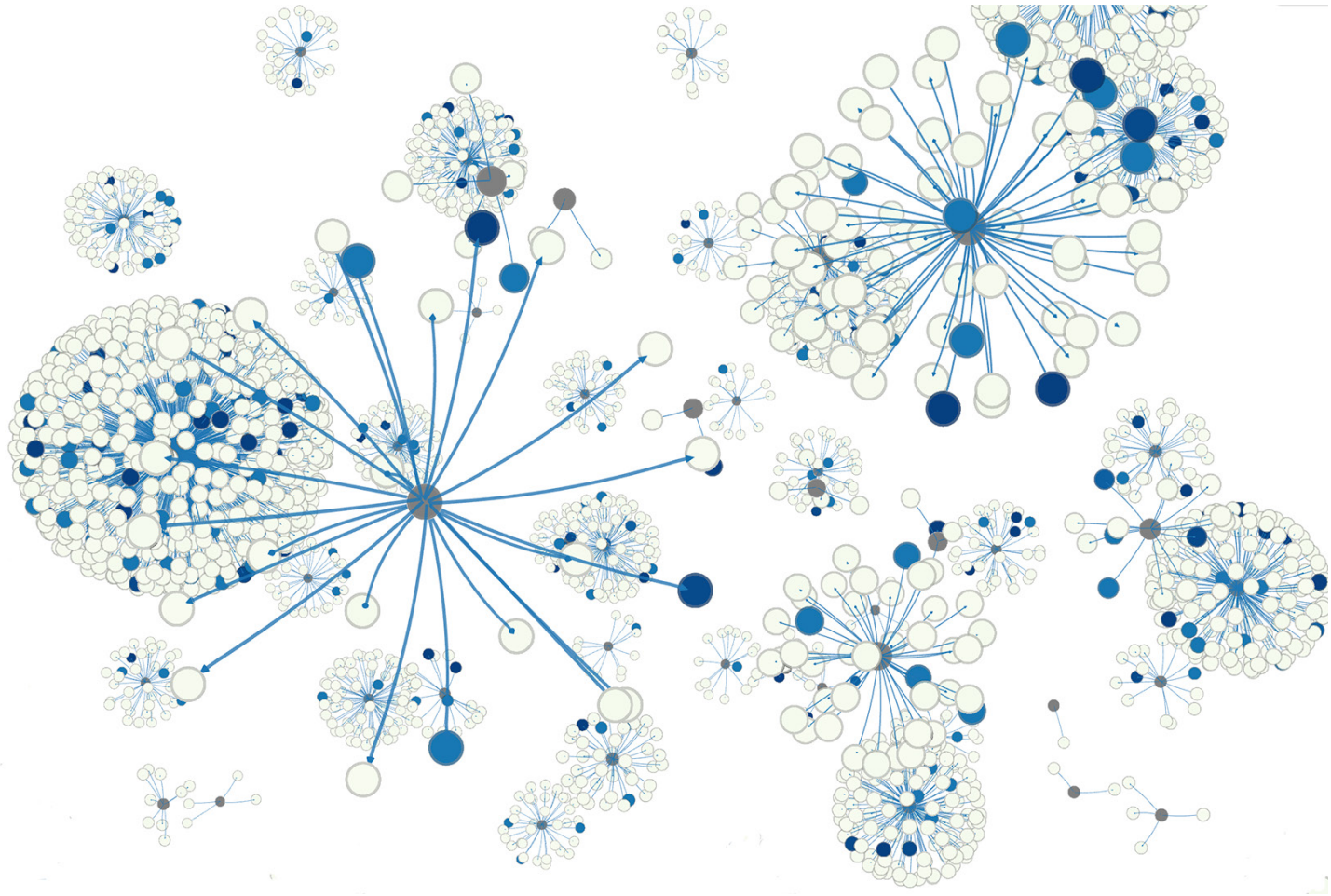
KINEVIZ

WHITE PAPER

Rapid Threat Response with Automated Feature Extraction

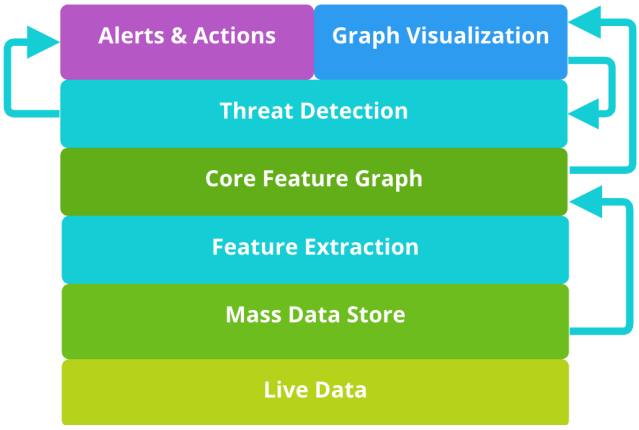


Contents

Overview

Automated detection of patterns of events representing threat or risk is essential for many enterprises. Current solutions can mine enterprise-scale data to flag known threats and manage responses. But they are not effective at detecting emerging or hidden threats. That's because in massive relational database environments, patterns of events which are related in unexpected ways and which unfold over time are difficult to discover. Data modeled as a graph enables rapid, efficient discovery of hidden relationships, making the porting of enterprise data to a graph database seem like an appealing solution. However at scale, this approach can dramatically increase infrastructure overhead without necessarily speeding up delivery of actionable intelligence or supporting the management of response.

A way forward is to build a tiered system that leverages the strengths of traditional database infrastructure, yet provides live visibility into patterns of risk or threat in big data. A connection to the data store can be established that reduces risk-related events to a much smaller set of aggregate patterns, or features, which are stored as event nodes in a graph database. Through rapid graph visualization, exploration, and analysis, an initial set of basic features can become the basis for quickly discovering and defining higher-tier threat patterns, even those involving previously unsuspected threshold or trigger events.



Threat landscapes are always changing, which means that near real-time connection to a data store is essential for detecting known patterns as well as discovering emerging ones. The data store provides access speed, update, security, and other mature big data infrastructure. As the exploratory hub in a data mesh architecture, GraphXR provides rapid visibility into the feature graph. The data store is always connected, so it becomes possible to discover and define threats whose complex connections and sequences of events are otherwise difficult or impossible to extract from massive enterprise data. And detecting those threats automatically in near real time leads to greater success in preventing loss or damage.

Why Feature Extraction?

Feature extraction brings rapid graph visualization and automated response to enterprise-scale risk management. For rapid response to risks or threats, enterprises must address two main issues: business intelligence (BI), and operational response. Put simply, data must be accessed and organized to reveal threats whose occurrence can in turn trigger automated responses.

But a BI solution connected to massive enterprise data inherits the difficulty of querying relational data for unexpected patterns which have not been explicitly modeled. Detecting needle-in-a-haystack patterns such as connected networks of colluding actors or sequences of seemingly unrelated events fast enough to prevent loss is often too difficult.

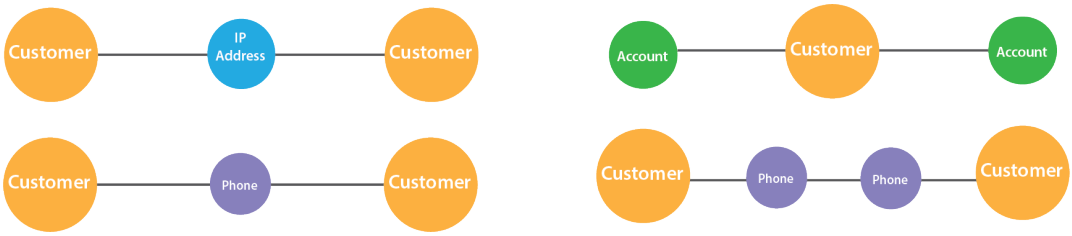
Data modeled as a property graph can reveal unexpected patterns rapidly, so why not map all data to a graph database? Unfortunately, queries on partitioned graph databases perform poorly unless they are carefully managed to avoid graph traversal across server boundaries. The issue is greatly complicated when data is being continuously updated.

A hybrid approach is to build a two-tier solution for rapid pattern detection that provides live, focused graph visualization of risk-related patterns, while leveraging the strengths of existing data infrastructure. Aggregated risk features can be extracted efficiently from the data store and re-cast as graph patterns. In practice, even very large data stores yield a reduced feature graph that can reside on a single server, and this avoids the problem of non-performant graph queries. With feature extraction, a daily flow of over 100 million data points can be condensed to about ten thousand features, or event nodes–– a 10,000-fold reduction. With this daily flow, one needs only about 4 million nodes per year, which is trivial to implement on a single graph database server.

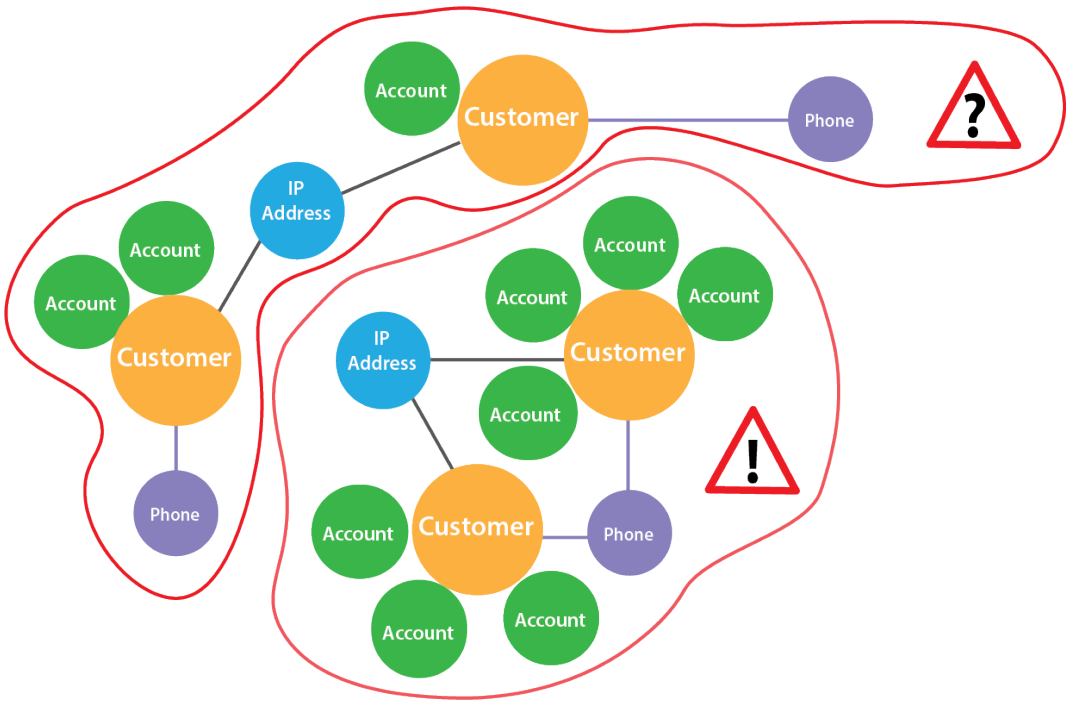
The aggregated features can then be used to define patterns of emerging threat. Often, a threat is composed of multiple features put together, in much the same way that image recognition uses elements such as edges, corners and shapes to recognize human faces in a photo or video.

An example illustrates how feature extraction can be used to identify patterns of fraudulent collusion among an enterprise's customers. An initial set of patterns can be extracted, such as:

- Customers sharing the same IP address.
- Customers using the same phone number.
- Customers who control multiple accounts, above a specific threshold.
- Interactions between customers such as messaging or email.



Visualizing the feature graph, we can see that interesting patterns already emerge, and we can quickly trace what's happening with just those events.



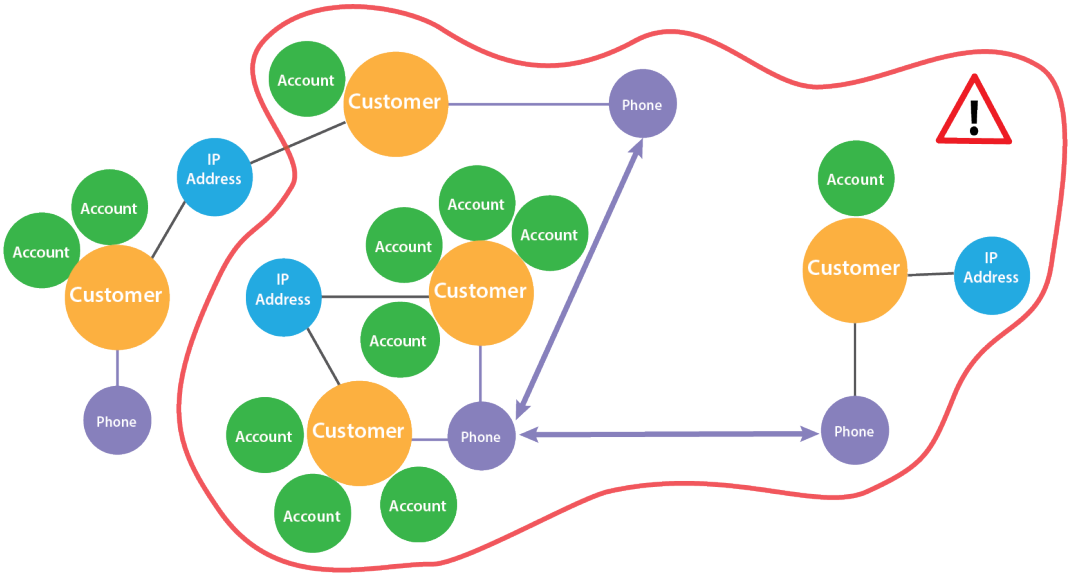
Now questions about specific suspected users, including interactions among those who have been in phone or email contact can be explored.

Questions might include:

- What was this customer's involvement in the past?
- Do we see specific customers sharing accounts in the past?
- Are groups of customers who are now doing something odd related in some way?
- By tracing connections further, do we uncover accounts that could be controlled by a specific individual or group?

Expanding the graph on phone contacts between our suspects now reveals a seemingly unconnected customer who could ultimately control the network of accounts that we've uncovered.

... In practice, analysts have been able to speed up detection and response times from days to under an hour.



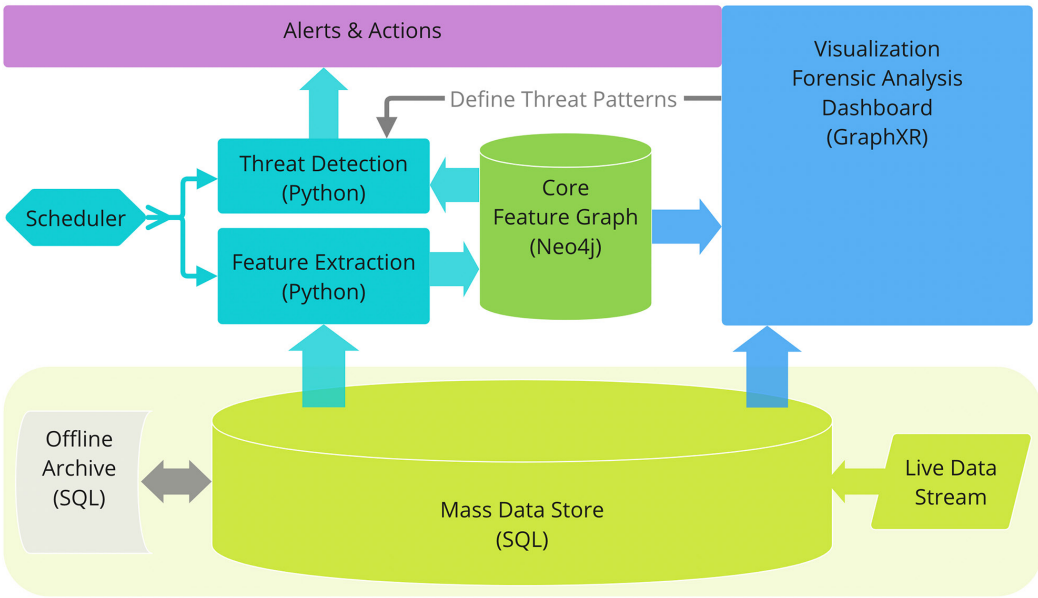
Within a massive SQL database environment, competent data scientists might be able to extract some of these results, but only with considerable time and effort. Adding an intermediate graph database to store the feature graph makes exploratory discovery intuitive, seamless, and cost-effective. The connected data store and automatically updated feature graph grant us high quality visibility into current status of likely threats. We can also easily investigate specific aggregated features in detail, and query the data store for additional related information as needed. This exponentially enhances the capability to spotlight emerging risks rapidly.

Threat detection also feeds into automated operational response. Higher tier features can include triggers for alerts that must be sent when the pattern is detected. This enables rapid response to complex threat patterns. In practice analysts have been able to speed up detection and response times from days to under an hour.

System Architecture

Automated threat detection and alert capability can be built on top of existing data stores. The required add-ons reduce threat patterns to combinations of live data and the extracted features stored in an intermediate graph database, enable ongoing discovery and definition of threats using rapid graph visualization and analytics, and automate the queries for threats and the triggers for alerts. Elements include:

- **Mass Data Store** (SQL). The data store is always connected, and updated continuously through live data streaming. Typically, only a few months of live data are included and older data accessed as needed from offline archives.
- **Feature Extraction** (Python). Specified features are extracted from the live data store, re-cast as graph event nodes, and stored in the feature graph.
- **Threat Detection** (Python). Threats defined through the core feature graph and SQL queries on the live data store are automatically scheduled for detection, and the appropriate alert triggers defined.
- **Core Feature Graph** (Neo4j). The feature database contains current features which have been extracted from the data store.
- **Visualization & Forensic Analysis** (GraphXR). Features can be displayed in a 3D project space and expanded upon for detailed analysis. Additional information can be delivered through SQL query on the live data store, and automatically connected to the graph.
- **Alerts & Actions**. Alerts, the response and the outcomes can be organized and monitored in a dashboard available to stakeholders.



Data Store

Architecture built on top of an existing big data store can provide always-on live access to the continuously updated data needed for risk management. SQL architecture lets enterprises build big data stores at massive scale, combining data about the current state of the business with historical data that can be used to identify patterns and trends for forecasting. Data collection, normalization, cleaning, security, and archiving are mature, well-optimized processes. Large databases can be partitioned effectively, with access and analytics provided through serverless big data services such as Athena or Google BigQuery. An enterprise might keep a few months of the data as hot storage, while sending older data to secondary storage, where, again, access can be provided seamlessly.

A SQL/relational database environment solves many big data issues, yet struggles to deliver complex, real time business intelligence that addresses new situations. To drive intelligence around emerging threats, re-casting data as a graph does let you flexibly connect categories of information and pursue investigations that involve multiple systems, individuals, and kinds of entities. These are questions that are difficult or impossible to pursue with relational database queries.

So why not simply port all the data to a graph database? For real time risk management, a strategy of mapping massive, continually updated relational databases to an equally massive graph database, and then exploring the patterns is unlikely to deliver adequate performance at a reasonable cost. Graph database providers are currently trying to guarantee the ability to host data on the scale of traditional SQL databases, but it unfortunately remains a challenge. This is largely because partitioning a massive graph database requires reconciling multiple servers across the cluster to manage performance of queries where edges cross different servers. You must traverse the graph to see if other links that are needed are worth duplication. And you must manage replication of the edge table across different servers in the cluster, so that a graph traversal doesn't have to send a query to another server. That's a much more complex problem than partitioning relational data, where you just append data to the table, and do the necessary cleaning and checking. Additional challenges exist around managing live updates, data security, archiving, and more.

In any case, regardless of how a massive data store is modeled, visibility into actionable business intelligence requires thoughtful data reduction, typically involving filtering and/or aggregation of collections of events. Rather than throwing away or duplicating existing database investment, one can instead craft a data mesh solution by building add-ons to the live data stores that automatically extract and reduce risk-related data and recast it as graph data for rapid visualization.

Feature Extraction

Feature extraction is a multi-tiered process that starts with delivering a basic set of aggregated risk-related features, which are then explored visually to discover more complex, higher-tier threat features. Through extraction, even large data stores yield a reduced feature graph that can reside in a single server, ensuring fast graph queries.

Implemented in Python, the resulting extract, transform, and load (ETL) process reduces and re-casts live data as graph event nodes, or features, and stores them in a graph database.

The feature graph initially consists of basic features defined by domain experts. These don't change very much. For example, fraud-related features could include "different users who log in with the same phone number", or "users who make multiple login attempts".

The extraction process maintains a live connection to the data store. Extraction is controlled by a scheduler which also controls automated threat detection and alerts.

Feature Graph Store

Aggregate events or features are stored in a graph database such as Neo4J Aura. The initial feature graph store reduces data to patterns that one can immediately start to learn from. This first tier typically won't change very much over time. The data store is always connected so that updates are automatically reflected in the aggregated feature graph.

Higher-tier threats are defined by exploring and expanding upon specific features of immediate interest. This can provide visibility into emerging threat patterns such as user collusion, or deteriorating performance over time.

Visualization and Analysis

A hub is needed for tracing and defining the complex associations that signal threats, and this is provided by GraphXR. Essentially it serves two purposes. The first is to provide basic visibility of extracted features–a BI component. The second is to provide a way to interact with and refine algorithms for higher-tier threat detection. In addition to connecting to the feature graph, the live data store can be queried for further related events, and these can be modeled directly as graph data and merged with the feature graph store. This lets us explore what's behind the core events, and enables evaluation of whether a threat or risk has emerged.

In GraphXR we can:

- Immediately visualize specific suspicious behavior patterns.
- Capture a pattern and further trace and analyze its connections.
- Apply thresholds or additional data to an initial result.

A feature graph may already show enough information to identify a threat with clarity. But at any time we can go deeper into specific connections of interest by querying the live data store. We know that many attributes that we'd like to explore will be available only in the mass data store, and we also know that the data will keep on changing. Live access lets us refine and expand into the more complex, higher tier detection algorithms required to forestall emerging threats. Focusing on specifics related to particular core features lets us narrow down additional information that must be retrieved from the data store. This rapid engagement with live data is the key to speeding up response.

For example, exploring networks of user collusion over time might start with an initial suspect pattern. Expanding the graph through its existing relationships can reveal hundreds of other connected features. We can then expand that graph further through SQL queries on the live data store to discover details such as:

- When each user in a network joined the platform.
- How many times and when each user took a specific action.

Automated Threat Detection and Alerts

A detection module senses higher-tier threat patterns as they occur and immediately triggers appropriate alerts for human or automated response. Detection can be implemented in Python and controlled through an automated scheduler such as Apache Airflow. Progress and outcome of the response can also be securely logged.

Automated system response can include blocking transactions, suspending accounts, or shutting down threatened devices or servers. For human response, email alerts can be sent to various stakeholders, such as system administrators, risk managers and analysts, or entire response teams.

Triggers can specify the number of specific events that will raise an alert, or the number of events beyond a given threshold. For an imminent threat that must be stopped immediately, a system action can be triggered to block further user activity or to shut down compromised servers. Alerts can also be sent on a predetermined schedule, and calibrated for severity and the expected level of response. For example, a trigger might specify a daily email report on users who seem to be colluding with one another, and show when the events in question happened.

Conclusion

Effective risk management depends upon mining relationships in enterprise-scale data in real time. Threat landscapes are always changing, which means that near real-time connection to a data store is essential for detecting known patterns as well as discovering emerging ones. But patterns of events related in unexpected ways and which unfold over time are difficult to discover in enterprise-scale relational databases.

Reducing the risk-related but sparse events in a big data store to a much smaller set of aggregate graph patterns is a highly effective solution. Extraction is a multi-tiered process that starts with delivering a basic set of risk-related patterns, or features, which are then explored visually to discover more complex, higher-tier threat features. The feature pipeline reduces and re-casts live data as graph event nodes stored in a graph database. Even large data stores yield a reduced feature graph that can reside in a single server, ensuring fast graph queries.

As the exploratory hub in a data mesh architecture, GraphXR enables rapid engagement with the high-quality, up-to-the-minute information available in a feature graph. This enables rapid discovery and definition of threat patterns whose complex connections and sequences of events are otherwise difficult or impossible to detect. An automated detection module for higher-tier threat patterns immediately triggers appropriate alerts for human or automated response. Connecting the live feature graph and automated threat detection can flag emerging patterns that signal a threat well before loss or damage to the enterprise occurs.

Resources

Alert Dashboard (Open Source)
Grafana OSS | Metrics, logs, traces, and more

Partitioning Large Graph Databases
Cascade-aware partitioning of large graph databases | SpringerLink
Can you partition a graph database? If so, how? - Stack Overflow
Distribution and Partitioning in Graph Databases - DZone

Partitioning SQL Databases
Data partitioning guidance - Azure Architecture Center | Microsoft Learn

Python
Python for Data Analysis

Workflow Scheduling
Apache Airflow
Open source workflow authoring, scheduling, and monitoring. Python-native.