Contents

[Scope of WSCarrierTracker 2](#_Toc518578132)

[Logical Flow Diagram 2](#_Toc518578133)

[What does WSCarrierTracker do 3](#_Toc518578134)

[Features 3](#_Toc518578135)

[Single thread asynchronous 3](#_Toc518578136)

[Auto adjust piston 3](#_Toc518578137)

[Database channels 3](#_Toc518578138)

[Settings 3](#_Toc518578139)

[Deployment steps 4](#_Toc518578140)

[Performance and tuning 4](#_Toc518578141)

[Pending work 4](#_Toc518578142)

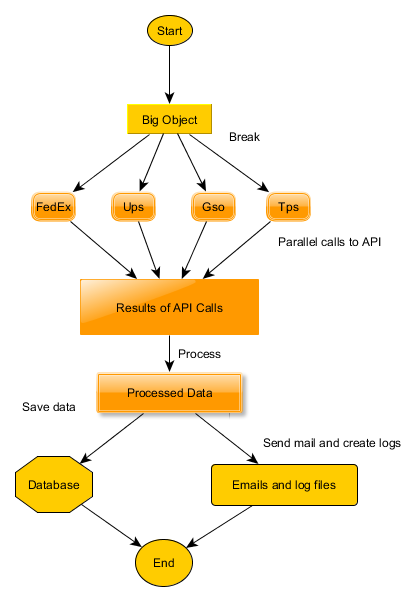
[Known issues 4](#_Toc518578143)

[Database scripts 5](#_Toc518578144)

# Scope of WSCarrierTracker

1. Updates the info table the latest status values for different carriers.
2. Inserts complete thread of transactions related to a Tracking Number in the PackageHistory table in form of JSON data. This table will be used later for reporting purposes.
3. Inserts a row in PackageLog table for each job run.
4. Sends status mails to configured Admins after completion of Job.
5. Creates file log for each job run.

# Logical Flow Diagram



# What does WSCarrierTracker do

* WSCarrierTracker reads records from Info table. Each records pertain to one tracking number against a carrier such as FedEx, UPS, GSO or TPS. All these records are converted to a big JSON object called BigObject.
* The BigObject is decomposed to four smaller objects corresponding to four carrier operators. Each smaller object such created is called carrierObject.
* The carrierObject is processed to contain information such as carrier url, the parameters and API call method. In this manner each carrierObject is self contained.
* Now API calls are made in parallel for each carrierObject and responses are collected and embedded in respective carrierObject.
* Now each carrierObject is further processed with corresponding response obtained from API. Now carrierObject is ready to be updated to database Info table.
* Database Info table is updated for latest status information obtained from API calls by using carrierObject.
* Logs are generated, and status mails are fired to Admins.

# Features

## Single thread asynchronous

Entire application runs on single thread. The api calls are non-blocking and asynchronous.

## Auto adjust piston

There needs to be a delay in API calls for the carriers. We call this delay as piston. When you increase the piston i.e increase the delay then pending responses from API reduces and vice versa. The pending responses from API is called apiQueue. So when you increase the piston the apiQueue decreases. For a better throughput there is an optimum level of apiQueue. We found that if apiQueue is kept at 50 then the performance is better. If we increase apiQueue then the carrier may complain.

The program does auto adjustment of piston values based on the queue value in settings.json. You can see in the logs how piston values are incremented or decremented by 5 in order to maintain a constant queue.

## Database channels

30 simultaneous connections are made to database through prepared statements.

# Settings

* Following configurations are stored in settings.json file. User is free to change them as required.
  + Email settings for Mandrill which is used to send mail
  + Configuration settings for each carrier
  + Database connection information

# Deployment steps

1. **WebApp**

Create a node.js based web app in Azure.

1. **Upgrade node.js**

By default the version of node.js in the web app will be 0.10. You need to upgrade it to the latest version.

1. **Web job**

Create a web job in context of above web app.

1. **Upload code**

Upload WsCarrierTracker code in web job. You can upload the zip file of application or deploy it from GitHub.

1. **Install**

You need to run npm install to install the modules being used by the application.

1. **Configuration**

Open the settings.json file of the application and modify the credentials for email, database and carrier accounts as and where necessary.

1. **Database**

After doing proper configurations of database in the settings.js file the application will point to the desired database. The application uses two extra tables which are PackageHistory and PackageLog. You need to create those tables in your database. The table creation script for those tables are provided separately.

Your application is now deployed. You can run or schedule the web job for desired results.

# Performance and tuning

1. Entire lifecycle for processing 10,000 records take around 5 minutes.
2. Can process 80,000 records. Beyond that memory consumption is very high and program stops. There needs to be better memory management algorithms implemented.
3. Bottleneck is database updates. By using multi core in Azure for database the performance can be increased significantly.

# Pending work

1. Memory management for huge number of records
2. Cleanup of codebase
3. Better management of log files and tuning of status mail
4. Check accuracy of data and bug fixing
5. Azure deployment

# Known issues

1. Queue count for api requests is sometimes becoming negative at the end. This should always be zero.
2. The exact count of total api requests, api responses and api errors may not be correct.
3. Logging, error handlings, error reporting and logs management need to be improved.
4. Calculated data accuracy is not 100% guaranteed this is 90% correct.
5. The app inserts in database table PackageHistory a json object in column ActivityJson for reporting purposes. There are rows in table even if ActivityJson is null. Rows with null ActivityJson should not be inserted.
6. FedEx status has issues of more than 50 characters which gives overflow error in database. In present code the status of FedEx records is hand crafted to ‘delivered’, ‘returned’, ‘exception’ , ‘inTransit’, ‘noStatus’, ‘readyForPickup’ and ‘orderProcessed’.
7. When there are huge records (> 50000) there appears to be some lag between successful API responses and Database calls. Theoretically the api responses should equal to database calls but presently the database calls being made are less than successful api responses.

# Database scripts

Two new tables PackageHistory and PackageLog are to be created in database. The scripts for those two tables are given below:

/\*\*\*\*\*\* Object: Table [dbo].[PackageHistory] Script Date: 05/07/2018 5:00:08 PM \*\*\*\*\*\*/

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

CREATE TABLE [dbo].[PackageHistory](

[ID] [bigint] IDENTITY(1,1) NOT NULL,

[rn] [varchar](50) NOT NULL,

[TrackingNumber] [varchar](50) NOT NULL,

[ShippingAgentCode] [varchar](50) NOT NULL,

[ActivityJson] [nvarchar](max) NULL,

[IsDeleted] [bit] NOT NULL,

CONSTRAINT [PK\_PackageHistory] PRIMARY KEY CLUSTERED

(

[ID] ASC

)WITH (STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF) ON [PRIMARY]

) ON [PRIMARY] TEXTIMAGE\_ON [PRIMARY]

GO

ALTER TABLE [dbo].[PackageHistory] ADD DEFAULT ((0)) FOR [IsDeleted]

GO

/\*\*\*\*\*\* Object: Table [dbo].[PackageLog] Script Date: 05/07/2018 5:00:48 PM \*\*\*\*\*\*/

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

CREATE TABLE [dbo].[PackageLog](

[Id] [bigint] IDENTITY(1,1) NOT NULL,

[ApiRequests] [int] NOT NULL,

[ApiResponses] [int] NOT NULL,

[ApiErrors] [int] NOT NULL,

[DbRequests] [int] NOT NULL,

[DbResponses] [int] NOT NULL,

[DbErrors] [int] NOT NULL,

[StartTime] [varchar](50) NULL,

[EndTime] [varchar](50) NULL,

[Duration] [varchar](20) NULL,

CONSTRAINT [PK\_PackageLog] PRIMARY KEY CLUSTERED

(

[Id] ASC

)WITH (STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF) ON [PRIMARY]

) ON [PRIMARY]

GO

ALTER TABLE [dbo].[PackageLog] ADD CONSTRAINT [DF\_Table\_1\_Apirequests] DEFAULT ((0)) FOR [ApiRequests]

GO

ALTER TABLE [dbo].[PackageLog] ADD CONSTRAINT [DF\_PackageLog\_ApiResponses] DEFAULT ((0)) FOR [ApiResponses]

GO

ALTER TABLE [dbo].[PackageLog] ADD CONSTRAINT [DF\_PackageLog\_ApiErrors] DEFAULT ((0)) FOR [ApiErrors]

GO

ALTER TABLE [dbo].[PackageLog] ADD CONSTRAINT [DF\_PackageLog\_DbRequests] DEFAULT ((0)) FOR [DbRequests]

GO

ALTER TABLE [dbo].[PackageLog] ADD CONSTRAINT [DF\_PackageLog\_DbResponses] DEFAULT ((0)) FOR [DbResponses]

GO

ALTER TABLE [dbo].[PackageLog] ADD CONSTRAINT [DF\_PackageLog\_DbErrors] DEFAULT ((0)) FOR [DbErrors]

GO