BIG DATA PROJECT WORKFLOW: Kafka + Producer + Consumer + DynamoDB (on EC2)

---

1. Launch EC2 Instance (Amazon Linux 2)

Connect using: ssh -i "kafka.pem" ec2-user@<EC2_PUBLIC_IP>

```
neha n@Neha MINGW64 ~/Downloads/bdpro (master)
$ ssh -i "kafka.pem" ec2-user@ec2-44-211-126-1.compute-1.amazonaws.com
The authenticity of host 'ec2-44-211-126-1.compute-1.amazonaws.com (44.211.126.1
)' can't be established.
ED25519 key fingerprint is SHA256:++NAgBPhXfZpLRu3hAKKn1EQNYRlu5ws3X+ETOwnU7c.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ec2-44-211-126-1.compute-1.amazonaws.com' (ED25519)
to the list of known hosts.
   ,     #_
   ~\_  ####_        Amazon Linux 2
  ~~  \_#####\
  ~~     \###|       AL2 End of Life is 2026-06-30.
  ~~       \#/ ___
   ~~       V~' '->
    ~~~         /    A newer version of Amazon Linux is available!
      ~~._.   _/
         _/ _/       Amazon Linux 2023, GA and supported until 2028-03-15.
       _/m/'           https://aws.amazon.com/linux/amazon-linux-2023/
```

---

2. Install Java, Kafka, and Zookeeper

sudo yum update -y
sudo yum install java-1.8.0-openjdk -y
java -version

Download & extract Kafka:

wget https://downloads.apache.org/kafka/3.5.1/kafka_2.12-3.5.1.tgz
tar -xvf kafka_2.12-3.5.1.tgz
cd kafka_2.12-3.5.1

```
[ec2-user@ip-172-31-95-125 ~]$ cd kafka_2.12-3.5.1
[ec2-user@ip-172-31-95-125 kafka_2.12-3.5.1]$
```

---

3. Start ZooKeeper (Window 1)

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

```
[2025-04-15 01:20:37,592] INFO  (org.apache.zookeeper.server.ZooKeeperServer)
[2025-04-15 01:20:37,592] INFO   ______                   _                                          (org.apache.zookeeper.server.ZooKeeperServer)
[2025-04-15 01:20:37,592] INFO  |___  /                  | |                                         (org.apache.zookeeper.server.ZooKeeperServer)
[2025-04-15 01:20:37,592] INFO     / /    ___     ___    | | __   ___    ___   _ __    ___   _ __    (org.apache.zookeeper.server.ZooKeeperServer)
[2025-04-15 01:20:37,592] INFO    / /    / _ \   / _ \   | |/ /  / _ \  / _ \ | '_ \   / _ \ | '__| (org.apache.zookeeper.server.ZooKeeperServer)
[2025-04-15 01:20:37,593] INFO   / /__  | (_) | | (_) |  |   <  |  __/ |  __/ | |_) | |  __/ | |     (org.apache.zookeeper.server.ZooKeeperServer)
[2025-04-15 01:20:37,593] INFO  /_____|  \___/   \___/  |_|\_\  \___|  \___| | .__/   \___| |_| (org.apache.zookeeper.server.ZooKeeperServer)
[2025-04-15 01:20:37,593] INFO                                               | |                     (org.apache.zookeeper.server.ZooKeeperServer)
[2025-04-15 01:20:37,593] INFO                                               |_|                     (org.apache.zookeeper.server.ZooKeeperServer)
[2025-04-15 01:20:37,593] INFO  (org.apache.zookeeper.server.ZooKeeperServer)
[2025-04-15 01:20:37,602] INFO Server environment:zookeeper.version=3.6.4--d65253dcf68e9097c6e95a126463fd5fdeb4521c, built on 12/18/2022 18:10 GMT (org.a
[2025-04-15 01:20:37,602] INFO Server environment:host.name=ip-172-31-95-125.ec2.internal (org.apache.zookeeper.server.ZooKeeperServer)
```

---

4. Start Kafka Broker (Window 2)

export KAFKA_HEAP_OPTS="-Xmx256M -Xms128M"
cd kafka_2.12-3.5.1
bin/kafka-server-start.sh config/server.properties

Edit config/server.properties first:

sudo nano config/server.properties

Set:

advertised.listeners=PLAINTEXT://<EC2_PUBLIC_IP>:9092
listeners=PLAINTEXT://0.0.0.0:9092

```
[KafkaServer id=0] started (kafka.server.KafkaServer)
[Controller id=0, targetBrokerId=0] Disconnecting from node 0 due to socket connection setup timeout. The timeout value is 10253 ms. (org.apache.kafka.clients.NetworkClient)
[Controller id=0, targetBrokerId=0] Client requested connection close from node 0 (org.apache.kafka.clients.NetworkClient)
[Controller id=0, targetBrokerId=0] Disconnecting from node 0 due to socket connection setup timeout. The timeout value is 8611 ms. (org.apache.kafka.clients.NetworkClient)
[Controller id=0, targetBrokerId=0] Client requested connection close from node 0 (org.apache.kafka.clients.NetworkClient)
[Controller id=0, targetBrokerId=0] Disconnecting from node 0 due to socket connection setup timeout. The timeout value is 9254 ms. (org.apache.kafka.clients.NetworkClient)
[Controller id=0, targetBrokerId=0] Client requested connection close from node 0 (org.apache.kafka.clients.NetworkClient)
[Controller id=0, targetBrokerId=0] Disconnecting from node 0 due to socket connection setup timeout. The timeout value is 10870 ms. (org.apache.kafka.clients.NetworkClient)
[Controller id=0, targetBrokerId=0] Client requested connection close from node 0 (org.apache.kafka.clients.NetworkClient)
```


---

5. Create Kafka Topic, Producer (Window 3)

cd kafka_2.12-3.5.1
bin/kafka-topics.sh --create --topic demo --bootstrap-server <EC2_PUBLIC_IP>:9092 --replication-factor 1 --partitions 1

```
[ec2-user@ip-172-31-95-125 ~]$ cd kafka_2.12-3.5.1
[ec2-user@ip-172-31-95-125 kafka_2.12-3.5.1]$ bin/kafka-topics.sh --create --topic demo_testing2 --bootstrap-server 44.211.126.1:9092 --replication-factor 1 --partitions 1
OpenJDK 64-Bit Server VM warning: If the number of processors is expected to increase from one, then you should configure the number of parallel GC threads appropriately using -XX:ParallelGCThreads=N
WARNING: Due to limitations in metric names, topics with a period ('.') or underscore ('_') could collide. To avoid issues it is best to use either, but not both.
Created topic demo_testing2.
[ec2-user@ip-172-31-95-125 kafka_2.12-3.5.1]$
```

```
Created topic demo_testing2.
[ec2-user@ip-172-31-95-125 kafka_2.12-3
OpenJDK 64-Bit Server VM warning: If th
>neha
>vikas
>kavya
>pranav
>jeswanth
>hello consumer this is producer!!!!!
>neha
>vikas
>kavya
>pranav
>jeswanth
>
```

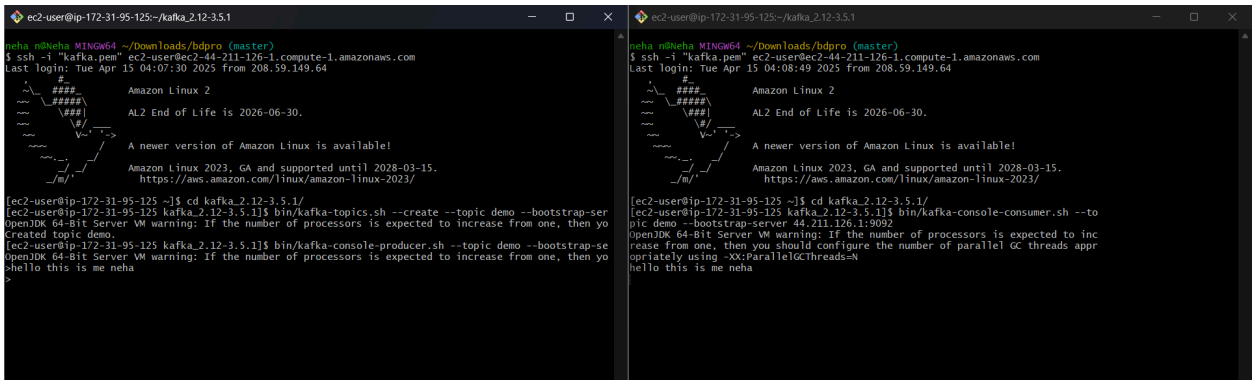
---

6. Start Kafka Consumer (Window 4)

bin/kafka-console-consumer.sh --topic demo --bootstrap-server <EC2_PUBLIC_IP>:9092 --from-beginning

```
[ec2-user@ip-172-31-95-125 kafka_2.12-3.5.1]$ bin/kafka-console-consumer.sh --to
pic demo_testing2 --bootstrap-server 44.211.126.1:9092
OpenJDK 64-Bit Server VM warning: If the number of processors is expected to inc
rease from one, then you should configure the number of parallel GC threads appr
opriately using -XX:ParallelGCThreads=N

hello consumer this is producer!!!!!
neha
vikas
kavya
pranav
jeswanth
```

DATA STREAMING FROM PRODUCER TO CONSUMER:



---

7. Creating a dataset, Python Producer (Colab)

Dataset is created using **finance and we select the companies** we want namely: Meta, NVIDEA, Tesla, Amazon and so on.

```python
import yfinance as yf
import pandas as pd
from datetime import date
from IPython.display import display

# Define tickers and company names
tickers = {
    'NVDA': 'NVIDIA',
    'TSLA': 'Tesla',
    'META': 'Meta',
    'AMZN': 'Amazon',
    'AAPL': 'Apple',
    'NFLX': 'Netflix',
    'GOOGL': 'Google'
}

# List to collect all rows
all_data = []

# Fetch and process stock data
for ticker, company_name in tickers.items():
    stock = yf.Ticker(ticker)
    # hist = stock.history(period='1mo')
    hist = stock.history(start='2023-04-01', end=str(date.today()))

    for index, row in hist.iterrows():
        all_data.append({
            'Date': index.date(),
            'Open': row['Open'],
            'High': row['High'],
            'Low': row['Low'],
            'Close': row['Close'],
            'Volume': row['Volume'],
            'Company Name': company_name
        })

# Create DataFrame at once — no warnings, faster
result_df = pd.DataFrame(all_data, columns=['Date', 'Open', 'High', 'Low', 'Close', 'Volume', 'Company Name'])

# Show result
result_df.head()
```

A brief view of the dataframe, which is then stored as a csv file called “stock_data.csv” to be used later.

	Date	Open	High	Low	Close	Volume	Company Name
0	2023-04-03	27.491146	27.981827	27.318258	27.946850	398716000.0	NVIDIA
1	2023-04-04	27.947848	27.981827	27.289276	27.435181	368592000.0	NVIDIA
2	2023-04-05	26.811588	26.980477	26.377869	26.863554	515015000.0	NVIDIA
3	2023-04-06	26.566744	27.062422	26.409846	27.019451	397654000.0	NVIDIA
4	2023-04-10	26.805590	27.603073	26.651691	27.561100	395279000.0	NVIDIA

```
!pip install kafka-python pandas

from kafka import KafkaProducer
from json import dumps
import pandas as pd
from time import sleep

producer = KafkaProducer(
    bootstrap_servers=['<EC2_PUBLIC_IP>:9092'],
    value_serializer=lambda x: dumps(x).encode('utf-8')
)

df = pd.read_csv("stock_data.csv")

for _, row in df.iterrows():
    data = row.to_dict()
    producer.send('demo', value=data)
    sleep(1)

producer.flush()
```

KafkaProducer.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text

```python
!pip install kafka-python pandas

import pandas as pd
from kafka import KafkaProducer
from json import dumps
from time import sleep

# Load CSV data
df = pd.read_csv("stock_data.csv")

# Set up Kafka Producer
producer = KafkaProducer(
    bootstrap_servers=['44.211.126.1:9092'],  # Your EC2 public IP
    value_serializer=lambda x: dumps(x).encode('utf-8')
)

# Stream each row to Kafka
for i in range(len(df)):
    row = df.iloc[i].to_dict()
    producer.send('demo', value=row)
    print(f"Sent: {row}")
    sleep(1)  # simulate 1-second interval streaming

producer.flush()
```

```
Sent: {'Date': '06-02-2025', 'Open': 127.4082733, 'High': 128.7581552, 'Low': 125.1984776, 'Close': 128.6681519, 'Volume': 251483600,
Sent: {'Date': '07-02-2025', 'Open': 129.2081106, 'High': 130.3579987, 'Low': 124.9884977, 'Close': 129.8280487, 'Volume': 228186300,
Sent: {'Date': '10-02-2025', 'Open': 130.078033, 'High': 134.9875852, 'Low': 129.9480554, 'Close': 133.557724, 'Volume': 216989100, '
Sent: {'Date': '11-02-2025', 'Open': 132.567815, 'High': 134.4676343, 'Low': 131.0079609, 'Close': 132.787796, 'Volume': 178902400, '
Sent: {'Date': '12-02-2025', 'Open': 130.0080528, 'High': 132.2278499, 'Low': 129.0681367, 'Close': 131.1279449, 'Volume': 160278600,
Sent: {'Date': '13-02-2025', 'Open': 131.5479045, 'High': 136.4874529, 'Low': 131.157941, 'Close': 135.2775574, 'Volume': 197430000,
Sent: {'Date': '14-02-2025', 'Open': 136.4674421, 'High': 139.2371916, 'Low': 135.4875365, 'Close': 138.8372345, 'Volume': 195479600,
Sent: {'Date': '18-02-2025', 'Open': 141.2570149, 'High': 143.4268136, 'Low': 137.9173105, 'Close': 139.3871765, 'Volume': 219176600,
Sent: {'Date': '19-02-2025', 'Open': 139.4971666, 'High': 141.3470026, 'Low': 137.2073839, 'Close': 139.2171936, 'Volume': 167536000,
Sent: {'Date': '20-02-2025', 'Open': 140.0171277, 'High': 140.6470747, 'Low': 136.77742, 'Close': 140.0971222, 'Volume': 143903600, '
Sent: {'Date': '21-02-2025', 'Open': 140.0271179, 'High': 141.4470007, 'Low': 134.0176759, 'Close': 134.4176331, 'Volume': 228217600
```

---

8. Create IAM User & Setup

- Go to AWS Console → IAM
- Create new user → **programmatic access**
- Attach policy: AmazonDynamoDBFullAccess
- Save access key and secret

---

9. Create DynamoDB Table (Python)

import boto3
import time

```
session = boto3.Session(
    aws_access_key_id='YOUR_KEY',
    aws_secret_access_key='YOUR_SECRET',
    region_name='us-east-1'
)

dynamodb = session.resource('dynamodb')

table = dynamodb.create_table(
    TableName='StockMkt',
    AttributeDefinitions=[
        {'AttributeName': 'Company Name', 'AttributeType': 'S'},
        {'AttributeName': 'Date', 'AttributeType': 'S'}
    ],
    KeySchema=[
        {'AttributeName': 'Company Name', 'KeyType': 'HASH'},
        {'AttributeName': 'Date', 'KeyType': 'RANGE'}
    ],
    ProvisionedThroughput={'ReadCapacityUnits': 5, 'WriteCapacityUnits': 5}
)

print("Creating table...")
table.meta.client.get_waiter('table_exists').wait(TableName='StockMkt')
print("Table is ACTIVE.")
```

```python
# Step 1: Start session with credentials
session = boto3.Session(
    aws_access_key_id='AKIA6D6JBQNFBRIBT2NC',
    aws_secret_access_key='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',
    region_name='us-east-1'
)

# Step 2: Connect to DynamoDB
dynamodb = session.resource('dynamodb')

# Step 3: Create the table
table = dynamodb.create_table(
    TableName='StockMktKafka',
    AttributeDefinitions=[
        {
            'AttributeName': 'Company Name',
            'AttributeType': 'S'  # String
        },
        {
            'AttributeName': 'Date',
            'AttributeType': 'S'  # String
        }
    ],
    KeySchema=[
        {
            'AttributeName': 'Company Name',
            'KeyType': 'HASH'  # Partition Key
        },
        {
            'AttributeName': 'Date',
            'KeyType': 'RANGE'  # Sort Key
        }
    ],
    ProvisionedThroughput={
        'ReadCapacityUnits': 5,
        'WriteCapacityUnits': 5
    }
)

print("⏳ Table creation started. Waiting for it to become ACTIVE...")

# Step 4: Wait until table becomes active
table.meta.client.get_waiter('table_exists').wait(TableName='StockMkt')

# Step 5: Confirm status
table = dynamodb.Table('StockMkt')
print("✅ Table is now:", table.table_status)
```

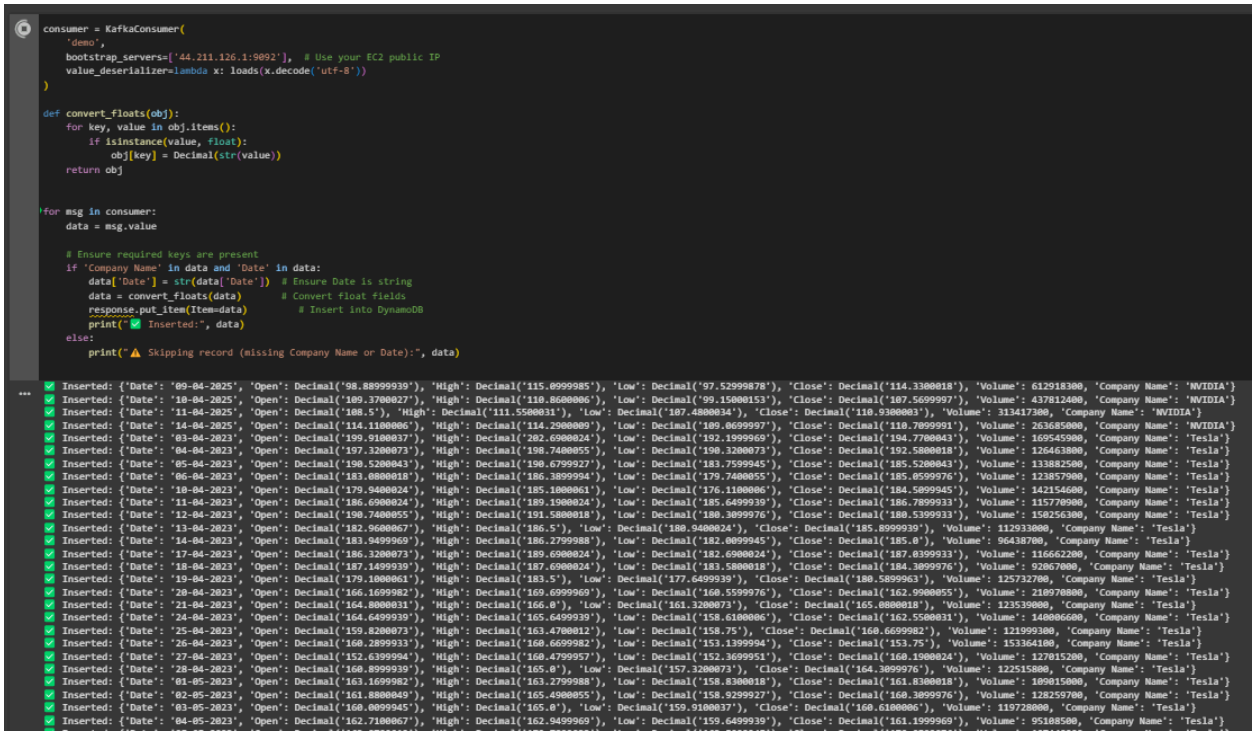
```
⏳ Table creation started. Waiting for it to become ACTIVE...
✅ Table is now: ACTIVE
```

---

10. Kafka Consumer → DynamoDB Insert (Python)

```
from kafka import KafkaConsumer
from json import loads
from decimal import Decimal

consumer = KafkaConsumer(
    'demo',
    bootstrap_servers=['<EC2_PUBLIC_IP>:9092'],
```

```
    value_deserializer=lambda x: loads(x.decode('utf-8'))
)

table = dynamodb.Table('StockMkt')

def convert_floats(obj):
    for k, v in obj.items():
        if isinstance(v, float):
            obj[k] = Decimal(str(v))
    return obj

for msg in consumer:
    data = msg.value
    if 'Company Name' in data and 'Date' in data:
        data['Date'] = str(data['Date'])
        table.put_item(Item=convert_floats(data))
        print("Inserted:", data)
    else:
        print("Skipping:", data)
```



```
consumer = KafkaConsumer(
    'demo',
    bootstrap_servers=['44.211.126.1:9092'],  # Use your EC2 public IP
    value_deserializer=lambda x: loads(x.decode('utf-8'))
)

def convert_floats(obj):
    for key, value in obj.items():
        if isinstance(value, float):
            obj[key] = Decimal(str(value))
    return obj


for msg in consumer:
    data = msg.value

    # Ensure required keys are present
    if 'Company Name' in data and 'Date' in data:
        data['Date'] = str(data['Date'])  # Ensure Date is string
        data = convert_floats(data)       # Convert float fields
        response.put_item(Item=data)         # Insert into DynamoDB
        print("✅ Inserted:", data)
    else:
        print("⚠️ Skipping record (missing Company Name or Date):", data)
```

```
✅ Inserted: {'Date': '09-04-2025', 'Open': Decimal('98.88999939'), 'High': Decimal('115.8999985'), 'Low': Decimal('97.52999878'), 'Close': Decimal('114.3300018'), 'Volume': 612918300, 'Company Name': 'NVIDIA'}
✅ Inserted: {'Date': '10-04-2025', 'Open': Decimal('109.3700027'), 'High': Decimal('110.8600006'), 'Low': Decimal('99.15000153'), 'Close': Decimal('107.5699997'), 'Volume': 437812400, 'Company Name': 'NVIDIA'}
✅ Inserted: {'Date': '11-04-2025', 'Open': Decimal('108.5'), 'High': Decimal('111.5500031'), 'Low': Decimal('107.4800034'), 'Close': Decimal('110.9300003'), 'Volume': 313417300, 'Company Name': 'NVIDIA'}
✅ Inserted: {'Date': '14-04-2025', 'Open': Decimal('114.1100006'), 'High': Decimal('114.2900009'), 'Low': Decimal('109.0699997'), 'Close': Decimal('110.7099991'), 'Volume': 263685000, 'Company Name': 'NVIDIA'}
✅ Inserted: {'Date': '03-04-2023', 'Open': Decimal('199.9100037'), 'High': Decimal('202.6900024'), 'Low': Decimal('192.1999969'), 'Close': Decimal('194.7700043'), 'Volume': 169545900, 'Company Name': 'Tesla'}
✅ Inserted: {'Date': '04-04-2023', 'Open': Decimal('197.3200073'), 'High': Decimal('198.7400055'), 'Low': Decimal('190.3200073'), 'Close': Decimal('192.5800018'), 'Volume': 126463800, 'Company Name': 'Tesla'}
✅ Inserted: {'Date': '05-04-2023', 'Open': Decimal('190.5200043'), 'High': Decimal('190.6799927'), 'Low': Decimal('183.7599945'), 'Close': Decimal('185.5200043'), 'Volume': 133882500, 'Company Name': 'Tesla'}
✅ Inserted: {'Date': '06-04-2023', 'Open': Decimal('183.0800018'), 'High': Decimal('186.3899994'), 'Low': Decimal('179.7400055'), 'Close': Decimal('185.0599976'), 'Volume': 123857900, 'Company Name': 'Tesla'}
✅ Inserted: {'Date': '10-04-2023', 'Open': Decimal('179.9400024'), 'High': Decimal('185.1000061'), 'Low': Decimal('176.1100006'), 'Close': Decimal('184.5099945'), 'Volume': 142154600, 'Company Name': 'Tesla'}
✅ Inserted: {'Date': '11-04-2023', 'Open': Decimal('186.6900024'), 'High': Decimal('189.1900024'), 'Low': Decimal('185.6499939'), 'Close': Decimal('186.7899933'), 'Volume': 115770900, 'Company Name': 'Tesla'}
✅ Inserted: {'Date': '12-04-2023', 'Open': Decimal('190.7400055'), 'High': Decimal('191.5800018'), 'Low': Decimal('180.3099976'), 'Close': Decimal('180.5399933'), 'Volume': 150256300, 'Company Name': 'Tesla'}
✅ Inserted: {'Date': '13-04-2023', 'Open': Decimal('182.9600067'), 'High': Decimal('186.5'), 'Low': Decimal('180.9400024'), 'Close': Decimal('185.8999939'), 'Volume': 112933000, 'Company Name': 'Tesla'}
✅ Inserted: {'Date': '14-04-2023', 'Open': Decimal('183.9499969'), 'High': Decimal('186.2799988'), 'Low': Decimal('182.0099945'), 'Close': Decimal('185.0'), 'Volume': 96438700, 'Company Name': 'Tesla'}
✅ Inserted: {'Date': '17-04-2023', 'Open': Decimal('186.3200073'), 'High': Decimal('189.6900024'), 'Low': Decimal('182.6900024'), 'Close': Decimal('187.0399933'), 'Volume': 116662200, 'Company Name': 'Tesla'}
✅ Inserted: {'Date': '18-04-2023', 'Open': Decimal('187.1499939'), 'High': Decimal('187.6900024'), 'Low': Decimal('183.5800018'), 'Close': Decimal('184.3099976'), 'Volume': 92067000, 'Company Name': 'Tesla'}
✅ Inserted: {'Date': '19-04-2023', 'Open': Decimal('179.1000061'), 'High': Decimal('183.5'), 'Low': Decimal('177.6499939'), 'Close': Decimal('180.5899963'), 'Volume': 125732700, 'Company Name': 'Tesla'}
✅ Inserted: {'Date': '20-04-2023', 'Open': Decimal('166.1699982'), 'High': Decimal('169.6999969'), 'Low': Decimal('160.5599976'), 'Close': Decimal('162.9900055'), 'Volume': 210970800, 'Company Name': 'Tesla'}
✅ Inserted: {'Date': '21-04-2023', 'Open': Decimal('164.8000031'), 'High': Decimal('166.0'), 'Low': Decimal('161.3200073'), 'Close': Decimal('165.0800018'), 'Volume': 123539000, 'Company Name': 'Tesla'}
✅ Inserted: {'Date': '24-04-2023', 'Open': Decimal('164.6499939'), 'High': Decimal('165.6499939'), 'Low': Decimal('158.6100006'), 'Close': Decimal('162.5500031'), 'Volume': 140006600, 'Company Name': 'Tesla'}
✅ Inserted: {'Date': '25-04-2023', 'Open': Decimal('159.8200073'), 'High': Decimal('163.4700012'), 'Low': Decimal('158.75'), 'Close': Decimal('160.6699982'), 'Volume': 121999300, 'Company Name': 'Tesla'}
✅ Inserted: {'Date': '26-04-2023', 'Open': Decimal('160.2899933'), 'High': Decimal('160.6699982'), 'Low': Decimal('153.1399994'), 'Close': Decimal('153.75'), 'Volume': 153364100, 'Company Name': 'Tesla'}
✅ Inserted: {'Date': '27-04-2023', 'Open': Decimal('152.6399994'), 'High': Decimal('160.4799957'), 'Low': Decimal('152.3699951'), 'Close': Decimal('160.1900024'), 'Volume': 127015200, 'Company Name': 'Tesla'}
✅ Inserted: {'Date': '28-04-2023', 'Open': Decimal('160.8999939'), 'High': Decimal('165.0'), 'Low': Decimal('157.3200073'), 'Close': Decimal('164.3099976'), 'Volume': 122515800, 'Company Name': 'Tesla'}
✅ Inserted: {'Date': '01-05-2023', 'Open': Decimal('163.1699982'), 'High': Decimal('163.2799988'), 'Low': Decimal('158.8300018'), 'Close': Decimal('161.8300018'), 'Volume': 109015000, 'Company Name': 'Tesla'}
✅ Inserted: {'Date': '02-05-2023', 'Open': Decimal('161.8800049'), 'High': Decimal('165.4900055'), 'Low': Decimal('158.9299927'), 'Close': Decimal('160.3099976'), 'Volume': 128259700, 'Company Name': 'Tesla'}
✅ Inserted: {'Date': '03-05-2023', 'Open': Decimal('160.0099945'), 'High': Decimal('165.8'), 'Low': Decimal('159.9100037'), 'Close': Decimal('160.6100006'), 'Volume': 119728000, 'Company Name': 'Tesla'}
✅ Inserted: {'Date': '04-05-2023', 'Open': Decimal('162.7100067'), 'High': Decimal('162.9499969'), 'Low': Decimal('159.6499939'), 'Close': Decimal('161.1999969'), 'Volume': 95108500, 'Company Name': 'Tesla'}
```

---

11. Verify DynamoDB Records

From Python:

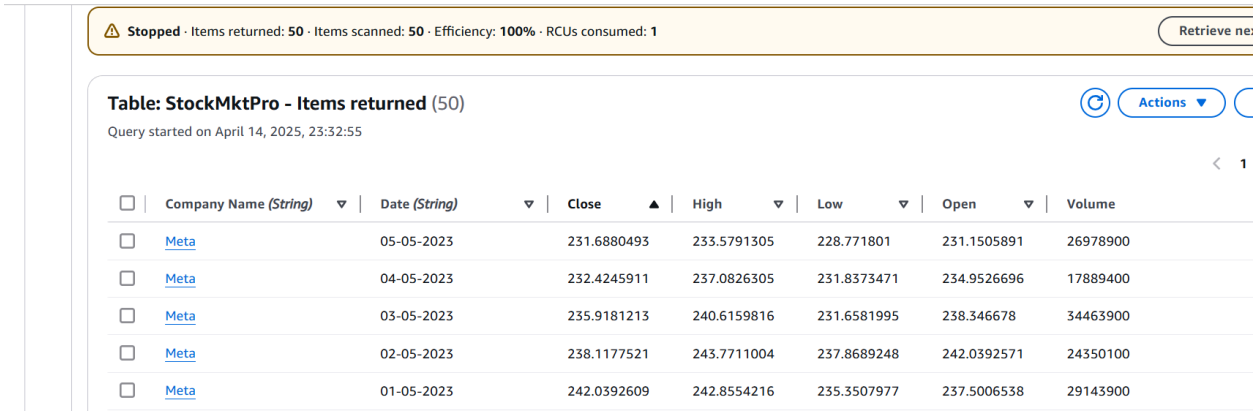
```
response = table.scan(Limit=5)
```

```
for item in response['Items']:
    print(item)
```

From Console:

- Go to DynamoDB Console → Tables → StockMkt → Explore Table Items
- Apply queries to get different Views:

VIEW 1: DATA FOR COMPANY META SORTED BY CLOSING PRICE OF STOCK.



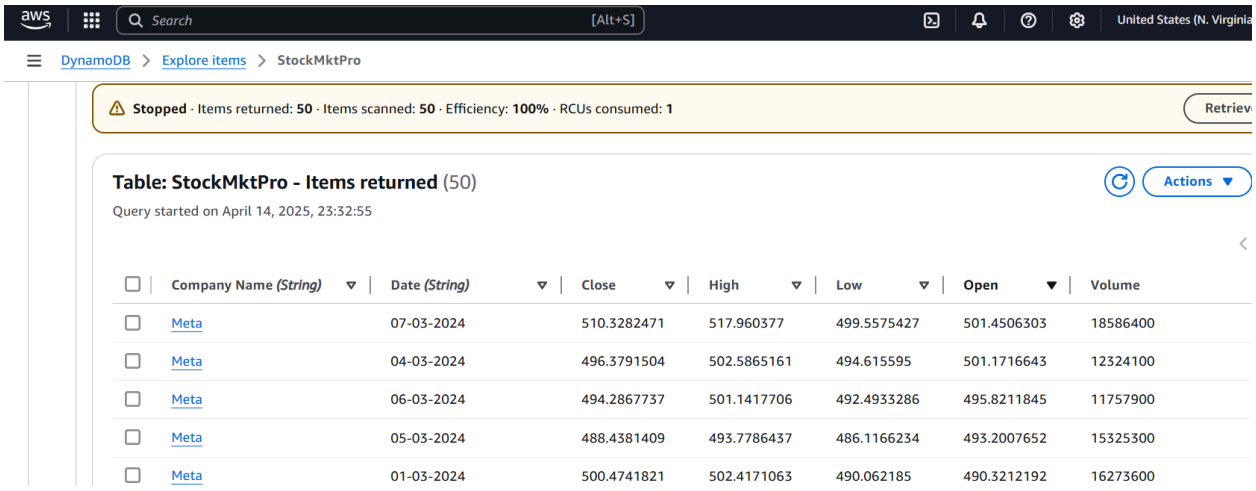
Stopped · Items returned: 50 · Items scanned: 50 · Efficiency: 100% · RCUs consumed: 1

Table: StockMktPro - Items returned (50)

Query started on April 14, 2025, 23:32:55

Company Name (String)	Date (String)	Close	High	Low	Open	Volume
Meta	05-05-2023	231.6880493	233.5791305	228.771801	231.1505891	26978900
Meta	04-05-2023	232.4245911	237.0826305	231.8373471	234.9526696	17889400
Meta	03-05-2023	235.9181213	240.6159816	231.6581995	238.346678	34463900
Meta	02-05-2023	238.1177521	243.7711004	237.8689248	242.0392571	24350100
Meta	01-05-2023	242.0392609	242.8554216	235.3507977	237.5006538	29143900

VIEW 2: DATA FOR COMPANY META SORTED BY OPENING PRICE OF STOCK.



DynamoDB > Explore items > StockMktPro

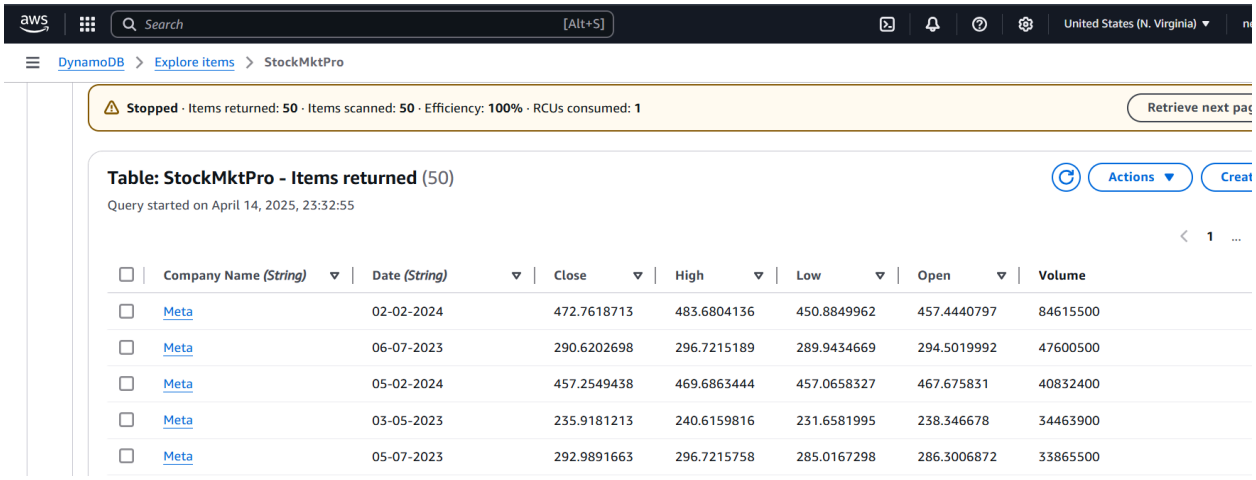
Stopped · Items returned: 50 · Items scanned: 50 · Efficiency: 100% · RCUs consumed: 1

Table: StockMktPro - Items returned (50)

Query started on April 14, 2025, 23:32:55

Company Name (String)	Date (String)	Close	High	Low	Open	Volume
Meta	07-03-2024	510.3282471	517.960377	499.5575427	501.4506303	18586400
Meta	04-03-2024	496.3791504	502.5865161	494.615595	501.1716643	12324100
Meta	06-03-2024	494.2867737	501.1417706	492.4933286	495.8211845	11757900
Meta	05-03-2024	488.4381409	493.7786437	486.1166234	493.2007652	15325300
Meta	01-03-2024	500.4741821	502.4171063	490.062185	490.3212192	16273600

VIEW 3: DATA FOR COMPANY META SORTED BY VOLUME OF STOCK.



Company Name (String)	Date (String)	Close	High	Low	Open	Volume
Meta	02-02-2024	472.7618713	483.6804136	450.8849962	457.4440797	84615500
Meta	06-07-2023	290.6202698	296.7215189	289.9434669	294.5019992	47600500
Meta	05-02-2024	457.2549438	469.6863444	457.0658327	467.675831	40832400
Meta	03-05-2023	235.9181213	240.6159816	231.6581995	238.346678	34463900
Meta	05-07-2023	292.9891663	296.7215758	285.0167298	286.3006872	33865500

We now have a full working pipeline: **CSV → Kafka Producer → EC2 Kafka Broker → Kafka Consumer → DynamoDB; where queries can be performed on real time data at DynamoDB as shown.**