Amazon Kinesis Setup and Usage Guide

1. What is Amazon Kinesis?

Amazon Kinesis is a fully managed service for real-time data streaming at scale. It enables the collection, processing, and analysis of real-time data to gain timely insights and react to new information quickly.

---

2. Core Components

1. **Kinesis Data Streams**:
    - Stores real-time data streams for processing by consumers.
    - Retains data for 24 hours (up to 7 days).
2. **Kinesis Data Firehose**:
    - Delivers streaming data to destinations like Amazon S3, Redshift, or Elasticsearch.
3. **Kinesis Data Analytics**:
    - Allows SQL queries on streaming data for real-time analytics.

---

3. Steps to Set Up and Use Amazon Kinesis

Step 1: AWS Account Setup

1. Sign up or log in to your AWS Management Console.
2. Ensure you have permissions to create Kinesis resources.

Step 2: Create a Kinesis Data Stream

1. Open the **Amazon Kinesis** service from the AWS Management Console.
2. Select **Data Streams**.
3. Click on **Create Stream**.
    - Name your stream.
    - Specify the number of shards (each shard supports 1 MB/sec write and 2 MB/sec read throughput).
4. Click **Create Stream** and wait for the stream to become active.

Step 3: Configure a Producer to Send Data

1. Install the AWS SDK for Java in your application.
2. Use the SDK to send data to the stream.

```
AmazonKinesis kinesisClient =
AmazonKinesisClientBuilder.defaultClient();
String streamName = "my-stream";
String data = "Sample data";
PutRecordRequest request = new PutRecordRequest()
    .withStreamName(streamName)
    .withData(ByteBuffer.wrap(data.getBytes()))
    .withPartitionKey("partition-key");
kinesisClient.putRecord(request);
```

Step 4: Set Up a Consumer to Process Data

1. Use the **Kinesis Client Library (KCL)** to consume data.
2. Implement a record processor to handle data.

```
public class SampleRecordProcessor implements IRecordProcessor {
    @Override
    public void initialize(InitializationInput initializationInput) {
        // Initialization logic
    }

    @Override
    public void processRecords(ProcessRecordsInput processRecordsInput) {
        for (KinesisClientRecord record : processRecordsInput.getRecords()) {
            String data = new String(record.data().array(),
StandardCharsets.UTF_8);
            System.out.println("Processed record: " + data);
        }
    }

    @Override
    public void shutdown(ShutdownInput shutdownInput) {
        // Cleanup logic
    }
}
```

Register the record processor with the KCL.

Step 5: Use Kinesis Data Firehose (Optional)

1. In the AWS Console, create a Firehose delivery stream.
2. Configure the destination (e.g., Amazon S3).
3. Set up a transformation Lambda function if needed.
4. Send data to the Firehose using the AWS SDK.

Step 6: Monitor Your Stream

1. Use Amazon CloudWatch to monitor Kinesis Streams metrics like data throughput and shard utilization.
2. Set alarms for thresholds.

---

4. Integrating with Java Spring Boot

Step 1: Add Dependencies

Add the AWS SDK and Spring Cloud AWS dependencies to your pom.xml.

```
<dependency>
   <groupId>software.amazon.awssdk</groupId>
   <artifactId>kinesis</artifactId>
   <version>2.x.x</version>
</dependency>
<dependency>
   <groupId>org.springframework.cloud</groupId>
   <artifactId>spring-cloud-starter-aws</artifactId>
</dependency>
```

Step 2: Configure AWS Credentials

Set up your AWS credentials using a properties file or environment variables.

Step 3: Implement Producer and Consumer Services

1. **Producer Service**

```
@Service
public class KinesisProducer {
   private final String streamName = "my-stream";

   public void sendData(String data) {
      AmazonKinesis kinesisClient =
AmazonKinesisClientBuilder.defaultClient();
      PutRecordRequest request = new PutRecordRequest()
         .withStreamName(streamName)
```

```
            .withData(ByteBuffer.wrap(data.getBytes()))
            .withPartitionKey("partition-key");
        kinesisClient.putRecord(request);
    }
}
```

2. **Consumer Service** Use the Kinesis Client Library to poll and process data.

Step 4: Testing the Application

- Test by sending data to the producer and verifying the consumer processes it.
- Use logs and CloudWatch for debugging.

---

5. Monitoring and Scaling

- Use CloudWatch metrics for throughput and latency.
- Adjust shard count to handle increased data load.

---

6. Summary

Amazon Kinesis simplifies real-time data streaming and processing. By integrating it with a Java Spring Boot application, developers can build scalable, real-time solutions for various use cases like analytics, monitoring, and data pipelines.