Calling all Apache Cassandra® developers

# Cassandra Day + Wakanda Forever

## 11 // 10 // 2022

Santa Clara, CA | Bellevue, WA | Houston, TX

Apache CASSANDRA™ | DataStax | intel.

# Cassandra NoSQL for Real-time graph ETL and modern Data Pipelines


Michael Aglietti - Developer Relations - thatDot

# Michael Aglietti

Director, Developer Relations

- Long time DevRel leader
- Data-driven fanatic
- Solutions Architecture
- Systems Engineer
- Information Architect

@michael-aglietti

@maglietti

@michaelaglietti

# Who is thatDot

- Creators of Quine
- Based in Portland, OR
- Roots in DARPA R&D
- Team of serial entrepreneurs
- VC Backed

DARPA

The DARPA Transparent Computing (TC) program aims to provide high-fidelity visibility into component interactions during system operation across all layers of software abstraction, while imposing minimal performance overhead.

# Turn high-volume **event streams** into high-value **insights**

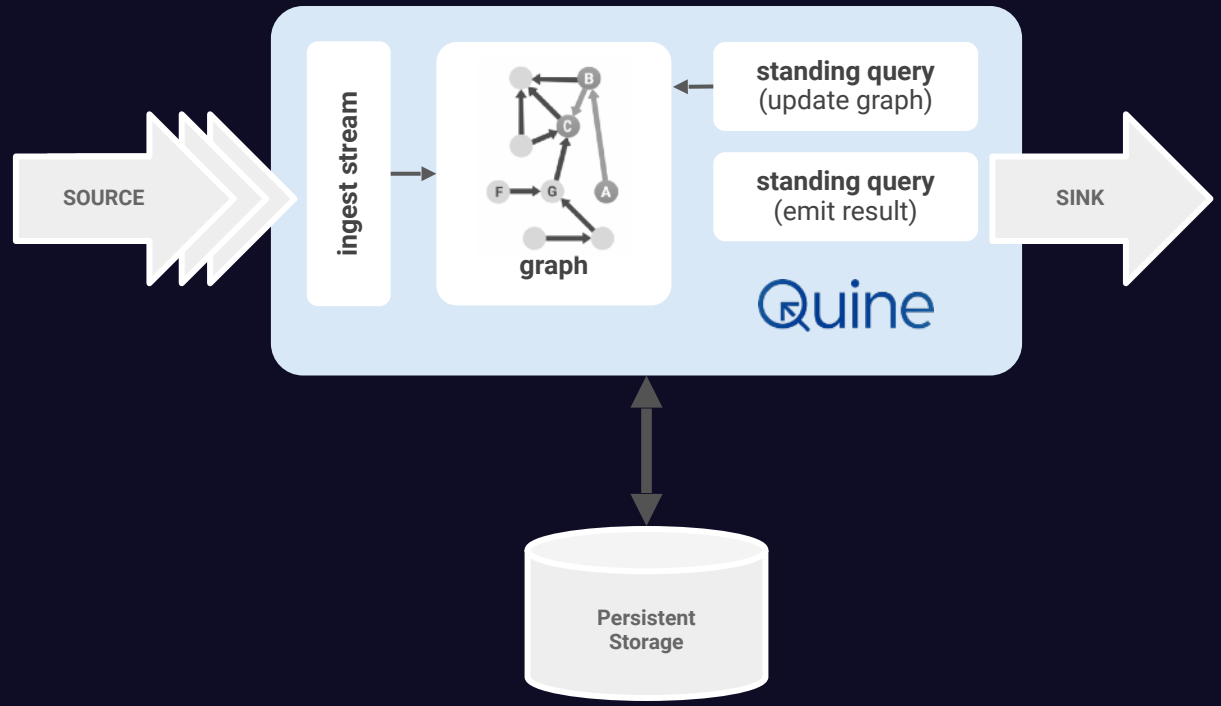

Quine is a streaming ETL interpreter that consumes data, builds a graph, runs live computation on that graph to answer questions or compute results, and then stream them out.

- Everything needed between two streams (Kafka, Kinesis, etc.)
- Data streams in, meaningful interpreted events stream out
- Be notified of complex data patterns in real-time
- Graph model for data AND computation

- Real-time pattern matching with standing queries
- Select nodes vs search for nodes
- Historical versioning instead of managing time windows
- Streaming system back pressure
- Stateful, event-driven computation

# Processing Event Streams

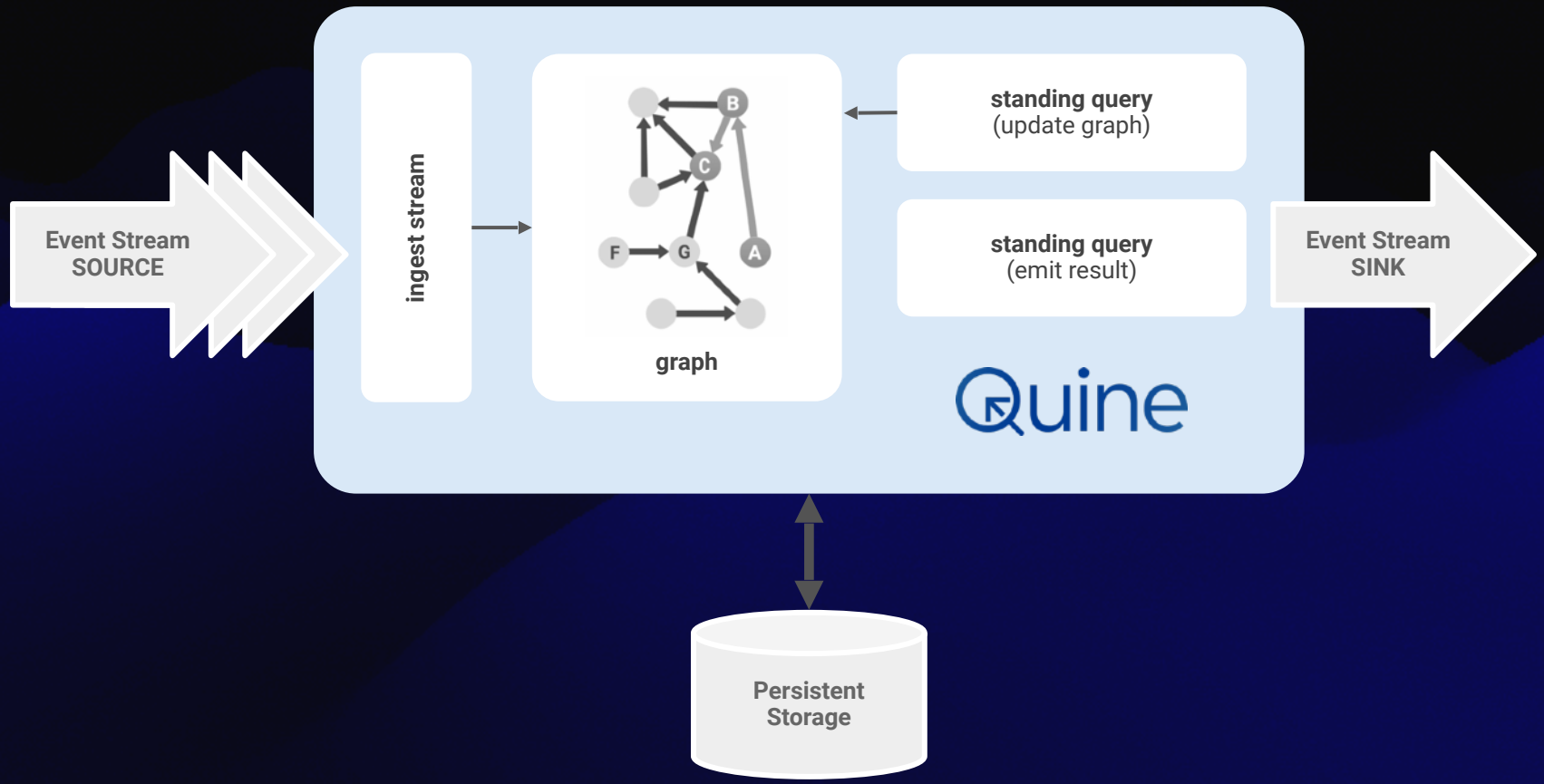
- Quine shapes event streams – data that is **continuously generated**, often in high volumes and at **high velocity** – into a graph
- Traditional node indexing methods are **slow**
- That is why we use a custom Cypher function idFrom() which creates a **deterministic** ID based on intuitive developer-configured characteristics of the node
- Eliminates querying for node properties
- Cornerstone concept for Quine Cypher queries



# Environment Set-Up

Two Kafka topics and a Cassandra instance walk into a bar ...

thatDot


Event Stream
SOURCE
ingest stream
B
C
F
G
A
graph
standing query
(update graph)
standing query
(emit result)
Quine
Event Stream
SINK
Persistent
Storage

# Quine Configuration

*Quine supports multiple persistent data storage layers. Apache Cassandra is used in production deployments to ensure performance and resilience.*

- → Configure Cassandra in quine.store
- → Pass the configuration to Quine when launching the jar file
- → View the running configuration via the API

The complete Cassandra Persistor documentation is available on docs.quine.io in components.

```
# Use Astra DB as persistent storage
quine.store {
  # store data in an Apache Cassandra instance
  type = cassandra

  # the keyspace to use
  keyspace = quine

  # whether the application should create the keyspace
  should-create-keyspace = true

  # whether the application should create tables
  should-create-tables = true

  # how many copies of each datum the Cassandra cluster should retain
  replication-factor = 3

  # how many hosts must agree on a datum
  write-consistency = LOCAL_QUORUM
  read-consistency = LOCAL_QUORUM

  # passed through to Cassandra
  local-datacenter = ASTRA_CLOUD_REGION

  # how long to wait before considering a write operation failed
  write-timeout = "10s"

  # how long to wait before considering a read operation failed
  read-timeout = "10s"
}
```

# Analyze, Shape, and Load

Understand first … then ask your event stream for an insight.

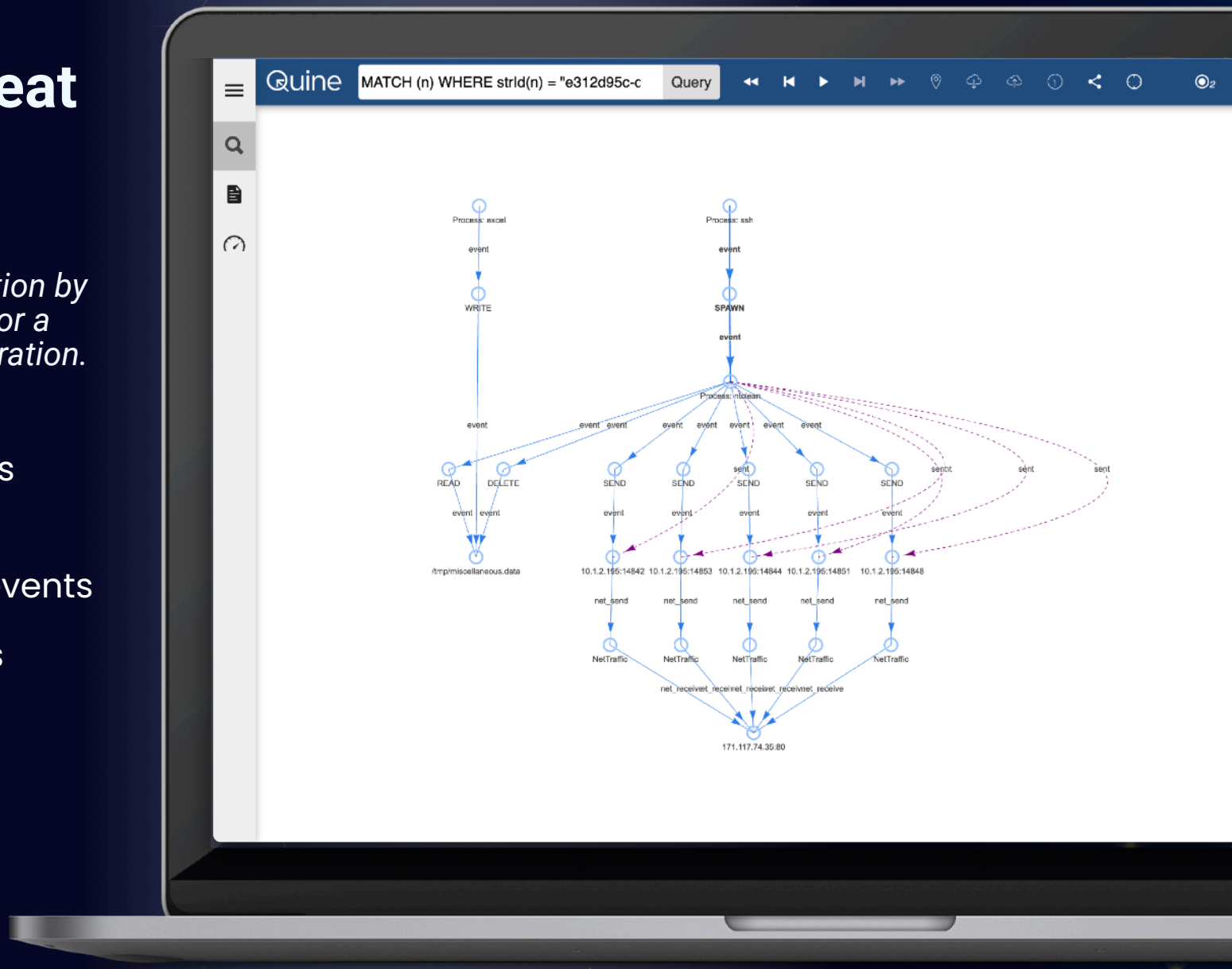
thatDot

# Advanced Persistent Threat

Streaming Event Scenario

*This cyber security scenario is monitoring for advanced persistent threats inside an organization by watching endpoint AND network data, looking for a typical pattern of behavior indicating data exfiltration.*

- Interleave logs from multiple sources
- Apply graph techniques to detect a known pattern in the stream of log events
- Produce an alert when the pattern is encountered
- Explore the event and take action



# Ingest Stream

*The ingest stream is where a streaming graph starts. It connects to data producers, transforms the data, then populates a streaming graph to be analyzed by standing queries.*

- Custom Cypher Query
- Ingested data passed as $that
- Consume a record and pull it apart into multiple nodes in the graph then string them together
- The built in function idFrom allows Quine to be fast and efficient
- Create nodes with MATCH
- Set properties
- Create the graph structure

```
ingestStreams:
  - type: FileIngest
    path: endpoint.json
    format:
      type: CypherJson
      query: |-
        MATCH (proc), (event), (object)
        WHERE id(proc) = idFrom($that.pid)
          AND id(event) = idFrom($that)
          AND id(object) = idFrom($that.object)

        SET proc.id = $that.pid,
            proc: Process

        SET event.type = $that.event_type,
            event.time = datetime({epochMillis: $that.time}),
            event: EndpointEvent

        SET object.data = $that.object

        CREATE (proc)-[:EVENT]->(event)-[:EVENT]->(object)
```

# Ingest Stream

*The ingest stream is where a streaming graph starts. It connects to data producers, transforms the data, then populates a streaming graph to be analyzed by standing queries.*

- Custom Cypher Query
- Ingested data passed as $that
- Consume a record and pull it apart into multiple nodes in the graph then string them together
- The built in function idFrom allows Quine to be fast and efficient
- Create nodes with MATCH
- Set properties
- Create the graph structure

```
ingestStreams:
  - type: FileIngest
    path: endpoint.json
    format:
      type: CypherJson
      query: |-
        MATCH (proc), (event), (object)
        WHERE id(proc) = idFrom($that.pid)
          AND id(event) = idFrom($that)
          AND id(object) = idFrom($that.object)

        SET proc.id = $that.pid,
            proc: Process

        SET event.type = $that.event_type,
            event.time = datetime({epochMillis: $that.time}),
            event: EndpointEvent

        SET object.data = $that.object

        CREATE (proc)-[:EVENT]->(event)-[:EVENT]->(object)
```

# Ingest Stream

*The ingest stream is where a streaming graph starts. It connects to data producers, transforms the data, then populates a streaming graph to be analyzed by standing queries.*

- Custom Cypher Query
- Ingested data passed as $that
- Consume a record and pull it apart into multiple nodes in the graph then string them together
- The built in function idFrom allows Quine to be fast and efficient
- Create nodes with MATCH
- Set properties
- Create the graph structure

```
22  ingestStreams:
23    - type: FileIngest
24      path: endpoint.json
25      format:
26        type: CypherJson
27        query: |-
28          MATCH (proc), (event), (object)
29          WHERE id(proc) = idFrom($that.pid)
30            AND id(event) = idFrom($that)
31            AND id(object) = idFrom($that.object)
32
33          SET proc.id = $that.pid,
34              proc: Process
35
36          SET event.type = $that.event_type,
37              event.time = datetime({epochMillis: $that.time}),
38              event: EndpointEvent
39
40          SET object.data = $that.object
41
42          CREATE (proc)-[:EVENT]->(event)-[:EVENT]->(object)
```

# Ingest Stream

*The ingest stream is where a streaming graph starts. It connects to data producers, transforms the data, then populates a streaming graph to be analyzed by standing queries.*

- Custom Cypher Query
- Ingested data passed as $that
- Consume a record and pull it apart into multiple nodes in the graph then string them together
- The built in function idFrom allows Quine to be fast and efficient
- Create nodes with MATCH
- Set properties
- Create the graph structure

```
22  ingestStreams:
23    - type: FileIngest
24      path: endpoint.json
25      format:
26        type: CypherJson
27        query: |-
28          MATCH (proc), (event), (object)
29          WHERE id(proc) = idFrom($that.pid)
30            AND id(event) = idFrom($that)
31            AND id(object) = idFrom($that.object)
32
33          SET proc.id = $that.pid,
34              proc: Process
35
36          SET event.type = $that.event_type,
37              event.time = datetime({epochMillis: $that.time}),
38              event: EndpointEvent
39
40          SET object.data = $that.object
41
42          CREATE (proc)-[:EVENT]->(event)-[:EVENT]->(object)
```

# Ingest Stream

*The ingest stream is where a streaming graph starts. It connects to data producers, transforms the data, then populates a streaming graph to be analyzed by standing queries.*

- Custom Cypher Query
- Ingested data passed as $that
- Consume a record and pull it apart into multiple nodes in the graph then string them together
- The built in function idFrom allows Quine to be fast and efficient
- Create nodes with MATCH
- Set properties
- Create the graph structure

```
22  ingestStreams:
23    - type: FileIngest
24      path: endpoint.json
25      format:
26        type: CypherJson
27        query: |-
28          MATCH (proc), (event), (object)
29          WHERE id(proc) = idFrom($that.pid)
30            AND id(event) = idFrom($that)
31            AND id(object) = idFrom($that.object)
32
33          SET proc.id = $that.pid,
34              proc: Process
35
36          SET event.type = $that.event_type,
37              event.time = datetime({epochMillis: $that.time}),
38              event: EndpointEvent
39
40          SET object.data = $that.object
41
42          CREATE (proc)-[:EVENT]->(event)-[:EVENT]->(object)
```

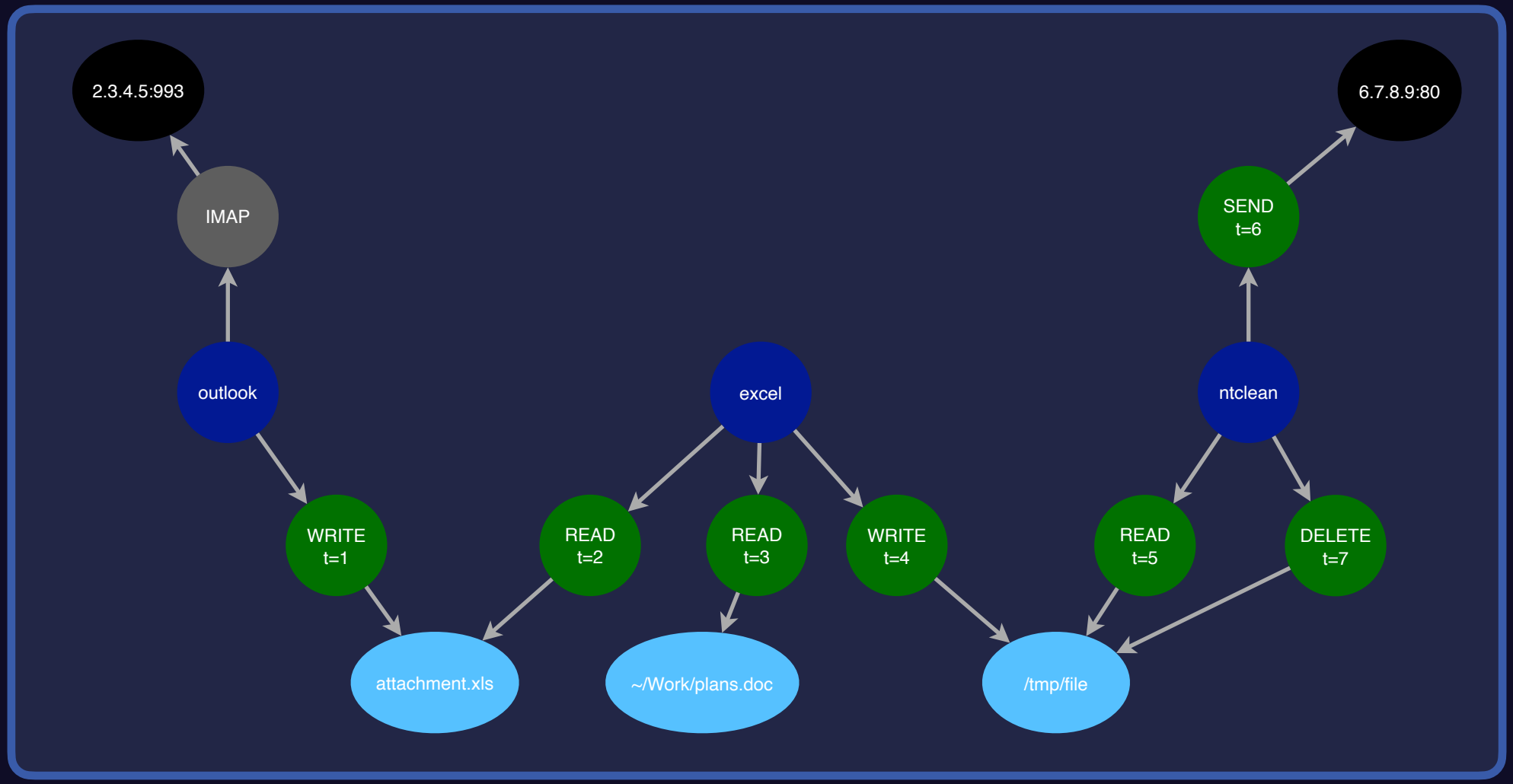
# Find, Report, and Understand

Understand first ... then ask your event stream for an insight.

thatDot

# Attack Graph

APT uses email to trigger private data exfiltration



# Standing Query

*Standing queries monitor streams for specified patterns, maintaining partial matches, and executing user-specified actions the instant a full match is made.*

- There are two parts to a standing query; the pattern match and the output query
- The pattern match finds patterns of interest in the graph
- Partial matches persist until a full match is made
- The match is passed to the output query in $that
- The output query forms a new event to be emitted to one or more output types

```
standingQueries:
  - pattern:
      type: Cypher
      query: |-
        MATCH (e1)-[:EVENT]->(f)<-[:EVENT]-(e2),
              (f)<-[:EVENT]-(e3)<-[:EVENT]-(p2)-[:EVENT]->(e4)
        WHERE e1.type = "WRITE"
          AND e2.type = "READ"
          AND e3.type = "DELETE"
          AND e4.type = "SEND"
        RETURN DISTINCT id(f) as id
    outputs:
      stolen-data:
        type: CypherQuery
        query: |-
          MATCH (p1)-[:EVENT]->(e1)-[:EVENT]->(f)<-[:EVENT]-(e2)<-[:EVENT]-(p2),
                (f)<-[:EVENT]-(e3)<-[:EVENT]-(p2)-[:EVENT]->(e4)-[:EVENT]->(ip)
          WHERE id(f) = $that.data.id
            AND e1.type = "WRITE"
            AND e2.type = "READ"
            AND e3.type = "DELETE"
            AND e4.type = "SEND"
            AND e1.time < e2.time
            AND e2.time < e3.time
            AND e2.time < e4.time
          WITH p1, p2, f, e1, e2, e3, e4, ip, "http://localhost:8080/#MATCH" + text.
          urlencode(" (p1),(p2),(f),(e1),(e2),(e3),(e4),(ip) WHERE id(p1)='"+strId(p1)
          +"' AND id(e1)='"+strId(e1)+"' AND id(f)='"+strId(f)+"' AND id(e2)='"+strId(e2)
          +"' AND id(p2)='"+strId(p2)+"' AND id(e3)='"+strId(e3)+"' AND id(e4)='"+strId
          (e4)+"' AND id(ip)='"+strId(ip)+"' RETURN p1, p2, f, e1, e2, e3, e4, ip") as
          URL
          RETURN URL
        andThen:
          type: WriteToKafka
          topic: results
          bootstrapServers: "localhost:9092"
          format:
            type: JSON
```

# Standing Query

*Standing queries monitor streams for specified patterns, maintaining partial matches, and executing user-specified actions the instant a full match is made.*

- There are two parts to a standing query; the pattern match and the output query
- The pattern match finds patterns of interest in the graph
- Partial matches persist until a full match is made
- The match is passed to the output query in $that
- The output query forms a new event to be emitted to one or more output types

```
standingQueries:
  - pattern:
      type: Cypher
      query: |-
        MATCH (e1)-[:EVENT]->(f)<-[:EVENT]-(e2),
              (f)<-[:EVENT]-(e3)<-[:EVENT]-(p2)-[:EVENT]->(e4)
        WHERE e1.type = "WRITE"
          AND e2.type = "READ"
          AND e3.type = "DELETE"
          AND e4.type = "SEND"
        RETURN DISTINCT id(f) as id
    outputs:
      stolen-data:
        type: CypherQuery
        query: |-
          MATCH (p1)-[:EVENT]->(e1)-[:EVENT]->(f)<-[:EVENT]-(e2)<-[:EVENT]-(p2),
                (f)<-[:EVENT]-(e3)<-[:EVENT]-(p2)-[:EVENT]->(e4)-[:EVENT]->(ip)
          WHERE id(f) = $that.data.id
            AND e1.type = "WRITE"
            AND e2.type = "READ"
            AND e3.type = "DELETE"
            AND e4.type = "SEND"
            AND e1.time < e2.time
            AND e2.time < e3.time
            AND e2.time < e4.time
          WITH p1, p2, f, e1, e2, e3, e4, ip, "http://localhost:8080/#MATCH" + text.
          urlencode(" (p1),(p2),(f),(e1),(e2),(e3),(e4),(ip) WHERE id(p1)='"+strId(p1)
          +"' AND id(e1)='"+strId(e1)+"' AND id(f)='"+strId(f)+"' AND id(e2)='"+strId(e2)
          +"' AND id(p2)='"+strId(p2)+"' AND id(e3)='"+strId(e3)+"' AND id(e4)='"+strId
          (e4)+"' AND id(ip)='"+strId(ip)+"' RETURN p1, p2, f, e1, e2, e3, e4, ip") as
          URL
          RETURN URL
        andThen:
          type: WriteToKafka
          topic: results
          bootstrapServers: "localhost:9092"
          format:
            type: JSON
```

# Standing Query

*Standing queries monitor streams for specified patterns, maintaining partial matches, and executing user-specified actions the instant a full match is made.*

- There are two parts to a standing query; the pattern match and the output query
- The pattern match finds patterns of interest in the graph
- Partial matches persist until a full match is made
- The match is passed to the output query in $that
- The output query forms a new event to be emitted to one or more output types

```
standingQueries:
  - pattern:
      type: Cypher
      query: |-
        MATCH (e1)-[:EVENT]->(f)<-[:EVENT]-(e2),
              (f)<-[:EVENT]-(e3)<-[:EVENT]-(p2)-[:EVENT]->(e4)
        WHERE e1.type = "WRITE"
          AND e2.type = "READ"
          AND e3.type = "DELETE"
          AND e4.type = "SEND"
        RETURN DISTINCT id(f) as id
    outputs:
      stolen-data:
        type: CypherQuery
        query: |-
          MATCH (p1)-[:EVENT]->(e1)-[:EVENT]->(f)<-[:EVENT]-(e2)<-[:EVENT]-(p2),
                (f)<-[:EVENT]-(e3)<-[:EVENT]-(p2)-[:EVENT]->(e4)-[:EVENT]->(ip)
          WHERE id(f) = $that.data.id
            AND e1.type = "WRITE"
            AND e2.type = "READ"
            AND e3.type = "DELETE"
            AND e4.type = "SEND"
            AND e1.time < e2.time
            AND e2.time < e3.time
            AND e2.time < e4.time
          WITH p1, p2, f, e1, e2, e3, e4, ip, "http://localhost:8080/#MATCH" + text.
          urlencode(" (p1),(p2),(f),(e1),(e2),(e3),(e4),(ip) WHERE id(p1)='"+strId(p1)
          +"' AND id(e1)='"+strId(e1)+"' AND id(f)='"+strId(f)+"' AND id(e2)='"+strId(e2)
          +"' AND id(p2)='"+strId(p2)+"' AND id(e3)='"+strId(e3)+"' AND id(e4)='"+strId
          (e4)+"' AND id(ip)='"+strId(ip)+"' RETURN p1, p2, f, e1, e2, e3, e4, ip") as
          URL
          RETURN URL
        andThen:
          type: WriteToKafka
          topic: results
          bootstrapServers: "localhost:9092"
          format:
            type: JSON
```

# Standing Query

*Standing queries monitor streams for specified patterns, maintaining partial matches, and executing user-specified actions the instant a full match is made.*

- There are two parts to a standing query; the pattern match and the output query
- The pattern match finds patterns of interest in the graph
- Partial matches persist until a full match is made
- The match is passed to the output query in $that
- The output query forms a new event to be emitted to one or more output types

```
standingQueries:
  - pattern:
      type: Cypher
      query: |-
        MATCH (e1)-[:EVENT]->(f)<-[:EVENT]-(e2),
              (f)<-[:EVENT]-(e3)<-[:EVENT]-(p2)-[:EVENT]->(e4)
        WHERE e1.type = "WRITE"
          AND e2.type = "READ"
          AND e3.type = "DELETE"
          AND e4.type = "SEND"
        RETURN DISTINCT id(f) as id
    outputs:
      stolen-data:
        type: CypherQuery
        query: |-
          MATCH (p1)-[:EVENT]->(e1)-[:EVENT]->(f)<-[:EVENT]-(e2)<-[:EVENT]-(p2),
                (f)<-[:EVENT]-(e3)<-[:EVENT]-(p2)-[:EVENT]->(e4)-[:EVENT]->(ip)
          WHERE id(f) = $that.data.id
            AND e1.type = "WRITE"
            AND e2.type = "READ"
            AND e3.type = "DELETE"
            AND e4.type = "SEND"
            AND e1.time < e2.time
            AND e2.time < e3.time
            AND e2.time < e4.time
          WITH p1, p2, f, e1, e2, e3, e4, ip, "http://localhost:8080/#MATCH" + text.
          urlencode(" (p1),(p2),(f),(e1),(e2),(e3),(e4),(ip) WHERE id(p1)='"+strId(p1)
          +"' AND id(e1)='"+strId(e1)+"' AND id(f)='"+strId(f)+"' AND id(e2)='"+strId(e2)
          +"' AND id(p2)='"+strId(p2)+"' AND id(e3)='"+strId(e3)+"' AND id(e4)='"+strId
          (e4)+"' AND id(ip)='"+strId(ip)+"' RETURN p1, p2, f, e1, e2, e3, e4, ip") as
          URL
          RETURN URL
        andThen:
          type: WriteToKafka
          topic: results
          bootstrapServers: "localhost:9092"
          format:
            type: JSON
```

# Standing Query

*Standing queries monitor streams for specified patterns, maintaining partial matches, and executing user-specified actions the instant a full match is made.*

- There are two parts to a standing query; the pattern match and the output query
- The pattern match finds patterns of interest in the graph
- Partial matches persist until a full match is made
- The match is passed to the output query in $that
- The output query forms a new event to be emitted to one or more output types

```
standingQueries:
  - pattern:
      type: Cypher
      query: |-
        MATCH (e1)-[:EVENT]->(f)<-[:EVENT]-(e2),
              (f)<-[:EVENT]-(e3)<-[:EVENT]-(p2)-[:EVENT]->(e4)
        WHERE e1.type = "WRITE"
          AND e2.type = "READ"
          AND e3.type = "DELETE"
          AND e4.type = "SEND"
        RETURN DISTINCT id(f) as id
    outputs:
      stolen-data:
        type: CypherQuery
        query: |-
          MATCH (p1)-[:EVENT]->(e1)-[:EVENT]->(f)<-[:EVENT]-(e2)<-[:EVENT]-(p2),
                (f)<-[:EVENT]-(e3)<-[:EVENT]-(p2)-[:EVENT]->(e4)-[:EVENT]->(ip)
          WHERE id(f) = $that.data.id
            AND e1.type = "WRITE"
            AND e2.type = "READ"
            AND e3.type = "DELETE"
            AND e4.type = "SEND"
            AND e1.time < e2.time
            AND e2.time < e3.time
            AND e2.time < e4.time
          WITH p1, p2, f, e1, e2, e3, e4, ip, "http://localhost:8080/#MATCH" + text.
          urlencode(" (p1),(p2),(f),(e1),(e2),(e3),(e4),(ip) WHERE id(p1)='"+strId(p1)
          +"' AND id(e1)='"+strId(e1)+"' AND id(f)='"+strId(f)+"' AND id(e2)='"+strId(e2)
          +"' AND id(p2)='"+strId(p2)+"' AND id(e3)='"+strId(e3)+"' AND id(e4)='"+strId
          (e4)+"' AND id(ip)='"+strId(ip)+"' RETURN p1, p2, f, e1, e2, e3, e4, ip") as
          URL
          RETURN URL
        andThen:
          type: WriteToKafka
          topic: results
          bootstrapServers: "localhost:9092"
          format:
            type: JSON
```

# DEMO

# Summary

- Graphs are for streaming now!
- Quine stores persistent data in Cassandra for performance and resilience
- Scale past 1,000,000 events per second
- Dramatically simplify your efforts building streaming data pipelines

Try Quine Open Source @ quine.io

# Turn high-volume **event streams** into high-value **insights**