



AWS Kinesis Data Stream using Console

AWS Kinesis Data Streams (KDS) is a real-time, scalable, and fully managed service designed to continuously collect, process, and analyze streaming data at a massive scale. It's part of the broader **Amazon Kinesis** suite, which enables real-time data streaming for various use cases, such as real-time analytics, machine learning, and data transformation.

Key Features of AWS Kinesis Data Streams:

1. **Real-time Data Streaming:** Kinesis Data Streams allows ingestion of data in real-time from multiple sources (IoT devices, application logs, social media feeds, etc.) and processes it continuously.
2. **Massive Throughput:** The service supports high volumes of data, where streams can handle terabytes of data per hour from hundreds of thousands of data producers.
3. **Data Durability and Retention:** Data is stored across multiple availability zones, ensuring high availability and durability. Streams can retain data for up to 7 days (default is 24 hours).
4. **Scalable Architecture:** Kinesis Data Streams is highly scalable. You can add or remove shards (units of capacity) to adjust the throughput capacity of your stream.
5. **Custom Processing:** You can consume the stream in real-time using consumer applications such as **AWS Lambda**, **Amazon EC2**, or **Amazon Kinesis Data Analytics**.
6. **Cost-Effective:** Pricing is based on the volume of data you stream and the resources used for processing it.

Common Use Cases:

- **Real-time analytics:** Analyze logs, events, or sensor data as it arrives.
- **Application monitoring:** Collect and process logs and metrics from applications in real-time.
- **Machine learning:** Stream data to a machine learning model for real-time predictions.
- **Data lakes and warehousing:** Continuously stream data into data lakes (e.g., Amazon S3) or warehouses for further storage and analysis.

How It Works:

1. **Data Producers:** Devices or applications send data to the stream in real time. Each data point sent is referred to as a **record**.
2. **Data Stream:** Consists of **shards**, each capable of ingesting data and enabling consumers to process it.
3. **Data Consumers:** Applications, services, or analytics tools read the data from the stream for further processing, storage, or real-time analysis.

This real-time streaming service helps businesses react to data instantly, making it valuable for time-sensitive applications.

Benefits of AWS Kinesis Data Streams:

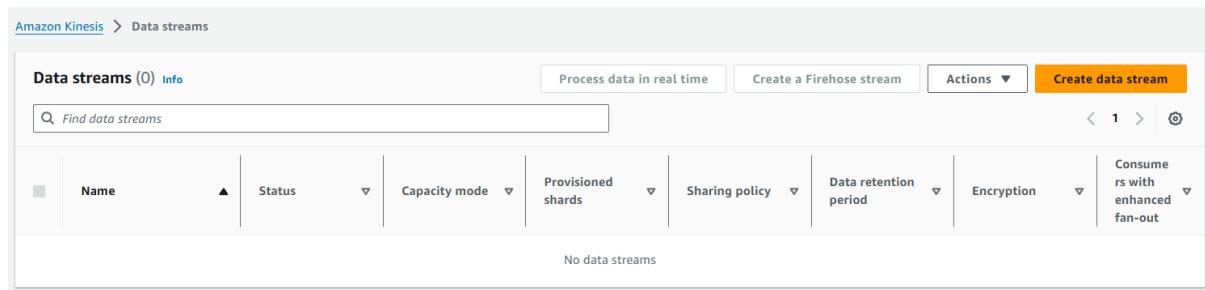
1. **Instant Insights:** Kinesis lets you see and act on data the moment it arrives, which is useful if you want to stay ahead of problems or make real-time decisions.
2. **Scalable:** Whether you have a small or massive amount of data, Kinesis can handle it without slowing down. It grows as your needs grow.
3. **Cost-Effective:** You only pay for what you use, which makes it affordable for businesses of all sizes. You don't need to invest in expensive infrastructure to handle real-time data.
4. **No Need to Manage Servers:** Kinesis is fully managed by AWS, meaning you don't have to worry about setting up or maintaining servers to process your data. This saves you time and effort.
5. **Works with Other Tools:** You can easily connect Kinesis to other AWS services or external tools to store, analyze, or visualize your data in real-time.

In this exercise, you will create a Kinesis Data Stream, an S3 bucket, and three Lambda functions (one for producing data and two for consuming it). The Kinesis Data Stream will collect data from an S3 bucket, where objects are stored. You'll set up event notifications in S3 to trigger a Lambda function whenever a new text file is uploaded. This function reads the file and sends its contents to the Kinesis stream. The other two Lambda functions act as consumers, processing data from the Kinesis stream. You'll monitor the flow of data through logs in CloudWatch and view the processed records using Kinesis Data Viewer.

The end goal is to understand how Kinesis streams data from S3 to different consumers using Lambda functions and track this flow with CloudWatch.

😃 To begin with the Lab:

1. In this lab, we are going to create a data stream in Kinesis using Console. In your AWS Console search for Kinesis and go to Data Streams. Click on Create Data stream.



2. First, give your stream a name then choose Provisioned and for provisioned shards capacity choose 1. After that just click on Create Stream.

Amazon Kinesis > Data streams > Create data stream

Create data stream Info

Data stream configuration

Data stream name
 Acceptable characters are uppercase and lowercase letters, numbers, underscores, hyphens and periods.

Data stream capacity Info

Capacity mode

On-demand
Use this mode when your data stream's throughput requirements are unpredictable and variable. With on-demand mode, your data stream's capacity scales automatically.

Provisioned
Use provisioned mode when you can reliably estimate throughput requirements of your data stream. With provisioned mode, your data stream's capacity is fixed.

Provisioned shards
The total capacity of a stream is the sum of the capacities of its shards. Enter number of provisioned shards to see total data stream capacity.
 [Shard estimator](#)
Minimum: 1, Maximum available: 500, Account quota limit: 500. [Request shard quota increase](#)

Total data stream capacity
Shard capacity is determined by the number of provisioned shards. Each shard ingests up to 1 MiB/second and 1,000 records/second and emits up to 2 MiB/second. If writes and reads exceed capacity, the application will receive throttles.

Write capacity Maximum 1 MiB/second and 1,000 records/second	Read capacity Maximum 2 MiB/second
---	---

ⓘ Provisioned mode has a fixed-throughput pricing model. See [Kinesis pricing for Provisioned mode](#)

3. Below you can see that your data stream has been created.

Data streams (1) Info								
Process data in real time				Create a Firehose stream		Actions ▾		Create data stream
<input type="text"/> Find data streams								
□	Name	▲	Status	▼	Capacity mode	▼	Provisioned shards	▼
□	demo-data-stream			Active	Provisioned		1	No
□					Sharing policy	▼	Data retention period	▼
□					No		1 day	Encryption
□							Disabled	Consumers with enhanced fan-out
□							0	

4. Now go inside your data stream and choose configuration tab, here you scroll down to encryption and click on Edit.

The screenshot shows the 'Configuration' tab selected in the top navigation bar. Below it, there are sections for 'Data stream capacity' and 'Tags - optional'. The 'Encryption' section at the bottom right is highlighted with a red box.

Data stream capacity Info	
Capacity mode Provisioned	Provisioned shards 1
	Write capacity Maximum 1 MiB/second 1,000 records/second
	Read capacity Maximum 2 MiB/second

Tags - optional [Info](#)

Key	Value
No tags No tags associated with this stream	
Manage tags	

Encryption [Info](#)

5. Here you need to enable server-side encryption and choose AWS-managed CMK. Then click on Save Changes.

Edit encryption for demo-data-stream

Encryption Info

Enable server-side encryption

Kinesis Data Stream uses AWS Key Management Service (KMS) to encrypt your data. You can choose the AWS managed customer master key (CMK) to encrypt your data or specify a customer-managed CMK.

Use AWS managed CMK

The AWS managed CMK (aws/kinesis) in your account is created, managed, and used on your behalf by Kinesis Data Streams.

Use customer-managed CMK

Customer-managed CMKs in your AWS account are created, owned, and managed by you.

[Cancel](#)

[Save changes](#)

- Now we are going to create an S3 bucket. For this, you need to give your bucket a unique name and you must enable the Bucket Versioning and just create your bucket.

Bucket Versioning

Versioning is a means of keeping multiple variants of an object in the same bucket. You can use versioning to preserve, retrieve, and restore every version of every object stored in your Amazon S3 bucket. With versioning, you can easily recover from both unintended user actions and application failures. [Learn more](#)

Bucket Versioning

Disable

Enable

- After that we are going to create 3 Lambda functions. We will create 1 lambda function for the Producer and 2 lambda functions for the Consumers.
- Go to lambda and click on create function then give it a name choose Python 3.11 as your runtime and click on create function.

Basic information

Function name

Enter a name that describes the purpose of your function.

Function name must be 1 to 64 characters, must be unique to the Region, and can't include spaces. Valid characters are a-z, A-Z, 0-9, hyphens (-), and underscores (_).

Runtime [Info](#)

Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.

Architecture [Info](#)

Choose the instruction set architecture you want for your function code.

- x86_64
- arm64

9. Once your function is created then you need to paste the code mentioned below and deploy it in your lambda function. In this code, you just need to change the Kinesis Data Stream name.

```
import json
import boto3
import logging

# Configure logging
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger()

# Initialize AWS clients
s3 = boto3.client('s3', region_name='us-east-1')
kinesis = boto3.client('kinesis', region_name='us-east-1')

def lambda_handler(event, context):
    logger.info(json.dumps(event))

    bucket_name = event['Records'][0]['s3']['bucket']['name']
    key_name = event['Records'][0]['s3']['object']['key']

    try:
        data = s3.get_object(Bucket=bucket_name, Key=key_name)
        data_string = data['Body'].read().decode('utf-8')

        payload = {
            'data': data_string
        }

        # Add code here to send data to Kinesis Data Stream

    except Exception as e:
        logger.error(f'Error processing event: {e}')

    return {'statusCode': 200, 'body': 'Success'}
```

```

send_to_kinesis(payload, key_name)

except Exception as e:
    logger.error(e)

def send_to_kinesis(payload, partition_key):
    params = {
        'Data': json.dumps(payload),
        'PartitionKey': partition_key,
        'StreamName': 'demo-data-stream'
    }

    try:
        response = kinesis.put_record(**params)
        logger.info(response)
    except Exception as e:
        logger.error(e)

```

10. Then go to the configuration tab in your lambda function and open the execution role for it. You need to add S3 full access and Kinesis full access to your IAM role attach with your lambda function.



The screenshot shows the AWS Lambda Configuration page. The top navigation bar has tabs: Code, Test, Monitor, Configuration (which is highlighted in blue), Aliases, and Versions. On the left, there's a sidebar with General configuration, Triggers, and Permissions (which is also highlighted in blue). The main content area is titled "Execution role". It shows the "Role name" as "demo-producer-role-a54zyq62" with a link icon. Below this, there's a "Permissions" tab, followed by Trust relationships, Tags, Last Accessed, and Revoke sessions. Under the Permissions tab, there's a section for "Permissions policies (3) Info". It says "You can attach up to 10 managed policies." There's a search bar, a "Filter by Type" dropdown set to "All types", and a pagination indicator showing page 1 of 1. A table lists three managed policies: "AmazonKinesisFullAccess" (AWS managed, 2 attached entities), "AmazonS3FullAccess" (AWS managed, 2 attached entities), and "AWSLambdaBasicExecutionRole-8f6dec5..." (Customer managed, 1 attached entity).

Policy name	Type	Attached entities
AmazonKinesisFullAccess	AWS managed	2
AmazonS3FullAccess	AWS managed	2
AWSLambdaBasicExecutionRole-8f6dec5...	Customer managed	1

11. Now come back to S3, inside your bucket go to Properties scroll down to event notification and click on Create.

Event notifications (0)

Send a notification when specific events occur in your bucket. [Learn more](#)

Name	Event types	Filters	Destination type	Destination
No event notifications				
Choose Create event notification to be notified when a specific event occurs.				
Create event notification				

12. Here you need to give your event a name and, in the Suffix, write **.txt** then in the event types just choose All objects to create events. After that scroll down to the bottom.

General configuration

Event name
upload-object
Event name can contain up to 255 characters.

Prefix - optional
Limit the notifications to objects with key starting with specified characters.
images/

Suffix - optional
Limit the notifications to objects with key ending with specified characters.
.txt

Event types

Specify at least one event for which you want to receive notifications. For each group, you can choose an event type for all events, or you can choose one or more individual events.

Object creation

All object create events
s3:ObjectCreated:
 Put
s3:ObjectCreated:Put

13. Now you need to choose a destination which is the Lambda function and choose your function. Click on Save Changes.
14. So, what it means is whenever an object is created, the producer lambda function will be triggered.

Destination

Info Before Amazon S3 can publish messages to a destination, you must grant the Amazon S3 principal the necessary permissions to call the relevant API to publish messages to an SNS topic, an SQS queue, or a Lambda function. [Learn more](#)

Destination

Choose a destination to publish the event. [Learn more](#)

Lambda function

Run a Lambda function script based on S3 events.

SNS topic

Fanout messages to systems for parallel processing or directly to people.

SQS queue

Send notifications to an SQS queue to be read by a server.

Specify Lambda function

Choose from your Lambda functions

Enter Lambda function ARN

Lambda function

demo-producer

Cancel

Save changes

15. Along with this we are going to create 2 more lambda functions which we are going to call consumer lambda functions. Choose the runtime as Python 3.11.

Basic information

Function name

Enter a name that describes the purpose of your function.

consumer1

Function name must be 1 to 64 characters, must be unique to the Region, and can't include spaces. Valid characters are a-z, A-Z, 0-9, hyphens (-), and underscores (_).

Runtime [Info](#)

Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.



Architecture [Info](#)

Choose the instruction set architecture you want for your function code.

x86_64

arm64

16. Once the function is created then we need to modify the code with the code given below.

```
import base64
import json
```

```

def lambda_handler(event, context):
    print(json.dumps(event))

    for record in event['Records']:
        data = json.loads(base64.b64decode(record['kinesis']['data']).decode('utf-8'))
        print('This is consumer - 1', data)

```

17. For this consumer lambda function we need to modify the IAM role, so go to its configurations tab choose Permissions, and click on the execution role.

The screenshot shows the AWS Lambda Configuration tab. The 'Configuration' tab is selected. On the left, there are tabs for 'Code', 'Test', 'Monitor', 'Configuration' (which is active), 'Aliases', and 'Versions'. Below these tabs, there are three sections: 'General configuration', 'Triggers', and 'Permissions'. The 'Permissions' section is currently selected. On the right, under the heading 'Execution role', the 'Role name' is listed as 'consumer1-role-bdgrtatc' with a blue edit icon next to it.

18. For permission you need to provide Kinesis full access. Then come back to lambda.

The screenshot shows the AWS IAM Permissions page. The 'Permissions' tab is selected. At the top, there are buttons for 'Trust relationships', 'Tags', 'Last Accessed', and 'Revoke sessions'. Below this, there is a section for 'Permissions policies (2) Info'. It shows two managed policies: 'AmazonKinesisFullAccess' and 'AWSLambdaBasicExecutionRole-2e6981...'. The 'AmazonKinesisFullAccess' policy is highlighted with a blue border. At the bottom of the page, there are buttons for 'Search', 'Filter by Type', 'All types', and navigation controls.

Policy name	Type	Attached entities
AmazonKinesisFullAccess	AWS managed	1
AWSLambdaBasicExecutionRole-2e6981...	Customer managed	1

19. Now for the consumer 1 lambda function we are going to add a trigger. Click on Add trigger.

consumer1

▼ Function overview [Info](#)

[Diagram](#) [Template](#)

consumer1

Layers (0)

[+ Add trigger](#) [+ Add destination](#)

20. Then in the trigger search for Kinesis and choose your data stream, keep everything as it is and click on Add.

Add trigger

Trigger configuration [Info](#)

Kinesis

aws analytics event-source-mapping polling streaming

Kinesis stream

Select a Kinesis stream to listen for updates on. To select a stream in another shared AWS account, enter its Amazon Resource Name (ARN).

X C

Consumer - optional

Select an optional consumer of your stream to listen for updates on. To select a consumer in another shared AWS account, enter its ARN.

C

21. Similarly, you are going to create another lambda function for Consumer 2 and just add the same IAM role to this new lambda function. Also, modify the code by using the same code as before.
22. Just remember to change the consumer value in the code.

Code | Test | Monitor | Configuration | Aliases | Versions

Code source [Info](#)

File Edit Find View Go Tools Window **Test** Deploy

Go to Anything (Ctrl-P) lambda_function Environment Var +

Environment Consumer2 - / lambda_function.py

```
1 import base64
2 import json
3
4 def lambda_handler(event, context):
5     print(json.dumps(event))
6
7     for record in event['Records']:
8         data = json.loads(base64.b64decode(record['kinesis']['data']).decode('utf-8'))
9         print('This is consumer - 2', data)
10
```

Basic information

Function name
Enter a name that describes the purpose of your function.

Function name must be 1 to 64 characters, must be unique to the Region, and can't include spaces. Valid characters are a-z, A-Z, 0-9, hyphens (-), and underscores (_).

Runtime [Info](#)
Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.

Architecture [Info](#)
Choose the instruction set architecture you want for your function code.
 x86_64
 arm64

Permissions [Info](#)

By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.

▼ Change default execution role

Execution role

Choose a role that defines the permissions of your function. To create a custom role, go to the [IAM console](#).

- Create a new role with basic Lambda permissions
- Use an existing role
- Create a new role from AWS policy templates

Existing role

Choose an existing role that you've created to be used with this Lambda function. The role must have permission to upload logs to Amazon CloudWatch Logs.

service-role/consumer1-role-bdgrtac



[View the consumer1-role-bdgrtac role](#) on the IAM console.

► Additional Configurations

Use additional configurations to set up code signing, function URL, tags, and Amazon VPC access for your function.

[Cancel](#)

[Create function](#)

23. Once your function is created then you need to add a trigger for Kinesis like you did before.

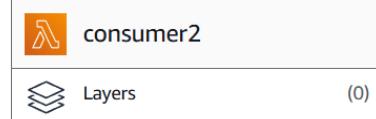
consumer2

The trigger demo-data-stream was successfully added to function consumer2. The trigger is in a disabled state.

▼ Function overview [Info](#)

[Diagram](#)

[Template](#)



Kinesis

[+ Add destination](#)

[+ Add trigger](#)

24. Now we are going to upload an object in our S3 bucket. This object should have a .txt type so that it can trigger our lambda function.

25. Below you can see that we have uploaded a text file in our S3 bucket.

The screenshot shows the AWS S3 console interface. At the top, it says 'Amazon S3 > Buckets > aws-datasource-bucket-2024'. Below that, the bucket name 'aws-datasource-bucket-2024' is displayed with a 'Info' link. A navigation bar with tabs like 'Objects', 'Properties', 'Permissions', 'Metrics', 'Management', and 'Access Points' is visible. Under the 'Objects' tab, there is a table showing one object: 'example.txt' (Type: txt, Last modified: October 4, 2024, 20:19:06 (UTC+05:30), Size: 86.0 B, Storage class: Standard). There are also buttons for 'Copy S3 URI', 'Copy URL', 'Download', 'Open', 'Delete', 'Actions', 'Create folder', and 'Upload'.

26. Now go to Cloud Watch and look for the logs, here you can see that the producer lambda function was triggered along with both of the consumer lambda functions when we uploaded a text file in our S3 bucket.

The screenshot shows the AWS CloudWatch Log Groups interface. At the top, it says 'CloudWatch > Log groups'. Below that, the heading 'Log groups (3)' is shown. A table lists three log groups: '/aws/lambda/Consumer1', '/aws/lambda/Consumer2', and '/aws/lambda/Producer'. Each entry includes columns for Log group, Log class, Anomaly d..., Data pr..., Sensit..., Retention, and Metrics. Buttons for 'Actions', 'View in Logs Insights', 'Start tailing', and 'Create log group' are at the top right.

27. If you go inside your log stream for the consumers and expand the logs you will be able to see the record as shown below.

The screenshot shows the AWS CloudWatch Log Stream interface. It displays two log entries. The first entry is from '2024-10-04T21:29:49.685+05:30' and contains a JSON object with a 'Records' array. The second entry is from '2024-10-04T21:29:49.685+05:30' and contains the text 'This is consumer - 1 {'data': 'This is an Dummy data\nWe are trying to upload\nHope you are enjoying the Lab!!'}'. Both entries have copy and delete icons.

28. Navigate to Kinesis, go to Data viewer and choose your shard ID then the starting position and click on get records and you will see your text file and the data here.

The screenshot shows the AWS Lambda Data viewer interface. At the top, there are tabs: Applications, Monitoring, Configuration, Enhanced fan-out (0), Data viewer (which is selected and highlighted in blue), Data analytics - new, Data stream sharing, and a search bar. Below the tabs, there's a section for 'Shard' with a dropdown menu set to 'shardId-000000000000'. To the right of the shard ID is a 'Starting position' dropdown set to 'Trim horizon' and a 'Get records' button. A 'Records (1)' section follows, containing a message: 'Shard: shardId-000000000000 Starting position: Trim horizon'. Below this is a search bar with placeholder text 'Find records' and navigation controls (back, forward, refresh). A table then displays the single record: 'example.txt' (Partition key) has a 'Data' value of '>{"data": "This is an Dummy data\\nWe are trying ...}' (with the rest of the string ellipsed). The table also includes columns for 'Approximate arrival timestamp' (October 04, 2024 at 21:29:48 GMT+5:30) and 'Sequence number' (4965644358643266124479640141...).

Partition key	Data	Approximate arrival timestamp	Sequence number
example.txt	{"data": "This is an Dummy data\\nWe are trying ..."}	October 04, 2024 at 21:29:48 GMT+5:30	4965644358643266124479640141...

29. Once you are done just delete all the resources one by one.