# System Description

The system is designed for processing market data, specifically focusing on stock prices and trades. It emphasizes real-time data flow and processing, utilizing technologies like Apache Kafka and Apache Spark. The system ingests data from various sources, processes it in real-time, stores the processed data, and pushes updates to a web/mobile application.

# System Components

1. **Data Sources**: These are the sources from where market data, stock prices, and trades are ingested. The specific technology used for this is not mentioned in the diagram.
2. **Kafka Streaming Cluster**: This component handles data ingestion in real-time. It uses Apache Kafka, a distributed streaming platform.
3. **Spark Streaming**: This component processes data for transformations, aggregations, and joins. It uses Apache Spark, a unified analytics engine for large-scale data processing.
4. **Data Lake (Cassandra)**: This component stores the processed data. It uses Apache Cassandra, a highly scalable and distributed database.
5. **Direct Data Push**: This component sends real-time updates to a web/mobile application. It uses Web Socket or HTTP streaming, which are communication protocols for real-time data transfer.
6. **Web/Mobile App**: These are the user interfaces, utilizing libraries like Bootstrap.js, Chart.js, and jQuery.js.

# Assets and Data Flow

- **Data Sources -> Kafka Streaming Cluster**: The data sources send market data, stock prices, and trades to the Kafka Streaming Cluster. The communication protocol is not explicitly mentioned, but it's likely to be a data ingestion protocol compatible with Apache Kafka.

- **Kafka Streaming Cluster -> Spark Streaming**: The Kafka Streaming Cluster sends the ingested data to Spark Streaming for processing. The communication protocol is likely to be a data streaming protocol compatible with Apache Kafka and Apache Spark.

- **Spark Streaming -> Data Lake (Cassandra)**: Spark Streaming sends the processed data to the Data Lake for storage. The communication protocol is likely to be a data storage protocol compatible with Apache Spark and Apache Cassandra.

- **Data Lake (Cassandra) -> Direct Data Push**: The Data Lake sends the stored data to Direct Data Push for real-time updates. The communication protocol is not explicitly mentioned, but it's likely to be a data retrieval protocol compatible with Apache Cassandra.
- **Direct Data Push -> Web/Mobile App**: Direct Data Push sends real-time updates to the Web/Mobile App. The communication protocol is either Web Socket or HTTP streaming, which are protocols for real-time data transfer.

## Trust Boundaries:

- Data Sources: 1
- Kafka Streaming Cluster: 1
- Spark Streaming: 4
- Data Lake (Cassandra): 8
- Direct Data Push: 1
- Web/Mobile App: 1

## Threat Scenarios

- An External Attacker can perform Network Sniffing on the Kafka Streaming Cluster to intercept sensitive data. (CVSS: High)
- A Malicious Insider can execute a SQL Injection attack on the Data Lake (Cassandra) to manipulate or delete critical data. (CVSS: High)
- A Phishing Attacker can send deceptive emails to users of the Web/Mobile App to steal their login credentials. (CVSS: Medium)
- A Denial of Service (DoS) Attacker can flood the Spark Streaming nodes with excessive requests, causing system downtime. (CVSS: High)
- A Man-in-the-Middle Attacker can intercept Direct Data Push communication to inject malicious code or alter data in transit. (CVSS: High) #

## Controls

- **Network Sniffing on Kafka Streaming Cluster:**
    - Implement encryption for data in transit within the Kafka cluster to prevent interception.

    - Utilize network segmentation to restrict access to the Kafka cluster only to authorized users.
    - Enable strong authentication mechanisms such as mutual TLS for communication within the cluster.
- **SQL Injection on Data Lake (Cassandra):**
    - Input validation and parameterized queries should be implemented to prevent SQL injection attacks.
    - Regularly patch and update the Cassandra database to address any known vulnerabilities.
    - Implement least privilege access controls to limit the impact of a successful SQL injection attack.
- **Phishing Attacks on Web/Mobile App Users:**
    - Conduct regular security awareness training for users to recognize and report phishing attempts.
    - Implement multi-factor authentication to add an extra layer of security for user login credentials.
    - Use email filtering solutions to detect and block phishing emails before they reach users.
- **Denial of Service (DoS) on Spark Streaming Nodes:**
    - Implement rate limiting and request validation to mitigate the impact of excessive requests.
    - Utilize load balancers to distribute incoming traffic and prevent overwhelming specific nodes.
    - Monitor network traffic and system performance to detect and respond to potential DoS attacks.
- **Man-in-the-Middle Attacks on Direct Data Push Communication:**
    - Implement end-to-end encryption for data transmission to protect against interception and tampering.
    - Use digital signatures to verify the integrity and authenticity of data exchanged between systems.
    - Implement certificate pinning to ensure secure communication channels and prevent MITM attacks.