System Design Proposal:

The proposed system consists of four distinct components:

1) Data Integration
2) Data Warehousing (and Deep Storage)
3) OLAP Cubes for Fast Querying
4) DashBoard

Data Integration:

We have to realistically plan for increasing data sources of the following types:

1. Unstructured data (flat files, binarys - images etc)
2. Semi-structured data (Document stores, NoSql with Json/documents etc)
3. Structured (RDMS with schematic data)

The strategy for integration of these data sources is often dependent on the nature of usage of the data. Data teams often build pipelines with the sole purpose of capturing the data into a centralized system and move the burden of imposing structure to the analysts/scientists. This is a viable strategy with unstructured data.

This proposal concentrates effort on integrating semi-structured/structured data onto a defined (structured) data model with consideration for the following integration patterns:

a. Event Driven Data integration
b. Adhoc Data Integration

Let us define the terms Event Driven and Adhoc in relation to the different types of data sources one might encounter

Kafka: Is an Event Driven system with consumers and producers of the data writing each message to a distinct offset.
RDBMS/Column Stores (PostgresSQL/MySQL/CouchBase/Redshift): These systems primarily support requests for Data using synchronous JDBC connections, making them suitable for ad-hoc query requests. But there are solutions such as Debezium in place that convert Modify/Update/Create operations on these sources into event streams. This solution tries to consider both approaches.

Proposal for integration:

1. Use Kafka as the backbone for the data
Pro: Kafka can scale, and we can leverage its awesomeness out of the box.
Pro: Fault-Tolerant, Distributed
Con: Nothing significant really.

2. Use Kafka connect to capture changed data (CDC) from PostgreSQL/MySQL/CouchBase
Each Kafka-connect application does one thing well. Moves CDC data from a target table to destination topic, and can be deployed as a Microservice(containerized on a Kubernetes cluster) with a configure-to-deploy approach. (Configuration being the target hosts/table and destination host/topic)
Pro: Event streams from Databases. The whole data pipeline is a stream!
Cons: More instrumentation. If done correctly,we can create a micro-service-like template which uses configuration only for new pipelines.

3. Use Apache Nifi as the Data Integration tool to move Data from Kafka to the Data Warehouse, with JSON Schema validation, transformation built in! We have not yet discussed the Data Warehouse in play, but rest assured, Nifi provides integration with the all popular data warehouses which support JDBC(From Redshift to Oracle) and also supports Streaming to warehouses such as Hive.
Additionally, when there is a need to integrate non-CDC data from PostgreSQL/MySQL/CouchBase/MongoDB etc, NIFI provides built-in JDBC connection pools to 'Query' historical data from these systems and integrate them into a Warehouse (with Json Schema validation, transformation).
Pro: Built-in integration for hundreds of data sources. No need to solve boiler plate integration problem over and over.
Pro: Built in processors for Validation, transformation lookups etc.
Pro: Allows us to write our own custom processing
Pro: Distributed/Clustered system
Pro: Realtime data management using a UI
Pro: Persistence based Fault tolerance
Con: Stream(Fact-Fact) Joins not supported. And even if we roll.out our own, I suspect this could be slow. (Fact-Dim) Joins however are supported.
Con: Window operation and custom event detection not supported
Con: Comes with its own framework for writing highly concurrent Scala/Java code.
Con: Difficult Ops process. Needs a separate controller for Rest based Data flow manipulation (if one is not inclined to use the UI of course)

4. Use Apache Spark for event stream which require Window/Stream-Join operations
Pro: Highly Scalable and performant
Pro: Supports real-time custom event detection which could be important for businesses
Con: A solution for a specific use cases. Not a generic data integrator.

Data Warehousing:
There are many Warehousing solutions out there. PostgreSQL has rolled out their own Columnar storage (which Amazon neatly packages as Redshift) , but I would like to propose
1. *Apache Hive:* as a possible solution with ORC based columnar storage and HDFS/S3/Google Cloud based deep storage.

Pro: Hive is a mature solution, open-sourced(no licenses!) and is cloud native.

Pro: Provides ACID compliant Streaming API (To which NIFI already provides a built-in client)
Pro: ORC based columnar storage great for ad-hoc historical queries with Apache Spark /Tez based query engine
Pro: Scales for peta-bytes of data Data
Con: Instrumentation and Systems Integration is not as simple as it should be.
Con: What about reasonably large data?
Con: Good for historical/batch queries. But we may need to use a Column Store (such as redshift, presto) for interactive queries.
Con: Slow compared to Redshift for reasonably large interactive queries.

2. Redshift(PostgreSQL(column store) : A cloud-native column store
Pro: Easy integration with S3
Pro: Easy instrumentation
Pro: Very good for reasonably large data
Pro: JDBC connections and API gateway support
Con: API gateway is not suitable for a REST interaction model
Con: Great for large datasets, but may not scale over peta-bytes of data.
Hopefully there will be asnyc solutions in the future (eg: aiopg for redshift)

The Warehouse is suitable for Analysts to run ad-hoc interactive queries. Service, dashboards and reports often need different pre-aggregated data. One approach is to create pre-aggregated tables on the warehouse itself and create services/dashboards on top of the Warehouse. But for real-time use cases, I would look at OLAP cubes.

OLAP Cubes:

We can obviously create pre-aggregated tables on the Warehouse itself. But to move to a more real-time solution , we need a specialized OLAP cube.This is a system that may be instrumented for specific real-time use cases

Druid is one such cube which created pre-aggregated facts from JSON.
Pro: It has integration With Kafka/Hive built in, and is good for aggregating data over hundreds of billions of rows
Pro: REST API to read pre-aggregates onto any Dashboard/Services.
Con: Instrumentation/Integration effort

Dashboarding:
There are many licenced Dashboarding solutions out there such as Looker/Tableau/Qlik. From a Data Engineering standpoint, what is essential is that the Data Warehouse/Cube provide clean integration with the Data using REST/JDBC and the above system design does that.

Architecture Rough Diag: