Dashboard reports. Carrier tracker will provide all required data in a database table while processing.
* Caution should be taken for the time zone of server and destination.