Spark Day 3 Assignment

1. Pipeline 1 — RDBMS → Spark → Amazon Keyspaces (Cassandra)

Purpose: Migrate and denormalize relational data from RDS MySQL to Amazon Keyspaces for analytics and high-performance queries.

Architecture:

- Source: AWS RDS MySQL (3 normalized tables: customers, orders, order_items)
- Processing: Apache Spark (Scala) with JDBC and Cassandra Connector
- Target: Amazon Keyspaces table sales_data in keyspace spark_keyspace

Database Schema:

```
CREATE TABLE customers (
   customer_id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
   name        VARCHAR(100) NOT NULL,
   email       VARCHAR(150) UNIQUE NOT NULL,
   city        VARCHAR(80)  NOT NULL
);

CREATE TABLE orders (
   order_id    INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
   customer_id INT UNSIGNED NOT NULL,
   order_date  DATE NOT NULL,
   amount      DECIMAL(12,2) NOT NULL,
   CONSTRAINT fk_orders_customer
      FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
      ON DELETE RESTRICT ON UPDATE CASCADE
);

CREATE TABLE order_items (
   item_id       INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
   order_id      INT UNSIGNED NOT NULL,
   product_name  VARCHAR(200) NOT NULL,
   quantity      INT UNSIGNED NOT NULL DEFAULT 1,
   CONSTRAINT fk_order_items_order
      FOREIGN KEY (order_id) REFERENCES orders(order_id)
      ON DELETE CASCADE ON UPDATE CASCADE
);
```

Table Relationship:

```
customers (1) ────────< orders (M)
              │
              │
              └─────────< order_items (M)
```

Data Flow:

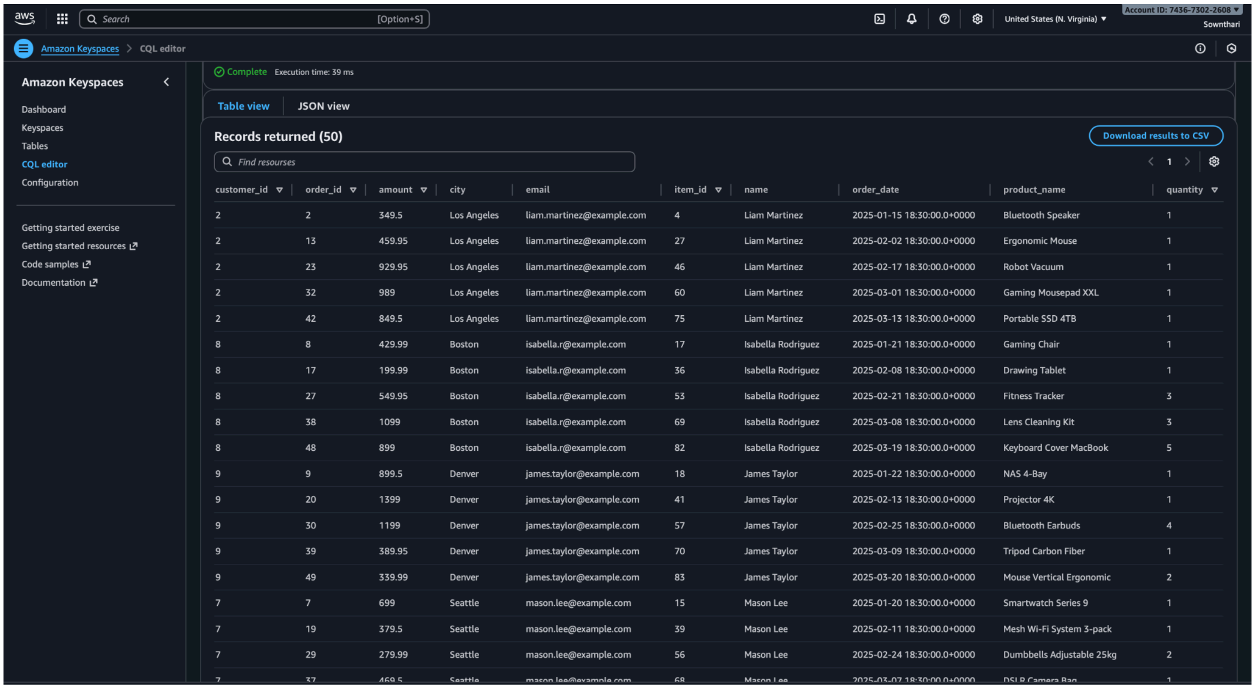
1. Extract: Read 3 tables from RDS MySQL via JDBC connector
2. Transform: Perform inner joins on customer_id and order_id to create denormalized view
3. Load: Write to Keyspaces using Spark Cassandra Connector

Key Configuration:

- SSL enabled for Keyspaces connection (port 9142)
- IAM service-specific credentials for authentication
- Truststore configured for SSL certificate validation

Output Schema:

- Partition Key: customer_id (int)
- Clustering Key: order_id (int)
- Additional columns: name, email, city, order_date, amount, item_id, product_name, quantity



✓ Complete Execution time: 39 ms

Records returned (50)

customer_id	order_id	amount	city	email	item_id	name	order_date	product_name	quantity
2	2	349.5	Los Angeles	liam.martinez@example.com	4	Liam Martinez	2025-01-15 18:30:00.0+0000	Bluetooth Speaker	1
2	13	459.95	Los Angeles	liam.martinez@example.com	27	Liam Martinez	2025-02-02 18:30:00.0+0000	Ergonomic Mouse	1
2	23	929.95	Los Angeles	liam.martinez@example.com	46	Liam Martinez	2025-02-17 18:30:00.0+0000	Robot Vacuum	1
2	32	989	Los Angeles	liam.martinez@example.com	60	Liam Martinez	2025-03-01 18:30:00.0+0000	Gaming Mousepad XXL	1
2	42	849.5	Los Angeles	liam.martinez@example.com	75	Liam Martinez	2025-03-13 18:30:00.0+0000	Portable SSD 4TB	1
8	8	429.99	Boston	isabella.r@example.com	17	Isabella Rodriguez	2025-01-21 18:30:00.0+0000	Gaming Chair	1
8	17	199.99	Boston	isabella.r@example.com	36	Isabella Rodriguez	2025-02-08 18:30:00.0+0000	Drawing Tablet	1
8	27	549.95	Boston	isabella.r@example.com	53	Isabella Rodriguez	2025-02-21 18:30:00.0+0000	Fitness Tracker	3
8	38	1099	Boston	isabella.r@example.com	69	Isabella Rodriguez	2025-03-08 18:30:00.0+0000	Lens Cleaning Kit	3
8	48	899	Boston	isabella.r@example.com	82	Isabella Rodriguez	2025-03-19 18:30:00.0+0000	Keyboard Cover MacBook	5
9	9	899.5	Denver	james.taylor@example.com	18	James Taylor	2025-01-22 18:30:00.0+0000	NAS 4-Bay	1
9	20	1399	Denver	james.taylor@example.com	41	James Taylor	2025-02-13 18:30:00.0+0000	Projector 4K	1
9	30	1199	Denver	james.taylor@example.com	57	James Taylor	2025-02-25 18:30:00.0+0000	Bluetooth Earbuds	4
9	39	389.95	Denver	james.taylor@example.com	70	James Taylor	2025-03-09 18:30:00.0+0000	Tripod Carbon Fiber	1
9	49	339.99	Denver	james.taylor@example.com	83	James Taylor	2025-03-20 18:30:00.0+0000	Mouse Vertical Ergonomic	2
7	7	699	Seattle	mason.lee@example.com	15	Mason Lee	2025-01-20 18:30:00.0+0000	Smartwatch Series 9	1
7	19	379.5	Seattle	mason.lee@example.com	39	Mason Lee	2025-02-11 18:30:00.0+0000	Mesh Wi-Fi System 3-pack	1
7	29	279.99	Seattle	mason.lee@example.com	56	Mason Lee	2025-02-24 18:30:00.0+0000	Dumbbells Adjustable 25kg	2
7	37	469.5	Seattle	mason.lee@example.com	68	Mason Lee	2025-03-07 18:30:00.0+0000	DSLR Camera Bag	1

2.Read Keyspaces → Write Parquet to S3

Purpose: Extract denormalized sales data from Amazon Keyspaces and export to S3 as partitioned Parquet files for analytics and data lake storage.

Architecture:

- **Source**: Amazon Keyspaces table sales_data in keyspace spark_keyspace
- **Processing**: Apache Spark (Scala) with Cassandra Connector
- **Target**: AWS S3 bucket as partitioned Parquet files (s3://sparkdemo-bucket-1
- /sales/parquet/)

Data Flow:

1. **Extract**: Read sales_data from Keyspaces using Spark Cassandra Connector
2. **Transform**: Select subset of columns (customer_id, order_id, amount, product_name, quantity)
3. **Load**: Write to S3 as Parquet format, partitioned by customer_id

Key Configuration:

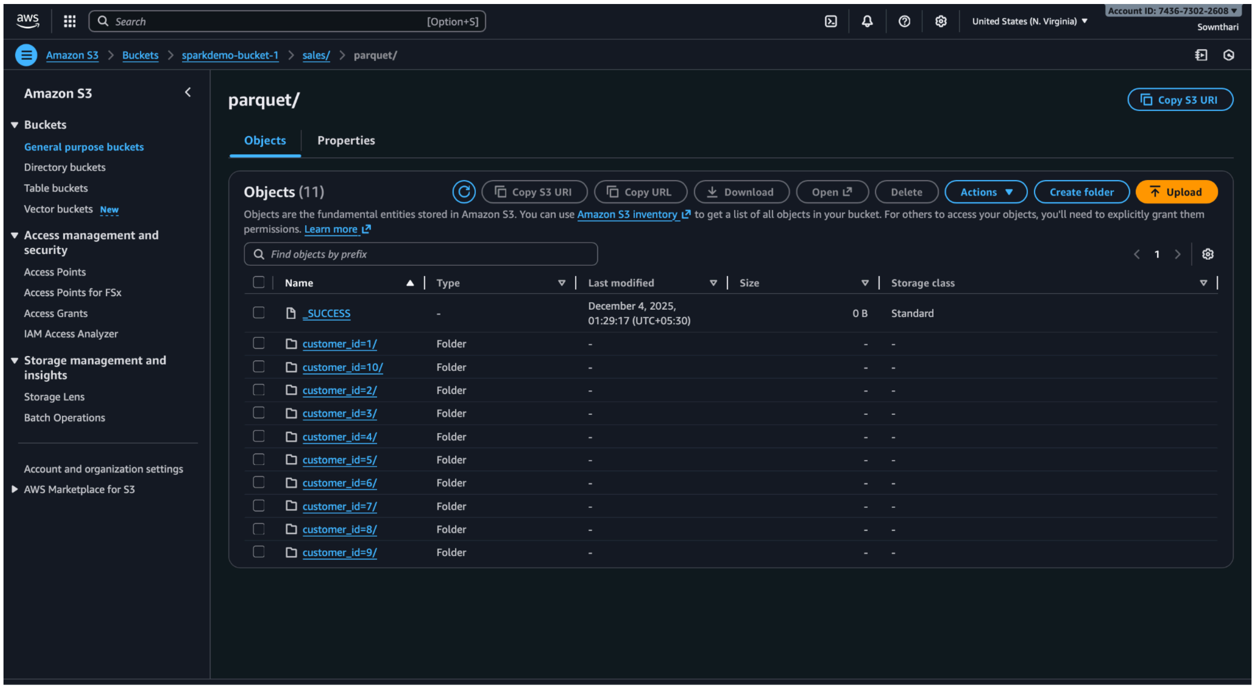
- SSL enabled for Keyspaces connection (port 9142)
- S3A file system with SimpleAWSCredentialsProvider for S3 access
- Extended timeouts for Keyspaces latency handling

Output Structure:

- Format: Parquet (columnar, compressed)
- Partitioning: By customer_id for optimized query performance
- Location: s3a://sparkdemo-bucket-1/sales/parquet/customer_id=X/

Directory Structure:

```
s3://sparkdemo-bucket-1/
│
├── sales/
    └── parquet/
        ├── customer_id=1/
        │   ├── part-00000-xxx.snappy.parquet
        │   └── part-00001-xxx.snappy.parquet
        ├── customer_id=2/
        │   └── part-00000-xxx.snappy.parquet
        └── customer_id=3/
            └── part-00000-xxx.snappy.parquet
```



parquet/

Objects (11)

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

□	Name	Type	Last modified	Size	Storage class
□	_SUCCESS	-	December 4, 2025, 01:29:17 (UTC+05:30)	0 B	Standard
□	customer_id=1/	Folder	-	-	-
□	customer_id=10/	Folder	-	-	-
□	customer_id=2/	Folder	-	-	-
□	customer_id=3/	Folder	-	-	-
□	customer_id=4/	Folder	-	-	-
□	customer_id=5/	Folder	-	-	-
□	customer_id=6/	Folder	-	-	-
□	customer_id=7/	Folder	-	-	-
□	customer_id=8/	Folder	-	-	-
□	customer_id=9/	Folder	-	-	-

3. Read Parquet → Aggregate → Write JSON

Purpose: Read partitioned Parquet sales data from S3, compute product-level aggregations, and export results as JSON for downstream reporting and API consumption.

Architecture:

- **Source**: S3 Parquet files from Pipeline 2 (s3://sparkdemo-bucket-1/sales/parquet/)
- **Processing**: Apache Spark (Scala) with DataFrame aggregations
- **Target**: S3 JSON output (s3://sparkdemo-bucket-1/aggregates/products.json)

Data Flow:

1. **Extract**: Read partitioned Parquet files from S3 (customer_id partitions)
2. **Transform**: Aggregate sales data by product_name
    - Sum quantity → total_quantity
    - Sum amount → total_revenue
3. **Load**: Write aggregated results as single JSON file to S3

Aggregation Logic:

- **Group By**: product_name
- **Metrics**:
    - total_quantity = SUM(quantity)
    - total_revenue = SUM(amount)
- **Sort**: Descending by total_revenue (top revenue products first)

Output Format:

json

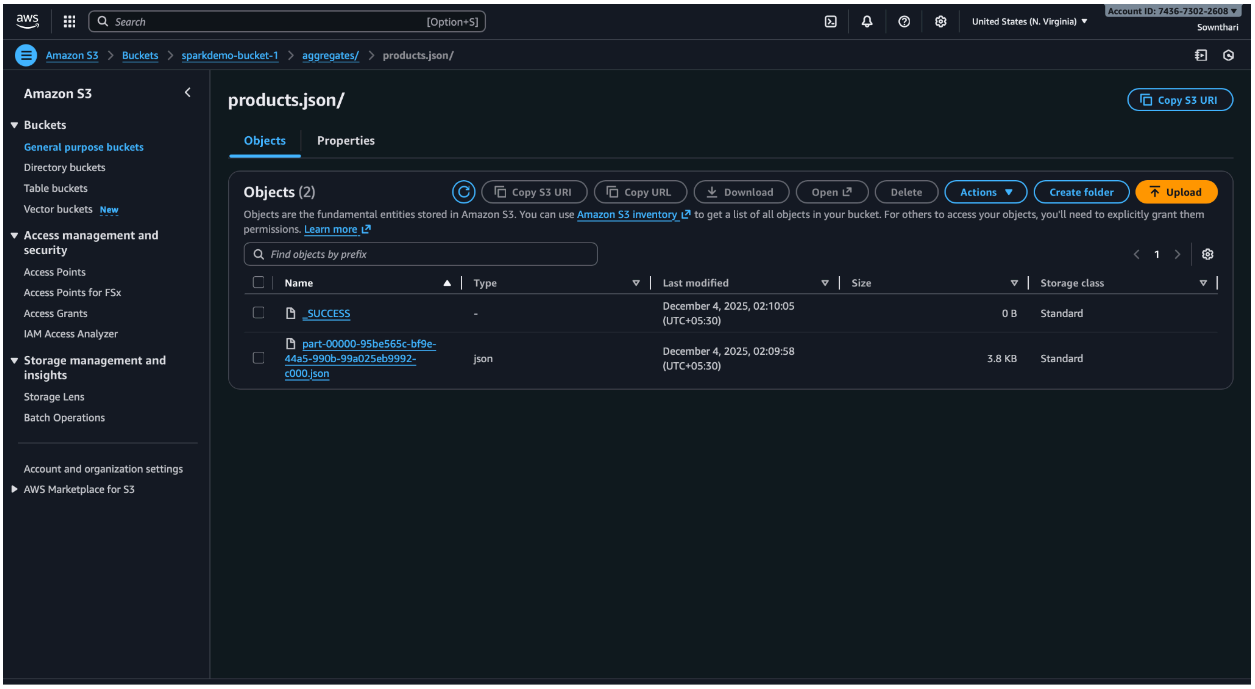
```
{"product_name":"Widget A","total_quantity":150,"total_revenue":45000.0}
{"product_name":"Widget B","total_quantity":120,"total_revenue":36000.0}
```

Key Configuration:

- S3A file system with SimpleAWSCredentialsProvider
- Configuration values stored as variables at top of code for easy management
- .coalesce(1) to produce single output file

Directory Structure:

```
s3://sparkdemo-bucket-1/
│
├── aggregates/
│   └── products.json/
│       └── part-00000-xxx.json
```



products.json/

Objects (2)

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

□	Name	Type	Last modified	Size	Storage class
□	_SUCCESS	-	December 4, 2025, 02:10:05 (UTC+05:30)	0 B	Standard
□	part-00000-95be565c-bf9e-44a5-990b-99a025eb9992-c000.json	json	December 4, 2025, 02:09:58 (UTC+05:30)	3.8 KB	Standard

4.Spark Structured Streaming → Check RDBMS for New Records → Write to Kafka (Avro)

Purpose: Stream newly inserted order records from MySQL into Kafka as Avro messages for real-time processing and event-driven workflows.

Architecture:

- **Source:** MySQL table new_orders
- **Processing:** Spark Structured Streaming (Scala), JDBC incremental scan
- **Target:** Kafka topic orders_avro_topic (Avro-encoded)

Data Flow:

- **Extract:** Poll MySQL every 5 seconds using incrementColumn = order_id

- **Transform:** Convert each new row into Avro (using orders.avsc)
- **Load:** Publish Avro bytes to Kafka with order_id as the message key

Key Configuration:

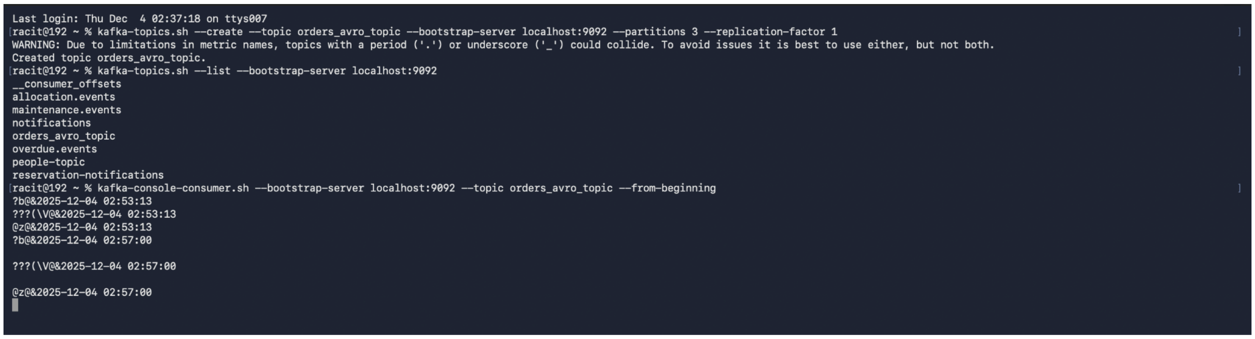
- MySQL JDBC with incremental reading
- Spark trigger interval: 5 seconds
- Avro GenericRecord serialization
- Kafka producer using ByteArraySerializer

Avro Structure: orders.avsc

```
{
  "type": "record",
  "name": "OrderRecord",
  "namespace": "com.retail",
  "fields": [
    { "name": "order_id",    "type": "int" },
    { "name": "customer_id", "type": "int" },
    { "name": "amount",      "type": "double" },
    { "name": "created_at",  "type": "string" }
  ]
}
```

Output Structure:

- **Format:** Avro binary
- **Topic:** orders_avro_topic
- **Key:** order_id
- **Value:** Avro payload containing order_id, customer_id, amount, created_at



```
Last login: Thu Dec  4 02:37:18 on ttys007
racit@192 ~ % kafka-topics.sh --create --topic orders_avro_topic --bootstrap-server localhost:9092 --partitions 3 --replication-factor 1
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 orders_avro_topic.
racit@192 ~ % kafka-topics.sh --list --bootstrap-server localhost:9092
__consumer_offsets
allocation.events
maintenance.events
notifications
orders_avro_topic
overdue.events
people-topic
reservation-notifications
racit@192 ~ % kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic orders_avro_topic --from-beginning
?b@&2025-12-04 02:53:13
???(\V@&2025-12-04 02:53:13
@z@&2025-12-04 02:53:13
?b@&2025-12-04 02:57:00

???(\V@&2025-12-04 02:57:00

@z@&2025-12-04 02:57:00
```

5. Kafka Consumer Stream → Write JSON to S3

Purpose: Consume Avro messages from Kafka in real-time, decode them, and stream as JSON to S3.

Architecture:

- **Source**: Kafka topic orders_avro_topic (Avro binary from Pipeline 4)
- **Processing**: Spark Structured Streaming with Avro deserialization
- **Target**: S3 JSON files (s3://sparkdemo-bucket-1/stream/json/)

Data Flow:

1. **Consume**: Read from Kafka topic in real-time
2. **Decode**: Deserialize Avro binary using schema (order_id, customer_id, amount, created_at)
3. **Load**: Write JSON to S3 every 10 seconds

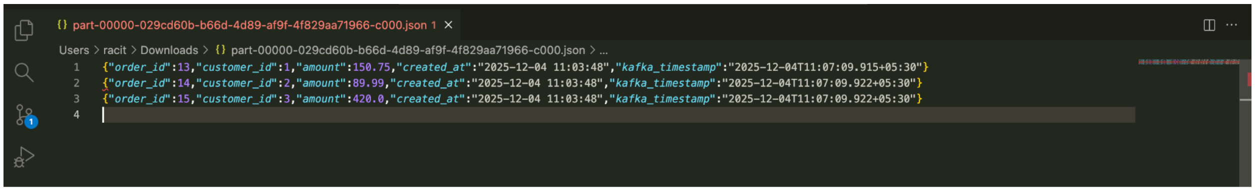
Key Configuration:

- Starting offsets: earliest (reads all messages)
- Trigger: 10-second micro-batches
- Checkpoint: /tmp/kafka-s3-checkpoint (fault tolerance)
- Output mode: Append

Directory Structure:

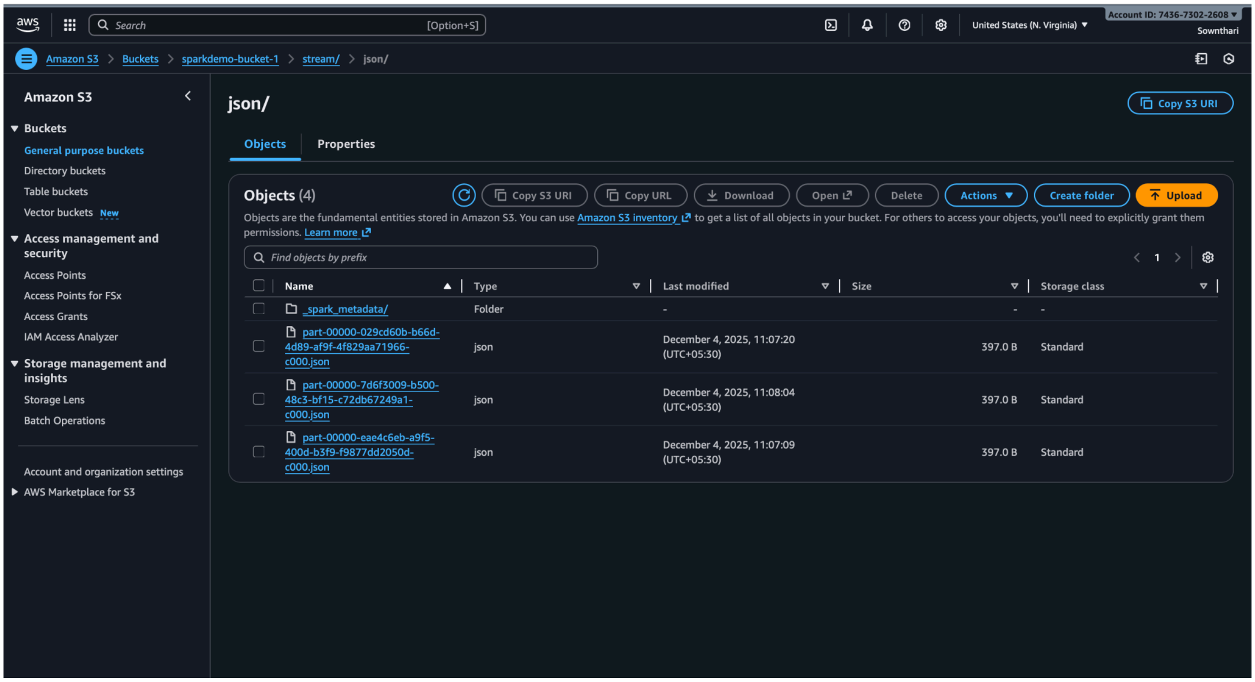
```
s3://sparkdemo-bucket-1/

├── stream/
│   └── json/
│       ├── part-00000-xxx.json
│       ├── part-00001-xxx.json
│       └── part-00002-xxx.json
│
└── checkpoints/
    └── kafka-stream/
        ├── commits/
        ├── offsets/
        └── metadata
```

Output Format:



{} part-00000-029cd60b-b66d-4d89-af9f-4f829aa71966-c000.json 1 ×
Users > racit > Downloads > {} part-00000-029cd60b-b66d-4d89-af9f-4f829aa71966-c000.json > ...

```
{"order_id":13,"customer_id":1,"amount":150.75,"created_at":"2025-12-04 11:03:48","kafka_timestamp":"2025-12-04T11:07:09.915+05:30"}
{"order_id":14,"customer_id":2,"amount":89.99,"created_at":"2025-12-04 11:03:48","kafka_timestamp":"2025-12-04T11:07:09.922+05:30"}
{"order_id":15,"customer_id":3,"amount":420.0,"created_at":"2025-12-04 11:03:48","kafka_timestamp":"2025-12-04T11:07:09.922+05:30"}
```



Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Find objects by prefix

	Name	Type	Last modified	Size	Storage class
□	_spark_metadata/	Folder	-	-	-
□	part-00000-029cd60b-b66d-4d89-af9f-4f829aa71966-c000.json	json	December 4, 2025, 11:07:20 (UTC+05:30)	397.0 B	Standard
□	part-00000-7d6f3009-b500-48c3-bf15-c72db67249a1-c000.json	json	December 4, 2025, 11:08:04 (UTC+05:30)	397.0 B	Standard
□	part-00000-eae4c6eb-a9f5-400d-b3f9-f9877dd2050d-c000.json	json	December 4, 2025, 11:07:09 (UTC+05:30)	397.0 B	Standard