**Introduction**

Amazon Kinesis is a service that helps handle real-time streaming data by processing and analyzing it at scale. This service is very commonly used in many different architectures, including IoT and telemetrics.

In this lab, we're going to build a Kinesis stream which will receieve events we're producing with an AWS Lambda function, and then consume them with another Lambda function. Finally, we'll build a delivery stream using Amazon Kinesis Data Firehose to push our events into an S3 bucket.

**Solution**

**Log in to the Lab Environment**

1. To avoid issues with the lab, open a new Incognito or Private browser window to log in to the lab. This ensures that your personal account credentials, which may be active in your main window, are not used for the lab.
2. Log in to the AWS Management Console using the credentials provided on the lab instructions page. Make sure you're using the *us-east-1* region.

**Create a Kinesis Data Stream**

1. In the search bar on top, type "Kinesis" to search for the Amazon Kinesis service.
2. Click on the **Amazon Kinesis** result to go directly to the Kinesis service.
3. Click on the **Data streams** button in the sidebar.
4. Click on the **Create data stream** button to launch the wizard to create the Data Stream.
5. Enter the following details for the new Data Stream:
   * Data stream name: TelemetricsStream
   * Capacity mode: On-demand
6. Click on the **Create data stream** button to confirm the creation of the new Data Stream.

**Develop the Kinesis Producer Lambda Function**

1. In the search bar on top, type "Lambda" to search for the AWS Lambda service.
2. Click on the **Lambda** result to go directly to the Lambda service on a new tab.
3. Click on the **Create function** button to create a new AWS Lambda function
4. Enter the following details for the new Lambda function:
   * Function name: produceKinesisEvents
   * Runtime: Python 3.11 (or the latest)
5. Click the **Change default execution drop** toggle to manually change the execution role.
6. Change the radio button to **Use an existing role**.
7. From the dropdown, select the IAM role containing the name KinesisLambdaRole, which will provide the necessary privileges to the Lambda function.
8. Click on the **Create function** button to create the function.
9. Once the page has loaded, update the following code into the Lambda function:
10. import json
11. import boto3
12. def lambda\_handler(event, context):
13. client = boto3.client('kinesis')
14. data = {
15. "id": "0",
16. "latitude": "0",
17. "longtitude": "0"
18. }
19. response = client.put\_record(
20. StreamName="TelemetricsStream",
21. PartitionKey="geolocation",
22. Data=json.dumps(data)
23. )
24. return response
25. Click on the **Deploy** button to save and deploy the new version of the Lambda function.
26. Click on the **Test** button to prepare to test the functions execution.
27. Enter a short name for the test event (like *test*), and click on the **Save** button to save the test event.
28. Click on the **Test** button again to trigger the test
29. Confirm that the execution results returned with a **HTTPStatusCode** of 200.

**Help!**

Did your function return an error? Double-check the execution role you set when creating your Lamdba Function. Since the Lambda Function is sending a record to our Kinesis Data Stream, it needs the extra permissions, or it will trigger an error.

1. Return to the Lambda function code, and update the code with the following to make it return a continuous stream of events:
2. import json
3. import boto3
4. import uuid
5. import random
6. import time
7. def lambda\_handler(event, context):
8. client = boto3.client('kinesis')
10. while True:
11. data = {
12. "id": str(uuid.uuid4()),
13. "latitude": random.uniform(-90, 90),
14. "longtitude": random.uniform(0, 180)
15. }
17. response = client.put\_record(
18. StreamName="TelemetricsStream",
19. PartitionKey="geolocation",
20. Data=json.dumps(data)
21. )
23. print(response)
25. time.sleep(random.random())
26. Navigate to the **Configuration** tab of your Lambda Function, and ensure the **General configuration** sidebar tab is selected.
27. Click on the **Edit** button to change the configuration settings of your Lambda function.
28. Change the **Timeout** value to **30** seconds, then click on the **Save** button to update the function configuration.
29. Test the function again, and ensure it runs for the 30 seconds before timing out.

**Develop the Kinesis Consumer Lambda Function**

1. Open a new tab on the Lambda Functions page.
2. Click on the **Create function** button to create a new AWS Lambda function
3. Enter the following details for the new Lambda function:
   * Function name: consumeKinesisEvents
   * Runtime: Python 3.11 (or the latest)
4. Click the **Change default execution drop** toggle to manually change the execution role.
5. Change the radio button to **Use an existing role**.
6. From the dropdown, select the IAM role containing the name KinesisLambdaRole, which will provide the necessary privileges to the Lambda function.
7. Click on the **Create function** button to create the function.
8. Once the page has loaded, click on the **Add trigger** button.
9. From the dropdown, select **Kinesis**.
10. Enter the following details:
    * Kinesis stream: kinesis/TelemetricsStream
    * Batch size: 10
11. Click on the **Add** button to confirm the creation of the new trigger.
12. Update the following code into the Lambda function:
13. import json
14. def lambda\_handler(event, context):
15. print(json.dumps(event))
16. Click on the **Deploy** button to save and deploy the new version of the Lambda function.
17. Return to the tab containing the **produceKinesisEvents** function.
18. Click on the **Test** button again to trigger the test.
19. Return to the tab containing the **consumeKinesisEvents** function.
20. Navigate to the **Monitoring** tab of the Lambda function.
21. Click on the **View CloudWatch logs** button to launch CloudWatch Logs in a new tab.
22. Open the current log stream.
23. Verify that the records are present, with the data encoded in Base64.
24. Return to the tab containing the **consumeKinesisEvents** function.
25. Update the following code into the Lambda function:
26. import json
27. import base64
28. def lambda\_handler(event, context):
29. records = []
31. for record in event["Records"]:
32. data = base64.b64decode(record["kinesis"]["data"]).decode()
33. records.append(json.loads(data))
35. output = {
36. "count": str(len(records)),
37. "data": records
38. }
40. print(json.dumps(output))
41. Click on the **Deploy** button to save and deploy the new version of the Lambda function.
42. Return to the tab containing the **produceKinesisEvents** function.
43. Click on the **Test** button again to trigger the test.
44. Return to the tab containing the **consumeKinesisEvents** function.
45. Navigate to the **Monitoring** tab of the Lambda function.
46. Click on the **View CloudWatch logs** button to launch CloudWatch Logs in a new tab.
47. Open the current log stream.
48. Verify that the records are present, with the data decoded in JSON format, containing the various GPS coordinates.

**Setup Kinesis Data Firehose**

1. In the search bar on top, type "Kinesis" to search for the Amazon Kinesis service.
2. Click on the **Amazon Kinesis** result to go directly to the Kinesis service.
3. Click on the **Data Firehose** button in the sidebar.
4. Click on the **Create delivery stream** button to launch the wizard to create the Data Firehose delivery stream.
5. Enter the following details for the new Data Stream:
   * Source: Amazon Kinesis Data Streams
   * Destination: Amazon S3
   * Kinesis data stream: Use **Browse** to find your TelemetricsStream
   * S3 bucket: Use **Browse** to find your firehosedeliverybucket
   * Buffer interval: 60 seconds
6. Click on the **Create deliverys stream** button to confirm the creation of the new Data Firehose.
7. Return to the tab containing the **produceKinesisEvents** function.
8. Navigate to the **Configuration** tab of your Lambda Function, and ensure the **General configuration** sidebar tab is selected.
9. Click on the **Edit** button to change the configuration settings of your Lambda function.
10. Change the **Timeout** value to **5 minutes**, then click on the **Save** button to update the function configuration.
11. Click on the **Test** button again to trigger the test.
12. In the search bar on top, type "S3" to search for the Amazon S3 service.
13. Click on the **Amazon S3** result to go directly to the S3 service.
14. Open the firehosedeliverybucket in S3.
15. Wait up to a minute or so for the first events to arrive
16. Navigate through the date-based folders to find the delivered events
17. Download the file to confirm that it contains the GPS coded events.

**Conclusion**

Congratulations — you just learned how to create an Amazon Kinesis Data Stream, along with publishing events into the stream, and consuming events from the stream! And as a bonus, you've learned about using Kinesis Data Firehose to deliver your events to a destination, like Amazon S3.