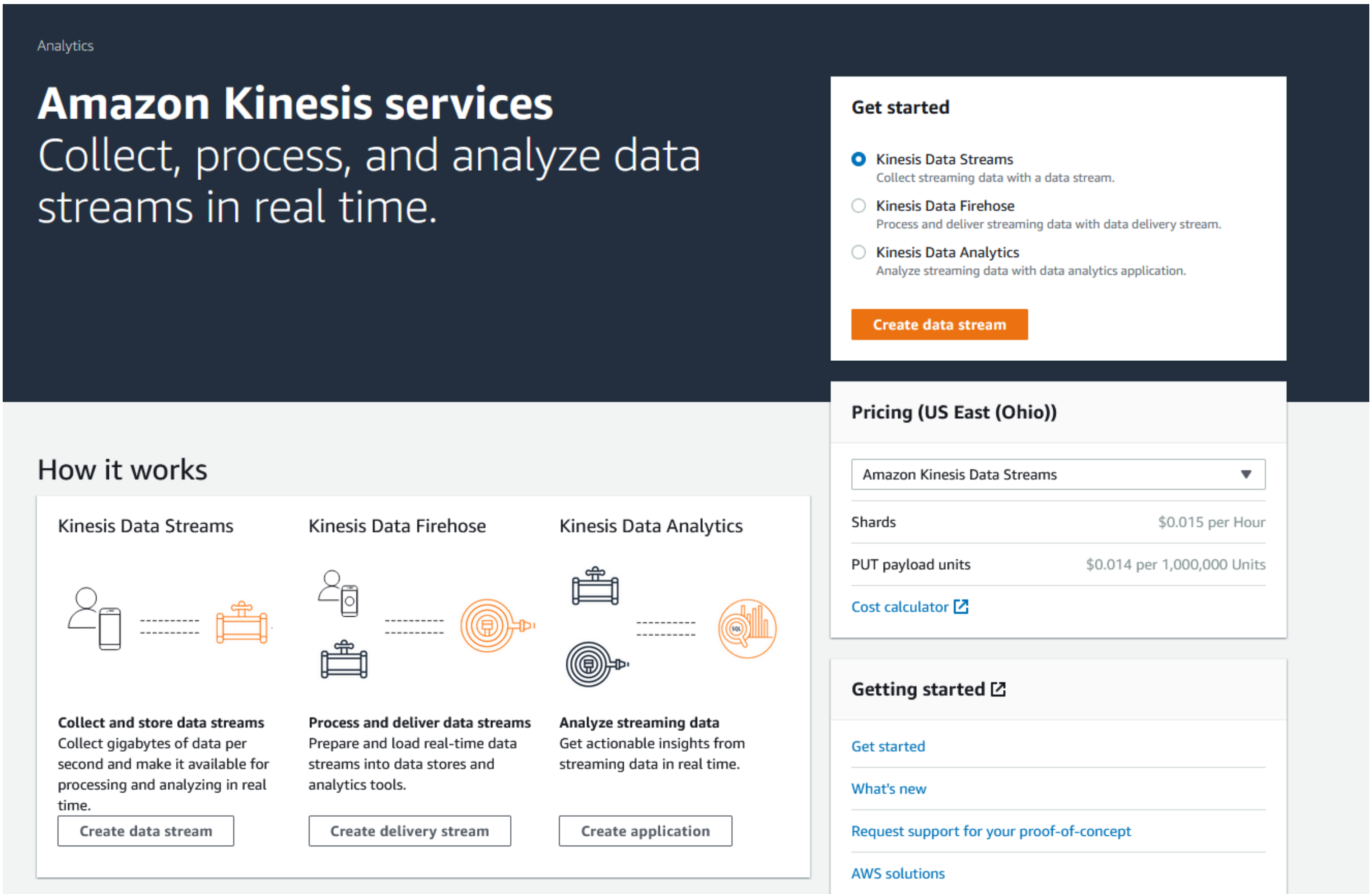
1. Kinesis Firehose Delivery Stream

In the AWS Console -> Kinesis



Analytics

Amazon Kinesis services
Collect, process, and analyze data streams in real time.

Get started

● Kinesis Data Streams
Collect streaming data with a data stream.

○ Kinesis Data Firehose
Process and deliver streaming data with data delivery stream.

○ Kinesis Data Analytics
Analyze streaming data with data analytics application.

Create data stream

Pricing (US East (Ohio))

Amazon Kinesis Data Streams ▼

Shards	$0.015 per Hour
PUT payload units	$0.014 per 1,000,000 Units

Cost calculator

Getting started

Get started

What's new

Request support for your proof-of-concept

AWS solutions

How it works

Kinesis Data Streams

Collect and store data streams
Collect gigabytes of data per second and make it available for processing and analyzing in real time.

Create data stream

Kinesis Data Firehose

Process and deliver data streams
Prepare and load real-time data streams into data stores and analytics tools.

Create delivery stream

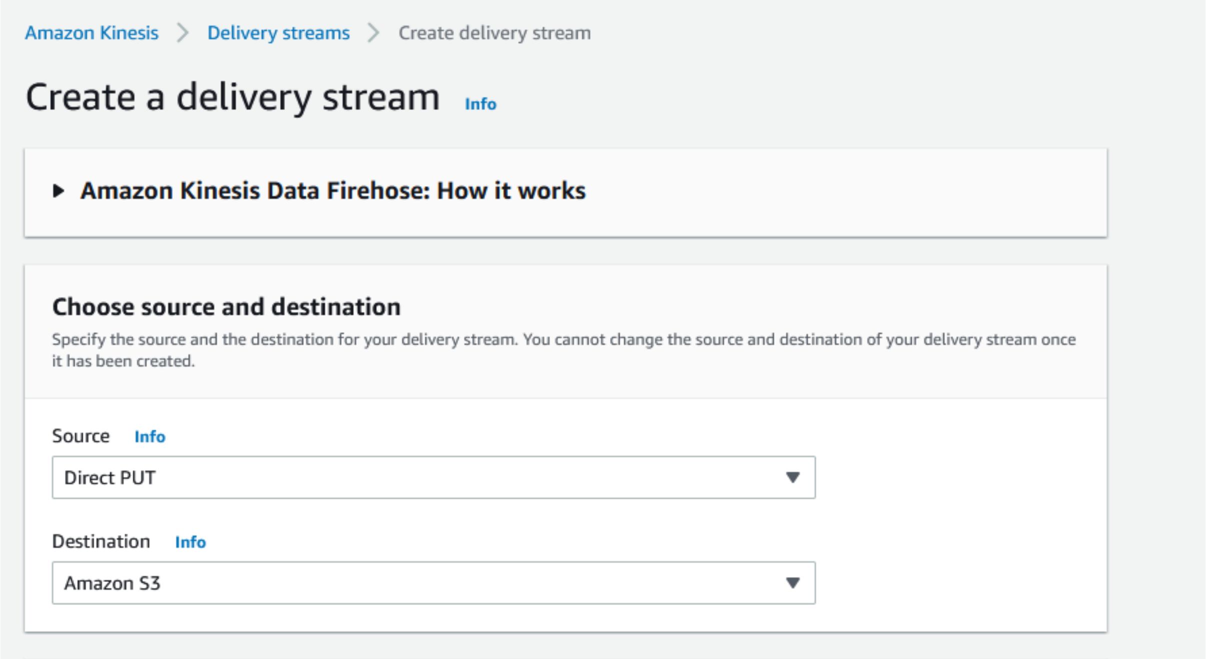
Kinesis Data Analytics

Analyze streaming data
Get actionable insights from streaming data in real time.

Create application

Click “Create Delivery Stream”

Select source as –



Amazon Kinesis > Delivery streams > Create delivery stream

Create a delivery stream Info

▶ Amazon Kinesis Data Firehose: How it works

Choose source and destination

Specify the source and the destination for your delivery stream. You cannot change the source and destination of your delivery stream once it has been created.

Source Info

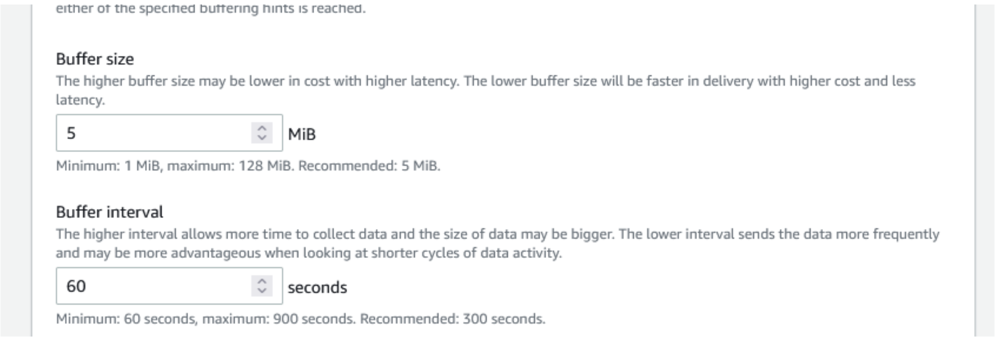
Direct PUT ▼

Destination Info

Amazon S3 ▼

Selecting Destination as S3, create a new bucket. The bucket should have a unique name.

Change the buffer size and buffer interval as needed.



either of the specified buffering hints is reached.

Buffer size

The higher buffer size may be lower in cost with higher latency. The lower buffer size will be faster in delivery with higher cost and less latency.

5 MiB

Minimum: 1 MiB, maximum: 128 MiB. Recommended: 5 MiB.

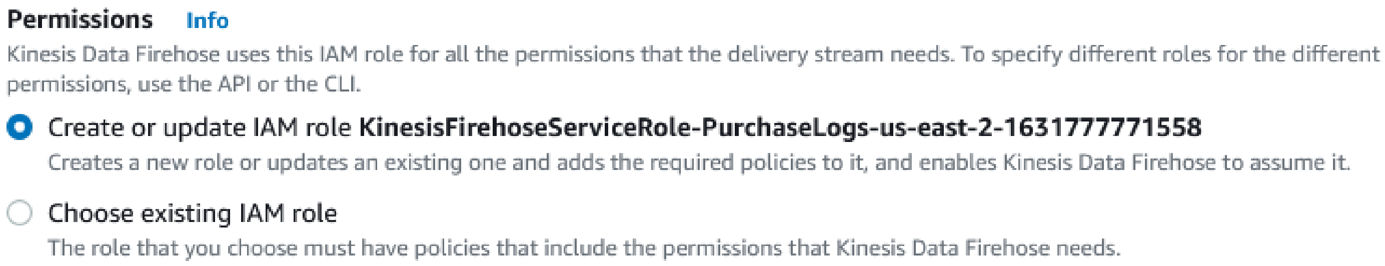
Buffer interval

The higher interval allows more time to collect data and the size of data may be bigger. The lower interval sends the data more frequently and may be more advantageous when looking at shorter cycles of data activity.

60 seconds

Minimum: 60 seconds, maximum: 900 seconds. Recommended: 300 seconds.

Create a new IAM role –



Permissions Info

Kinesis Data Firehose uses this IAM role for all the permissions that the delivery stream needs. To specify different roles for the different permissions, use the API or the CLI.

● Create or update IAM role **KinesisFirehoseServiceRole-PurchaseLogs-us-east-2-1631777771558**

Creates a new role or updates an existing one and adds the required policies to it, and enables Kinesis Data Firehose to assume it.

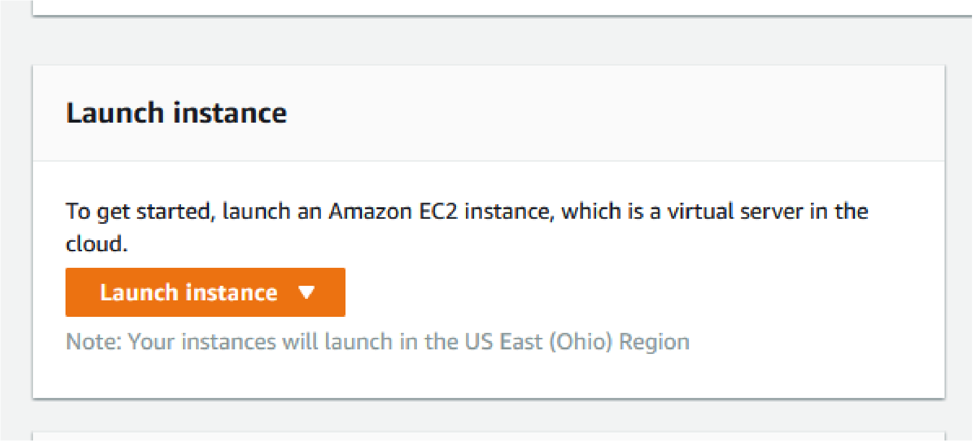
○ Choose existing IAM role

The role that you choose must have policies that include the permissions that Kinesis Data Firehose needs.

2. Now that the delivery stream is set up, we need to set up an EC2 instance to feed data into the stream.

Click on the EC2 service.

Then launch instance.

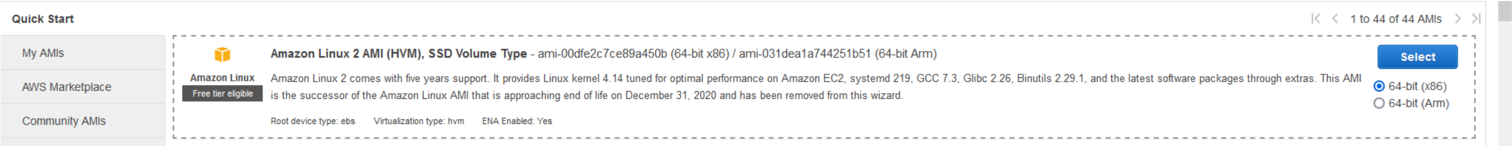


Launch instance

To get started, launch an Amazon EC2 instance, which is a virtual server in the cloud.

Note: Your instances will launch in the US East (Ohio) Region

Select the Amazon Linux 2 AMI



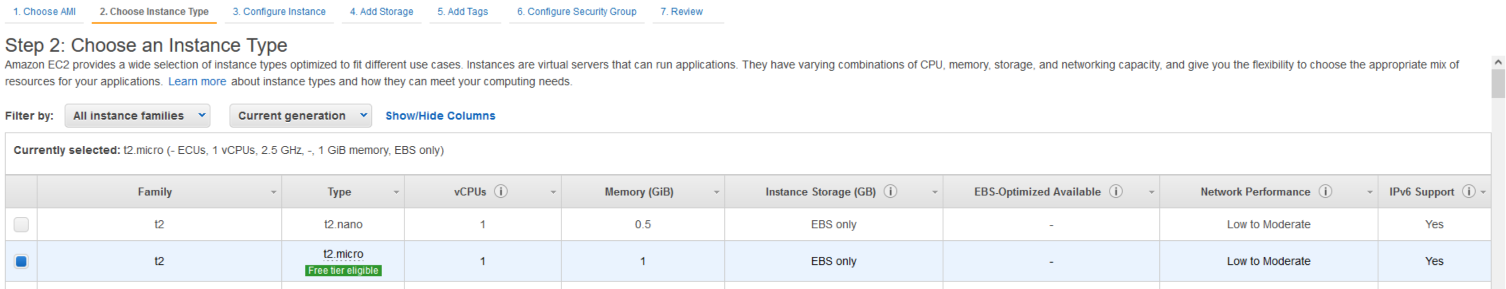
Amazon Linux 2 AMI (HVM), SSD Volume Type - ami-00dfe2c7ce89a450b (64-bit x86) / ami-031dea1a744251b51 (64-bit Arm)

Amazon Linux 2 comes with five years support. It provides Linux kernel 4.14 tuned for optimal performance on Amazon EC2, systemd 219, GCC 7.3, Glibc 2.26, Binutils 2.29.1, and the latest software packages through extras. This AMI is the successor of the Amazon Linux AMI that is approaching end of life on December 31, 2020 and has been removed from this wizard.

Root device type: ebs Virtualization type: hvm ENA Enabled: Yes

● 64-bit (x86)
○ 64-bit (Arm)

Select t2 micro free tier.



1. Choose AMI | 2. Choose Instance Type | 3. Configure Instance | 4. Add Storage | 5. Add Tags | 6. Configure Security Group | 7. Review

Step 2: Choose an Instance Type

Amazon EC2 provides a wide selection of instance types optimized to fit different use cases. Instances are virtual servers that can run applications. They have varying combinations of CPU, memory, storage, and networking capacity, and give you the flexibility to choose the appropriate mix of resources for your applications. Learn more about instance types and how they can meet your computing needs.

Filter by: All instance families | Current generation | Show/Hide Columns

Currently selected: t2.micro (- ECUs, 1 vCPUs, 2.5 GHz, -, 1 GiB memory, EBS only)

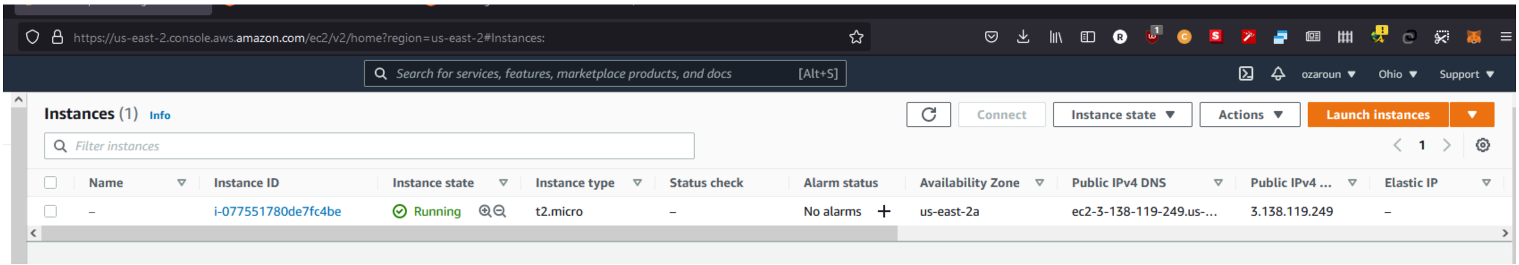
	Family	Type	vCPUs	Memory (GiB)	Instance Storage (GB)	EBS-Optimized Available	Network Performance	IPv6 Support
□	t2	t2.nano	1	0.5	EBS only	-	Low to Moderate	Yes
■	t2	t2.micro Free tier eligible	1	1	EBS only	-	Low to Moderate	Yes

Click Launch again on the next screen. It would ask you to select an existing key pair if you haven’t created one already.

Create one.

Finish creating the instance.

Connect to the instance form the screen after selecting it here –

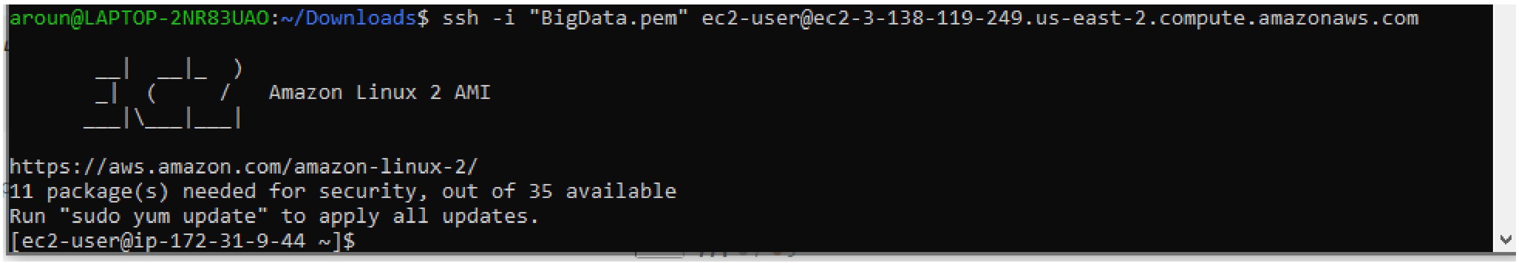


https://us-east-2.console.aws.amazon.com/ec2/v2/home?region=us-east-2#Instances:

Instances (1) Info

	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 ...	Elastic IP
□	–	i-077551780de7fc4be	Running	t2.micro	–	No alarms	us-east-2a	ec2-3-138-119-249.us-...	3.138.119.249	–

Connect to the EC2 using SSH



```
aroun@LAPTOP-2NR83UAO:~/Downloads$ ssh -i "BigData.pem" ec2-user@ec2-3-138-119-249.us-east-2.compute.amazonaws.com

       __|  __|_  )
       _|  (     /   Amazon Linux 2 AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-2/
11 package(s) needed for security, out of 35 available
Run "sudo yum update" to apply all updates.
[ec2-user@ip-172-31-9-44 ~]$
```

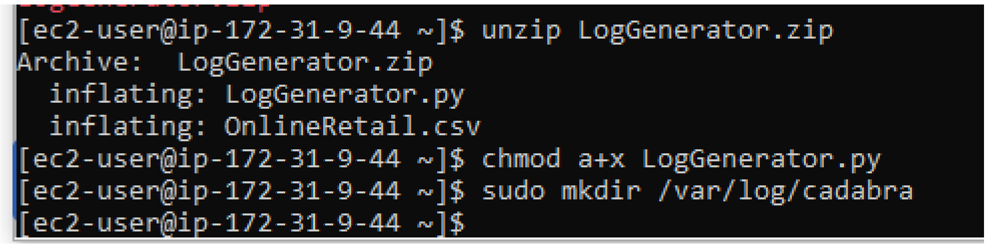
Install Kinesis agent with the command –

sudo yum install -y aws-kinesis-agent

Download the log generator for the purpose of the project – wget https://media.sundog-soft.com/AWSBigData/LogGenerator.zip

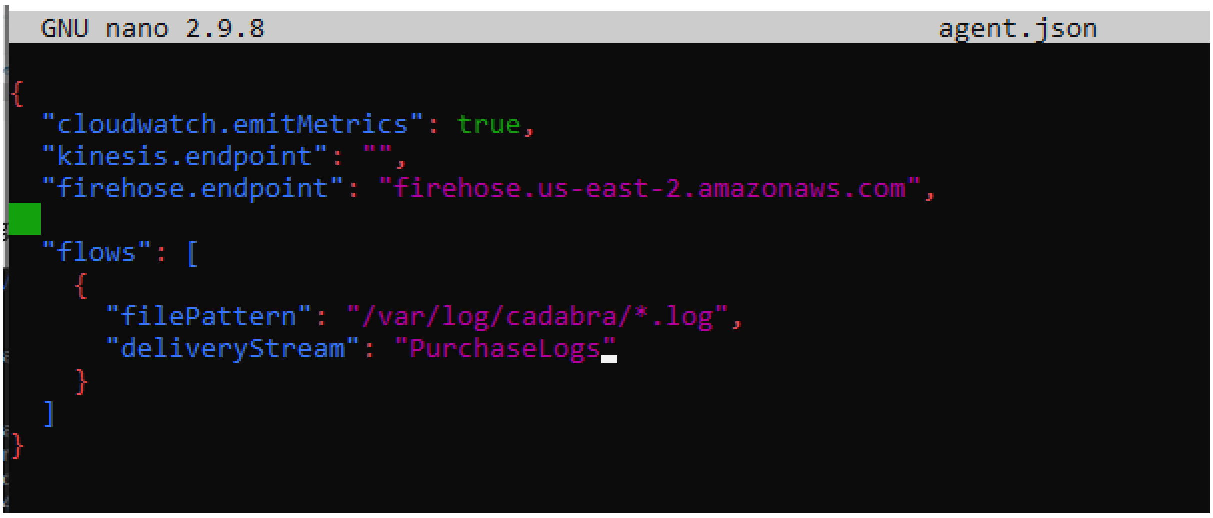
Unzip it – unzip LogGenerator.zip

Change permissions for the python file



```
[ec2-user@ip-172-31-9-44 ~]$ unzip LogGenerator.zip
Archive:  LogGenerator.zip
  inflating: LogGenerator.py
  inflating: OnlineRetail.csv
[ec2-user@ip-172-31-9-44 ~]$ chmod a+x LogGenerator.py
[ec2-user@ip-172-31-9-44 ~]$ sudo mkdir /var/log/cadabra
[ec2-user@ip-172-31-9-44 ~]$
```

Change Kinesis Agent setting in -

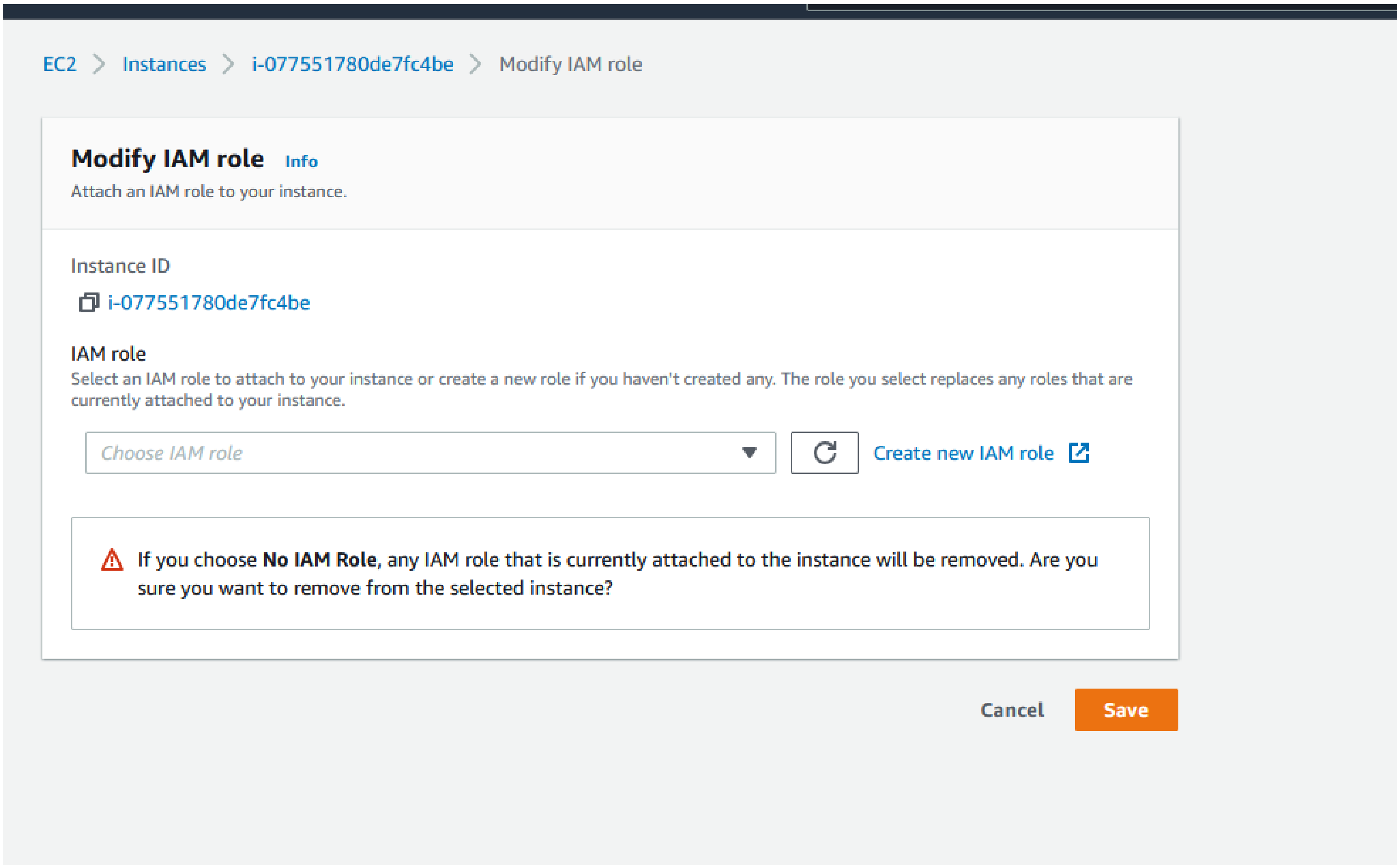


```
  GNU nano 2.9.8                                             agent.json

{
  "cloudwatch.emitMetrics": true,
  "kinesis.endpoint": "",
  "firehose.endpoint": "firehose.us-east-2.amazonaws.com",

  "flows": [
    {
      "filePattern": "/var/log/cadabra/*.log",
      "deliveryStream": "PurchaseLogs"
    }
  ]
}
```

In EC2 console screen select the instance, go to actions, go to security, go to IAM Role.



Modify IAM role Info

Attach an IAM role to your instance.

Instance ID

i-077551780de7fc4be

IAM role

Select an IAM role to attach to your instance or create a new role if you haven't created any. The role you select replaces any roles that are currently attached to your instance.

If you choose **No IAM Role**, any IAM role that is currently attached to the instance will be removed. Are you sure you want to remove from the selected instance?

Click Create new IAM role.

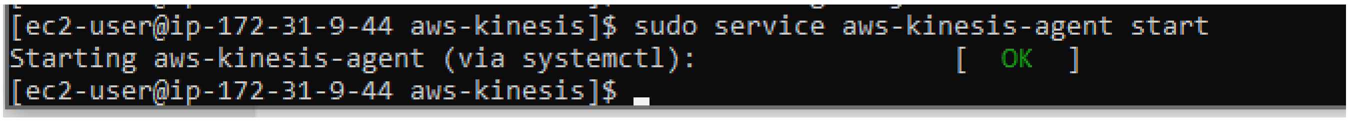
Select Create Role

Select EC2. Select AdministratorAccess for ease. Name it and then create role.

Back to the EC2 screen where we selected the create new IAM role, use the refresh button and select the new IAM role and save.

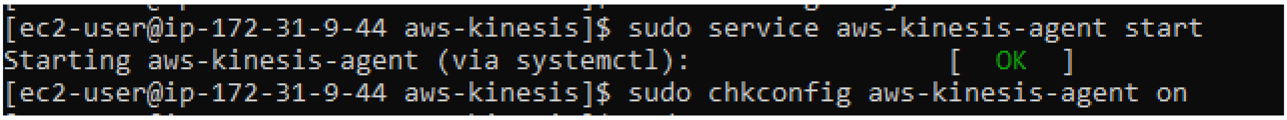
Return to the Kinesis agent json file.

Start the kinesis agent –



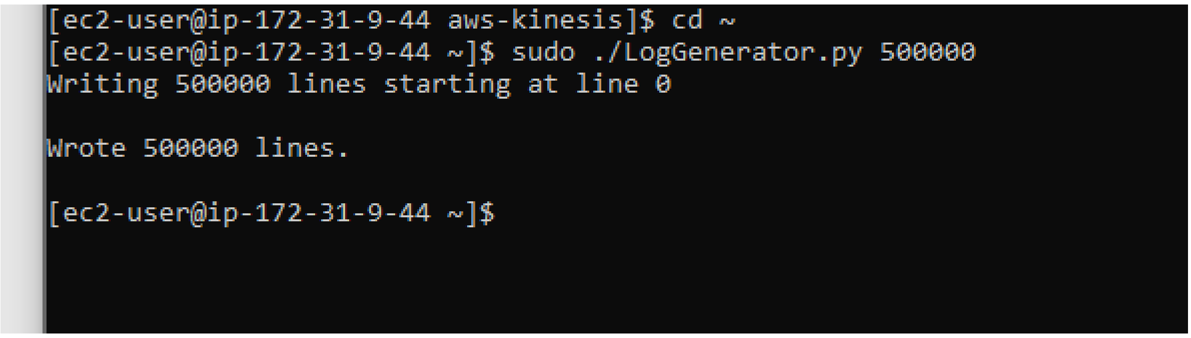
```
[ec2-user@ip-172-31-9-44 aws-kinesis]$ sudo service aws-kinesis-agent start
Starting aws-kinesis-agent (via systemctl):                [  OK  ]
[ec2-user@ip-172-31-9-44 aws-kinesis]$ _
```

To make it start automatically



```
[ec2-user@ip-172-31-9-44 aws-kinesis]$ sudo service aws-kinesis-agent start
Starting aws-kinesis-agent (via systemctl):                [  OK  ]
[ec2-user@ip-172-31-9-44 aws-kinesis]$ sudo chkconfig aws-kinesis-agent on
```

Start the stream –

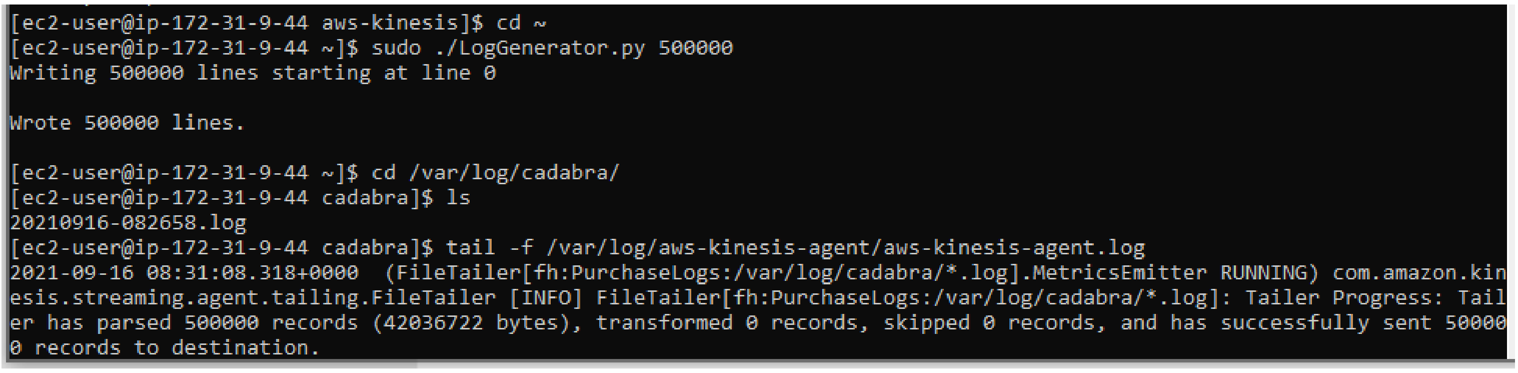


```
[ec2-user@ip-172-31-9-44 aws-kinesis]$ cd ~
[ec2-user@ip-172-31-9-44 ~]$ sudo ./LogGenerator.py 500000
Writing 500000 lines starting at line 0

Wrote 500000 lines.

[ec2-user@ip-172-31-9-44 ~]$
```

Check for the log and also can check the live number of logs generated using the tail function

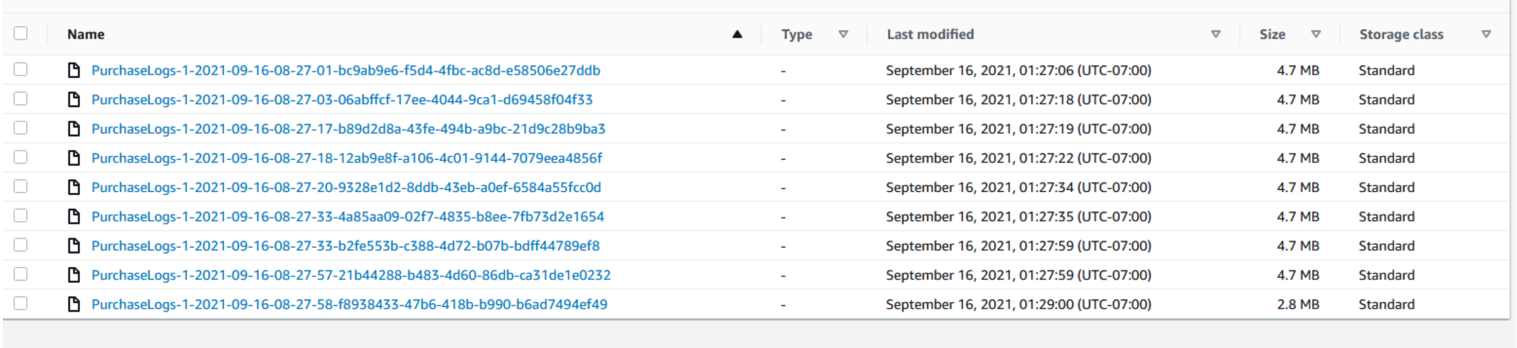


```
[ec2-user@ip-172-31-9-44 aws-kinesis]$ cd ~
[ec2-user@ip-172-31-9-44 ~]$ sudo ./LogGenerator.py 500000
Writing 500000 lines starting at line 0

Wrote 500000 lines.

[ec2-user@ip-172-31-9-44 ~]$ cd /var/log/cadabra/
[ec2-user@ip-172-31-9-44 cadabra]$ ls
20210916-082658.log
[ec2-user@ip-172-31-9-44 cadabra]$ tail -f /var/log/aws-kinesis-agent/aws-kinesis-agent.log
2021-09-16 08:31:08.318+0000  (FileTailer[fh:PurchaseLogs:/var/log/cadabra/*.log].MetricsEmitter RUNNING) com.amazon.kin
esis.streaming.agent.tailing.FileTailer [INFO] FileTailer[fh:PurchaseLogs:/var/log/cadabra/*.log]: Tailer Progress: Tail
er has parsed 500000 records (42036722 bytes), transformed 0 records, skipped 0 records, and has successfully sent 50000
0 records to destination.
```

Data loaded into S3 in 5mb chunks



□	Name ▲	Type ▽	Last modified ▽	Size ▽	Storage class ▽
□	PurchaseLogs-1-2021-09-16-08-27-01-bc9ab9e6-f5d4-4fbc-ac8d-e58506e27ddb	-	September 16, 2021, 01:27:06 (UTC-07:00)	4.7 MB	Standard
□	PurchaseLogs-1-2021-09-16-08-27-03-06abffcf-17ee-4044-9ca1-d69458f04f33	-	September 16, 2021, 01:27:18 (UTC-07:00)	4.7 MB	Standard
□	PurchaseLogs-1-2021-09-16-08-27-17-b89d2d8a-43fe-494b-a9bc-21d9c28b9ba3	-	September 16, 2021, 01:27:19 (UTC-07:00)	4.7 MB	Standard
□	PurchaseLogs-1-2021-09-16-08-27-18-12ab9e8f-a106-4c01-9144-7079eea4856f	-	September 16, 2021, 01:27:22 (UTC-07:00)	4.7 MB	Standard
□	PurchaseLogs-1-2021-09-16-08-27-20-9328e1d2-8ddb-43eb-a0ef-6584a55fcc0d	-	September 16, 2021, 01:27:34 (UTC-07:00)	4.7 MB	Standard
□	PurchaseLogs-1-2021-09-16-08-27-33-4a85aa09-02f7-4835-b8ee-7fb73d2e1654	-	September 16, 2021, 01:27:35 (UTC-07:00)	4.7 MB	Standard
□	PurchaseLogs-1-2021-09-16-08-27-33-b2fe553b-c388-4d72-b07b-bdff44789ef8	-	September 16, 2021, 01:27:59 (UTC-07:00)	4.7 MB	Standard
□	PurchaseLogs-1-2021-09-16-08-27-57-21b44288-b483-4d60-86db-ca31de1e0232	-	September 16, 2021, 01:27:59 (UTC-07:00)	4.7 MB	Standard
□	PurchaseLogs-1-2021-09-16-08-27-58-f8938433-47b6-418b-b990-b6ad7494ef49	-	September 16, 2021, 01:29:00 (UTC-07:00)	2.8 MB	Standard

So now we have a S3 data lake ready.

The end goal is the create an order history app. So that a user is able to access their order history on cadabra.com or whatever client they are using.

So we already have server logs from the EC2 instance.

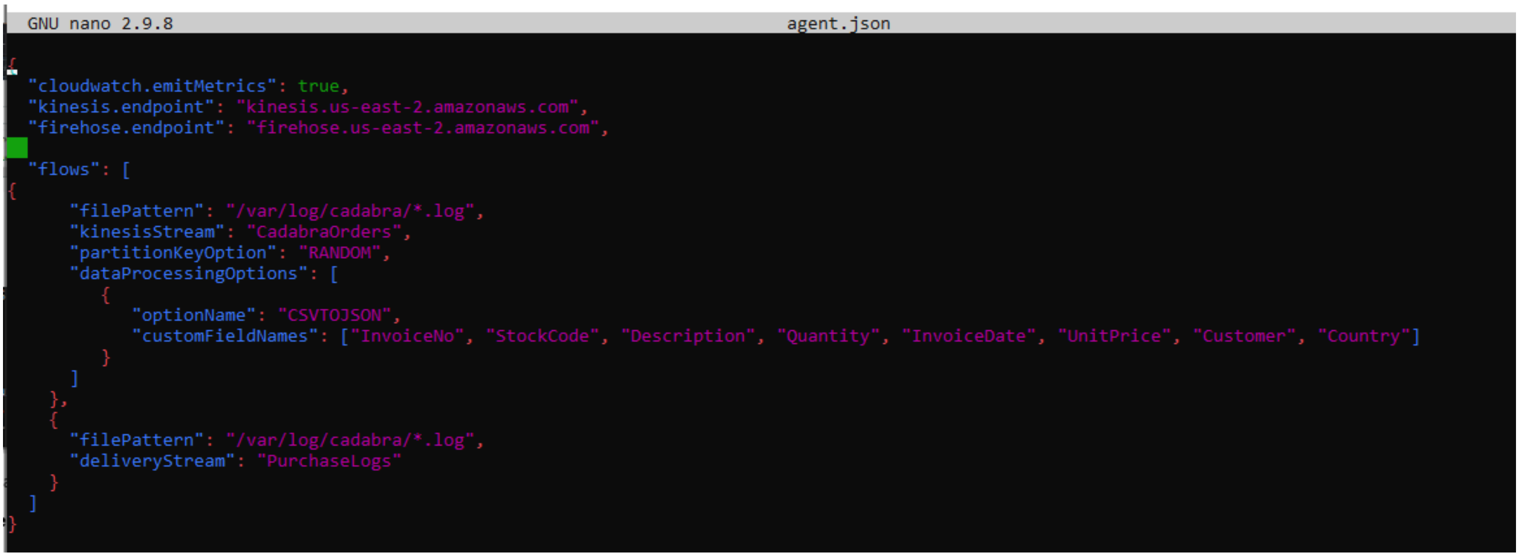
Now instead of Firehose we're gonna publish it into Kinesis Data Stream so that the data is accessible in real time

3. Connecting the EC2 to a Kinesis Data Stream

On the Kinesis Console. Create Data Stream.

Name it and select the number of shards you are going to use. We are going to select 1.

Go to the ec2 instance and configure the kinesis agent json file for the logs to be sent to this stream.



```
  GNU nano 2.9.8                                                        agent.json

{
  "cloudwatch.emitMetrics": true,
  "kinesis.endpoint": "kinesis.us-east-2.amazonaws.com",
  "firehose.endpoint": "firehose.us-east-2.amazonaws.com",

  "flows": [
{
      "filePattern": "/var/log/cadabra/*.log",
      "kinesisStream": "CadabraOrders",
      "partitionKeyOption": "RANDOM",
      "dataProcessingOptions": [
         {
            "optionName": "CSVTOJSON",
            "customFieldNames": ["InvoiceNo", "StockCode", "Description", "Quantity", "InvoiceDate", "UnitPrice", "Customer", "Country"]
         }
      ]
    },
    {
      "filePattern": "/var/log/cadabra/*.log",
      "deliveryStream": "PurchaseLogs"
    }
  ]
}
```

Now the data is going to the S3 from the firehose and to the Kinesis Data Stream

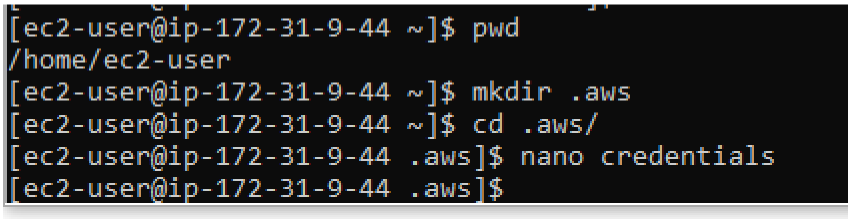
4. Integrating DynamoDB

From the AWS console, select DynamoDB.

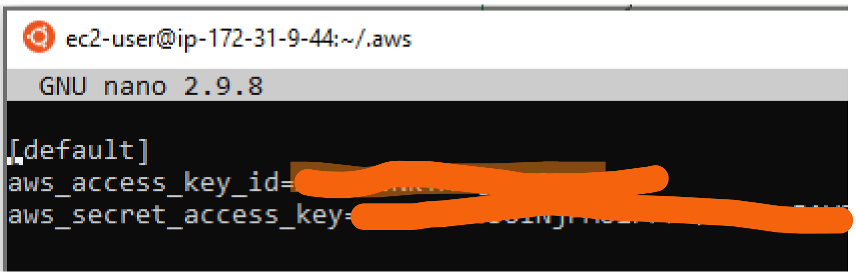
Create a table – CadabraOrders. Give Partition key – CustomerID of type Number. Add sort key – OrderID of type String.

On the EC2 instance, install the boto3 library – pip3 install boto3

Now we need to create some credentials files so that boto3 knows how to log on to S3.



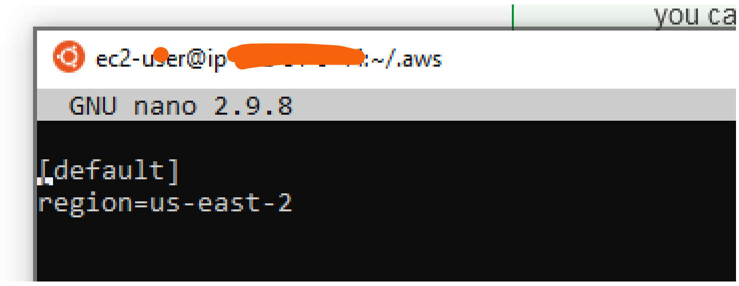
```
[ec2-user@ip-172-31-9-44 ~]$ pwd
/home/ec2-user
[ec2-user@ip-172-31-9-44 ~]$ mkdir .aws
[ec2-user@ip-172-31-9-44 ~]$ cd .aws/
[ec2-user@ip-172-31-9-44 .aws]$ nano credentials
[ec2-user@ip-172-31-9-44 .aws]$
```



```
ec2-user@ip-172-31-9-44:~/.aws
  GNU nano 2.9.8

[default]
aws_access_key_id=[redacted]
aws_secret_access_key=[redacted]
```

Create config file in the same place.



```
ec2-u[redacted]er@ip[redacted]:~/.aws
  GNU nano 2.9.8

[default]
region=us-east-2
```

Now, downloading the consumer script in the home directory.

wget https://media.sundog-soft.com/AWSBigData/Consumer.py

Make the script executable - chmod a+x Consumer.py

From a second ssh into the EC2 instance, run LogGeneratory.py file for 100 records - ./LogGenerator.py 100

After a minute or so the first console where Consumer.py is running would pick up the records for the Kinesis Stream.

Now, we have created a system that works from end-to-end. We monitor new information being uploaded into the log directory of the EC2 host. The kinesis stream picks that data up. Another app in our case the Comsumer.py inserts that data into a DynamoDB table. We can now imagine a mobile application that talks directly to the DynamoDB instance and returns the information.

4. Integrating AWS Lambda

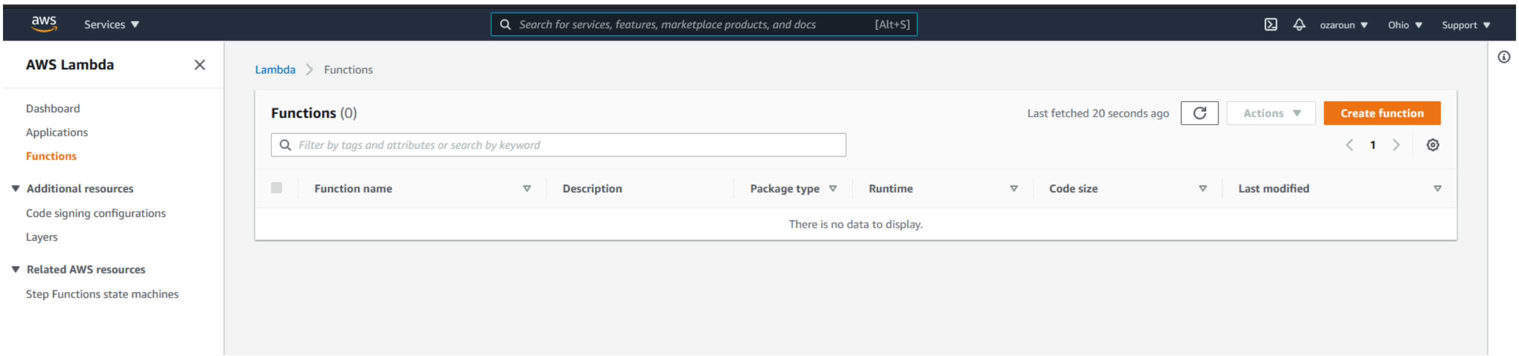
Right now we have a consumer script that is running on our EC2 instance playing the role of a lambda function. It is not a very scalable function.

Our Lambda function in terms of security would need to consume data from the kinesis stream and write data into the DynamoDB.

Let’s create an IAM role for lambda.

Go to IAM console -> Go to Roles -> Click create role -> Select Lambda -> Attach the following permission to read from the Kinesis Stream and Write to the DynamoDB : AmazonKinesisReadOnlyAccess , AmazonDynamoDBFullAccess -> Click Next -> Skip tags -> Give the role a name (in our case CadabraOrders) -> Create Role

Now back to the AWS console. Select Lambda.


aws
Services ▼
Search for services, features, marketplace products, and docs [Alt+S]
ozaroun ▼
Ohio ▼
Support ▼
AWS Lambda
Dashboard
Applications
Functions
Additional resources
Code signing configurations
Layers
Related AWS resources
Step Functions state machines
Lambda > Functions
Actions ▼
Create function
Filter by tags and attributes or search by keyword

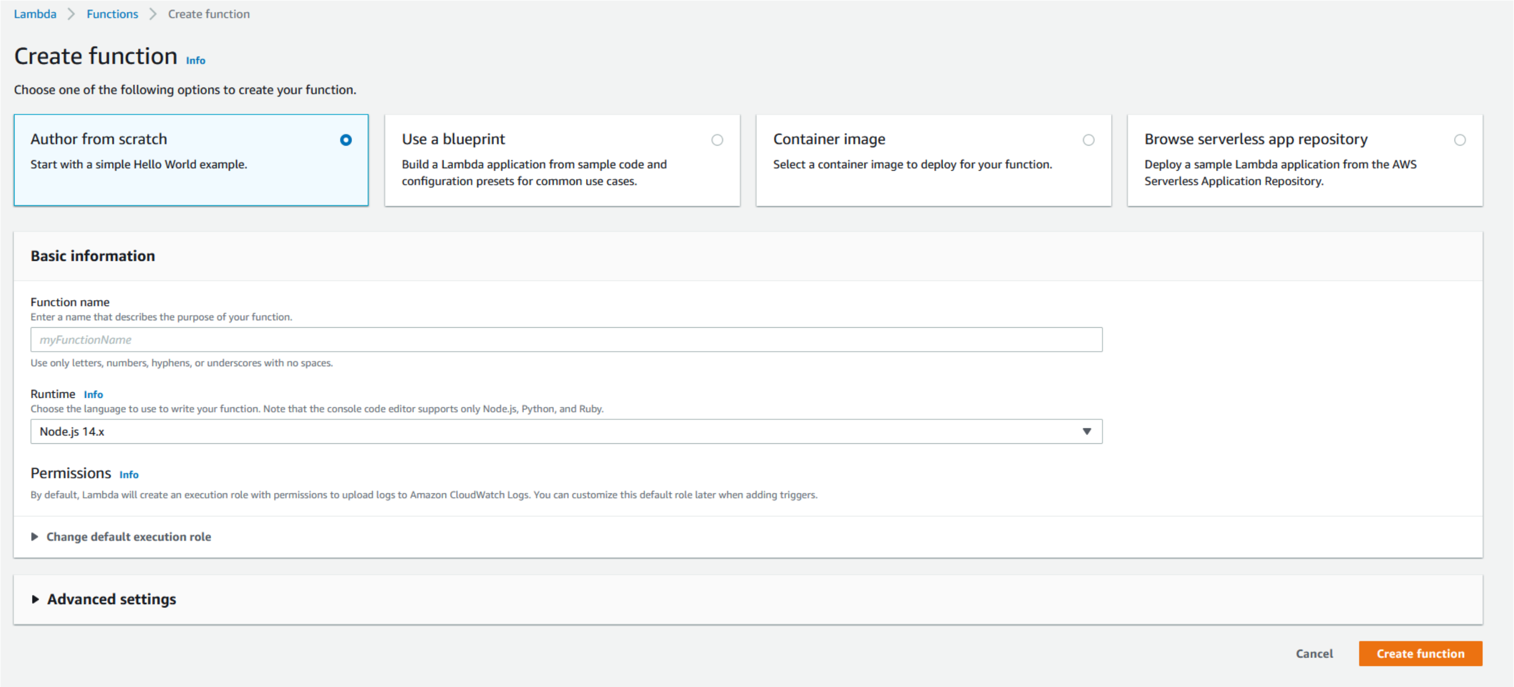
Functions (0)

Last fetched 20 seconds ago

Function name	Description	Package type	Runtime	Code size	Last modified

There is no data to display.

Select the create function button.



Create function

Choose one of the following options to create your function.

● Author from scratch
Start with a simple Hello World example.

○ Use a blueprint
Build a Lambda application from sample code and configuration presets for common use cases.

○ Container image
Select a container image to deploy for your function.

○ Browse serverless app repository
Deploy a sample Lambda application from the AWS Serverless Application Repository.

Basic information

Function name
Enter a name that describes the purpose of your function.
myFunctionName
Use only letters, numbers, hyphens, or underscores with no spaces.

Runtime
Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.
Node.js 14.x

Permissions
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.

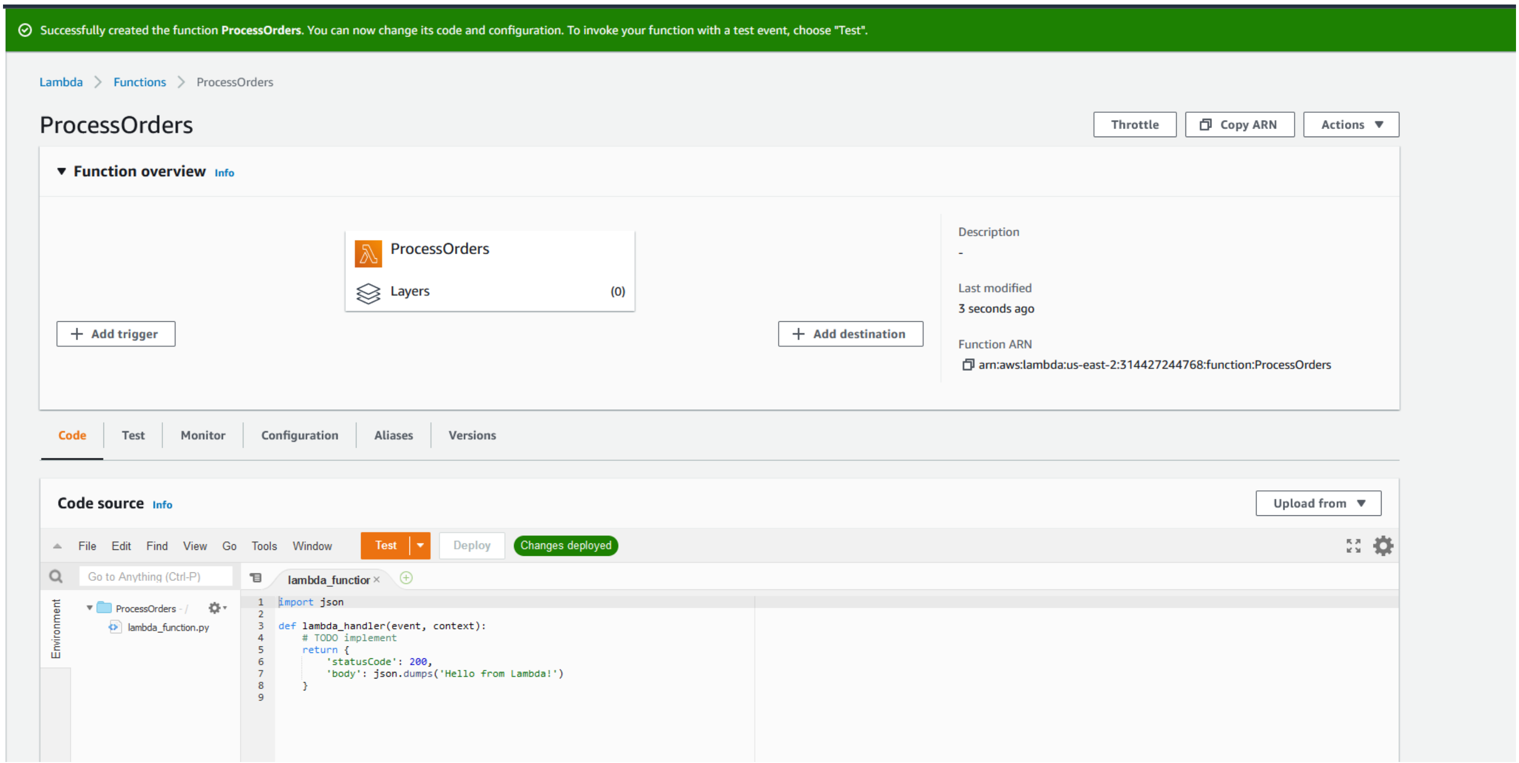
▸ Change default execution role

▸ Advanced settings

Give it a name (in our case ProcessOrders). Select the runtime environment – Python 3.x.

Select the existing IAM role that we create for it.

The Next Screen –



Successfully created the function **ProcessOrders**. You can now change its code and configuration. To invoke your function with a test event, choose "Test".

ProcessOrders

Function overview

ProcessOrders
Layers (0)

Description
-

Last modified
3 seconds ago

Function ARN
arn:aws:lambda:us-east-2:314427244768:function:ProcessOrders

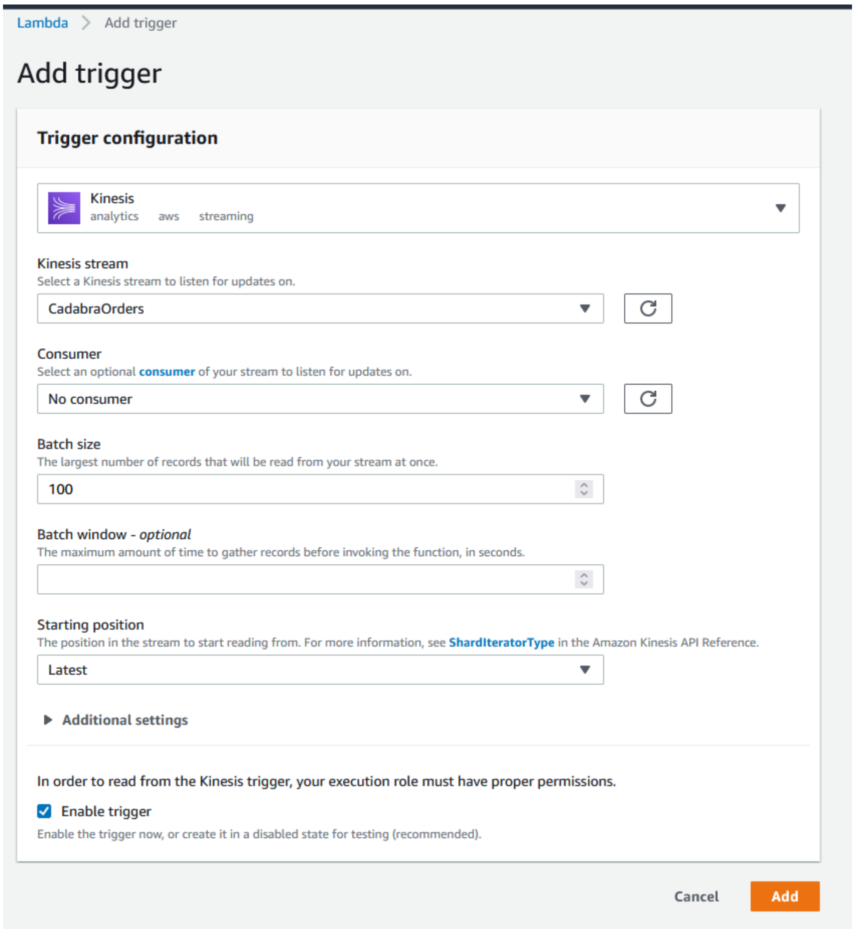
Code source

```
import json

def lambda_handler(event, context):
    # TODO implement
    return {
        'statusCode': 200,
        'body': json.dumps('Hello from Lambda!')
    }
```

We want to add a trigger to feed data into this Lambda function from our Kinesis Stream.

Select Add Trigger -> Select Kinesis Streams -> Select the Kinesis Stream (in our case Cadabra Orders) -> Click Add.



Lambda > Add trigger

Add trigger

Trigger configuration

Kinesis
analytics aws streaming

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

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

Batch window - *optional*
The maximum amount of time to gather records before invoking the function, in seconds.

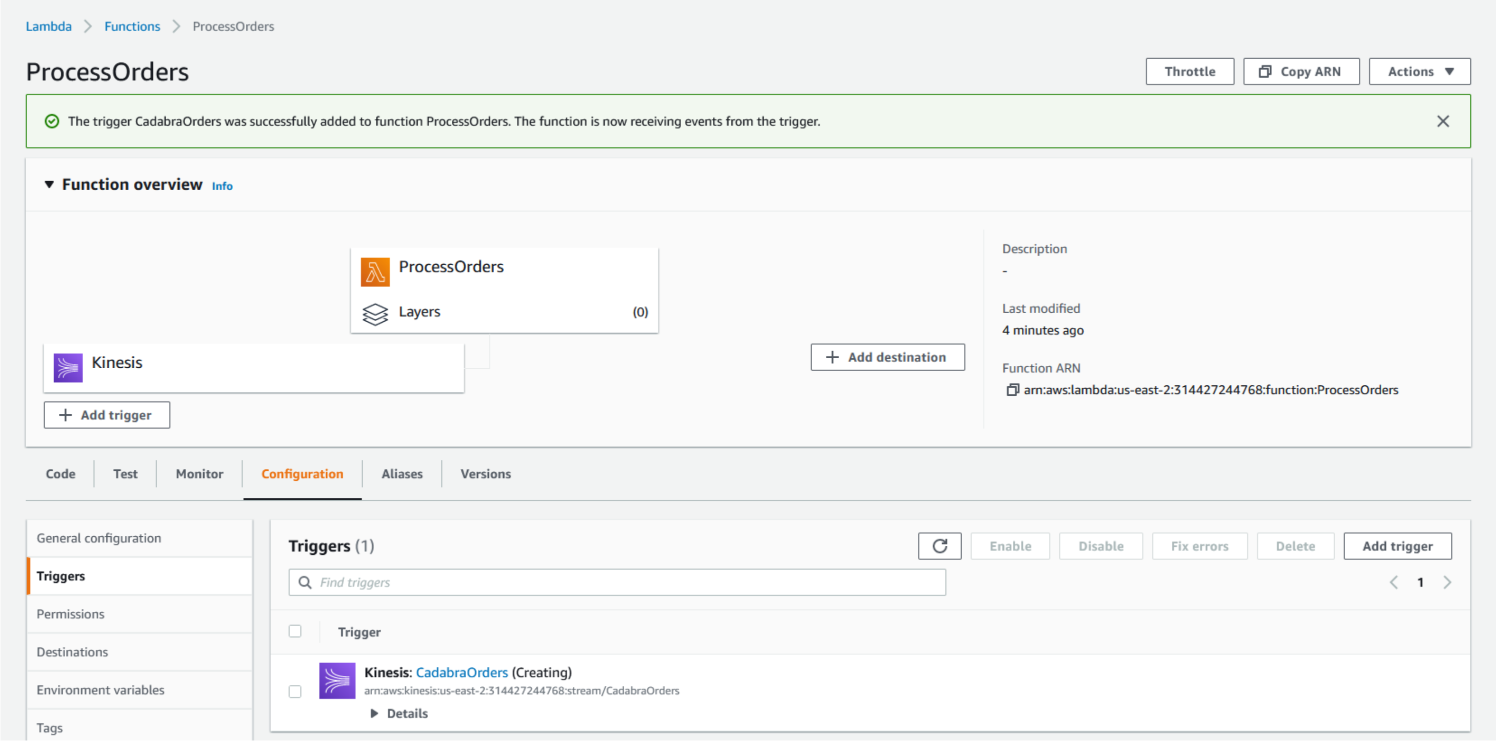
Starting position
The position in the stream to start reading from. For more information, see ShardIteratorType in the Amazon Kinesis API Reference.
Latest

▶ Additional settings

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).

The Lambda function screen looks like –



Lambda > Functions > ProcessOrders

ProcessOrders

The trigger CadabraOrders was successfully added to function ProcessOrders. The function is now receiving events from the trigger.

▼ Function overview Info

Description
-

Last modified
4 minutes ago

Function ARN
arn:aws:lambda:us-east-2:314427244768:function:ProcessOrders

Triggers (1)

□	Trigger
□	Kinesis: CadabraOrders (Creating) arn:aws:kinesis:us-east-2:314427244768:stream/CadabraOrders ▶ Details

Click on code and add the following code –

```
import base64
import json
import boto3
import decimal
import uuid

def lambda_handler(event, context):
    item = None
    dynamo_db = boto3.resource('dynamodb')
    table = dynamo_db.Table('CadabraOrders')
    decoded_record_data = [base64.b64decode(record['kinesis']['data']) for record in event['Records']]
    deserialized_data = [json.loads(decoded_record) for decoded_record in decoded_record_data]

    with table.batch_writer() as batch_writer:
        for item in deserialized_data:
            # We've added a try / except block here to deal with invalid input rows more gracefully.
            # Be aware there are stretches of the input data that have no customer ID's at all,
            # keep trying the LogGenerator script to get past that if you run into it.
            try:
                invoice = item['InvoiceNo']
                customer = int(item['Customer'])
                orderDate = item['InvoiceDate']
                quantity = item['Quantity']
                description = item['Description']
                unitPrice = item['UnitPrice']
                country = item['Country'].rstrip()
                stockCode = item['StockCode']

                # Construct a unique sort key for this line item
                # We've added a uuid at the end as there is some duplicate invoice/stockcode
                # data in our sample data.
                orderID = invoice + "-" + stockCode + "-" + uuid.uuid4().hex

                batch_writer.put_item(Item = {
                        'CustomerID': decimal.Decimal(customer),
                        'OrderID': orderID,
                        'OrderDate': orderDate,
                        'Quantity': decimal.Decimal(quantity),
                        'UnitPrice': decimal.Decimal(unitPrice),
                        'Description': description,
                        'Country': country
                    }
                )
                print("Wrote item into batch.")
            except:
                print("Error processing invalid input row.")
```

Save the changes. Deploy the code.

We don’t explicitly have to mention DynamoDB in the destination, our script is doing that for us.

Lets ssh into our EC2 instance.

Run the command - sudo ./LogGenerator.py 100

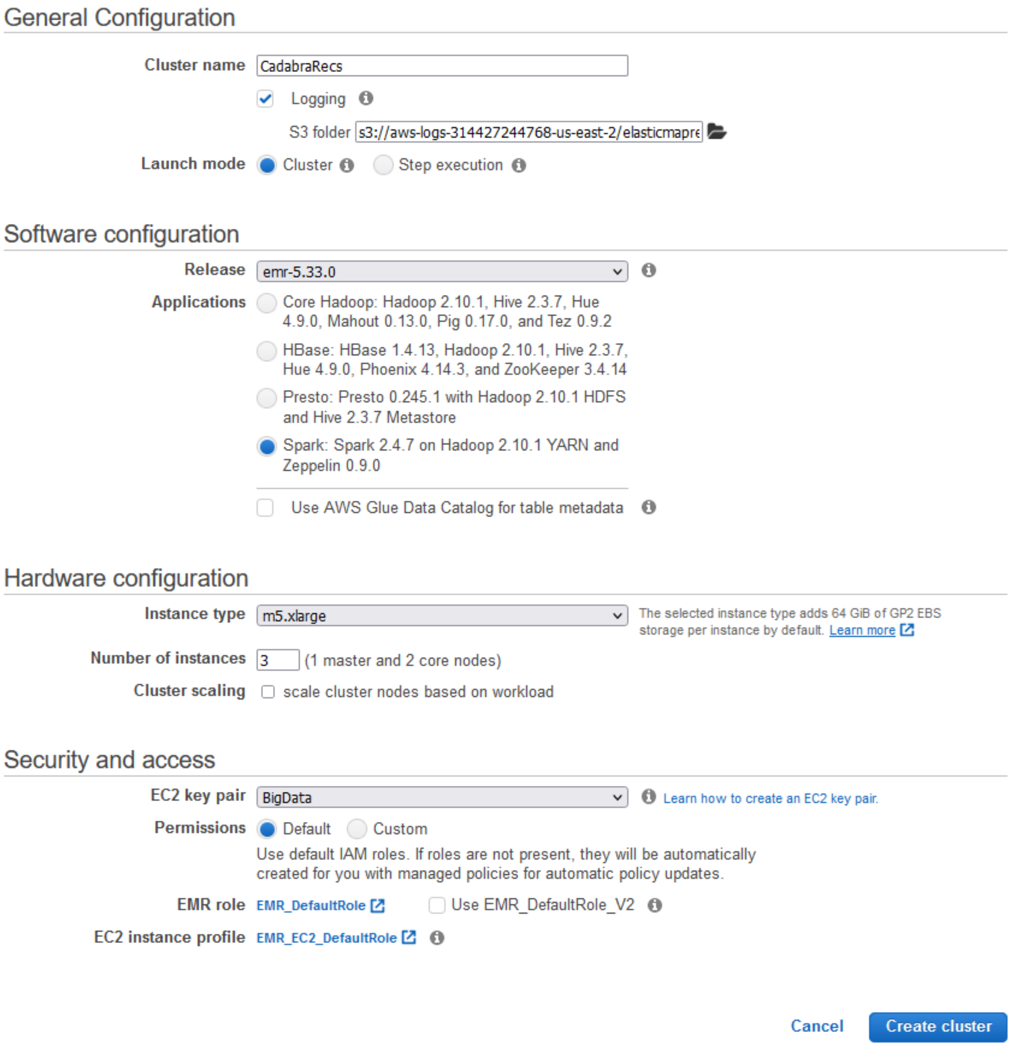
Restart the aws-kinesis-agent - sudo service aws-kinesis-agent restart

The newer records for some reason didn't show up on the DynamoDB as quickly as I had expected but they did show up after a few minutes.

5. EMR MapReduce –

We'll build a product recommendations system for "kadabra.com" , the application we have been building. We already have deployed a firehose that dumps data from the EC2 instance to the S3 bucket. We can now deploy an EMR Cluster and using Apache Spark and MLlib we can generate recommendations based on order data in S3.

Head over to the AWS console and click on EMR -> Click on create cluster ->



General Configuration

Cluster name: CadabraRecs

☑ Logging

S3 folder: s3://aws-logs-314427244768-us-east-2/elasticmapre

Launch mode: ● Cluster ○ Step execution

Software configuration

Release: emr-5.33.0

Applications:
○ Core Hadoop: Hadoop 2.10.1, Hive 2.3.7, Hue 4.9.0, Mahout 0.13.0, Pig 0.17.0, and Tez 0.9.2
○ HBase: HBase 1.4.13, Hadoop 2.10.1, Hive 2.3.7, Hue 4.9.0, Phoenix 4.14.3, and ZooKeeper 3.4.14
○ Presto: Presto 0.245.1 with Hadoop 2.10.1 HDFS and Hive 2.3.7 Metastore
● Spark: Spark 2.4.7 on Hadoop 2.10.1 YARN and Zeppelin 0.9.0

□ Use AWS Glue Data Catalog for table metadata

Hardware configuration

Instance type: m5.xlarge

The selected instance type adds 64 GiB of GP2 EBS storage per instance by default. Learn more

Number of instances: 3 (1 master and 2 core nodes)

Cluster scaling: □ scale cluster nodes based on workload

Security and access

EC2 key pair: BigData

Learn how to create an EC2 key pair.

Permissions: ● Default ○ Custom

Use default IAM roles. If roles are not present, they will be automatically created for you with managed policies for automatic policy updates.

EMR role: EMR_DefaultRole □ Use EMR_DefaultRole_V2

EC2 instance profile: EMR_EC2_DefaultRole

The EMR cluster does not come under the free tier. Create the cluster.