Graph of Enterprise Metadata– Powered by Neo4J & Spark

Vladimir Bacvanski
Deepak Chandramouli

October 20

[NODES 2020]

Neo4j Online Developer Expo and Summit

neo4j

Knowledge Graphs

4:35 - 5:20

https://neo4j.com/nodes-2020/

Mastering Enterprise Metadata with Neo4j

Modern enterprises not only have a myriad of data sources, from real-time events, transactional, Big Data, and many other systems, but they also boast a rich ecosystem of thousands of APIs & treasure of deep technical metadata. How do you organize and gain insights from all of this? In addition, there is a trove of data coming from other sources such as millions of datasets, SQL queries, slack chats, thousands user hierarchies, orgs & locations, access controls, Wiki pages, JIRA tickets and more. Normally, these sources are all disconnected from each other, and valuable insights are missed.

At PayPal, we are implementing GEM: the Graph of Enterprise Metadata, a system that connects and puts all the critical metadata under one umbrella. GEM is built on top of Neo4j and Apache Spark and sports a range of metadata ingestion components. GEM manages a rich graph of entities and connections, it applies graph algorithms for analysis and recommendations. And in the future - GEM would apply ML model to derive insights. These help answer critical questions around data catalog, security, and governance initiatives for systems supporting financial transactions for our 346 millions of users. In addition, we envision this graph of enterprise metadata to empower PayPal at scale & accelerate the journey of reaching 1 Billion Customers.

Dr. Vladimir Bacvanski

Principal Architect, PayPal

Dr. Vladimir Bacvanski is a Principal Architect with Strategic Architecture at PayPal. He is the lead architect for Privacy and Developer Experience. Before joining PayPal, Vladimir was the CTO and founder of a custom development and consulting firm. He is the author of the popular O'Reilly course "Introduction to Big Data" and a coauthor of the O'Reilly course on Kafka. Vladimir received a PhD degree in Computer Science from Aachen University of Technology in Germany.

Deepak Chandramouli

Software Engineering | Enterprise Data Platforms, PayPal

Deepak Chandramouli is an Engineering Lead in PayPal's Enterprise Data Platforms Organization. Deepak currently manages the engineering for products - UDC (Unified Data Catalog) and Gimel.io (Apache Spark based Data Abstraction Layer). Deepak incubated Gimel and helped open source it. More recently, Deepak is focusing on building scalable Data Catalog in the context of emerging Data Governance & Regulatory demands. Deepak's prior speaking experiences at conferences include:

PayPal

AGENDA

- Prelude
- Introduction
- Architecture & Flow
- Scaling for enterprise
- Future
- Questions

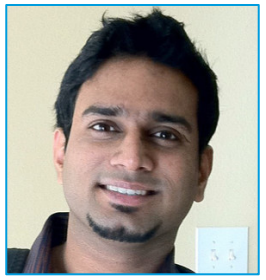
About us

Vladimir Bacvanski

vbacvanski@paypal.com

Twitter: @OnSoftware

- Principal Architect, Strategic Architecture at PayPal
- In previous life: CTO of a development and consulting firm
- PhD in Computer Science from RWTH Aachen, Germany
- O'Reilly author: Courses on Big Data, Kafka



Deepak Chandramouli

dmohanakumarchan@paypal.com

LinkedIn: @deepakmc

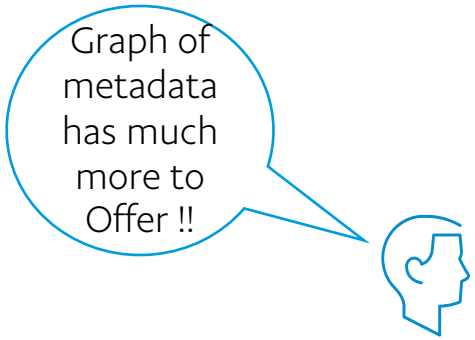
- MT2 Software Engineer, Data Platform Services at PayPal
- Data Enthusiast
- Tech lead
  - Gimel (Big Data Framework for Apache Spark)
  - Unified Data Catalog – PayPal's Enterprise Data Catalog

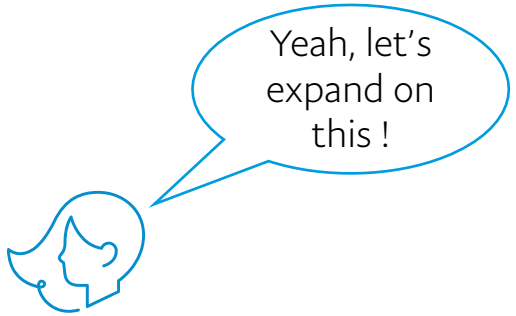
PayPal

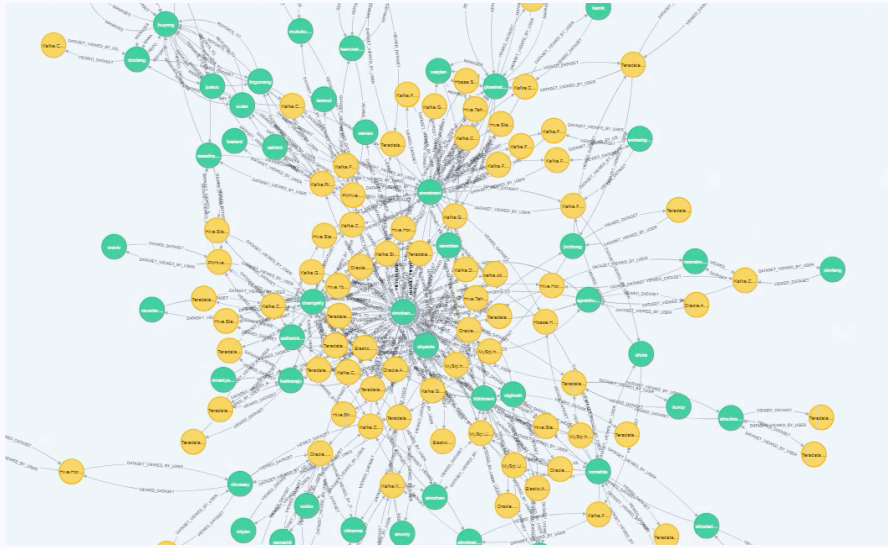
Prelude

2019 | Recommendation System – Unified Data Catalog

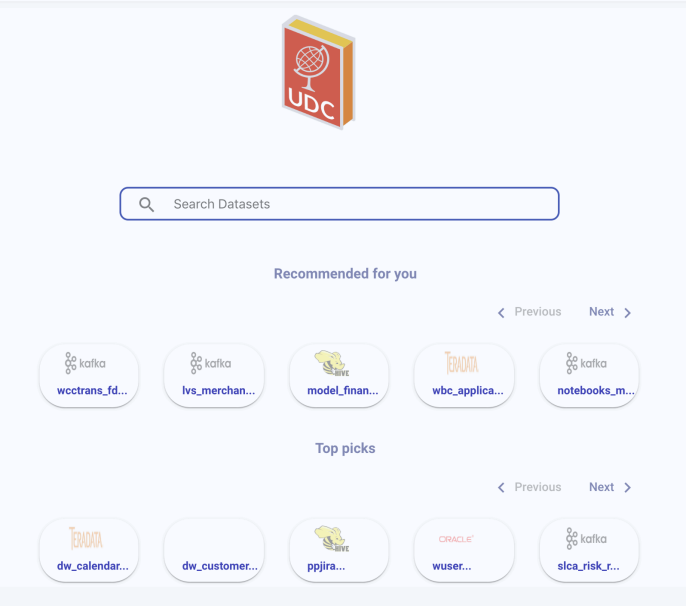
Building Recommendations for a Data Catalog







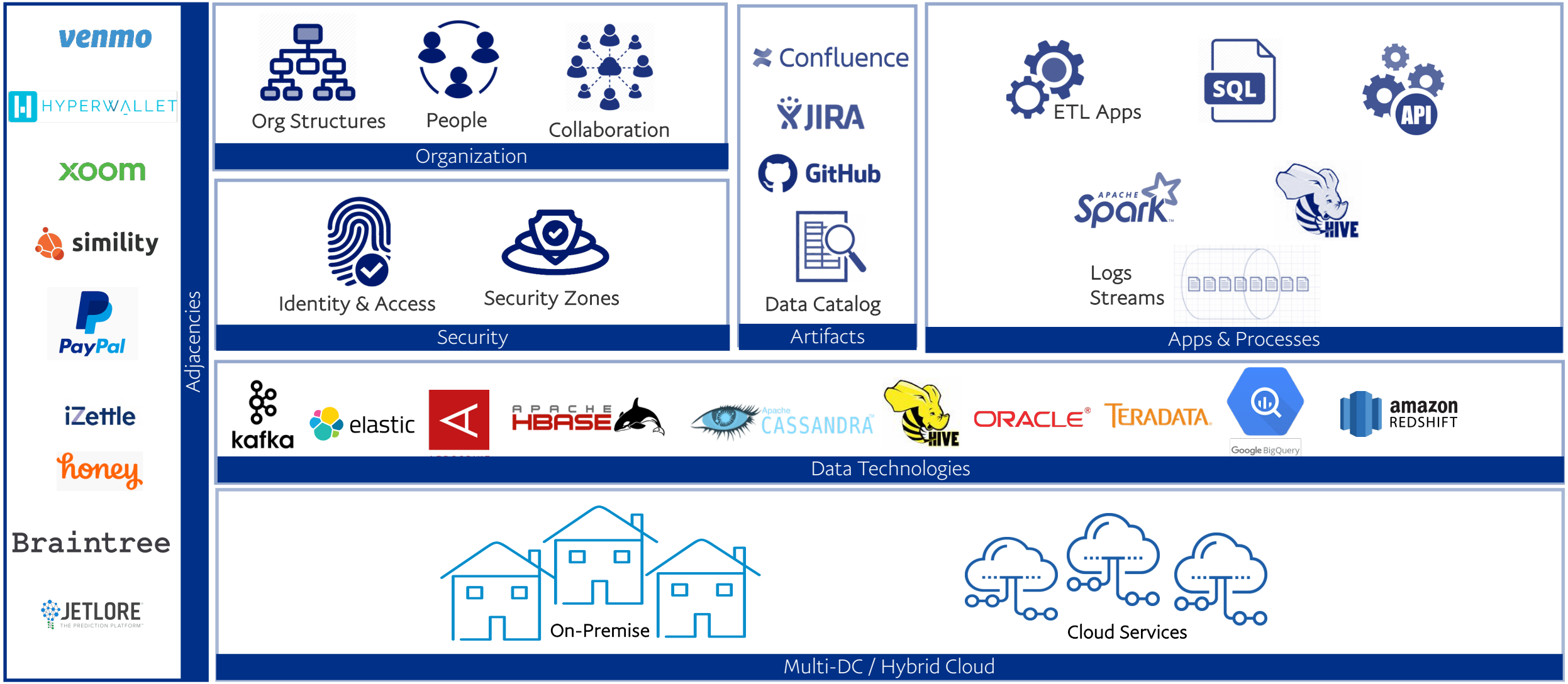
ABSTRACTION LAYER



PayPal

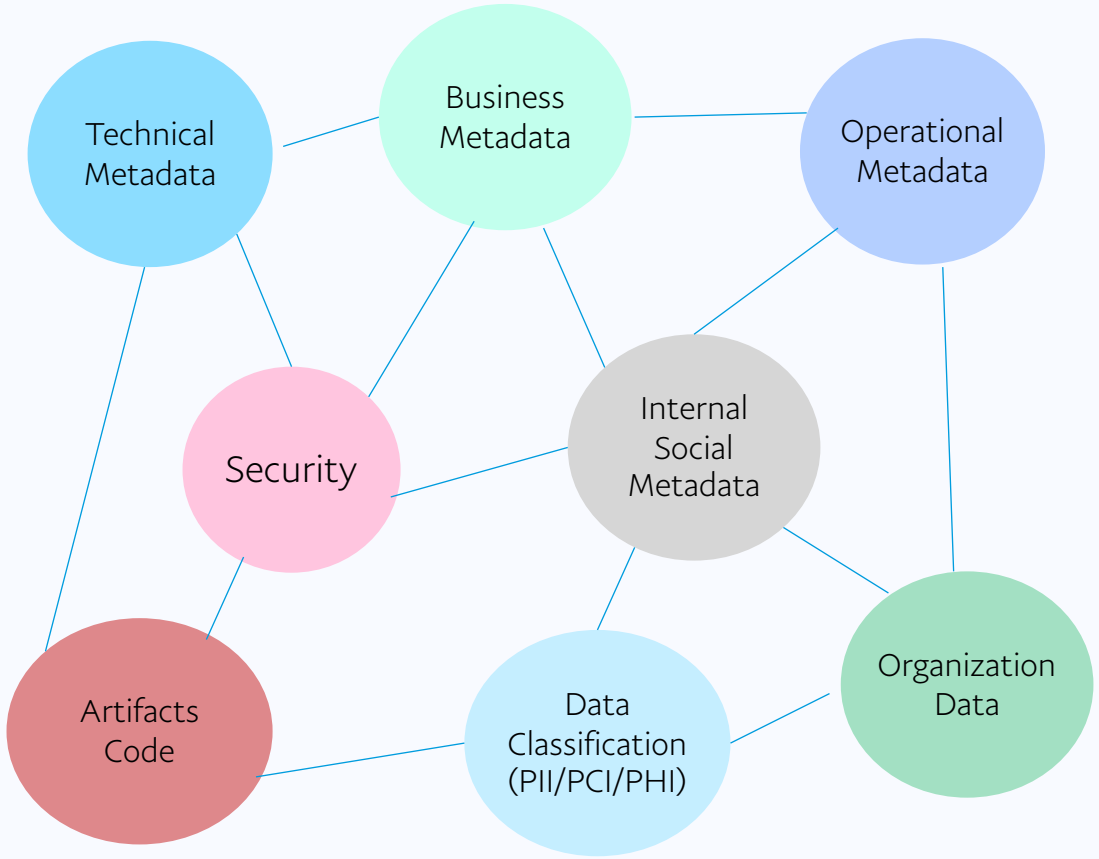
Graph of Enterprise Metadata Introduction

The Enterprise Landscape



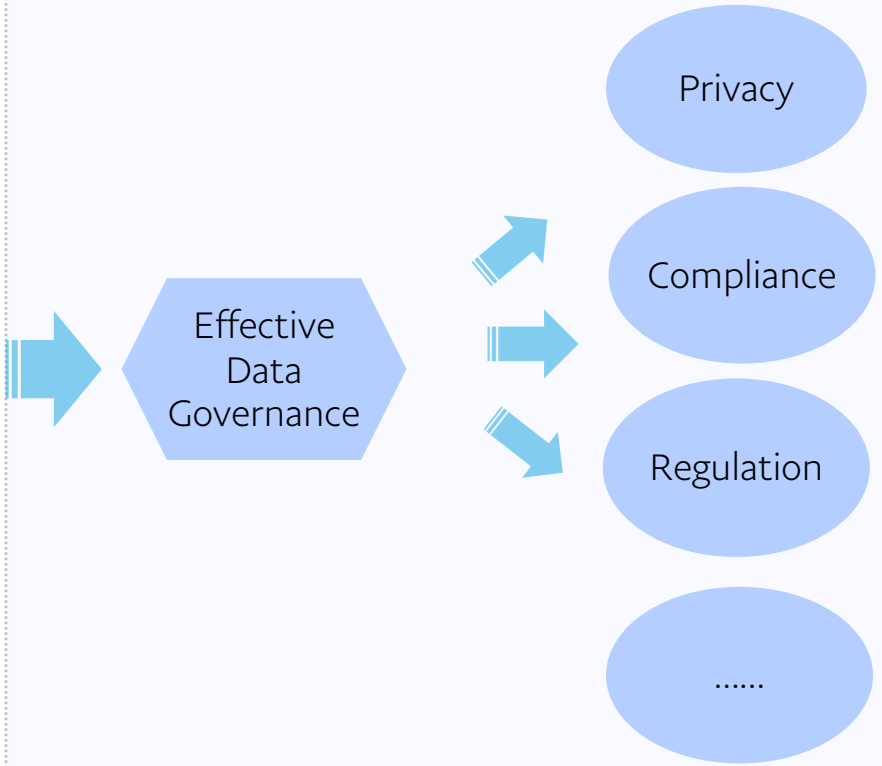
Connecting the Dots...

Connected Components



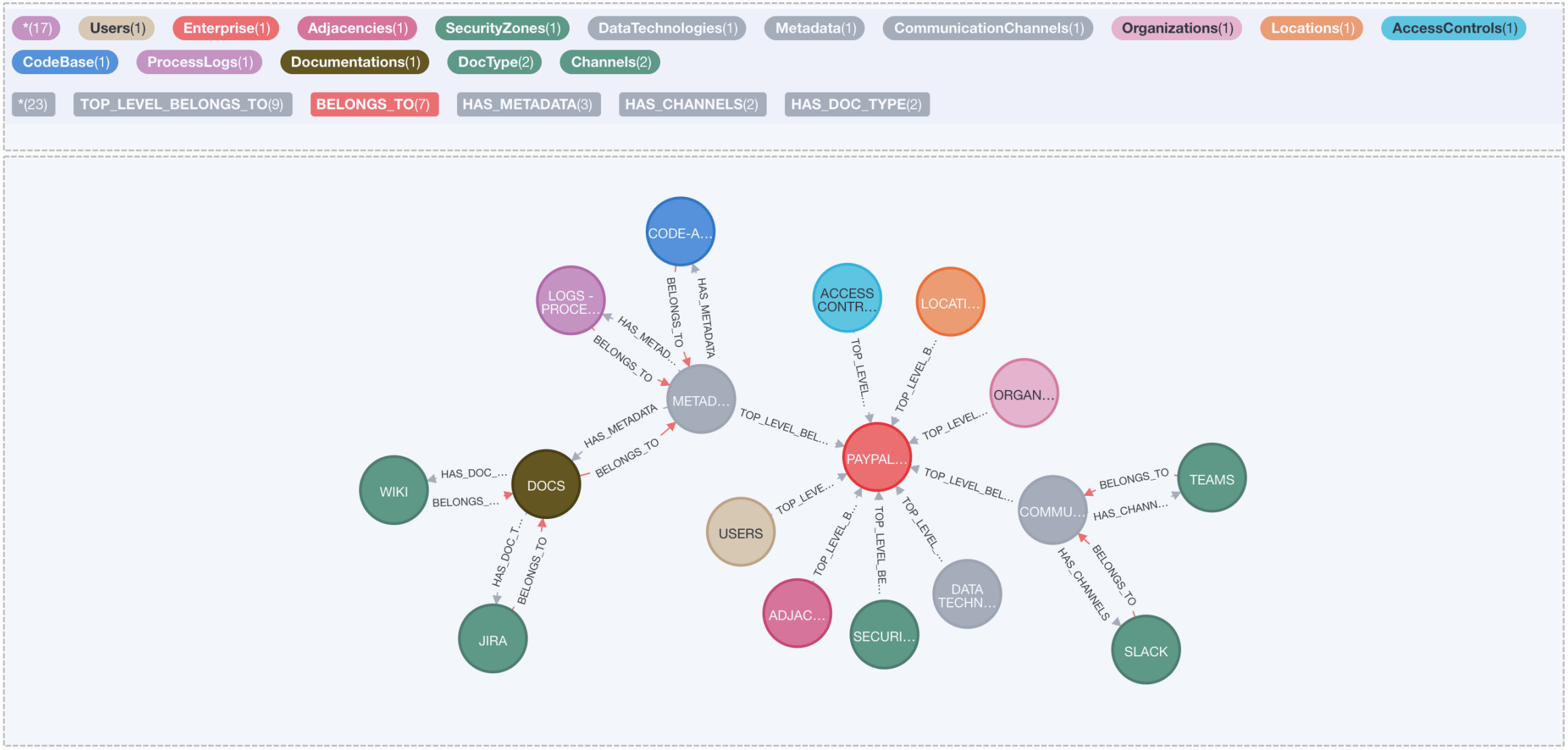
The whole is greater !

Efficiency.



PayPal

Connected components = Graph of Enterprise Metadata

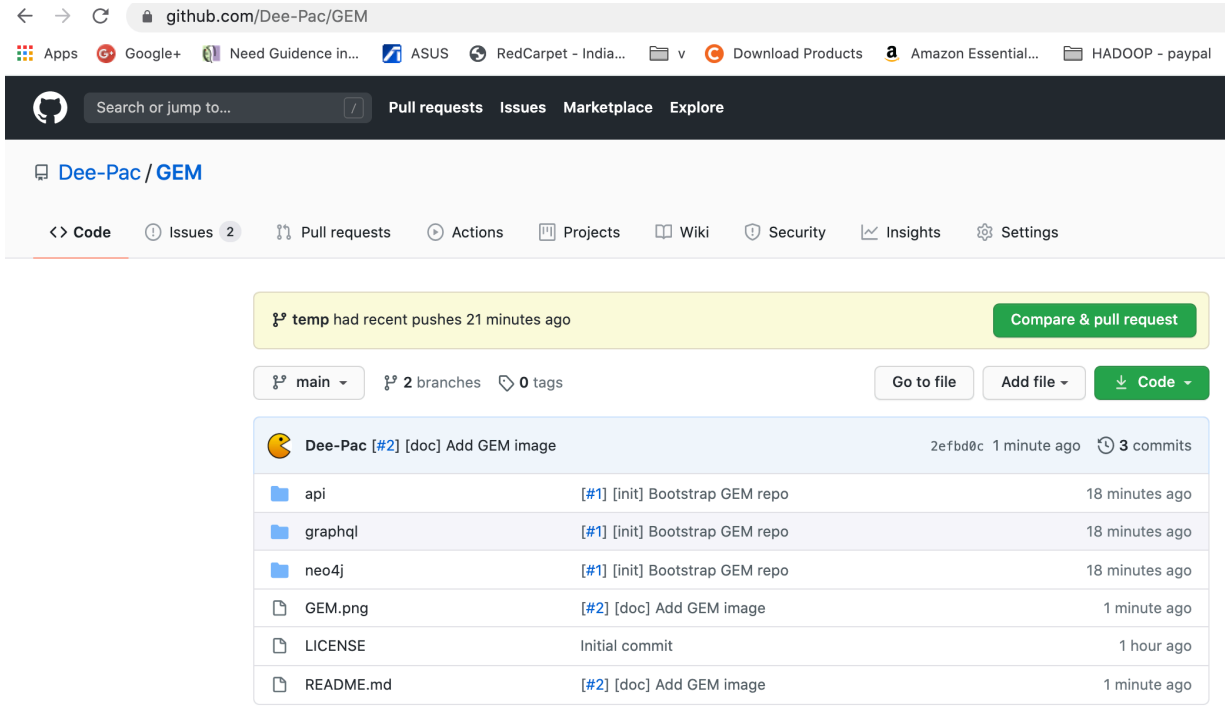


PayPal

So What → Top Down View !

Demo & Sandbox Code

Code Base | Playground



Github

https://github.com/Dee-Pac/GEM

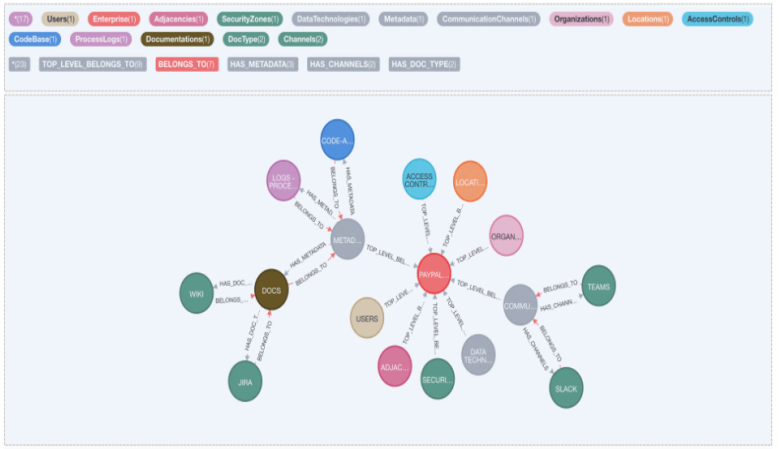
Gitter

https://gitter.im/graph_of_enterprise_metadata/GEM-Ask-Us-Anything

README.md

GEM | Graph of Enterprise Metadata

Code base for the proof of concept on the Graph of Enterprise Metadata



Quick Start

- Pull the repo
- Start neo4j
- Start GraphQl
- Bootstrap GEM in neo4j
- Play with GEM on neo4j
- Play with GEM via GraphQL

PayPal

Graph Build Architecture & Data Flow

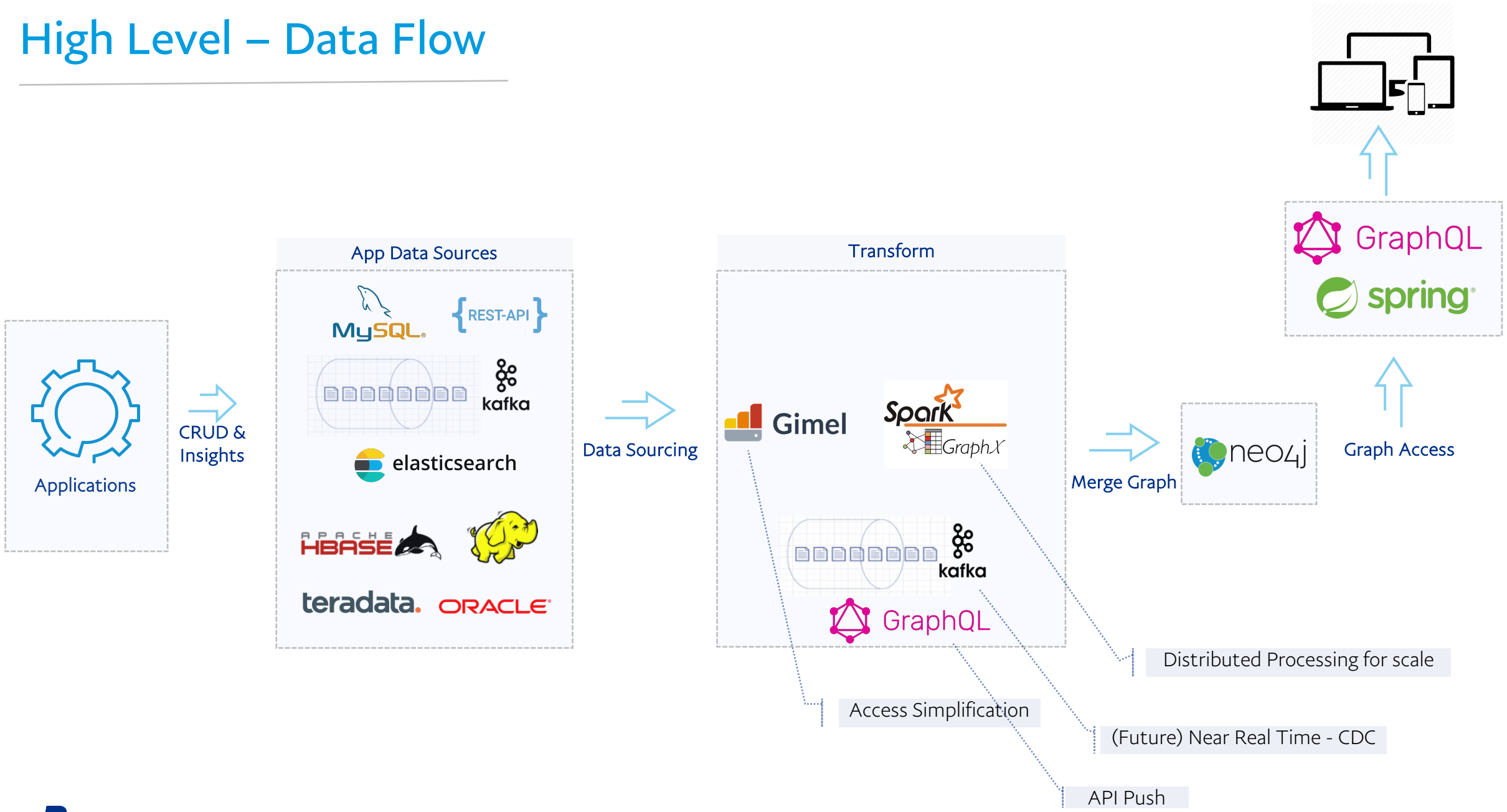
Metadata Data Sources...

- Technical Data Catalog
  - Data Stores
  - Databases / Containers
  - Tables / Objects
  - Columns / Fields
  - Business Metadata / Glossary / Annotation
- Data Classification
  - PII / PCI / PHI
- Org Structure
  - People, Geo , Hierarchies
- IAM
  - Access controls
  - Owners, Users

- Artifacts
  - Code base – GitHub/SQL
  - Wiki – Documents, Mentions
  - JIRA – Issues, Mentions
- Operational Metadata
  - Jobs & Apps
  - Databases & App – Query Logs
- Social Metadata (Internal)
  - Slack Chats, Threads, Annotations

PayPal

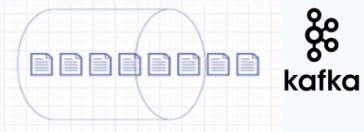
High Level – Data Flow



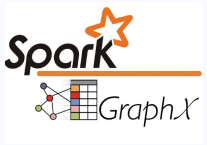
PayPal

neo4j + Apache Spark + Gimel + GraphQL

Multiple Sources Systems



Data Processing



- Distributed Processing - SCALE
- Spark-ML + Graphx. - COMPLEXITY
- GIMEL + Spark - Data access simplified
- Kafka Streams – Near Real Time CDC in the future

Connected Components

- Easy Bootstrap
- Cypher & APOCs
- Robust libraries - Spark, GraphQL
- Rich Graph Algorithms

Unified Access Pattern

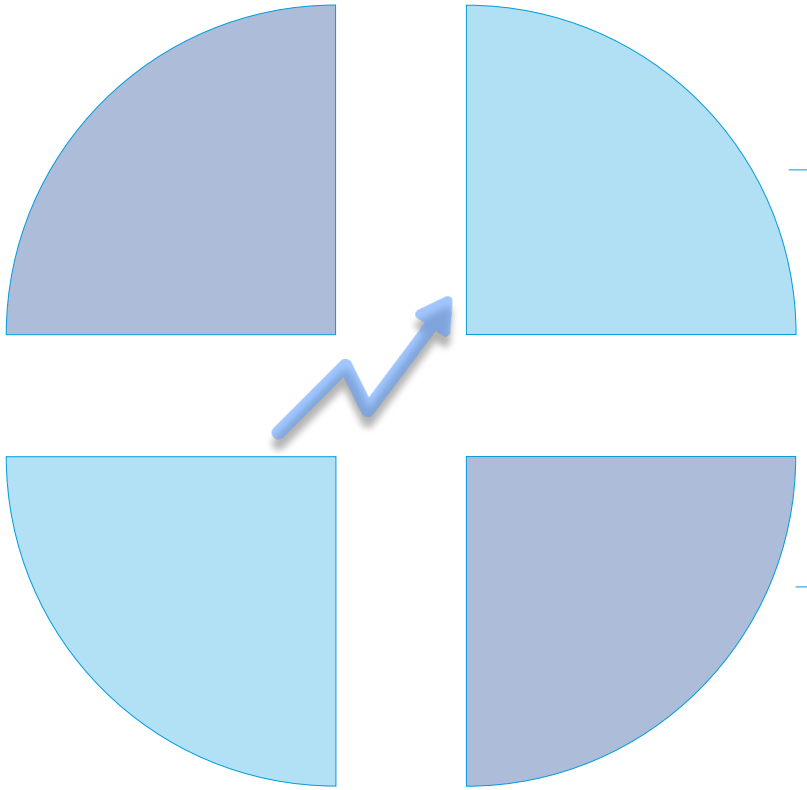
- Abstracts Graph Complexity
- Supports - Multiple Front-end clients
- Accelerates UI Integration
- Reference | neo4j & GraphQL
- Spring-Data-neo4j [Scale]

<<<<<< Data Processing @ Scale >>>>>>

Metadata Growth

Technical Metadata

>1000s of Data Stores
>100K Containers
>7 Million Datasets
>50 Million Fields
>10K PII/PCI Class elements



Security & Organization

>1000's of IAM roles
>20K Internal Users & Role Owners

Operational Metadata

>100K Data Apps Per Day
>25 Data Tech Stacks
>API, Streams & Batch Apps

Artifacts & Social Metadata,

Artifacts
>10K Scripts & App code
>1000s of Wiki, JIRA Annotations

Data Social
~ 12 Active Analytics Channel in Slack
~ Evolving Teams conversations for data

Gimel + spark + neo4j

Start Spark Session With Gimel Libraries

Add neo4j drivers

```
%%configure -f
{
        "conf":{"spark.neo4j.bolt.url":"bolt://neo4j_host:7687",
                "spark.neo4j.bolt.user":"neo4j",
                "spark.neo4j.bolt.password":"xxx",
                "spark.driver.extraClassPath":"neo4j-spark-connector-2.2.1-M5.jar:gimel-library.jar",
                "spark.executor.extraClassPath":"neo4j-spark-connector-2.2.1-M5.jar:gimel-library.jar",
                "spark.jars":"hdfs:///user/xxx/neo4j-spark-connector-2.2.1-M5.jar,hdfs://gimel-library.jar"
                }
}
```

NEO4J Connector

GIMEL Data API

Initialize

```
import org.neo4j.spark._
import org.graphframes._
import org.apache.spark.graphx._
import org.apache.spark.graphx.lib._


val neo = Neo4j(sc)
val gsql = com.paypal.gimel.scaas.GimelQueryProcessor.executeBatch(_:String,spark)
val dataset = com.paypal.gimel.DataSet(spark)
```

Gimel + spark + neo4j (Continued..)

```
// Connector Configuration

val dataSetProperties_kafka = s"""
  {
      "datasetType": "kafka",
      "fields": [],
      "partitionFields": [],
      "props": {
            "gimel.storage.type":"kafka",
            "datasetName":"query_logs",
            "auto.offset.reset":"earliest",
            "gimel.kafka.checkpoint.zookeeper.host":"zk1:2181,zk2:2181",
            "gimel.kafka.avro.schema.source.url":"Defaults",
            "gimel.kafka.avro.schema.source.wrapper.key":"Defaults",
            "gimel.kafka.avro.schema.source":"INLINE",
            "zookeeper.connection.timeout.ms":"3000",
            "bootstrap.servers":"broker1:9092,broker2:9092,broker4:9092",
            "gimel.kafka.checkpoint.zookeeper.path":"/kafka_consumer/checkpoint",
            "key.deserializer":"org.apache.kafka.common.serialization.StringDeserializer",
            "value.deserializer":"org.apache.kafka.common.serialization.StringDeserializer",
            "key.serializer":"org.apache.kafka.common.serialization.StringSerializer",
            "value.serializer":"org.apache.kafka.common.serialization.StringSerializer",
            "gimel.kafka.whitelist.topics":"kafka_topic_name",
            "gimel.kafka.message.value.type":"json",
            "gimel.kafka.api.version":"1"
        }
  }"""

// Catalog Provider can be HIVE, USER or external Catalog

spark.sql("set gimel.catalog.provider=USER")

// Read Data via Unified API

val options = Map("query_logs.dataSetProperties" -> dataSetProperties_kafka)
val df = dataset.read("query_logs",options)
df.show
```

Unified Connector Config

```
// Connector Configuration

val dataSetProperties_elastic = s"""
  {
      "datasetType": "ELASTIC_SEARCH",
      "fields": [],
      "partitionFields": [],
      "props": {
            "datasetName":"search_logs",
            "es.port":"9200",
            "es.nodes":"http://elastic_host",
            "gimel.storage.type":"ELASTIC_SEARCH",
            "gimel.storage.version":"6",
            "gimel.es.index.partition.list":"202009,202010",
            "es.mapping.date.rich":"false",
            "es.resource":"search/log",
            "gimel.es.index.partition.isEnabled":"true",
            "gimel.es.index.partition.delimiter":"_",
            "es.read.field.exclude":"appStartTseaime,appEndTime,jobStartTime,jobEndTime",
            "es.read.field.as.array.include":"columns,logTime"
        }
  }"""

// Catalog Provider can be HIVE, USER or external Catalog

spark.sql("set gimel.catalog.provider=USER")

// Read Data via Unified API

val options = Map("search_logs.dataSetProperties" -> dataSetProperties_elastic)
val df = dataset.read("search_logs",options)
df.show
```

Unified Data API

PayPal

Gimel + spark + neo4j (Continued..)

```
// Connector Configuration

val dataSetProperties_mysql = s"""
  {   "datasetName":"table_metadata_details",
      "datasetType": "JDBC",
      "fields": [],
      "partitionFields": [],
      "props": {
        "gimel.jdbc.p.strategy":"file",
        "gimel.jdbc.p.file":"hdfs_password_file",
        "gimel.jdbc.username":"mysql_user",
         "gimel.jdbc.input.table.name":"schema.table_name",
        "gimel.storage.type":"JDBC",
        "gimel.jdbc.url":"jdbc:mysql://mysql_host:3115/database?useSSL=true&requireSSL=true&
        "gimel.jdbc.driver.class":"com.mysql.jdbc.Driver"
         }
  }"""

// Catalog Provider can be HIVE, USER or external Catalog

spark.sql("set gimel.catalog.provider=USER")

// Read Data via Unified API

val options = Map("table_metadata_details.dataSetProperties" -> dataSetPrope
val df = dataset.read("table_metadata_details",options)
df.show
```

Unified Data API

Nodes & Relationship

User --> Search String

```
Neo4jDataFrame.mergeEdgeList(sc,
                             udc_user_searches,
                             ("user",Seq("username")),
                             ("SEARCHED",Seq("search_repeated","first_search_date","last_search_date")),
                             ("search",Seq("search_string"))
                            )
```

Dataset --> Viewed by User

```
Neo4jDataFrame.mergeEdgeList(sc,
                             udc_user_views,
                             ("dataset",Seq("storage_dataset_name")),
                             ("DATASET_VIEWED_BY_USER",Seq("view_repeated","first_viewed_date","last_viewed_date")),
                             ("user",Seq("username"))
                            )
```

Exposing Graph to Clients

Cypher -> GraphQL

Cypher

```
neo4j$ MATCH (d:dataset)-[:DATASET_HAS_COLUMN]-(c:column) RETURN d,c
```

Define GraphQL Schema

```
type column {
  column:String!
  storage_system_name:String!
  dataset:String!
}

type dataset {
   dataset:String!
   storage_system_name:String!
   column: [column]
@relation(name:"DATASET_HAS_COLUMN", direction:OUT)
}

type Enterprise {
   name:String!
}
```

GraphQL Query

```
# GraphQL Query - Dataset & Columns
{
  dataset {
    dataset
    column {
      column
    }
    storage_system_name
  }
}
```

GraphQL Query - Response


```
{
  "data": {
    "dataset": [
      {
        "dataset": "edw.customer_profile",
        "column": [
          {
            "column": "total_transaction_count"
          },
          {
            "column": "name"
          },
          {
            "column": "country"
          },
          {
            "column": "customer_id"
          },
          {
            "column": "total_transaction_amt"
          },
          {
            "column": "total_logins"
          }
        ],
        "storage_system_name": "Hive.1"
      },
```


PayPal

Conclusion & Next Steps

Key Take-aways

- **The whole is greater than Sum !**
- **Data Awareness**: Data can be anywhere across the ecosystem : but we should know its shape & form of existence.
- **Graphs are everywhere**: Full potential can be realized by connecting the dots - across board.
- **Enriching metadata**: It is a collective effort; silos rarely achieve effectiveness.
- **Graph's Effectiveness**: expose the data in a consumable way for – all tech stacks alike !

What's next?

- Technical Data Catalog
  - Data Stores
  - Databases / Containers
  - Tables / Objects
  - Columns / Fields
  - Business Metadata / Glossary / Annotation
- Data Classification
  - PII / PCI / PHI
- IAM
  - Access controls
  - Owners, Users
- Org Structure
  - People, Geo , Hierarchies

Connecting Further…

- Artifacts
  - Code base – GitHub/SQL
  - Wiki – Documents, Mentions
  - JIRA – Issues, Mentions
- Operational Metadata
  - Jobs & Apps
  - Databases & App – Query Logs
  - Lineage
- Social Metadata (Internal)
  - Slack Chats, Threads, Annotations

Foundational Capabilities…

Scale – Millions of Nodes X Edges

Graph – Data Security & Multitenancy

Access - Simplification

Insights driven by –

Graph / ML Algorithms

PayPal

Questions?

Code base – GITHUB_FOR_GEM
Let's collaborate - GITTER_FOR_GEM

Thank You!