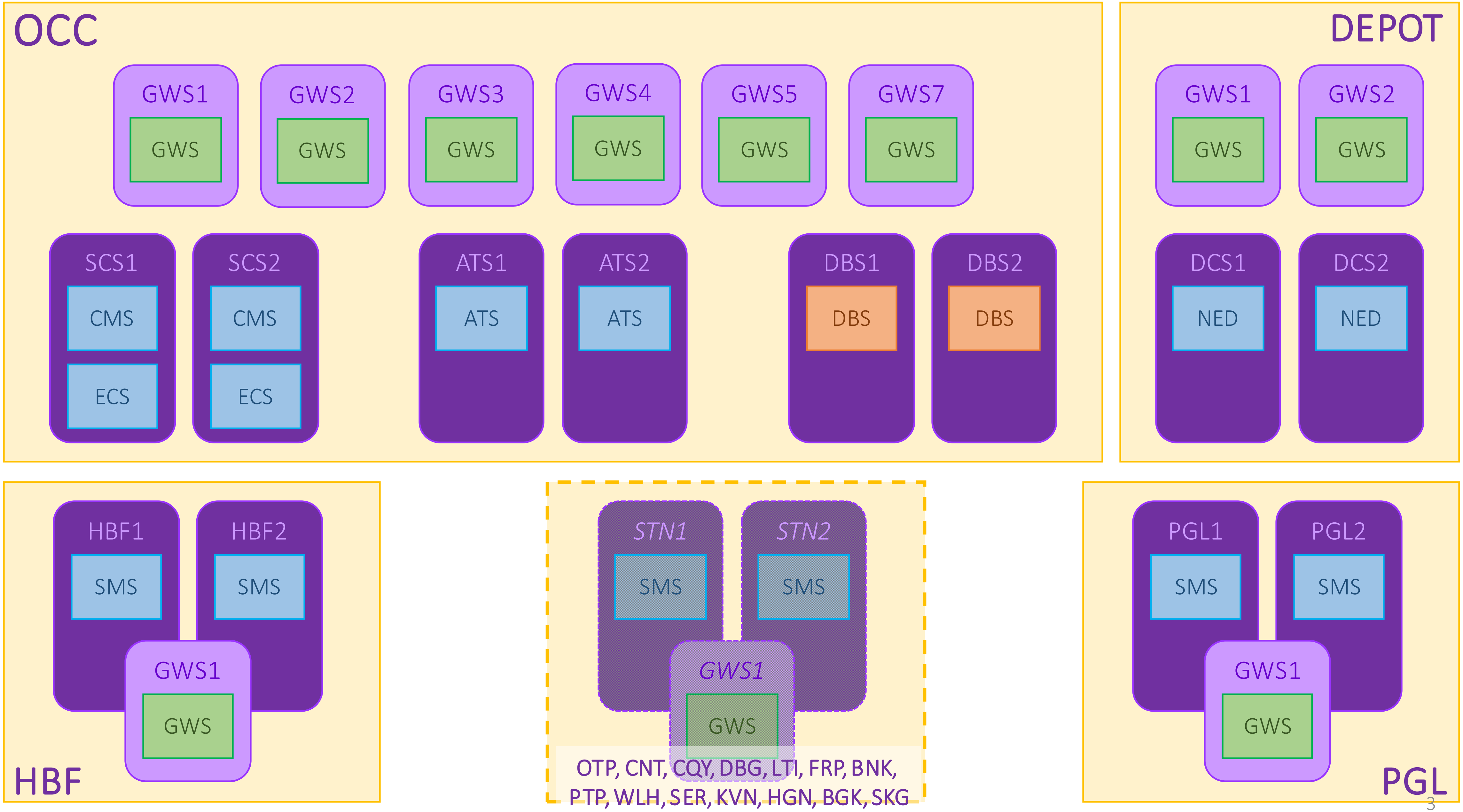
Guide to Parsing & Cleaning ISCS Event Logs

Updated as of 19th April 2022

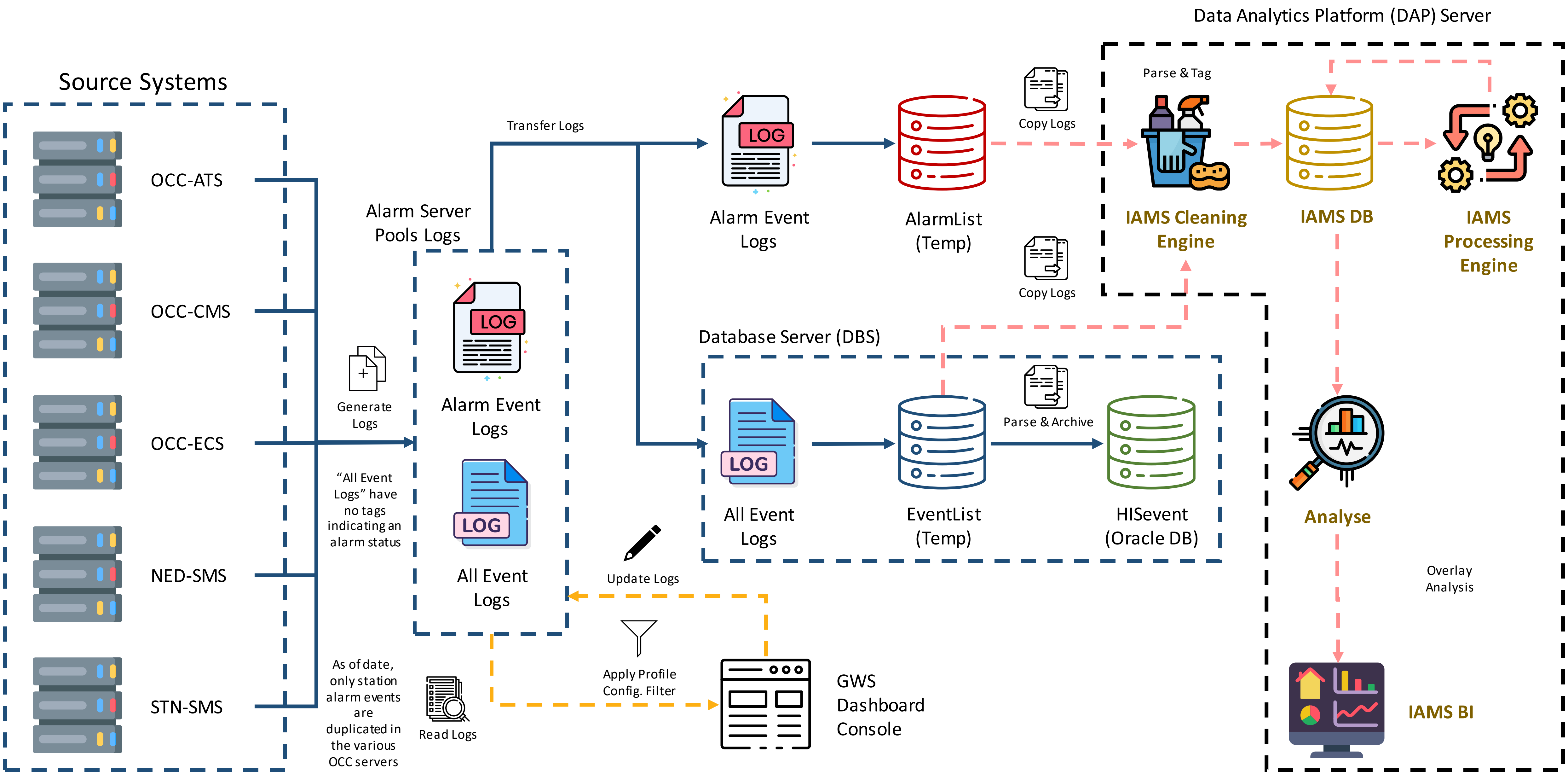
Preprocessing Overview

- File Preparation
    - Script(s):
        - The Great Unzipper.ipynb
    - Purpose:
        - For batch processing only where log files are compressed into multiple 7zip folders and files
        - This script would decompress the files and folder layers, placing the files in multiple fixed size single layer folders
- File Preprocessing
    - Script(s):
        - dataIngestion v2.x_packagedVer.ipynb (Default)
        - dataIngestion v2.x_packagedVer-AlarmList.ipynb (Optimised for Alarm Events)
        - dataIngestion v2.x_packagedVer-SQL_ServerOutput.ipynb (Default | Export to SQL Server instead of CSV)
        - dataIngestion v2.x_packagedVer-AlarmList-SQL_ServerOutput.ipynb (Optimised for Alarm Events | Export to SQL Server instead of CSV)
    - Purpose:
        - Perform basic preprocessing of event log data into tabular data format for further processing
        - Perform optional step to aggregate and export log data in CSV format using parallel processing
        - The production environment is meant to have near real time performance
- Baseline Alarm Tagging
    - Script(s):
        - Baseline Alarm Tagging v1.4.ipynb
    - Purpose:
        - To be used after the File Preprocessing step
        - Performs basic alarm tagging of events
        - Performs tagging of nuisance events
        - The production environment is meant to have near real time performance

ISCS Servers & Workstations



ISCS / IAMS Platform Design Overview



Data Analytics Platform (DAP) Server

Source Systems

OCC-ATS

OCC-CMS

OCC-ECS

NED-SMS

STN-SMS

Transfer Logs

Alarm Server Pools Logs

Generate Logs

Alarm Event Logs

All Event Logs

"All Event Logs" have no tags indicating an alarm status

As of date, only station alarm events are duplicated in the various OCC servers

Read Logs

Apply Profile Config. Filter

Update Logs

GWS Dashboard Console

Alarm Event Logs

AlarmList (Temp)

Copy Logs

Copy Logs

Database Server (DBS)

All Event Logs

EventList (Temp)

Parse & Archive

HISevent (Oracle DB)

Parse & Tag

IAMS Cleaning Engine

IAMS DB

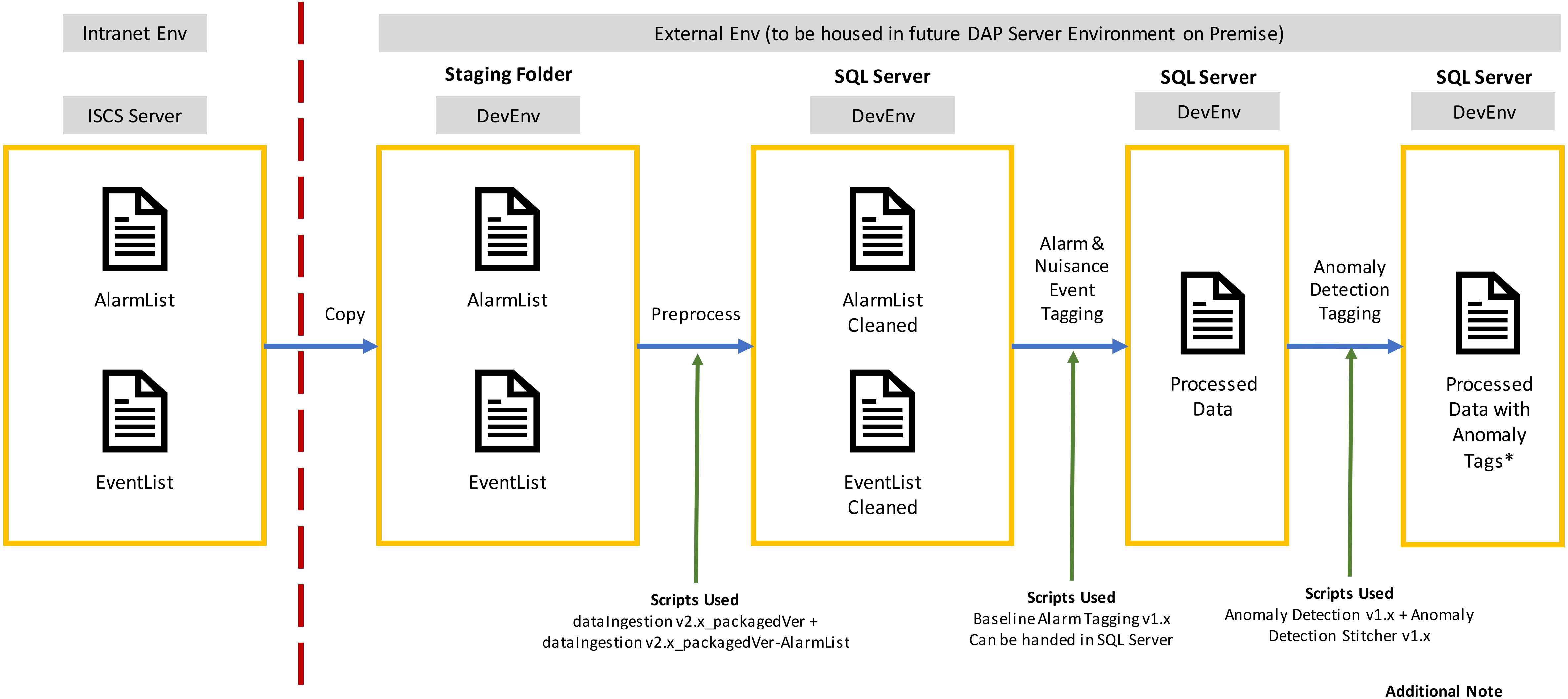
IAMS Processing Engine

Analyse

Overlay Analysis

IAMS BI

Dev WorkFlow For Anomaly Detection



Intranet Env

External Env (to be housed in future DAP Server Environment on Premise)

ISCS Server

AlarmList

EventList

Copy

Staging Folder

DevEnv

AlarmList

EventList

Preprocess

SQL Server

DevEnv

AlarmList Cleaned

EventList Cleaned

Alarm & Nuisance Event Tagging

SQL Server

DevEnv

Processed Data

Anomaly Detection Tagging

SQL Server

DevEnv

Processed Data with Anomaly Tags*

Scripts Used
dataIngestion v2.x_packagedVer +
dataIngestion v2.x_packagedVer-AlarmList

Scripts Used
Baseline Alarm Tagging v1.x
Can be handed in SQL Server

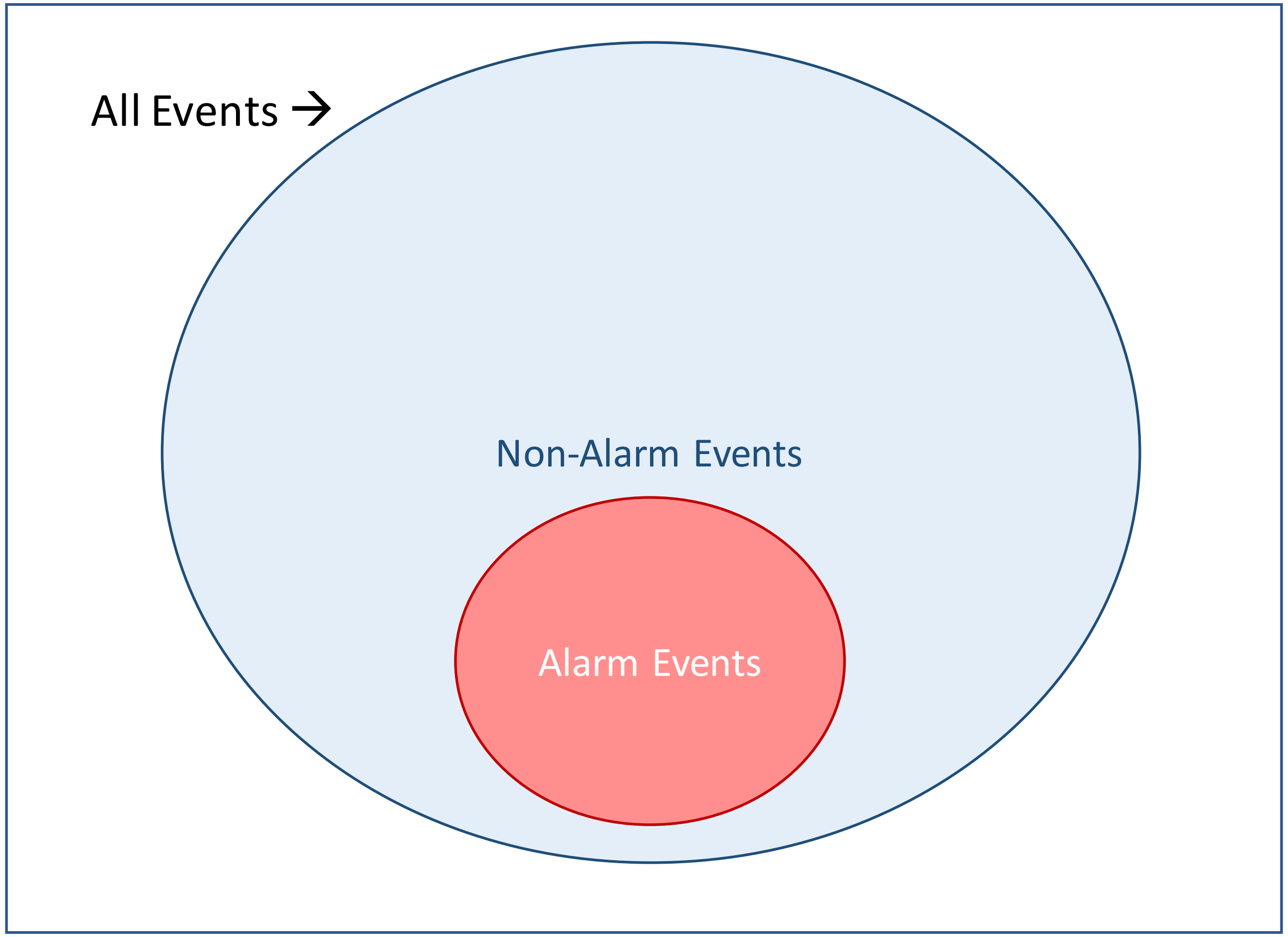
Scripts Used
Anomaly Detection v1.x + Anomaly Detection Stitcher v1.x

Note
In the original batch run, the processed files are exported as CSV files, which makes indexing and querying the files computationally expensive. Going forward, we would be exploring the storage of the data in SQL Server for the production version in the DAP server, though the intermediate processing steps of Alarm Tagging and Nuisance Event Tagging may be performed via SQL Server or Kafka.

Additional Note
Processed data + processed data with anomaly tags can then be piped to Splunk or Power BI for further analysis after being stored in SQL Server

ISCS Log Data

Relationship between Alarms & Events

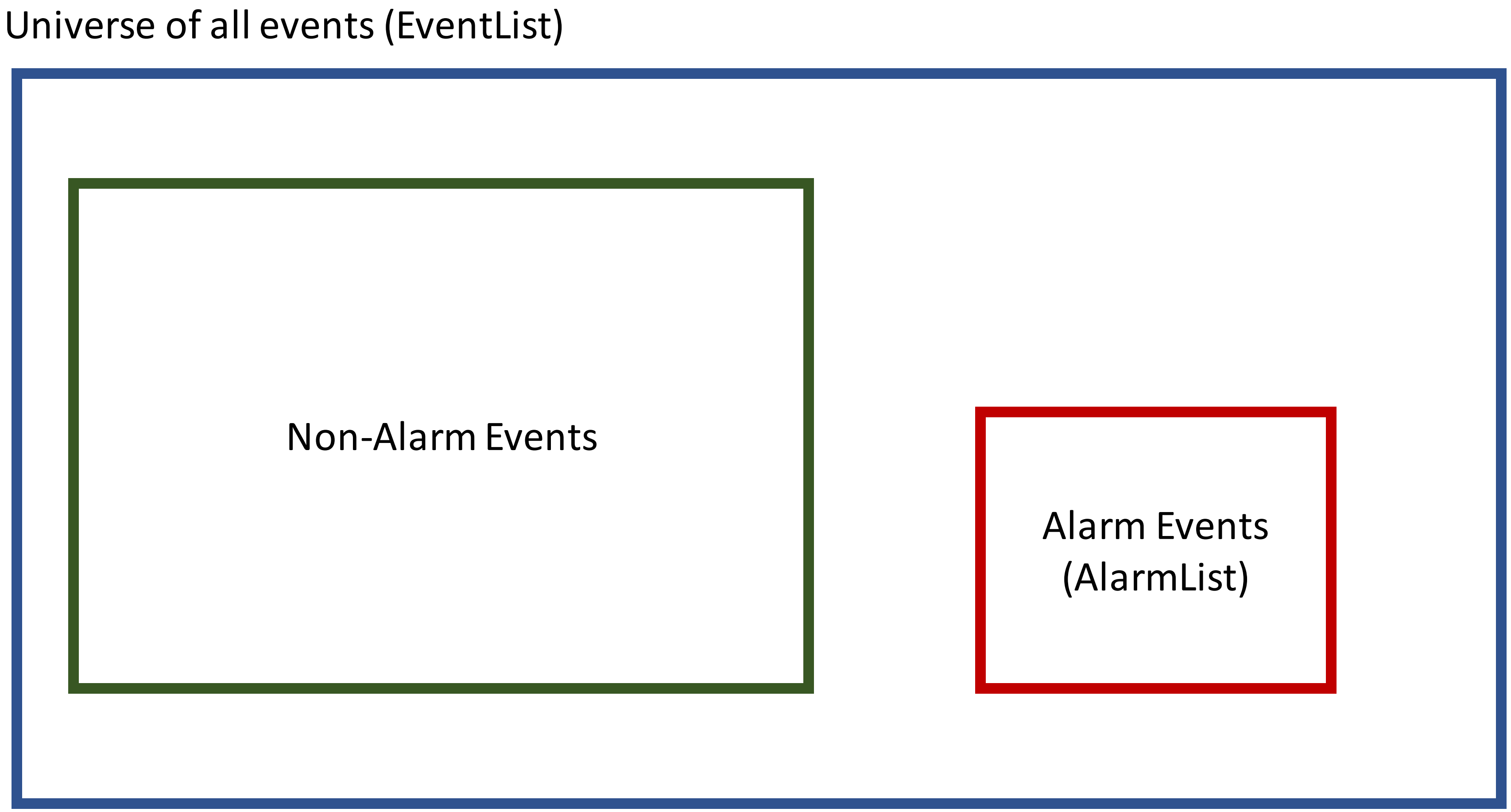


Note

1. Alarm Logs under AlarmList is a duplicated subset of Event Logs under EventList
2. In other words:
    1. All alarm logs are events logs; but
    2. Not all event logs are alarm logs
3. In practice, due to cut off errors in the log transmission, not all alarm logs can be matched with an event from EventList and vice versa at the tail ends of the block being sampled

Alarm Events vs Non-Alarm Events

General Definition



Universe of all events (EventList)

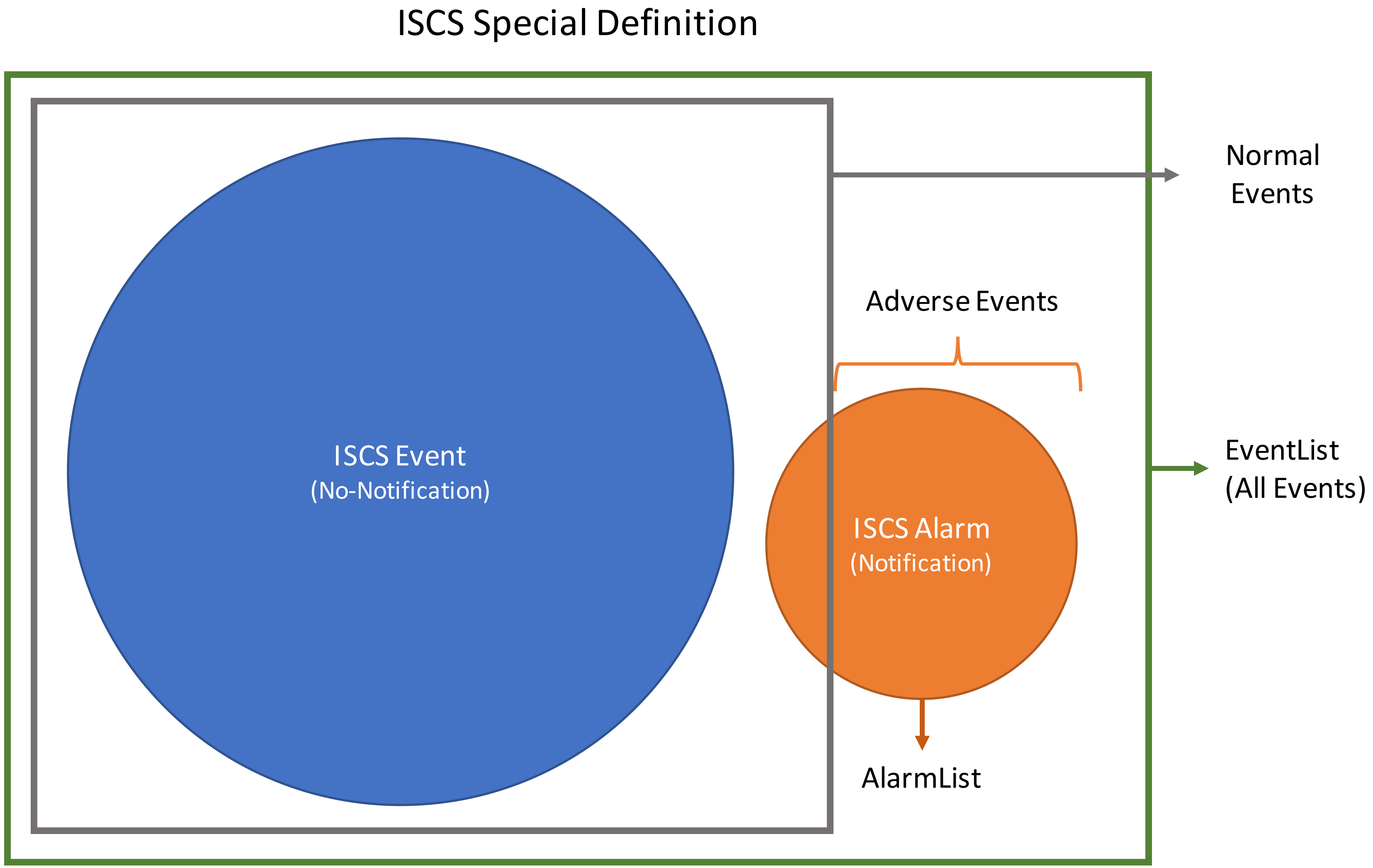
Non-Alarm Events

Alarm Events (AlarmList)

Note

1. ICS Data Logging
    1. ISCS only records changes in alarm states and the associated timestamp;
    2. Rather than the current state of all systems at a point in time
    3. There may be a 10-20 min lag during peak periods from the time an event occurs to the point an output log file is generated which can be read by IAMS
2. Normal definitions
    1. **Event** = A thing that happens; it can be good, bad or neutral
    2. **Alarm** = A notification event concerning an adverse scenario
3. The definition of alarms and events are a misnomer in ISCS
    1. **“Alarms” from AlarmList** are generally considered to be Alarm Events
    2. However, “Alarms” are still generated when the system normalizes from an abnormal state, which means that “Alarms” are just **alert notifications**
    3. **“Events” from EventList** refers to **“All Events”**, without distinction between whether it is an alarm of not

Alarm Events vs Non-Alarm Events



Note

1. ICS Data Logging
    1. ISCS only records changes in alarm states and the associated timestamp;
    2. Rather than the current state of all systems at a point in time
    3. There may be a 10-20 min lag during peak periods from the time an event occurs to the point an output log file is generated which can be read by IAMS
2. Normal definitions
    1. **Event** = A thing that happens; it can be good, bad or neutral
    2. **Alarm** = A notification concerning an **adverse** event
3. The definition of alarms and events are a misnomer in ISCS
    1. **"Alarms" from AlarmList** are generally considered to be Alarm Events
    2. However, "Alarms" are still generated when the system normalizes from an abnormal state, which means that **"Alarms" are just alert notifications**
    3. **"Events" from EventList** refers to **"All Events"**, without distinction between whether it is an alarm of not

Overview of ISCS Data

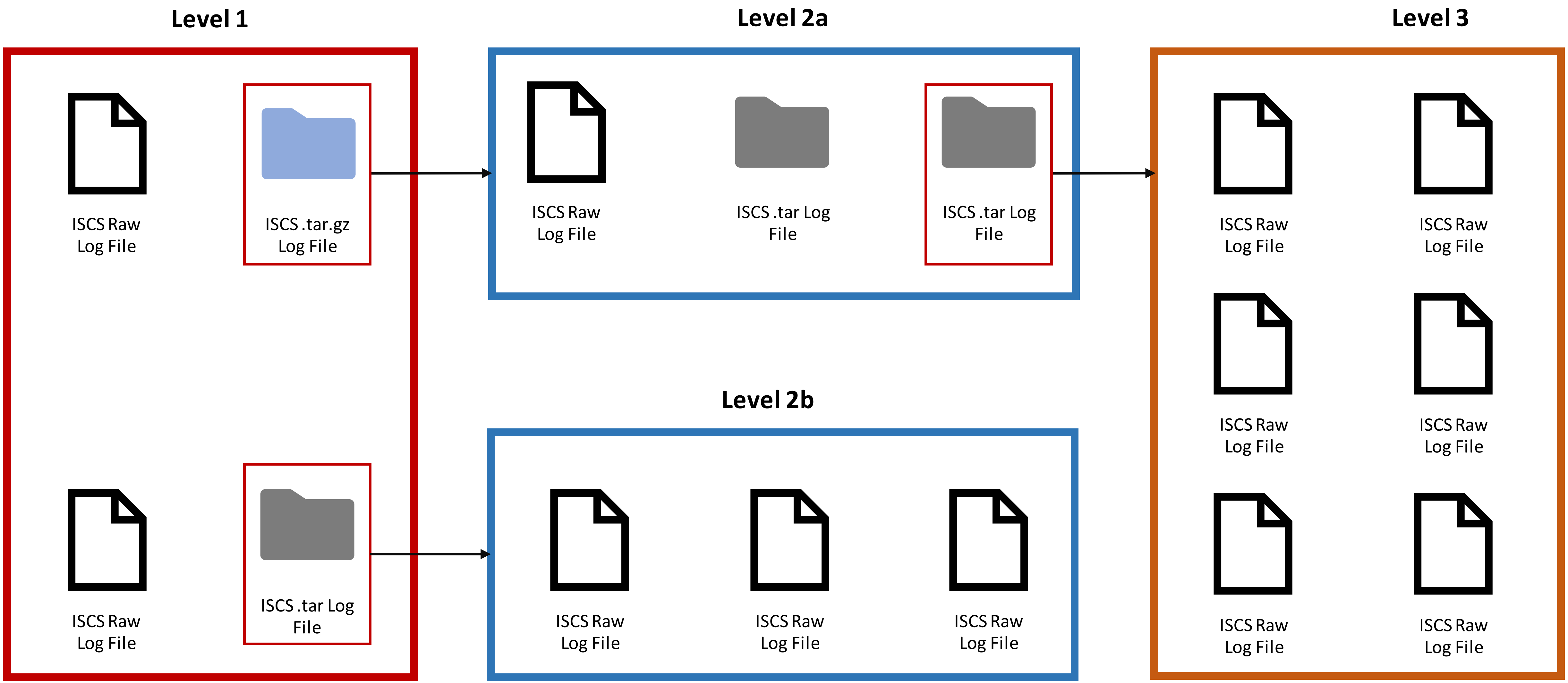
- The ISCS Platform gets its data from multiple source servers
    - ATS – Rolling Stock and Track Related Systems
    - CMS – General Systems
    - ECS – Environment Control System
- These servers are replicated at varying extents from their source geographical locations (stations and depots) to the Operati ons Command Centre (OCC)
- ISCS logs are stored in 3 formats
    - Event List: All events regardless of its alarm status; these files are only stored temporarily in the ISCS server
    - Alarm List: A duplicated subset of Event List, containing only alarm events ; these files are only stored temporarily in the ISCS server
    - HISevent: A semi-processed version of event list data stored for long term archival
- The ISCS log data has no indication on whether it is an alarm or not, and the ISCS alarm definition logic is not available, h ence the only way to differentiate alarm events from regular events would be to perform a comparison of raw alarm logs and raw event l ogs
    - Synchronisation of the parsing of both log files needs to be maintained for real-time systems, with the alarm list files needing to be processed ahead of time
    - Optimisation has been made to reduce unnecessary processing for the alarm list files, particularly on the “message” field, but otherwise t he scripts to process both types of files are identical
- ISCS only logs in changes to event state in the form of discrete data
- Basic nuisance alarm tagging can then be performed after alarm tagging
- The production environment is meant to have near real time performance

File Preparation

File Preparation Overview

- Issue:
  - Log files in ISCS are generated 1 file at a time, consolidating events within an approximately 1 second interval.
  - However, if there are no events generated within the 1 second interval, no file would be generated
  - Batches of these log files would be archived in a compressed tar.gz format
  - Each compressed tar.gz file may contain multiple subdirectories layers of more log files in the form of a compressed tar.gz file
    - The volume of files makes it very onerous to click through thousands of such folders to inspect for more tar.gz files
  - For the purpose of R&D, the log files would be sent in a mixture of ISCS log files, .tar files and .tar.gz files, which are not in chronological order
    - This is due to the manual intervention for data collection whereas the real time system extracts the files at a more frequent interval before compression could take place, hence there would not be any file compression
  - Decompression of tar.gz files
    - Tar.gz file → .tar file → assorted file(s) (Raw Log file)
  - Files need to be decompressed to the original log file format for it to be ingested
  - Log files are small in size but voluminous in nature
    - Having a large number of files in a single directory layer would negatively impact performance as the OS's indexer would require exponentially higher computing power as the list grows longer
    - Hence, files needs to be partitioned into smaller sub-folders to be more efficiently processed

File Preparation Overview – Folder Structure



File Preparation Treatment

- Method Overview:
    1. File extraction loop:
        1. Get list of .tar and tar.gz files in the master directory (including descendant directories)
        2. Loop through each file in list to extract file contents
    2. Repeat file extraction loop till there are no more .tar and tar.gz files left in the master directory (including descendant directories)
    3. Loop through each file in the master directory (including descendant directories) and transfer it into a new subfolder from the master directory in batches of 50K files
        1. New subfolders are created accordingly
        2. Subfolder names are to be created dynamically in running order
    4. Delete any empty folders

File Preprocessing

Raw Data Issues

```
1. [blank]

2. Begin Notification: 10/27/20 03:09:11.754 1603735751 754218

3. Number of updated DataSet : 1

4. -162(+) : -162 AlarmId:-162 AutoId:-9223372036329635829 UserID:0 EquipmentName::OCC:COMS:RADS_0001 Value:A ValueState:1
   AcknowledgeRequired:1 Severity:5 Shelve:0 Hidden:0 Message:COM/NED/1116/SCN$;BS_1_14/SRS_1/Shelf_1/Port_1 from ScEquip (Equipment) :
   $;MAJOR$; Theme:-1 EquipmentDate:1603735751 534215 AcquisitionDate:1603735751 534215 SCSTime:1603735751 534215 FunctionalCategory:46
   GeographicalCategory:1 AckAutomatonPointer: Environment:OCCCMS User1: User2: DssEventType:

5. End Notification: 10/27/20 03:09:11.754
```

Notes on Event Log File

1. The log file data is not in a neat tabular format; and doesn't have a consistent single delimiter throughout
2. The 1st row is blank; and can be ignored
3. The 2nd row contains information on the timestamp on which the event log file was sent to the server
4. The 3rd row contains the number of event logs within the same file; and can be ignored
5. The 4th row all the way to Nth - 1 row contains event log information (1 log per row); each attribute is delimited primarily by space " "(\s); and the "message" attribute is further delimited by "$", albeit not standardised to account for null values too
6. Some log entries denoted with "-" enclosed within the parenthesis after the AlarmID value are redundant and needs to be removed
7. The Nth row contains information on the timestamp on which the event log file was received by the server
8. Events recorded are discrete and only record a change of state in the asset
9. File Storage
   1. All events are stored in the folder EventList, with not tagging on whether each event is an alarm or not
   2. A duplicate list of alarm events are stored in the folder AlarmList – further meta tagging of logs in EventList required

Raw Data issues

DateTime with ms resolution

UnixTime in seconds resolution with decimal place; can be used as file serial number

1. [blank]
2. Begin Notification: 10/27/20 03:09:11.754 1603735751 754218
3. Number of updated DataSet : 1

The number of log entries in a single log file

Log file metadata to each row

    1. -162(+) : -162
    2. AlarmId:-162

Alarm ID is unique to environment up to a specific time duration, hence not useful as an event identifier on its own; (+) suffix refers to a new event entry; (=) suffix refers to an updated event entry; (-) suffix refers to a deleted entry due to memory limitations

    3. AutoId:-9223372036329635829
    4. UserID:0
    5. EquipmentName::OCC:COMS:RADS_0001
    6. Value:A
    7. ValueState:1
    8. AcknowledgeRequired:1
    9. Severity:5
    10. Shelve:0
    11. Hidden:0
    12. Message:COM/NED/1116/SCN$;BS_1_14/SRS_1/Shelf_1/Port_1 from ScEquip (Equipment) : $;MAJOR$;

Delimited by dollar sign ($)

    13. Theme:-1
    14. EquipmentDate:1603735751 534215
    15. AcquisitionDate:1603735751 534215
    16. SCSTime:1603735751 534215

UnixTime in seconds resolution with decimal place

    17. FunctionalCategory:46
    18. GeographicalCategory:1
    19. AckAutomatonPointer:
    20. Environment:OCCCMS
    21. User1:
    22. User2:
    23. DssEventType:

4. End Notification: 10/27/20 03:09:11.754

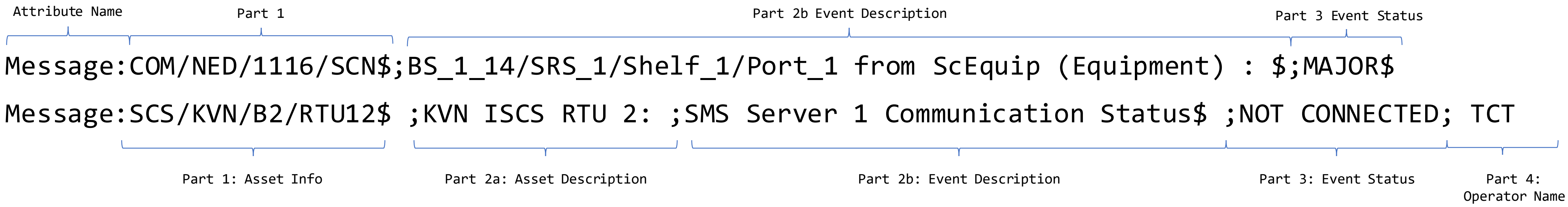
Log file metadata to be appended to each row

DateTime with ms resolution

- Unix time value is pegged to GMT time (GMT+0) whereas human readable time is in local time (GMT+8). Hence Unix time values needs adjustments to be in sync with local time values. However, there are rare cases where the Unix time values is (GMT-1), though such cases are ignored as no discernable pattern could be found to address it.

- Delimited by space (\s), except for unix time values which has a space separating the microseconds resolution in time;
- Each log entry is denoted as 1 row per entry, but in this case, it has been broken down here into indented rows for ease of reading;
- Removal of the attribute name after delimiting and appending the relevant attribute headers is required too;
- Items highlighted in red are deemed to be irrelevant, and shall be dropped

Raw Data Issues



```
Message:COM/NED/1116/SCN$;BS_1_14/SRS_1/Shelf_1/Port_1 from ScEquip (Equipment) : $;MAJOR$
Message:SCS/KVN/B2/RTU12$ ;KVN ISCS RTU 2: ;SMS Server 1 Communication Status$ ;NOT CONNECTED; TCT
```

Notes

1. Unlike the other fields, the [Message] field is not delimited by space but by the dollar sign instead ($) and has numerous spaces in them, hence require it to be spliced out and parsed separately
2. Message Log components have inconsistent formatting
    1. It cannot always be nicely delimited into 4 main parts
        1. Part 1: Asset Info
        2. Part 2: Asset Description + Event Description
        3. Part 3: Event Status
        4. Part 4: Operator Name
    2. Part 1 (Asset Info) cannot be consistently delimited
        1. Default: Asset Class / Location / Sub-Location / Asset Sub-Class + ID
        2. Exceptions:
            1. Delimited by underscore “_”
            2. Missing sub-components

Data Preprocessing Overview

1. Preprocessing script is to be packaged into a single function cleaningScript_vector() to allow for reuseability
2. Event logs are to be processed on a file by file basis
3. Parallel processing is then used to process batches of 50K files simultaneously with cleaningScript_vector() before being merged into a single file for export (1 export file per batch of 50K source files)

Initialise Parameters

- Create a list of all the target output column names [col_names]
  - This list contains additional fields generated from the data
  - This list is used to create an empty dataframe if there are no valid events
- Create a list of all column names of the fields to be extracted from the log files [headerList_core]
  - This is used to parse the data accordingly later
  - Not all the fields would be used, some fields are intermediary and it lacks the additional custom fields generated for final output, and some hence this list of column names would differ from [col_names]

Initialise Parameters

[col_names]

1. 'ENTRY_CODE_SUFFIX'
2. 'ENTRY_CODE'
3. 'ALARM_ID'
4. 'USER_ID'
5. 'EQUIPMENT_NAME'
6. 'VALUE'
7. 'VALUE_STATE'
8. 'ACKNOWLEDGEMENT_REQUIRED'
9. 'SEVERITY'
10. 'HIDDEN'
11. 'THEME'
12. 'EQUIPMENT_DATE'
13. 'ACQUISITION_DATE'
14. 'SCS_TIME'
15. 'FUNCTIONAL_CATEGORY'
16. 'GEOGRAPHICAL_CATEGORY'
17. 'ENVIRONMENT'
18. 'USER1'
19. 'ASSET_ID_RAW'
20. 'ASSET_DESCRIPTION'
21. 'EVENT_DESCRIPTION'
22. 'EVENT_STATUS'
23. 'OPERATOR_INITIALS'
24. 'ASSET_DESC_CAT'
25. 'EVENT_DESC_CAT'
26. 'TrainID'
27. 'CarID'
28. 'ServiceID'
29. 'AssetClass'
30. 'AssetSubClass'
31. 'DATETIME_SENT'
32. 'DATETIME_RECEIVED'
33. 'TIME_CODE'

[headerList_core]

1. 'ENTRY_CODE_RAW'
2. 'ENTRY_CODE_SUFFIX'
3. 'ENTRY_CODE'
4. 'ALARM_ID'
5. 'AUTO_ID'
6. 'USER_ID'
7. 'EQUIPMENT_NAME'
8. 'VALUE'
9. 'VALUE_STATE'
10. 'ACKNOWLEDGEMENT_REQUIRED'
11. 'SEVERITY'
12. 'SHELVE'
13. 'HIDDEN'
14. 'THEME'
15. 'EQUIPMENT_DATE’
16. 'EQUIPMENT_DATE0’
17. 'ACQUISITION_DATE'
18. 'ACQUISITION_DATE0'
19. 'SCS_TIME'
20. 'SCS_TIME0'
21. 'FUNCTIONAL_CATEGORY'
22. 'GEOGRAPHICAL_CATEGORY'
23. 'ACKNOWLEDGEMENT_AUTOPOINTER'
24. 'ENVIRONMENT’
25. 'USER1'
26. 'USER2'
27. 'DSS_EVENT_TYPE'

Ingesting the Data

- Issue:
  - Event logs lack any doc type classification tag
  - Event logs can be approximated to be treated as a pseudo CSV file with some properties akin to JSON format
- Method Overview:
  - File ingestion method
    - Python’s native `open(fileName, ‘r’)` function
    - Each row would be read in as a continuous string which needs to be parsed further
    - Close file after reading data
  - Note:
    - Pandas’ `read_csv()` function is unable to work for all event logs for unknown reasons hence Python’s native function must be used instead

Initialise Data Splicing Functions

- Key Functions
  - replaceTextBetween1_series
  - parseMessageField
  - parseMetaData

Initialise Data Splicing Functions

replaceTextBetween1_series

1. Output: raw event info sans message field
2. Splices out the message field from log event string to allow easing parsing of the other fields
3. This is achieved by specifying “Message:” and “Theme:” as the start and end codon strings to splice out the message field, retaining the rest of the event information

parseMessageField

1. Output: ASSET_ID_RAW, ASSET_DESCRIPTION, EVENT_DESCRIPTION, EVENT_STATUS, OPERATOR_INITIALS
2. Splices out the message field from log event string to be parsed
3. Splicing is achieved by by specifying “Message:” and “Theme:” as the start and end codon strings to splice out the message field
4. Remove any semi-colons (;) in the extracted message info
5. Delimit message info by dollar sign ($) into 4 parts
    1. Part 1: Asset Info
    2. Part 2: Asset Description + Event Description
    3. Part 3: Event Status
    4. Part 4: Operator Name
6. Delimit Part 2 into its constituent components (ASSET_DESCRIPTION, EVENT_DESCRIPTION) by colon (:)

Initialise Data Splicing Functions

parseMetaData

1. Output: DATETIME_SENT, DATETIME_RECEIVED, TIME_CODE
2. Header Meta Data
    1. Splice out string after “Beginning Notification: ”
    2. Delimit spliced out string into 3 parts using space
        1. Log Sent Time in human readable time
        2. Log Sent Time in Unix Time (Part 1) – seconds resolution
        3. Log Sent Time in Unix Time (Part 2) – microseconds resolution component
    3. Join the 2 Log Sent Time in Unix Time components into a single value, using the period sign “.” as a decimal place separator
3. Footer Meta Data
    1. Splice out string after “End Notification: ”
    2. Delimit spliced out string into 3 parts using space
        1. Log Received Time in Unix Time (Part 1) – seconds resolution
        2. Log Received Time in Unix Time (Part 2) – microseconds resolution component
    3. Join the 2 Log Received Time in Unix Time components into a single value, using the period sign “.” as a decimal place separator

Extract Key Components to Parse

1. Save values from rows containing meta data “f_meta1” (header) & “f_meta2” (footer)
2. Remove redundant rows (first 3 rows and last row) retaining the core event info [fileContents]
3. Check if the string “first notification on online” is found in the meta data,
    1. If so,
        1. Create an empty dataframe using the [col_names] list defined earlier
            1. This is because there is no valid data in such cases; and an empty dataframe makes generalization of processed events a lot easier for joining regardless of their contents.
        2. Remove any redundant intermediary variables
        3. End all further processing.
    2. Else,
        1. Convert [fileContents] to a single column dataframe, df, with column named “rawData”
        2. Continue additional preprocessing

Check for Row Corruption

1. Remove any leading or lagging non-printable characters in “rawData” field
2. Check if the rows if they start with “º÷” or ends with “(.UàÑ” (these substrings don’t appear on the same row),
    1. If so,
        1. Split for rows containing “(.UàÑ” as df_head,
            1. Split off “(.UàÑ” from the rest of the row by a pattern match, and retain the rest of the string
            2. Remove the substring “;(” from the rest of the string
        2. Split for rows containing “º÷” as df_tail,
            1. Split off “º÷” by dropping the first 3 characters of the string (direct targeting is not possible due to the unknown symbol used), and retain the rest of the string
        3. Reset indices of df_head and df_tail
        4. Merge df_head and df_tail with df
        5. Filter out original corrupted rows
        6. Remove “message” field from df[“rawData”] using replaceTextBetween1_series() as df[“text0”]
        7. Delete any redundant intermediate variables
    2. Else,
        1. Remove “message” field from df[“rawData”] using replaceTextBetween1_series() as df[“text0”]
        2. Delete any redundant intermediate variables

Check for Row Corruption

- Examples of Event Data Corruption:

1.
   ```
   -162(+) : -162 AlarmId:-162 AutoId:-9223372036329635829
   UserID:0 EquipmentName::OCC:COMS:RADS_0001 Value:A
   ValueState:1 AcknowledgeRequired:1 Severity:5 Shelve:0
   Hidden:0
   Message:COM/NED/1116/SCN$;BS_1_14/SRS_1/Shelf_1/Port_1
   from ScEquip (Equipment) : $;MAJOR$; (.UàÑ [unknown
   non-printable char]
   ```

2.
   ```
   º÷□ Theme:-1 EquipmentDate:1603735751 534215
   AcquisitionDate:1603735751 534215 SCSTime:1603735751
   534215 FunctionalCategory:46 GeographicalCategory:1
   AckAutomatonPointer: Environment:OCCCMS User1: User2:
   DssEventType:
   ```

Parse Main Body df[“text0”]

1. Delimit df[“text0”] by space, using [headerList_core] as the list of all the field names
2. Merge columns containing time values in Unix time as they are separated into 2 parts, by concatenating both columns into with a period sign “.” as a decimal point separator
    1. Time is broken up into 2 parts
        1. X seconds from epoch time
        2. X microseconds resolution component
    2. Affected fields
        1. df[“EQUIPMENT_DATE”]
        2. df[“ACQUISITION_DATE”]
        3. df[“ENTRY_CODE_SUFFIX”]
3. Extract entry code suffix as df[“ENTRY_CODE_SUFFIX”] from df[“ENTRY_CODE_RAW”]
4. Drop redundant columns / variables

Parse Main Body df[“text0”]

5. Proof check – the all fields except the first 3 should be in this format:
    1. df[“attributeXYZ”] = “attributeXYZ:loremipsum01”, “attributeXYZ:loremipsum02”, …
    2. df[“attributeDT”] = “attributeDT:0123456789.012345”, “attributeDT:0123456789.012345”, …
6. Loop through each column to delimit each column by colon (:), retaining the 2nd half to remove the attribute name prefix
    1. Warning: Perform the split only once per column to avoid errors in extracting values from the df[“EQUIPMENT_NAME”] field as it has multiple colon “:” characters in its string values
7. Drop redundant columns / variables

Parse Main Body df[“text0”]

8. Drop any rows with a minus value (-), as those are duplicated entries made when the ISCS system deletes an event from active memory due to memory limitations – not a real event
    1. AlarmList
        1. (+): New event entry
        2. (=): Update to event entry
        3. (-): Deletion of event entry due to memory limit
    2. EventList
        1. (+): New event entry or update to event entry
        2. (-): Deletion of event entry due to memory limit
9. Reset index
10. Check if the dataframe is empty after dropping redundant rows
    1. If the dataframe is empty,
        1. Generate default empty dataframe using predefined column list [col_names]
        2. Stop further processing steps
    2. Else,
        1. Continue with the preprocessing of event logs

Parse Message Field

1. Use parseMessageField() on df[“rawData”] to extract the following components from the “message” field
    1. df[“ASSET_ID_RAW”]
    2. df[“ASSET_DESCRIPTION”]
    3. df[“EVENT_DESCRIPTION”]
    4. df[“EVENT_STATUS”]
    5. df[“OPERATOR_STATUS”]
2. Reorder values for cases where df[“EVENT_DESCRIPTION”] is missing by swapping empty df[“EVENT_DESCRIPTION”] values with df[“ASSET_DESCRIPTION”]

Parse Message Field

1. Exception handling to guess df[“ASSET_DESCRIPTION”] based on df[“EVENT_DESCRIPTION”]
   1. GWS Broadcast
      1. df[“EVENT_DESCRIPTION”] contains both “gws” and “msg”
   2. NelVisu
      1. df[“EVENT_DESCRIPTION”] contains “NelVisu”
   3. Train Radio
      1. df[“EVENT_DESCRIPTION”] contains both “TR___” and “radio”
   4. Trainborne Camera
      1. df[“EVENT_DESCRIPTION”] contains “Trainborne Camera”
   5. Trainborne Quad
      1. df[“EVENT_DESCRIPTION”] contains “Trainborne Quad”
   6. Tunnel LTG
      1. df[“EVENT_DESCRIPTION”] contains “Tunnel Light” and df[“ASSET_DESCRIPTION”] is empty
   7. Control Take Over for X
      1. df[“EVENT_DESCRIPTION”] contains “Control Take Over for”
   8. Traction Control
      1. df[“EVENT_DESCRIPTION”] contains “Close Control” and df[“ASSET_DESCRIPTION”] is empty
2. Define location names [locNamesList] and corresponding values [locNamesVal] to be replaced with as a set of stop words

Parse Message Field

[locNamesList]	[locNamesVal]	[locNamesList]	[locNamesVal]	[locNamesList]	[locNamesVal]	[locNamesList]	[locNamesVal]
NED		LTI		Concourse	SUBLOCATION	B2	
FRP		CQY		Mezzaninne	SUBLOCATION	B3	
SKG		BGK		Mid-Landing Entrance	SUBLOCATION	Entrance	SUBLOCATION
HGN		OCC		AL	SUBLOCATION	Mid Landing	
KVN		WLH		Dirty Area	SUBLOCATION	Mid-Landing	SUBLOCATION
SER		PTP		IAP	SUBLOCATION	Subway	SUBLOCATION
HBF		BNK		1st Storey	SUBLOCATION	Underpass Link	SUBLOCATION
DBG		PGL		2nd Storey	SUBLOCATION	Underpass To EXT'G STN	SUBLOCATION
OTP		TUNNEL		3rd Storey	SUBLOCATION	1st	
CNT		Sector		B1		2nd	

Parse Message Field

[locNamesList]	[locNamesVal]	[locNamesList]	[locNamesVal]	[locNamesList]	[locNamesVal]	[locNamesList]	[locNamesVal]
SUBLOCATIONN	SUBLOCATION	Underpass to EXT'G STN	SUBLOCATION				
SUBLOCATIONS	SUBLOCATION	-SUBLOCATION					
North End		SUBLOCATION-					
South End							
South Adjacent							
North Adjacent							
Mezzanine	SUBLOCATION						
Linkway	SUBLOCATION						
Smoke Free Lobby	SUBLOCATION						
Storey	SUBLOCATION						

Parse Message Field

3. Duplicate df[“ASSET_DESCRIPTION”] as df[“ASSET_DESC_CAT”]
4. Loop through the list of block words and substitute words in [locNamesList] and [locNamesVal] respectively to simplify the different categories in df[“ASSET_DESC_CAT”]
5. Remove all numeric characters in df[“ASSET_DESC_CAT”] to further simplify the categories
    1. Simplification of asset description data to asset description category would allow for macro analysis of trends without the use of complex NLP methods

Parse Message Field

6. Exception handling to further simplify df[“ASSET_DESC_CAT”] based on df[“ASSET_DESC_CAT”] value
   1. Replace “ kV” with “22 kV”
   2. Replace “at KV SW” with “at 22 kV SW”
   3. Replace “DC V” with “DC 1500 V”
   4. Replace “SUBLOCATION SUBLOCATION” with “SUBLOCATION”
   5. Replace “(” with “(”
   6. Replace r“\A(:)” with “”
   7. Replace “Cameras” with “Camera”

Parse Message Field

7. Exception handling to further simplify df[“ASSET_DESC_CAT”] based on df[“ASSET_DESCRIPTION”] value
    1. If df[“ASSET_DESCRIPTION”] is “22 kV Feeder CB” impute df[“ASSET_DESC_CAT”] with “22 kV Feeder CB”
    2. If df[“ASSET_DESCRIPTION”] is “22 kV Loop CB” impute df[“ASSET_DESC_CAT”] with “22 kV Loop CB”
    3. If df[“ASSET_DESCRIPTION”] is “22 kV Rectifier CB” impute df[“ASSET_DESC_CAT”] with “22 kV Rectifier CB”
    4. If df[“ASSET_DESCRIPTION”] is “DC 1500 V Backup HSCB” impute df[“ASSET_DESC_CAT”] with “DC 1500 V Backup HSCB”
    5. If df[“ASSET_DESCRIPTION”] is “DC 1500 V Bus Section” impute df[“ASSET_DESC_CAT”] with “DC 1500 V Bus Section”
    6. If df[“ASSET_DESCRIPTION”] is “DC 1500 V Feeder CB” impute df[“ASSET_DESC_CAT”] with “DC 1500 V Feeder CB”
    7. If df[“ASSET_DESCRIPTION”] is “DC 1500 V Rectifier CB” impute df[“ASSET_DESC_CAT”] with “DC 1500 V Rectifier CB”
    8. If df[“ASSET_DESCRIPTION”] is “DC 1500 V Inverter CB” impute df[“ASSET_DESC_CAT”] with “DC 1500 V Inverter CB”

Parse Message Field

8. Exception handling to further simplify df[“ASSET_DESC_CAT”] based on df[“ASSET_DESCRIPTION”] value
    1. If df[“ASSET_DESCRIPTION”] contains “CCTV Controller Power Supply” impute df[“ASSET_DESC_CAT”] with “CCTV Controller Power Supply”
    2. If df[“ASSET_DESCRIPTION”] contains “CBN Access Multiplexer” impute df[“ASSET_DESC_CAT”] with “CBN Access Multiplexer”
    3. If df[“ASSET_DESCRIPTION”] contains “CI Gas Panel” impute df[“ASSET_DESC_CAT”] with “CI Gas Panel”
    4. If df[“ASSET_DESCRIPTION”] contains “RI Gas Panel” impute df[“ASSET_DESC_CAT”] with “RI Gas Panel”
    5. If df[“ASSET_DESCRIPTION”] contains “CROSS-CONNECT ACCESS Multiplexer” impute df[“ASSET_DESC_CAT”] with “CROSS-CONNECT ACCESS Multiplexer”
    6. If df[“ASSET_DESCRIPTION”] contains “Electrically Supervised Valve” impute df[“ASSET_DESC_CAT”] with “Electrically Supervised Valve”
    7. If df[“ASSET_DESCRIPTION”] contains “Hosereel Pump” impute df[“ASSET_DESC_CAT”] with “Hosereel Pump”
    8. If df[“ASSET_DESCRIPTION”] contains “Level Fire Shutter” impute df[“ASSET_DESC_CAT”] with “Level Fire Shutter”
    9. If df[“ASSET_DESCRIPTION”] contains “Level Roller Shutter” impute df[“ASSET_DESC_CAT”] with “Level Roller Shutter”
    10. If df[“ASSET_DESCRIPTION”] contains “Main Fire Alarm Panel” impute df[“ASSET_DESC_CAT”] with “Main Fire Alarm Panel”
    11. If df[“ASSET_DESCRIPTION”] contains “Traffic Direction” impute df[“ASSET_DESC_CAT”] with “Traffic Direction”
    12. If df[“ASSET_DESCRIPTION”] contains “Tunnel LTG Ctrl Panel” impute df[“ASSET_DESC_CAT”] with “Tunnel LTG Ctrl Panel”
    13. If df[“ASSET_DESCRIPTION”] contains “Zone -” impute df[“ASSET_DESC_CAT”] with “ZONE SUBLOCATION”

Parse Message Field

9. Exception handling to further simplify df[“ASSET_DESC_CAT”] to strip out any residual location information
    1. Split and retain df[“ASSET_DESC_CAT”] before the breakpoint “ at ”
    2. Split and retain df[“ASSET_DESC_CAT”] before the breakpoint “ for ”
    3. Replace “SUBLOCATION-SUBLOCATION” with “”
    4. Replace “” with “SUBLOCATION”
    5. Replace “SUBLOCATION-” with “SUBLOCATION”
    6. Replace “-SUBLOCATION” with “-SUBLOCATION”
    7. Replace r“(at)$” with “”
    8. Replace r“(for)$” with “”
    9. Replace r“^(:)” with “”
10. Remove any redundant spaces (\s) in df[“ASSET_DESC_CAT”]

Parse Message Field

11. Duplicate df[“EVENT_DESCRIPTION”] as df[“EVENT_DESC_CAT”]
12. Loop through the list of block words and substitute words in [locNamesList] and [locNamesVal] respectively to simplify the different categories in df[“EVENT_DESC_CAT”]
13. Remove all numeric characters in df[“EVENT_DESC_CAT”] to further simplify the categories
    1. Simplification of event description data to event description category would allow for macro analysis of trends without the use of complex NLP methods
14. Remove any redundant spaces (\s) in df[“EVENT_DESC_CAT”]

Parse Message Field

15. Simplify existing df[“EVENT_DESC_CAT”] values
    1. If df[“EVENT_DESC_CAT”] contains “logged”, “Operator” and “NelVisu”, impute as “Operator Logged In/Out of NelVisu”
    2. Replace “ /, /...” with “”
    3. Replace “ /” with “”
    4. Replace “_:” with “”
    5. Replace r“([.]+){2}” with “”
    6. Replace r“(_+){2}” with “”
    7. Replace “::” with “”
    8. Replace “ : ” with “: ”
    9. Replace “@n” with “”
    10. Replace “ ,” with “”
    11. Replace “< >” with “”

Parse Message Field

15. Simplify existing df[“EVENT_DESC_CAT”] values (con’t)
    12. Replace “-:” with “:”
    13. Replace “^(:)” with “”
    14. Replace r“(-)\S” with “ - ”
    15. Replace “)” with “)”
    16. Replace “()” with “”
    17. Replace “ ump Rm” with “Pump Rm”
    18. Replace “ latform” with “ Platform”
    19. Replace r“( )+” with “ ”
    20. Replace “SUBLOCATION-SUBLOCATION” with “SUBLOCATION”
    21. Replace “SUBLOCATION-” with “SUBLOCATION”
    22. Replace “-SUBLOCATION” with “SUBLOCATION”

Parse Message Field

16. Exception handling to further simplify df[“EVENT_DESC_CAT”]
    1. If df[“EVENT_DESC_CAT”] contains “ at ” and “: ”
        1. Splice out and junk location info between “at” and “:” e.g. at LOCATION:
        2. replace “ : ” with “: ”
    2. If df[“EVENT_DESC_CAT”] contains “Gws” and “msg in”
        1. Splice out and junk location info after “msg in” and replace subphrase with “msg in SUBLOCATION”
    3. If df[“EVENT_DESC_CAT”] contains “Gws” and “bcast in”
        1. Splice out and junk location info after “msg in” and replace subphrase with “bcast in SUBLOCATION”
    4. If df[“EVENT_DESC_CAT”] contains “Train”, “Car”, “assigned”, “Manoeuvre”, impute as “Manoeuvre assigned to Train Car”
    5. If df[“EVENT_DESC_CAT”] contains “Train”, “Car”, “abandoned”, “Manoeuvre”, impute as “Manoeuvre assigned to Train Car”
    6. If df[“EVENT_DESC_CAT”] or df[“ASSET_DESC_CAT”] contains “Display of Free-Text”, impute as “Display of Free-Text”

Parse Message Field

16. Exception handling to further simplify df[“EVENT_DESC_CAT”] (con’t)
    7. If df[“EVENT_DESC_CAT”] contains “DVA version mismatch”, impute as “DVA version mismatch”
    8. If df[“EVENT_DESC_CAT”] contains “Automatic hand-over”, impute as “Automatic hand-over”
    9. If df[“EVENT_DESC_CAT”] contains “Automatic Hold Applied”, impute as “Automatic Hold Applied”
    10. If df[“EVENT_DESC_CAT”] contains “Communication between”, impute as “Communication between Nodes”

Parse Message Field

17. Remove train direction info in df[“EVENT_DESC_CAT”]
    1. E.g. N/B , N-N, N, S/B, S-S, S
18. Simplify control takeover scenarios for df[“EVENT_DESC_CAT”]
    1. If contains “Automatic Hold applied to TrainCar stalled” to “Automatic Hold applied to TrainCar stalled”
    2. If contains “Control Hand Over for ECS - Environmental Control System” to “Control Hand Over for ECS - Environmental Control System”
    3. If contains “Control Hand Over for ECS - Smoke Extraction System” to “Control Hand Over for ECS - Smoke Extraction System”
    4. If contains “Control Hand Over for ECS - Tunnel Ventilation System” to “Control Hand Over for ECS - Tunnel Ventilation System”
    5. If contains “Control Hand Over for SIG - Control Train ATC” to “Control Hand Over for SIG - Control Train ATC”
    6. If contains “Control Hand Over for SIG - Platform Equipment” to “Control Hand Over for SIG - Platform Equipment”
    7. If contains “Control Hand Over for SIG - Track Side Equipment” to “Control Hand Over for SIG - Track Side Equipment”
    8. If contains “Control Hand Over for TrainBorne CCTV” to “Control Hand Over for TrainBorne CCTV”
    9. If contains “Control Hand Over for TrainBorne PA” to “Control Hand Over for TrainBorne PA”
    10. If contains “Control Hand Over for TrainBorne PEC” to “Control Hand Over for TrainBorne PEC”
    11. If contains “Control Hand Over for TrainBorne PIS/VPIS” to “Control Hand Over for TrainBorne PIS/VPIS”
    12. If contains “Control Take Over for All Functions” to “Control Take Over for All Functions”
    13. If contains “Control Take Over for PIS - Passenger Information” to “Control Take Over for PIS - Passenger Information”

Parse Message Field

19. Simplify Operator Calls scenarios for df[“EVENT_DESC_CAT”]
    1. If contains “accepts a PEC call” to “OPERATOR accepts a PEC call”
    2. If contains “terminates all PEC calls” to “OPERATOR terminates all PEC calls”
    3. If contains “terminates PEC call” to “OPERATOR terminates PEC call”
20. Simplify df[“EVENT_DESC_CAT”] for the following scenarios
    1. If contains “Automatic Hold applied to TrainCar stalled” to “Automatic Hold applied to TrainCar stalled”
    2. If contains “Free all paths for Station” to “Free all paths for Station”
    3. If contains “Gama Status Request For an Atc” to “Gama Status Request For an Atc”
    4. If contains “change password on NelVisu” to “OPERATOR change password on NelVisu”
    5. If contains “Track-Side Atc Status Request for An Atc” to “Track-Side Atc Status Request for An Atc”

Parse Message Field

21. Simplify Operator Calls scenarios for df[“EVENT_DESC_CAT”]
    1. If contains “Train found at” and “instead of Train” to “Train found at SUBLOCATION instead of Train”
    2. If contains “Train still not a Man RTL” and “origin after wait period” to “Train still not a Man RTL origin after wait period”
    3. If contains “Timetable” and “download” to “Timetable download”
    4. If contains “Timetable” and “successfully autoloaded” to “Timetable successfully autoloaded”
22. Remove any redundant spaces (\s) in df[“EVENT_DESC_CAT”]

Extract Train Information

1. Extract train ID intermediary by extracting digits from the r“Train (\d+)” pattern in df[“EVENT_DESCRIPTION”] as df[“TrainID1”]
2. Extract train ID intermediary by extracting digits from the r“TR___(\d+)” pattern in df[“ASSET_ID_RAW”] as df[“TrainID2”]
3. Extract train ID intermediary by extracting digits from the r“TR___(\d+)” pattern in df[“EVENT_DESCRIPTION”] as df[“TrainID3”]
4. Merge df[“TrainID1”], df[“TrainID2”] and df[“TrainID3”] as a single field as df[“TrainID”], whilst removing null values in the intermediaries
5. Standardise null values in df[“TrainID”] as np.nan
6. Drop redundant variables

Extract Train Information

7. Extract car ID intermediary by extracting digits from the r“Car (\d+) pattern in df[“EVENT_DESCRIPTION”] as df[“CarID1”]
8. Extract car ID intermediary and service ID by extracting digits from the r“cars (\d+)/(\d+)pattern in df[“ASSET_ID_RAW”] as df[“CarID2”] and df[“ServiceID”] respectively
9. Merge df[“CarID1”] and df[“CarID2”] as a single field as df[“CarID”], whilst removing null values in the intermediaries
10. Standardise null values in df[“CarID”] as np.nan
11. Drop redundant variables

Extract Asset Information

1. Duplicate df[“ASSET_ID_RAW] as a new column df[“AssetClass”]
2. Loop through [locNamesList] to remove location information in df[“AssetClass”]
3. Remove numerical characters in df[“AssetClass”]
4. Exception Handling of df[“AssetClass”]
    1. If df[“AssetClass”] contains “TRACTION”, impute as "TRACTION/TRACTION“
    2. If df[“AssetClass”] contains “TUNNEL” and “LIGHT”, impute as "TUNNEL/LIGHT “
5. Clean up string in df[“AssetClass”]
    1. Replace r“\A(_)” in df[“AssetClass”] with “”
    2. Replace r“(_)\Z” in df[“AssetClass”] with “”
    3. Replace r“_+” in df[“AssetClass”] with “/”
6. Extract Asset Sub-class from df[“AssetClass”] using the pattern r“/(\w+)$”
7. Extract cleaned Asset Class from df[“AssetClass”] using the pattern r“(\w+)/”
8. Delete redundant variables

Extract Meta Data Info

1. Parse the variables “f_meta1” and “f_meta2” using parseMetaData() to extract the relevant time values
2. If there is an error in the time values, assign the default value of [“01/01/00 00:00:00.000”, “01/01/00 00:00:00.000”, 0]
3. Delete redundant variables
4. Assigned the time values as such based on the parsed meta data:
    1. df[“DATETIME_SENT”] = metaData[0]
    2. df[“DATETIME_RECEIVED”] = metaData[1]
    3. df[“TIME_CODE”] = metaData[2]
5. Delete redundant variables

Clean Up and Format Data

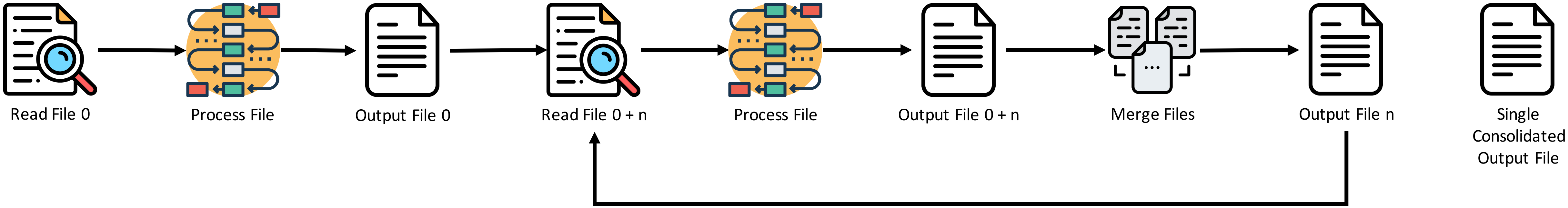
1. If df[“OPERATOR_INITIALS”] contains “(”, impute df[“OPERATOR_INITIALS”] as np.nan
2. Standardised all null values as np.nan for every column / field
3. Convert fields with binary values as a Boolean True / False. Target fields:
    1. df[“ACKNOWLEDGEMENT_REQUIRED”]
    2. df[“HIDDEN”]
4. Convert the datatype of all fields from strings to a suitable datatype appropriate to the field
5. Convert all fields with Unix time values to human readable time, making a +8hr time adjustment to align the time to local Singapore time (GMT+8) from GMT+0 (default)
6. Drop redundant columns
    1. df[“AUTO_ID”]
    2. df[“SHELVE”]
    3. df[“ACKNOWLEDGEMENT_AUTOPOINTER”]
    4. df[“USER2”]
    5. df[“DSS_EVENT_TYPE”]

Parallel Processing

This is for fast batch processing a vast quantity of files in one go

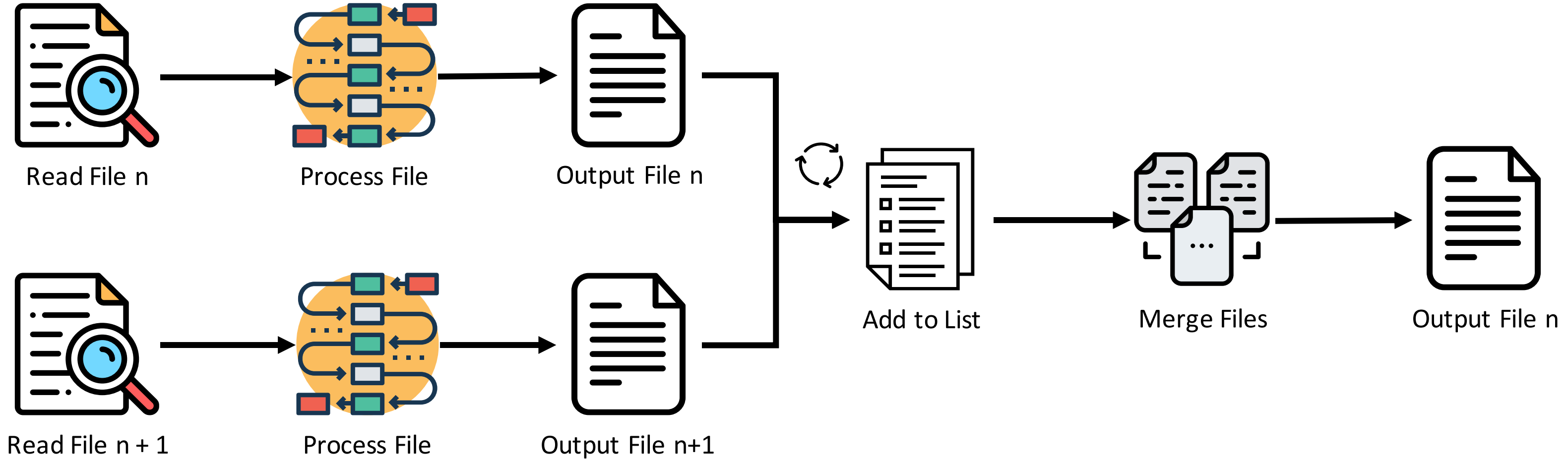
PreProcessing Using Parallel Processing

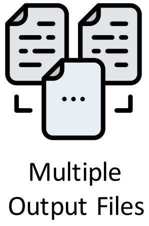
Linear Process (1 Core | Single Thread; Loop Process for each file in Folder)



Vectorised Parallel Process (15 Cores | 15 parallel threads; Loop Process for Each Subfolder)

The cleaningScript_vector() function defined previously is reused each time an event log file is processed in both linear and parallel processing methods





PreProcessing speed Optimisation

- Key enhancements
  - Further vectorise functions in file cleaning script, for example
    - Avoid for for/while loops where possible
    - Avoid regex where possible
    - Avoid lists in favour of sets and dictionaries where possible
  - Drop the creation of unnecessary columns
  - Reduction in unnecessary data joins
  - Vectorise cleaned dataframes into a single list for a single file merge rather than multiple incremental merger of files
  - Use of parallel processing techniques to maximise hardware resources for the processing and merger of files
  - Batching of files into subfolders (of under 50K files each) for processing to avoid overloading RAM

- Results (Test)
  - Parameters:
    - Source: ATS Server
    - 1K Log Files
    - 9220 rows | 33 cols output
  - Baseline: ~93s
  - Vectorised Cleaning: ~87s (↓6%)
  - Vectorised Cleaning + File Merge: ~74s (↓20%)
  - Vectorised Cleaning + File Merge + Parallel Process: ~18s (↓80%)
    - Est. Mean Time 0.018s per file
    - Est. Mean Speed: 55.556 files/s

Pre-Processing Speed Optimisation Test

- Input:
  - 356K+ ATS Alarm Event Log Files
- Original Runtime:
  - ~76.5hrs
- Optimised Script
  - ~1.6hrs (5760s)
  - ~47.8x speed increase
  - ~97.9% reduction in time taken
  - Est. Mean Time: 0.0162s per file
  - Est. Mean Speed: 61.806 files/s

Preprocessing Output

Sample Output

```
No. of Files Loaded Successfully: 52055

No. of Files Loading Failed: 0


<class 'pandas.core.frame.DataFrame'>

RangeIndex: 186796 entries, 0 to 186795

Data columns (total 34 columns):

 #   Column                    Non-Null Count   Dtype

---  ------                    --------------   -----

 0   ENTRY_CODE_SUFFIX         186796 non-null  string

 1   ENTRY_CODE                186796 non-null  string

 2   ALARM_ID                  186796 non-null  object

 3   USER_ID                   186796 non-null  object

 4   EQUIPMENT_NAME            186796 non-null  string

 5   VALUE                     186796 non-null  object

 6   VALUE_STATE               186796 non-null  object

 7   ACKNOWLEDGEMENT_REQUIRED  186796 non-null  object

 8   SEVERITY                  186796 non-null  object

 9   HIDDEN                    186796 non-null  category

 10  THEME                     186796 non-null  object

 11  EQUIPMENT_DATE            186796 non-null  datetime64[ns]

 12  ACQUISITION_DATE          186796 non-null  datetime64[ns]

 13  SCS_TIME                  186796 non-null  datetime64[ns]

 14  FUNCTIONAL_CATEGORY       186796 non-null  object

 15  GEOGRAPHICAL_CATEGORY     186796 non-null  object

 16  ENVIRONMENT               186796 non-null  string

 17  USER1                     186796 non-null  string

 18  ASSET_ID_RAW              186796 non-null  object

 19  ASSET_DESCRIPTION         186796 non-null  object

 20  EVENT_DESCRIPTION         186796 non-null  object

 21  EVENT_STATUS              186796 non-null  object

 22  OPERATOR_INITIALS         186796 non-null  object

 23  ASSET_DESC_CAT            186760 non-null  object

 24  EVENT_DESC_CAT            186796 non-null  object

 25  TrainID                   186796 non-null  object

 26  CarID                     186796 non-null  object

 27  ServiceID                 186796 non-null  object

 28  AssetClass                186796 non-null  object

 29  AssetSubClass             186796 non-null  object

 30  LOG_TYPE                  186796 non-null  string

 31  DATETIME_SENT             186796 non-null  datetime64[ns]

 32  DATETIME_RECEIVED         186796 non-null  datetime64[ns]

 33  TIME_CODE                 186796 non-null  datetime64[ns]

dtypes: category(1), datetime64[ns](6), object(21), string(6)
```

Sample Output

SN	Index	0
1	ENTRY_CODE_SUFFIX	+
2	ENTRY_CODE	-1139445989
3	ALARM_ID	14503
4	USER_ID	0
5	EQUIPMENT_NAME	
6	VALUE	5
7	VALUE_STATE	0
8	ACKNOWLEDGEMENT_REQUIRED	1
9	SEVERITY	2
10	HIDDEN	0
11	THEME	0
12	EQUIPMENT_DATE	27/10/2020 2:26:05 265
13	ACQUISITION_DATE	27/10/2020 2:26:05 265
14	SCS_TIME	27/10/2020 2:26:05 265
15	FUNCTIONAL_CATEGORY	60
16	GEOGRAPHICAL_CATEGORY	1
17	ENVIRONMENT	OCCCMS

SN	Index	0
18	USER1	1
19	ASSET_ID_RAW	OCC_LENV_CMS_
20	ASSET_DESCRIPTION	CMS SCS Server - Environment OCCCMS
21	EVENT_DESCRIPTION	Environment 1 Status
22	EVENT_STATUS	HOT
23	OPERATOR_INITIALS	None
24	ASSET_DESC_CAT	CMS SCS Server - Environment CMS
25	EVENT_DESC_CAT	Environment Status
26	TrainID	<NA>
27	CarID	<NA>
28	ServiceID	<NA>
29	AssetClass	LENV
30	AssetSubClass	CMS
31	LOG_TYPE	AE
32	DATETIME_SENT	27/10/2020 2:26:05 265
33	DATETIME_RECEIVED	27/10/2020 2:26:05 265
34	TIME_CODE	27/10/2020 2:26:05 265

Alarm List & Event List fields

FIELD	DESCRIPTION	FIELD	DESCRIPTION
AlarmId	number used for alarm identification	**Theme**	unused in display or archiving
AutoId	unused in display or archiving	**EquipmentDate**	date & time of last equipment state change
UserID	unused in display or archiving	**AcquisitionDate**	date & time of last data acquisition
EquipmentName	equipment address in DbmServer	**SCSTime**	SCADAsoft date & time of last action on alarm
Value	the alarm value or index, “N” or “A” if external	**FunctionalCategory**	function/system number of the alarm
ValueState	0 if event, positive value if alarm?	**GeographicalCategory**	location number of the alarm
AcknowledgeRequired	acknowledgement status	**AckAutomatonPointer**	unused in display or archiving
Severity	alarm severity level	**Environment**	ISCS server environment of the alarm (there are different ISCS server environments)
Shelve	unused in display or archiving	**User1**	used for MMS flag
Hidden	used for AVL flag	**User2**	unused in display or archiving
Message	alarm text with asset name, status, and operator	**DssEventType**	unused / blank

Note: Same fields are used in **AlarmListTest** and **EventListTest** logs.

HISEVENT Log fields

FIELD	VALUE SOURCE		FIELD	VALUE SOURCE
ALARMID	value of **AlarmId** field		EQUIPMENTNAME	value of **EquipmentName** field
ENVIRONMENT	value of **Environment** field		ASSETNAME	extracted from **Message** field
VALUE	value of **ValueState** field		MESSAGE	extracted from **Message** field
ACKREQUIRED	value of **AcknowledgeRequired** field		STATUS	extracted from **Message** field
SEVERITY	value of **Severity** field		GROUP1	unknown field
EQUIPMENTCLASS	deprecated		GROUP2	unknown field
FUNCTIONALCAT	value of **FunctionalCategory** field		FORMAT	deprecated
GEOGRAPHICALCAT	value of **GeographicalCategory** field		DSSEVENTTYPE	value of **DssEventType** field
DATEANDTIME	formatted value of **SCSTime** field		OPER	extracted from **Message** field

Functional Category

VALUE	FUNCTION		VALUE	FUNCTION		VALUE	FUNCTION
1	Depot OCS (Traction)		12	Train		30	Station Equipment (PSS)
2	Power System (1500 VDC)		55	Train Wash Plant		31	Drainage
3	Emergency Trip System		40	CCTV System		32	Fire Protection
4	Power System (22 kV)		42	Public Address System		33	Lift/Escalator/Travelator
5	ITESS		44	Passenger Info System		34	Shutter
6	Station Lighting		46	Radio System		35	Station Fire Summary
7	Power System (400 VAC)		48	Telephone System		36	Access Mgmt System
8	Tunnel Lighting		17	Station Environmental Ctrl		51	ISCS Eqpt & Intf Devices
10	Track Side Equipment		18	Smoke Extraction System		60	Handover Alarm
11	Platform Equipment		19	Tunnel Ventilation System		…	others*

**other values/functions not shown are not in GWS alarm/event filter dialog (values 100…195)*

Geographical Category

VALUE	LOCATION	VALUE	LOCATION	VALUE	LOCATION
1	OCC Operation Control Centre	9	LTI Little India station	17	BGK Buangkok station
2	SOCC* Standby OCC	10	FRP Farrer Park station	18	SKG Sengkang station
3	NED NEL Depot	11	BNK Boon Keng station	19	PGL Punggol station
4	HBF Harbour Front station	12	PTP Potong Pasir station	27	RS Rolling Stock/Trains
5	OTP Outram Park station	13	WLH Woodleigh station	30	MOV* Movable asset
6	CNT Chinatown station	14	SER Serangoon station	99	DEFAULT* Default
7	CQY Clarke Quay station	15	KVN Kovan station		
8	DBG Dhoby Ghaut station	16	HGN Hougang station		

**not in GWS alarm/event filter dialog*

Baseline Alarm Tagging

Overview

- Key scripts
  - Python method
    - Baseline Alarm Tagging v1.x.ipynb
  - SQL Server Method
    - Update Alarm Tag - Prod Workflow .sql
      - Requires files to be exported to SQL Server instead of CSV
    - Nuisance Event Tag - Prod Workflow.sql
      - Triggers after alarm tagging
    - SQL Agent
      - Trigger the above scripts every 1 min

Alarm Tagging

1. Load preprocessed AlarmList File(s) of period X as df_al
2. Create a new column df_ae[“isAlarm”], with a default value of “True”
3. Drop all redundant columns in df_ae, except for the following using for performing a lookup later and the alarm tag
    1. df_ae[“ALARM_ID”]
    2. df_ae[“SCS_TIME”]
    3. df_ae[“ENVIRONMENT”] – not required if files are already partitioned by server
    4. df_ae[“ASSET_ID_RAW”] – EQUIPMENT_NAME may be used as an alternative as well
    5. df_ae[“EVENT_DESCRIPTION”]
    6. df_ae[“EVENT_STATUS”]
    7. df_ae[“isAlarm”]
4. Load the preprocessed EventList File(s) of period X as df_ae
5. Create a new dataframe, df, by performing a left merge lookup on df_ae to df_ae
6. Set alarm status value as False in df[“isAlarm”] if the value is null
7. Format fields containing time information as a datetime datatype
8. Delete redundant variables e.g. “LOG_TYPE”

Nuisance Alarm Suppression

1. Due to the large number of incoming alarms, operators needs to discern which alarms are nuisance alarms or risk alarm fatigue
2. Tagging of nuisance alarms allows one to
    1. Suppress nuisance alarms
    2. Identify problematic assets generating excessive alarms

Nuisance Alarm Suppression

1. Nuisance alarm (1 min Sliding Window)
    1. Pattern code: df[“EVENT_DESCRIPTION”] + “ | ” + df[“EVENT_STATUS”]
    2. Pattern comparison grouping: group by df[“ENVIRONMENT”] and df[“ASSET_ID_RAW”]
    3. Nuisance alarm = Repeat alarm events (AA Pattern) OR Toggle alarm events (2x; ABAB Pattern)
    4. Detection method
        1. Generate a new column based on the pattern code
        2. Index the data based on time
        3. Group the pattern codes based on the defined pattern comparison grouping
        4. Extract the latest 6 values of the group for each event entry (these are intermediary values used for comparison)
        5. Repeat alarm events are identified by comparing the latest 2 values if they are identical and have occurred within a rolling / sliding 60s window, and are hence tagged as such
        6. Toggle alarm events are identified by comparing if the latest 2 values are the same as the latest 3rd and 4th values respectively; and they had occurred within a rolling / sliding 60s window, and are hence tagged as such
        7. Delete intermediary values
        8. Nuisance alarm tagging is then based on whether an event is registered

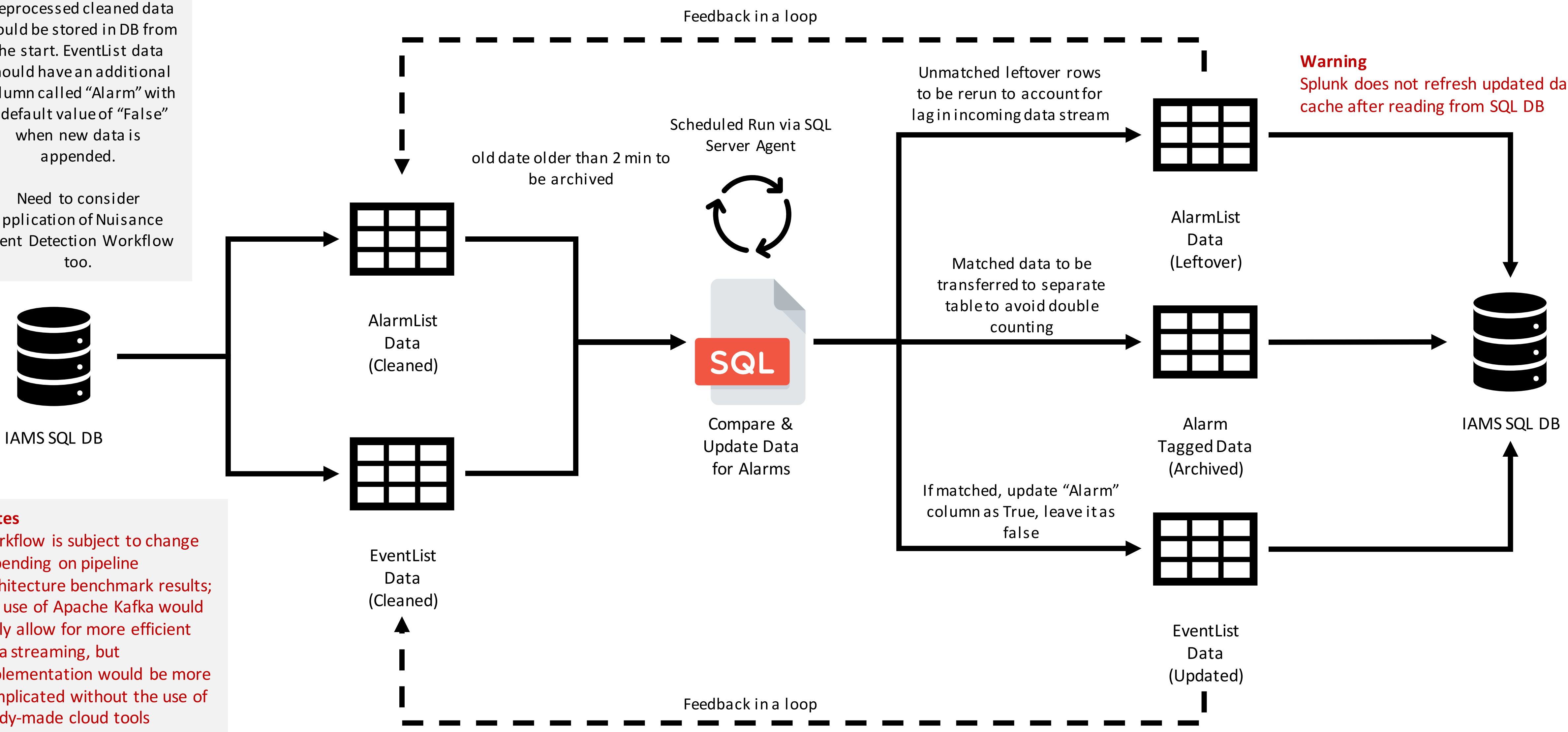
Nuisance Alarm Tagging

AlarmID	eventValue_N-0	eventValue_N-1	eventValue_N-2	eventValue_N-3	eventValue_N-4	eventValue_N-5	SCS_TIME-0	SCS_TIME-1	SCS_TIME-2	SCS_TIME-3	SCS_TIME-4	SCS_TIME-5	Repeat	Toggle x2	Toggle x3
1	A	B					T	T+30					F	F	F
2	A	A					T	T+30					T	F	F
3	A	A					T	T+65s					F	F	F
4	A	C	A				T	T+5s	T+6s	T+7s			F	F	F
5	A	B					T	T+5s	T+6s				F	F	F
6	A	B	A				T	T+5s	T+6s	T+7s			F	F	F
7	A	B	A	B			T	T+5s	T+6s	T+7s	T+8s		F	T	F
8	A	B	A	B			T	T+5s	T+6s	T+7s	T+80s		F	F	F
9	A	B	A	B	A	B	T	T+5s	T+6s	T+7s	T+8s	T+9s	F	T	T
10	A	B	A	B	A	B	T	T+5s	T+6s	T+7s	T+8s	T+80s	F	T	F

Step 1: Alarm Data Tagging with SQL Workflow

Preprocessed cleaned data should be stored in DB from the start. EventList data should have an additional column called "Alarm" with a default value of "False" when new data is appended.

Need to consider application of Nuisance Event Detection Workflow too.



Feedback in a loop

IAMS SQL DB

AlarmList Data (Cleaned)

EventList Data (Cleaned)

old date older than 2 min to be archived

Scheduled Run via SQL Server Agent

Compare & Update Data for Alarms

Unmatched leftover rows to be rerun to account for lag in incoming data stream

Matched data to be transferred to separate table to avoid double counting

If matched, update "Alarm" column as True, leave it as false

AlarmList Data (Leftover)

Alarm Tagged Data (Archived)

EventList Data (Updated)

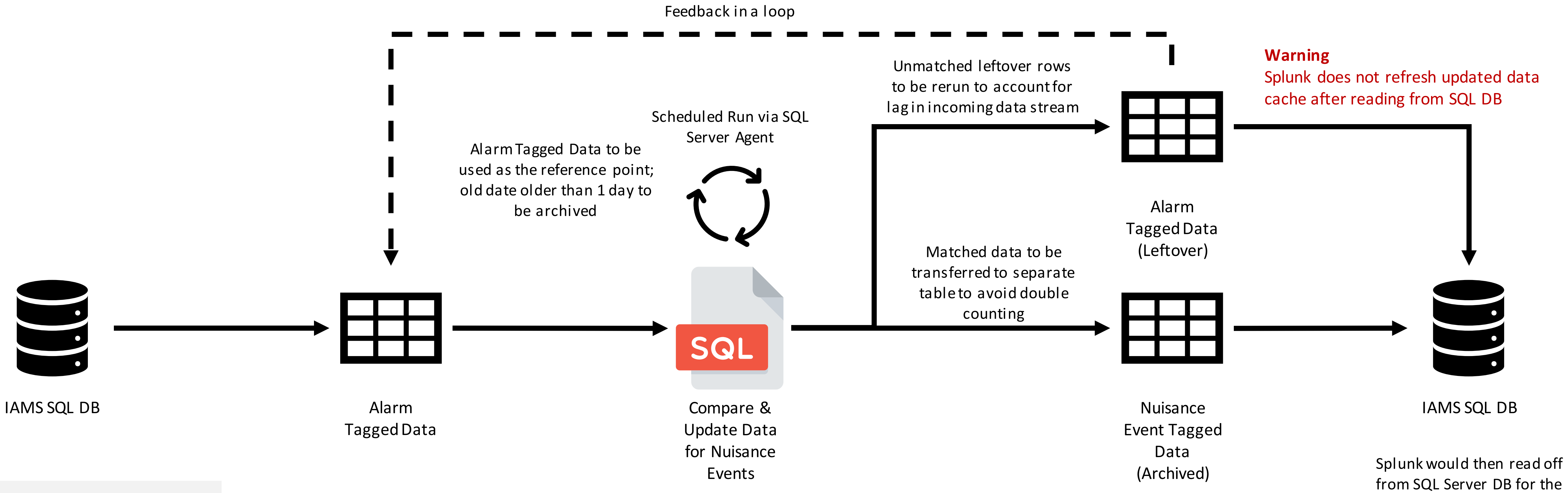
IAMS SQL DB

Feedback in a loop

Warning
Splunk does not refresh updated data cache after reading from SQL DB

Notes
Workflow is subject to change depending on pipeline architecture benchmark results; the use of Apache Kafka would likely allow for more efficient data streaming, but implementation would be more complicated without the use of ready-made cloud tools

Step 2: Nuisance event Data Tagging with SQL Workflow



Feedback in a loop

Alarm Tagged Data to be used as the reference point; old date older than 1 day to be archived

Scheduled Run via SQL Server Agent

Unmatched leftover rows to be rerun to account for lag in incoming data stream

Matched data to be transferred to separate table to avoid double counting

Warning
Splunk does not refresh updated data cache after reading from SQL DB

IAMS SQL DB

Alarm Tagged Data

Compare & Update Data for Nuisance Events

Alarm Tagged Data (Leftover)

Nuisance Event Tagged Data (Archived)

IAMS SQL DB

Splunk would then read off from SQL Server DB for the Nuisance Event Tagged Data

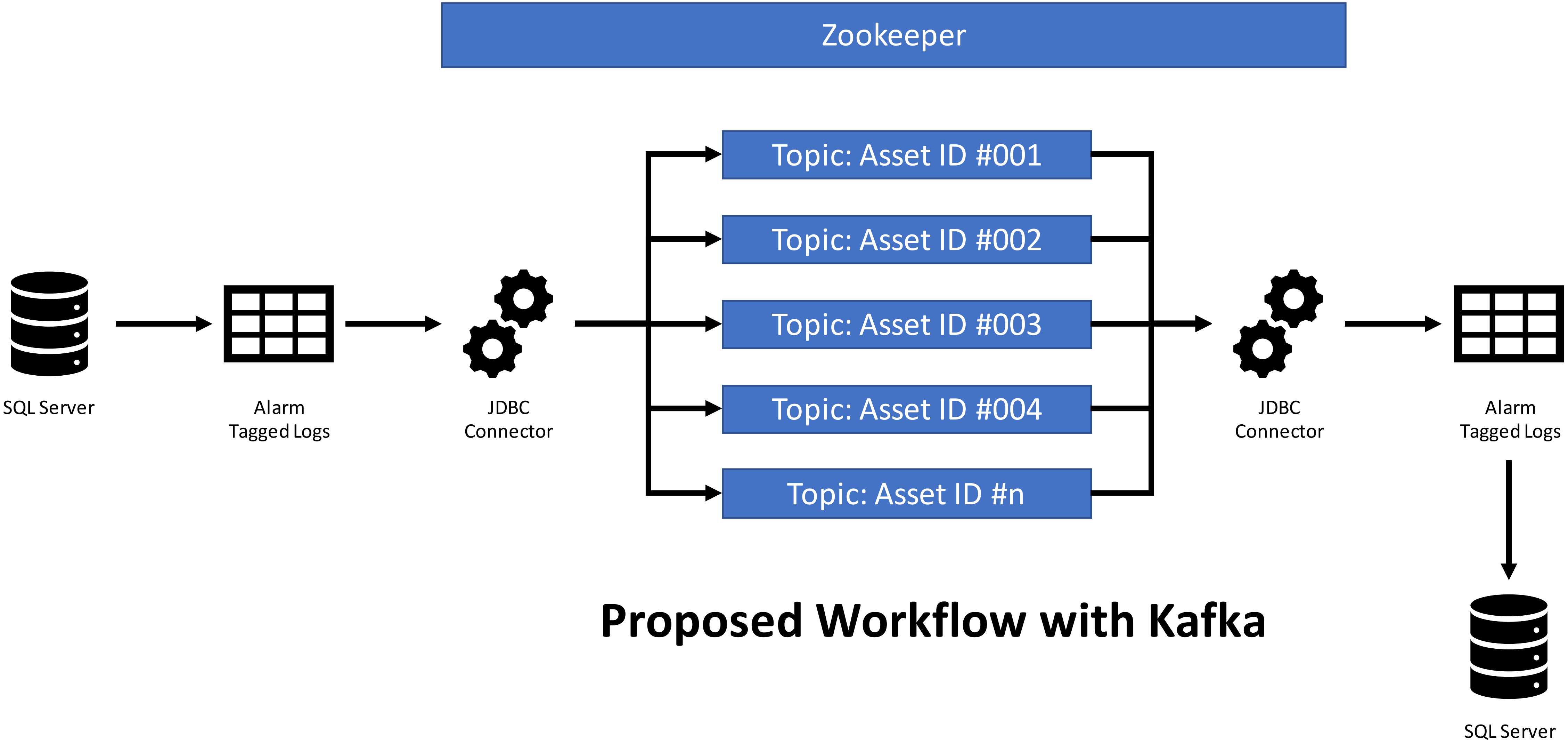
Notes
Workflow is subject to change depending on pipeline architecture benchmark results; the use of Apache Kafka would likely allow for more efficient data streaming, but implementation would be more complicated without the use of ready-made cloud tools

General Workflow

- SQL Agent to trigger at 1 min intervals based on system time
- Windowing of data to acquire subset of data at each stage
    - This is to keep compute loads manageable
    - Periods:
        - N-0 (Latest Time of event loaded into SQL Table) to N-1min: Buffer period for files to load
        - N-1min to N-2min: target period for analytics applications before archival
        - Exceeding N-2min: catch all for archival
    - Time pegged to the time which the event is recorded into SQL Server Table
        - One can only work with the data available at the instant of time; and there is a limit of which how long one can wait for near real time performance
    - Windowing performed for each individual Alarm Tagging and Nuisance Alarm Tagging run independently
- Alarm Tagging
    - Data matched from eventList and alarmList based on matching event attributes and common SCS_Time
    - SCS_Time would be used as the reference time for Alarm Tagging after the initial windowing
- Nuisance Alarm Tagging
    - Data from Alarm Tagged list matched against lag values based on matching event attributes, asset ID and SCS_Time
    - SCS_Time would be used as the reference time for Alarm Tagging after the initial windowing

Comments

- SQL may not be the most efficient approach to handling streaming data due to the amount of read/write cycles which can be more computationally expensive compared to Big Data approaches like Apache Kafka.
- However, all things considered, the volume of incoming logs may not be very high (~120 files per min), allowing one to potentially get away with using SQL even if it may be slower
- On the other hand, despite being more complex to implement, Apache Kafka could allow for better real time streaming of data.
- But it should also be noted that during peak hours, the saving of logs into ISCS's Alarm Server can be 10-20min lag from the time the event occurred



Proposed Workflow with Kafka

Final Output Sample

SN	Index	0
1	ENTRY_CODE_SUFFIX	+
2	ENTRY_CODE	-1139445989
3	ALARM_ID	14503
4	USER_ID	0
5	EQUIPMENT_NAME	
6	VALUE	5
7	VALUE_STATE	0
8	ACKNOWLEDGEMENT_REQUIRED	1
9	SEVERITY	2
10	HIDDEN	0
11	THEME	0
12	EQUIPMENT_DATE	27/10/2020 2:26:05 265
13	ACQUISITION_DATE	27/10/2020 2:26:05 265
14	SCS_TIME	27/10/2020 2:26:05 265
15	FUNCTIONAL_CATEGORY	60
16	GEOGRAPHICAL_CATEGORY	1
17	ENVIRONMENT	OCCCMS
18	USER1	1
19	ASSET_ID_RAW	OCC_LENV_CMS_
20	ASSET_DESCRIPTION	CMS SCS Server - Environment OCCCMS
21	EVENT_DESCRIPTION	Environment 1 Status
22	EVENT_STATUS	HOT
23	OPERATOR_INITIALS	None
24	ASSET_DESC_CAT	CMS SCS Server - Environment CMS
25	EVENT_DESC_CAT	Environment Status
26	TrainID	<NA>
27	CarID	<NA>
28	ServiceID	<NA>
29	AssetClass	LENV
30	AssetSubClass	CMS
31	DATETIME_SENT	27/10/2020 2:26:05 265
32	DATETIME_RECEIVED	27/10/2020 2:26:05 265
33	TIME_CODE	27/10/2020 2:26:05 265
34	isAlarm	True
35	NuisanceAlarm	True
36	RepeatAlarm	True
37	AltAlarm2	True
38	AltAlarm3	True
39		
40		