Stock Market Project

This is a shell script, written in the Bash programming language, that automates the installation of Apache Kafka and Java on an EC2 instance.

```
#!/bin/bash
s
# Download and extract Kafka
wget https://downloads.apache.org/kafka/3.4.0/kafka_2.13-3.4.0.tgz
tar -xvf kafka_2.13-3.4.0.tgz

# Install Java if it's not already installed
if ! type java >/dev/null 2>&1; then
  sudo yum install java-1.8.0-openjdk
fi
```

The script begins by using the wget command to download the Kafka binary files from the Apache website, and then uses the tar command to extract the files. The version being downloaded is 3.4.0, which can be changed to a different version if needed.

Next, the script checks if Java is already installed on the EC2 instance by running the type java command. If Java is not already installed, the script uses the yum package manager to install Java 1.8.0 OpenJDK, a popular implementation of the Java Virtual Machine.

The purpose of this script is to streamline the installation process of Kafka and Java on an EC2 instance, saving time and reducing the risk of errors that could occur during a manual installation.

Step-by-step explanation for setting up Kafka in an EC2 instance and creating a topic for message production and consumption. The first step involves connecting to the Zookeeper server by running the command "bin/zookeeper-server-start.sh config/zookeeper.properties" in one console. This starts the Zookeeper server, which is a distributed coordination service used by Kafka for managing its cluster.

The second step is to open another console and connect to the Kafka server. The command "export KAFKA_HEAP_OPTS="-Xmx256M -Xms128M"" allocates some memory to the Kafka server. Then, the user changes the directory to the Kafka installation directory by running the command "cd kafka_2.12-3.3.1" and starts the Kafka server by running the command "bin/kafka-server-start.sh config/server.properties".

The next step involves modifying the IP address from private to public to match the EC2 instance being used. This is important for the Kafka clients to be able to connect to the Kafka cluster.

To allow traffic to flow to the Kafka server, the user creates a security group that allows all traffic with a source of their IP address.

Finally, the user creates a Kafka topic named "demo_testing2" by running the command "bin/kafka-topics.sh --create --topic demo_testing2 --bootstrap-server {Public IP of the EC2 Instance:9092} --replication-factor 1 --partitions 1". This command creates a new topic with the specified name and settings, including the replication factor and the number of partitions. Once this command is executed, the Kafka topic can be used for producing and consuming messages.

The Kafka producer reads data from a CSV file and sends it to a Kafka topic. The data is converted to JSON format using pandas and the Kafka Python library is used to send the data to the Kafka topic. The producer sends data to the topic every second using a while loop.

On the other hand, the Kafka consumer reads the data from the Kafka topic and writes it to an S3 bucket. The Kafka consumer is implemented using the Kafka Python library, and it connects to the Kafka topic and subscribes to it. The consumer then retrieves messages from the Kafka topic and writes them to S3 using the S3FileSystem module. The S3 file name is unique every time and is given by the count of the messages.

After the data is stored in S3, it can be queried and analysed. Amazon Glue is used to create a Glue Crawler, which crawls the S3 bucket and creates a table that can be queried using Amazon Athena. The Glue Crawler is given full access to the S3 bucket, and it creates a target database for the crawled data. Once the Glue Crawler stops, the data is ready to be queried and analysed using Amazon Athena.