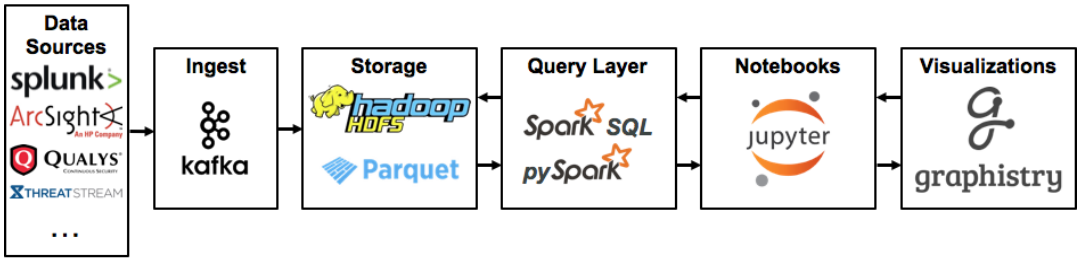
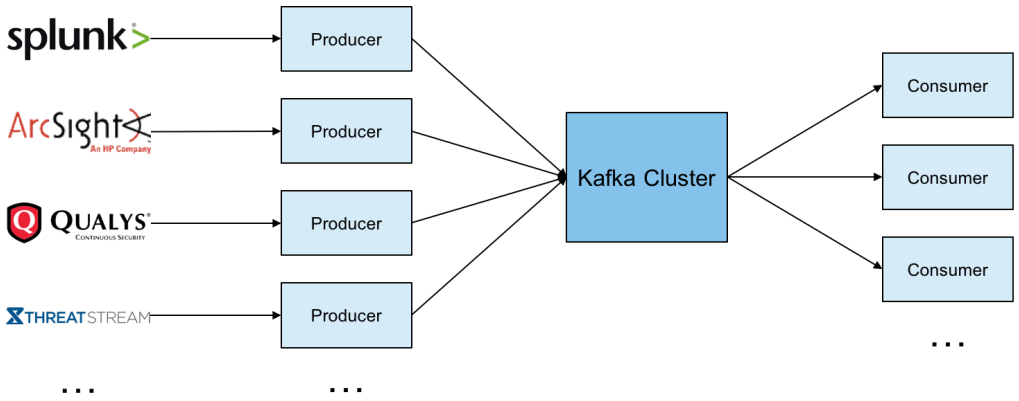
Architecture Documentation

- **Install**
  - Kafka
    - https://kafka.apache.org/documentation.html#quickstart
  - Hadoop / HDFS
    - https://hadoop.apache.org/docs/r2.7.2/hadoop-project-dist/hadoop-common/ClusterSetup.html
  - Spark
    - http://spark.apache.org/docs/latest/
  - JupyterHub / Jupyter
    - https://github.com/jupyterhub/jupyterhub/blob/master/README.md
    - http://jinyi.me/2015/12/Setting-up-IPython-Jupyterhub-for-pyspark.htm
  - Graphistry
    - https://github.com/graphistry/pygraphistry#installation

- **Architecture**

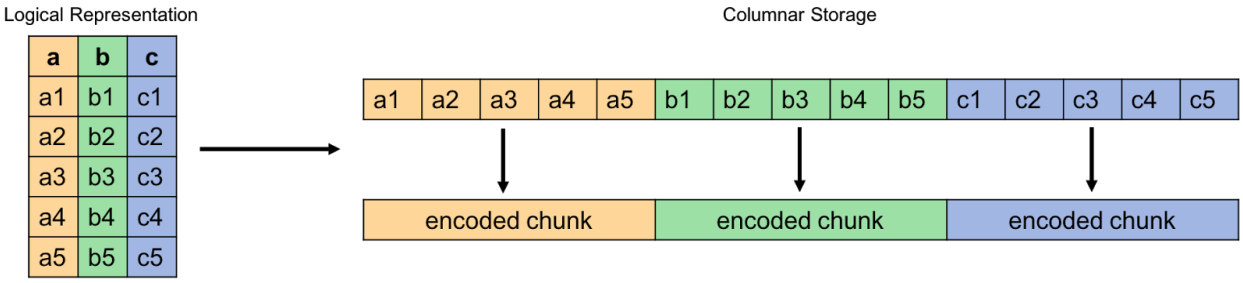


  - **Kafka**
    - Kafka is a distributed publish-subscribe messaging queue that is extremely fast, scalable, and durable



        - At a lower level, Kafka is composed of Producers, Brokers, and Consumers
            - Producers send messages to the Kafka cluster
            - Consumers consume messages from the Kafka cluster
            - Producers and Consumers can write and read messages from different sources called topics
            - Brokers are internal to the Kafka Cluster and manage the messages flowing through the cluster in topics
            - Brokers receive messages from Producers and send messages to Consumers
        - Messages are ordered as they are sent by the producers and processed in the same order by the consumers
        - Messages are delivered with “at least once” reliability, and allow for replication which protects against the losing of data in the case of failure
    - Kafka gives us the ability to ingest a multitude of diverse data sources such as SIEM, vulnerability scanning reports, threat intelligence, or any other data source, and combine it into a centralized location
        - It does this with speed and reliability through horizontal scaling, which allows for it to accommodate even the largest of enterprises
- **HDFS**
    - HDFS or the Hadoop Distributed File System is a distributed file system that provides scalable and reliable data storage using commodity hardware
        - HDFS is the storage backbone of big data with nearly all big data technologies supporting reading and writing data to and from it
            - It’s integration with big data technologies allows for the exploitation of data locality, which means that data on this distributed file system can be accessed as fast as possible with minimal network transfer

- **Parquet**
  - Parquet is a columnar storage format that was built from the ground up to support very efficient compression and encoding schemes



    - Data is stored much more efficiently and can be read much faster
      - In a dataset with 450+ columns:
        - Counting events was ~29x faster using Parquet than CSV
        - The file size was ~45x smaller using Parquet than CSV
    - In a typical row based storage, data would be stored as a1b1c1a2b2c2… which doesn't allow for effective encoding or compression
    - In the columnar storage you can see in the diagram how data is stored a1a2a3… where those values are typically going to be somewhat similar, so encoding and compression can be extremely effective
    - Additionally, Parquet supports adding columns to an existing file due to the columnar storage format
      - If new fields are added to a data source in the future it will maintain compatibility with past data
- **Spark**
  - Spark is a general purpose cluster computing framework that is designed around performing in-memory computations to accelerate performance
    - Spark follows a master-worker model, where the master issues tasks to workers and the workers deliver results back to the master
    - Under the hood in Spark, uses a lazy evaluation model which helps with optimizing operations
      - Things like data locality and predicate pushdown are handled automatically within Spark
  - Spark has very easy programming interfaces of PySpark for Python and SparkSQL for SQL which most security analysts are already familiar with

        - Spark also supports Scala and Java which give a small amount of advanced features and typically add a very small amount of performance
    - Additional Spark Libraries
        - Spark MLlib
            - Machine Learning
        - Spark Streaming
            - Stream Processing in real-time micro batches
        - Spark GraphX
            - Graph computation and graph algorithms
- **Jupyter/IPython Notebooks**
    - A Jupyter notebook, previously known as an IPython notebook is an interactive computational environment where code, rich text, and data visualizations can be combined
        - Code that is executed in a notebook is sent to a kernel in the backend which handles the code execution and messaging back to the frontend
            - There are a large number of kernels available for jupyter notebooks to support different languages including but not limited to:
                - Python
                - Julia
                - Haskell
                - Ruby
                - JavaScript
                - Go
                - Scala
    - **JupyterHub**
        - JupyterHub is a set of processes that together provide a multi-tenant Jupyter Notebook server
            - Single User Server
                - Typical Jupyter Notebook process
            - Proxy
                - Public facing process that uses a dynamic proxy to route HTTP requests to the hub and single user servers
            - Hub
                - Manages user accounts, authentication, and single user servers
- **Graphistry**
    - Graphistry is a GPU-accelerated interactive graph visualization engine capable of visualizing 1m+ events or entities at a time
        - It uses GPU-acceleration both in the backend and frontend

        - Frontend uses WebGL to render the visualization
        - Backend uses OpenCL for clustering and layout computations as well as data querying
    - Graphistry has an open API
        - API key can be requested by emailing pygraphistry@graphistry.com

- **How to use**
    - Programming is done primarily in Python, but can be done in Scala or Java if desired.
        - Additional documentation for programming in Apache Spark available at: http://spark.apache.org/docs/1.6.2/api.html
    - There are three main data structures: RDD, Dataframe, and Pandas Dataframe.
    - RDDs and Dataframes are Spark data structures while the Pandas dataframe is a python library's data structure.
        - RDDs and Dataframes are distributed data structures while the pandas dataframe is a local data structure.

    - **RDD**
        - RDDs are the default way of loading new data into a notebook. If the data is in a simple format or unformatted, RDDs will be the easiest way to load in the data.

```
data = sc.textFile("file.csv")
```

        - The map function for the RDD is the primary means to do transformations on variables in the RDD.

```
data = data.map(lambda x: x.split(","))
```

        - For more complicated transformations, you can write python definitions that will get distributed to every node in the spark engine and run.

```
def do_something(x):
    return something(x)
data = data.map(lambda x: do_something(x))
```

    - **Dataframe**
        - The Spark dataframe is the most common format to operate on data as it has many built in aggregation functions that are less syntax dependent and easier to use than the RDD functions.
        - An RDD can be converted to a dataframe by providing a schema with which to build the "table".

```
df = sqlContext.createDataFrame(data, schema)
```

- Dataframes can saved as parquet files and then reloaded directly back into a dataframe.

```
df.write.parquet("file.parquet")
df = sqlContext.read.parquet("file.parquet")
```

- Dataframes allow common SQL queries to be easily performed on the data as if it were in a relational database table.

```
df.registerTempTable("table")
sqlContext.sql("SELECT * FROM table")
```

- **Pandas Dataframe**
  - Pandas is a python library that provides high performance data structures and data analysis tools.
    - Additional documentation for Pandas available at: http://pandas.pydata.org/pandas-docs/stable/
  - Typically, a pandas dataframe is obtained by converting a spark dataframe.

```
pdf = df.toPandas()
```

  - Once a pandas dataframe is obtained, it can be visualized using Graphistry.

```
plotter = graphistry.bind(source="src_col",
                          destination="dst_col")
plotter.plot(pdf)
```

  - Additional documentation and tutorials for graphistry available at: https://github.com/graphistry/pygraphistry