

Loading Data to AWS Elasticsearch with DynamoDB Streams and Lambda



Janitha Tennakoon

Follow

May 24 · 9 min read ★



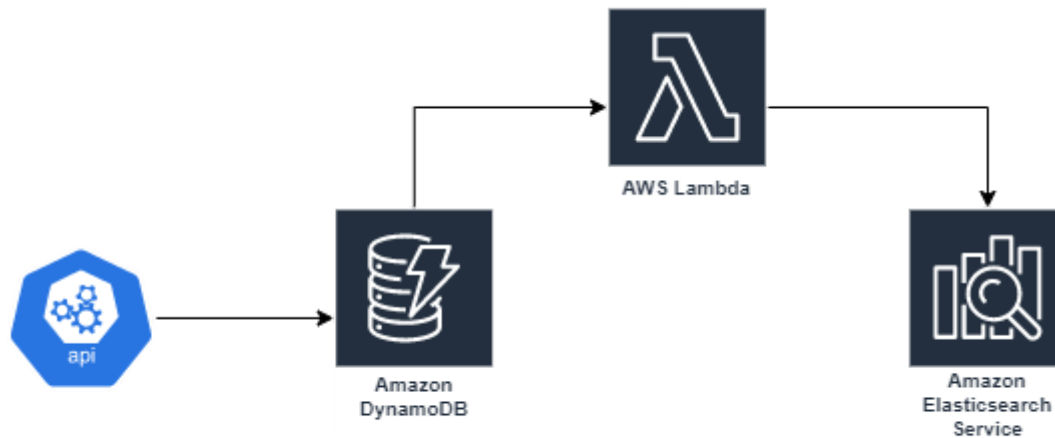
Amazon Elasticsearch Service



Elasticsearch is an open-source, RESTful, distributed search, and analytics solution that is currently widely used by many companies worldwide. It is a search engine based on Apache Lucene library. Elasticsearch is widely used in text analysis, log analysis, text-based search and many more use cases.

Amazon Elasticsearch is a fully managed service that makes it easy for us to deploy, secure, and run Elasticsearch. The service provides support for open-source Elasticsearch APIs, **Kibana** integration with **Logstash**, and other AWS services. In this post I am going to talk about how we can load streaming data into AWS Elasticsearch

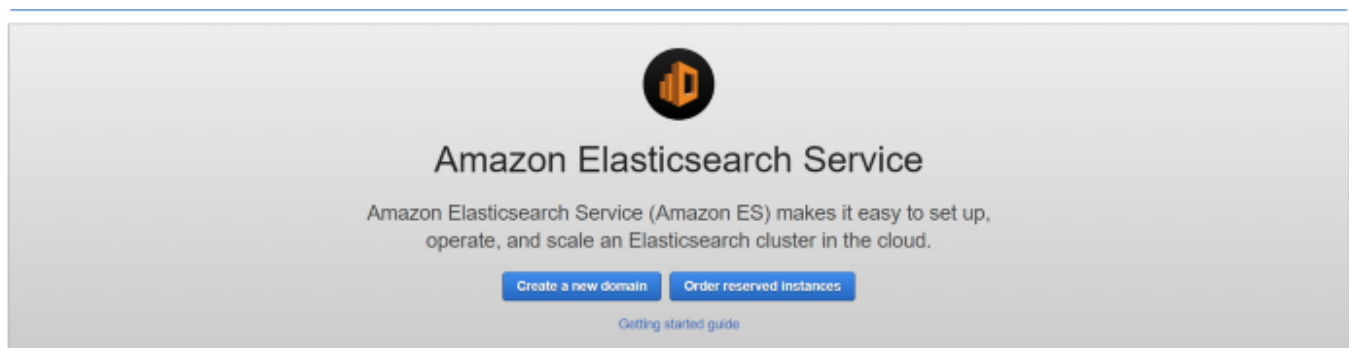
service using AWD DynamoDB streams and AWS Lambda. So in our scenario when a web application or a web service adds data to our dynamoDB database, it will trigger an AWS Lambda function which will automatically insert those data to our Elasticsearch service which we can use in our use case.



I will assume for this post you have a basic idea on what is Elasticsearch, dynamoDB, and Lambda. Hence in this post I am not going to discuss thoroughly what Elasticsearch, dynamoDB, and Lambda does. Ok, enough on explanations, let's dive straight to implementation.

Creating an AWS Elasticsearch Instance

Let's first create our Elasticsearch instance on our AWS. Search for Elasticsearch service from services and go to the dashboard for creating Elasticsearch service. If it is your first time creating an elasticsearch service you will be greeted by the following screen. Click on Create a new domain to start with creating our new elasticsearch domain.



We are going to create our domain only for learning purpose so the configurations that we will select for our domain is only applicable for learning purposes. For production you must follow the correct configuration setup. Since we are using this domain for only learning select **Deployment type** as Development and testing and click next.

Create Elasticsearch domain

Step 1: Choose deployment type

Step 2: Configure domain

Step 3: Configure access and security

Step 4: Review

Choose deployment type

Deployment types specify common settings for your use case. After creating the domain, you can change these settings at any time.

Deployment type

- ☐ Production
Multiple Availability Zones and dedicated master nodes for higher availability.
- ☒ Development and testing
One Availability Zone for when you just need an Elasticsearch endpoint.
- ☐ Custom
Choose settings from all available options.

Version

Select the version of Elasticsearch for your domain.

Elasticsearch version 7.4 (latest)

Cancel Next

On the next screen we need to provide a domain name. After that select the **instance type** as t2.small.elasticsearch because that is the only instance type that is available for the free tier. You can leave all the other options as default values without any change.

Create Elasticsearch domain

Step 1: Choose deployment type

Step 2: Configure domain

Step 3: Configure access and security

Step 4: Review

Configure domain

A domain is the collection of resources needed to run Elasticsearch. The domain name will be part of your domain endpoint.

Elasticsearch domain name movie-es

The name must start with a lowercase letter and must be between 3 and 28 characters. Valid characters are a-z (lowercase only), 0-9, and - (hyphen).

Data nodes

Select an instance type that corresponds to the compute, memory, and storage needs of your application. Consider the size of your Elasticsearch indices, number of shards and replicas, type of queries, and volume of requests. [Learn more](#)

Instance type t2.small.elasticsearch

The AWS Free Tier includes usage of up to 750 hours per month of t2.micro or t2.small instance usage and up to 10 GB of Magnetic or General Purpose EBS storage.
Amazon Elasticsearch Service Free Tier
t2.small.elasticsearch instance type needs EBS storage.

The selected instance type (t2.small.elasticsearch) does not support encryption at rest.

Number of nodes 1

On the next screen we will be asked to configure security for our elasticsearch domain. The recommended is to use VPC access where our domain will be in a private network with only instances within our VPC network that will have access. But in our case make it as Public access.

Configure access and security

Amazon Elasticsearch Service offers numerous security features, including fine-grained access control, IAM, Cognito authentication for Kibana, encryption, and VPC access. [Learn more](#)

Network configuration

Choose Internet or VPC access. To enable VPC access, we use private IP addresses from your VPC, which provides an inherent layer of security. You control network access within your VPC using security groups. Optionally, you can add an additional layer of security by applying a restrictive access policy. Internet endpoints are publicly accessible. If you select public access, you should secure your domain with an access policy that only allows specific users or IP addresses to access the domain.

- ☐ VPC access (Recommended)
☒ Public access

Fine-grained access control – powered by Open Distro for Elasticsearch

Fine-grained access control provides numerous features to help you keep your data secure. Features include document-level security, field-level security, read-only Kibana users, and Kibana tenants. Fine-grained access control requires a master user.

Set a master user to an IAM account using an ARN, or store a master user in the Elasticsearch internal database by creating a master username and password. After your domain is set up, you can use Kibana or the REST APIs to configure additional users and permissions. [Learn more](#)

Below on the same page you will see where we are setting access policy for our domain. Since this is only for learning add **Allow open access to the domain** which will make any IP address have access. Here we can specify either an IAM role or specific AWS user accounts as well.

Access policy

Access policies control whether a request is accepted or rejected when it reaches the Amazon Elasticsearch Service domain. If you specify an account, user, or role in this policy, you must sign your requests. [Learn more](#)

Custom policy builder allows at most 10 elements. Use a JSON-defined access policy to define a policy with more than 10 elements.

Domain access policy Custom access policy

Allow or deny access by AWS account ID, account ARN, IAM user ARN, IAM role ARN, IPv4 address, or CIDR block.

IPv4 address * Allow Remove element

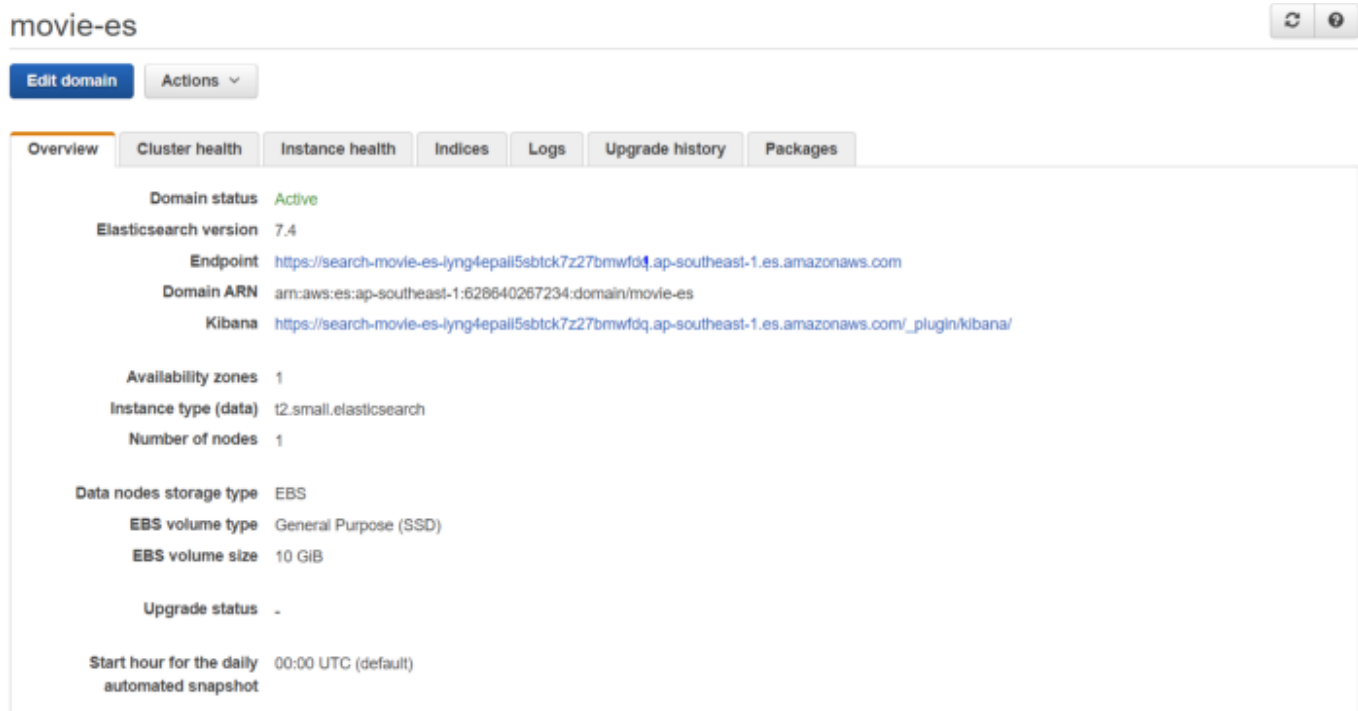
[Add element](#)

Encryption

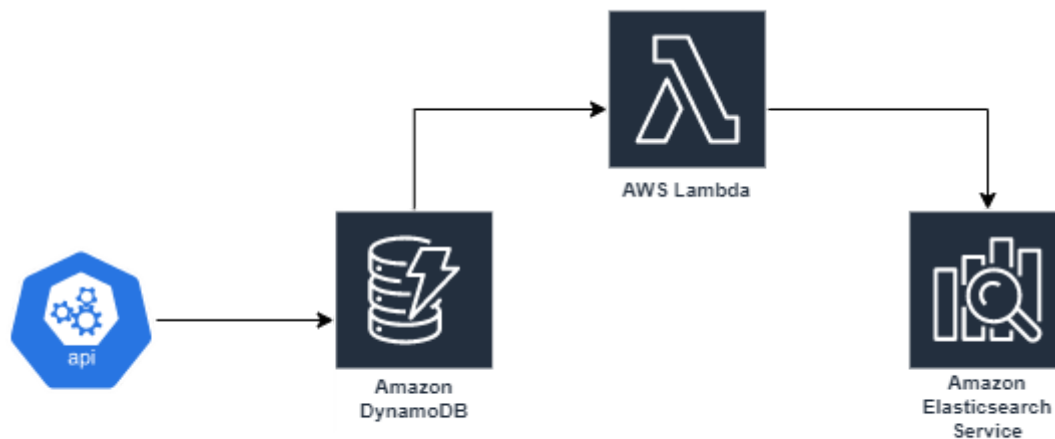
These features help protect your data. After creating the domain, you can't change most encryption settings.

- Encryption**
- ☐ Require HTTPS for all traffic to the domain ⓘ
- ☐ Node-to-node encryption ⓘ
- ☐ Enable encryption of data at rest ⓘ

Now that is it for creating our testing elasticsearch domain. After a couple of minutes our elasticsearch domain will be up and running. So creating our Elasticsearch service is done now.



Creating a DynamoDB table and DynamoDB Streams



In our post as mentioned earlier we are going to use DynamoDB data streams for our elasticsearch domain to do analytics. To send our dynamoDB table data as a stream to the elasticsearch we are going to use and AWS Lambda function. So when a new data

item arrives in the database table triggers an event notification to Lambda, which then runs our custom code to perform the indexing and adding item in the elasticsearch.

Let's first create our DynamoDB table. Go to the DynamoDB dashboard and create a new table. Make sure to create the dynamoDB table in the **same region** where our elasticsearch domain is also created.

Create DynamoDB table

[Tutorial](#) [?](#)

DynamoDB is a schema-less database that only requires a table name and primary key. The table's primary key is made up of one or two attributes that uniquely identify items, partition the data, and sort data within each partition.

Table name* ⓘ

Primary key* Partition key

ⓘ

☐ Add sort key

Table settings

Default settings provide the fastest way to get started with your table. You can modify these default settings now or after your table has been created.

☒ Use default settings

- No secondary indexes.
- Provisioned capacity set to 5 reads and 5 writes.
- Basic alarms with 80% upper threshold using SNS topic "dynamodb".
- Encryption at Rest with DEFAULT encryption type.

ⓘ You do not have the required role to enable Auto Scaling by default.
Please refer to [documentation](#).

+ Add tags **NEW!**

Additional charges may apply if you exceed the AWS Free Tier levels for CloudWatch or Simple Notification Service. Advanced alarm settings are available in the CloudWatch management console.

In order to stream our dynamoDB data to elasticsearch we need to enable stream on our dynamoDB table. To do that go to the overview tab our table and click on Manage Stream. There make sure to select **View type** as New image and Enable.

Manage Stream



- View type**
- ☐ Keys only - only the key attributes of the modified item
 - ☒ New image - the entire item, as it appears after it was modified
 - ☐ Old image - the entire item, as it appeared before it was modified
 - ☐ New and old images - both the new and the old images of the item

Cancel

Enable

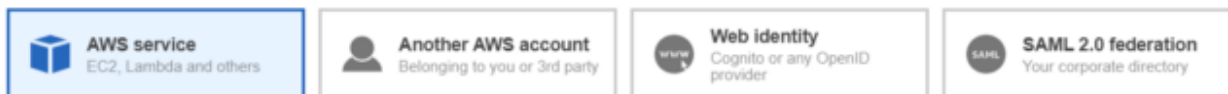
Creating a Lambda Function

Before creating our Lambda function we need to create an IAM role that will be assigned to our lambda function. To create a new IAM role go to the IAM dashboard and click on Create Role. After that for the type select Lambda.

Create role

1 2 3 4

Select type of trusted entity



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

Choose a use case

Common use cases

EC2

Allows EC2 instances to call AWS services on your behalf.

Lambda

Allows Lambda functions to call AWS services on your behalf.

Next we will be asked for what kind of permissions that we are going to assign for this role. Click on create policy to create a custom policy for our role which will redirect you to a policy creation page. There select JSON and paste below permissions.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "es:ESHttpPost",
        "es:ESHttpPut",
        "dynamodb:DescribeStream",
        "dynamodb:GetRecords",
        "dynamodb:GetShardIterator",
        "dynamodb:ListStreams",
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Resource": "*"
    }
  ]
}
```

```

    }
  ]
}

```

The above policy will give permission for Elasticsearch POST and PUT requests, get and process DynamoDB streams and to create log streams on AWS Cloudwatch.

Create policy

1 2

A policy defines the AWS permissions that you can assign to a user, group, or role. You can create and edit a policy in the visual editor and using JSON. [Learn more](#)

Visual editor

JSON

[Import managed policy](#)

```

1 {
2   "Version": "2012-10-17",
3   "Statement": [
4     {
5       "Effect": "Allow",
6       "Action": [
7         "es:ESHttpPost",
8         "es:ESHttpPut",
9         "dynamodb:DescribeStream",
10        "dynamodb:GetRecords",
11        "dynamodb:GetShardIterator",
12        "dynamodb:ListStreams",
13        "logs:CreateLogGroup",
14        "logs:CreateLogStream",
15        "logs:PutLogEvents"
16      ],
17      "Resource": "*"
18    }
19  ]
20 }

```

Character count: 278 of 6,144.

[Cancel](#)

[Review policy](#)

After that we will be asked to give a name to the policy. Next we can click on create policy. Now let's go back to the previous page for the new IAM role where we were asked for the policy. Make sure to refresh the policy table and select the policy we created.

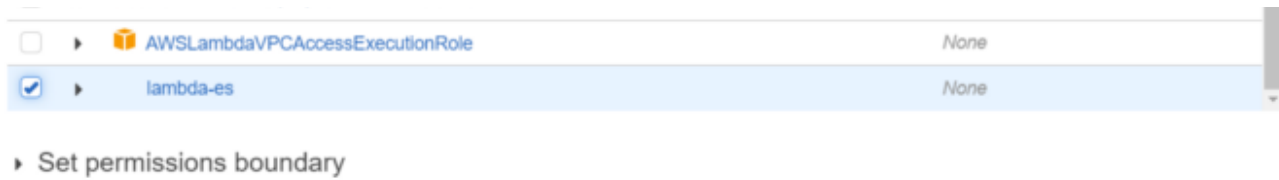
Attach permissions policies

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

[Create policy](#)



Filter policies <input type="text" value="Q lambda"/>		Showing 18 results
	Policy name	Used as
<input type="checkbox"/>	AWSLambdaInvocation-DynamoDB	None
<input type="checkbox"/>	AWSLambdaKinesisExecutionRole	None
<input type="checkbox"/>	AWSLambdaReadOnlyAccess	None
<input type="checkbox"/>	AWSLambdaReplicator	None
<input type="checkbox"/>	AWSLambdaRole	None
<input type="checkbox"/>	AWSLambdaSQSQueueExecutionRole	None



Finally after that we can add tags, give a name to our role, and create our role for our lambda function.

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

Allows Lambda functions to call AWS services on your behalf.

Maximum 1000 characters. Use alphanumeric and '+', '@', '-' characters.

Trusted entities

AWS service: lambda.amazonaws.com

Policies

lambda-es

Permissions boundary

Permissions boundary is not set

No tags were added.

We have one more task for creating a role, which is to add a trust relationship for our role. For that go to the trust relationships tab and add the following trust relationship JSON if it is not already there.

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

```
}  
]
```

Now in our lambda code we are going to do PUT requests. For that I am going to use **axios** to make HTTP requests. There are two ways in AWS Lambda where we can get external dependencies. The first is to create our code locally with dependencies and zip the file, then upload it to AWS Lambda. The second option is to create an AWS Layer in lambda and use that layer on our function. I prefer layers because then we can reuse the same dependency in many lambda functions.

Creating a lambda layer

Create a new folder in your system named **nodejs**. Make sure to use the name as **nodejs** as this is a requirement when adding layers. Inside that nodejs folder issue following commands.

```
npm init  
npm i axios
```

This will initialize a package.json file and install OUR axios dependency. Now there should be a folder named node_modules in our nodejs folder. Now let's zip our nodejs folder and go to the AWS Lambda dashboard and click on Layers. There click on add new layer.

Create layer

Layer configuration

Name

Description - optional

☒ Upload a .zip file

☐ Upload a file from Amazon S3



Upload

node.js.zip (136.7 kB)

For files larger than 10 MB, consider uploading using Amazon S3.

Compatible runtimes - optional [Info](#)

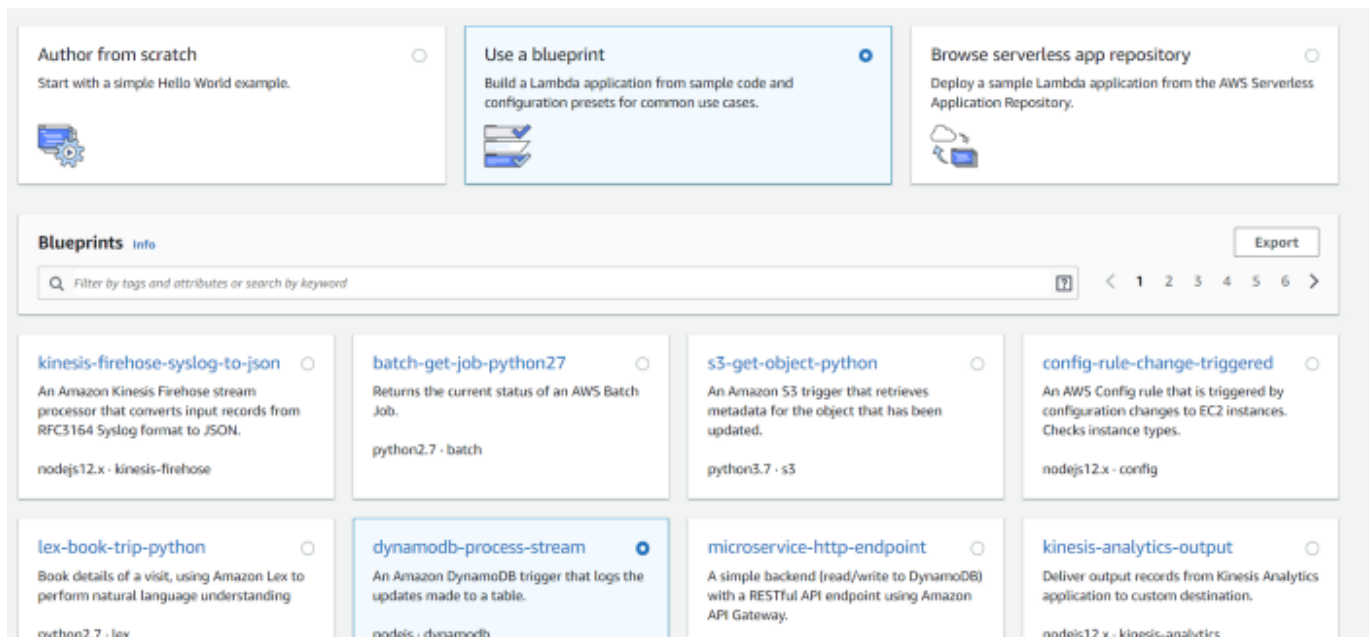
Choose up to 5 runtimes.

Runtimes

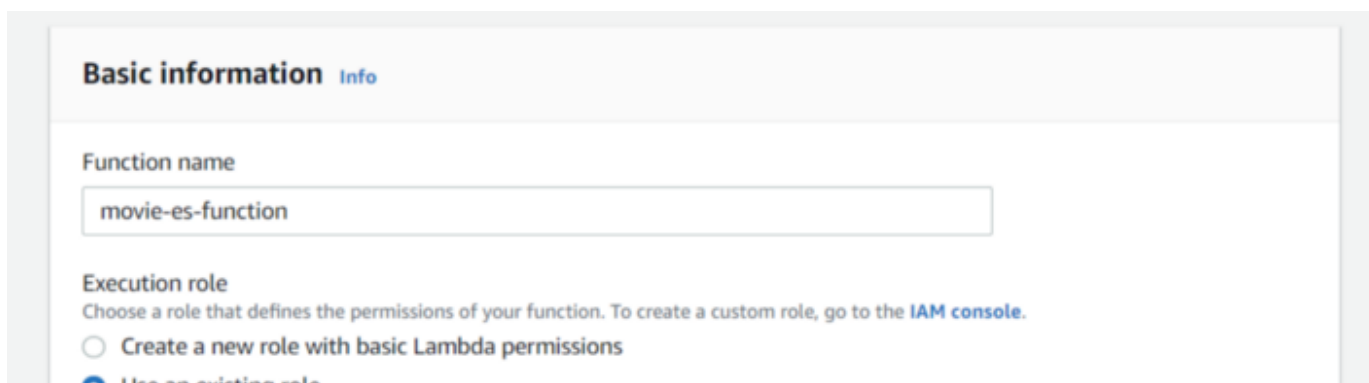
Node.js 12.x X

Here give a name for our layer and upload the zip file we created. After that select the runtime as Node.js 12.x. That is it, now our layer is available for use in our lambda functions.

Next let's create our Lambda function. Go to create a new function and select **Use a blueprint**. There select **dynamodb-process-stream** as our blueprint.



After that give our function a name and select our already created IAM role for this function as execution role.



☒ 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.

lambda-es-role ▼ ↻

[View the lambda-es-role role on the IAM console.](#)

Next for dynamodb trigger select our table. Make starting position as **trim horizon**.

DynamoDB trigger Remove

DynamoDB table
Choose or enter the ARN of a DynamoDB table.

arn:aws:dynamodb:ap-southeast-1:628640267234:table/movie-db X ↻

Batch size
The largest number of records that will be read from your table's update 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 DynamoDB Streams API Reference.

Latest ▼

Now our function is generated. The first thing is to do is add the layer to our function we added earlier. Click on Layers and then click on add layer.

Add layer to function

Layer selection
Choose from layers that are compatible with your function's runtime, or specify the Amazon Resource Name (ARN) of a layer version.

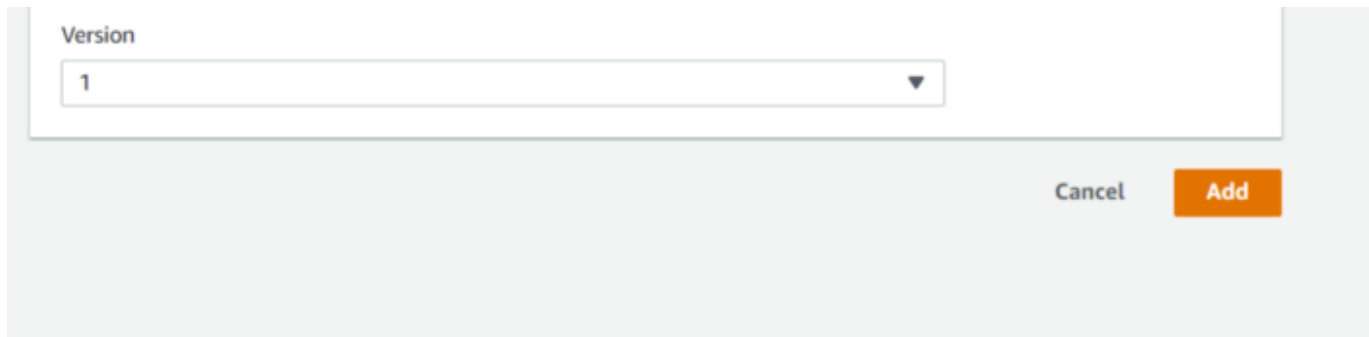
☒ Select from list of runtime compatible layers

☐ Provide a layer version ARN

Compatible layers

Name

axios-layer ▼



Ok, now our runtime node.js environment will have axios dependency already added. Next let's create our function.

```
const axios = require('axios');

const host = '{Our-ES-Domain-Name}';
const index = 'movies';
const type = 'movie';

const url = `${host}/${index}/${type}/`
const headers = { "Content-Type": "application/json" }

exports.handler = async (event, context) => {
  let count = 0;
  for (const record of event.Records) {
    const id = record.dynamodb.Keys.id.N;

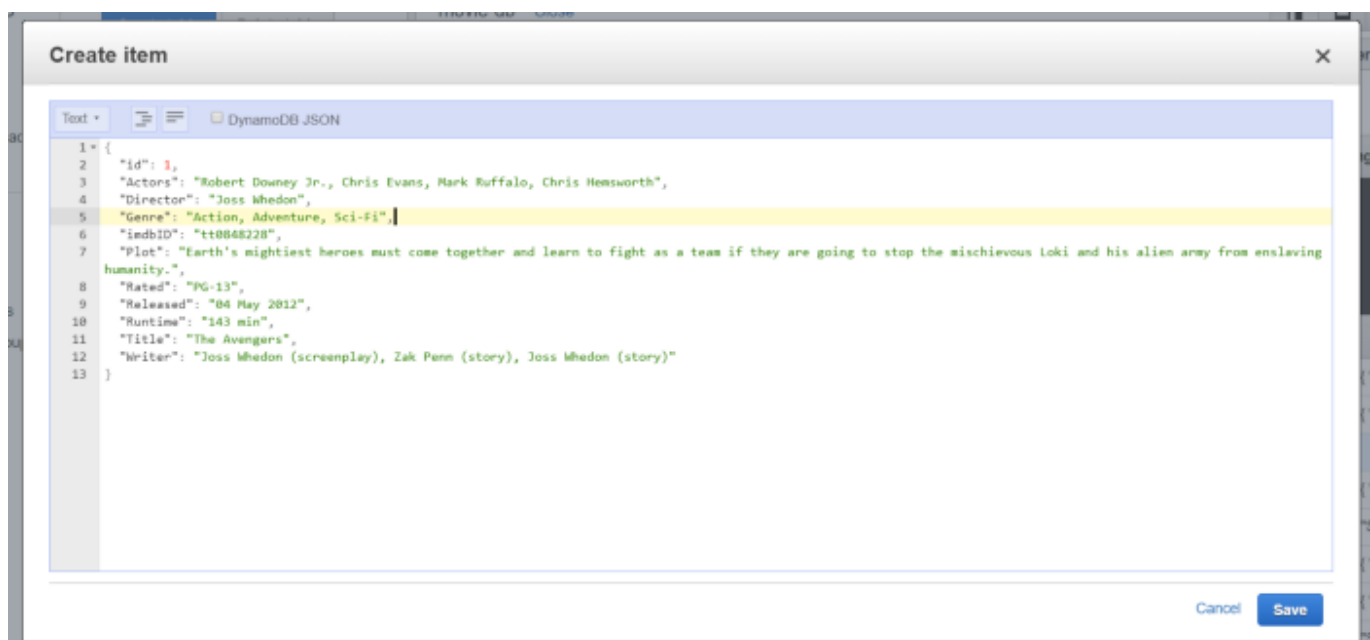
    if (record.eventName == 'REMOVE') {
      await axios.delete(url + id);
      return 'Item removed'
    }
    else {
      const document = record.dynamodb.NewImage;
      console.log('Adding document');
      console.log(document)
      await axios.put(url + id, document)
    }
    count += 1;
  }
  return `Successfully processed ${count} records.`;
};
```

In the above function dynamoDB stream gives us the stream event where it will provide us with a list of stream records. We will go through all of these records and check for the event name. If the event name is **REMOVE** which means that item is removed from the dynamoDB table we will remove that item from our elasticsearch as well. Else we will

add this item to our elasticsearch. Note that we have created our index name as **movies** and type as **movie**. More about on what is the index and what is the type will be discussed on the next post where I will talk about more on Elasticsearch functionalities.

Now our lambda function is ready to deploy. Make sure to save the function and go to dynamoDB table to test our whole streaming process. Create a new item in the table and add the following response which I got from OMDB API.

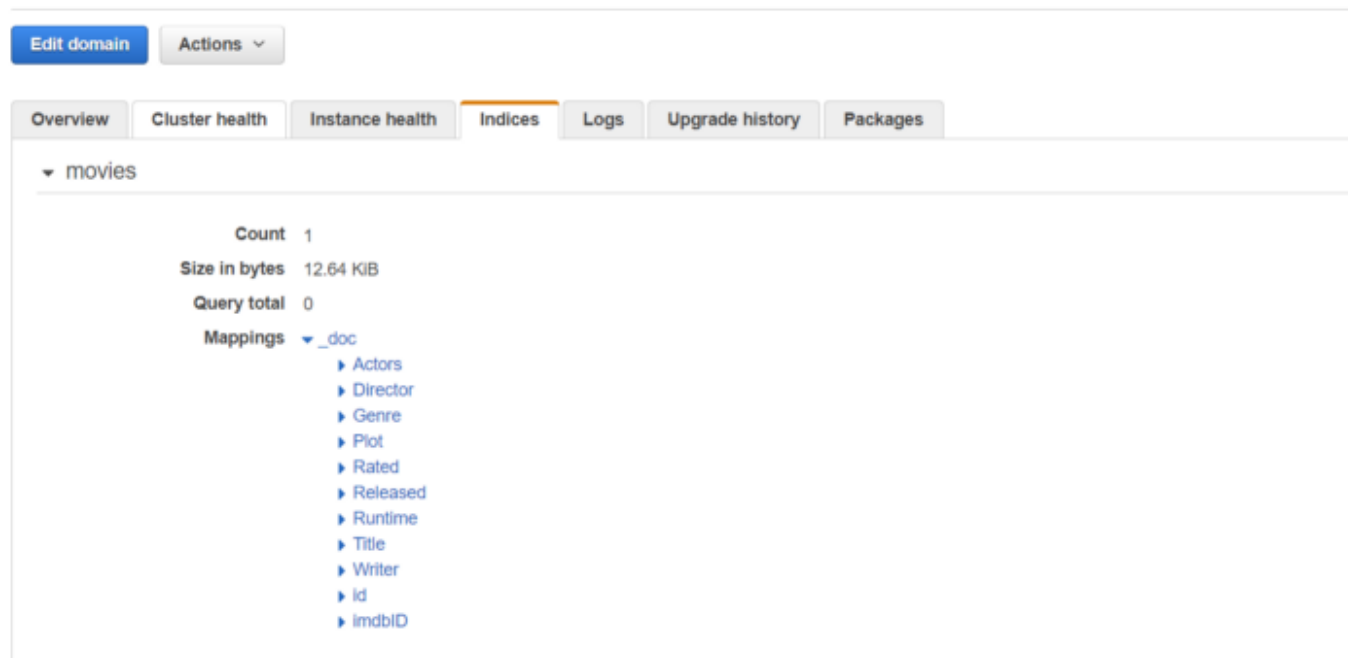
```
{
  "id": 1,
  "Actors": "Robert Downey Jr., Chris Evans, Mark Ruffalo, Chris Hemsworth",
  "Director": "Joss Whedon",
  "Genre": "Action, Adventure, Sci-Fi",
  "imdbID": "tt0848228",
  "Plot": "Earth's mightiest heroes must come together and learn to fight as a team if they are going to stop the mischievous Loki and his alien army from enslaving humanity.",
  "Rated": "PG-13",
  "Released": "04 May 2012",
  "Runtime": "143 min",
  "Title": "The Avengers",
  "Writer": "Joss Whedon (screenplay), Zak Penn (story), Joss Whedon (story)"
}
```



Now when we add the item to our table, the lambda function should be triggered and should add this record to our elasticsearch. We can confirm whether our item is added to the elasticsearch in several ways.

The first way is to go to our elasticsearch dashboard and go to the Indices tab. In there now new indices should be created as movies and mappings as out item properties.

movie-es



Or else we can query our elasticsearch instance by issuing below GET request.

https://{YOUR_ES_DOMAIN_NAME}/movies/movie/1

This should give the inserted document as the response. That is it for loading data to AWS Elasticsearch service using DynamoDB streams and AWS Lambda. In the next post let's talk more about Elasticsearch functionalities and a complete implementation of a web service using Node.js which will use our Elasticsearch service. Thank you.

AWS Elasticsearch Dynamodb Lambda Analytics

Get the Medium app

