

IBM Surveillance Insight for Financial
Services
Version 2.0.2

*IBM Surveillance Insight for Financial
Services Solution Guide*



Note

Before using this information and the product it supports, read the information in [“Notices” on page 117](#).

Product Information

This document applies to Version 2.0.2 and may also apply to subsequent releases.

Copyright

Licensed Materials - Property of IBM

© Copyright IBM Corp. 2017.

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

IBM, the IBM logo and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "[Copyright and trademark information](http://www.ibm.com/legal/copytrade.shtml)" at www.ibm.com/legal/copytrade.shtml.

© **Copyright International Business Machines Corporation 2016, 2017.**

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Introduction.....	V
Chapter 1. IBM Surveillance Insight for Financial Services.....	1
The solution architecture.....	2
Deploy the IBM Surveillance Insight for Financial Services software.....	3
Chapter 2. Surveillance Insight Workbench.....	5
Dashboard page.....	5
Alert Details page.....	6
Employee Details page.....	8
Notes page.....	9
Search pages.....	12
Voice evidence page.....	14
Chapter 3. Trade surveillance.....	15
Trade Surveillance Toolkit.....	15
Ticker price schema.....	20
Execution schema.....	20
Order schema.....	22
Quote schema.....	24
Trade schema.....	25
End of day (EOD) schema.....	26
Market reference schema.....	27
Transaction schema.....	27
Risk event schema.....	27
Trade evidence schema.....	27
Event schema.....	27
Event data schema.....	28
Pump-and-dump use case.....	28
Spoofing detection use case.....	29
Off-market use case.....	31
Front running use case.....	32
Extending Trade Surveillance.....	34
Chapter 4. E-Comm surveillance.....	37
E-Comm data ingestion.....	38
E-Comm feature extraction.....	39
Communication schema.....	40
E-Comm risk scoring.....	41
E-Comm Spark job configuration.....	43
End-to-end flow for e-comm processing.....	46
Chapter 5. Voice surveillance.....	49
Voice Ingestion service.....	49
Voice data services.....	51
Voice Surveillance Toolkit metadata schema.....	54
WAV adaptor processing.....	55
PCAP format processing.....	56
Chapter 6. Surveillance Insight data schemas.....	59

Party view.....	60
Communication view.....	61
Alert view.....	64
Trade view.....	66
Chapter 7. NLP libraries.....	69
Emotion Detection library.....	69
Concept Mapper library.....	71
Classifier library.....	74
Chapter 8. Inference engine.....	79
Inference engine risk model.....	79
Run the inference engine.....	80
Chapter 9. Indexing and searching.....	85
Chapter 10. Conduct Surveillance.....	87
Raw data schema and ingestion.....	87
Analysis pipeline.....	87
Create an analysis pipeline.....	88
Trend analysis.....	91
Complaints dashboard.....	99
Complaints data model.....	102
Complaint features.....	102
Solr data model for complaints.....	103
Chapter 11. Health Check User Interface.....	105
Health Check tabs.....	105
Date ranges.....	105
Health Check dashboards.....	106
Ecomm dashboards.....	106
Voice dashboards.....	108
Chapter 12. Troubleshooting.....	113
CDISI5060E No default Java found.....	113
CDISI3059W You may be running a firewall which may prevent communication between the cluster hosts.....	113
CDISI5070E The perl-XML-Simple software dependency is not installed.....	113
Appendix A. Accessibility features.....	115
Notices.....	117
Index.....	119

Introduction

Use IBM® Surveillance Insight® for Financial Services to proactively detect, profile, and prioritize non-compliant behavior in financial organizations. The solution ingests unstructured and structured data, such as trade, electronic communication, and voice data, to flag risky behavior. Surveillance Insights helps you investigate sophisticated misconduct faster by prioritizing alerts and reducing false positives, and reduces the cost of misconduct.

Some of the key problems that financial firms face in terms of compliance misconduct include:

- Fraudsters using sophisticated techniques thereby making it hard to detect misconduct.
- Monitoring and profiling are hard to do proactively and efficiently with constantly changing regulatory compliance norms.
- A high rate of false positives increases the operational costs of alert management and investigations.
- Siloed solutions make fraud identification difficult and delayed.

IBM Surveillance Insight for Financial Services addresses these problems by:

- Leveraging key innovative technologies, such as behavior analysis and machine learning, to proactively identify abnormalities and potential misconduct without pre-defined rules.
- Using evidence-based reasoning that aids streamlined investigations.
- Using risk-based alerting that reduces false positives and negatives and improves the efficiency of investigations.
- Combining structured and unstructured data from different siloed systems into a single platform to perform analytics.

IBM Surveillance Insight for Financial Services takes a holistic approach to risk detection and reporting. It combines structured data such as stock market data (trade data) with unstructured data such as electronic emails and voice data, and it uses this data to perform behavior analysis and anomaly detection by using machine learning and natural language processing.

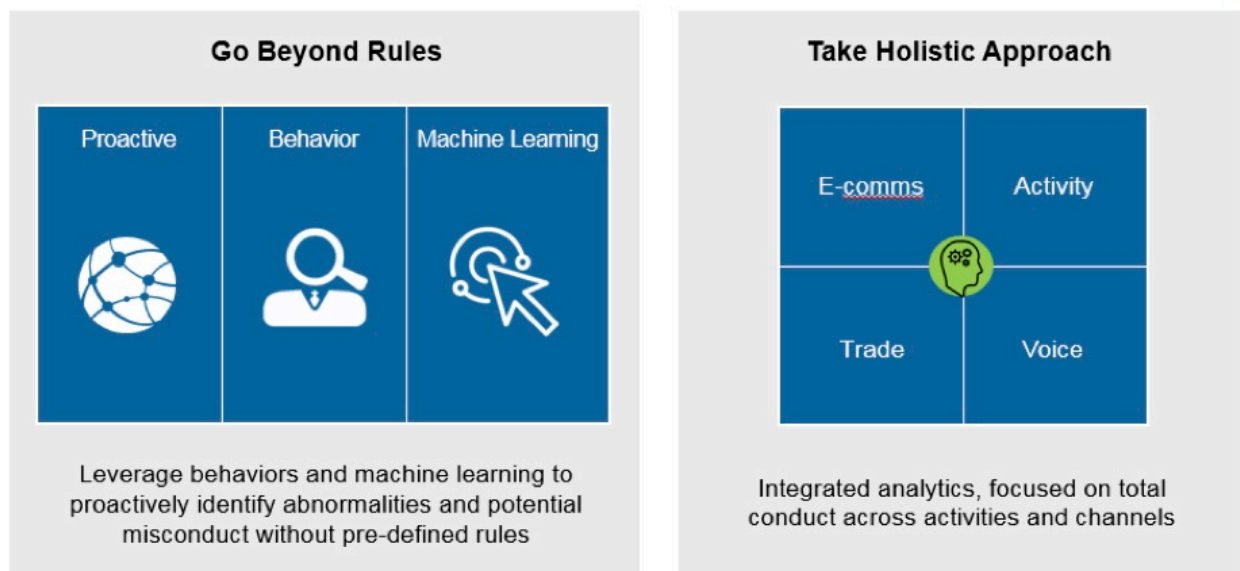


Figure 1: Surveillance Insight overview

Audience

This guide is intended for administrators and users of the IBM Surveillance Insight for Financial Services solution. It provides information on installation and configuration of the solution, and information about using the solution.

Finding information and getting help

To find product documentation on the web, access [IBM Knowledge Center](http://www.ibm.com/support/knowledgecenter/SSWTQQ) (www.ibm.com/support/knowledgecenter/SSWTQQ).

Accessibility features

Accessibility features help users who have a physical disability, such as restricted mobility or limited vision, to use information technology products. Some of the components included in the IBM Surveillance Insight for Financial Services have accessibility features. For more information, see [Appendix A, “Accessibility features,”](#) on page 115.

The HTML documentation has accessibility features. PDF documents are supplemental and, as such, include no added accessibility features.

Forward-looking statements

This documentation describes the current functionality of the product. References to items that are not currently available may be included. No implication of any future availability should be inferred. Any such references are not a commitment, promise, or legal obligation to deliver any material, code, or functionality. The development, release, and timing of features or functionality remain at the sole discretion of IBM.

Samples disclaimer

Sample files may contain fictional data manually or machine generated, factual data that is compiled from academic or public sources, or data that is used with permission of the copyright holder, for use as sample data to develop sample applications. Product names that are referenced may be the trademarks of their respective owners. Unauthorized duplication is prohibited.

Chapter 1. IBM Surveillance Insight for Financial Services

IBM Surveillance Insight for Financial Services provides you with the capabilities to meet regulatory obligations by proactively monitoring vast volumes of data for incriminating evidence of rogue trading or other wrong-doing through a cognitive and holistic solution for monitoring all trading-related activities. The solution improves current surveillance process results and delivers greater efficiency and accuracy to bring the power of cognitive analysis to the financial services industry.

The following diagram shows the high-level IBM Surveillance Insight for Financial Services process.



Figure 2: High-level process

1. As a first step in the process, data from electronic communications (such as email and chat), voice data, and structured stock market data are ingested into IBM Surveillance Insight for Financial Services for analysis.
2. The data is analyzed.
3. The results of the analysis are risk indicators with specific scores.
4. The evidences and their scores are used by the inference engine to generate a consolidated score. This score indicates whether an alert needs to be created for the current set of risk evidences. If needed, an alert is generated and associated with the related parties and stock market tickers.
5. The alerts and the related evidences that are collected as part of the analysis can be viewed in the IBM Surveillance Insight for Financial Services Workbench.

After the alerts are created and the evidences are collected, the remaining steps in the process are completed outside of IBM Surveillance Insight for Financial Services. For example, case investigators must work on the alerts and confirm or reject them, and then investigation reports must be sent out to the regulatory bodies as is required by compliance norms.

The solution architecture

IBM Surveillance Insight for Financial Services is a layered architecture is made up of several components.

The following diagram shows the different layers that make up the product:

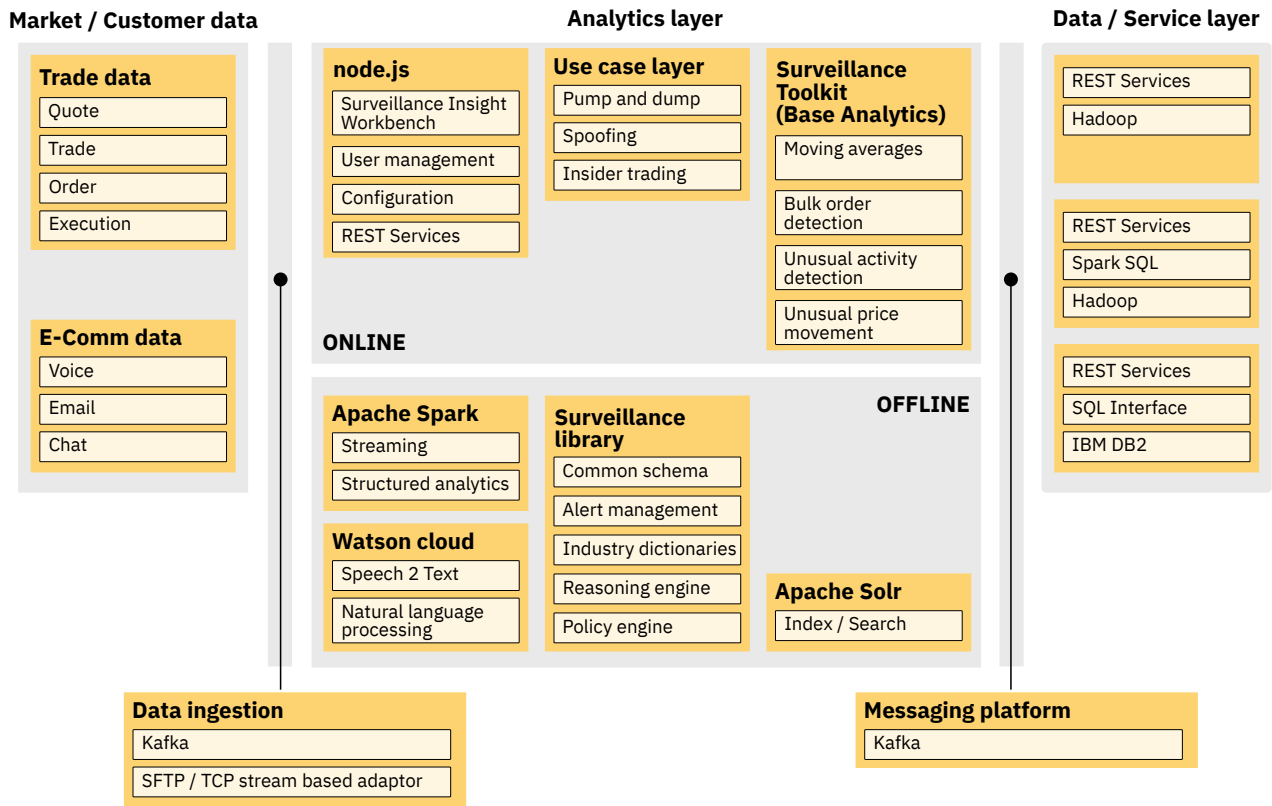


Figure 3: Product layers

- The data layer shows the various types of structured and unstructured data that is consumed by the product.
- The data ingestion layer contains the FTP/TCP-based adaptor that is used to load data into Hadoop. The Kafka messaging system is used for loading e-communications into the system.

Note: IBM Surveillance Insight for Financial Services does not provide the adaptors with the product.

- The analytics layer contains the following components:
 - The Workbench components and the supporting REST services for the user interfaces.
 - Specific use case implementations that leverage the base toolkit operators.
 - The surveillance library that contains the common components that provide core platform capabilities such as alert management, reasoning, and the policy engine.
 - The Spark Streaming API is used by Spark jobs as part of the use case implementations.
 - Speech 2 Text and the NLP APIs are used in voice surveillance and eComms surveillance.
 - Solr is used to index content to enable search capabilities in the Workbench.
- Kafka is used as an integration component in the use case implementations and to enable asynchronous communication between the Streams jobs and the Spark jobs.
- The data layer primarily consists of data in Hadoop and IBM DB2®. The day-to-day market data is stored in Hadoop. It is accessed by using the spark-sql or spark-graphx APIs. Data in DB2 is accessed by using traditional relational SQL. REST Services are provided for data that needs to be accessed by the user interfaces and for certain operations such as alert management.

- The output, or the risk evidences from the use case implementations (trade, e-comm, and voice), are dropped into the Kafka messaging topics for the use case-specific Spark jobs. The Spark jobs perform the post processing after the evidences are received from the Streams jobs.

Deploy the IBM Surveillance Insight for Financial Services software

IBM Surveillance Insight for Financial Services is deployed on different node computers that host different parts of the solution. Some prerequisite components are required on each of the nodes.

The following diagram provides a high-level overview of the solution architecture.

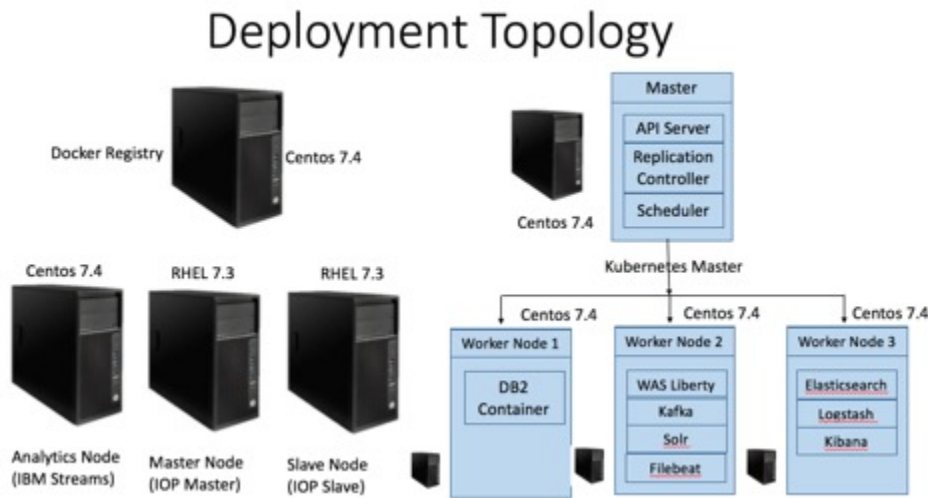


Figure 4: Deployment topology

There is a separate installer for each of the components that comprise IBM Surveillance Insight for Financial Services.

- IBM Surveillance Insight for Financial Services
- IBM Trade Surveillance Analytics
- IBM Electronic Communication Surveillance Analytics
- IBM Voice Surveillance Analytics
- IBM Complaints Analytics

The IBM Surveillance Insight for Financial Services base component also requires the following parts:

- IBM Surveillance Insight for Fin Serv DB2AWSE (1 of 8) 2.0.2 CentOS EN - CNPY7EN
- IBM Surveillance Insight for Fin Serv Liberty (2 of 8) 2.0.2 CentOS EN - CNPY8EN
- IBM Surveillance Insight for Fin Serv Kafka (3 of 8) 2.0.2 CentOS EN - CNPY9EN
- IBM Surveillance Insight for Fin Serv Solr (4 of 8) 2.0.2 CentOS EN - CNPZ0EN
- IBM Surveillance Insight for Fin Serv Kibana (5 of 8) 2.0.2 CentOS EN - CNPZ1EN
- IBM Surveillance Insight for Fin Serv Logstash (6 of 8) 2.0.2 CentOS EN - CNPZ2EN
- IBM Surveillance Insight for Fin Serv Elasticsearch (7 of 8) 2.0.2 CentOS EN - CNPZ3EN
- IBM Surveillance Insight for Fin Serv Filebeat (8 of 8) 2.0.2 CentOS EN - CNPZ4EN

Chapter 2. Surveillance Insight Workbench

Users access the product through the Surveillance Insight Workbench, a web-based interface that provides users with the results of the analysis that is performed by the solution.

You access the Surveillance Insight Workbench by entering the following URL in your web browser:

`https://hostname:port/surveillance/dashboard/index.html`

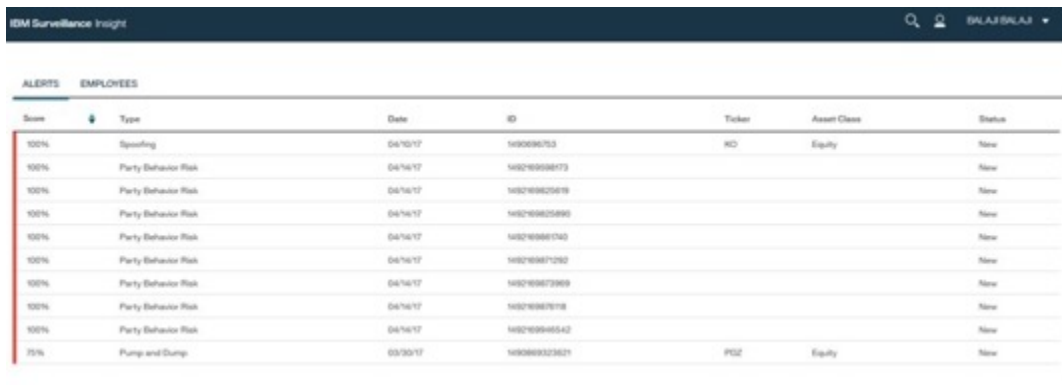
You must enter your log in credentials.

Dashboard page

The Dashboard page shows the Alerts and Employees tabs.

Alerts tab

The **Alert** tab shows open alerts that were created in the past 30 days from the date that the last alert was created. The results are sorted by risk score.

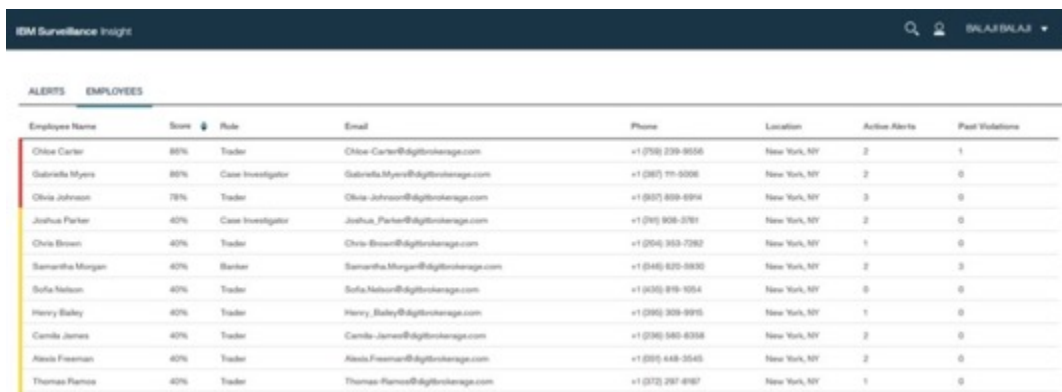


Score	Type	Date	ID	Ticker	Asset Class	Status
100%	Spooling	04/10/17	14926986703	RO	Equity	New
100%	Party Behavior Risk	04/16/17	149269938173			New
100%	Party Behavior Risk	04/16/17	149269925819			New
100%	Party Behavior Risk	04/16/17	149269925890			New
100%	Party Behavior Risk	04/16/17	149269981740			New
100%	Party Behavior Risk	04/16/17	1492699871282			New
100%	Party Behavior Risk	04/16/17	1492699873989			New
100%	Party Behavior Risk	04/16/17	1492699878118			New
100%	Party Behavior Risk	04/16/17	1492699846542			New
75%	Pump and Dump	03/30/17	1493869323821	PGZ	Equity	New

Figure 5: Alert tab

Employees tab

The **Employees** tab displays the top 50 employees sorted by their risk score. Only employees with a positive risk score value are displayed. The risk score of an employee is based on their past and currently active alerts. If an employee does not have any alerts in the past 90 days and does not have any currently active alerts, they will not appear in this list.



Employee Name	Score	Role	Email	Phone	Location	Active Alerts	Past Violations
Olivia Carter	80%	Trader	Olivia.Carter@dgfbrokerage.com	+1 (718) 239-8806	New York, NY	2	1
Sabrina Myers	80%	Case Investigator	Sabrina.Myers@dgfbrokerage.com	+1 (202) 711-9096	New York, NY	2	0
Olivia Johnson	78%	Trader	Olivia.Johnson@dgfbrokerage.com	+1 (807) 808-6994	New York, NY	3	0
Joshua Parker	40%	Case Investigator	Joshua.Parker@dgfbrokerage.com	+1 (745) 908-5701	New York, NY	2	0
Chris Brown	40%	Trader	Chris.Brown@dgfbrokerage.com	+1 (204) 352-7282	New York, NY	1	0
Samantha Morgan	40%	Banker	Samantha.Morgan@dgfbrokerage.com	+1 (545) 625-8830	New York, NY	2	3
Sofia Nelson	40%	Trader	Sofia.Nelson@dgfbrokerage.com	+1 (435) 819-1054	New York, NY	0	0
Henry Bailey	40%	Trader	Henry.Bailey@dgfbrokerage.com	+1 (202) 309-8995	New York, NY	1	0
Camille James	40%	Trader	Camille.James@dgfbrokerage.com	+1 (212) 585-6358	New York, NY	2	0
Alexis Freeman	40%	Trader	Alexis.Freeman@dgfbrokerage.com	+1 (888) 448-3545	New York, NY	2	0
Thomas Flores	40%	Trader	Thomas.Flores@dgfbrokerage.com	+1 (212) 297-8987	New York, NY	1	0

Figure 6: Employees tab

Alert Details page

The Alert Details page shows the basic information about the alert in the header region of the page, and then more information on the tabs of the page.

Overview tab

The **Overview** tab shows the reasoning graph and the associated evidences that created the alert. You can change the start and end dates of the reasoning graph can be changed to show the change in the reasoning over time.

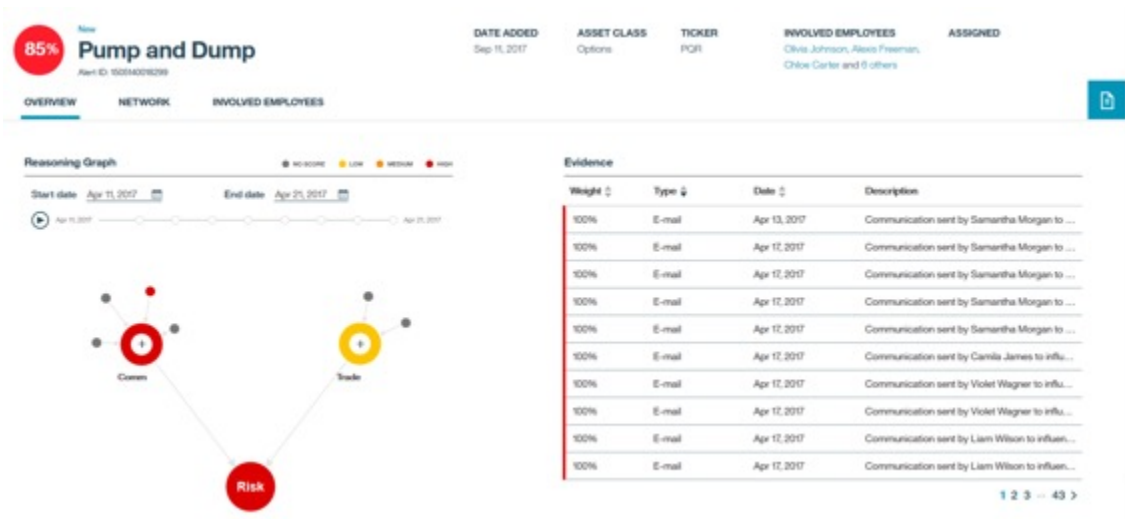


Figure 7: Alert Overview tab

Alert Network tab

The **Network** tab shows the network of communications that were analyzed from the electronic communications. The nodes on the network chart are entities such as a person, an organization, or a ticker. The links between the nodes represent a communication between the entities. You can click the link to show the email that were proof of the communication.

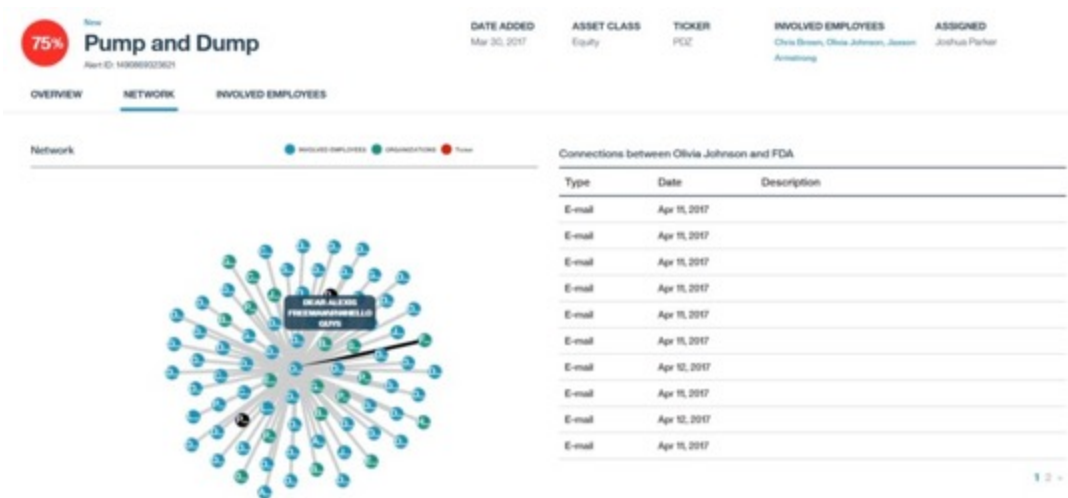


Figure 8: Alert Network tab

Involved Employees tab

The **Involved Employees** tab displays a list of employees that are involved in an alert. You can click an employee to display information about that employee, such as personal information, history, anomalies, and emotion analysis.

The personal information shows the location details, contact information, supervisor details, an alert summary, and a risk score for the employee. The risk score is the overall risk score that is associated with the employee based on their current and past alerts. It is not the risk score that is associated with a specific alert.

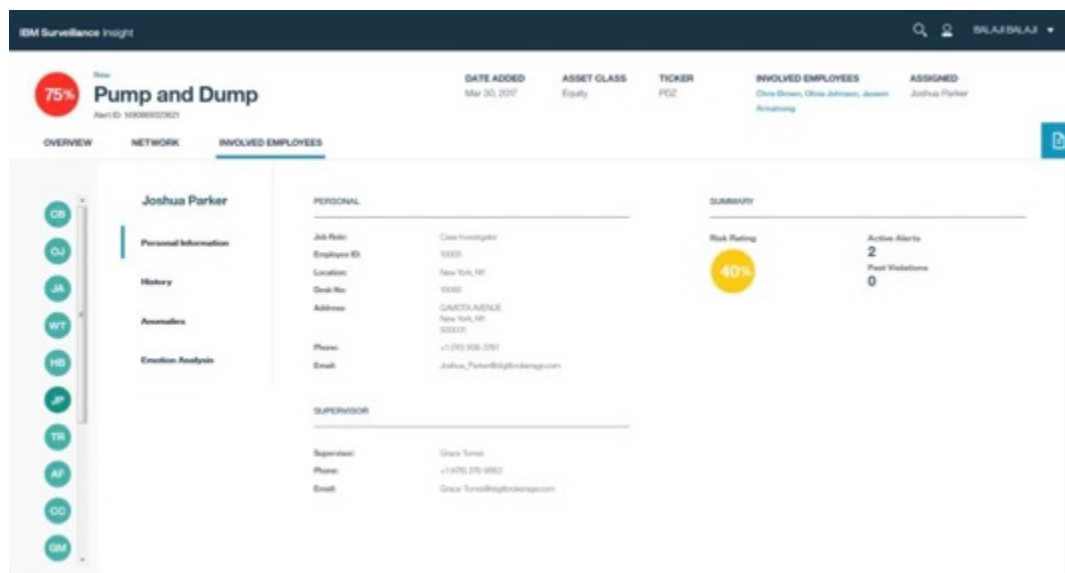


Figure 9: Personal Information

The **History** tab shows the history of current and past alerts for the employee.

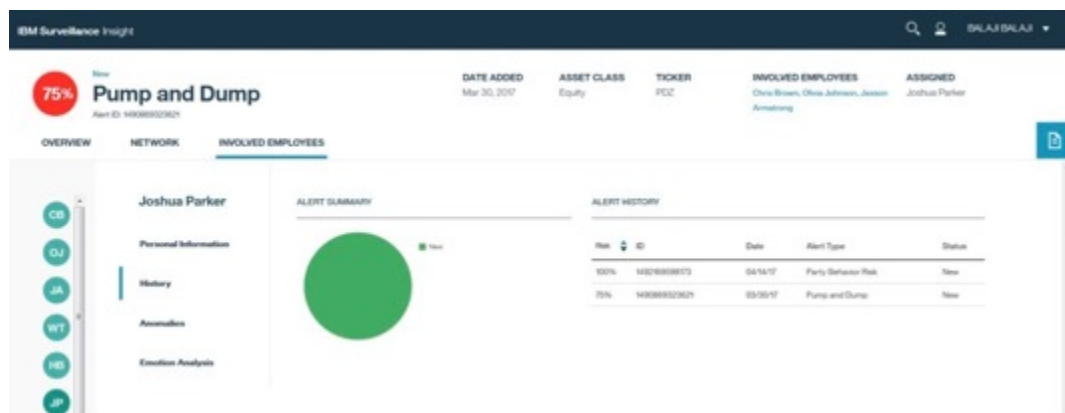


Figure 10: History

The **Anomalies** tab shows the behavior anomalies of the selected employee. Each anomaly has a risk score and a date that is associated with it. These factors determine the position of the circle in the chart. The color of the chart represents the type of anomaly. The data that is used to plot the chart is determined by the start and end dates of the alert.

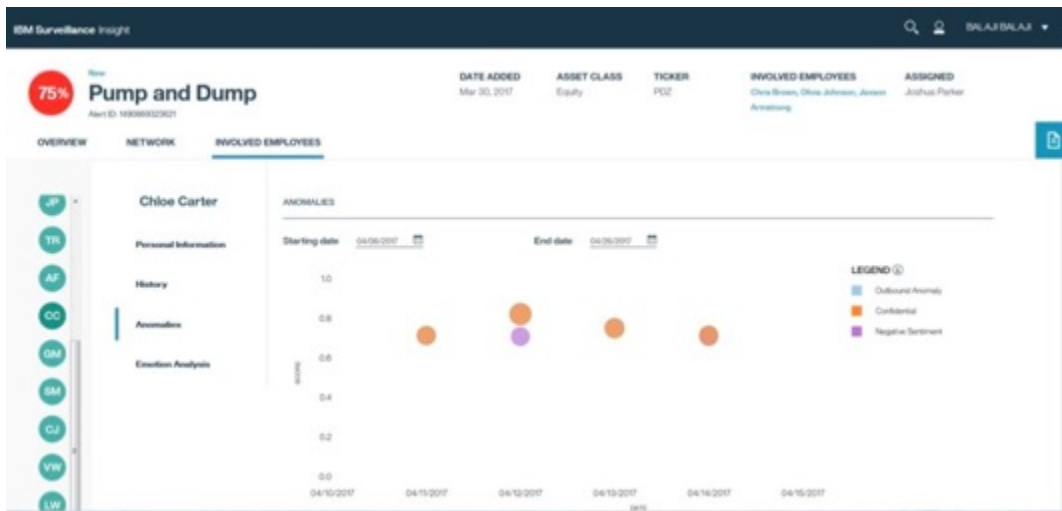


Figure 11: Anomalies tab

The **Emotion Analysis** tab shows the emotional behavior that is portrayed by an employee based on their communications. The chart displays a circle for each instance where the employee's emotion score crosses a threshold. You can click the circle to display a list of communication evidences that contain that specific emotion. The data that is used to plot the chart is determined by the start and end dates of the alert.



Figure 12: Emotion Analysis tab

Employee Details page

The **Employee Details** page shows the same information as the Involved Employees section of the **Alert** page.

The only difference is that the anomalies chart and the emotional analysis chart use the last 10 days of available data in the database. Whereas, the start and end dates of the alert are used in the **Alert** page.

For more information about the content, see the [“Involved Employees tab”](#) on page 7.

Notes page

Alert investigators can add notes and attach files to an alert from the **Notes** page. You can view the **Notes** page from any of the pages in the **Alert Details** view.

View notes

Click the note icon to view the notes for an alert.

ASSET CLASS

Equity

TICKER

EuroUSD

INVOLVED EMPLOYEES

Olivia Johnson

ASSIGNED

Evidence

Risk

Figure 13: View notes

TICKER

KO

INVOLVED EMPLOYEES

Olivia Johnson

ASSIGNED

Joshua Parker

Notes

Notes

Screenshot

Screenshot-2017-04-20

Additional notes added.

Sadie Ross Apr 20, 2017 at 15:51 pm

2017-04-07

Go to Note Summary

Evidence

Weight	Type	Date
99%	Trade	Apr 10, 2017
99%	Trade	Apr 10, 2017
99%	Trade	Apr 10, 2017
99%	Trade	Apr 10, 2017
99%	Trade	Apr 10, 2017
99%	Trade	Apr 10, 2017
99%	Trade	Apr 10, 2017
99%	Trade	Apr 10, 2017
99%	Trade	Apr 10, 2017

Figure 14: Displaying notes

Create notes

From [Figure 14 on page 10](#) page, click **Notes** to add a note. You can also click **Screenshot** to add a screen capture of the current screen.

TICKER

KO

INVOLVED EMPLOYEES

Olivia Johnson

ASSIGNED

Joshua Parker

Notes

Notes

Screenshot

Cancel

Add

Screenshot-2017-04-20

Go to Note Summary

Evidence

Weight	Type	Date
99%	Trade	Apr 10, 2017
99%	Trade	Apr 10, 2017
99%	Trade	Apr 10, 2017
99%	Trade	Apr 10, 2017
99%	Trade	Apr 10, 2017
99%	Trade	Apr 10, 2017
99%	Trade	Apr 10, 2017
99%	Trade	Apr 10, 2017
99%	Trade	Apr 10, 2017
99%	Trade	Apr 10, 2017

Figure 15: Create notes

Update notes

You can click **Edit** to change or update a note.

Delete notes

To delete a note, click **Go to Note Summary**, and delete the note.

Notes summaries

You can save a notes summary to a PDF file. To generate a summary, click **Go to Note Summary**, and click **Generate PDF**.

Note actions

The alert investigator can agree or disagree with the notes on the **Note Summary** page. This updates the status of the note in the system.

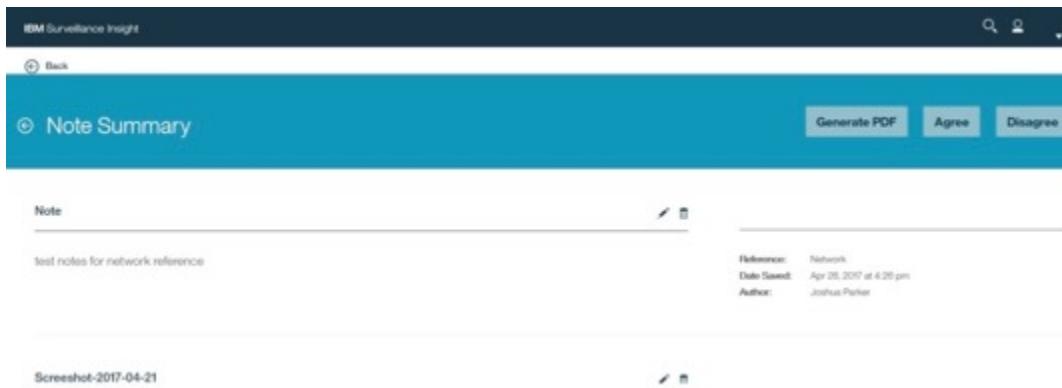


Figure 16: Note Summary page

Search pages

You can search for alerts, employees, and communication types.

Alert Search

You can search for an alert by different criteria, such as by date, alert type, employee, ticker, and status. After you select your criteria, click **Apply** to display the results.

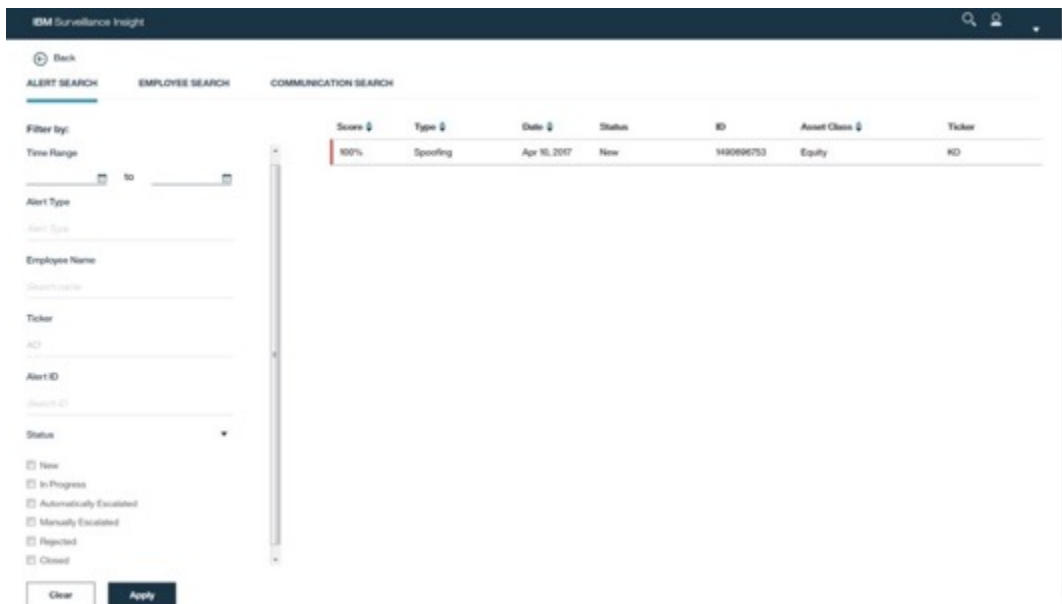


Figure 17: Alert Search page

Employee Search

You can search for an employee by their name, ID, location, or role.

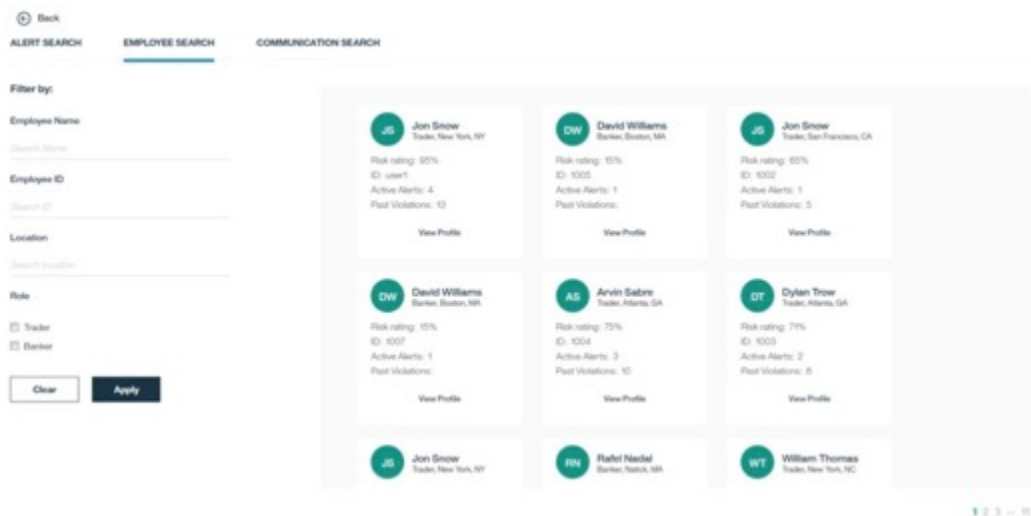


Figure 18: Employee Search page

Communication Search

The **Communication Search** allows you to search by communication type, by the people involved in the communication, by the entities, and by the emotions detected in the communication.

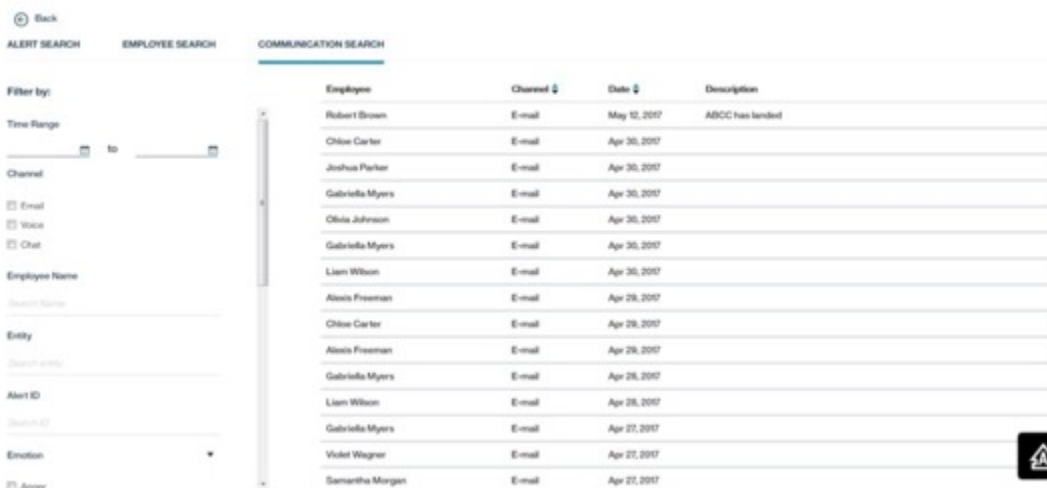


Figure 19: Communication Search page

Search capabilities

Select **Voice** as the channel value on the **Communication Search** page to display all of the existing voice communications. You can filter the voice communications by properties such as "Login Name", "Phone", "Extension", "Device Id".

You can also search the existing voice communications by using **Advance Query**. **Advance Query** allows you to use both free-form text searches or Solr query style searches. For example, a search that contains "ABCC" returns all voice communications that contain the text "ABCC".

Voice evidence page

The **Voice Evidence** page lets you view the metadata and transcript of the voice data.



Figure 20: Voice Evidence page

User can add annotations against selected text from the transcript, and they can add an existing Evidence Type to a communication.

Users should be part of the Compliance Officer group to be able to edit a communication.

1. Click the **Edit** button for a voice communication to make its transcript editable.
2. Select a section of text and click the annotation button. After you add the annotation, the text is highlighted in the color of the annotation.
3. Select an existing evidence, such as Anger, from the menu. You can also slide the risk probability bar to set a value between 0 and 1. Click **Select** to associate the evidence type to the communication. This enables users to manually create risk evidences.
4. To remove an existing annotation, select the text (the text should already be annotated) and click **Remove Annotation**.
5. Click **Finish Editing** after you have made all of your changes.

By default, there are 3 supported annotations: People, Organization, and Ticker. Users can update the ENTITY_TYPE_MASTER table in the database to add new entities and specify a color code. The newly added entities are also displayed as annotations, which allows users to use the new entities to annotate existing communications.

You can annotate only a single line of text at a time. You cannot select multiple lines and add an annotation.

Only one user can be editing a communication at any time. After you click **Finish Editing**, another user will be able to annotate the communication.

Bulk export for voice communications

Select one or more existing voice communications and click **Export**. You can export data by using one of 4 options: All, Audio, Transcript, or Metadata.

- All exports the voice metadata, transcript, and the audio file
- Audio exports the metadata and the audio file
- Transcript exports the metadata and the transcript
- Metadata exports the metadata

The data is exported in `tar.gz` format. The file is exported to the location that is identified in the JNDI variable, `VOICE_BULK_EXPORT_ARCHIVE_FILE_DESTINATION`, on the server.

Chapter 3. Trade surveillance

IBM Surveillance Insight for Financial Services trade surveillance offers mid and back-office surveillance on market activities and communication to detect and report possible market abuse activity.

The trade component monitors trade data, detects suspicious patterns against the predefined risk indicators, and reports the patterns. The trade data includes order data, trade data, quotes, executions, and end of the day summary data. Also included are the transactions and market reference data from the equity market.

The risk indicators are analyzed by the inference engine. The inference engine uses a risk model to determine whether an alert needs to be created.

Two use cases are provided:

- Pump-and-dump
- Spoofing
- Off-market (equity market)

The following trade-related risk indicators are available in the Surveillance Insight for Financial Services master data:

- Bulk orders
- High order-to-order cancel ratio
- Bulk executions
- Unusual quote price movement
- Pump in the stock
- Dump in the stock
- Deal rate anomaly (equity market)
- Party past alerts

Data ingestion

Market data, such as trade, order, quote, and execution data, are uploaded to the Hadoop file system (HDFS) by using the HDFS terminal. The naming conventions for the files and folder are as follows:

- /user/sifsuser/trade/Trade_<yyyy-mm-dd>.csv
- /user/sifsuser/order/Order_<yyyy-mm-dd>.csv
- /user/sifsuser/execution/Execution_<yyyy-mm-dd>.csv
- /user/sifsuser/quote/Quote_<yyyy-mm-dd>.csv
- /user/sifsuser/EOD/EOD_<yyyy-mm-dd>.csv
- /user/sifsuser/transactions/transactions_<yyyy-mm-dd>.csv
- /user/sifsuser/marketReference/marketReference_<yyyy-mm-dd>.csv

The current implementation of the trade use cases expects that there is one file of each type for each day.

The IBM InfoSphere® Streams data loader job monitors the folders. The job reads any new file that is dropped into the folder and sends it for downstream processing.

Trade Surveillance Toolkit

The Trade Surveillance Toolkit helps the solution developers to focus on specific use case development.

The toolkit contains basic data types, commonly used functional operators relevant to trade analytics, and adapters for some data sources.

The Surveillance Base Toolkit includes the following risk indicator operators:

- Bulk orders detection
- High order-to-order cancel ratio
- Bulk execution detection
- Unusual quote price movement
- Deal rate anomaly (equity market)

The risk evidence sink operator is also included.

The Surveillance Base Toolkit includes the following schema type definitions:

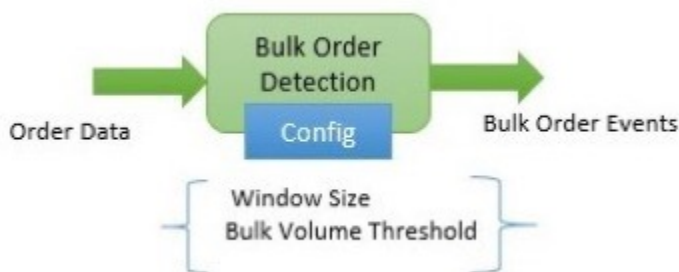
- Order
- Quote
- Execution
- Trade
- Transaction
- Market reference
- Risk event
- Trade evidence

For information about the schemas for the types that are defined in the toolkit, see [Trade Toolkit data schemas](#).

Note:

- The risk event and trade evidence schemas are new in this release. All new risk indicator implementations must use these types to create events. The deal rate anomaly risk indicator provides an example of how to use these types. The other risk indicator implementations use the event and event data types, which are deprecated. It is not recommended to use the event and event data types.
- The risk evidence sink operator makes it easier to create risk evidences for downstream consumption. It uses the risk event type events as input. Users of available risk indicators (other than deal rate anomaly), must convert the event type event to the risk event type event before you can push the event to the risk evidence sink operator.

Bulk Order Detection operator



Purpose

Looks at a sliding window of orders and checks if total order volume is over the Bulk Volume Threshold. It is grouped by trader, ticker, and order side (buy/sell). The sliding window moves by 1 second for every slide.

Input

Order Data according to the schema.

Output event contents

Id: unique ID for this event

Event Time: time in input data, not system time

Event Type: BULK_ORDER

Trader ID: ID of the trader who is placing the order

Ticker

Event Data

orderQty: total volume of orders in the window for Trader ID

Side: BUY or SELL

maxOrderPrice: maximum order price that was seen in the current window

Configuration

Window Size: time in seconds for collecting data to analyze

Bulk Volume Threshold: Volume threshold that is used to trigger events

High order cancellation operator



Purpose

Looks at a sliding window of window size (in seconds) and checks if total order volume to order cancellation volume for a trader is above the cancellation threshold. It is grouped by trader, ticker, and order side (buy/sell).

Input

Order Data according to the schema.

Output event contents

Id: unique ID for this event

Event Time: time in input data, not system time

Event Type: HIGH_CANCEL_RATIO

Trader ID: ID of the trader who is placing the order

Ticker

Event Data

Side: BUY or SELL

Ratio: order volume versus cancellation ratio

Configuration

Window Size: time in seconds for collecting data to analyze

Window Slide: Slide value for the window in seconds

Cancellation Threshold: Volume threshold that is used to trigger events

Price Trend operator



Purpose

Looks at a sliding window of quotes and computes the rise or drop trend (slope) for offer and bid prices. It fires an event if the price slope rises above the Rise Threshold or drops below the Drop Threshold. The former indicates an unusual rise in the quotes and the latter indicates an unusual drop in the quotes. The analysis is grouped by ticker.

Input

Quote Data according to the schema.

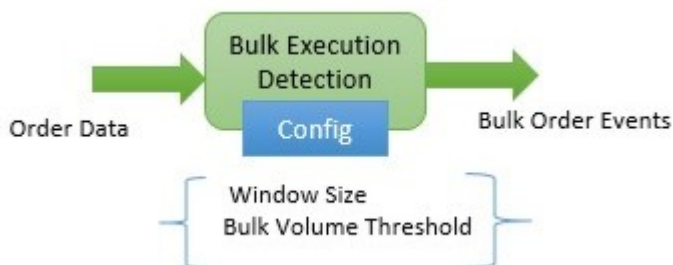
Output event contents

Id: unique ID for this event
Event Time: time in input data, not system time
Event Type: PRICE_TREND
Trader ID: not applicable
Ticker
Event Data
Side: BID or OFFER
Slope: slope of the bid or offer price

Configuration

Window Size: time in seconds for collecting data to analyze
Window Slide: Slide value for the window in seconds
Drop Threshold: Threshold that indicates an unusual downward trend in the quotes
Rise Threshold: Threshold that indicates an unusual rise trend in the quotes

Bulk Execution Detection operator



Purpose

Looks at a sliding window of executions and checks if the total executed volume is above the Bulk Volume Threshold. It is grouped by trader, ticker, and order side (buy/sell). The sliding window moves by 1 second for every slide.

Input

Execution Data according to the schema.

Output event contents

Id: unique ID for this event

Event Time: time in input data, not system time

Event Type: BULK_EXEC

Trader ID: ID of the trader who is placing the order

Ticker

Event Data

orderQty: total volume of executions in the window for Trader ID

Side: BUY or SELL

TotalExecValue: price * execution quantity for this window. It is grouped by ticker, trader, and side

Configuration

Window Size: time in seconds for collecting data to analyze

Bulk Volume Threshold: The volume threshold that is used to trigger events

Deal Rate Anomaly operator



Purpose

Looks at forex transaction data from the equity market and matches the deal rate against the high and low market values as mentioned in the market reference data at the time of the transaction. If the deal rate falls outside of the high and low range, a deal rate anomaly event is fired.

Input

Transaction data and market reference data.

Output event contents

Note: This operator outputs riskEvent type events. For more information, see [“Risk event schema” on page 27](#).

Id: unique ID for this event

Event Time: time in input data, not system time

Event Type: Transaction Deal Rate Anomaly

Trader ID: ID of the trader who is placing the transaction

Trade evidence data:

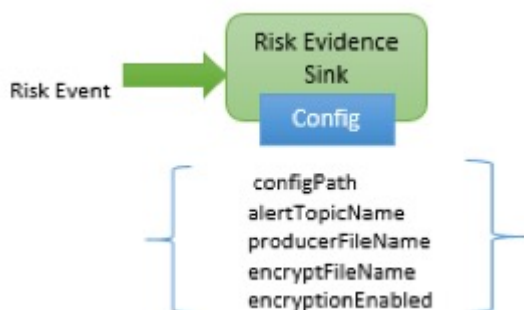
- dataType: transaction
- startTime: ""
- windowSize: 0.0
- id: transaction ID

Configuration

Start time: The time in the input data that corresponds to the first record in the input. This is used to compute the one-minute window durations to read the market reference data.

Risk Indicator Look Up Code: This is the look-up code for Deal Rate Anomaly in the master data in the SIFS database. This value is used to send out the risk indicator to the downstream processes for persisting the risk evidence with the corresponding look-up code.

Risk Evidence Sink operator



Purpose

This operator abstracts the job of creating a risk evidence message, encrypting it, and the passing it on to a Kafka topic for the downstream processes to consume. It takes risk event type as the input and converts it into a risk evidence JSON. It then, optionally, encrypts the JSON and drops it into a pre-configured Kafka topic.

Input

Risk event.

Configuration

configPath: The file system path where the Kafka producer properties file can be found

alertTopicName: The topic name in Kafka where the operator drops the risk evidences

producerFileName: The name of the Kafka producer properties file

encryptFileName: The name of the encryption properties file

encryptionEnabled: Toggles encryption of the risk evidence JSON before it is dropped into the Kafka topic

Ticker price schema

symbol,datetime,price

Table 1: Ticker price schema		
Field name	Field type	Description
Symbol	String	The ticker corresponding to the trade
Datetime	String	The date and time at which the trade occurred
Price	Float	The unit price of the stocks traded

Execution schema

Id, Symbol, Datetime, Brokerid, Traderid, Clientid, effectiveTime, expireTime, timeInForce, exposureDuration, tradingSession, tradingSessionSub, settlType, settlDate, Currency, currencyFXRate, execType, trdType, matchType, Side, orderQty, Price, exchangeCode, refQuoteId, refOrderId

For more information about the fields in this schema, refer to the [FIX wiki](http://fixwiki.org/fixwiki/ExecutionReport/FIX.5.0SP2%2B) (<http://fixwiki.org/fixwiki/ExecutionReport/FIX.5.0SP2%2B>)

Table 2: Execution schema

Field name	Field type	Description
Id	String	Unique identifier for the execution
Symbol	String	The ticker corresponding to the trade
Datetime	String	The date and time at which the trade occurred. The format is yyyy-mm-dd hh:mm:ss
Brokerid	String	The ID of the broker that is involved in this execution
Traderid	String	The ID of the trader that is involved in this execution
Clientid	String	The ID of the client that is involved in this execution
effectiveTime	String	The date and time stamp at which the execution is effective
expireTime	String	The date and time stamp when this execution will expire
timeInForce	String	Specifies how long the order remains in effect. Absence of this field is interpreted as DAY
exposureDuration	String	The time in seconds of a "Good for Time" (GFT) TimeInForce
tradingSession	String	Identifier for a trading session
tradingSessionSub	String	Optional market assigned sub identifier for a trading phase within a trading session
settlType	String	Indicates order settlement period. If present, SettlDate overrides this field. If both SettlType and SettlDate are omitted, the default for SettlType is 0 (Regular)
settlDate	String	Specific date of trade settlement (SettlementDate) in YYYYMMDD format
Currency	String	The currency in which the execution price is represented
currencyFXRate	Float	The foreign exchange rate that is used to calculate SettlCurrAmt from Currencyto SettlCurrency

Table 2: Execution schema (continued)		
Field name	Field type	Description
execType	String	Describes the specific ExecutionRpt (for example, Pending Cancel) while OrdStatus will always identify the current order status (for example, Partially Filled)
trdType	String	Type of trade
matchType	String	The point in the matching process at which this trade was matched
Side	String	Denotes BUY or SELL execution
orderQty	Int	The volume that is fulfilled by this execution
Price	Float	The price per unit for this execution
exchangeCode	String	
refQuoteId	String	The quote that corresponds to this execution
refOrderId	String	Refers to the order corresponding to this execution

Order schema

Id, Symbol, Datetime, effectiveTime, expireTime, timeInForce, exposureDuration, settlType, settlDate, Currency, currencyFXRate, partyId, orderType, Side, orderQty, minQuantity, matchIncr, Price, manualOrderIndicator, refOrderId, refOrderSource

For more information about the fields in this schema, refer to the [FIX wiki](http://fixwiki.org/fixwiki/ExecutionReport/FIX.5.OSP2%2B) (<http://fixwiki.org/fixwiki/ExecutionReport/FIX.5.OSP2%2B>)

Table 3: Order schema		
Field name	Field type	Description
Id	String	Unique identifier for the order
Symbol	String	The ticker corresponding to the trade
Datetime	String	The date and time at which the order was placed. The format is yyyy-mm-dd hh:mm:ss
effectiveTime	String	The date and time stamp at which the order is effective
expireTime	String	The date and time stamp when this order will expire

Table 3: Order schema (continued)

Field name	Field type	Description
timeInForce	String	Specifies how long the order remains in effect. If this value is not provided, DAY is used as the default
exposureDuration	String	The time in seconds of a "Good for Time" (GFT) TimeInForce
settlType	String	Indicates order settlement period. If present, SettlDate overrides this field. If both SettlType and SettlDate are omitted, the default for SettlType is 0 (Regular)
settlDate	String	Specific date of trade settlement (SettlementDate) in YYYYMMDD format
Currency	String	The currency in which the order price is represented
currencyFXRate	Float	The exchange rate that is used to calculate the SettlCurrAmt from Currency to SettlCurrency
partyId	String	The trader that is involved in this order
orderType	String	CANCEL represents an order cancellation. Used with refOrderId.
Side	String	Indicates a BUY or SELL order
orderQty	Int	The order volume
minQuantity	Int	Minimum quantity of an order to be executed
matchIncr	Int	Allows orders to specify a minimum quantity that applies to every execution (one execution might be for multiple counter-orders). The order can still fill against smaller orders, but the cumulative quantity of the execution must be in multiples of the MatchIncrement
Price	Float	The price per unit for this order
manualOrderIndicator	boolean	Indicates whether the order was initially received manually (as opposed to electronically) or if it was entered manually (as opposed to it being entered by automated trading software)

Table 3: Order schema (continued)		
Field name	Field type	Description
refOrderId	String	Used with the orderType. Refers to the order that is being canceled
refOrderSource	String	The source of the order that is represented by a cancellation order

Quote schema

Id, Symbol, Datetime, expireTime, exposureDuration, tradingSession, tradingSessionSub, settlType, settlDate, Currency, currencyFXRate, partyId, commPercentage, commType, bidPrice, offerPrice, bidSize, minBidSize, totalBidSize, bidSpotRate, bidFwdPoints, offerSize, minOfferSize, totalOfferSize, offerSpotRate, offerFwdPoints

For more information about the fields in this schema, refer to the [FIX wiki](http://fixwiki.org/fixwiki/ExecutionReport/FIX.5.0SP2%2B) (<http://fixwiki.org/fixwiki/ExecutionReport/FIX.5.0SP2%2B>)

Table 4: Quote schema		
Field name	Field type	Description
Id	String	Unique identifier for the quote
Symbol	String	The ticker corresponding to the trade
Datetime	String	The date and time at which the quote was placed. The format is yyyy-mm-dd hh:mm:ss
expireTime	String	The date and time stamp when this quote will expire
exposureDuration	String	The time in seconds of a "Good for Time" (GFT) TimeInForce
tradingSession	String	Identifier for a trading session
tradingSessionSub	String	Optional market assigned sub identifier for a trading phase within a trading session
settlType	String	Indicates order settlement period. If present, SettlDate overrides this field. If both SettlType and SettlDate are omitted, the default for SettlType is 0 (Regular)
settlDate	String	Specific date of trade settlement (SettlementDate) in YYYYMMDD format
Currency	String	The currency in which the quote price is represented

Table 4: Quote schema (continued)		
Field name	Field type	Description
currencyFXRate	Float	The exchange rate that is used to calculate SettlCurrAmt from Currencyto SettlCurrency
partyId	String	The trader that is involved in this quote
commPercentage	Float	Percentage of commission
commType	String	Specifies the basis or unit that is used to calculate the total commission based on the rate
bidPrice	Float	Unit price of the bid
offerPrice	Float	Unit price of the offer
bidSize	Int	Quantity of bid
minBidSize	Int	Type of trade
totalBidSize	Int	
bidSpotRate	Float	Bid F/X spot rate
bidFwdPoints	Float	Bid F/X forward points added to spot rate. This can be a negative value
offerSize	Int	Quantity of the offer
minOfferSize	Int	Specifies the minimum offer size
totalOfferSize	Int	
offerSpotRate	Float	Offer F/X spot rate
offerFwdPoints	Float	Offer F/X forward points added to spot rate. This can be a negative value

Trade schema

Id, Symbol, Datetime, Brokerid, Traderid, Clientid, Price, Volume, Side

Table 5: Trade schema		
Field name	Field type	Description
Id	String	Unique identifier for the trade
Symbol	String	The ticker corresponding to the trade
Datetime	String	The date and time at which the trade occurred. The format is yyyy-mm-dd hh:mm:ss
Brokerid	String	The id of the broker involved in the trade
Traderid	String	The id of the trader involved in the trade

Table 5: Trade schema (continued)		
Field name	Field type	Description
Clientid	String	The id of the client involved in the trade
Price	Float	The unit price of the stocks traded
Volume	Int	The volume of stocks traded
Side	String	The BUY or SELL side of the trade

End of day (EOD) schema

Id, Symbol, Datetime, openingPrice, closingPrice, dayLowPrice, dayHighPrice, Week52LowPrice, Week52HighPrice, marketCap, totalVolume, industryCode, div, EPS, beta, description

Table 6: End of day (EOD) schema		
Field name	Field type	Description
Id	String	Unique identifier for the trade
Symbol	String	The ticker corresponding to the trade
Datetime	String	The date and time at which the trade occurred. The format is yyyy-mm-dd hh:mm:ss
openingPrice	Float	The opening price of the ticker for the date that is specified in the datetime field
closingPrice	Float	The closing price of the ticker for the date that is specified in the datetime field
dayLowPrice	Float	The lowest traded price for the day for this ticker
dayHighPrice	Float	The highest traded price for the day for this ticker
Week52LowPrice	Float	The 52-week low price for this ticker
Week52HighPrice	Float	The 52-week high price for this ticker
marketCap	Float	The market cap for this ticker
totalVolume	Int	The total outstanding volume for this ticker as of today
industryCode	String	The industry to which the organization that is represented by the ticker corresponds to
Div	Float	
EPS	Float	

Table 6: End of day (EOD) schema (continued)		
Field name	Field type	Description
Beta	Float	
Description	String	The description of the organization that is represented by the ticker

Market reference schema

symbol, periodDate, periodNumber, periodStart, periodEnd, open, high, low, close, mid

Transaction schema

transactionID, linkedOrderID, tradeDate, timeExecuted, valueDate, productType, dealerCode, portfolioCode, counterpartyCode, counterpartyName, counterpartyLocation, channel, broker, buyCCY, sellCCY, dealRate, buyCCYAmount, sellCCYAmount, transactionStatus, traderId, symbol

Risk event schema

rstring id, rstring description, rstring eventType, rstring startTime, float64 windowSize, rstring traderId, rstring symbol, float64 score, list<tradeEvidence> evidenceData ;

Trade evidence schema

rstring dataType, rstring startTime, float64 windowSize, list<rstring> id;

Event schema

id, eventType, startTime, windowSize, traderId, symbol, data

Table 7: Event schema		
Field name	Field type	Description
id	String	System generated id for the event
eventType	String	The type of the event
startTime	String	The system time when the event occurred
windowSize	Float	The size (in seconds) of the data window that the operator used while looking for events in the input data stream.
traderId	String	The trader id associated with the event
symbol	String	The symbol associated with the event
data	List of event data	Event specific data list. See Event Data schema

Event data schema

name, value

Table 8: Event data schema		
Field name	Field type	Description
name	String	The name of the event property
value	String	The value of the event property

Pump-and-dump use case

The solution contains a pump-and-dump use case, which carries out structured analysis of trade, order, and execution data and unstructured analysis of email data. The result is a daily score for the pump-and-dump indication.

The pump-and-dump score is distributed daily among the top five traders. Top five is determined based on the positions that are held by the traders.

Triggering the pump-and-dump rules

Ensure that the following folders exist on the Hadoop file system. The folders are:

- /user/sifsuser/trade/
- /user/sifsuser/order/
- /user/sifsuser/execution/
- /user/sifsuser/quote/
- /user/sifsuser/EOD/
- /user/sifsuser/sifsdata/ticker_summary/ticker_summary/
- /user/sifsuser/sifsdata/position_summary/
- /user/sifsuser/sifsdata/positions/
- /user/sifsuser/sifsdata/pump_dump/
- /user/sifsuser/sifsdata/trader_scores/

Both structured market data and unstructured email data are used for pump-and-dump detection. For accurate detection, ensure that you load the email data before you load the structured data. After structured data is pushed into Hadoop, the pump-and-dump implementation processes this data and automatically triggers the inference engine. The inference engine considers evidences from both email and structured data analysis to determine the risk score.

Understanding the pump-and-dump analysis results

When the data is loaded into Surveillance Insight for Financial Services, the pump-and-dump rules are triggered and the following files are created on the Hadoop file system:

- Date-wise trade summary data, including moving averages, is created in /user/sifsuser/sifsdata/ticker_summary/ticker_summary_<date>.csv
- Date-wise position summary data is created in /user/sifsuser/sifsdata/positions/top5Positions_<date>.csv and /user/sifsuser/sifsdata/position_summary/position_summary_<date>.csv
- Date-wise pump-and-dump score data is created in /user/sifsuser/sifsdata/pump_dump/pump_dump_<date>.csv
- Date-wise trader score data is created in /user/sifsuser/sifsdata/trader_scores/trader_scores_<date>.csv

The Spark job for pump-and-dump evidence collection is run for each date. This job collects all of the evidences for the day from Hadoop and populates the following tables in the SIFS database:

- Risk_Evidence
- Evidence_Ticker_Rel
- Evidence_Involved_Party_Rel

The Spark job also runs the inference engine, which applies a risk model and detects whether an alert needs to be generated for the evidence. Based on the result, either a new alert is generated, an existing alert is updated, or no action is taken. The alert information is populated to the following tables:

- Alert
- Alert_Ticker_Rel
- Alert_Involved_Party_Rel
- Alert_Risk_Indicator_Score
- Alert_Evidence_Rel

After the evidence and alert tables are updated, the pump-and-dump alert appears in the dashboard.

Pump-and-dump alerts are long running in that they can span several days to weeks or months. The same alert is updated daily if the risk score does not decay to 0.

The following rules explain when an alert is generated versus when an alert is updated:

1. If no evidence of pump-and-dump activity for a ticker from either structured or unstructured analysis exists, or if the risk score is too low, then no alerts are created.
2. If the inference engine determines that an alert must be created, then an alert is created in the Surveillance Insight database against the ticker. The top 5 traders for the day for that ticker are also associated with the alert.
3. After the alert is created, the alert is updated daily with the following information while the ticker remains in a pump or dump state:
 - New alert risk indicator scores are created for each risk indicator that is identified on the current date.
 - The alert end date is updated to the current date.
 - The alert score is updated if the existing score is less than the new score for the day.
 - The new evidences for the day is linked to the existing alert.
 - New parties that are not already on the alert are linked to the alert. New parties would be the top 5 parties for the ticker for the current date.
4. After the alert is create, if the ticker goes into an undecided state, the risk score will start decaying daily. If the score is not 0, the alert is updated as indicated in step 3. For an undecided state, the alert has no pump or dump evidences for the date.

Spooing detection use case

The spoofing detection use case implementation analyzes market data events and detects spoofing patterns.

A spoofer is a trader who creates a series of bulk buy or sell orders with increasing bid or decreasing ask prices with the intention of misleading the buyers and sellers in a direction that results in a profit for the spoofer. The spoofer cancels the bulk orders before they are completed and then sells or buys the affected stocks at a favorable price that results from the spoofing activity. By analyzing the stock data that is streaming in from the market, the spoofing detection use case detects spoofing activity in near real time.

Triggering the spoofing rules

The spoofing use case implementation requires order, execution, and quote data to detect the spoofing pattern. Loading the data triggers the spoofing rules and updates the alert and score tables in the database.

Understanding the spoofing results

The spoofing use case leverages the Trade Surveillance Toolkit to detect spoofing. It analyzes the market data by looking at the events that are fired by the toolkit and generates alerts if a spoofing pattern is detected. The evidence is then used to determine whether an alert needs to be generated. This decision is made by the inference engine. The alert and the evidence are stored in the Surveillance Insight database by using the REST services.

Spoofing user interface

A spoofing alert appears in the **Alerts** tab.

ALERTS		EMPLOYEES				
Score	Type	Date	ID	Ticker	Asset Class	Status
100%	Spoofing	Apr 10, 2017	1480696753	NO	Equity	New
100%	Party Behavior Risk	Apr 21, 2017	148075040803			New
100%	Party Behavior Risk	Apr 21, 2017	148075449558			New
100%	Party Behavior Risk	Apr 21, 2017	1480756957506			New
100%	Party Behavior Risk	Apr 21, 2017	1480756334789			New
100%	Party Behavior Risk	Apr 21, 2017	1480758992079			New
100%	Party Behavior Risk	Apr 21, 2017	1480758954805			New

Figure 21: Spoofing alert

Click the alert to see the alert overview and reasoning.

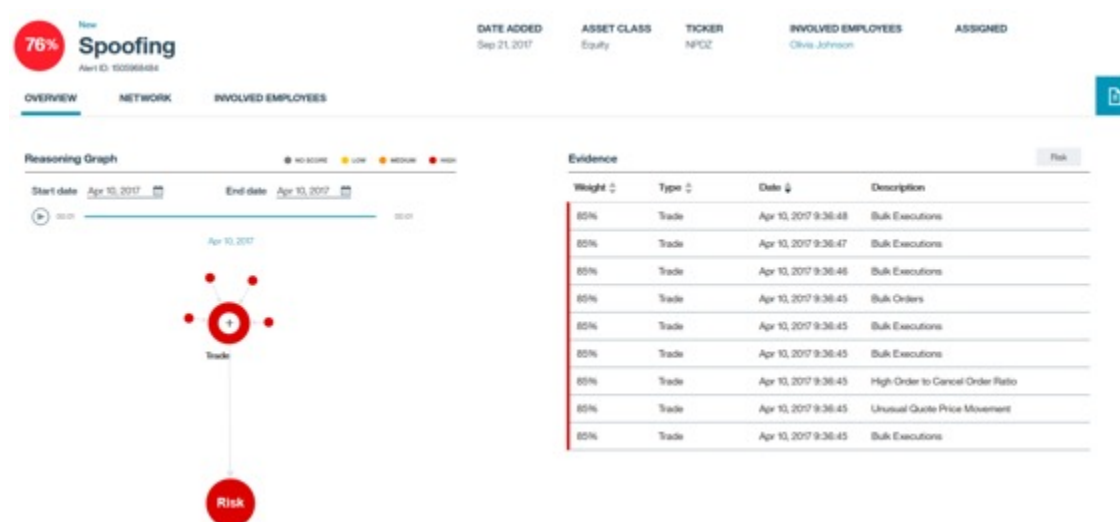


Figure 22: Spoofing overview page

The evidence shows the spoofing pattern where in the bulk orders, unusual quote price movement, and high ratio of orders to cancellation are followed by a series of bulk executions. These evidences contribute to the overall risk as shown in the reasoning graph. In this example, all of the evidences have a 99% weight. This is because for spoofing to happen, each of the events, represented by the risk indicators, should necessarily happen. Otherwise, the pattern would not qualify for spoofing.

Off-market use case

The off-market use case works on forex transaction data and detects transactions that have a deal rate that are out side of the market reference data range at the time of the transaction. The use case combines this evidence with the history of alerts of the trader that are involved in the transaction and decides whether an alert needs to be generated.

Triggering the off-market use case

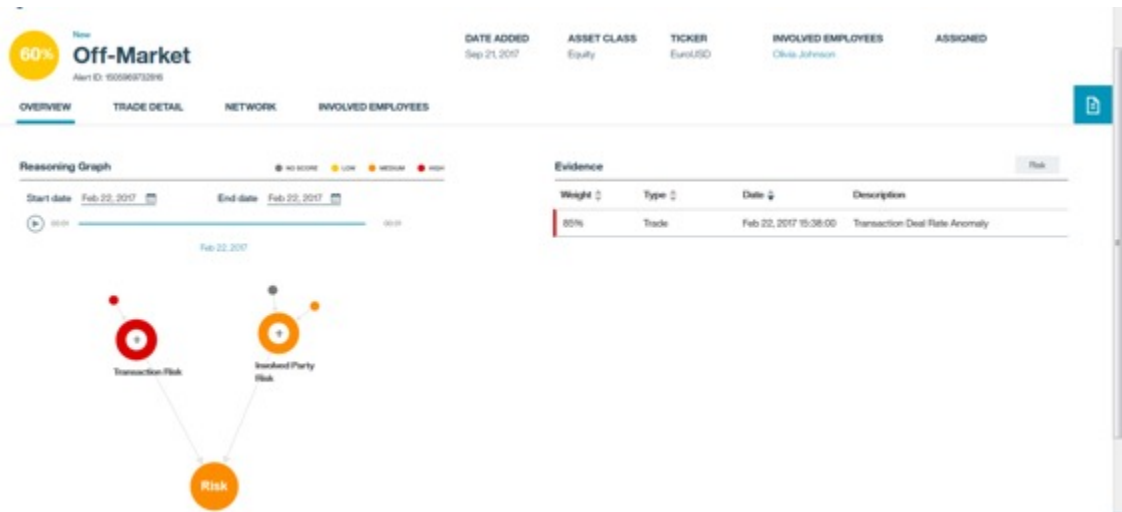
The off-market use case implementation requires forex transaction data and market reference data—high and low deal rates for every minute—from the equity market. Loading the data into HDFS triggers the off-market detection.

Understanding the off-market results

An off-market alert appears in the Surveillance Insights dashboard as follows:

ALERTS						
Score	Type	Date	ID	Ticker	Asset Class	Status
60%	Off-Market	Sep 21, 2017	150596652117	EuroUSD	Equity	New
60%	Off-Market	Sep 21, 2017	150596673066	EuroUSD	Equity	New
60%	Off-Market	Sep 21, 2017	1505966732916	EuroUSD	Equity	New
60%	Off-Market	Sep 21, 2017	1505966733668	EuroUSD	Equity	New
60%	Off-Market	Sep 21, 2017	1505966734837	EuroUSD	Equity	New

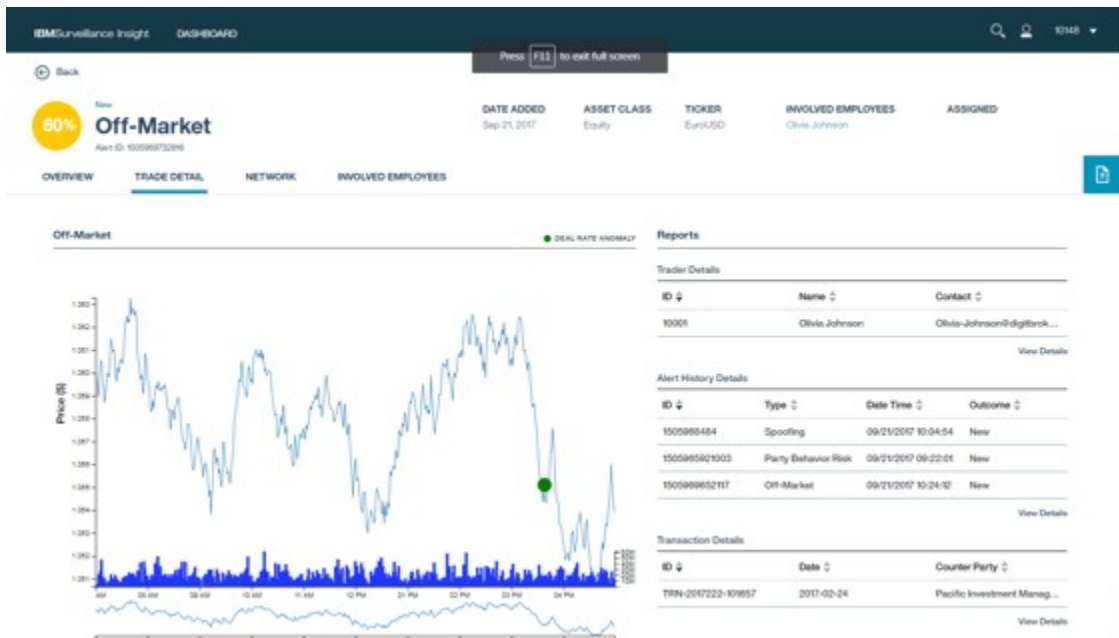
Click the alert to view the off-market alert overview.



As shown in the diagram, the reasoning chart contains two categories of risk indicators: transaction risk and party risk. Each category has one risk indicator:

1. Deal rate anomaly, which indicates that there is an anomaly in the transaction deal rate.
2. Past Alert History, which indicates that the involved party has a history of alerts in the past.

The off-market alert also contains the trade details page that shows the following trade chart:



The trade chart contains the price and volume chart of the transactions and also the point at which the deal rate anomaly occurred, which is indicated by a circle. Further details about the alert history, the transaction, and the party appear in the details sections of the trade report that is displayed on the right side.

Front running use case

The front running use case is designed to detect potential cases where an employee deals ahead of a client. To do this, the employee takes advantage of advance knowledge of large pending orders from a client that has a potential to impact the market price significantly.

Triggering the front running use case

The front running use case works on order data.

1. Run the Trade Data Processor job (`TradeDataProcessor_SI.sh`) to generate the necessary risk indicators for front running.

Ensure that the order data for the date that needs to be processed is loaded into the following folder in HDFS:

```
/user/sifsuser/order/Order_<yyyy-mm-dd>.csv
```

The schema for the data is as follows:

- ClOrdID
- Symbol
- TransactTime
- OrderType
- OrderQty
- Price
- Side
- PartyID

Refer to the New Single Order specification in FIX 4.4 for a description of the above fields.

2. Run the Front Running Inference job (`FrontRunningInference.sh`).

This job creates the front running alerts, which are displayed in the interface.

Viewing the alerts



Score	Type	Date	ID	Ticker	Asset Class	Status
75%	FrontRunning	Mar 13, 2018	102027810214	PGZ	Equity	New
75%	FrontRunning	Mar 13, 2018	102027810490	PGZ	Equity	New

Figure 23: Front running alerts

Click an alert to display the details.



Figure 24: Front running alert details

Note: The current implementation of the front running use case includes the bulk order and front order risk indicators. It does not include e-comm and party risk indicators.

Click one of the trade evidences or the **Trade Detail** tab to show the trade chart.



Figure 25: Front running trade chart

Extending Trade Surveillance

Solution developers can use the solution's architecture to develop new trade surveillance use cases.

The Surveillance Insight platform is built around the concept of risk indicators, evidences, and alerts. A use case typically identifies a certain type of risk in terms of risk indicators.

One of the first things for any use case implementation on the platform is to identify the risk indicators that are to be detected by the use case. After the risk indicators are identified, the kinds of evidence for the risk must be identified. For example, the indicators might be trade data or email data that showed signs of risk during analysis.

A risk model must be built that uses the evidence so that the inference engine can determine whether an alert must be generated.

The type of alerts that are generated by the use case must also be identified.

The identified risk indicators, the model, and the alert types are then loaded into the The Surveillance Insight database:

- The risk indicators must be populated in the RISK_INDICATOR_MASTER table.
- The risk model must be populated in the RISK_MODEL_MASTER table.
- The alert type must be populated in the ALERT_TYPE_MASTER table.

End-to-end implementation

The following diagram shows the sequence of steps that are involved in developing a new Surveillance Insight for Financial Services (SIFS) trade use case:

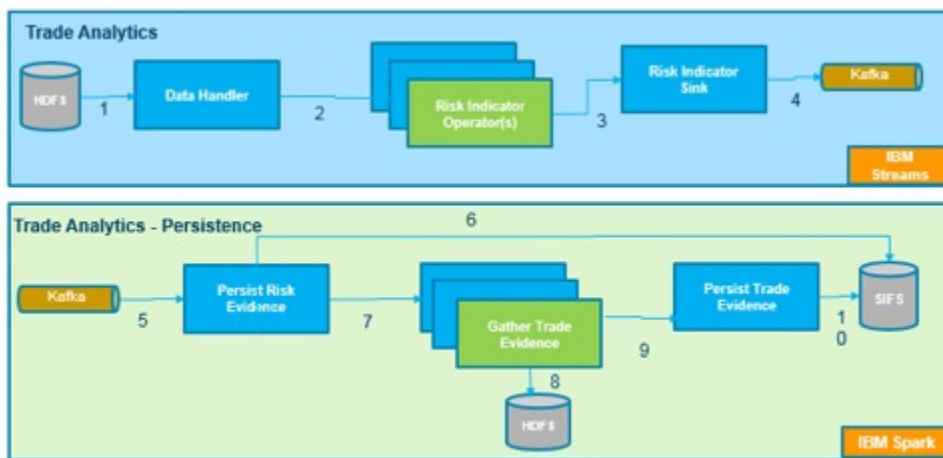


Figure 26: End-to-end implementation for a new use case

1. Data Handler: The DataLoader Streams job that is part of the Surveillance Insights installation monitors certain folders in HDFS. When data is loaded into these folders, the DataLoader picks the content and make it available to the use case implementations.
2. Implementing Risk Indicators: Typically a new use case uses one or more of the existing risk indicators or you must implement new indicators. New risk indicators can be implemented in Streams or in Spark. This decision is made based on what data the risk indicator needs to work on. Typically all indicators that need to work on market data are implemented as Streams operators. They tap into the data that is coming from the DataLoader and perform the necessary risk detection. Operators that need to work on non-market data such as party alert history or proximity of risk indicators should to be implemented in Spark, and then integrated into the end of the day job that uses all of the evidences and invokes the inference engine.

Implementing a new risk indicator involves implementing the core logic that is involved in reading and analyzing the market data for patterns of interest based on the use case requirements.

- a. Understand the event-based approach that is needed to perform trading analytics.

One of the fundamental principles on which the Trade Surveillance Toolkit operates is generating events that are based on stock market data. The toolkit defines a basic event type that is extensible with event-specific parameters. Different types of stock data, such as orders, quotes, executions, and trade data, are analyzed by the operators in the toolkit. Based on the analysis, these operators generate different types of events.

The primary benefit of the event-based model is that it allows the specific use case implementation to delegate the basic functions to the toolkit and focus on the events that are relevant. Also, this model allows the events to be generated one time and then reused by other use cases. It also drastically reduces the volume of data that the use case must process.

- b. Identify the data types and analytics that are relevant to the use case.

Identify what data is relevant and what analytics need to be performed on the data. These analytic measures are then used to identify the events that are of interest to the use case.

- c. Identify Trading Surveillance Toolkit contents for reuse.

Map the data types and events that are identified to the contents in the toolkit. The result of this step is a list of data types and operators that are provided by the toolkit.

- d. Design the Streams flows by leveraging the toolkit operators.

This step is specific to the use case that you are implemented.

In this step, the placement of the Trading Surveillance Toolkit operators in the context of the larger solution is identified. The configuration parameter values for the different operators are identified. Also, data types and operators that are not already present in the toolkit are designed.

- e. Implement the use case and generate the relevant risk indicators.

3. Send out a riskEvent to the RiskEvidenceSink operator. This task takes care of generating a risk evidence and dropping the evidence JSON files into a configured Kafka topic.

It is important to understand the structure and contents of the risk event so that the right information is populated into the SIFS database. For more information about the schema, see [“Risk event schema” on page 27](#).

Every risk event contains an evidence data list. Each item in the list is of the type tradeEvidence. This field is used by the TradeEvidencePersistence spark job to decide what evidence data needs to be populated in the SIFS database so that the trade charts and evidences can be shown in the Surveillance Insights dashboard.

For example, a risk indicator that requires evidences of type orders and quotes to be shown in the dashboard requires two evidence data records in the risk event.

The following is an example of one evidence data record with the type of quote:

```
Datatype : "quote"
  startTime : indicates from where to start reading the quote.csv file in
HDFS
  windowSize : startTime + windowSize (in seconds) is the duration for
which records
will be fetched from the quotes.csv in HDFS.
  list<> id : if there are specific quotes to be shown in the trade
charts, provide the
list of ids here. In this case, the startTime and windowSize need
not be provided.
They will be ignored. Provide empty string for startTime and 0.0 for
windowSize.
```

The second evidence data record is similar, but the type is "order".

4. The Trade Evidence Persistence Spark job that is part of the SIFS installation waits for risk evidences on a configured topic. When it receives evidences, it persists the evidence in the SIFS database. It also

brings in additional trade evidences that might be required to generate the trade charts in the Surveillance Insights dashboard.

5. Create an end-of-day Spark job that reads all of the relevant evidences for the use case (from the SIFS database) and invokes the inference engine which is a REST API call.

The inference engine applies the risk model for the use case and determines whether a new alert must be created for the identified risk evidences. If the engine returns a positive result, it create an alert in the Surveillance Insight database by invoking the createAlert REST service.

Summary

A new trade use case requires the following elements:

1. Implementing a Streams job with the required risk indicators and wiring them to the data loader job exports and the risk evidence sink operator.
2. Implementing any risk indicators that are not directly dependent on market data. This is done by using Spark APIs.
3. Implementing an end-of-day Spark job that uses all of the risk evidences and invokes the inference engine. If necessary, the job also creates an alert in the SIFS database.

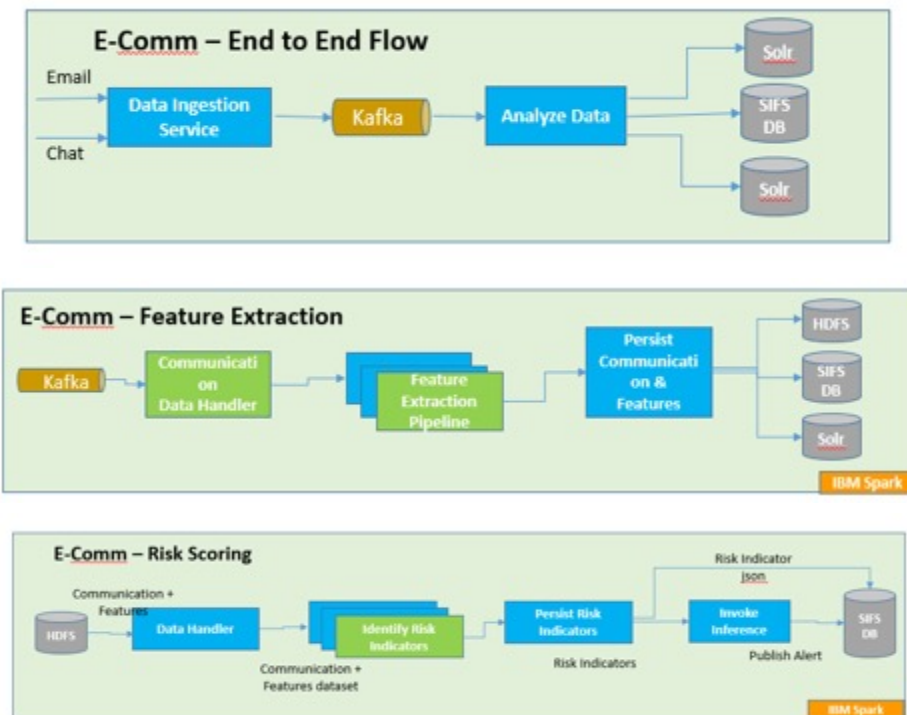
Surveillance Insight for Financial Services provides the following components to help you build your own use cases:

- Surveillance Base Toolkit with reusable types and operators
- Data loader to abstract HDFS folder monitoring
- Evidence persistence job in Spark
- Inference engine APIs
- Alert APIs

Chapter 4. E-Comm surveillance

The e-comm component processes unstructured data such as chat and email data. E-comm data is evaluated against various features, and certain risk indicators are computed. These risk indicators are later analyzed by the inference engine to detect alarming conditions and generate alerts. The e-comm component evaluates the features based on the policies that are defined in the system.

The following diagram shows the end-to-end flow for E-Comm Surveillance. The flow uses Spark.



Some items in the diagrams above, such as the Identify Risk Indicators, the Feature Extraction Pipeline, and the Communication Data Handler, are custom built components that might have to be implemented specifically for a use case, based on the requirements. The other items are pre-built components that are available in Surveillance Insight for Financial Services.

The following e-comm-related feature extractors that are available in Surveillance Insight for Financial Services:

- Emotion and sentiment detection
- Concept mapper
- Entity extractor
- Document classifier (confidential)

The following e-comm-related risk indicators are available in Surveillance Insight for Financial Services:

- Anger anomaly
- Sad anomaly
- Negative sentiment anomaly
- Inbound anomaly
- Outbound anomaly
- Confidential anomaly

- Unusual mail size
- Unusual number of attachments
- Unusual communication timings
- Unusual number of recipients
- Recruit victims
- Recruit co-conspirators
- Efforts on recruit victims
- Efforts on recruit co-conspirators

E-Comm data ingestion

The Surveillance Insight for Financial Services solution processes e-comm data based on policy. At least one policy must be defined in the system to be able to process the e-comm data. A policy is a user-defined document that controls the features that need to be extracted.

After policies are created, the e-comm data can be ingested into the solution. Policies can be created and updated by using the REST services. For more information, see [Policy service APIs](#).

A policy can be defined at the system level or per role.

System level policy

System level features are extracted from every communication. The following is an example of a system level policy:

```
{
  "policy": {
    "policyname": "Policy 1",
    "policycode": "POL1",
    "policytype": "system",
    "policysubcategory": "Sub1",
    "policydescription": "System Policy 1",
    "features": [{
      "name": "emotion"
    }]
  }
}
```

Role level policy

Role level features are extracted based on the initiator party's role and the features that are defined for the role. The following is an example of a role level policy:

```
{
  "policy": {
    "policyname": "Policy 2",
    "policycode": "POL2",
    "policytype": "role",
    "policysubcategory": "Sub2",
    "policydescription": "Role Level Policy",
    "role": [
      "Trader",
      "Banker"
    ],
    "features": [{
      "name": "document classifier"
    }, {
      "name": "concept mapper"
    }, {
      "name": "concept mapper"
    }, {
      "name": "concept mapper"
    }
  ]
}
```

```

    "name": "entity extractor"
  }
}

```

The Surveillance Insight for Financial Services solution expects e-comm data, such as email and chat, in XML format. E-comm data such as email and chat content are ingested by using REST services. These services ingest data into Kafka topics.

Sample e-comm email and chat

A sample email xml is available [here](http://www.ibm.com/support/knowledgecenter/SSWTQQ_2.0.2/samplefile/SurveillanceInsightSampleEcommEmail.xml). (www.ibm.com/support/knowledgecenter/SSWTQQ_2.0.2/samplefile/SurveillanceInsightSampleEcommEmail.xml)

A sample chat xml is available [here](http://www.ibm.com/support/knowledgecenter/SSWTQQ_2.0.2/samplefile/SurveillanceInsightSampleEcommChat.xml). (www.ibm.com/support/knowledgecenter/SSWTQQ_2.0.2/samplefile/SurveillanceInsightSampleEcommChat.xml)

E-Comm feature extraction

After the e-comm data is ingested into a Kafka topic, it is processed by set of Spark jobs.

PersistEmail job

This job processes email xml and extracts features based on the defined policies.



PersistChat job

This job processes chat xml and extracts features based on the defined policies



PersistComm job

This job processes voice and communication data in JSON format and extracts features based on the defined policies.



The above Spark jobs use the ProcessCommunication API.

```

ProcessCommunication processCommunication = new ProcessCommunication();
processCommunication.setConfigProp(sifsProperties);
processCommunication.setMessageType(ECommConstants.EMAIL_MESSAGE_TYPE);

processCommunication.loadSIFSFeatures();

```

```
processCommunication.process("PersistEmail")
```

The Spark job must invoke the interface as shown in the code example above.

The ProcessCommunication API provides an interface to add custom features to the e-comm pipeline. For example, if you have implemented an emotion feature transformer using the Spark ML Pipeline Transformer API and you want it to be invoked as part of e-comm pipeline, you must add that feature before you invoke the process. For example:

```
ProcessCommunication processCommunication = new ProcessCommunication();
//Initialize Transformer
EmotionFeatureExtractor emotionFeatureExtractor = new
EmotionFeatureExtractor(configProp.getProperty("dictPath"),
configProp.getProperty("rulesPath"))
.setInputCol("commText")
.setOutputCol("emotionFeature")
.setFeatureName("emotion");
processCommunication.addFeature(emotionFeatureExtractor);
processCommunication.process("PersistEmail");
```

The PersistChat and the PersistComm jobs use the same ProcessCommunication API and set the appropriate message type so that data is read from the respective Kafka topic and further processed to extract features.

The PersistEmail, PersistChat, and PersistComm jobs performs following tasks:

1. The job loads all of the parties and their contact points in memory as part of the initialization.
2. The job reads communication data from the Kafka topic in micro batches.
3. The data in each micro batch is processed further:
 - a. The data from Kafka is parsed and converted into communication objects.
 - b. The policy service is invoked and all of the policies are registered in the system.
 - c. The communication object is enriched with master data, such as party id, job role for initiator, and all participants, and a feature matrix is created based on the eligible policy per initiator.
 - d. The e-comm pipeline is executed for the features identified in the feature matrix.
 - e. The REST service is invoked to persist the communication and its associated entities in the SIFS database and in Solr. The REST service (/SIFSServices/commsservice/v1/createComm/) creates data in the Comm mapping table and the Comm entity related tables.
 - f. The communication and its extracted features are persisted in the HDFS. Data is stored in HDFS in columnar format in form of csv files. A sample file with data is available [here](http://www.ibm.com/support/knowledgecenter/SSWTQQ_2.0.2/samplefile/SurveillanceInsightSampleEcommFeatureExtraction.csv). (www.ibm.com/support/knowledgecenter/SSWTQQ_2.0.2/samplefile/SurveillanceInsightSampleEcommFeatureExtraction.csv).

Communication schema

Surveillance Insight for Financial Services transforms email and chat xml structure to communication objects.

The communication objects use the following fields:

Table 9: Communication schema objects		
Field name	Field type	Field description
commType	String	Communication Type such as email, chat, phone

Table 9: Communication schema objects (continued)

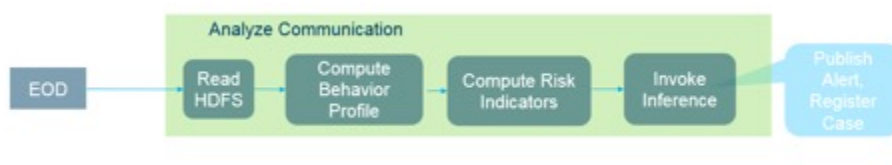
Field name	Field type	Field description
commChannel	String	Communication channel such as voice and e-comm
commText	String	Communication text
commStartTime	String	Communication start time
commEndTime	String	Communication end time
commSubject	String	Communication subject
globalCommId	String	Communication reference ID of the source
metaFeatures	String	The meta features of the communication in JSON format
initiator	String	Communication initiator contact details
participants	Array	Communication participants contact details

E-Comm risk scoring

Communication data is extracted in form of features and is further analyzed by the inference engine to detect if any risk is found and then publishes that in the form of alerts to the Case Manager.

Analyze communication job

After the e-comm data is processed by the Spark job, it is available in HDFS in form of communication data and its extracted features. This data is further analyzed at the end of day by the analyze communication Spark job.



The analyze communication job is implemented by using the ProcessCommunication API. It supports a mechanism to add new risk indicators.

```

ProcessCommunication processCommunication = new ProcessCommunication();
processCommunication.setConfigProp(sifsProperties);

BehaviorRiskIndicator behaviorRiskIndicator = new BehaviorRiskIndicator()
.setInputCol("emotionFeature").setSadOutputCol("RD2")
.setAngerOutputCol("RD1").setSentimentalOutputCol("RD3")
.setSelfThreshold(Double.valueOf(sifsProperties.getProperty("behSelfThreshold")))
.setPopThreshold(Double.valueOf(sifsProperties.getProperty("behPopThreshold")))
.setRiskScoreThreshold(Double.valueOf(sifsProperties.getProperty("riskScoreThreshold")))

processCommunication.addRiskIndicator(behaviorRiskIndicator);
  
```

```
processCommunication.analyze("AnalyzeComm",analysisDate);
```

This job performs the following tasks:

1. The job reads the communication data and extracted features from HDFS for the date for which analysis is being done.
2. This job aggregates the data per party and computes the behavior profile, such as the max anger score or the max disgust score, and persists the data in SIFS database.
3. The following risk indicators are computed:
 - a. Anger anomaly
 - b. Sad anomaly
 - c. Negative sentiment anomaly
 - d. Inbound anomaly
 - e. Outbound anomaly
 - f. Confidential anomaly
 - g. Unusual mail size
 - h. Unusual number of attachments
 - i. Unusual communication timings
 - j. Unusual number of recipients
 - k. Recruit victims
 - l. Recruit co-conspirators
 - m. Efforts on recruit victims
 - n. Efforts on recruit co-conspirators
4. The risk indicators are implemented by using the Spark ML Pipeline Transformer API. To create a new risk indicator, you must write a custom transformer and get the transformer invoked through the pipeline or invoke it independently by using the Transformer transform API.
5. Risk Indicators are persisted to the SIFS database and to Solr through the REST service (/SIFSServices/alertservice/v1/alert/createEvidence).
6. For anomaly based risk indicators, the job computes a profile per party to get the average score for the day and then persists the scores in the SIFS database.
7. The job invokes the inference model—the Party Behavior Risk Model—to detect if the risk evidences have any alert conditions and then persists those alerts in the SIFS database and in Solr and also publishes the alerts to Case Manager.

Profile aggregator job

This job computes the reference profile for each party and then updates the party profile and enterprise profile in the SIFS database.



1. This job computes the profile—MEAN and STD for all parties for a given date. The job expects the date as an input parameter and expects the window parameter to be set in the `sifs.spark.properties` file, where window is the number of days for which the MEAN and STD need to be calculated.
2. This job updates the MEAN and STD values in the `PARTY_PROFILE` for the profile date. It also inserts the MEAN and STD in the `ENTERPRISE_PROFILE` for the profile date. Surveillance Insight expects that the job is run for a given date only one time. If the same job is run for the same date more than one

time, an exception is logged by the job. Only INSERT functions are supported. UPDATE functions are not supported.

Party risk scoring job

This job computes the risk score of a party based on the past alerts for that party.



The job requires the following values:

- PartyRiskDateWindow = 90
- SolrProxyURL = <https://localhost:9443/SIFSServices/surveillanceui/v1/index/update>

The job reads the past alerts based on the PartyRiskDateWindow value.

E-Comm Spark job configuration

The E-Comm Spark job uses the `sifs.spark.properties` file for the job parameters.

`sifs.spark.properties` contains the following parameters:

Table 10: E-Comm Spark job parameters		
Property name	Property value	Description
metadata.broker.list	<IP>:9093	The IP address and port number where the Kafka server is running
security.protocol	SSL	
ssl.truststore.location	/home/sifsuser/security/SIKafkaClientSSLTruststore.jks	The Kafka client SSL truststore location as configured in Kafka
ssl.truststore.password	SurveillanceInsightPass2016	The SSL truststore password as configured in Kafka
ssl.keystore.location	/home/sifsuser/security/SIKafkaClientSSLKeystore.jks	The Kafka client SSL keystore location as configured in Kafka
ssl.keystore.password	SurveillanceInsightPass2016	The SSL keystore password as configured in Kafka
ssl.key.password	SIKafkaKeyPass	SSL key password
bootstrap.servers	<IP>:9093	The IP address and port number where the Kafka server is running
group.id	spark-streaming-notes	Group name to subscribe to the Kafka topic
auto.offset.reset	earliest	Kafka auto offset setting
KafkaEncryptKeyStore	/home/sifsuser/security/SIKafkaDecrypt.jks	Keystore location to decrypt the encrypted data
KafkaEncryptKeyStorePassword	SurveillanceInsightPass2016	The Keystore password
KafkaEncryptKeyPassword	SIKafkaKeyPass	Key password

Table 10: E-Comm Spark job parameters (continued)

Property name	Property value	Description
KafkaEncryptAlias	SIKafkaSecurityKey	Key alias
sparkWarehousePath	file:///home/sifsuser/spark-2.1.1-hadoop2.7/bin/spark-warehouse	
HDFSFilePath	hdfs://<IP>:8020/user/sifsuser/	HDFS path
KafkaSSLEnabled	true	If SSL is enabled for Kafka. The value can be true or false.
KafkaEncryptionEnabled	true	If encryption is enabled for Kafka. The value can be true or false.
master	yarn	If the job is running on a yarn cluster
InferenceREST	https://<IP>:<PORT>/analytics/models/v1/model_predict/	The URL where the Inference REST service is hosted
CreateAlertREST	https://<IP>:<PORT>/SIFSServices/alertservice/v1/alert/createAlert	The REST service URL to create alerts
CreateEvidenceREST	https://<IP>:<PORT>/SIFSServices/alertservice/v1/alert/createEvidence	The REST service URL to create risk evidences
UpdateAlertREST	https://<IP>:<PORT>/SIFSServices/alertservice/v1/alert/updateAlert	The REST service URL to update alerts
PartyRiskDateWindow	90	To get past alerts for last 90 days. Used by party risk scoring job
SolrProxyURL	https://localhost:9443/SIFSServices/surveillanceui/v1/index/update	The REST service URL to update the party risk scoring job in Solr
db2jdbcurl	jdbc:db2://<IP>:50001/SIFS:sslConnection=true;currentSchema=SIFS;	JDBC URL to connect to the SIFS database
db2user	db2inst1	Database user
db2password	db2inst1	Database password
db2TrustStore	/home/sifsuser/SIDB2StreamsClient.jks	Keystore to connect to the secure database
db2TrustStorePassword	SurveillanceInsightPass2016	Keystore password
spark.streaming.kafka.consumer.poll.ms	512	Kafka polling time
spark.streaming.backpressure.enabled	true	Spark streaming parameter
spark.streaming.receiver.maxRate	20	Spark streaming parameter

Table 10: E-Comm Spark job parameters (continued)

Property name	Property value	Description
spark.streaming.kafka.maxRatePerPartition	20	
ecommTopic	sifs.ecomm.in	Kafka topic to which PersistComm spark job is polling to and accepting communication formatted as JSON
window	30	Number of days used to compute the rolling average by the ProfileAggregator Spark job
riskModelCode	PR	Risk model code to invoke the Party Behavior Risk Model. If you have more than one model, separate each by a comma
kafkaDuration	20	Duration in seconds after which the Kafka topic is polled by the Spark jobs
emailTopic	sifs.email.in	Kafka topic to which the PersistEmail Spark job is polling to and accepting email as XML files
chatTopic	sifs.chat.in	Kafka topic to which the PersistChat Spark job is polling to and accepting chat data as XML files
dictPath	/home/sifsuser/dict	Path where the dictionaries for Emotion and Concept Mapper are copied
rulesPath	/home/sifsuser/rules	Path where the rules for Emotion and Concept Mapper are copied
commFolderPath	comm/	Folder name in HDFS where the communication and extracted features are stored
policyServiceUrl	https://<IP>:<PORT>/CommServices/ecomm/policy	The REST service URL for querying policies
policyServiceUser	ibmrest1	Policy service user
policyServicePassword	ibmrest@pwd1	Policy service password
CreateCommREST	https://<IP>:<PORT>/SIFSServices/commervice/v1/createComm	The REST service URL to create comm mapping and entity relationship in the SIFS database
entityServiceUrl	https://<IP>:<PORT>/analytics/models/v1/analyzetext/	The REST service URL to invoke the entity extractor feature
confidentialServiceHost	https://<IP>:<PORT>	The REST service host to invoke the document classifier feature

Table 10: E-Comm Spark job parameters (continued)

Property name	Property value	Description
confidentialServicePath	/nlc/v1/models/ 2017-08-22_17:15:54.243935/ classify/	The REST service URI to invoke the document classifier feature
riskScoreThreshold	0.5	Risk score threshold above which the risk evidences are stored in the SIFS database
behSelfThreshold	1.0	Self-threshold for the Behavior Anomaly Risk Indicator
behPopThreshold	2.0	Population threshold for the Behavior Anomaly Risk Indicator
mailSize	2000	Threshold for the size of the communication content
numberOfAttachments	3	Threshold for the number of attachments
windowStartTime	09:00:00	Acceptable communication start time window
windowEndTime	18:00:00	Acceptable communication end time window
numberOfRecipients	4	Threshold for the number of recipients
commVolumeSelfThreshold	1.0	Self-threshold for inbound and outbound anomaly risk indicator
commVolumePopThreshold	2.0	Population threshold for inbound and outbound anomaly risk indicator
recruitSelfThreshold	1.0	Self-threshold for the recruit victims and recruit conspirators for the anomaly risk indicator
recruitPopThreshold	2.0	Population threshold for recruit victims and recruit conspirators for the anomaly risk indicator
commContentSelfThreshold	1.0	Self-threshold for the confidential anomaly risk indicator
commContentPopThreshold	2.0	Population threshold for the confidential anomaly risk indicator

End-to-end flow for e-comm processing

The following describes the end-to-end flow for e-comm processing.

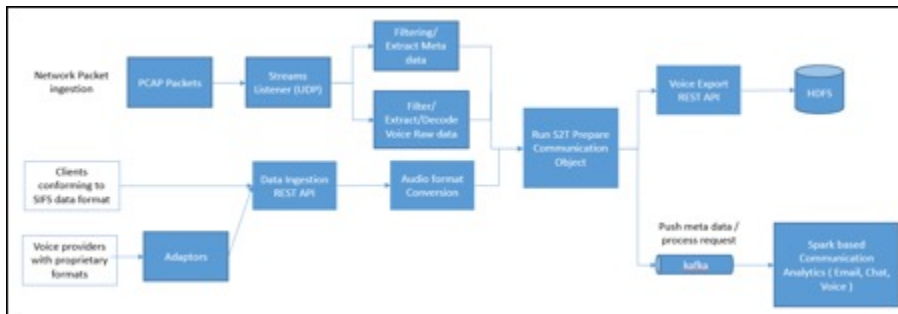
1. Create a risk model for party behavior by using the Model Building tool.
 - a. Create training data and train the model.

- b. Publish the model.
2. Create a natural language classifier (NLC) by using the Model Building tool.
 - a. Create training data and train the classifier model.
 - b. Publish the classifier model.
3. Configure the newly created models for the Spark jobs to consume.
4. Deploy the feature extraction Spark jobs: PersistEmail, PersistChat, PersistComm
5. Deploy the inference job: AnalyzeCommunication
6. Create policy.
7. Ingest the data.
8. Review the alerts in the Surveillance Insight dashboard.

Chapter 5. Voice surveillance

The voice component processes voice data files either in WAV or PCAP formats into text. The text from the voice data files is then evaluated against various features and different risk indicators are calculated. The risk indicators are then analyzed by the inference engine to detect alarming conditions and generate alerts if needed.

The following diagram shows the data flow for voice surveillance.



1. In IBM Surveillance Insight for Financial Services, the voice data can either be fed through network packets or through the voice data ingestion services.
2. For audio files, if the audio file or metadata format is different an adaptor must be built to invoke the voice data ingestion service.
3. The voice data ingestion service triggers the Speech to Text Streams processing flow.
4. The PCAP Streams processing flow reads the voice data from network packets and fetches the metadata from Bluewave APIs.
5. The Speech to Text operators translate the voice data into transcripts with Speak diarisation.
6. The voice artifacts can be optionally exported via an export interface.
7. After the Speech to Text transcript is done, a communication object is then published to the downstream analysis pipeline.

Voice Ingestion service

IBM Surveillance Insight for Financial Services processes voice data in the following formats:

- WAV file in uncompressed PCM, 16-bit little endian, 8 kHz sampling, and mono formats
- PCAP files and direct network port PCAP

The voice ingestion service accepts multipart requests from the user. The multipart requests should contain the following parts:

- A part name "metadata" containing the metadata JSON
- A part name "audiofile" containing the audio binary data

The voice metadata JSON is parsed to get the call start date and gcid values. These values are used to store the audio binary data on the HDFS, where the converted audio file is persisted. The voice metadata JSON is published to the Kafka topic for further processing by the Voice Streams application. The incoming audio file can be WAV, MP3, or OPUS formats. MP4, M4A, M4P, M4B, M4R, and M4V are not supported.

The audio file is converted by using the ffmpeg utility to an uncompressed PCM, 16-bit little endian, 8 kHz sampling mono format WAV file. For example, if an audio file named call001.mp3 is passed to the Voice Ingestion service, the file is converted to cal001.wav and persisted on HDFS.

A sample voice dataset consisting of audio and metadata JSON is provided to help ingest voice files. Use the following command to run the script:

```
./processvoice.sh
```

The following is a sample voice ingestion service multipart request:

HEADERS

```
Content-Type: multipart/form-data; boundary=-----  
-----e73b4c199aee
```

BODY

```
-----e73b4c199aee  
Content-Disposition: form-data; name="metadata"; filename="meta.  
json"  
Content-Type: application/octet-stream  
  
{  
  "Initiator": {  
    "ContactID": "(+1)-204-353-7282",  
    "Name": "Chris Brown",  
    "DeviceID": "dev004",  
    "CallStartTimeStamp": "2017-04-13 11:18:20",  
    "CallEndTimeStamp": "2017-04-13 11:19:26"  
  },  
  "Participants": [{  
    "ContactID": "(+1)-687-225-8261",  
    "Name": "Jaxson Armstrong",  
    "DeviceID": "dev002",  
    "CallStartTimeStamp": "2017-04-13 11:18:20",  
    "CallEndTimeStamp": "2017-04-13 11:19:26"  
  }, {  
    "ContactID": "(+1)-395-309-9915",  
    "Name": "Henry Bailey",  
    "DeviceID": "dev003",  
    "CallStartTimeStamp": "2017-04-13 11:18:20",  
    "CallEndTimeStamp": "2017-04-13 11:19:26"  
  }],  
  "ContactIDType": "phone",  
  "AudioFileName": "File1.wav",  
  "CallStartTimeStamp": "2017-04-13 11:18:20",  
  "CallEndTimeStamp": "2017-04-13 11:19:26",  
  "GlobalCommId": "gcid100906524390995"  
}  
  
-----e73b4c199aee  
Content-Disposition: form-data; name="audiofile"; filename="File  
_2.mp3"  
Content-Type: application/octet-stream  
  
ID3.....vTSS....Logic 10.1.0COM..h.engiTunNORM. 00000284 000002  
87 00004435 000043F8 00005A00 00005A00 00007D9C 00007E47 0000715  
E 0000715E.COM....engiTunSMPB. 00000000  
.....  
.....  
-----e73b4c199aee--
```


Voice data services

IBM Surveillance Insight for Financial Services provides REST API services to allow persisting of voice artifacts and retrieval of audio streams required for playback.

Export REST API service

The Export REST service facilitates the voice streams application to export voice metadata, transcript, and audio file (in WAV format). The default implementation persists these onto HDFS.

The Export REST service accepts a multipart request. The multipart request should contain the following parts:

- A part name "metadata" containing the voice metadata. This part is mandatory.
- A part name "audiofile" containing the audio data. This part is optional.
- A part name "transcript" containing the voice transcript data. This part is optional.

A sample export service multipart request is shown below:

```
Content-Type: multipart/form-data; boundary=-----
-----84989e398b28

-----84989e398b28
Content-Disposition: form-data; name="metadata"; filename="metad
ata.json"
Content-Type: application/octet-stream

{
  "Initiator": {
    "ContactID": "(+1)-204-353-7282",
    "Name": "Chris Brown",
    "DeviceID": "dev004",
    "CallStartTimeStamp": "2017-04-13 11:18:20",
    "CallEndTimeStamp": "2017-04-13 11:19:26"
  },
  "Participants": [{
    "ContactID": "(+1)-687-225-8261",
    "Name": "Jaxson Armstrong",
    "DeviceID": "dev002",
    "CallStartTimeStamp": "2017-04-13 11:18:20",
    "CallEndTimeStamp": "2017-04-13 11:19:26"
  }, {
    "ContactID": "(+1)-395-309-9915",
    "Name": "Henry Bailey",
    "DeviceID": "dev003",
    "CallStartTimeStamp": "2017-04-13 11:18:20",
    "CallEndTimeStamp": "2017-04-13 11:19:26"
  }],
  "ContactIDType": "phone",
  "AudioFileName": "File1.wav",
  "CallStartTimeStamp": "2017-04-13 11:18:20",
  "CallEndTimeStamp": "2017-04-13 11:19:26",
  "GlobalCommId": "gcid100906524390995"
}

-----84989e398b28
Content-Disposition: form-data; name="audiofile"; filename="File
1.wav"
Content-Type: application/octet-stream

RIFF=..WAVEfmt .....@....>.....data.<..INFOISFT....Lavf57.7
1.100.data.<.....
.....
```

```

.....
.....
.....
.....$.....0.3.'.....!. *.....$...
-----84989e398b28

```

Content-Disposition: form-data; name="transcript"; filename="transcript.txt"

Content-Type: text/plain

```

{
  "allUtterances": [{
    "utterances": [{
      "SpeakerId": 0,
      "Utterance": "Hello"
    }, {
      "SpeakerId": 1,
      "Utterance": "Hiebert boss Mister mac was calling"
    }, {
      "SpeakerId": 2,
      "Utterance": "Hey Bob how are you"
    }, {
      "SpeakerId": 3,
      "Utterance": "How's the wife"
    }
  ],
  "utteranceStart": 0.83,
  "utteranceEnd": 7.86,
  "utteranceConfidence": 0.7305149015323721
}, {
  "utterances": [{
    "SpeakerId": 1,
    "Utterance": "She was asking about your day"
  }, {
    "SpeakerId": 2,
    "Utterance": "Ask"
  }, {
    "SpeakerId": 1,
    "Utterance": "You anyway came across some interesting research
about cement company"
  }
  ],
  "utteranceStart": 7.86,
  "utteranceEnd": 14.91,
  "utteranceConfidence": 0.6774061718072295
}, {
  "utterances": [{
    "SpeakerId": 1,
    "Utterance": "Store very good money get into one discuss with
you any interest"
  }
  ],
  "utteranceStart": 14.93,
  "utteranceEnd": 20.25,
  "utteranceConfidence": 0.5302127616382294
}, {
  "utterances": [{
    "SpeakerId": 2,
    "Utterance": "Sure you want to launch we can talking details on"
  }
  ],
  "utteranceStart": 20.26,
  "utteranceEnd": 24.95,
  "utteranceConfidence": 0.7640036944982442
}, {
  "utterances": [{
    "SpeakerId": 1,
    "Utterance": "Sure we can go for lunch at noon today"
  }
  ],

```

```

        "utteranceStart": 24.98,
        "utteranceEnd": 27.07,
        "utteranceConfidence": 0.9037086843769477
    }
}
-----84989e398b28--

```

Data retrieve REST API service

The Data Retrieve REST service is primarily used by the UI to retrieve voice metadata, the URL of the converted audio file (in WAV format), and the voice transcript from HDFS.

The Data retrieve REST service accepts a communication ID. For a given communication ID, the service fetches the available voice metadata, the voice transcript, and the URL of the converted audio file from HDFS and sends a JSON response to the caller.

The service response JSON data contains:

1. Voice metadata under the field "metadata".
2. The field attribute "audiourl" contains the URL for the audio file.
3. Voice transcript under the field attribute "transcript".

A sample Data Retrieve service multipart response is shown below:

```

{
  "metadata": {
    "Initiator": {
      "ContactID": "(+1)-204-353-7282",
      "Name": "Chris Brown",
      "DeviceID": "dev004",
      "CallStartTimeStamp": "2017-04-13 11:18:20",
      "CallEndTimeStamp": "2017-04-13 11:19:26"
    },
    "Participants": [
      {
        "ContactID": "(+1)-687-225-8261",
        "Name": "Jaxson Armstrong",
        "DeviceID": "dev002",
        "CallStartTimeStamp": "2017-04-13 11:18:20",
        "CallEndTimeStamp": "2017-04-13 11:19:26"
      },
      {
        "ContactID": "(+1)-395-309-9915",
        "Name": "Henry Bailey",
        "DeviceID": "dev003",
        "CallStartTimeStamp": "2017-04-13 11:18:20",
        "CallEndTimeStamp": "2017-04-13 11:19:26"
      }
    ],
    "ContactIDType": "phone",
    "AudioFileName": "File_2.wav",
    "CallStartTimeStamp": "2017-04-13 11:18:20",
    "CallEndTimeStamp": "2017-04-13 11:19:26",
    "GlobalCommId": "gcid100906524390995"
  },
  "audiourl": "/SIFSVoiceDataService/voice/v1/2017-04-13/c1cd03c1869f73b079ec6e151ba89d04c6b7452f/audio",
  "transcript": [
    {
      "Speech": "I",
      "endtime": 22.25,
      "starttime": 2.19,
      "Speaker": "Speaker 0"
    }
  ]
}

```

```

    },
    {
      "Speech": "This is Bob joy wells on the call",
      "endtime": 22.25,
      "starttime": 2.19,
      "Speaker": "Speaker 1"
    },
    {
      "Speech": "Merry here",
      "endtime": 22.25,
      "starttime": 2.19,
      "Speaker": "Speaker 0"
    },
    {
      "Speech": "Merry one of you want to take us off strategy",
      "endtime": 35.96,
      "starttime": 22.57,
      "Speaker": "Speaker 1"
    },
    {
      "Speech": "Sure about Stacy I'll play small part orders increase
the protocol by ratio for the strike price of twenty two with an expiration
date of five thirty one",
      "endtime": 35.96,
      "starttime": 22.57,
      "Speaker": "Speaker 2"
    }
  ]
}

```

Audio Streaming REST API Service

The Audio Streaming API is used by the UI to allow audio playback of voice evidences in an alert context. The resultant URL can be played through a standard web browser or an audio player that supports audio streaming.

The JSON response of the Data retrieve REST service contains this API URL under the field "audiourl".

Voice Surveillance Toolkit metadata schema

Table 11: Metadata schema	
Field name	Description
Initiator.ContactID	Call initiator's Contact ID. This can be a phone number or a login name.
Initiator.Name	Call initiator's name
Initiator.DeviceID	Device ID from which the call was initiated by initiator.
Initiator.CallStartTimeStamp	The date and time when the call is initiated by the initiator. The value should be in YYYY-MM-DD HH:MM:SS format.
Initiator.CallEndTimeStamp	The date and time when the call is left by initiator. The value should be in YYYY-MM-DD HH:MM:SS format.

Table 11: Metadata schema (continued)

Field name	Description
Participants.ContactID	Participant's Contact ID. This can be a phone number, IP address, or a login name.
Participants.Name	Name of the participant.
Participants.DeviceID	Device ID from which the call was initiated by Participant.
Participants.CallStartTimeStamp	The date and time when the call is joined by the participant. The value should be in YYYY-MM-DD HH:MM:SS format.
Participants.CallEndTimeStamp	The date and time when the call is left by participant. The value should be in YYYY-MM-DD HH:MM:SS format.
ContactIDType	Allowed values are "loginname" and "phone". The value "phone" is set when a user is identified by their phone number. The value "loginname" is set when a user is identified by their login names, for example, while they are using Cloud9 Trader.
AudioFileName	Audio file name that needs to be processed.
CallStartTimeStamp	The date and time when the call is initiated. The value should be in YYYY-MM-DD HH:MM:SS format.
CallEndTimeStamp	The date and time when the call is ended. The value should be in YYYY-MM-DD HH:MM:SS format.
GlobalCommId	Unique global communication ID attached to this audio.

WAV adaptor processing

Voice communications in WAV format are processed through different adaptors, Kafka processes, and Streams jobs.

IBM Surveillance Insight for Financial Services processes WAV files based on the metadata trigger that is received through pre-defined Kafka topics. The WAV adaptor reads the data from the Kafka topic, decrypts the Kafka message, parses it, and fetches the voice audio file location. The audio content is then passed to the SpeechToText (S2T) toolkit operator for translation. All of the utterances and the speaker diarization are aggregated. The aggregated conversation text is then converted to a communication object and then published to the Kafka topic.

Also, if an export URL is configured, the voice artifacts—the metadata, utterances, and the audio binary—are sent to the export service.

The export capability allows you to export individual voice artifacts to different endpoints. You can specify the following parameters when you submit the Streams job:

- To export all of the voice-related artifacts to the HDFS on hostname1:

```
EXPORTALLURL=https://<hostname1>:<port>/SIFSVoiceDataService/voice/v1/export
```

- To export the voice metadata-related artifacts to the HDFS on hostname2:

```
EXPORTMETADATAURL=https://<hostname2>:<port>/SIFSVoiceDataService/voice/v1/export
```

- To export the voice metadata and transcript to the HDFS on hostname3:

```
HDFS.EXPORTTRANSCRIPTURL=https://<hostname3>:<port>/SIFSVoiceDataService/voice/v1/export
```

- To export the voice metadata and audio data to the HDFS on hostname2 and hostname4:

```
EXPORTAUDIOURL=https://<host2>:<port>/SIFSVoiceDataService/voice/v1/export;https://<hostname4>:<port>/SIFSVoiceDataService/voice/v1/export
```

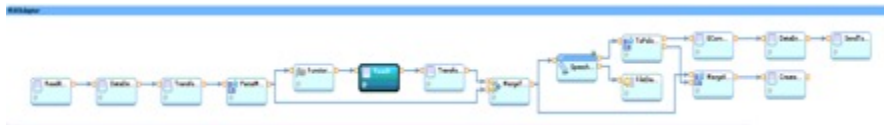


Figure 27: WAV

PCAP format processing

Processing voice data from network involves speech extraction from network packets and IPC-based call metadata extraction.

Networktap job

This job sniffs the network packets from the defined network interface, and then takes the received packets and transfers them to the downstream job. The Standalone job connects to the network interface card by using the IBM Streams Network Toolkit's PacketLiveSource operator. This operator puts the network interface into promiscuous mode to enable gathering of all network packets. The packets are then forwarded to the downstream PCAP Adaptor job by using the IBM Streams Standard Toolkit's TcpSink operator.



Figure 28: Networktap job

PCAP Adaptor job

The PCAP Adaptor job parses PCAP data from a network port. The raw packet data is also exported to the IPC job. Packets are filtered based on IP addresses, subnets, or by login names. The filtered RTP packets are processed and all of the audio packet data that is collected for a given call is exported to the RouteSpeech job. Certain call attributes, such as the callid, channel_id, source, and destination port, are exported to the CorrelateCallMetadata job.

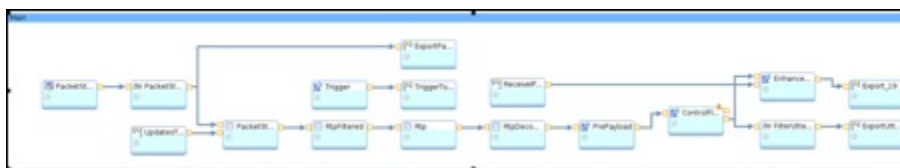


Figure 29: PCAP Adaptor job

IPC job

The IPC metadata extraction jobs consists of two SPLs: IPC and CorrelateCallMetadata. The IPC job receives raw socket data from the PCAP Adaptor job. It identifies the SIP Invite messages of new user logins to their turret devices. It then parses the XML data packets to fetch the device ID and session ID that corresponds to the handsets and stores it in an internal monitored list. This is done to avoid monitoring audio data from speaker ports. After the SIP ACK messages are received, it verifies that the device ID from the ACK is present in the monitored list. It then emits the device ID and the destination port.



Figure 30: IPC job

CorrelateCallMetadata job

The CorrelateCallMetadata job uses Bluewave's (BW) LogonSession API to prepare a list of all of the users who are logged on to the voice network. From the LogonSession response XML, certain attributes about the users, such as their IP address, loginname, userid, zoneid, zonename, loginname, firstname, lastname, and emailid are extracted and cached. Subsequently, for users who log in , their corresponding device IDs are sent by the IPC job. For the incoming device ID, the LogonSession details are fetched from the BW API and the user list in the cache is updated.

Any access to BW needs an authentication token. After an authentication token is created, it is refreshed at regular intervals. Also at regular intervals, a call is made to get the communication history for the last 5 seconds. This is compared with the call records that are extracted from the RTP packets based on the loginname and call start and end times. If the call timing of the communication history record and of RTP packets are within a tolerable deviation, then that communication history record is assigned as the metadata record for the call that is identified in the RTP packets. The identified metadata record is then exported to the RouteSpeech job.



Figure 31: CorrelateCallMetadata job

RouteSpeech job

The RouteSpeech receives audio packets and metadata as tuples. In an organization's voice network, calls emanate from different departments and each department may have vocabulary that is specific to its business. As a result, calls must be routed through specific Speech to Text language models that are based on the source of the call, for example, calls from the Foreign Exchange department might be routed through a S2T language model that was developed specifically for Foreign Exchange whereas calls from the Equity team are routed through a S2T language model that was developed for Equity. This allows a better accuracy rate in speech recognition. Based on the department of the loginname that is associated with the call, the raw speech files are created in a directory that is assigned to a route. Metadata tuples are updated with the partyid and exported to ProcessMetadata job.

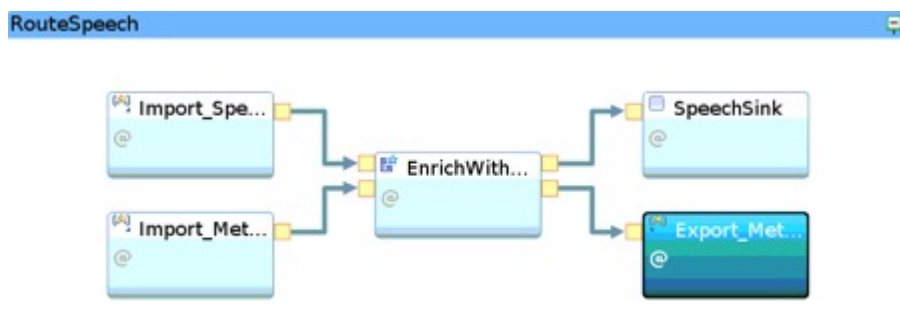


Figure 32: RouteSpeech job

PCAPSpeech job

This job contains the SpeechToText operators for processing audio data from the raw speech files that are created in the RouteSpeech job. After the S2T is complete, it checks if metadata for the concerned call is available. If metadata is available, it is correlated with the converted text from the call and a CommData object is created and published to Kafka. Also, if an export URL is configured, the voice artifacts—the metadata, utterances, and the audio binary—are sent to the export service. The default export service persists the artifacts to the HDFS file system. If the metadata is not available, the audio binary and utterances are persisted to HDFS.



Figure 33: PCAPSpeech job

ProcessMetadata job

The ProcessMetadata job consumes the metadata tuples that are sent by the RouteSpeech job. It checks if transcripts for the concerned call is available. If a transcript is available, it is correlated with metadata and a CommData object is created and published to Kafka. Also, if an export URL is configured, the voice artifacts—the metadata, utterances, and the audio binary—are sent to the export service. The default export service persists the artifacts to the HDFS file system. If a transcript for the call is not available, the metadata is persisted to HDFS.



Figure 34: ProcessMetadata job

Chapter 6. Surveillance Insight data schemas

The Surveillance Insight for Financial Services database schema holds data for the major entities of the business domain.

The major entities are:

- Party

For more information, see [“Party view” on page 60](#).

- Communication

For more information, see [“Communication view” on page 61](#).

- Alert

For more information, see [“Alert view” on page 64](#).

- Trade

For more information, see [“Trade view” on page 66](#).

The following diagram shows the components that update the key sets of tables.

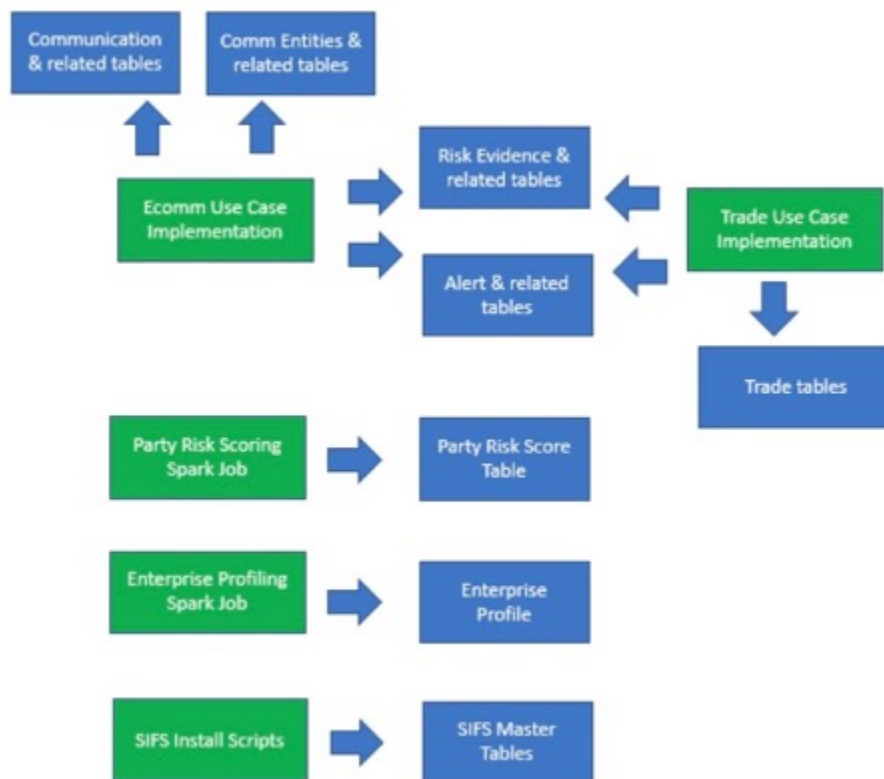


Figure 35: Components that update key tables

Party view

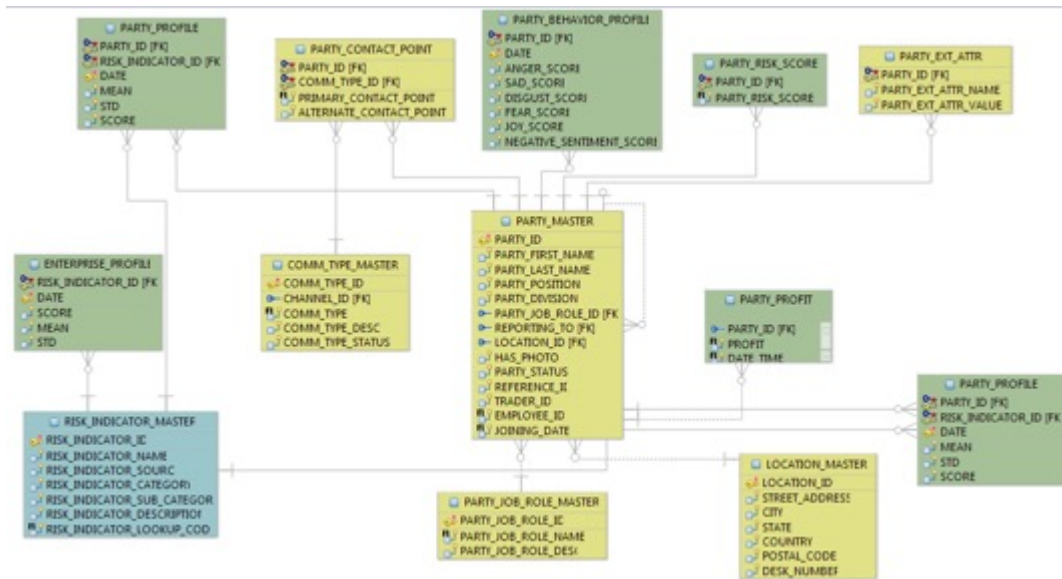


Figure 36: Party view schema

Table name	Description	Populated by
Party_Master	Stores basic party details. Reference ID refers to id in the core systems. TraderId contains mapping to the id that the party uses for trading.	Populated during initial data load and periodically kept in sync with the customer's core master data system.
Party Risk Score	Stores overall risk score for the party based on the party's current and past alert history.	Scheduled Surveillance Insight job that runs daily.
Party Behavior Profile	Stores date-wise scores for various behavior parameters of the party.	Surveillance Insight Ecomm Job daily.
Party Profile	Stores date wise, risk indicator wise statistics for every party.	Surveillance Insight Ecomm Job that updates count field for every communication that is analyzed. Updates the other fields on a configurable frequency.
Enterprise Profile	This table maintains the Mean and Std Deviation for each date and Risk Indicator combination. The mean and standard deviation is for the population. For example, the values are computed using data for all parties.	This table is populated by a Spark job that is based on the frequency at which the Spark job is run. The job reads the date parameter and, for that date, populates the Enterprise profile table. The solution expects to populate the data in the Enterprise profile only once for a specific date.

Table name	Description	Populated by
Party Contact Point	Contains email, voice, and chat contact information for each party.	Populated during the initial data load and periodically kept in sync with the customer's core master data system.
Party Profit	Profit made by the party from trades.	Populated by the spoofing use case implementation.
Party Job Role Master	Master table for party job roles.	Populated during the initial data load and periodically kept in sync with the customer's core master data system.
Comm Type Master	Master table for communication types such email, voice, and chat.	Populated during the initial data load and periodically kept in sync with the customer's core master data system.
Location Master	Master table with location details of the parties.	Populated during the initial data load and periodically kept in sync with the customer's core master data system.
Party_Ext Attr	Table to allow extension of party attributes that are not already available in the schema.	Populated during implementation. Not used by the provided use cases.

Communication view

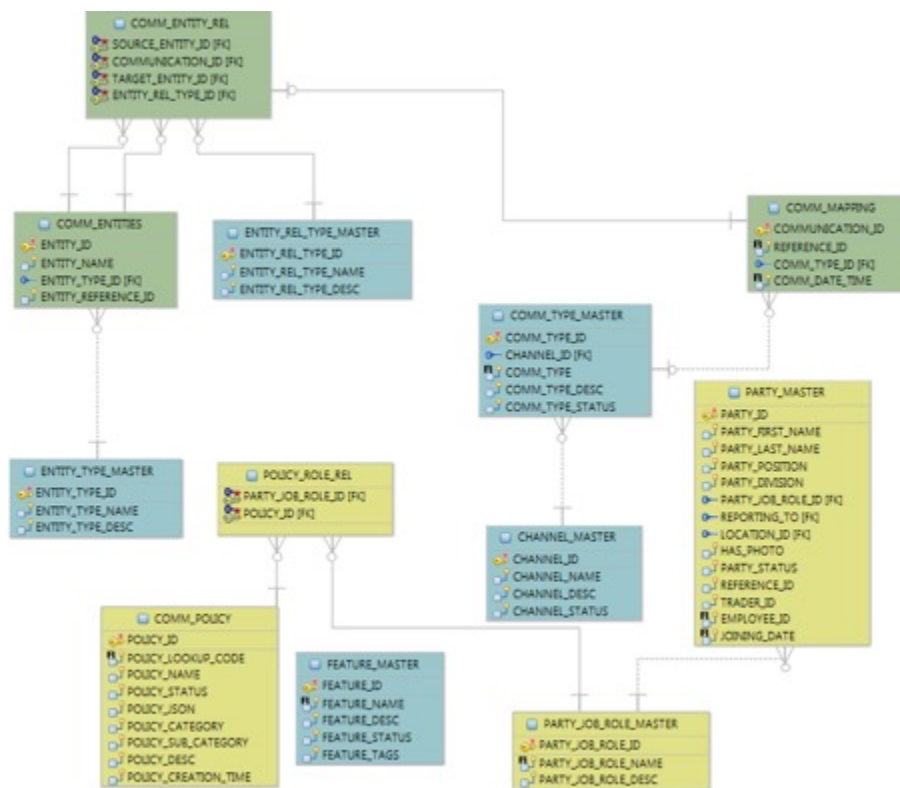


Figure 37: Communication view schema

Table name	Description	Populated by
Communication	Core table that stores extracted metadata from electronic communications (e-comm). It does not store the actual content of the email or voice communication. It stores data such as initiator, participants, associated tags.	This table is populated by the SIFS e-comm streams that analyze e-comm data as each communication comes in.
Comm Involved Party Rel	Stores parties that are involved in a communication.	This table is populated by the SIFS e-comm streams that analyze the e-comm data as each communication comes in.
Comm Entities	Stores entities that are extracted from electronic communications.	Populated by the SIFS e-comm components during e-comm analysis.
Comm Entities Rel	Stores relationships between entities that are extracted from the electronic communications.	Populated by the SIFS e-comm components during e-comm analysis.
Comm Policy	This table maintains the policy details registered in Surveillance Insight. The table has data for both system and role level policies.	Populated through the Policy REST service. The service supports create, update, activate, and deactivate features.
Policy Role Rel	This table maintains the policy to role mapping. For role level policies, the relationship for policy and job role is stored in this table.	Populated when the policy is created in the system by using the REST service. Updates to this table are not supported. It is recommended to create a new policy if there any changes in role.
Feature Master	This table contains a master list of all of the features that are supported by Surveillance Insight for Financial Services.	Master table. Populated during Surveillance Insight product setup.
Comm Feature	This table contains the feature JSON for each communication that is processed by Surveillance Insight. The CORE_FEATURES_JSON column contains the JSON for all of the features in the Feature Master. For metadata, the JSON is stored in the META_DATA_FEATURES_JSON column. The table also provides a provision to store custom feature values in the CUSTOM_FEATURES_JSON column.	This table is populated for every communication that is processed by Surveillance Insight for Financial Services.

Table name	Description	Populated by
Comm Type Master	Master table that stores the different communication types such as voice, email, and chat.	Populated with the supported communication types during product installation.
Channel Master	Master table that stores the different communication channels.	Populated with the supported channel types during product installation. The channels are e-comm and voice.
Entity Type Master	Master table for the type of entities that are extracted from the electronic communications.	Populated with the supported types during product installation. The types are people, organization, and ticker.
Entity Rel Type Master	Master table for types of relationships that are possible between entities that are extracted from the electronic communications.	Populated with the supported types during product installation. The types are Mentions and Works For.
Comm Ext Attr	Extension table to store additional communication attributes during customer implementation.	
NOTES	Stores all of the notes that are created by the case investigators for the alerts.	Surveillance Insight Note service populates this table when triggered from the Surveillance Insight front-end.
OBJECT_TYPE_MASTER	Master table for storing different type of objects against which notes are added.	The table is populated by the communication and alert objects.
VOICE_COMM_EDIT_FEATURES	The table stores all of the annotations that are created against voice communications.	The table is populated when annotations are created or removed. It also populates when a user starts and finishes editing a voice communication.
REVIEW_STATUS_TYPE_MASTER	Master table for storing the different types of review statuses for a voice communication.	The table is populated by the in-progress and completed statuses.

Table name	Description	Populated by
Alert Ticker Rel	Links the tickers that are associated with an alert to the alert itself.	This table is populated by any use case that creates an alert. The createAlert REST service must be used to populate this table.
NOTES	Stores all of the notes that are created by the case investigators for the alerts.	Surveillance Insight Note service populates this table when triggered from the Surveillance Insight front-end.
OBJECT_TYPE_MASTER	Master table for storing different type of objects against which notes are added.	The table is populated by the communication and alert objects.
Evidence Involved Party Rel	Links the parties that are involved in a risk evidence to the evidence itself.	This table is populated by any use case that creates risk evidences. The createEvidence REST service can be used to populate this table.
Evidence Ticker Rel	Links any tickers that are associated with an evidence to the evidences itself.	This table is populated by any use case that creates risk evidences. The createEvidence REST service can be used to populate this table.
Alert Type Master	Master table for the various types of alerts that can be created by the use cases.	Populated with certain alert types during product installation. More types can be created when you create new use cases.
Alert Source Master	Master table for source systems that can generate alerts.	Populated with one alert source for Surveillance Insight during product installation. More sources can be created, depending on the requirements.
Risk Indicator Master	Master table for risk indicators that are generated by the use cases.	Populated with certain indicators for e-comm and trade risk during product installation. More indicators can be created, depending on the requirements.
Risk Evidence Type Master	Master table for evidence types, such as trade, email, and voice.	Populated with a certain type during product installation. More types can be created, depending on the requirements.
Risk Model Master	Master table for the risk models that are used for generating the reasoning graph.	Populated during the product installation with following models: pump-and-dump, spoofing, and party risk behavior. More models can be populated, depending on the requirements of new use cases.

Table name	Description	Populated by
Quote Sample	This table contains samples of quote data for certain durations of time. The quotes that go into this table depend on the specific use case. The quotes are evidences of some kind of risk that is identified by the specific use case. These quotes are sampled from the market data and stored in Hadoop. They are stored in this table for easy access by the user interface layer.	Currently, the spoofing use case populates this table whenever a spoofing alert is created. The sampled quote is the max (bid price) and min (offer price) for every time second.
Order	This table contains orders that need to be displayed as evidence for some alert in the front end. The contents are copied from the market data in Hadoop. The specific dates for which the data is populated depends on the alert.	Currently, the spoofing use case populates this table whenever a spoofing alert is identified.
Execution	This table contains orders that need to be displayed as evidence for some alert in the front end. The contents are copied from the market data in Hadoop. The specific dates for which the data is populated depends on the alert.	Currently, the spoofing use case populates this table whenever a spoofing alert is identified.
Pump Dump Stage	This table contains the pump-and-dump stage for each ticker that shows pump or dump evidence.	This table is populated by the pump-and-dump use case implementation.
Trade Summary	This table contains the ticker summary for each date for tickers that show pump or dump evidence.	This table is populated by the pump-and-dump use case implementation.
Top5Traders	This table contains the top five traders for buy and sell sides for each ticker that shows pump or dump evidence. This table is populated daily.	This table is populated by the pump-and-dump use case implementation.
Intra Day Trade Summary	This table is meant to be used for the Trade charts in the trade use cases. This table contains summary data for trade/ transaction data on a daily basis.	This table is populated by the Trade Evidence Persistence Spark job as and when it receives the risk evidence from the use case implementation.

Table name	Description	Populated by
Trade Evidence	This table contains evidences for specific type like order, trade, execution etc., that needs to be shown in the trade charts for trade use cases. This table, along with the evidence_trade_rel table allow linking the evidences to the alerts through the evidence ids	This table is populated by the Trade Evidence Persistence Spark job as and when it receives the risk evidence from the use case implementation.
Evidence Trade Rel	This table links the trade evidences to the risk evidences and thus indirectly link them to the alert. This table helps the UI services figure out what trade evidences are relevant for a specific alert.	This table is populated by the Trade Evidence Persistence Spark job as and when it receives the risk evidence from the use case implementation.
FX Transaction	This table contains the transaction details for forex transactions. This table is synonymous to the Trade Evidence table except that it has additional fields that are specific to forex.	This table is populated by the Trade Evidence Persistence Spark job as and when it receives the risk evidence from the use case implementation.

Chapter 7. NLP libraries

The solution offers prebuilt custom libraries for some of the Natural Language Processing (NLP) capabilities

The following pre-built libraries are provided:

- Emotion Detection library
- Concept Mapper library
- Document Classifier library

The solution uses Open source frameworks / Libraries such as [Apache UIMA](#) (Unstructured Information Management Application) and [MALLET](#) (MAchine Learning for LanguagE Toolkit).

Note: The libraries come with dictionaries and rules that can be customized.

Emotion Detection library

The Emotion Detection library uses Apache UIMA Ruta (RULE based Text Annotation) and a custom scoring model to detect emotions and sentiment in unstructured data, such as text from emails, instant messages, and voice transcripts.

The library detects the following emotions from the text:

- Anger
- Disgust
- Joy
- Sadness
- Fear

It assigns a score from 0-1 for each emotion. A higher value indicates a higher level of the emotion in the content. For example, an Anger score of 0.8 indicates that the anger is likely to be present in the text. A score of 0.5 or less indicates that anger is less likely to be present.

The library also detects the sentiment and indicates it as positive, negative, or neutral with a score of 0-1. For example, a positive sentiment score is 0.8 indicates that positive sentiment is likely expressed in the text. A score of 0.5 or less indicates that positive sentiment is less likely expressed in the text. The sentiment score is derived from the emotions present in the text.

How the library works

The solution uses dictionaries of emotions and rules to detect the emotions in text and a scoring model to score the emotions.

The dictionaries are contained in the `anger_dict.txt`, `disgust_dict.txt`, `fear_dict.txt`, `joy_dict.txt`, and `sad_dict.txt` files. Each dictionary is a collection of words that represent emotion in the text.

The rule file is based on Ruta Framework and it helps the system to annotate the text based on the dictionary lookup. For example, it annotates all the text that is found in the anger dictionary as Anger Terms. The position of this term is also captured. All the inputs are fed into the Scoring model to detect the sentence level emotions and also the document level emotion. The document level emotion is returned as the overall emotion at the document level.

The following code is an example of a rule definition.

```
PACKAGE com.ibm.sifs.analytics.emotion.types;
```

```

# Sample Rule
# load dictionary
WORDLIST anger_dict = 'anger_dict.txt';
WORDLIST joy_dict = 'joy_dict.txt';

# Declare type definitions
DECLARE Anger;
DECLARE Joy;

# Detect sentence
DECLARE Sentence;
PERIOD #{ -> MARK(Sentence)} PERIOD;

MARKFAST(Anger, anger_dict, true);
MARKFAST(Joy, joy_dict, true);
# Same for other emotions

```

The emotion detection dictionary

Emotion detection is a java based library and is available as JAR. Currently, it is used in the Real-time Analytics component to detect the emotions in real time and score the emotions in the incoming documents.

As shown in the following diagram, it offers two functions:

- Initialize, which initializes the Emotion library by loading the dictionary and the rules. This function needs to be called only once, and must be started when dictionaries or rules are changed.
- Detect Emotion, which takes text as input and returns a JSON string as a response.

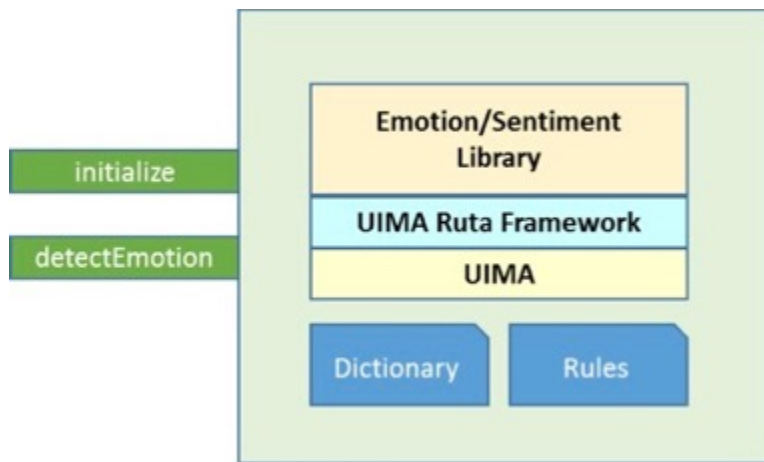


Figure 40: Emotion detection library

Definitions

```
public static void initialize(String dictionaryPath, String rulePath) throws Exception
```

```
public static String detectEmotion(String text)
```

Sample input

```

The investment you made with ABC company stocks are doing pretty good. It
has increased
50 times. I wanted to check with you to see if we can revisit your
investment portfolios for better

```

investments and make more profit. Please do check the following options and let me know. I can visit you at your office or home or at your preferred place and we can discuss on new business. Market is doing pretty good. If you can make right investments now, it can give good returns on your retirement.

Sample response

```
{
  "sentiment": {
    "score": 1,
    "type": "positive"
  },
  "emotions": {
    "joy": "1.0",
    "sad": "0.0",
    "disgust": "0.0",
    "anger": "0.0",
    "fear": "0.0"
  },
  "keywords": {
    "negative": [],
    "joy": ["pretty", "retirement", "good", "profit"],
    "sad": ["retirement"],
    "disgust": [],
    "anger": [],
    "fear": ["retirement"]
  },
  "status": {
    "code": "200",
    "message": "success"
  }
}
```

Starting the module

```
// Initialize Module (ONLY ONCE)
EmotionAnalyzer.initialize(<path to dictionaries>, <path to rule file>);

// Invoke the library (For every incoming document)
String resultJSON = EmotionAnalyzer.detectEmotion(text);
```

Note: Errors or exceptions are returned in the JSON response under the Status element with a code of 500 and an appropriate message, as shown in the following example.

```
"status": {
  "code": "200",
  "message": "success"
}
```

Concept Mapper library

The Concept Mapper library uses Apache UIMA Ruta (RULe based Text Annotation) to detect the concepts in unstructured text such as emails, instant messages, or voice transcripts.

The library detects the following concepts from the text.

- Tickers—Stock Symbol

- **Recruit Victims**—Evidence of a trader who is trying to get clients to invest in a specific ticker. This activity is indicated as "Recruit Victims."
- **Recruit Conspirators**—Evidence of a trader who is collaborating with other traders to conduct a market abuse activity such as "pump/dump." This activity is indicated as "Recruit Conspirators" in the surveillance context.

Note: If there is more than one ticker in the text, all the tickers are extracted and returned as a comma-separated string.

How the library works

The solution uses a dictionary of tickers, keywords, or phrases that represent Recruit Victims and Recruit Conspirators, and concepts and rules to detect the concepts in the text. The dictionaries include the `recruit_conspirators.txt`, `recruit_victims_dict.txt`, and `tickers_dict.txt` files. Each dictionary is a collection of words that represent different concepts in the text.

The rule file is based on Ruta Framework and it helps the system to annotate the text based on the dictionary lookup. For example, it annotates all the text that is found in the Recruit Victims dictionary as Recruit Victims Terms. The position of this term is also captured.

The following code is an example of a rule.

```
PACKAGE com.ibm.sifs.analytics.conceptmapper.types;

# Sample Rule
# Load Dictionary
WORDLIST tickers_dict = 'tickers_dict.txt';
WORDLIST recruit_victims_dict = 'recruit_victims_dict.txt';
WORDLIST recruit_conspirators_dict = 'recruit_conspirators.txt';
WORDLIST negative_dict = 'negative_dict.txt';

# Type definitions
DECLARE Ticker;
DECLARE RecruitConspirators;
DECLARE RecruitVictims;
DECLARE Negative;

# Annotate/Identify the concepts
MARKFAST(Negative, negative_dict, true);
MARKFAST(Ticker, tickers_dict, false);
MARKFAST(RecruitConspirators, recruit_conspirators_dict, true);
MARKFAST(RecruitVictims, recruit_victims_dict, true);
```

The Concept Mapper dictionary

Concept Mapper is a java-based library and is available as JAR. Currently, it is used in the Real-time analytics component to detect the concepts in real time from the incoming text. As shown in the following diagram, it offers the following functions:

- **Initialize**, which initializes the library by loading the dictionary and the rules. This function needs to be called only once, and must be started when dictionaries or rules are changed.
- **Detect concepts**, which take text as input and returns a JSON string as a response.

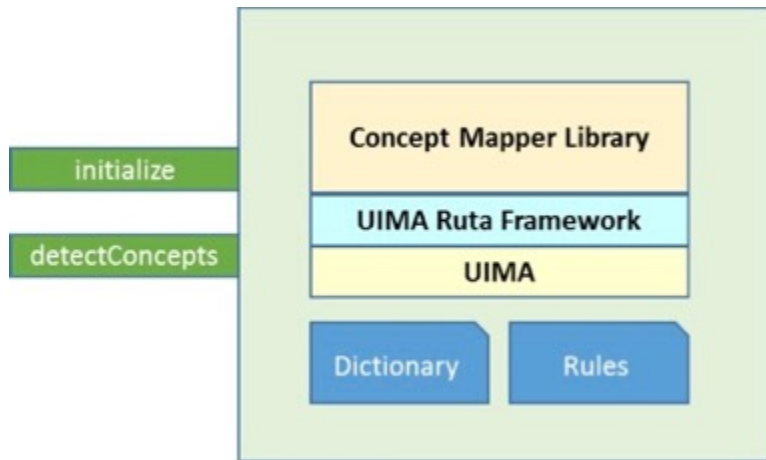


Figure 41: Concept mapper library

Definitions

```
public static void initialize(String dictionaryPath, String rulePath) throws
Exception
```

```
public static String detectConcepts(String text)
```

Sample input

I wanted to inform you about an opportunity brought to us by an insider, Mr. Anderson, from ABC Corporation. They specialize in manufacturing drill bits for deep-sea oil rigs. Mr. Anderson owns about 35% of the float and would like us to help increase the price of his company's stock price. If we can help increase the price of the stock by 150%, we would be eligible for a substantial fee and also 1.5% of the profit Mr. Anderson will make disposing the shares at the elevated price. Would you be interested in joining our group in helping Mr. Anderson?

Sample response

```
{
  "concepts": {
    "recruitconspirators": true,
    "tickers": ["ABC"],
    "recruitvictims": false
  },
  "status": {
    "code": "200",
    "message": "success"
  }
}
```

Starting the module

```
// Initialize Module (ONLY ONCE)
ConceptMapper.initialize(<path to dictionaries>, <path to rule file>);
```

```
// Invoke the library (For every incoming document)
String resultJSON = ConceptMapper.detectConcepts(text);
```

Note: Errors or exceptions are returned in the JSON response under the Status element with a code of 500 and an appropriate message, as shown in the following example.

```
"status": {
  "code": "200",
  "message": "success"
}
```

Classifier library

The Classifier library uses MALLET to classify documents into predefined classes and associate probability scores to the classes.

The library can be used to define the following classifications:

- Confidential / Non-confidential documents
- Business / Personal
- News / Announcement / Promotional
- Trading / Non-Trading

How the library works

The Classifier library uses a client/server model. The server library is used to train the model and for the export of the classifier models. The client library uses the classifier model and to classify the incoming documents in real time.

The Classifier library

Classifier is a java based library and is available as JAR. Currently, it is used in the Real-time analytics component to detect the concepts in real time from the incoming text. As shown in the following diagram, it offers the following functions:

- Initialize, which initializes the library by loading the prebuilt classification models. The library can be initialized with multiple classifiers. This function needs to be called only one time, and must be started when dictionaries or rules are changed.
- Classify docs, which take text as input and returns a JSON string as a response.

Sample input

```
I wanted to inform you about an opportunity brought to us by an insider, Mr. Anderson, from ABC Corporation. They specialize in manufacturing drill bits for deep-sea oil rigs. Mr. Anderson owns about 35% of the float and would like us to help increase the price of his company's stock price. If we can help increase the price of the stock by 150%, we would be eligible for a substantial fee and also 1.5% of the profit Mr. Anderson will make disposing the shares at the elevated price. Would you be interested in joining our group in helping Mr. Anderson?
```

Sample response

```
{
  "classes": [{
```



```

        "confidence": 0.22,
        "class_name": "Confidential"
    }, {
        "confidence": 0.77,
        "class_name": "Non-Confidential"
    }
  ],
  "top_class": "Non-Confidential",
  "status": {
    "code": "200",
    "message": "success"
  }
}

```

Server-side library

The server-side library is RESTful and exposes APIs to operate with the Classifier model. It offers the following services.

Table 12: Server-side library			
Method	URL	Input	Output
POST	/text/v1/classifiers	JASON Payload	JSON response
GET	/text/v1/classifiers		JSON response
GET	/text/v1/classifiers/{classifierid}		JSON response
DELETE	/text/v1/classifiers/{classifierid}		JSON response
POST	/text/v1/classifiers/{classifierid}/export	Query param: Export model path	JSON response Exported Model

Service details

1. Create classifier

Table 13: Create classifier			
Method	URL	Input	Output
POST	/text/v1/classifiers	JASON Payload	JSON response

The service allows users to create and train any number of classifiers. It also allows users to export the trained model to use it from the client side, for example, by using a CURL command to try the POST:

```

curl -k -H 'Content-Type:application/json' -X POST --data-binary
@payload.json
http://localhost:9080/analytics/text/v1/classifiers

```

The payload provides details, such as the Classifier name and training data folders for each class in the classifier. The documents need to be available in the server. Currently, the library does not support uploading of training documents.

Note: If the existing classifier name is provided, the classifier overrides it.

The following code is an example JSON payload:

```

{
  "name": "confidential",
  "training-data": [
    { "class": "confidential",

```

```

        "trainingdatafolder":"/home/sifs/training_data/
confidential"},
    {"class":"non-confidential",
     "trainingdatafolder":"/home/sifs/training_data/non-
confidential"}
  ]
}

```

The following code is an example response:

```

{
  "status": {
    "message": "Successfully created Classifier - confidential",
    "status": "200"
  }
}

```

2. Get all classifiers

The service lists the available classifiers in the system.

Table 14: Get all classifiers			
Method	URL	Input	Output
POST	/text/v1/classifiers		JSON response

The following code is an example CURL command:

```

curl -k -H 'Content-Type:application/json'
http://localhost:9080/analytics/text/v1/classifiers

```

The following code is an example response:

```

{
  "classifiers": ["confidential"],
  "status": {
    "message": "Success",
    "code": "200"
  }
}

```

3. Get details on a classifier

The service retrieves the details of the requested classifier.

Table 15: Get details on a classifier			
Method	URL	Input	Output
GET	/text/v1/classifiers/ {classifierid}		JSON response

The following code is an example CURL command:

```

curl -k -H 'Content-Type:application/json'
http://localhost:9080/analytics/text/v1/classifiers/confidential

```

The following code is an example response:

```

{
  "classifiers": ["confidential"],
  "status": {
    "message": "Success",

```

```
    "code": "200"
  }
}
```

4. Delete Classifier

This service deletes the requested classifier from the library.

Table 16: Delete classifier			
Method	URL	Input	Output
DELETE	/text/v1/classifiers/ {classifierid}		JSON response

The following code is an example CURL command:

```
curl -k -X DELETE http://localhost:9080/analytics/text/v1/classifiers/  
confidential/
```

The following code is an example response:

```
{"status":{"message":"Classifier - confidential is successfully  
deleted"},"code":"200"}}
```

5. Export Classification Model

The service exports the model file for the classifier. The model file can be used on the client side. It is a serialized object and it can be deserialized on the client side to create the classifier instance and classify the documents.

Table 17: Export classification model			
Method	URL	Input	Output
POST	/text/v1/classifiers/ {classifierid}/export	Query param: Export model path	JSON response, Exported Model

The following code is an example CURL command:

```
curl -k -X POST  
http://localhost:9080/analytics/text/v1/classifiers/confidential/export/?exportmodelpath=/home/  
sifs/classifiers
```

The following code is an example response:

```
{  
  "status": {  
    "message": "Classification Model successfully exported /home/sifs/  
classifiers",  
    "code": "200"  
  }  
}
```

Chapter 8. Inference engine

IBM Surveillance Insight for Financial Services contains an implementation of a Bayesian inference engine that helps in deciding whether an alert needs to be created for a set of evidences available on a specific date.

The inference engine takes the risk model (in JSON format) and the scores for each risk indicator that is referred to by the risk model as input. The output of the engine is a JSON response that gives the scores of the risk indicators and the conclusion of whether an alert needs to be created.

Inference engine risk model

The IBM Surveillance Insight for Financial Services inference engine risk model is generated by using the Surveillance Design Studio.

The following code is a sample.

```
    "y": 132
  }, {
    "parents": [],
    "desc": "Transaction Deal Rate Anomaly",
    "lookupcode": "TXN09",
    "outcome": ["low", "medium", "high"],
    "threshold": [0.33, 0.75],
    "id": "109",
    "category": "Transaction",
    "level": 2,
    "source": "TRADE",
    "subcategory": "NO_DISPLAY",
    "name": "Transaction Deal Rate Anomaly",
    "probabilities": [0.33, 0.33, 0.33],
    "x": 440.5,
    "y": 117
  }, {
    "parents": ["109"],
    "desc": "Transaction Risk",
    "lookupcode": "TXN07",
    "outcome": ["low", "medium", "high"],
    "rules": ["RD109 == 'low'", "RD109 == 'medium'", "RD109 == 'high'"],
    "threshold": [0.33, 0.75],
    "id": "107",
    "category": "Derived",
    "level": 1,
    "source": "TRADE",
    "subcategory": "NO_DISPLAY",
    "name": "Transaction Risk",
    "probabilities": [1, 0, 0, 0, 1, 0, 0, 0, 1],
    "x": 443,
    "y": 242
  }, {
    "parents": ["105", "106"],
    "desc": "Involved Party Risk",
    "lookupcode": "TXN08",
    "outcome": ["low", "medium", "high"],
    "rules": ["RD105 == 'low' && RD106 == 'low'", "(RD105 == 'medium' &&
RD106 ==
'low') || (RD105 == 'low' && RD106 == 'medium') || (RD105 == 'medium' &&
RD106 ==
'medium') || (RD105 == 'high' && RD106 == 'low') || (RD105 == 'low' && RD106
==

```

```

'high')", "(RD105 == 'high' && RD106 == 'high') || (RD105 == 'high' && RD106
==
'medium') || (RD105 == 'medium' && RD106 == 'high')"],
    "threshold": [0.33, 0.75],
    "id": "108",
    "category": "Derived",
    "level": 1,
    "source": "TRADE",
    "subcategory": "NO_DISPLAY",
    "name": "Involved Party Risk",
    "probabilities": [1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0,
1, 0, 1,
0, 0, 0, 1, 0, 0, 1],
    "x": 126,
    "y": 247
  }, {
    "parents": ["107", "108"],
    "desc": "Risk",
    "lookupcode": "RD15",
    "outcome": ["low", "medium", "high"],
    "rules": ["RD107 == 'low' && RD108 == 'low'", "(RD107 == 'medium' &&
RD108 ==
'low') || (RD107 == 'low' && RD108 == 'medium') || (RD107 == 'medium' &&
RD108 ==
'medium') || (RD107 == 'high' && RD108 == 'low') || (RD107 == 'low' && RD108
==
'high')", "(RD107 == 'high' && RD108 == 'high') || (RD107 == 'high' && RD108
==
'medium') || (RD107 == 'medium' && RD108 == 'high')"],
    "threshold": [0.33, 0.75],
    "id": "15",
    "category": "Derived",
    "level": 0,
    "source": "COMM",
    "subcategory": "NO_DISPLAY",
    "name": "Risk",
    "probabilities": [1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0,
1, 0, 1,
0, 0, 0, 1, 0, 0, 1],
    "x": 290,
    "y": 424
  }
}
}

```

The content of the risk model is a network of nodes and edges. The nodes represent the different risk indicators and the edges represent the link between the risk indicators nodes and the aggregated risk indicator nodes.

The risk indicator look-up codes that are mentioned in the risk_indicator_master table are used to refer to specific risk indicators.

Each node also has an X,Y co-ordinate for the placement of the node in the user interface.

If you create a new model for a use case, the model must be updated in the risk_model_master table. The risk_model_master table contains one row per use case.

Run the inference engine

You can run the inference engine as a REST API.

```

POST
http://<IP>:<PORT>/analytics/models/v1/model_predict/

```

The inference engine input is a JSON that consists of the following elements:

- riskModelTrained: the trained risk model JSON
- toBeScoredData: the list of risk indicator values (current and historical if any) used for scoring

The following code is a sample JSON input:

```
{
  "riskModelTrained": {
    "metadata": {
      "applyDecay": true
    },
    "nodes": [{
      "lookupcode": "RD15",
      "level": 0,
      "name": "Risk",
      "threshold": [0.33, 0.75],
      "source": "Comm",
      "id": "15",
      "subcategory": "NO_DISPLAY",
      "category": "Derived",
      "probabilities": [
        [1, 1, 0.999, 1, 1, 0.999, 0.997, 0.001, 0.019],
        [0, 0, 0.001, 0, 0, 0.001, 0.003, 0.999, 0.981]
      ],
      "outcome": ["low", "medium", "high"],
      "parents": ["27", "26"],
      "desc": "Risk"
    }, {
      "lookupcode": "RD27",
      "level": 1,
      "name": "Comm",
      "threshold": [0.33, 0.75],
      "source": "Comm",
      "id": "27",
      "subcategory": "NO_DISPLAY",
      "category": "Derived",
      "probabilities": [
        [1, 0, 0, 0, 0, 0.001, 0, 0.002, 0.333, 0, 0, 0.001, 0, 0,
        0.005, 0.001, 0.003, 0.333, 0, 0.002, 0.333, 0.001, 0.004, 0.019, 0.019,
        0.333, 0.333, 0, 0, 0.002, 0, 0, 0.004, 0.002, 0.003, 0.019, 0, 0, 0.005, 0,
        0.001, 0.01, 0.002, 0.019, 0.333, 0.001, 0.005, 0.333, 0.002, 0.004, 0.333,
        0.333, 0.333, 0.333, 0, 0.001, 0.019, 0.001, 0.002, 0.019, 0.01, 0.333,
        0.333, 0.001, 0.003, 0.019, 0.003, 0.01, 0.333, 0.019, 0.333, 0.333, 0.01,
        0.01, 0.333, 0.333, 0.333, 0.333, 0.333, 0.333, 0.333],
        [0, 1, 0, 1, 1, 0.001, 0, 0.002, 0.333, 1, 1, 0.001, 1, 1,
        0.005, 0.001, 0.003, 0.333, 0, 0.002, 0.333, 0.001, 0.004, 0.019, 0.019,
        0.333, 0.333, 1, 1, 0.002, 1, 1, 0.004, 0.002, 0.003, 0.019, 1, 1, 0.005, 1,
        0.999, 0.01, 0.002, 0.019, 0.333, 0.001, 0.005, 0.333, 0.002, 0.004, 0.333,
        0.333, 0.333, 0.333, 0, 0.001, 0.019, 0.001, 0.002, 0.019, 0.01, 0.333,
        0.333, 0.001, 0.003, 0.019, 0.003, 0.01, 0.333, 0.019, 0.333, 0.333, 0.01,
        0.01, 0.333, 0.333, 0.333, 0.333, 0.333, 0.333, 0.333],
        [0, 0, 0.999, 0, 0, 0.998, 0.999, 0.997, 0.333, 0, 0, 0.998,
        0, 0, 0.99, 0.998, 0.994, 0.333, 0.999, 0.995, 0.333, 0.998, 0.992, 0.961,
        0.961, 0.333, 0.333, 0, 0, 0.997, 0, 0, 0.992, 0.996, 0.993, 0.961, 0, 0,
        0.99, 0, 0.001, 0.98, 0.996, 0.961, 0.333, 0.998, 0.99, 0.333, 0.995, 0.992,
        0.333, 0.333, 0.333, 0.333, 0.999, 0.998, 0.961, 0.998, 0.996, 0.961, 0.98,
        0.333, 0.333, 0.998, 0.995, 0.961, 0.994, 0.98, 0.333, 0.961, 0.333, 0.333,
        0.98, 0.98, 0.333, 0.333, 0.333, 0.333, 0.333, 0.333, 0.333]
      ],
      "outcome": ["low", "medium", "high"],
      "parents": ["25", "24", "23", "22"],
      "desc": "Risk observed from Communication data"
    }, {
      "lookupcode": "RD25",
```

```

        "level": 2,
        "name": "Efforts on Recruit Co-conspirators",
        "threshold": [0.33, 0.75],
        "source": "Comm",
        "id": "25",
        "subcategory": "NO_DISPLAY",
        "category": "Comm",
        "probabilities": [0.729, 0.258, 0.013],
        "outcome": ["low", "medium", "high"],
        "parents": [],
        "desc": "Mass communications sent by $party_name to influence co-
traders"
    }, {
        "lookupcode": "RD24",
        "level": 2,
        "name": "Efforts on Recruit Victims",
        "threshold": [0.33, 0.75],
        "source": "Comm",
        "id": "24",
        "subcategory": "NO_DISPLAY",
        "category": "Comm",
        "probabilities": [0.725, 0.263, 0.013],
        "outcome": ["low", "medium", "high"],
        "parents": [],
        "desc": "Mass communications sent by $party_name to influence
investors"
    }, {
        "lookupcode": "RD23",
        "level": 2,
        "name": "Recruit Co-conspirators",
        "threshold": [0.33, 0.75],
        "source": "Comm",
        "id": "23",
        "subcategory": "NO_DISPLAY",
        "category": "Comm",
        "probabilities": [0.726, 0.26, 0.013],
        "outcome": ["low", "medium", "high"],
        "parents": [],
        "desc": "Communication sent by $party_name to influence co-
traders"
    }, {
        "lookupcode": "RD22",
        "level": 2,
        "name": "Recruit Victims",
        "threshold": [0.33, 0.75],
        "source": "Comm",
        "id": "22",
        "subcategory": "NO_DISPLAY",
        "category": "Comm",
        "probabilities": [0.733, 0.251, 0.016],
        "outcome": ["low", "medium", "high"],
        "parents": [],
        "desc": "Communication sent by $party_name to influence
investors"
    }, {
        "lookupcode": "RD26",
        "level": 1,
        "name": "Trade",
        "threshold": [0.33, 0.75],
        "source": "Trade",
        "id": "26",
        "subcategory": "NO_DISPLAY",
        "category": "Derived",
        "probabilities": [
            1, 0, 0.002, 0, 0, 0.005, 0.002, 0.004, 0.072],

```



```

        [0, 1, 0.002, 1, 0.999, 0.005, 0.002, 0.004, 0.072],
        [0, 0, 0.996, 0, 0, 0.99, 0.996, 0.991, 0.855]
    ],
    "outcome": ["low", "medium", "high"],
    "parents": ["16", "21"],
    "desc": "Risk observed from Trade data"
}, {
    "lookupcode": "RD16",
    "level": 2,
    "name": "Pump",
    "threshold": [0.33, 0.75],
    "source": "Trade",
    "id": "16",
    "subcategory": "NO_DISPLAY",
    "category": "Trade",
    "probabilities": [0.721, 0.265, 0.015],
    "outcome": ["low", "medium", "high"],
    "parents": [],
    "desc": "Pump in progress in the ticker"
}, {
    "lookupcode": "RD21",
    "level": 2,
    "name": "Dump",
    "threshold": [0.33, 0.75],
    "source": "Trade",
    "id": "21",
    "subcategory": "NO_DISPLAY",
    "category": "Trade",
    "probabilities": [0.729, 0.257, 0.014],
    "outcome": ["low", "medium", "high"],
    "parents": [],
    "desc": "Dump in progress in the ticker"
}],
"edges": [{
    "source": "27",
    "target": "15"
}, {
    "source": "25",
    "target": "27"
}, {
    "source": "24",
    "target": "27"
}, {
    "source": "23",
    "target": "27"
}, {
    "source": "22",
    "target": "27"
}, {
    "source": "26",
    "target": "15"
}, {
    "source": "16",
    "target": "26"
}, {
    "source": "21",
    "target": "26"
}
}],
"toBeScoredData": {
    "ticker": "PDZ",
    "nodes": [{
        "id": "22",
        "value": 1
    }]
}

```

```
}  
}
```

The result is a string in JSON format that contains the outcome of the inference engine. The following is an example of a response.

```
{  
  "results": [{  
    "score": 1,  
    "id": "33"  
  }, {  
    "score": 0.9,  
    "id": "34"  
  }, {  
    "score": 0.7135125,  
    "id": "35"  
  }, {  
    "score": 0.15500289242473736,  
    "id": "36"  
  }, {  
    "score": 0,  
    "id": "37"  
  }, {  
    "score": 0.75,  
    "id": "15"  
  }, {  
    "score": 1,  
    "id": "26"  
  }, {  
    "score": 0.33,  
    "id": "27"  
  }, {  
    "score": 0.9,  
    "id": "30"  
  }, {  
    "score": 0,  
    "id": "31"  
  }, {  
    "score": 0.887188235294118,  
    "id": "32"  
  }],  
  "alert": {  
    "isAlert": true,  
    "score": 0.75  
  },  
  "status": {  
    "code": 200,  
    "message": "success"  
  }  
}
```

The `isAlert` field gives the outcome of whether an alert needs to be created.

The `score` field contains the alert score that is consolidated from the individual risk indicator scores.

The `status` field indicates the overall status of the inference.

The inference engine returns whether the inputs are risky enough to cause an alert. Whether a new alert needs to be created or an existing alert needs to be updated depends on the use case implementation.

The inference engine does not consider new alert or updating an existing alert.

Chapter 9. Indexing and searching

IBM Surveillance Insight for Financial Services provides indexing capabilities for analytics results.

Surveillance Insight uses IBM Solr for indexing and searching features. The search capabilities allow users to search for any kind of communication (email, chat, or voice message), by using keywords or other attributes. Indexing is performed on the content of each communication regardless of the channels and on the results of the analytics.

Indexing is performed on three objects:

- Alert
- E-Comm (includes email, chat, and voice)
- Employee

Employee data is indexed as a one-time initial load. Updates to employee data is not relayed into the indexing system in this release.

E-Comm content and the analytic results are indexed. The email body, collated chat content, and all utterances of voice speech are indexed.

When an alert is created, certain attributes of the alert are also indexed. Updates to an alert, such as adding new evidence to an existing alert is updated in Solr.

The global search feature is sourced from the indexing system.

Solr schema for indexing content

Table 18: Solr schema for indexing content				
Field Name	Field Type	Stored	Indexed	multiValued
commid	String	Y	Y	N
commstarttime	Date	Y	Y	N
commendtime	Date	Y	Y	N
commtype	String	Y	Y	N
commsubject	text_general	Y	Y	N
commtext	text_general	N	N	N
sourcereferenceid	String	Y	Y	N
comminitiatorname	text_general	Y	Y	N
comminitiatorid	String	Y	Y	N
comminitiatorcontact	text_general	Y	Y	N
commparticipantsid	String	Y	Y	Y
commparticipantsname	text_general	Y	Y	Y
commparticipantscontact	text_general	Y	Y	Y
commtags	String	Y	Y	Y

Table 18: Solr schema for indexing content (continued)

Field Name	Field Type	Stored	Indexed	multiValued
commdescription	text_general	Y	Y	N
commchannel	text_general	Y	Y	N
*_score	Float	Y	Y	N
*_keywords	text_general	Y	Y	Y
classification	String	Y	Y	Y
entity	text_general	Y	Y	Y
evidenceids	text_general	Y	Y	Y
doctype	text_general	Y	Y	N
alertid	text_general	Y	Y	Y
alerttype	text_general	Y	Y	N
alertstatus	text_general	Y	Y	N
assetclass	text_general	Y	Y	N
ticker	text_general	Y	Y	N
participants	text_general	Y	Y	Y
tradernames	text_general	Y	Y	Y
alertevidenceids	text_general	Y	Y	Y
id	text_general	Y	Y	N
activealerts	int	Y	Y	N
employeeid	string	Y	Y	N
description	string	Y	Y	N
name	text_general	Y	Y	N
partyid	String	Y	Y	N
pastviolations	int	Y	Y	N
city	text_general	Y	Y	N
state	text_general	Y	Y	N
role	text_general	Y	Y	N
riskrating	float	Y	Y	N
*_dt	date	Y	Y	N

Chapter 10. Conduct Surveillance

The Conduct Surveillance piece of IBM Surveillance Insight for Financial Services provides the components to build a solution that analysis customer complaints and identifies trends in the complaints with respect to the domain-specific theme of the complaint, the product, and sub-product to which the complaint corresponds, the geography, and the age of the complaining customer.

The key components of the conduct surveillance solution are:

- The Analysis Pipeline
- Trend Analysis Component
- Complaints Dashboard and Supporting Data Services



Figure 42: Conduct Surveillance workflow

The raw data is analyzed through a pipeline of machine learning models that output the features of each complaint, such as the theme, sub-theme, product, and sub-product, to which the complaint belongs. These features, along with the customer age and geography, are then fed as inputs to the trend analysis component that identifies trends across the various combinations of these features. The results of the trend analysis are published to a web-based dashboard for the compliance officers to review.

Raw data schema and ingestion

Raw complaints data is typically in CSV format. The schema of this data is not restricted by IBM Surveillance Insight for Financial Services. The analysis pipeline implementation must provide a schema transformation stage to map the input fields to the complaints features schema that is expected by the trend analysis component.

The sample analysis pipeline that is provided with IBM Surveillance Insight for Financial Services uses a specific schema.

The raw data is expected to be loaded into the HDFS for the analysis pipeline to consume as a Spark data set. To leverage the pipeline stages and API that are provided by Surveillance Insight, the data must be loaded into Surveillance Insight and read as a Spark data set. However, this is not a constraint if a non-Spark based pipeline implementation is the preferred way to implement the pipeline. As long as the complaints features are generated in HDFS in the required schema, the pipeline can be implemented using other technologies.

Analysis pipeline

The analysis pipeline processes complaint text through a series of machine learning models that analyze the textual content and identify whether it is a complaint, what keywords of interest are present in the complaint, and what theme (or category) of interest the complaint belongs to.

Input to the pipeline is raw data and the output is a CSV file that contains the complaint features. The schema or the complaint features are detailed in the complaint features section.

IBM Surveillance Insight for Financial Services also provides an API, that is a wrapper for the Spark ML APIs, to create custom pipelines and a sample implementation of a pipeline that uses the publicly available CFPB data.

Note: IBM Surveillance Insight for Financial Services does not provide machine learning models as part of the product. These models must be created as part of the specific engagement.

Create an analysis pipeline

Complaints pipeline API

IBM Surveillance Insight for Financial Services provides a wrapper API to create Spark pipelines. Typical usage of the API is as follows:

- Create a pipeline:

```
public ComplaintsPipeline(SparkSession spark, HashMap<String,String>
propertiesMap)
```

The propertiesMap is expected to hold the following properties that are read from the `sifs.spark.properties` file:

```
HDFSComplaintsPath=complaints
WatsonNLCThemeClassifierREST=https://gateway.watsonplatform.net/natural-
language-classifier/api/v1/classifiers/<THEME_CLASSIFIER_MODEL_ID>/classify
WatsonNLCComplaintsClassifierREST=https://gateway.watsonplatform.net/
natural-language-classifier/api/v1/classifiers/
<COMPLAINTS_CLASSIFIER_MODEL_ID>/classify
WatsonNLCCredentials=<nlc_username>:<nlc_password>
```

```
WatsonNLUREST=https://gateway.watsonplatform.net/natural-language-
understanding/api/v1/analyze?version=2017-02-27
WatsonNLUModelID=alchemy
WatsonNLUCredentials=<nlu_username>:<nlu_password>
```

```
SolrREST=https://<IP>:8984/solr/complaints/update?commit=true
```

```
ComplaintsRestServiceForTrendAnalysis
ComplaintTrendsREST=https://<IP>:9443/complaintsservices/surveillance/v1/
complaint/createTrend
```

- Add stages to the pipeline:

```
public boolean addStage(Transformer pipelineStage)
```

A pipeline stage is implemented as a Spark Transformer and passed to the API.

- Run the pipeline:

```
public Dataset<Row> run(Dataset<Row> complaintsDS)
```

Calling the run method starts the execution of the pipeline with the configured stages.

For more information about creating transformers by using the Spark API, see the Spark documentation.

Reusable pipeline stages

IBM Surveillance Insight for Financial Services provides the following Spark transformers that you can use as stages in an analysis pipeline:

- ComplaintsNLUStage
 - Provides an implementation that calls the Watson Natural Language Understanding (NLU) REST service to identify entities of interest in the complaint text for each complaint. This implementation expects the NLU model to return the complaint and process the entities by using the naming

convention "Complaint_xxxx" and "Process_yyy", where xxx and yyy are the category and process mapping for the complaint. The implementation also interprets relationships between a category and a process. This is expected to appear in the "relation" part of the Watson NLU service response.

- Input: Complaint features dataset with the raw complaint text and with the IS_COMPLAINT flag set.
- Output: Complaint features dataset with the NLU_Response field set to the response from the NLU service.
- ComplaintClassifierStage
 - Provides an implementation that calls the Watson Natural Language Classifier (NLC) REST service to identify whether a piece of text is a complaint. This implementation breaks the complaint text into chunks of 1024 characters and passes them to the NLC service. The first chunk that qualifies to be a complaint (that is, crosses the configurable ComplaintThreshold parameter), stops the processing of further chunks. This implementation expects the NLC to return a "Complaint" or "Non-Complaint" class.
 - Input: Complaint features dataset with the raw complaint text and the IS_COMPLAINT flag set (see schema section for details). The model that is required to perform the classification is expected to be created and configured as input to this stage. The model is not provided as part of the product.
 - Output: Complaint features dataset with the THEME field set to the response from the NLC service.

Reusable utilities

The following persistence components are available for reuse for implementing a pipeline:

- ComplaintsPersistence: This component provides methods for persisting the complaint features to HDFS and the Complaints table in DB2.
- ComplaintsSolrDataLoader: This component provides a method to persist the complaint features to Solr. This is to enable the search of complaint features through the complaints dashboard.

Sample pipeline implementation

IBM Surveillance Insights product provides a sample implementation of an analysis pipeline that uses the following stages, in order:

1. Schema Adaptor
2. Complaint Classifier
3. Theme Classifier

The sample implementation provides an example of a typical pipeline design for complaints analysis. It contains a schema adaptor that maps the contents of the raw data to the complaint features schema. This step creates a dataset that passes through all the stages in the pipeline. Each stage then completes the necessary fields based on its purpose.

In the case of the sample implementation, the NLU stage populates the results from the Watson NLU service into the NLU_Response field in the incoming dataset and also the Theme and Process fields. The Complaint Classifier runs an NLC model that classifies the text as complaint or non-complaint and sets the IS_COMPLAINT flag in the output dataset.

The persistence step saves the outcome of the pipeline, which is the complaint features dataset, to the HDFS. It also persists part of the response to the complaint table in DB2. The complaint text and some other related information are persisted to Solr. The schemas are detailed in the schema section. The persistence step is not implemented as a pipeline stage.

The header for the raw data used for the sample pipeline is as follows:

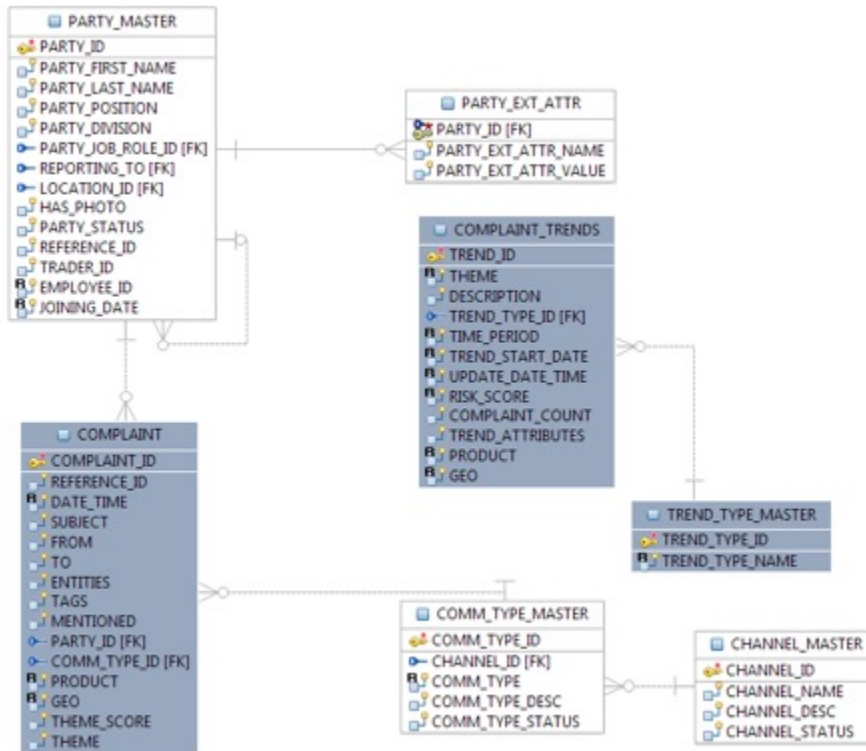
```
COMPLAINT_DATE, PRODUCT, SUB_PRODUCT, COMPLAINT_TEXT, GEO, REFERENCE_ID, CUSTOMER_ID
```

The data itself is taken from the publicly available CFPB database. The sample data contains 3 months of complaints for the Mortgage product.

An NLC model must be created for the Watson Bluemix NLC Service by using the sample CFPB data. The model ID must be configured in the `sifs.spark.properties` file. The same is true with the NLU service. The output results are saved to HDFS in the `complaints_features.csv` file in the `/user/sifsuser/complaints/output` directory.

Data schemas

The following diagram shows the logical data model of the complaints tables in the SIFS database:



- COMPLAINT contains the details of each complaint that is processed by the pipeline. It does not contain the actual complaint text. The text is saved in Solr to enable search through the UI. This table contains the results from the models that processed the complaint (THEME.THEME_SCORE, TAGS,ENTITIES) and some basic information obtained from the raw data.
- COMPLAINT_TRENDS contains the trends that are identified in the complaints and the trend attributes that were used to compute the trend.
- TREND_TYPE_MASTER contains the master data for the trend direction (upwards, downwards, neutral).
- THEME_RISK_MAP contains the risk level for each category.

The rest of the tables in the complaints view of the data model are from the SIFS data model.

Figure 43: Complaints View – SIFS Data model

Complaint features

The complaint features file is persisted to the HDFS as a CSV file. It contains the following data for each record that is processed through the pipeline. Every record may or may not be a complaint.

The `complaints_features.csv` file contains:

- complaint_id — system generated id
- reference_id — id of the complaint in the source file
- complaint_date — date mentioned in the complaint or date email was received
- theme_confidence_score — as returned by NLC model

- theme — as returned by NLC model
- sub-theme — as returned by the NLC model
- product — as returned by the NLU model
- sub-product — as returned by the NLU model
- product_ref — product reference in raw data
- sub-product_ref — sub-product reference in raw data
- geo — geo reference in raw data
- channel — channel reference in raw data
- customer_id — customer id reference in raw data
- customer_age — customer age reference in raw data
- customer_gender — customer gender reference in raw data
- customer_race — customer race reference in raw data
- is_complaint — true/false as returned by the NLC model
- is_complaint_score — as returned by the NLC model
- NLU_Response — as-is response of the NLU model
- Subject — subject of the complaint from raw data
- from_mailid — valid only for email complaints
- to_mailid — valid only for email complaints
- Tone_Analyzer_Response — not in use

Trend analysis

A trend is a change in the normal number of expected complaints. A trend is a consistent upward or downward movement, out of the ordinary range. For an upward trend, each count must be equal to or greater than the previous count. For a downward trend, each count must be less than or equal to the previous count.

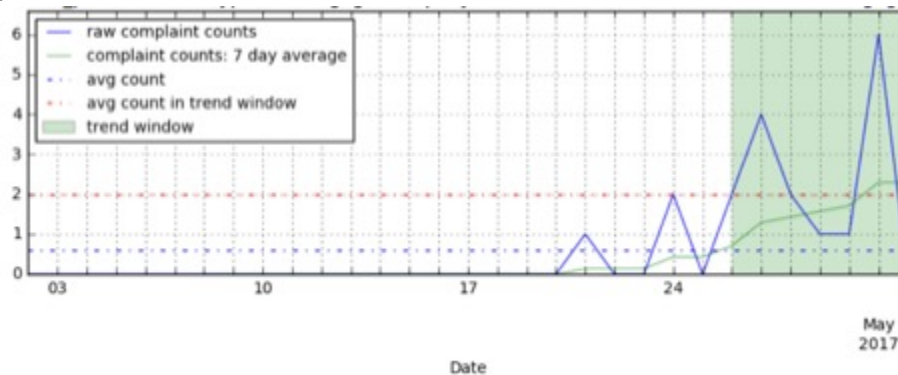


Figure 44: Detecting trends

The diagram shows the number of complaints for one issue for the 30 days before the end date (2017-05-01). The x-axis is the day of the month, and the y axis is the number of complaints per day for the theme. For the first 19 days of the month, there are no complaints for this theme. Then starting on the 19th day, there are increasing amounts of complaints, which indicates a trend.

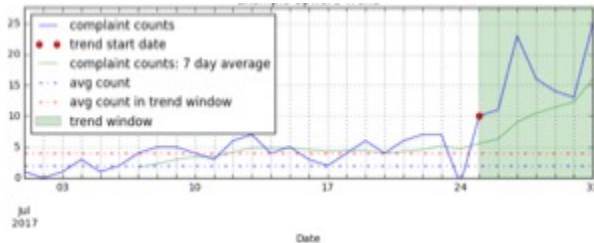
The last 7 days are highlighted in pale green. This is the short window that relates to the recent past: most of the time, the trends that are happening now are the most important. The blue line is the raw number of complaints per day. It is highly variable as the number of complaints vary between weekdays and weekends. A green line is also plotted by taking the average over the previous 7 days for each day. This removes the noise that is caused by weekdays/weekends. The pale red horizontal dotted line is the average number of complaints in the 30-day period. The pale cyan horizontal dotted line is the average

number of complaints in the last 7-day period. This clearly shows that the average number of complaints in the last 7 days is approximately three times the number of complaints over the whole month.

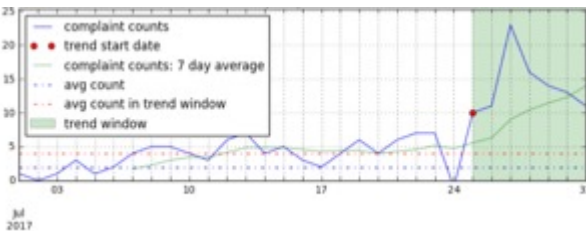
Examples of trends

It can be useful to look at real examples of trends to understand some of the issues about discovering and displaying trends.

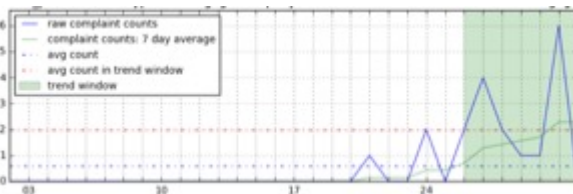
The following trend is noisy and doesn't increase in a smooth fashion:



The following trend is short lived and represents a single spike in complaints:

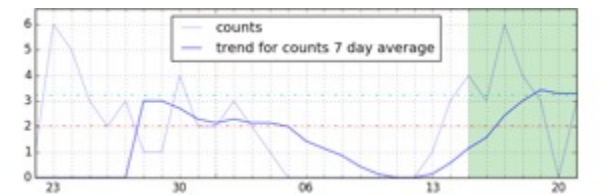


The following is a trend that overall is increasing but might be showing different scores due to the day of the week: fewer people make complaints at the weekends (see the two low scores in the green part on days 30 and 31):



The following is a more complicated trend. The counts are increasing (trending) on the right-hand side but they were also high at the start of the month and low in the middle. The possible causes are:

- This is just a noisy complaint theme and has naturally variability.
- This trend has a monthly periodicity. It might be that people complain at the same time every month, for example, over a few days when monthly bills are sent.



How trend detection works

The first step of the trend detection algorithm groups all the complaints according to their characteristics, counts the daily totals, and then identifies which of these are trending.

All complaints are classified by the Natural Language Classifier and Natural Language Understanding models, and the results are combined with customer data. This gives a number of variables for all

complaints: category, process, product, theme, sub-product, age, zip code, state, gender etc. The exact list of fields available depends on the customer data set. The user can determine which of these available fields are of interest for trend detection by updating the config file.

The trend detection algorithm derives all the combinations of these fields:

1. complaint_category
2. complaint_category + process
3. complaint_category + process + zip code
4. ...

This forms a hierarchy of complaint trends: the top level are very generic but more specific details appear as columns are added.

The trend detection algorithm then counts the complaints in each combination for all values in each field:

1. complaint_category = 'burdensome request', ...
2. complaint_category = 'burdensome request' + process = 'Mortgage Application',
3. complaint_category = 'burdensome request' + process = 'Mortgage Application' + zip code = 9410
4. ...

The complaints are counted for each of these combinations, and evaluated to see if a trend can be found. Complaints are counted over periods of 30, 60, and 90 days. Trend details and trend timelines are viewed in the Surveillance Insights Complaints Analytics Dashboard.

Trend Detection Rule

Given a timeline of complaint counts, it is flagged as trending if:

- There is a general increase or decrease in the last 7 days (short window). See section below for definition of a monotonic increase.
- There is at least one day where the count is unexpectedly large or small given the preceding counts. In technical terms, at least one day has exceeded the expected Poisson distribution score. See section below on Poisson distribution
- There is a clearly defined start and end to the trend within the short window. For the start-date this means there must be at least one day greater than the average count over the long window. For the end-date this means there must be one day with a count greater than the start date (for an increasing trend.)
- The end-count must be greater than the start-count for an increasing trend. The end-count must be less than the start-count for a decreasing trend.

Trend Risk Score

For each timeline that is detected as a trend: the risk score is calculated which expresses how significant each trend is as a percentage score.

Trends may vary widely in terms of size, speed of growth and other factors. It is difficult to find a “one-size fits all” method for scoring when high-level complaints may have tens or hundreds of thousands of complaints and increase slowly, while a low-level trend may only contain tens of complaints but be increasing rapidly. Early detection of growing trends is a key use case so the trend detection algorithm should highlight both.

The scoring method is designed to allow size and speed of increase and decrease and consistency of increase or decrease to all be considered.

The risk score is calculated from 3 components:

1. The number of complaints since the trend was detected: trends with bigger increases or falls in complaints score higher.
2. The gradient of the increase or decrease: trends which increase or fall more steeply score higher.

3. How "monotonic" the fall or increase is. Trends are often noisy i.e. even if there is an overall increase some points are less than the preceding count. A monotonic increase is one where all counts are either greater than or equal to the preceding days complaint count. Trends where the count only goes in one direction score higher because they are more consistent.

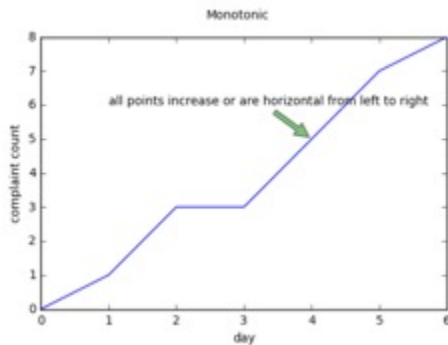


Figure 45: Monotonic trend

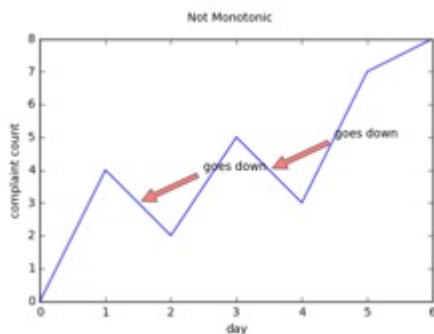


Figure 46: Non-monotonic trend

A score is created from these three factors between 0 and 85% (the maximum). Some tuning may be required for each data set

The risk score is calculated from:

1. Total number of complaints in trend
2. gradient of trend
3. monotone score – does the trend only increase or decrease

Each of these has a weight factor configurable by the user giving the relative importance of each as a fraction adding to 1. For example:

1. complaints weight = 0.5
2. gradient weight = 0.3
3. monotone weight = 0.2

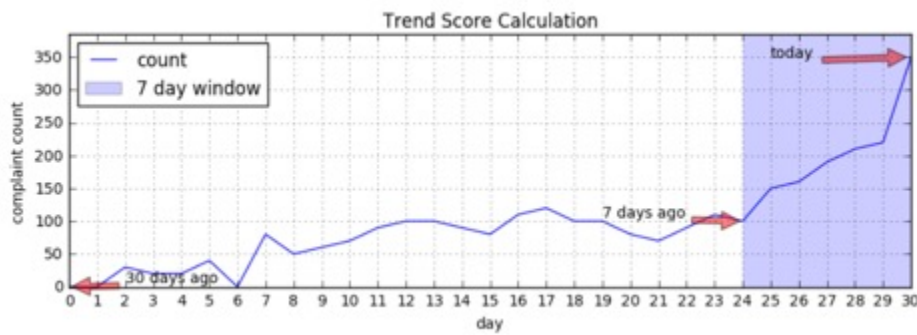
The score is calculated as:

$$\text{Score} = \left(\frac{\log(\text{number of complaints})}{\log(\text{max_count})} \right)^{\text{complaints weight}} * \left(\frac{\log(\text{gradient})}{\log(\text{max_gradient})} \right)^{\text{gradient weight}} * \text{monotone}^{\text{monotone weight}}$$

Where:

- logs are calculated using base 10
- All 3 main terms are raised to the power of the corresponding weight
- The results are multiplied together

The following is an example:



Here a trend has been identified in the last 7 days:

Number of complaints in last 7 days = 1380 : (100 + 150+ 160+ 190+ 210+ 220+ 350)

- Gradient = (final count – initial count)/ 7 days = 350 -100/7 = 35.7
- Monotone score = 100% (all counts from 24th to 30th are greater than preceding day count)

The user has determined these weights:

1. complaints weight = 0.5
2. gradient weight = 0.4
3. monotone weight = 0.1

This means that the total number of complaints is the most important factor, increasing gradient is almost as important, but they don't care so much if a trend is noisy.

The user has also determined these factors:

- For the number of complaints: all trends will be ranked on a scale between 0 and 10,000:

Max complaints = 10,000

- For the gradient: all trends will be ranked on a scale between 0 and 1000. A gradient of 1000 signifies that the fastest growing trend the user has seen or expects to see will grow at a rate of 1000 additional counts per day:

Max gradient = 1000

$$\text{Score} = \left(\frac{\log(\text{number of complaints})}{\log(\text{max_count})} \right) \text{complaints weight} * \left(\frac{\log(\text{gradient})}{\log(\text{max_gradient})} \right) \text{gradient weight} * \text{monotone} \text{monotone weight}$$

$$\text{Score} = \left(\frac{\log(1380)}{\log(10000)} \right)^{0.5} * \left(\frac{\log(35.7)}{\log(1000)} \right)^{0.4} * 1^{0.1}$$

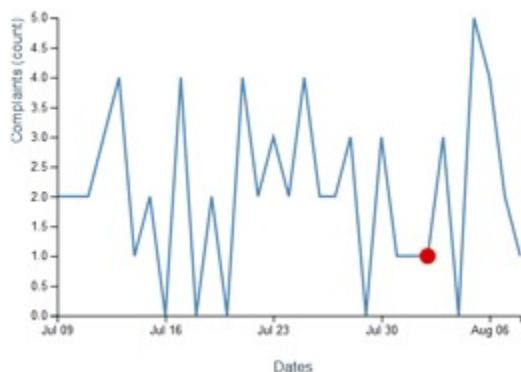
$$\text{Score} = (0.78)^{0.5} * (0.52)^{0.4} * 1^{0.1} = 0.7$$

The score is then scaled to have a maximum of 85%.

Final score = 0.7 * 85% = 60%

Earliest Trend Detection Date

If a trend is detected, the earliest trend detection date is designated as the first day on which the complaints counts exceeds the long-term average. The earliest trend detected date is shown with a red dot in the UI, see below. If the trend continues increasing over a prolonged period the earliest trend detection date is preserved as the first date on which the count exceeded the long-term average.



Trending 7 days and 30 days Calculation

This describes the method for calculating the trending 7 days and trending 30 days scores on the complaints UI. These are displayed in the UI:

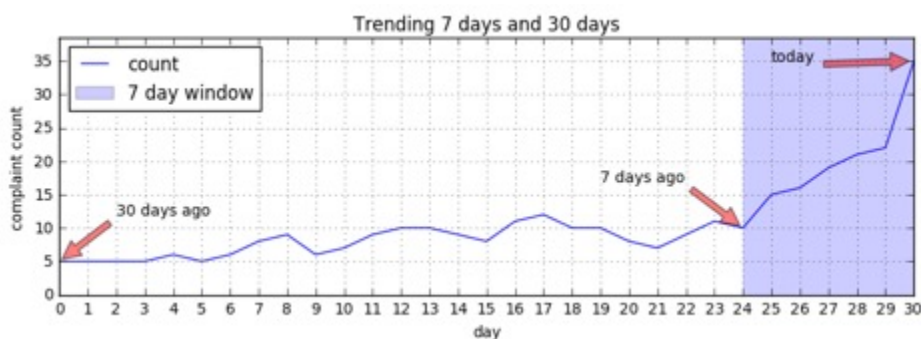
TRENDING LAST 7 DAYS

+0.0%

TRENDING LAST 30 DAYS

-50.0%

This calculates how much the trend has increased over a period as a percentage of the original amount: (increase or decrease / original amount) as a percent.



Trend over the last 7 days:

- amount 'today' = 35
- original amount (7 days ago) = 10
- increase over the last 7 days = $35 - 10 = 25$

Trending last 7 days = $25/10$ = a 250% increase

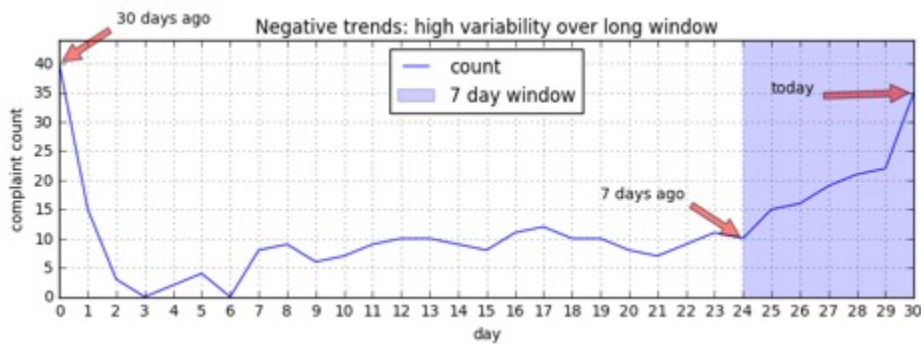
Trend over the last 30 days:

- amount 'today' = 35
- original amount (30 days ago) = 5
- increase over the last 30 days = $35 - 5 = 30$

Trending last 30 days = $30/5$ = a 600% increase

Negative scores

It is possible for the trend detection to find an increasing trend and still have negative or decreasing trend scores. There are two examples below.



Trend over the last 7 days:

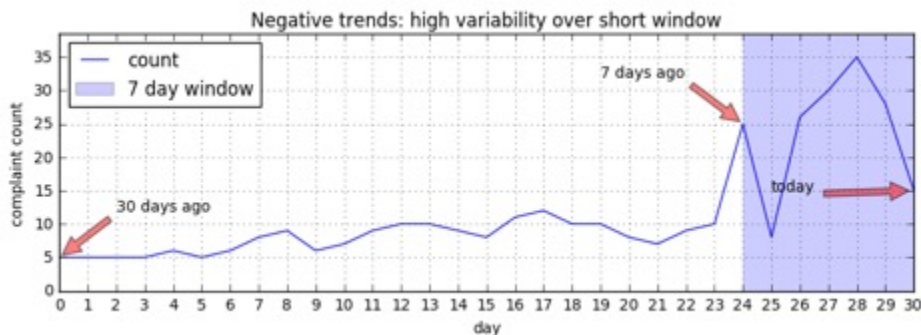
- amount 'today' = 35
- original amount (7 days ago) = 10
- increase over the last 7 days = $35 - 10 = 25$

Trending last 7 days = $25/10 =$ a 250% increase

Trend over the last 30 days:

- amount 'today' = 35
- original amount (30 days ago) = 40
- increase over the last 30 days = $35 - 40 = -5$

Trending last 30 days = $5/40 =$ a -12.50% increase (or a 12.50% decrease)



Trend over the last 7 days:

- amount 'today' = 15
- original amount (7 days ago) = 25
- increase over the last 7 days = $15 - 25 = -10$

Trending last 7 days = $10/25 =$ -40%

Trend over the last 30 days:

- amount 'today' = 15
- original amount (30 days ago) = 5
- increase over the last 30 days = $15 - 5 = 10$

Trending last 30 days = $10/5 =$ 200%

Trend statistics

Trends may vary widely in terms of size, speed of growth and other factors. It is recognized that it is difficult to find a “one-size fits all” method. For example, some of the high-level complaints have tens or hundreds of thousands of complaints and may increase slowly, while a low-level trend may only contain

tens of complaints. Early detection of growing trends is a key use case so the trend detection algorithm should highlight both. Given this, a number of statistics for each trend are calculated so that there is flexibility in updated the trend detection method in the future if required:

<i>Table 19: Trend statistics</i>	
Statistic	Description
count	total number of complaints in the long window
count in trend window	total number of complaints in the short window
mean count	average number of complaints per day for the long window (this includes the days in the short window)
mean count in trend window	average number of complaints per day for the short window
mean ratio	(mean count in the short window) / (mean count in the long window)
trendinglast7days	increase in % over the last 7 days using the raw counts, calculated as: (count on the last day - count on 7th day before this) / count on 7th day before this
trendinglast30days	increase in % over the last 30 days using the raw counts, calculated as: (count on the last day - count on 30th day before this) / count on 30th day before this
Trend type id	Type of trend: 'increasing'=1, 'decreasing'=2, 'no trend'=3
Mk trend	True if there is a trend (i.e. if trend type id = 1 or 2)
Mk score p-value	The confidence score (p-value) that the Mann-Kendall algorithm has discovered a trend: Score < .05 = low, Score < 0.1 = high, Score < 0.001 = very high confidence
poisson_scores_max	The highest Poisson score in the trend window
poisson_above_thresh_count_inc	how many days in the short window had a Poisson score greater than the Poisson threshold for an increasing trend
poisson_above_thresh_count_dec	how many days in the short window had a Poisson score greater than the Poisson threshold for a decreasing trend

Mann-Kendall score

The purpose of the Mann-Kendall score is to statistically assess if there is a monotonic upward or downward trend of the variable of interest over time. A monotonic upward (or downward) trend means that the variable consistently increases (or decreases) through time, but the trend might or might not be linear.

Point-by-point Poisson Model

The Poisson distribution model is a statistical model that is used to evaluate how unusual a point in a time-series is when compared to previous points. It is used in the trend analysis model to help identify sudden increases (or decreases) in the number of complaints seen that may help indicate the presence of a trend.

The Poisson distribution describes the probability of observing a count of some quantity, when many sources have individually low probabilities of contributing to the count. The point-by-point Poisson model uses the previous point in a time series to define the expectation for the current point, and gives us the unlikeliness of a count of complaints given the previous count.

Complaints dashboard

The Complaints dashboard is a web-based application that shows the complaints trend analysis results.

You access the Complaints dashboard by entering the following URL in your web browser:

`https://hostname:port/ui.complaintsdashboard/`

You must enter your log in credentials.

The Complaints dashboard shows the key trends in the analyzed complaints in a table. It also shows a bubble chart with further details on the trends. Each bubble corresponds to a single trend in the trend table. The size of the bubble corresponds to the volume of complaints that contributed to the trend. The color of the bubble corresponds to the level of risk that is associated with the complaint.



Figure 47: Complaints dashboard

Click a row in the trend table or a bubble in the bubble chart to display the **Theme Details** page.

Theme Details

The **Theme Details** page shows the complaints that contributed to the trend. It also shows a trend analysis chart with the complaint count and the date on the Y and X axis. Click one of the complaints in the table to show the evidence details page.

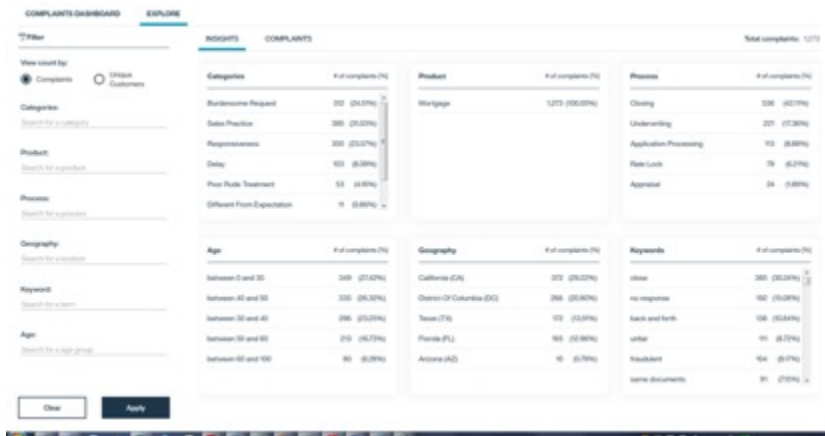


Figure 50: Explore view

The side pane displays the various filter parameters that you can use to filter the insights. The complaints part of this view shows the list of actual complaints.

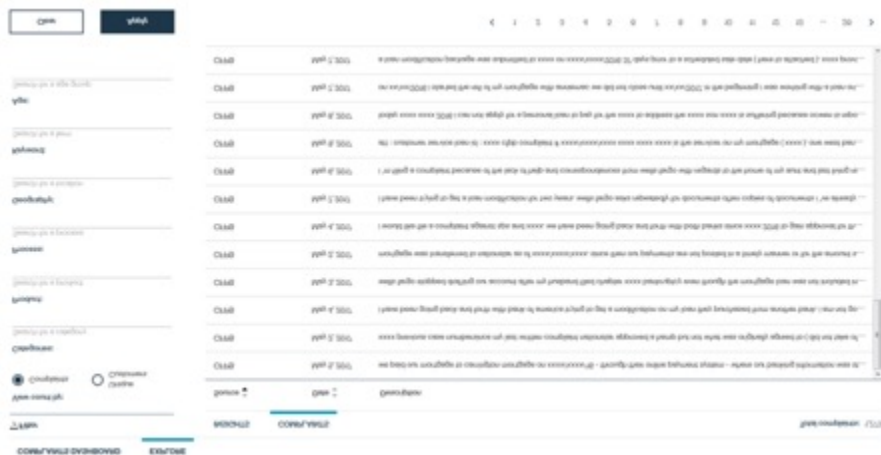


Figure 51: Explore view

Click a complaint from this list to display the complaint details similar to the one show in the **Evidence Details** page.

Complaints data model

The following diagram shows the logical data model of the complaints tables in the SIFS database.

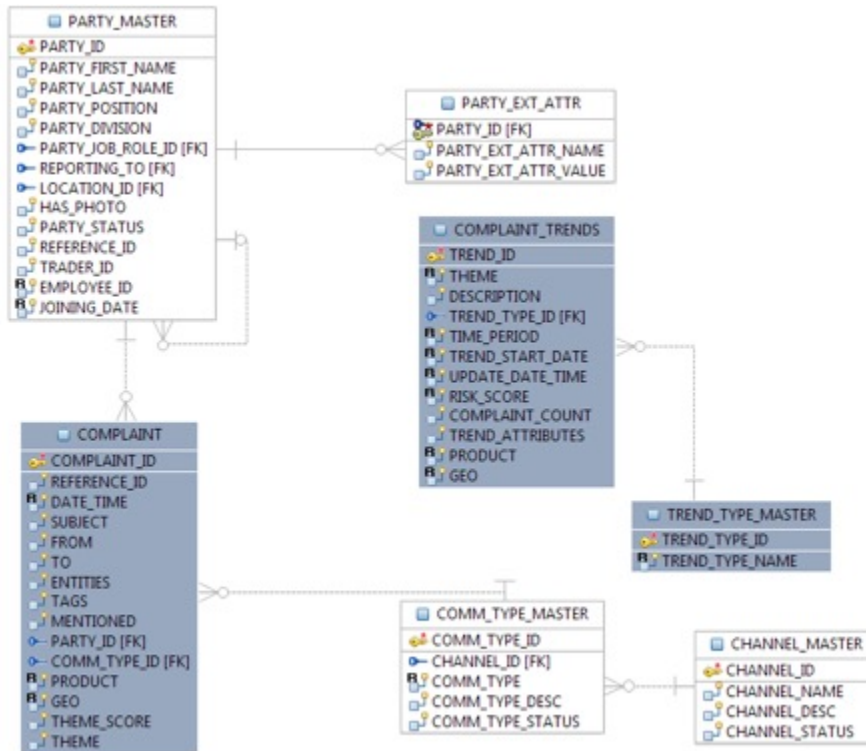


Figure 52: Complaints data model

The COMPLAINT table contains the details of each complaint that is processed by the pipeline. The table does not contain the actual complaint text. The text is saved in Solr to enable search through the dashboard. The table contains the results from the models that processed the complaint (THEME.THEME_SCORE, TAGS, ENTITIES) and some basic information that is obtained from the raw data.

The COMPLAINT_TRENDS table contains the trends that are identified in the complaints and the trend attributes that were used to compute the trend.

The TREND_TYPE_MASTER table contains the master data for the trend direction (upwards, downwards, neutral).

The rest of the tables in the complaints view of the data model are from the SIFS data model.

Complaint features

The complaint features file is persisted to the Hadoop file system (HDFS).

The complaint feature file is in CSV format and it contains the following data for each record that was processed through the pipeline. Note that every record might or might not be a complaint.

- complaint_id: system generated ID
- reference_id: ID of the complaint in the source file
- complaint_date: date that is mentioned in the complaint or the date that the email was received
- theme_confidence_score: as returned by NLC model
- theme: as returned by NLC model
- sub-theme: as returned by the NLC model
- product: as returned by the NLU model
- sub-product: as returned by the NLU model
- product_ref: product reference in raw data

- sub-product_ref: sub-product reference in raw data
- geo: geo reference in raw data
- channel: channel reference in raw data
- customer_id: customer ID reference in raw data
- customer_age: customer age reference in raw data
- customer_gender: customer gender reference in raw data
- customer_race: customer race reference in raw data
- is_complaint: true or false, as returned by the NLC model
- is_complaint_score: as returned by the NLC model
- NLU_Response: as-is response of the NLU model
- Subject: subject of the complaint from raw data
- from_mailid: valid only for email complaints
- to_mailid: valid only for email complaints
- Tone_Analyzer_Response: as-is response of the tone analyzer service

Solr data model for complaints

The complaints data in Solr is loaded into the complaints core by using a dynamic schema.

The schema contains the following fields for each complaint:

- customer_id_s
- subject_s
- theme_s
- complaint_text_s
- channel_s
- reference_id_s
- customer_age_i
- geo_ref_s
- complaint_date_dt
- id_s
- product_ref_s

Chapter 11. Health Check User Interface

Clicking the health check tab in the SIFS dashboard displays the Health Check page.

Health Check tabs

In this release there are two use cases.

Ecomm

In this use case there are three dashboards:

- Ingestion
- Processing
- Inference

Voice

In this use case there are six dashboards:

- Ingestion services
- Data services
- Streams PCAP
- Streams 3rd Party
- Services
- Streams WAV adaptor

By default, the **Ecomm – Ingestion** dashboard has a default date range of one year.

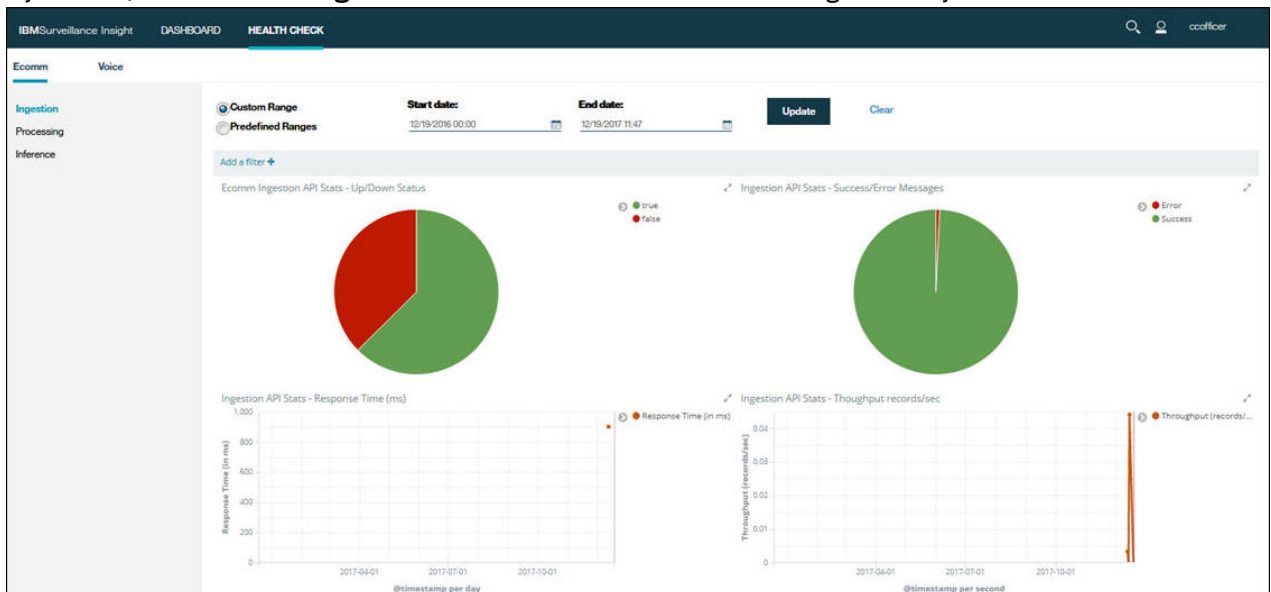


Figure 53: Default Health Check UI

Date ranges

There are two types of date ranges, Custom Range and Predefined Ranges.

Custom Range

The **Custom Range** option contains a **Start date** and an **End date**. By default, the **Start date** is the current day and the **End date** is one year after the **Start date**. You can search for any **Start date** or **End date**.

Predefined Ranges

The **Predefined Ranges** option is used to select dates quickly. From the drop-down lists for **Start date** and **End date**, you can select from a number of options: **Today**, **Yesterday**, **This Week**, **This Month**, and **This Year**.

After you select a date range from these options, click  to display data for the date range in the dashboards.

Health Check dashboards

There are two types of dashboards: Ecomm and Voice. Both display data based the selected date ranges.

Ecomm dashboards

There are three Ecomm dashboards: Ingestion, Processing, and Inference.

Ingestion

The **Ingestion** dashboard has four visualizations:

- **Ecomm Ingestion API Stats – Up/Down Status.** This chart displays the **Up/Down status** ratio for the Ecomm ingestion service during a specified time interval.
- **Ingestion API Stats – Success/Error Messages.** This chart displays the **Success/Error Messages** ratio for a specified time interval.
- **Ingestion API Stats – Response Time (ms).** This chart shows the average response time of ingested records per millisecond for a specified time interval.
- **Ingestion API Stats – Throughput records /sec.** This chart shows the average number of records that are ingested per second.

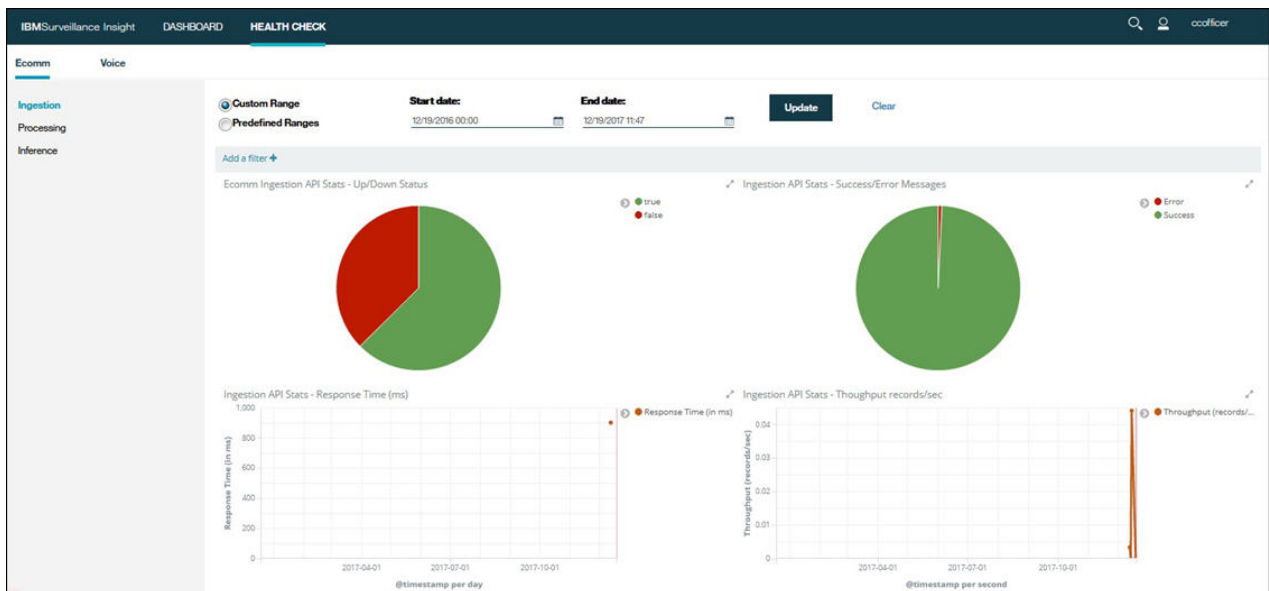


Figure 54: Ecomm Ingestion dashboard

Processing

The **Processing** dashboard has eight visualizations.

- **Ecomm Spark Job – AnalyzeComm.** This chart shows the **Up/Down status** ratio during a specified time interval. It runs for 2 to 5 minutes a day and has a higher percentage of Down status.
- **Ecomm Spark Job – Process Communication.** This chart shows the **Up/Down status** ratio during a specified time interval. It runs all the time.
- **Ecomm Spark Job – Profile Aggregator.** This chart shows the **Up/Down status** ratio during a specified time interval. It runs for 2 to 5 minutes a day and has a higher percentage of Down status.
- **Ecomm Spark Job – Persist Email.** This chart shows the **Up/Down status** ratio during a specified time interval. It runs all the time.
- **Ecomm Processing – No of Communication Processed.** This metric is the total number of emails that are processed by a **Persist Email** spark job for a specified time interval.
- **Ecomm Dashboard – Ingestion vs Processing.** This chart shows the real-time count of Ingestion Throughput & Processing Throughput, Backlog (Ingestion – Processing) counts for a specified time interval. In the chart:
 - Green represents Ecomm Ingestion.
 - Blue represents Ecomm Processing.
 - Orange represents backlog.
- **Ecomm Processing – GCID Errors.** This metric is the number of GCID errors for a specified time interval.
- **Ecomm Processing – parse email data errors.** This metric is the number of parse email errors for a specified time interval.

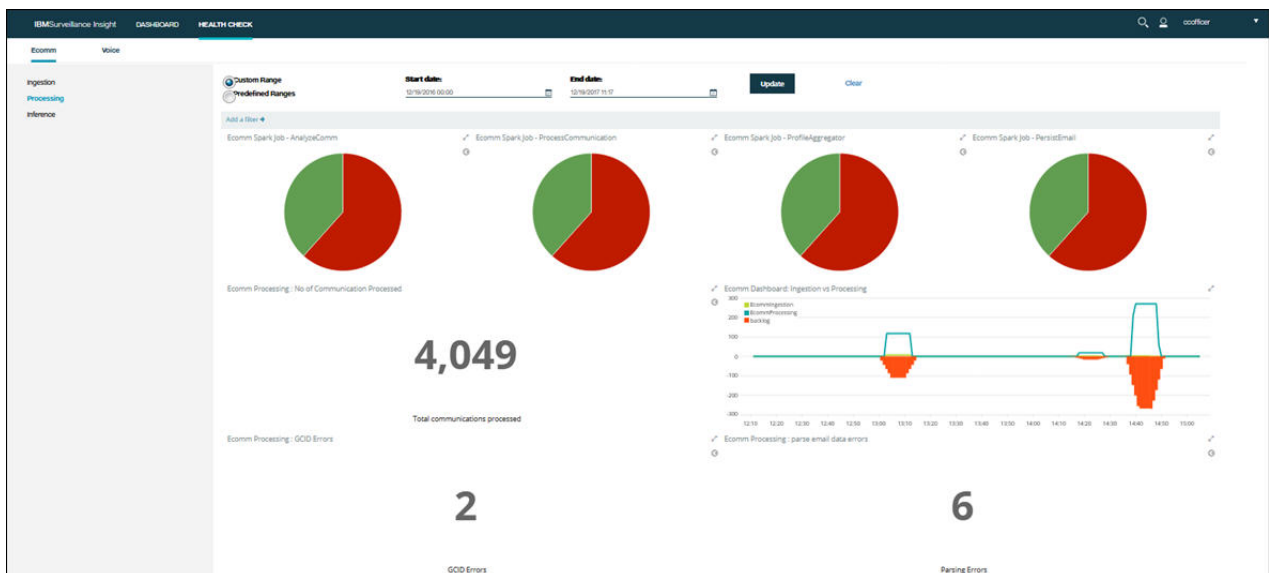


Figure 55: Ecomm Processing dashboard

Inference

The **Inference** dashboard has three visualizations.

- **Ecomm Processing – Inference Alerts bar chart.** This chart shows the number of inference alerts for a specified time interval.
- **Ecomm Processing – No of Communications Processed in Inference.** This metric is the sum of all communications that were processed in inference for a specified time interval.
- **Ecomm Processing – Inference Last Run Time.** This metric is the last run time of an inference spark job. The inference spark job generally runs once nightly in a 24-hour period.

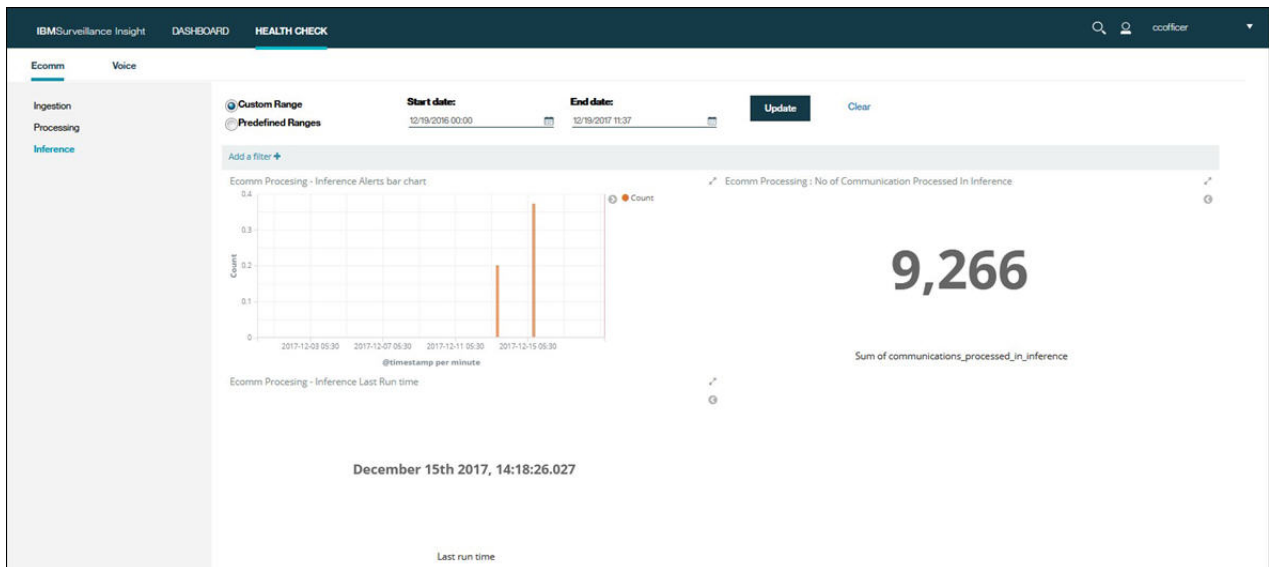


Figure 56: Ecomm Inference dashboard

Voice dashboards

There are five Voice dashboards: Ingestion Services, Data Services, Streams PCAP, Stream 3rd Party Services, and Stream WAV Adaptor.

Ingestion Services

The **Ingestion Services** dashboard has four visualizations:

- **Voice Ingestion API Stats – Uptime Status.** This chart displays the **Up/Down status** ratio for the Voice ingestion service during a specified time interval.
- **Voice Ingestion API Stats – Success/Error Messages.** This chart displays the **Success/Error Messages** ratio for a specified time interval.
- **Voice Ingestion API Stats – Response Time(in ms).** This chart shows the average response time of ingested records per millisecond for a specified time interval.
- **Voice Ingestion API Stats – Throughput Records /sec.** This chart shows the average number of records that are ingested per second.

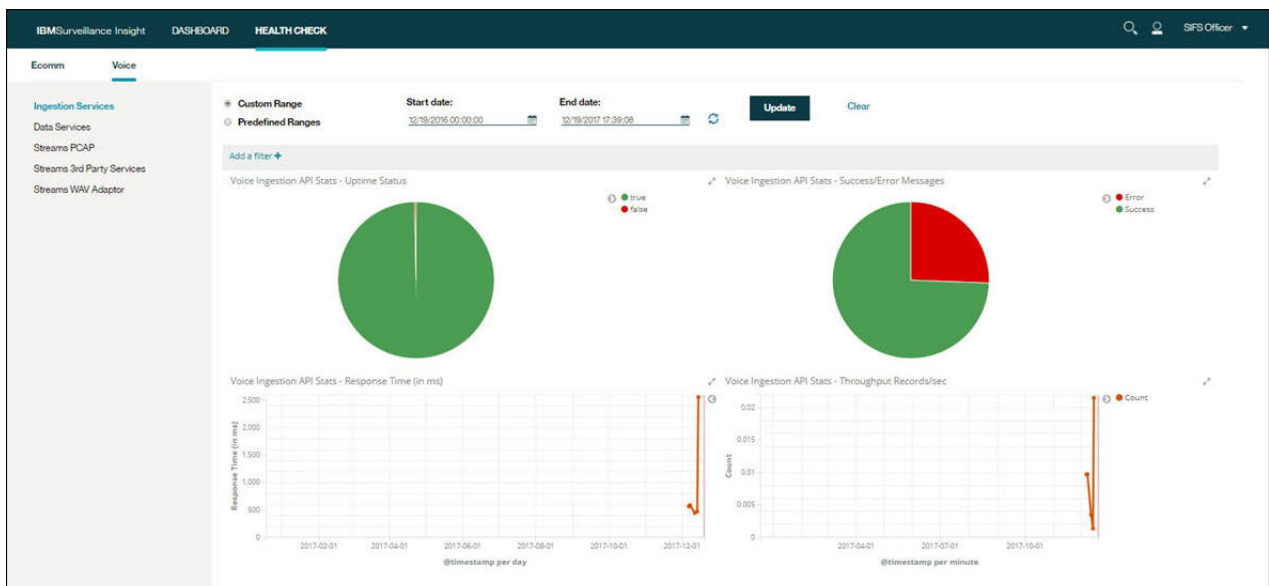


Figure 57: Voice Ingestion Services dashboard

Data Services

The **Data Services** dashboard has five visualizations.

- **Voice Data Services API Stats – Up/Down Status.** This chart shows the **Up/Down status** ratio during a specified time interval.
- **Voice Data Services API Stats – Success/Error Messages.** This chart shows the **Up/Down status** ratio during a specified time interval.
- **Voice Data Services API Stats – Response Time (in ms) for Import service.** This chart shows the average response time of ingested records of import service per millisecond for a specified time interval.
- **Voice Data Services API Stats – Response Time (in ms) for Export service.** This chart shows the average response time of ingested records of export service per millisecond for a specified time interval.
- **Ingestion API Stats – Throughput records /sec.** This chart shows the average number of records that are ingested per second.

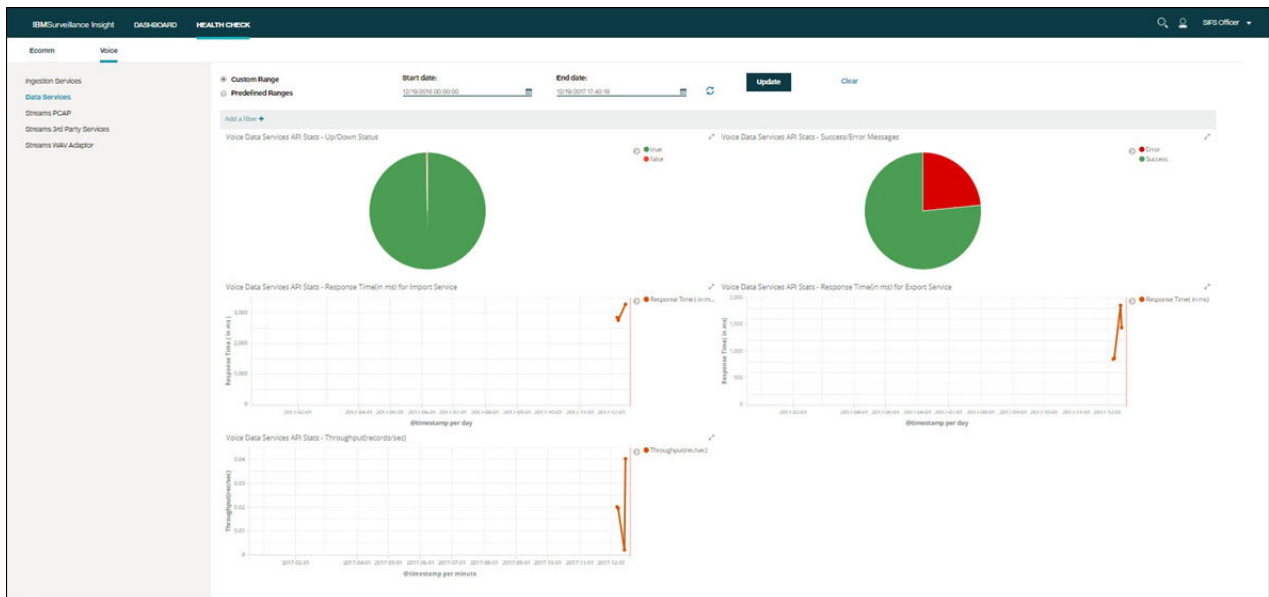


Figure 58: Voice Data Services dashboard

Streams PCAP

The **Streams PCAP** dashboard has nine visualizations:

- **ConfigureCallProgressEvent Streamjob Up/Down Status.** This chart displays the **Up/Down status** ratio for the ConfigureCallProgressEvent Stream job service during a specified time interval.
- **IPC Streamjob Up/Down status.** This chart displays the **Up/Down status** ratio for the IPC Stream job service during a specified time interval.
- **Watson Streamjob Up/Down Status.** This chart displays the **Up/Down status** ratio for the Watson Stream job service during a specified time interval.
- **Main Streamjob Up/Down Status.** This chart displays the **Up/Down status** ratio for the Main Stream job service during a specified time interval.
- **Voice Steam PCAP Stats – PCAP voice calls (avg processing time).** This chart displays the average processing time for voice calls per second for a specified time interval.
- **Voice Stream PCAP Stats – no of voice calls ingested (throughput).** This chart displays the average number of voice calls ingested during a specified time interval.
- **Voice Stream PCAP Stats – Total Call duration.** This metric is the total duration of all voice calls during a specified time interval.

- **Voice Stream PCAP Stats – Stream Errors.** This metric is the total number of PCAP stream errors during a specified time interval.
- **Voice Stream PCAP Stats – PCAP voice export error.** This metric is the total number of PCAP voice export errors during a specified time interval.

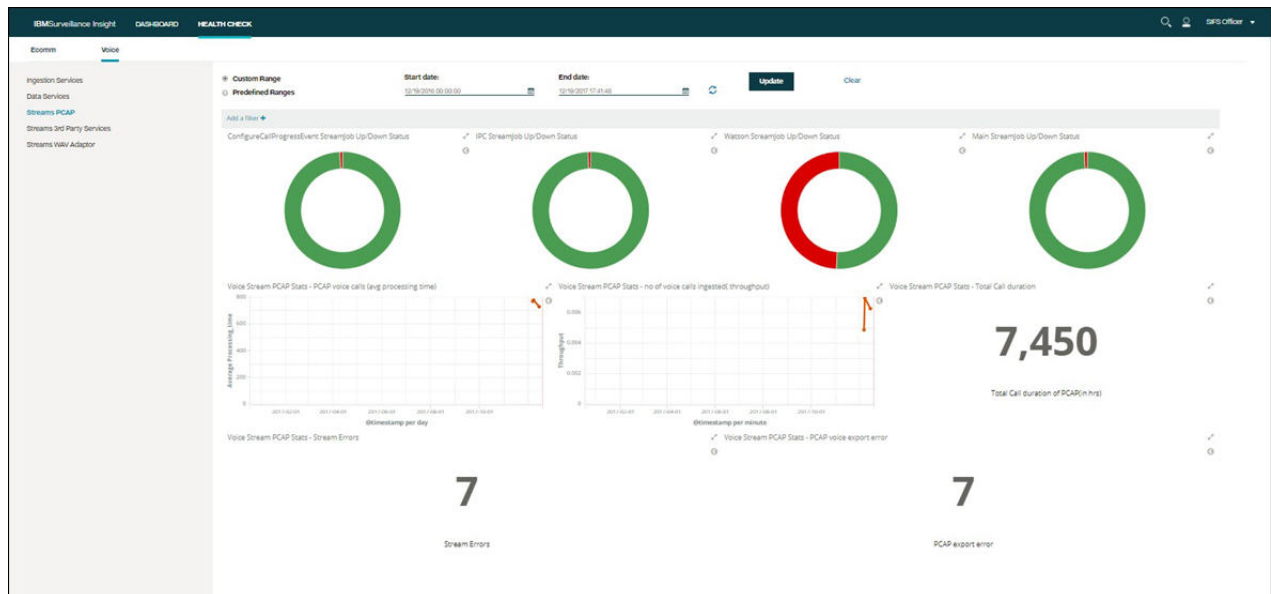


Figure 59: Voice Stream PCAP dashboard

Stream 3rd Party Services

The **Stream 3rd Party Services** dashboard has eight visualizations.

- **ConfigureCallProgressEvent StreamJob Up/Down Status.** This chart shows the **Up/Down status** ratio during a specified time interval.
- **Voice Stream ThirdParty API Stats – communicationHistory Response_time.** This chart shows the average response time of CommunicationHistory records of 3rd party service during a specified time interval.
- **Voice Stream ThirdParty API Stats – createSession Response_time.** This chart shows the average response time of CreateSession records of 3rd party service during a specified time interval.
- **Voice Stream ThirdParty API Stats –loginSession Response_time.** This chart shows the average response time of records of 3rd party service during a specified time interval.
- **Voice Stream ThirdParty API Stats –Response_time.** This chart shows the average response time of all records of 3rd party service during a specified time interval.
- **Voice Stream ThirdParty API Stats – CreateSession error/login Session error/ CommunicationHistory error.** This chart shows all three types of errors during a specified time interval.
- **Voice Stream ThirdParty API Stats – Success/Error.** This chart shows the success to error ratio for a specified time interval.
- **Voice Stream ThirdParty API Stats – Total Metadata Errors.** This metric is the total errors found with type Metadata in a specified date range.

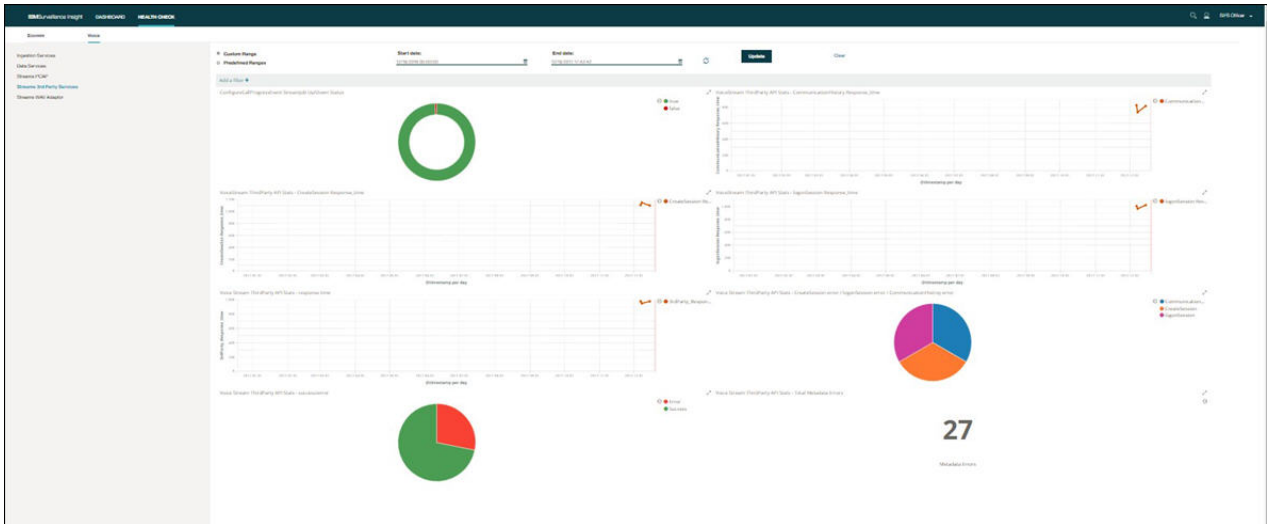


Figure 60: Voice Stream 3rd Party Services dashboard

Stream WAV Adaptor

The **Stream WAV Adaptor** dashboard has six visualizations.

- **WAVAdaptor StreamJob Up/Down Status.** This chart shows the **Up/Down status** ratio during a specified time interval.
- **Voice Stream WAVAdaptor Stats – voice calls (avg processing time).** This chart shows the average processing time of voice calls per second during a specified time interval.
- **Voice Stream WAVAdaptor Stats - no of voice calls ingested(Throughput).** This chart shows the number of voice calls in 10mins/hr/day interval based on the time frame length.
- **Voice Stream WAVAdaptor Stats – Total Call Duration.** This metric is the total call duration in seconds for a specified date range.
- **Voice Stream WAVAdaptor Stats – voice export error.** This metric is the total numbers total number of voice export errors for a specified date range.
- **Voice Stream WAVAdaptor Stats – stream error.** This metric is the total number of stream errors for a specified date range.

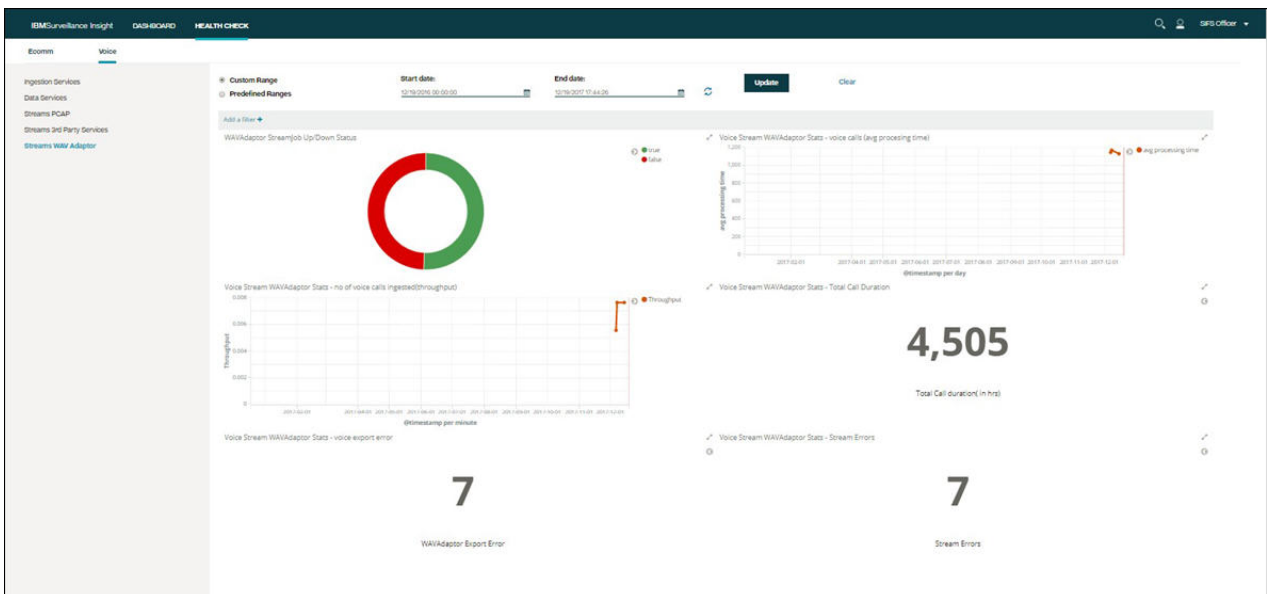


Figure 61: Voice Stream WAV Adaptor dashboard

Chapter 12. Troubleshooting

This section provides troubleshooting information.

CDISI5060E No default Java found

You receive the following message: CDISI5060E No default Java found.

To resolve this error, install Java version 1.6 or later and set it as the default version. Then, try the command again.

Update the PATH variable in your .bashrc file to point to the JAVA location.

```
export PATH=<location of jre/bin directory>:$PATH
```

CDISI3059W You may be running a firewall which may prevent communication between the cluster hosts

You receive the following error message: Warning: CDISI3059W You may be running a firewall which may prevent communication between the cluster hosts

To resolve this, run the following command to stop the firewall service, and try the command again:

```
systemctl stop firewalld
```

For more information, see [Firewall configuration guidelines for IBM Streams](#).

CDISI5070E The perl-XML-Simple software dependency is not installed

You receive the following error message: Error: CDISI5070E The perl-XML-Simple software dependency is not installed

To resolve this error, install the following RPMs as the root user:

- perl-XML-Namespacesupport-1.11-10.el7.noarch.rpm
- perl-XML-SAX-0.99-9.el7.noarch.rpm
- perl-XML-SAX-Base-1.08-7.el7.noarch.rpm
- perl-XML-Simple-2.20-5.el7.noarch.rpm

Use the following command to install each RPM:

```
rpm -ivh rpm_name
```

Appendix A. Accessibility features

Accessibility features help users who have a physical disability, such as restricted mobility or limited vision, to use information technology products.

For information about the commitment that IBM has to accessibility, see the [IBM Accessibility Center](http://www.ibm.com/able) (www.ibm.com/able).

HTML documentation has accessibility features. PDF documents are supplemental and, as such, include no added accessibility features.

Notices

This information was developed for products and services offered worldwide.

This material may be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service. This document may describe products, services, or features that are not included in the Program or license entitlement that you have purchased.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Software Group
Attention: Licensing
3755 Riverside Dr.
Ottawa, ON
K1V 1B7
Canada

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

This Software Offering does not use cookies or other technologies to collect personally identifiable information.

IBM Surveillance Insight for Financial Services includes Brat (v 1.3) from the following source and licensed under the following agreement:

- http://weaver.nplab.org/~brat/releases/brat-v1.3_Crunchy_Frog.tar.gz
- <https://creativecommons.org/licenses/by-sa/3.0/legalcode>

IBM Surveillance Insight for Financial Services includes spaCy Models (v 1.2.0) from the following source and licensed under the following agreement:

- [https://github.com/explosion/spacy-models\(en_core_web_sm 1.2.0\)](https://github.com/explosion/spacy-models(en_core_web_sm 1.2.0))
- <https://creativecommons.org/licenses/by-sa/3.0/legalcode>

Trademarks

IBM, the IBM logo and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "[Copyright and trademark information](http://www.ibm.com/legal/copytrade.shtml)" at www.ibm.com/legal/copytrade.shtml.

Index

A

accessibility [115](#)
architecture [2](#)

B

bulk execution detection [15](#)
bulk order detection [15](#)

C

chat data [37](#)
classifier library [74](#)
cognitive analysis and reasoning component [2](#)
compliance workbench [2](#)
concept mapper [69](#)
concept mapper library [71](#)

D

data ingestion
 e-comm data [38](#)
data store [2](#)
document classifier [69](#)

E

e-comm data ingestion [38](#)
e-comm surveillance [37](#)
email data [37](#)
emotion detection [69](#)
emotion detection library [69](#)
end of day schema [26](#)
event data schema [28](#)
event schema [27](#)
execution schema [20](#)

G

guidelines for new models [34](#)

H

high order cancellation [15](#)

I

indexing [85](#)
inference engine
 risk model [79](#)
 running the inference engine [80](#)
installation [3](#)
introduction [v](#)

M

market reference schema [27](#)
models
 guidelines [34](#)

N

natural language libraries
 classifier [74](#)
 concept mapper [71](#)
 emotion detection [69](#)

O

off-market
 use case [31](#)
order schema [22](#)
overview [1](#)

P

prerequisites [3](#)
price trend [15](#)
pump and dump
 use case [28](#)

Q

quote schema [24](#)

R

real-time analytics [2](#)
risk event schema [27](#)
risk model
 inference engine [79](#)
running the inference engine [80](#)

S

schemas
 end of day [26](#)
 event [27](#)
 event data [28](#)
 execution [20](#)
 market reference [27](#)
 order [22](#)
 quote [24](#)
 risk event [27](#)
 ticker price [20](#)
 trade [25](#)
 trade evidence [27](#)
 transaction [27](#)
 voice surveillance metadata [54](#)
searching [85](#)

solution architecture [2](#)
spoofing
 use case [29](#)

T

ticker price schema [20](#)
trade evidence schema [27](#)
trade schema [25](#)
trade surveillance component [15](#)
trade surveillance toolkit [15](#)
transaction schema [27](#)

U

use case
 off-market [31](#)
 pump and dump [28](#)
 spoofing [29](#)

V

voice surveillance [49](#)
voice surveillance schema [54](#)

