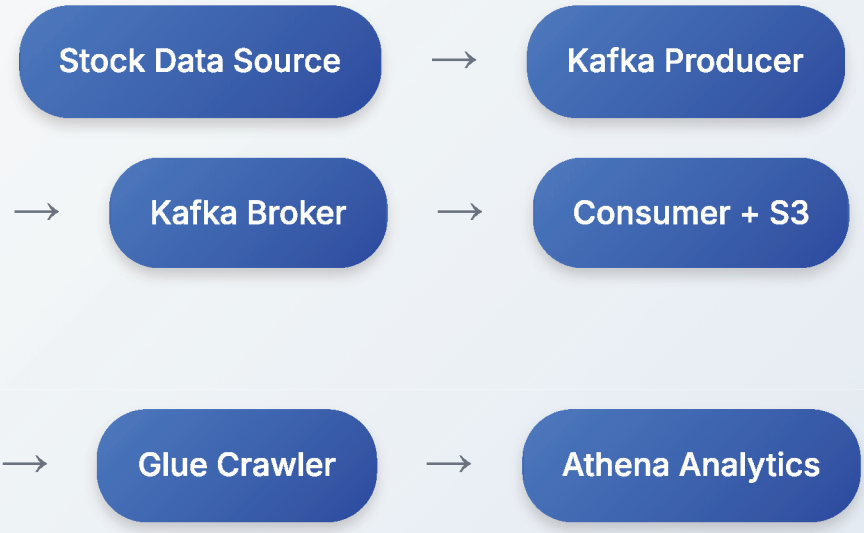
📊 Real-Time Stock Market Data Pipeline

Complete User Manual & Implementation Guide

🎯 Project Overview

This comprehensive project demonstrates an **end-to-end data engineering pipeline** for real-time stock market data processing. The system seamlessly integrates modern data streaming technologies with cloud infrastructure to provide scalable, real-time analytics capabilities.

🏗️ Architecture Flow



🔧 Technologies & Stack

💻 Programming Language

- Python 3.8+
- Pandas for data manipulation
- JSON for data serialization

🚀 Apache Kafka Stack

- Kafka Broker (v3.3.1)
- Zookeeper
- Kafka Streams
- Producer/Consumer APIs

☁️ Amazon Web Services

- EC2 (Compute Infrastructure)
- S3 (Object Storage)
- Glue (ETL & Catalog)
- Athena (SQL Analytics)

🐍 Python Libraries

- kafka-python
- s3fs
- pandas
- boto3

📦 Setup Instructions

1 Kafka Installation & Configuration

Begin by establishing a secure connection to your EC2 instance and installing the Kafka ecosystem.

💻 SSH Connection

```
ssh -i <your-key.pem> ec2-user@<your-ec2-public-ip>
```

1.1 Download and Extract Kafka

💻 Kafka Installation

```
wget https://downloads.apache.org/kafka/3.3.1/kafka_2.12-3.3.1.tgz tar -xvf kafka_2.12-3.3.1.tgz cd kafka_2.12-3.3.1
```

1.2 Java Environment Setup

💻 Java Installation

```
java -version sudo yum install java-1.8.0-openjdk java -version
```

⚠️ **Important:** Ensure Java 8 or higher is installed as Kafka requires JVM to run properly.

🚦 Starting Kafka Services

2 Service Initialization

2.1 ZooKeeper Service

Start ZooKeeper in the first terminal session:

💻 Terminal 1 - ZooKeeper

```
bin/zookeeper-server-start.sh config/zookeeper.properties
```

2.2 Kafka Broker Configuration

Configure server properties for external access:

💻 Server Configuration

```
sudo nano config/server.properties # Replace this line:
#advertised.listeners=PLAINTEXT://your.host.name:9092 #
With: advertised.listeners=PLAINTEXT://<your-ec2-public-
ip>:9092
```

2.3 Start Kafka Broker

Launch Kafka broker in a new terminal session:

💻 Terminal 2 - Kafka Broker

```
export KAFKA_HEAP_OPTS="-Xmx256M -Xms128M" bin/kafka-
server-start.sh config/server.properties
```

💡 **Pro Tip:** The heap size configuration optimizes memory usage for small to medium workloads.

🛠️ Kafka Topic Management

3 Topic Creation & Testing

3.1 Create Stock Market Topic

💻 Topic Creation

```
bin/kafka-topics.sh --create --topic stock-market \ --
bootstrap-server <your-ec2-public-ip>:9092 \ --
replication-factor 1 --partitions 1
```

3.2 Producer Testing

💻 Console Producer

```
bin/kafka-console-producer.sh --topic stock-market \ --
bootstrap-server <your-ec2-public-ip>:9092
```

3.3 Consumer Testing

💻 Console Consumer

```
bin/kafka-console-consumer.sh --topic stock-market \ --
bootstrap-server <your-ec2-public-ip>:9092 --from-
beginning
```

🐍 Python Implementation

4 Data Producer & Consumer

4.1 Real-Time Stock Data Producer

Create a Python script that continuously sends stock market data to Kafka:

💻 producer.py

```
from kafka import KafkaProducer import json import pandas as pd from time import sleep import random from datetime import datetime # Initialize Kafka Producer producer = KafkaProducer( bootstrap_servers=['<your-ec2-public-ip>:9092'], value_serializer=lambda x: json.dumps(x).encode('utf-8'), retries=3, batch_size=16384, linger_ms=10 ) # Load stock data df = pd.read_csv('stocks.csv') print("🚀 Starting real-time stock data producer...") try: while True: # Sample random stock record record = df.sample(1).to_dict(orient='records')[0] # Add timestamp record['timestamp'] = datetime.now().isoformat() # Send to Kafka producer.send('stock-market', value=record) print(f"📊 Sent: {record['symbol']} - ${record['price']}") sleep(1) # Send every second except KeyboardInterrupt: print("🛑 Producer stopped") finally: producer.close()
```

4.2 Kafka Consumer with S3 Integration

Implement a consumer that streams data directly to AWS S3:

💻 consumer.py

```
from kafka import KafkaConsumer import json import s3fs from datetime import datetime import uuid # Initialize Kafka Consumer consumer = KafkaConsumer( 'stock-market', bootstrap_servers=['<your-ec2-public-ip>:9092'], value_deserializer=lambda x: json.loads(x.decode('utf-8')), auto_offset_reset='latest', enable_auto_commit=True, group_id='stock-s3-consumer' ) # Initialize S3 FileSystem s3 = s3fs.S3FileSystem( anon=False, client_kwargs= {'region_name': 'us-east-1'} ) print("🔄 Starting Kafka consumer with S3 sink...") try: for count, message in enumerate(consumer): # Generate unique filename timestamp = datetime.now().strftime('%Y%m%d_%H%M%S') filename = f'stock_market_{timestamp}_{uuid.uuid4().hex[:8]}.json' # Write to S3 s3_path = f's3://stock-market-realtime-kafka/{filename}' with s3.open(s3_path, 'w') as f:
```

```
json.dump(message.value, f, indent=2) print(f"💾 Saved to
S3: {filename}") except KeyboardInterrupt: print("🛑
Consumer stopped") finally: consumer.close()
```

⚠️ **Security Note:** Always use IAM roles instead of hardcoded credentials in production environments.

⚙️ AWS Credentials Configuration

5 Authentication Setup

Option 1: AWS Credentials File (Recommended)

💻 ~/.aws/credentials

```
[default] aws_access_key_id = YOUR_ACCESS_KEY
aws_secret_access_key = YOUR_SECRET_KEY region = us-east-1
```

Option 2: Environment Variables

💻 Environment Setup

```
export AWS_ACCESS_KEY_ID=your-access-key export
AWS_SECRET_ACCESS_KEY=your-secret-key export
AWS_DEFAULT_REGION=us-east-1
```

Option 3: Direct Configuration in Code

💻 Direct Configuration

```
s3 = s3fs.S3FileSystem( key='your-access-key',
secret='your-secret-key', client_kwargs={'region_name':
```

🗃️ AWS Glue & Athena Integration

6 Data Catalog & Analytics

6.1 AWS Glue Crawler Setup

> 💡 **Configuration Steps:**
>
> 1. Navigate to AWS Glue Console
> 2. Create a new Crawler
> 3. Set data source: `s3://stock-market-realtime-kafka`
> 4. Choose IAM role with S3 and Glue permissions
> 5. Configure output database and table prefix
> 6. Set schedule (e.g., hourly or daily)

6.2 Required IAM Permissions

💻 IAM Policy JSON

```
{ "Version": "2012-10-17", "Statement": [ { "Effect":
"Allow", "Action": [ "s3:GetObject", "s3:ListBucket" ],
"Resource": [ "arn:aws:s3:::stock-market-realtime-kafka",
"arn:aws:s3:::stock-market-realtime-kafka/*" ] }, {
"Effect": "Allow", "Action": [ "glue:*", "athena:*" ],
"Resource": "*" } ] }
```

6.3 Athena Query Examples

💻 Sample Athena Queries

```
-- Basic stock data query SELECT * FROM
"stock_database"."stock_market_realtime_kafka" ORDER BY
timestamp DESC LIMIT 100; -- Top performing stocks SELECT
symbol, AVG(price) as avg_price, COUNT(*) as records FROM
"stock_database"."stock_market_realtime_kafka" GROUP BY
symbol ORDER BY avg_price DESC; -- Daily volume analysis
SELECT DATE(timestamp) as trade_date, symbol, SUM(volume)
as daily_volume, AVG(price) as avg_price FROM
"stock_database"."stock_market_realtime_kafka" GROUP BY
DATE(timestamp), symbol ORDER BY trade_date DESC,
```

✅ Expected Outcomes

✅ **Congratulations!** Upon successful completion, you will have achieved:

🔄 Real-Time Data Ingestion

Stock market data continuously flowing through Kafka with sub-second latency

☁️ Cloud Storage Integration

Automatic data persistence to AWS S3 with organized file structure

📊 Schema Discovery

Automated schema inference and metadata management via AWS Glue

🔍 SQL Analytics

Interactive querying capabilities through AWS Athena with standard SQL