

## Section 2: Process Data using a Lambda function and send to Kinesis Data Firehose

In this section, we create an Amazon Kinesis Data Firehose delivery stream and populate the delivery stream using a Lambda trigger which acts as a consumer for the Kinesis Data Stream with the historic dataset of taxi trips made in NYC. The Lambda function filters out the spurious data in the incoming events and then sends the clean events to a Kinesis Data Firehose Delivery Stream in batch mode (using the PutRecordBatch API). The Lambda trigger is configured both in standard (polling) and Enhanced FanOut mode to illustrate the differences between the two.

We have provided a Cloudformation template to spin up all the resources. If you prefer to set up the resources using the Console or the aws cli, please scroll down to that section in this guide.

### Create Resources using a Cloudformation template

Follow this link ( <https://console.aws.amazon.com/cloudformation/home#/stacks/new?stackName=NYCTaxiTripsKDFResources&templateURL=https://s3.amazonaws.com/shausma-public/public/cfn-templates/kinesis-analytics-workshop/kinesis-firehose-infrastructure.yml> ) to execute the CloudFormation template.

Click “Next” on the first screen.

In the next screen, copy the Stream arn of the Kinesis Data Stream from Section 1 and paste it in the “KDSStreamArn” box. Accept all other defaults and click “Next”

## Specify stack details

### Stack name

Stack name

NYCTaxiTripsKDFResources

Stack name can include letters (A-Z and a-z), numbers (0-9), and dashes (-).

### Parameters

Parameters are defined in your template and allow you to input custom values when you create or update a stack.

EventSourceMappingType

Standard

ExponentialBackoffSeed

35

GlueDatabaseName

kinesislab

GlueTableName

nyctaxitrips

KDFDeliveryStreamName

nyc-taxi-trips

KDSConsumerName

nyc-taxi-trips-cons-1

KDSSStreamArn

NumberOfLambdaRetries

3

Cancel

Previous

Next

Click “Next” on the next screen.

In the next screen, scroll down and click on the checkbox in the “Capabilities” section and click “Create stack”

### Capabilities

**The following resource(s) require capabilities: [AWS::IAM::Role]**

This template contains Identity and Access Management (IAM) resources that might provide entities access to make changes to your AWS account. Check that you want to create each of these resources and that they have the minimum required permissions. [Learn more.](#)

☒ I acknowledge that AWS CloudFormation might create IAM resources.

Cancel

Previous

Create change set

Create stack

Wait for the stack to get created

Events			
<input type="text" value="Search events"/>			
Timestamp	Logical ID	Status	Status reason
06 Jun 2019 20:06:40	NYCTaxiTripsKDFResources	CREATE_IN_PROGRESS	User Initiated

Logical ID	Status
NYCTaxiTripsKDFResources	CREATE_COMPLETE

Once the stack completes creation, look into the resources in the outputs tab, and go to the respective consoles and look at the resources created. Go to the [“When all resources are created”](#) section in this guide.

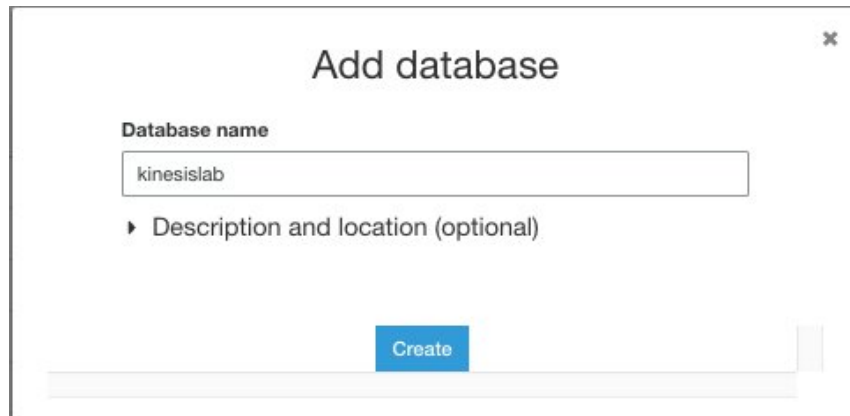
## Create Resources using the Console and AWS CLI

Before we create the Kinesis Data Firehose delivery stream, we need to create some prerequisites that the delivery stream references.

### Create the Glue database:

Go to the Glue console, click on "Databases" in the left pane and then click on "Add database". Type in a name for the database and click on "Create". This database will be used later to create an external table using the Athena console to provide a schema for data format conversion when creating a delivery stream using the Kinesis Data Firehose console.





**Add database**

Database name

kinesislabs

▸ Description and location (optional)

Create

AWS CLI: `aws glue create-database --database-input '{"Name": "kinesislabs"}'`

## Create the S3 bucket

Go to the S3 console, click on "Create bucket". This bucket will be used to store the data delivered to S3 via the Kinesis Data Firehose delivery stream.

S3 buckets



Search for buckets

+ Create bucket Edit public access settings Empty Delete

Type in the Bucket name, specify the Region and click on "Create".

Create bucket

1 Name and region 2 Configure options 3 Set permissions 4 Review

Name and region

Bucket name ⓘ

kinesislabs-datalake-bucket

Region

US East (N. Virginia) ▼

Copy settings from an existing bucket

Select bucket (optional) 17 Buckets ▼

Create Cancel Next

AWS CLI: `aws s3 mb s3://kinesislabs-datalake-bucket --region us-east-1`

### Create the external table that Kinesis Data Firehose will use as a schema for data format conversion

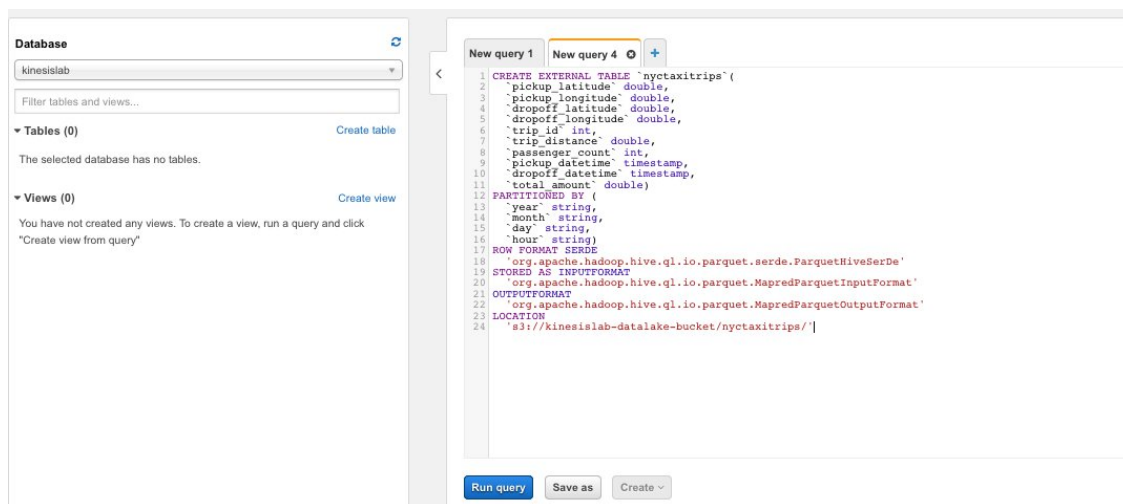
Go to the Athena console, in the left pane, under "Database", click on the dropdown and select the database that you created earlier through the Glue console. Paste the following sql statement in the query window and click on "Run query". Remember to replace the S3 bucket name with the name of the bucket you created earlier through the S3 console.

```
CREATE EXTERNAL TABLE `nyctaxitrips`(  
  `pickup_latitude` double,  
  `pickup_longitude` double,  
  `dropoff_latitude` double,  
  `dropoff_longitude` double,  
  `trip_id` int,  
  `trip_distance` double,  
  `passenger_count` int,  
  `pickup_datetime` timestamp,
```

```

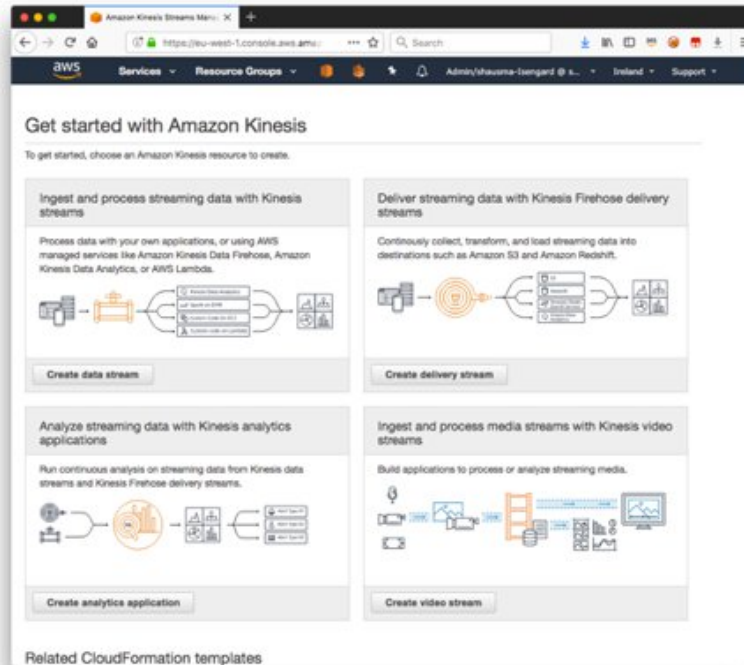
    `dropoff_datetime` timestamp,
    `total_amount` double)
PARTITIONED BY (
    `year` string,
    `month` string,
    `day` string,
    `hour` string)
ROW FORMAT SERDE
    'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
STORED AS INPUTFORMAT
    'org.apache.hadoop.hive.ql.io.parquet.MapredParquetInputFormat'
OUTPUTFORMAT
    'org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat'
LOCATION
    's3://kinesislabs-datalake-bucket/nyctaxitrips/'

```



## Create Kinesis Data Firehose delivery Stream

Navigate to the Amazon Kinesis services and press **Get Started** when prompted (you may not need to complete this, if you have already used Amazon Kinesis). Select **Create delivery stream** to navigate to the Amazon Kinesis Data Firehose service.



Enter a unique name for the Delivery stream name, eg, *nyc-taxi-trips*. For “Source”, choose “Direct PUT or other sources” as a lambda function would be used to feed events to the Kinesis Data Firehose delivery stream. Click **Next**.

**New delivery stream**

Delivery streams load data, automatically and continuously, to the destinations that you specify. Kinesis Firehose resources are not covered under the [AWS Free Tier](#), and **usage-based charges apply**. For more information, see [Kinesis Firehose pricing](#).

**Delivery stream name\***

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

**Choose source**

Choose how you would prefer to send records to the delivery stream.

**Firehose data flow overview**

**Source\*** ☒ **Direct PUT or other sources**  
Choose this option to send records directly to the delivery stream, or to send records from AWS IoT, CloudWatch Logs, or CloudWatch Events.

☐ Kinesis stream

Choose “Record Transformation” “as “Disabled” and “Record format conversion” as “Enabled” and choose “Output format” as “Apache Parquet”.

## Kinesis Firehose - Create delivery stream

### Step 1: Name and source

### Step 2: Process records

### Step 3: Choose destination

### Step 4: Configure settings

### Step 5: Review

### Process records

Kinesis Firehose can transform records or convert record format before delivery.



### Transform source records with AWS Lambda

To return records from AWS Lambda to Kinesis Firehose after transformation, the Lambda function you invoke must be compliant with the required record transformation output model. [Learn more](#)

Record transformation ☒ Disabled ☐ Enabled

### Convert record format

Data in Apache parquet or Apache ORC format is typically more efficient to query than JSON. Kinesis Data Firehose can convert your JSON-formatted source records using a schema from a table defined in AWS Glue. For records that aren't in JSON format, create a Lambda function that converts them to JSON in the [Transform source records with AWS Lambda](#) section above. [Learn more](#)

Record format conversion ☒ Disabled ☐ Enabled

If record format conversion is enabled, Firehose can deliver data to Amazon S3 using Record Format Conversion (RFC) configured using the OpenText JDBC-to-S3 tool. For other options, see the [RFC CLI](#).

Output format ☒ Apache Parquet ☐ Apache ORC

This AWS is implemented using Snappy compression before it is delivered to S3. To choose another compression method, or to disable data compression, use the [AWS CLI](#). [Learn more](#)

Specify a schema for source records. Kinesis Data Firehose references table definitions stored in AWS Glue. Choose an AWS Glue table to specify a schema for your source records. You can manually create a new table in AWS Glue ([?](#)), or add a crawler in AWS Glue ([?](#)) to create a new table using a schema from an existing JSON object in S3. [Learn more](#)

Choose the schema to use for data format conversion as illustrated in the screenshot below.

Specify a schema for source records. Kinesis Data Firehose references table definitions stored in AWS Glue. Choose an AWS Glue table to specify a schema for your source records. You can manually create a new table in AWS Glue ([?](#)), or add a crawler in AWS Glue ([?](#)) to create a new table using a schema from an existing JSON object in S3. [Learn more](#)

AWS Glue region\* US East (N. Virginia) ▼

AWS Glue database\* kinesislabs ▼ [View kinesislabs in AWS Glue](#)

AWS Glue table\* nyctaxitrips ▼ [View nyctaxitrips in AWS Glue](#)

AWS Glue table version\* Latest ▼

\* Required

Cancel

Previous

Next

Select the S3 destination. I chose "S3".



## Select destination



**i** Amazon S3 is the only available destination because you enabled **Record format conversion** in **Step 2: Process records**.

### Destination\*

☒ Amazon S3

Amazon S3 is an easy-to-use object storage, with a simple web service interface to store and retrieve any amount of data from anywhere on the web.

☐ Amazon Redshift

Amazon Redshift is a fast, fully managed, petabyte-scale data warehouse that makes it simple and cost effective to analyze all your data using your existing business intelligence tools.

☐ Amazon Elasticsearch Service

Elasticsearch is an open-source search and analytics engine for use cases such as log analytics, real-time application monitoring, and click stream analytics.

☐ Splunk

Splunk is an operational intelligence tool for analyzing machine-generated data in real-time.

Click on the dropdown next to "S3 bucket" and select the bucket that you created earlier through the S3 console. For "S3 prefix", copy and paste the following:

```
nyctaxitrips/year={!timestamp:YYYY}/month={!timestamp:MM}/day={!timestamp:dd}/hour={!timestamp:HH}/
```

For "S3 error prefix" copy and paste the following:

```
nyctaxitripserror/{firehose:error-output-type}/year={!timestamp:YYYY}/month={!timestamp:MM}/day={!timestamp:dd}/hour={!timestamp:HH}/
```

For "Source record S3 backup", click on "Disabled". Click on "Next".

S3 bucket\*

kinesislab-datalake-bucket

Create new

[View kinesislab-datalake-bucket in S3 console](#)

### S3 prefix

By default, Kinesis Data Firehose appends the prefix "YYYY/MM/DD/HH" (in UTC) to the data it delivers to Amazon S3. You can override this default by specifying a custom prefix that includes expressions that are evaluated at runtime.

If your custom prefix doesn't include expressions, Kinesis Data Firehose uses your prefix and appends "YYYY/MM/DD/HH". If your custom prefix includes a Firehose random string or timestamp expression, Kinesis Data Firehose doesn't append "YYYY/MM/DD/HH".[Learn more](#)

Prefix

nyctaxitrips/year={!timestamp:YYYY}/month={!timestamp:MM}/day={!timestamp:dd}/hour={!timestamp:HH}/

### S3 error prefix

You can specify an S3 bucket prefix to be used in error conditions. This prefix can include expressions for Kinesis Data Firehose to evaluate at runtime. [Learn more about the rules for specifying prefix expressions](#)

Error prefix

nyctaxitriperror/{!firehose:error-output-type}/year={!timestamp:YYYY}/month={!timestamp:MM}/day={!timestamp:dd}/hour={!timestamp:HH}/  
|

### S3 backup

Enabling source record backup ensures that source records can be recovered if record processing transformation does not produce the desired results. [Learn more](#)

Source record S3 backup\*

☒ Disabled

☐ Enabled

\* Required

Cancel

Previous

Next

Specify the **buffering hints** for the Amazon S3 destination. Type in 128 MB and 300 seconds.

### S3 buffer conditions

Firehose buffers incoming records before delivering them to your S3 bucket. Record delivery will be triggered once either of these conditions has been satisfied. [Learn more](#)

**i** Because of the high compression ratios that you typically get when converting JSON records to Parquet or ORC format, we recommend a buffer size value of 128 MB. Specifying a smaller buffer size can result in the delivery of very small S3 objects, which are less efficient to query.

Buffer size\*  MB

Specify a buffer size between 64-128 MB

Buffer interval\*  seconds

Specify a buffer interval between 60-900 seconds

Keep the default settings for "S3 compression and encryption".

### S3 compression and encryption

Firehose can compress records before delivering them to your S3 bucket. Compressed records can also be encrypted in the S3 bucket using a KMS master key. [Learn more](#)

**i** You don't need to enable compression. Snappy compression was enabled automatically when you chose **Record format conversion** in **Step 2: Process records**. To choose another compression method, or to disable data compression, use the AWS CLI. [Learn more](#)

S3 compression\* ☒ Disabled

☐ GZIP

☐ Snappy

☐ Zip

S3 encryption\* ☒ Disabled

☐ Enabled

Choose "Enabled" for "Error logging".

### Error logging

Firehose can log record delivery errors to CloudWatch Logs. If enabled, a CloudWatch log group and corresponding log streams are created on your behalf. [Learn more](#)

Error logging\* ☐ Disabled  
☒ Enabled

### Tags (optional)

You can add tags to organize your AWS resources, track costs, and control access. [Learn more](#)

Key	Value - optional	
<input type="text" value="Enter key"/>	<input type="text" value="Enter value"/>	<input type="button" value="Remove tag"/>
<input type="button" value="Add tag"/>		

You can add 49 more tag(s)

Specify the IAM role that you want Kinesis Data Firehose to assume to access resources on your behalf. Choose either **Create new** or **Choose** to display a new screen. Choose **Create a new IAM role**, name the role, and then choose **Allow**.

### IAM role

Firehose uses an IAM role to access your specified resources, such as the S3 bucket and KMS key. [Learn more](#)

IAM role\*

### IAM role

Firehose uses an IAM role to access your specified resources, such as the S3 bucket and KMS key. [Learn more](#)

IAM role\*

Choose **Create Delivery Stream**.

**Role Summary** ⓘ

**Role** Provides access to AWS Services and Resources

**Description**

**IAM Role**

**Policy Name**

[View Policy Document](#)

[Cancel](#) [Allow](#)

Copy and paste the json below into a file and use it to create the delivery stream using the AWS CLI.

AWS CLI:

- `aws firehose create-delivery-stream --cli-input-json file://createdeliverystream.json .`

### **createdeliverystream.json**

```
{
  "DeliveryStreamName": "nyc-taxi-trips",
  "DeliveryStreamType": "DirectPut",
  "ExtendedS3DestinationConfiguration": {
    "RoleARN": "arn:aws:iam::<account-id>:role/firehose_delivery_role",
    "BucketARN": "arn:aws:s3:::kinesislabs-datalake-bucket",
    "Prefix": "nyctaxitrips/year=!{timestamp:YYYY}/month=!{timestamp:MM}/day=!{timestamp:dd}/hour=!{timestamp:HH}/",
    "ErrorOutputPrefix": "nyctaxitriperror/!{firehose:error-output-type}/year=!{timestamp:YYYY}/month=!{timestamp:MM}/day=!{timestamp:dd}/hour=!{timestamp:HH}/",
    "BufferingHints": {
      "SizeInMBs": 128,
      "IntervalInSeconds": 300
    },
    "CompressionFormat": "UNCOMPRESSED",
    "EncryptionConfiguration": {
      "NoEncryptionConfig": "NoEncryption"
    },
    "CloudWatchLoggingOptions": {
      "Enabled": true,

```





```

        "LogGroupName": "KDF-NYCTaxiTrips",
        "LogStreamName": "S3Delivery"
    },
    "S3BackupMode": "Disabled",
    "DataFormatConversionConfiguration": {
        "SchemaConfiguration": {
            "RoleARN": "arn:aws:iam::<account-id>:role/firehose_delivery_role",
            "DatabaseName": "kinesislabs",
            "TableName": "nyctaxitrips",
            "Region": "us-east-1",
            "VersionId": "LATEST"
        },
        "InputFormatConfiguration": {
            "Deserializer": {
                "OpenXJsonSerDe": {}
            }
        },
        "OutputFormatConfiguration": {
            "Serializer": {
                "ParquetSerDe": {}
            }
        }
    },
    "Enabled": true
}
}
}

```

## Create the IAM role to use with the Lambda function

Go to the IAM console, click on "Roles" and then click "Create role". Click on "AWS service" and then click on "Lambda". Change the name of the Kinesis Data Stream if you created a stream named different than "nyc-taxi-trips".

 <b>AWS service</b> EC2, Lambda and others	 <b>Another AWS account</b> Belonging to you or 3rd party	 <b>Web identity</b> Cognito or any OpenID provider	 <b>SAML 2.0 federation</b> Your corporate directory
--	---	---	--

Allows AWS services to perform actions on your behalf. [Learn more](#)

Choose the service that will use this role

**EC2**

Allows EC2 instances to call AWS services on your behalf.

**Lambda**

Allows Lambda functions to call AWS services on your behalf.

Click on "Next:Permissions"


In the search box next to "Filter policies", type "AWSLambdaBasicExecutionRole" and select the checkbox next to "AWSLambdaBasicExecutionRole". Click on "Next:Tags" and add tags if you wish. Click "Next:Review"

▼ Attach permissions policies

Choose one or more policies to attach to your new role.

Create policy ↺

Filter policies ▼  Showing 2 results

	Policy name ▼	Used as	Description
<input checked="" type="checkbox"/>	 AWSLambdaBasicExecutionRole	Permissions policy (1)	Provides write permissions to CloudWat...

Fill in the details as show below and click on "Create role".

Create role 1 2 3 4


Review

Provide the required information below and review this role before you create it.

**Role name\***   
Use alphanumeric and '+-=,@-\_' characters. Maximum 64 characters.

**Role description**   
Maximum 1000 characters. Use alphanumeric and '+-=,@-\_' characters.

**Trusted entities** AWS service: lambda.amazonaws.com

**Policies**  AWSLambdaBasicExecutionRole [↗](#)

**Permissions boundary** Permissions boundary is not set

*No tags were added.*

\* Required Cancel Previous Create role

Click on the newly create role. Then click on "Add inline policy". Click on the "JSON" tab. Then copy and paste the json from the KinesisPolicy.json file below (remember

to change the account-id to your account id and the stream name and delivery stream name to your streams). Then click on "Review policy".

Fill in the details as shown below and click on "Create policy".

Review policy

Before you create this policy, provide the required information and review this policy.

Name\*

Maximum 128 characters. Use alphanumeric and '+', '=', '@', '-', '\_' characters.

Summary

This policy defines some actions, resources, or conditions that do not provide permissions. To grant access, policies must have an action that has an applicable resource or condition. For details, choose **Show remaining**. [Learn more](#)

Service	Access level	Resource	Request condition
Allow (2 of 176 services) <a href="#">Show remaining 174</a>			
<a href="#">Firehose</a>	Limited: List, Write	Multiple	None
<a href="#">Kinesis</a>	Limited: List, Read	Multiple	None

\* Required

[Cancel](#) [Previous](#) [Create policy](#)

The Lambda role is now created.

AWS CLI:

### TrustPolicyForLambda.json

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Principal": {
      "Service": "lambda.amazonaws.com"
    },
    "Action": "sts:AssumeRole"
  }]
}
```

- aws iam create-role --role-name NYCTaxiTripsLambdaRole --assume-role-policy-document file://TrustPolicyForLambda.json



- `aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole --role-name NYCTaxiTripsLambdaRole`
- `aws iam put-role-policy --role-name NYCTaxiTripsLambdaRole --policy-name NYCTaxiTripsKinesisPolicy --policy-document file://KinesisPolicy.json`

### KinesisPolicy.json:

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Sid": "Effect: ",
    "Effect": "Allow",
    "Action": [
      "kinesis:GetShardIterator",
      "kinesis:GetRecords",
      "firehose:PutRecordBatch",
      "kinesis:DescribeStream"
    ],
    "Resource": [
      "arn:aws:kinesis:us-east-1:<accountid>:stream/nyc-taxi-trips",
      "arn:aws:firehose:us-east-1:<accountid>:deliverystream/nyc-taxi-trips"
    ]
  },
  {
    "Sid": "KinesisPerm2",
    "Effect": "Allow",
    "Action": [
      "kinesis:ListStreams",
      "kinesis:SubscribeToShard",
      "kinesis:DescribeStreamSummary",
      "firehose:ListDeliveryStreams"
    ],
    "Resource": "*"
  },
  {
    "Sid": "KinesisPerm3",
    "Effect": "Allow",
    "Action": "kinesis:ListShards",
    "Resource": [
      "arn:aws:kinesis:us-east-1:<accountid>:stream/nyc-taxi-trips"
    ]
  }
]}
```

```
]
}
```

## Create the Lambda function to process records from the Kinesis Data Stream

The Lambda function does a few things:

1. It inspects the incoming message for unclean records with missing fields and filters them out.
2. It tries to send the clean records to Kinesis Data Firehose.
3. If it receives a throttling error, it determines if all the records received failed or some records failed.
  - a. If all records failed, it raises an exception, so the Lambda service can retry with the same payload (the service keeps retrying with the same payload until it receives a success). It also logs the corresponding trip ids so you can check to see if they actually made it through to the S3 bucket.
  - b. If some records failed, it retries based on the configured retries environment variable with exponential backoff. If any of the retries are successful, it returns a success. If none of the retries are successful, it raises an exception so the Lambda service can retry with the same payload. Note that in this case, there could be duplicate records sent. You can increase the number of retries or save the records somewhere to process later and move on as alternate strategies to prevent duplicates.

Go to the Lambda console and click on "Create function". Select "Author from scratch" and fill in the details as show below and click on "Create function".

The screenshot shows the 'Basic information' tab of the AWS Lambda 'Create function' wizard. The form includes the following sections:

- Function name:** A text input field containing 'NYCTaxiTripsProcess'. Below the field is a note: 'Use only letters, numbers, hyphens, or underscores with no spaces.'
- Runtime:** A dropdown menu showing 'Python 3.7'.
- Permissions:** A section with a link to 'Info' and a note: 'Lambda will create an execution role with permission to upload logs to Amazon CloudWatch Logs. You can configure and modify permissions further when you add triggers.' Below this is a link: 'Choose or create an execution role'.
- Execution role:** A dropdown menu showing 'Use an existing role'.
- Existing role:** A dropdown menu showing 'NYCTaxiTripsLambdaRole'. Below the dropdown is a link: 'View the NYCTaxiTripsLambdaRole role on the IAM console.' To the right of the dropdown is a refresh icon.

At the bottom right of the form are two buttons: 'Cancel' and 'Create function'.

Copy and Paste the following code in the "Function code" window overwriting the template code. Then click "Save".

```
import base64
from datetime import datetime
import time
import json
import boto3
import random
import uuid
import os

client = boto3.client('firehose')

def check_data(data):
    payload = json.loads(data)

    if payload['type'] == "trip" and
payload['pickup_longitude'] != 0 and
payload['pickup_latitude'] != 0 and
payload['dropoff_latitude'] != 0 and
payload['dropoff_longitude'] != 0:
        return True
    else:
        return False

def gen_retry_output_list(resp, outputRecList):
    recCount = 0
    retryOutputList = []
    for respRec in resp['RequestResponses']:
        try:
            respError = respRec['ErrorCode']
            if respError == "ServiceUnavailableException":
                retryOutputList.append(outputRecList[recCount])
        except KeyError:
            pass
        recCount += 1

    return retryOutputList

def get_sleep_time(retryCount, exponentialBackoff, seed):

    if (exponentialBackoff == True):
        return (2*(seed**retryCount))/1000
```

```

        else:
            return (500/1000)

def lambda_handler(event, context):
    print('Loading function' + ' ' +
+ datetime.now().strftime('%Y-%m-%dT%H:%M:%S.%fZ'))
    print('Processing {}
record(s) {}'.format(len(event['Records'])))
    output = {}
    outputRecList = []
    retryOutputList = []
    numRetries = int(os.environ['number_of_retries'])
    retryCount = 1
    eventRecords = len(event['Records'])
    tripIds = []
    deliveryStreamName = os.environ['delivery_stream_name']
    seed = int(os.environ['exponential_backoff_seed'])
    exponentialBackoff = False

    for record in event['Records']:
        recordData =
base64.b64decode(record['kinesis']['data'])
        recordDataJson = json.loads(recordData)
        if check_data(recordData):
            output['Data'] = recordData
            outputRecList.append(output)
            output = {}
            tripIds.append(recordDataJson['trip_id'])

    if len(outputRecList) > 0:
        resp =
client.put_record_batch(DeliveryStreamName=deliveryStreamNa
me,
                        Records=outputRecList
                        )
    else:
        print("No records to send ...")
        return {
            'statusCode': 200,
            'body': json.dumps('Lambda successful!')
        }

    if resp['FailedPutCount'] != 0:
        print('Failed to process {}
record(s) {}'.format(resp['FailedPutCount']))

```

```

        if resp['FailedPutCount'] != eventRecords:
            while (retryCount <= numRetries):
                print("Retrying {} failed records up to {}
times with exponential
backoff...".format(resp['FailedPutCount'], numRetries -
(retryCount - 1)))
                retryOutputList =
gen_retry_output_list(resp, outputRecList)
                if len(retryOutputList) > 0:
                    exponentialBackoff = True
                    print("Backing Off for {} seconds
...".format(get_sleep_time(retryCount, exponentialBackoff,
seed)))
                    time.sleep(get_sleep_time(retryCount,
exponentialBackoff, seed))
                retryResp =
client.put_record_batch(DeliveryStreamName=deliveryStreamNa
me,
                        Records=retryOutputList
                        )
                if retryResp['FailedPutCount'] == 0:
                    print("Retry successful after {} tries
...".format(retryCount))
                    return {
                        'statusCode': 200,
                        'body': json.dumps('Lambda
successful!')
                    }
                retryCount += 1
                outputRecList = retryOutputList
                retryOutputList = []
                resp = retryResp
                print(resp['RequestResponses'])

            print("All retries unsuccessful. Letting Lambda
retry but there could be duplicates ...")
            raise Exception("Records could not be sent.
Lambda to retry ...")
        else:
            print("Since all records failed, letting Lambda

```

```

retry..)
    print(tripIds)
    print(resp['RequestResponses'])
    raise Exception("Records could not be sent.
Lambda to retry ...")
else:
    print("Records successfully sent ...")
    return {
        'statusCode': 200,
        'body': json.dumps('Lambda successful!')
    }

```

Scroll down to the “Environment variables” section and fill it out as show below:

**Environment variables**

You can define environment variables as key-value pairs that are accessible from your function code. These are useful to store configuration settings without the need to change function code. [Learn more](#)

delivery_stream_name	nyc-taxi-trips	Remove
exponential_backoff_seed	35	Remove
number_of_retries	3	Remove
Key	Value	Remove

Go to the Triggers pane in the top left of the page, scroll down and click on "Kinesis".

**▼ Designer**

CloudFront

CloudWatch Events

CloudWatch Logs

CodeCommit

Cognito Sync Trigger

DynamoDB

Kinesis

S3

SNS

SQS

NYCTaxiTripsProcess  
Saved  
Layers (0)

Kinesis  
Configuration required

Add triggers from the list on the left

Amazon CloudWatch Logs

Amazon Kinesis

Amazon Kinesis Firehose

Resources that the function's role has access to appear here

Scroll down to the "Configure triggers" section and fill in the details as shown below. First we will setup the trigger in "Standard" mode which is with "No consumer". Click "Add". Then scroll up and click "Save".

### Configure triggers

**Kinesis stream**  
Select a Kinesis stream to listen for updates on.

nyc-taxi-trips

**Consumer**  
Select an optional [consumer](#) of your stream to listen for updates on.

No consumer

**Batch size**  
The largest number of records that will be read from your stream at once.

100

**Starting position**  
The position in the stream to start reading from. For more information, see [ShardIteratorType](#) in the Amazon Kinesis API Reference.

Latest

In order to read from the Kinesis trigger, your execution role must have proper permissions.

☒ **Enable trigger**  
Enable the trigger now, or create it in a disabled state for testing (recommended).

Cancel
Add

In order to setup the Lambda trigger in "Enhanced FanOut (EFO)" mode, first register a consumer using the AWS CLI. This cannot be done via the console.

AWS CLI:

- `aws kinesis register-stream-consumer --stream-arn <your-kinesis-stream-arn> --consumer-name nyc-taxi-trips-cons-1`

Then come back to the Lambda console, go to the Triggers pane in the top left of the page, scroll down and click on "Kinesis" as before. Scroll down to the "Configure triggers" section and fill in the details as shown below. In this case, click on the "Consumer" dropdown and select the consumer you just created in the AWS CLI. Click "Add". Then scroll up and click "Save".

### Configure triggers

**Kinesis stream**  
Select a Kinesis stream to listen for updates on.

nyc-taxi-trips

**Consumer**  
Select an optional [consumer](#) of your stream to listen for updates on.

nyc-taxi-trips-cons-1

**Batch size**  
The largest number of records that will be read from your stream at once.

100

**Starting position**  
The position in the stream to start reading from. For more information, see [ShardIteratorType](#) in the Amazon Kinesis API Reference.

Latest

In order to read from the Kinesis trigger, your execution role must have proper permissions.

☒ **Enable trigger**  
Enable the trigger now, or create it in a disabled state for testing (recommended).

Cancel
Add

AWS CLI:

- Save the Lambda function code in a “lambda\_function.py” file and zip it up into a lambda\_function.zip file using: `zip lambda_function.zip lambda_function.py`
- `aws lambda create-function --function-name NYCTaxiTrips --runtime python3.7 --role arn:aws:iam::<account-id>:role/NYCTaxiTripsLambdaRole --handler lambda_function.lambda_handler --zip-file "fileb://lambda_function.zip"`

### **For standard event source mapping:**

AWS CLI:

- `aws lambda create-event-source-mapping --event-source-arn <stream-arn> --function-name NYCTaxiTripsProcess --starting-position LATEST`

To get the stream-arn use:

- `aws kinesis describe-stream --stream-name nyc-taxi-trips`

### **For EFO event source mapping:**

AWS CLI:

- `aws lambda create-event-source-mapping --event-source-arn <consumer-arn> --function-name NYCTaxiTripsProcess --starting-position LATEST`

To get the consumer-arn use:

- `aws kinesis list-stream-consumers --stream-arn <stream-arn for the nyc-taxi-trips stream>`

The Lambda function that will process the records from the Kinesis Data stream is now created and the event source mapping is also created.

### **When all resources are created**

Start sending data to the Kinesis Data Streams stream created in Section 1 as outlined in Section 1.

Once the Lambda function starts processing (note that it will process from the tip of the stream as the starting position is set to LATEST), the Kinesis Data Firehose delivery stream you created will ingest the records, buffer it, transform it to parquet and deliver it to the S3 destination under the prefix provided. Go to the S3 console and navigate to the bucket and prefix and locate the files.

You can also navigate to Amazon Cloudwatch Logs to look at the output of the Kinesis Data Streams trigger Lambda function. To do that, go to the Lambda console, click on the function you created, click on the “Monitoring” tab, then click on “View logs in CloudWatch”. This should take you to the Cloudwatch Logs console and you



can see the log streams. Click on any one of them and scroll down to observe the lambda function execution outputs. In particular, look for the instances of throttling errors received from Kinesis Data Firehose and how the function handles them.