Designing Alert Correlation Engine using Mutual Information

Final Project Report

CS-410 – Fall 2021
Team Alpha
Abhijit Bhadra
abhadra2@illinois.edu
Sanjeev Kumar
sanjeev5@illinois.edu
Swati Nanda
swatin2@illinois.edu

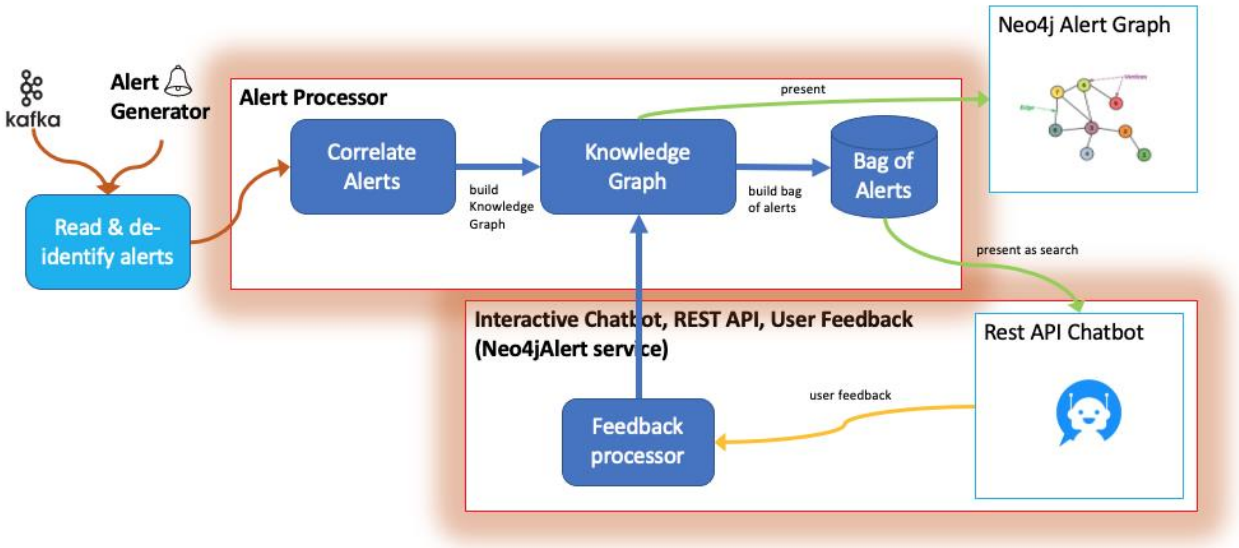
Overview

The modern scaled digital transformation has made it difficult for humans to keep up with the ephemeral state of IT workloads and processes although most of them have made significant investment on monitoring the application, infrastructure, and network. In case of any outage, these monitoring systems generate different alerts, but they do not generate them at the same time and not in sequence. It takes time for IT operations to find out the relation between these alerts and they end up creating too many tickets for different teams.

Application monitoring is the process of collecting different performance metrics and log data to help developers track availability, bugs, resource use, and changes to performance in applications that affect the end-user experience. Network monitoring provides the information that network administrators need to determine, in real time, whether a network is running optimally. Infrastructure monitoring provides visibility on all the assets that are necessary to deliver and support IT services: data centers, servers, storage, and other equipment. IT operations perform their day-to-day task and mitigate error scenarios via multiple alerts generated from these monitoring systems. It empowers the users- SRE's, L1/L2, developers, who rely on these alerts to ensure health and well-being of the services and environment. Sometimes these alerts become overwhelming and create unwanted noise. Reading through each alert and analyzing to get to the error pattern and identifying actionable alerts is a daunting task resulting in lengthy troubleshooting and remediation cycles and high mean time to resolution (MTTR).

In this project, we are correlating these alerts by collecting, analyzing, and building a knowledge graph between them so that it can predict and group future similar events that may affect availability and performance. We also improve the performance and reliability of the model through relevant feedback channels. This helps in identifying the symptoms and predicting probable problems upfront so that corrective actions can be taken to prevent system snags. By effectively identifying the correlated alerts, we are able to reduce the alert fatigue and work on the most urgent problem first, with reduced MTTRs.

Component Details



We have two major components

Alert Processor

This is the main component in our implementation and is responsible for

a) Alert Streaming (Alert Generator)
   Create a corpus of messages, hosts, services and sources
   Create any number of alerts based on the corpus
b) Alert Processing
   Create incidents based out of those alerts and assume these incidents consists of number of alerts which occur almost in the same interval of 5 minutes window
   Calculate Mutual Information between all alert pairs and dump the associated Mutual information and their relationships into CSV files.
c) Knowledge Graph
   Import those CSV files into neo4j database to build knowledge graph, to a running instance of Neo4j

Query Layer & Chat Client (with REST APIs)

The query layer, REST APIs and User interactive terminal client provides a medium to use Knowledge Graph for Alert diagnosis and providing Human in the loop feedback to update the Knowledge Graph with critical domain intelligence. It comprises of

a) REST APIs that can be plugged into desired UI to fetch the relevant Alert diagnostic information
b) Terminal interactive chat application to invoke different queries in order to show the power of knowledge graph.

c) The interactive terminal also enriches the Knowledge Graph with domain knowledge to mark the root cause alert enabling the human in the feedback loop). Once the user provides the input, if the same incident happens in the next time - we will be able to find out the root cause of that incident

Validation

To validate our implementations, we have Alerts collected from a real system. We process them under our solution and validate them against domain specific expected Alert correlation matrix. We have used Cranfield methodology for that

The Software Installation

Here is detailed documentation for how to install the software

Prerequisites

Java 11
Neo4j

Prepare the Dataset and calculate Mutual Information

- Ensure *java11* executable is present in the path
- Download *alert-correlation.zip* in the Artifacts directory and unzip it
- Run the following command from the extracted zip

```
java -jar alert-correlation.jar
```

- The guided instruction will help you to generate any number of dummy alerts and incidents so that you can mimic the data in your intended windows
- This will calculate the mutual information on various scenarios and prepare the data for building the knowledge graph

This is the sample output

% java -jar alert-correlation.jar
Enter the number of dummy incidents to create. (Default: 3) -> 4
Enter the number of correlated Alarm templates for Incident #1 (Default: 5) -> 6
Enter the number of correlated Alarm templates for Incident #2 (Default: 5) -> 7
Enter the number of correlated Alarm templates for Incident #3 (Default: 5) -> 9
Enter the number of correlated Alarm templates for Incident #4 (Default: 5) -> 4

Enter the number of total number of noise templates (Default: 200) ->500
Enter the number of intervals for the entire run (Max: 288) ->

####################################

For this demo scenarios, we have 5 incidents.
Incident #1 has 6 templates

Incident #2 has 7 templates
Incident #3 has 9 templates
Incident #4 has 4 templates
Incident #-1 has 500 templates

Total Templates = 526
Total Intervals = 287
Summary of Input data :

For Interval #1, 5 out of 6 templates will be enabled randomly , along with 8 out of 500 noise templates will be generated randomly
For Interval #2, 6 out of 7 templates will be enabled randomly , along with 8 out of 500 noise templates will be generated randomly
For Interval #3, 7 out of 9 templates will be enabled randomly , along with 8 out of 500 noise templates will be generated randomly
For Interval #4, 3 out of 4 templates will be enabled randomly , along with 8 out of 500 noise templates will be generated randomly

This will repeat 71 times

####################################

Complete Updating Mutual Information ... Took 261 ms

- This will create the following files

mutual_infomation.txt : This contains the mutual information for each and every Alert which has been generated based on the input provided

output/*.csv : This contains the metadata of knowledge graph in CSV format . This raw data will be used to populate Knowledge graph in neo4j database

Setup a Neo4j database

- On a Unix setup where docker is installed, create a directory named **csv** and place all the csv files generated on previous steps to this directory
- Spin off a Neo4j docker container

docker run -d -e NEO4J_AUTH=none -p 7474:7474 -v $PWD/csv:/var/lib/neo4j/import -p 7687:7687 neo4j:4.3

- Make sure you can access Neo4j browser http://<hostname>:7474/browser/

Load Data on Neo4j

- On a Unix setup where Neo4j docker is installed, create a directory named **csv** and place all the csv files generated on previous steps to this directory

- Run the following Cypher script to load the data

```
//id,incidentId,message
LOAD CSV WITH HEADERS FROM "file:///Alerts.csv" AS row CREATE (:Alert {id:row.id, incidentId:row.incidentId, name:row.message,
source:row.source, ci:row.ci, service:row.service});
// id,name
```

```
LOAD CSV WITH HEADERS FROM "file:///CIs.csv" AS row CREATE (:CI {id:row.id, name:row.name});
// id,intervalPeriod,startTime,endTime,date
LOAD CSV WITH HEADERS FROM "file:///Intervals.csv" AS row CREATE (:Interval {id:row.id, intervalPeriod:row.intervalPeriod,
startTime:row.startTime, endTime:row.endTime, date:row.date});
// id,name
LOAD CSV WITH HEADERS FROM "file:///Services.csv" AS row CREATE (:Service {id:row.id, name:row.name});
// id,name
LOAD CSV WITH HEADERS FROM "file:///Sources.csv" AS row CREATE (:Source {id:row.id, name:row.name});
// from,to
LOAD CSV WITH HEADERS FROM "file:///Source-CI-MONITORS.csv" AS row MATCH (s1:Source {id:row.from}), (c1:CI {id:row.to}) CREATE (s1)-
[:MONITORS{source: row.source, ci:row.target}]->(c1);
// from,to
LOAD CSV WITH HEADERS FROM "file:///Alert_Time-GENERATED_AT.csv" AS row MATCH (a1:Alert {id:row.from}), (t1:Interval {id:row.to})
CREATE (a1)-[:GENERATED_AT{alert: row.source, interval:row.target}]->(t1);
// from,to
LOAD CSV WITH HEADERS FROM "file:///CI-Alert-RAISED.csv" AS row MATCH (c1:CI {id:row.from}), (a1:Alert {id:row.to}) CREATE (c1)-
[:RAISED{ci: row.source, alert:row.target}]->(a1);
// from,to
LOAD CSV WITH HEADERS FROM "file:///CI-Time-ALERT_RAISED_AT.csv" AS row MATCH (c1:CI {id:row.from}), (t1:Interval {id:row.to}) CREATE
(c1)-[:ALERT_RAISED_AT{ci: row.source, interval:row.target}]->(t1);
// from,to
LOAD CSV WITH HEADERS FROM "file:///Service-CI-CONTAINS.csv" AS row MATCH (s1:Service {id:row.from}), (c1:CI {id:row.to}) CREATE (s1)-
[:CONTAINS{service: row.source, ci:row.target}]->(c1);
// from,to,mi
LOAD CSV WITH HEADERS FROM "file:///MutualInformation.csv" AS row MATCH (a1:Alert {id:row.from}), (a2:Alert {id:row.to}) CREATE (a1)-
[:CORRELATED_AT {mutual_information:row.mi}]->(a2);
```

Running the Neo4jAlerts service

a) Ensure Neo4j instance is running and set up with user and database.
b) Download **Neo4jAlerts.zip** in the Artifacts directory and unzip it.
c) Update your Neo4j instance details in application.properties:

org.neo4j.driver.uri=bolt://<neo4jsystem>:7687
org.neo4j.driver.authentication.username=<username>
org.neo4j.driver.authentication.password=<password>

d) Execute the service as follows on the terminal

java -jar demo-0.0.1-SNAPSHOT.jar com.example.demo.Neo4jAlertsApplication -a application.properties

e) The chat client will appear on the terminal. Through the guided instruction we can navigate the system to get additional info about alerts.
f) The chat client outputs the cypher query and that can be directly used. It also executes the queries automatically on Neo4j instance via Neo4j client.
g) The REST APIs can be executed on http://<locaalhost>:8080/ end points as below

- **What are the alerts counts in a given time duration?**
  HTTP GET /alertCounts
- **Diagnose the alert - get the top 5 correlated alerts**
  HTTP GET /alerts
- **What are the "root-cause" alerts ?**
  HTTP GET /rootCauseAlerts
- **What are impacted devices due to an alert?**
  HTTP GET /affectedCIs

- **What are the Unhealthy services?**
  HTTP GET /affectedServices
- **Provide feedback and mark an alert as "root-cause" alert**
  HTTP POST /setRootCause

Source Code Information

The *SourceCode* directory consists of
alarm_correlation_mutual_information – It has code for

- Alert Generation
- Mutual Information Calculation
- Generation of .csv files for importing data to Neo4j
- Load script for uploading data to Neo4j db

Neo4jAlerts – It has code for

- REST APIs
- Terminal Chatbot
- Cypher queries generation executes at Neo4j Client.

Simply download them or git clone from https://github.com/swatinanda/CS410-Fall2021-TeamAlpha.git

You can run that from IDE of your choice.

Concluding Remarks

The project was a great learning for all three of us and it was a very well-coordinated teamwork.
Initially we had two different use cases in mind, but with our combined research around the topics and their value as a solution, plus putting to use concepts grasped in the Text Information course class to play, we were able to zero down on this project. We researched Dataset preparation, finalizing Mutual Information calculation logic, using Neo4j as our Ontology and Knowledge graph, the best way to present this data to users to be able diagnose Alerts, how to incorporate humans in feedback loop and get an idea of impact and the validation methodology. We implemented each of our pieces and put it all together, it took us a few iterations and improvements to achieve the desired outcome.
We would like to extend thanks to Prof **ChengXiang Zhai,** his guidance and advice during his office hours, on multiple phases of the project from inception to intermediate progress to final output, helped us a lot. We demoed our system to him to get his opinion. His valuable feedback on validation with real data gave us an insight as to how we can validate our solution.
In all it was a satisfying experience and we worked as a cohesive team, to achieve the goal of the project – demonstrate the Text Information System concepts in a real-world scenario to solve a problem successfully.

Appendix: Detailed Concepts

Ingesting different kinds of Alerts

- *SourceCode/alarm_correlation_mutual_information/data* contains the corpus of alert message, hosts and services
- Based on the user input, we can create any number of alerts from the corpus
- A sample alert should contain the following information
  - id - Id of the alert
  - message - Message text of the alert
  - source - Source of the alert generated
  - host/entiry - Device from which the alert is generated
  - service – Service that the device is attached with
  - time – Time of the Alert generation
- These alerts will be used to for calculating Mutual Information
- These alerts are mimicked to be sent multiple times in different time interval

Alert Pre-processing and Alert Correlation Analysis

- We have divided 24hrs i.e 1 day in 288 windows where each window interval is 5 mnts duration . For eg. 00:00 – 00:05 is marked as interval window 1 , 00:15-00:20 is marked as Interval window 4 etc
- All alerts are marked to appear in one or multiple interval windows . A sample data will look like below :

	w1	w2	w3	w4	w5	w6	w7	w8	w9
A1	1	0	1	0	0	1	0	0	1
A2	1	0	0	1	0	0	0	0	1
A3	1	0	0	1	0	1	0	0	1
A4	1	0	0	1	0	1	0	1	1
A5	0	1	0	1	0	1	0	0	1
A6	0	1	0	0	1	0	1	0	0
A7	0	1	0	0	1	0	1	0	0
A8	0	1	0	0	1	0	1	0	0
A9	0	1	0	0	0	1	1	0	0
A10	0	1	1	0	1	0	1	0	0
A11	0	0	1	0	0	0	0	0	1
A12	0	1	0	0	0	0	0	0	0
A13	0	0	1	0	0	0	0	0	0
A14	0	0	0	1	0	0	1	0	0
A15	1	0	0	0	0	1	0	0	0
A16	0	0	1	0	1	0	0	0	1
A17	0	1	0	0	0	0	0	1	0
A18	1	0	0	0	0	0	1	0	0

A19	0	0	0	0	0	0	0	0	1

In this way, we can have data in a matrix format for N number of alerts of 288 window interval.
In the next step, we find the syntagmatic relationship between Alert A1 with Alert A2 i.e if A1 is more associated with A2, we can say that they tend to occur together
We take a binary random variable
t_{iA1} = {0,1} i.e presence of alert A1 at i^{th} window
t_{iA1} =1 means that A1 is present on i^{th} interval and t_{iA1} =0 means A1 is not present on the i^{th} interval
So,
$p(t_{iA1} = 1) + p(t_{iA2} = 0) = 1$
$H(A1) = -p(t_{iA1} = 0) \log_2 p(t_{iA1} = 0) - p(t_{iA1} = 1) \log_2 p(t_{iA1} = 1)$
Entropy (H) determines how difficult this is to predict the presence of Alert A1. Smaller entropy means easier to predict
Conditional Entropy for occurring A1 when A2 is present is
$H(A1|A2) = H(A1|A2=0) + H(A1|A2=1) = -p(t_{iA1} = 0 | t_{iA2} = 0) \log_2 p(t_{iA1} = 0 | t_{iA2} = 0) - p(t_{iA1} = 1 | t_{iA2} = 0) \log_2 p(t_{iA1} = 1 | t_{iA2} = 0) - p(t_{iA1} = 0 | t_{iA2} = 1) \log_2 p(t_{iA1} = 0 | t_{iA2} = 1) - p(t_{iA1} = 1 | t_{iA2} = 1) \log_2 p(t_{iA1} = 1 | t_{iA2} = 1)$
Mutual Information between A1 and A2 is defined as
$I(A1, A2) = H(A1) - H(A1|A2) = -p(t_{iA1} = 0) \log_2 p(t_{iA1} = 0) - p(t_{iA1} = 1) \log_2 p(t_{iA1} = 1) - p(t_{iA1} = 0 | t_{iA2} = 0) \log_2 p(t_{iA1} = 0 | t_{iA2} = 0) - p(t_{iA1} = 1 | t_{iA2} = 0) \log_2 p(t_{iA1} = 1 | t_{iA2} = 0) - p(t_{iA1} = 0 | t_{iA2} = 1) \log_2 p(t_{iA1} = 0 | t_{iA2} = 1) - p(t_{iA1} = 1 | t_{iA2} = 1) \log_2 p(t_{iA1} = 1 | t_{iA2} = 1)$

Mutual information between A1 and A2 indicates the amount of reduction of entropy when A2 is present. If the information is larger, we can say that they are syntagmatically related i.e alert A1 and A2 are part of the same incident
In order to calculate the Mutual Information , we need to calculate only these 3 probabilities

$p(t_{iA1} = 1)$ = (Total number of times alert A1 occurred + 0.5) /288 +1
$p(t_{iA2} = 1)$ = (Total number of times alert A2 occurred + 0.5) /288 + 1
$p(t_{iA1} = 1 | t_{iA2} = 1)$ = (Total number of times alert both A1 and A2 occurred + 0.25) /288+1

where 288 is the total number of windows interval
0.5 is added as smoothing parameter while calculating $p(t_{iA1} = 1)$ and $p(t_{iA2} = 1)$, similarly 0.25 is added for $p(t_{iA1} = 1 | t_{iA2} = 1)$. To balance the calculation , we added 1 in the denominator
Rest of the probabilities can easily be obtained since we have the following relationships and constraints

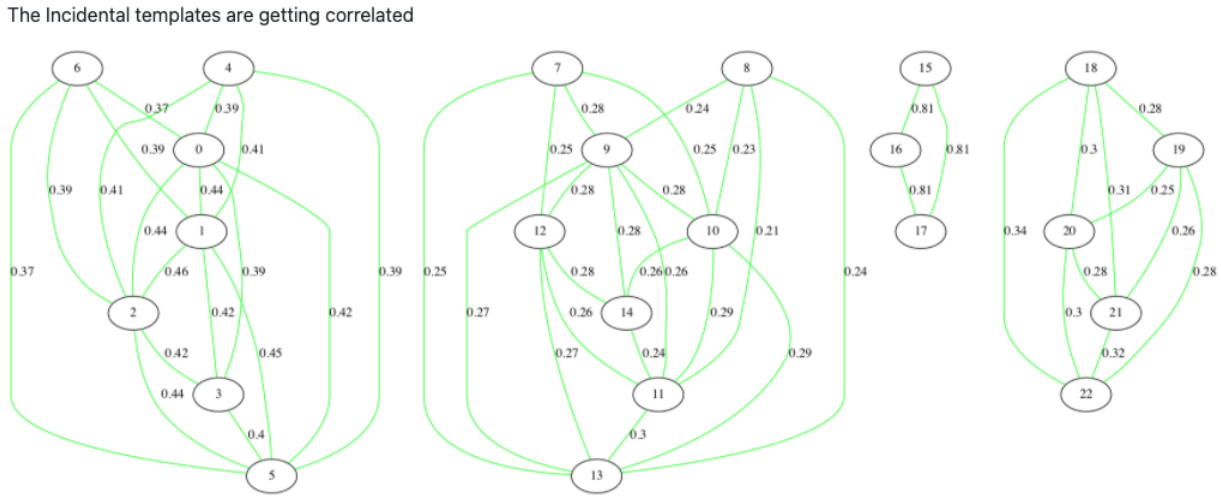
- Presence or absence of Alert A1 and A2 on i^{th} interval
  $p(t_{iA1} = 1) + p(t_{iA1} = 0) = 1$
  $p(t_{iA2} = 1) + p(t_{iA2} = 0) = 1$
- Co-occurrences of A1 and A2 on i^{th} interval
  $p(t_{iA1} = 1 | t_{iA2} = 1) + p(t_{iA1} = 1 | t_{iA2} = 0) + p(t_{iA1} = 0 | t_{iA2} = 1) + p(t_{iA1} = 0 | t_{iA2} = 0) = 1$
- Constraints
  $p(t_{iA1} = 1 | t_{iA2} = 1) + p(t_{iA1} = 1 | t_{iA2} = 0) = p(t_{iA1} = 1)$
  $p(t_{iA1} = 0 | t_{iA2} = 1) + p(t_{iA1} = 0 | t_{iA2} = 0) = p(t_{iA1} = 0)$
  $p(t_{iA1} = 1 | t_{iA2} = 1) + p(t_{iA1} = 0 | t_{iA2} = 1) = p(t_{iA2} = 1)$
  $p(t_{iA1} = 1 | t_{iA2} = 0) + p(t_{iA1} = 0 | t_{iA2} = 0) = p(t_{iA2} = 0)$

Using the above calculation, we could able to fetch the Mutual Information of all pair of alert generated

During the above process, we created the following CSV with the ontology and mutual information data as below:
Alert --------GENERATED_AT---------- > Window Interval
Device --------RAISED ---------- > Alert
Device --------ALERT_RAISED_AT------ > Window Interval

SERVICE --------CONTAINS---------- > ALERT
Source --------MONITORS---------- > Device
Alert --------CORRELATED_AT-------- > Alert

On the edge of the CORRELATED_AT, we pass the Mutual Information between the 2 alerts.
Below graph shows how we have achieved clustering of alerts (incident templates) using Mutual Information.



Building a querying layer to pull information from knowledge graph

The query layer is responsible for Creating Cypher queries that can be executed automatically or run in the Neo4j browser.

These Cypher queries can

1) Fetch various diagnostic information about the Alerts and their effects on the devices, entities, and services from the Neo4j Knowledge Graph.

- **What are the alerts counts in a given time duration?**
- **Diagnose the alert - get the top 5 correlated alerts**
- **What are the "root-cause" alerts ?**
- **What are impacted devices due to an alert?**
- **What are the Unhealthy services?**

2) Update any user feedback and enrich the Neo4j Knowledge Graph

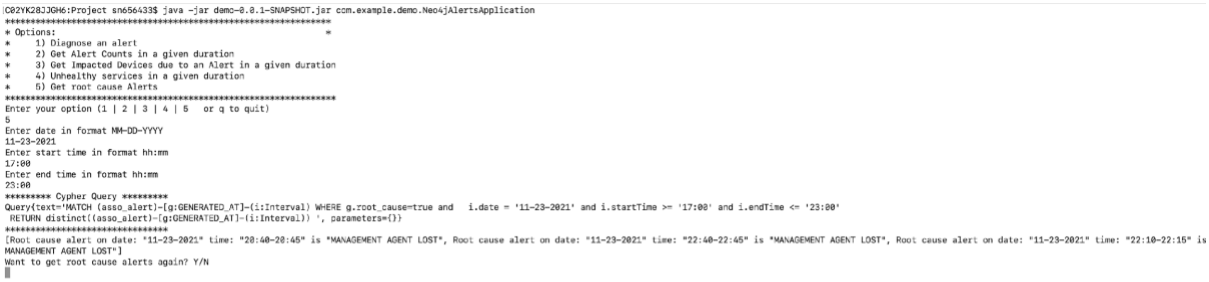
- **Provide feedback and mark an alert as "root-cause" alert**

In the current implementation the Neo4jAlerts service creates and executes these queries - generated per scenario through the Chat Client or REST APIs.

We are using Neo4j spring client to run and fetch response from our Neeo4j docker instance.

Building a Chat client that will interact with the Knowledge graph

The Neo4jAlerts service, also consists of a terminal-based User Interactive chatbot. It guides the user to diagnostic queries by entering relevant input params to get insights around the alerts.



Various REST APIs exposed to Chat client

The Neo4jAlerts service also, wraps the diagnostic cypher queries as REST APIs, so that they are available as backend to desired custom UI. Following are the endpoints.

- **What are the alerts counts in a given time duration?**
  HTTP GET /alertCounts

- **Diagnose the alert - get the top 5 correlated alerts**
  HTTP GET /alerts

- **What are the "root-cause" alerts ?**
  HTTP GET /rootCauseAlerts

- **What are impacted devices due to an alert?**
  HTTP GET /affectedCIs

- **What are the Unhealthy services?**
  HTTP GET /affectedServices

- **Provide feedback and mark an alert as "root-cause" alert**
  HTTP POST /setRootCause

Building a querying layer to pull information from knowledge graph

The query layer is responsible for Creating Cypher queries that can be executed automatically or run in the Neo4j browser.

These Cypher queries can

1) Fetch various diagnostic information about the Alerts and their effects on the devices, entities, and services from the Neo4j Knowledge Graph.

- **What are the alerts counts in a given time duration?**

- **Diagnose the alert - get the top 5 correlated alerts**
- **What are the “root-cause” alerts ?**
- **What are impacted devices due to an alert?**
- **What are the Unhealthy services?**

2) Update any user feedback and enrich the Neo4j Knowledge Graph

- **Provide feedback and mark an alert as “root-cause” alert**

In the current implementation the Neo4jAlerts service creates and executes these queries - generated per scenario through the Chat Client or REST APIs.

We are using Neo4j spring client to run and fetch response from our Neeo4j docker instance.

All the implemented Cypher queries are as follows:

Cypher Queries:

- **What are the alerts counts in a given time duration?**

MATCH(i:Interval)-[r]-(a:Alert) WHERE i.date >= '11-23-2021' AND i.startTime >= '17:00' AND i.date <= '11-23-2021' AND i.endTime <= '23:00’ return count(a) as cnt

- **Diagnose the alert - get the top 5 correlated alerts**

MATCH (n:Alert) WHERE toLower(n.ci)=toLower('bld-med-bca-05.ihl.broadcom.net’) AND toLower(n.name) CONTAINS toLower('Interface is down')
CALL { WITH n MATCH (n)-[r:CORRELATED_AT]-(asso_alert)
RETURN asso_alert,(n)-[r:CORRELATED_AT]-(asso_alert) as relation
ORDER BY r.mutual_information DESC. }
RETURN asso_alert as vertex, relation as edge LIMIT 5

- **Provide feedback and mark an alert as “root-cause” alert**

MATCH (n:Alert) WHERE toLower(n.name) CONTAINS toLower('Interface is down')
MATCH (n)-[r:CORRELATED_AT]-(asso_alert), (asso_alert)-[:GENERATED_AT]-(i:Interval)
WHERE i.date = '11-23-2021' AND i.startTime <= '20:43' AND i.endTime >= '20:43' AND toLower(asso_alert.name) CONTAINS toLower('MANAGEMENT AGENT LOST')
MERGE (asso_alert)-[g:GENERATED_AT]-(i)
ON MATCH SET g.root_cause=true

- **What are the “root-cause” alerts**

MATCH (asso_alert)-[g:GENERATED_AT]-(i:Interval)
WHERE g.root_cause=true and i.date = '11-23-2021' and i.startTime >= '16:00' and i.endTime <= '23:00'
RETURN distinct((asso_alert)-[g:GENERATED_AT]-(i:Interval))

- **What are impacted devices due to an alert?**

MATCH (n:Alert) WHERE toLower(n.name) CONTAINS toLower('interface is down')

MATCH (n)-[r:CORRELATED_AT]-(asso_alert)
with n,asso_alert, r ORDER BY r.mutual_information DESC limit 5
optional MATCH (n)-[ar]-(c:CI), (c:CI)-[ci]-(i:Interval)
WHERE i.date >= '11-23-2021' AND i.startTime >= '17:00' AND i.date <= '11-23-2021' AND i.endTime <= '18:00'
return c as ci ,(n)-[ar]-(c:CI) as ar
UNION
MATCH (n:Alert) WHERE toLower(n.name) CONTAINS toLower('interface is down')
MATCH (n)-[r:CORRELATED_AT]-(asso_alert)
with n,asso_alert, r ORDER BY r.mutual_information DESC limit 5
optional MATCH (asso_alert)-[ar]-(c:CI), (c:CI)-[ci]-(i:Interval)
WHERE i.date >= '11-23-2021' AND i.startTime >= '17:00' AND i.date <= '11-23-2021' AND i.endTime <= '18:00'
return c as ci ,(asso_alert)-[ar]-(c:CI) as ar

- **What are the Unhealthy services?**

MATCH (n:Alert) WHERE toLower(n.name) CONTAINS toLower('interface is down')
MATCH (n)-[r:CORRELATED_AT]-(asso_alert)
with n,asso_alert, r ORDER BY r.mutual_information DESC limit 5
optional MATCH (n)-[ar]-(c:CI), (c:CI)-[ci]-(i:Interval), (c:CI)-[]-(s:Service)
WHERE i.date >= '11-23-2021' AND i.startTime >= '17:00' AND i.date <= '11-23-2021' AND i.endTime <= '23:00'
return c as ci ,(n)-[ar]-(c:CI) as ar , i, (c:CI)-[]-(s:Service) as cis, s
UNION
MATCH (n:Alert) WHERE toLower(n.name) CONTAINS toLower('interface is down')
MATCH (n)-[r:CORRELATED_AT]-(asso_alert)
with n,asso_alert, r ORDER BY r.mutual_information DESC limit 5
optional MATCH (asso_alert)-[ar]-(c:CI), (c:CI)-[ci]-(i:Interval), (c:CI)-[]-(s:Service)
WHERE i.date >= '11-23-2021' AND i.startTime >= '17:00' AND i.date <= '11-23-2021' AND i.endTime <= '23:00'
return c as ci ,(asso_alert)-[ar]-(c:CI) as ar , i, (c:CI)-[]-(s:Service) as cis, s

Validation

We have collected a set of alerts where a real incident has happened. We have collected their timestamp and processed our solution to see if we find relatively high mutual information for the incidental alerts. We used Cranfield Methodology for that.
We prepared a baseline matrix whose snippet is shown below.

Id	0	41	82	205	567	789	987
0	N/A	Y	Y	Y	N	N	Y
41	Y	N/A	Y	Y	N	N	Y
82	Y	Y	N/A	Y	N	N	Y
205	Y	Y	Y	N/A	N	N	Y
567	N	N	N	N	N/A	N	N
789	N	N	N	N	N	N/A	N
987	Y	Y	Y	Y	N	N	N/A

From this table, we can see that Alert Ids 0, 41, 82, 205, 987 are part of the same incident and they are supposed to be correlated and should have high mutual information between them (marked as Y) whereas 567 and 789 alert Ids are not related to the same incident (marked as N) – so we expect to see low mutual information value

When we pass these set of alerts along with other thousands of alerts generated at different times – we found the following matrix of mutual information

Id	0	41	82	205	567	789	987
0	0	0.133	0.104	0.166	0.025	0.021	0.166
41	0.133	0	0.104	0.105	0.019	0.018	0.156
82	0.104	0.104	0	0.133	0.021	0.009	0.187
205	0.166	0.105	0.133	0	0.019	0.019	0.127
567	0.025	0.019	0.021	0.019	0	0.011	0.016
789	0.021	0.018	0.009	0.019	0.011	0	0.014
987	0.166	0.156	0.187	0.127	0.016	0.014	0

This data is quite inline to our expectation where we can see the mutual information between alert Ids 0, 41, 82, 205 and 787 is mostly greater than 0.1 whereas their mutual information with either 567 and 789 is very less

In conclusion, we must say correlating alerts and forming different clusters with mutual information is a novel idea and can widely be used in AIOPs domain for noise reduction and incidental cluster formation.