Data Vault 2.0 Model Documentation

1. High-Level Architecture

Overview

The high-level architecture includes the following components:

1. **Player**: End-users interacting with one or more applications (App_1, App_N).
2. **Applications (App_1, App_N)**: Frontend applications generating events based on user actions.
3. **Backend Services**: Microservices processing user actions into events:
    - auth: Generates auth_msg events for user authentication.
    - spins: Generates spins_msg events for user activity.
    - purchase: Generates purchase_msg events for purchases.
4. **Event Bus**: A distributed messaging system to ingest, process, and route events.
5. **Data Platform**: The destination where the events are processed, transformed, and stored in the Data Vault format.

2. Key Considerations

1. **Real-Time Event Processing**:
    - Applications generate high volumes of events (auth_msg, spins_msg, purchase_msg) in real-time. Streaming enables immediate ingestion and transformation, reducing latency.
2. **Scalability**:
    - Streaming systems (e.g., Kafka, Kinesis) handle large-scale data ingestion and ensure message durability.
3. **Event-Driven Design**:
    - The architecture's Event Bus layer naturally aligns with streaming solutions to process events incrementally.
4. **Seamless Integration**:
    - Streaming pipelines transform raw events into Hubs, Links, and Satellites, maintaining data lineage and auditability.

3. Streaming Solution

3.1 Components

1. **Event Bus**:
    - **Apache Kafka**:
        i. Acts as the Event Bus for real-time message ingestion.
        ii. Topics for each event type (auth_msg, spins_msg, purchase_msg).
2. **Stream Processing Framework**:
    - **Apache Spark Streaming**:
        i. Processes events in real-time.
        ii. Transforms raw events into structured data.
3. **Data Storage**:
    - **Delta Lake** (or Amazon S3):
        i. ACID-compliant storage for raw, processed, and modeled data.
    - **Amazon Redshift** or **BigQuery**:
        i. For analytics and reporting.

3.2 End-to-End Streaming Workflow

Step 1: Event Ingestion

- **Event Sources**: Backend services (auth, spins, purchase) publish events to Kafka topics:
    - auth_msg
    - spins_msg
    - purchase_msg

Step 2: Stream Processing

- **Consumer**: Spark Streaming reads from Kafka topics.
- **Transformations**:
    I. Extract and parse the JSON payload.
    II. Generate surrogate keys (e.g., User_Hub_Key, App_Hub_Key).
    III. Apply business logic to populate Hubs, Links, and Satellites.

Step 3: Data Storage

- Processed data is stored in:
    - **Hubs**: Unique identifiers (business keys).

- **Links**: Relationships between Hubs.
- **Satellites**: Descriptive and time-variant attributes.

Step 4: Querying and Analysis

- Data stored in the Data Vault format is queried using Redshift or BigQuery.

4. Data Vault 2.0 Implementation

4.1 Hubs

1. **User_Hub**:
   - User_Hub_Key (PK): SHA256 hash of UID *VARCHAR(64)*.
   - UID: Unique user identifier *VARCHAR(255)*.
   - Record_Source: Source system (e.g., auth_msg) *VARCHAR(255)*.
2. **App_Hub**:
   - App_Hub_Key (PK): SHA256 hash of App_Name *VARCHAR(64)*.
   - App_Name: Name of the application *VARCHAR(255)*.
   - Record_Source: Source system *VARCHAR(255)*.
3. **Event_Hub**:
   - Event_Hub_Key (PK): SHA256 hash of msg_id *VARCHAR(64)*.
   - Event_Type: Type of event (e.g., auth_event, spin_event) *VARCHAR(100)*.
   - Record_Source: Source system *VARCHAR(255)*.

4.2 Links

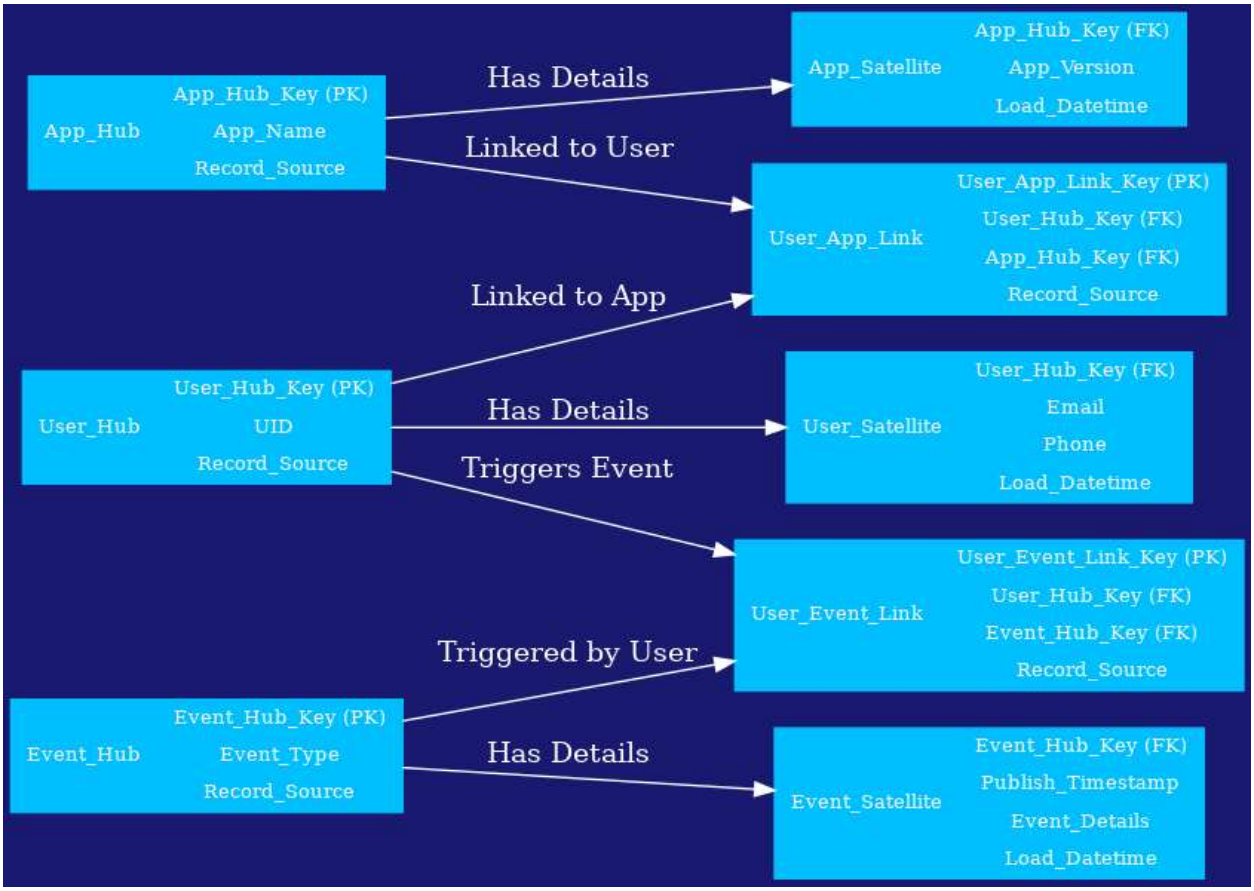
1. **User_App_Link**:
   - Links User_Hub and App_Hub.
   - Attributes:
     - User_App_Link_Key (PK): Composite hash of User_Hub_Key + App_Hub_Key *VARCHAR(128)*.
     - Foreign keys: User_Hub_Key, App_Hub_Key *VARCHAR(64)*.

2. **User_Event_Link**:
   - Links User_Hub and Event_Hub.
   - Attributes:
     - User_Event_Link_Key (PK): Composite hash of User_Hub_Key + Event_Hub_Key *VARCHAR(255)*.

4.3 Satellites

1. **User_Satellite**:
    - Attributes:
        - Email *VARCHAR(255)*
        - Phone *VARCHAR(20)*
        - Load_Datetime *TIMESTAMP*
2. **App_Satellite**:
    - Attributes:
        - App_Version *VARCHAR(50)*
        - Load_Datetime *TIMESTAMP*
3. **Event_Satellite**:
    - Attributes:
        - Publish_Timestamp *TIMESTAMP*
        - Event_Details *TEXT*

5. Diagram



6. Additional Components

1. **Data Quality Validation**:
    - Use **Great Expectations** for schema validation and quality checks.
2. **Error Handling and Logging**:
    - Centralized logging using **AWS CloudWatch** or the ELK stack.
3. **Automation**:
    - CI/CD pipelines for deployment using **GitHub Actions** or **Jenkins**.
4. **Orchestration**:
    - Orchestration tools like **Apache Airflow** and **AWS Step Functions** are employed to schedule and manage the streaming workflows.