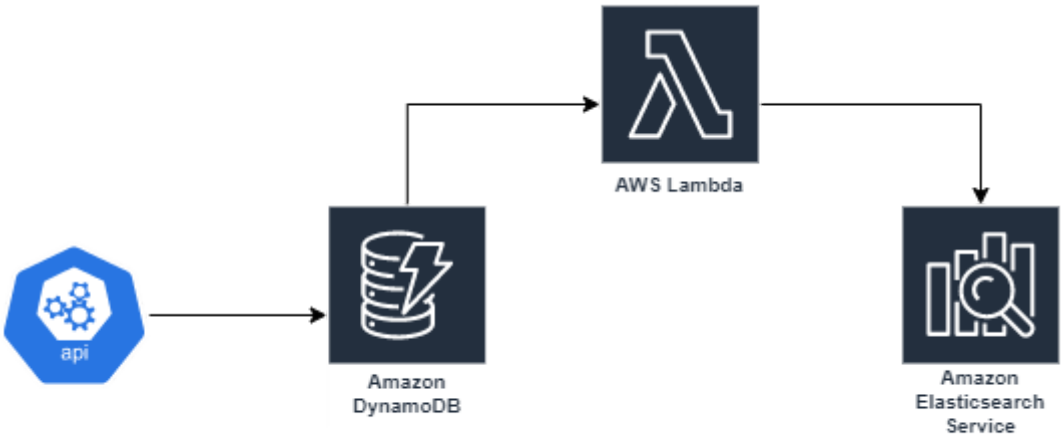
Loading Data to AWS Elasticsearch with DynamoDB Streams and Lambda

Janitha Tennakoon Follow
May 24 · 9 min read ★

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 **Logstach**, and other AWS services. In this post I am going to talk about how we can load streaming data into AWS Elasticsearch

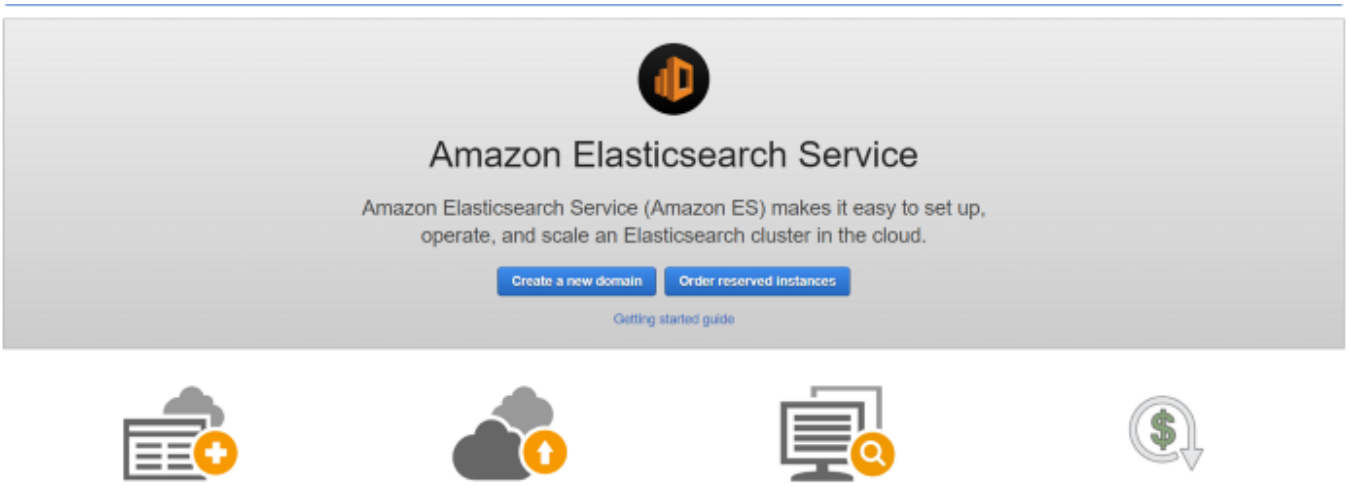
service using AWD DynamoDB streams and AWS Lambda. So in our scenario when a web application or a web service adds data to our dynamoDB database, it will trigger an AWS Lambda function which will automatically insert those data to our Elasticsearch service which we can use in our use case.



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.

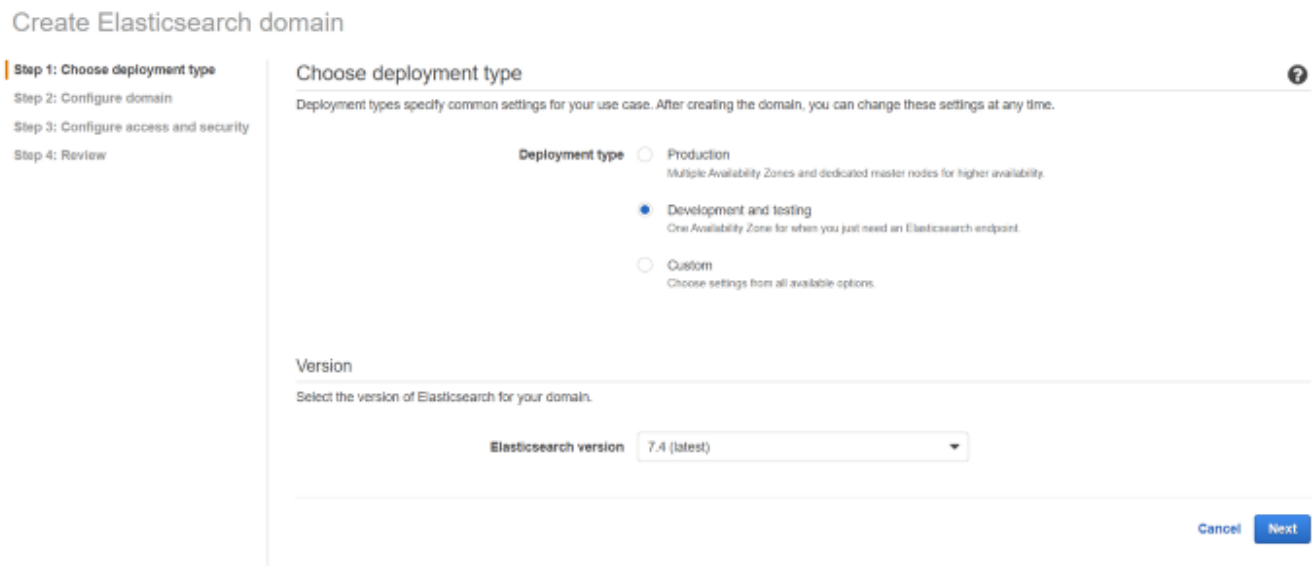
Creating an AWS Elasticsearch Instance

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.

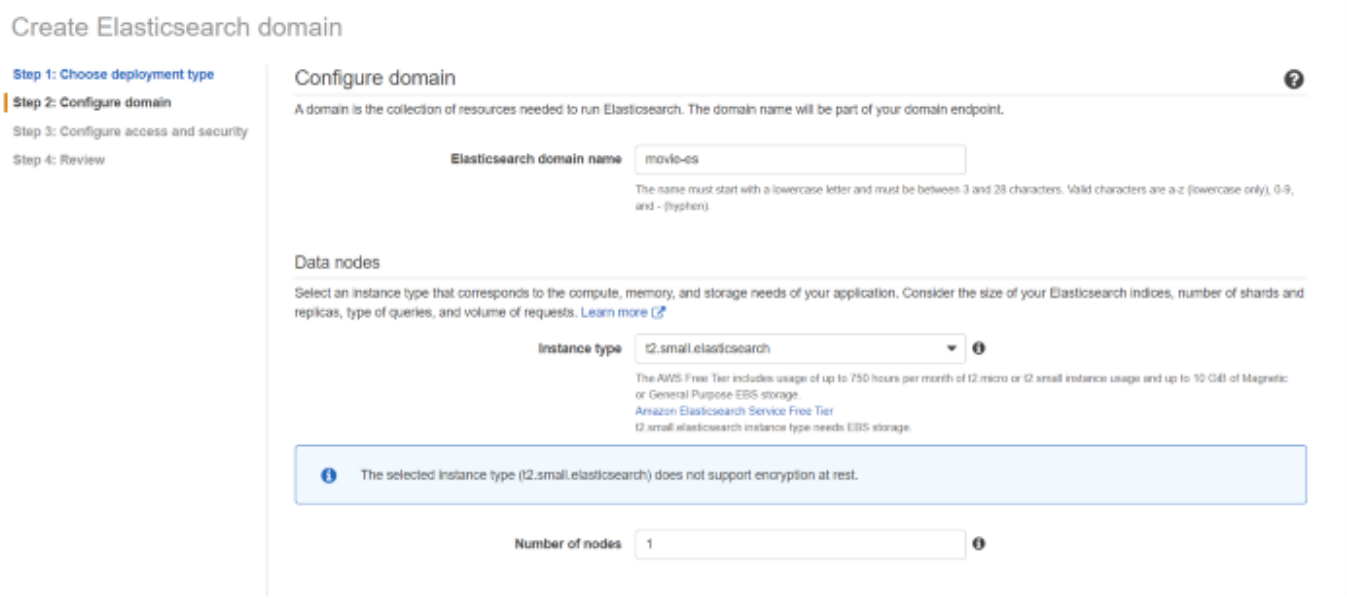


Launch an Elasticsearch cluster Manage and monitor Load and query data Order reserved instances

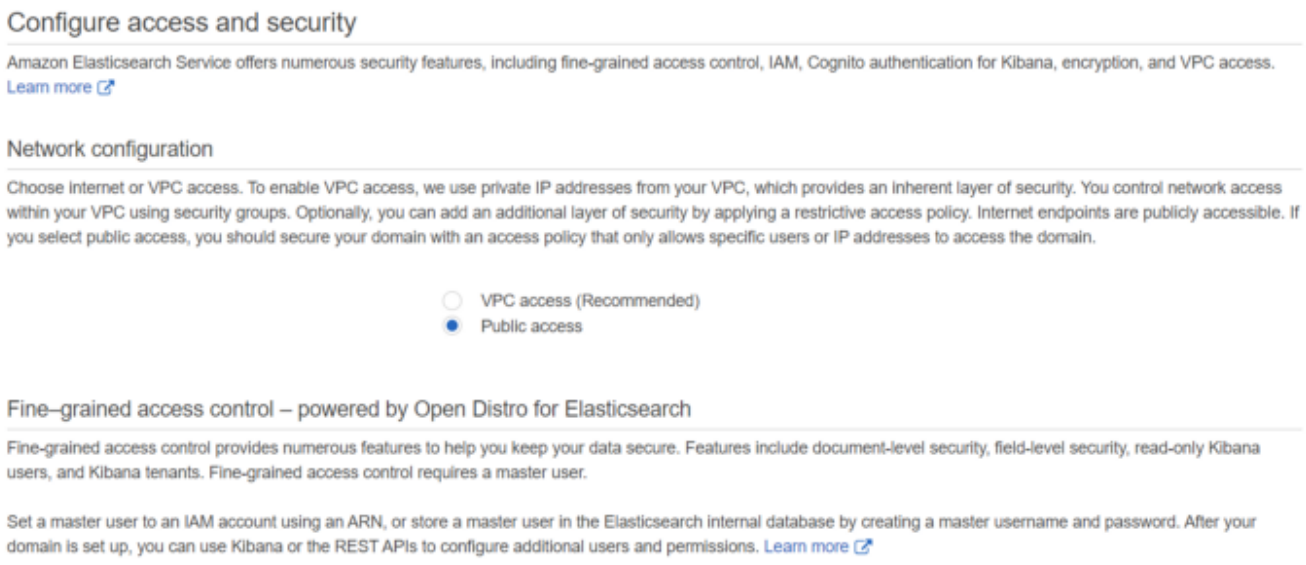
We are going to create our domain only for learning purpose so the configurations that we will select for our domain is only applicable for learning purposes. For production you must follow the correct configuration setup. Since we are using this domain for only learning select **Deployment type** as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



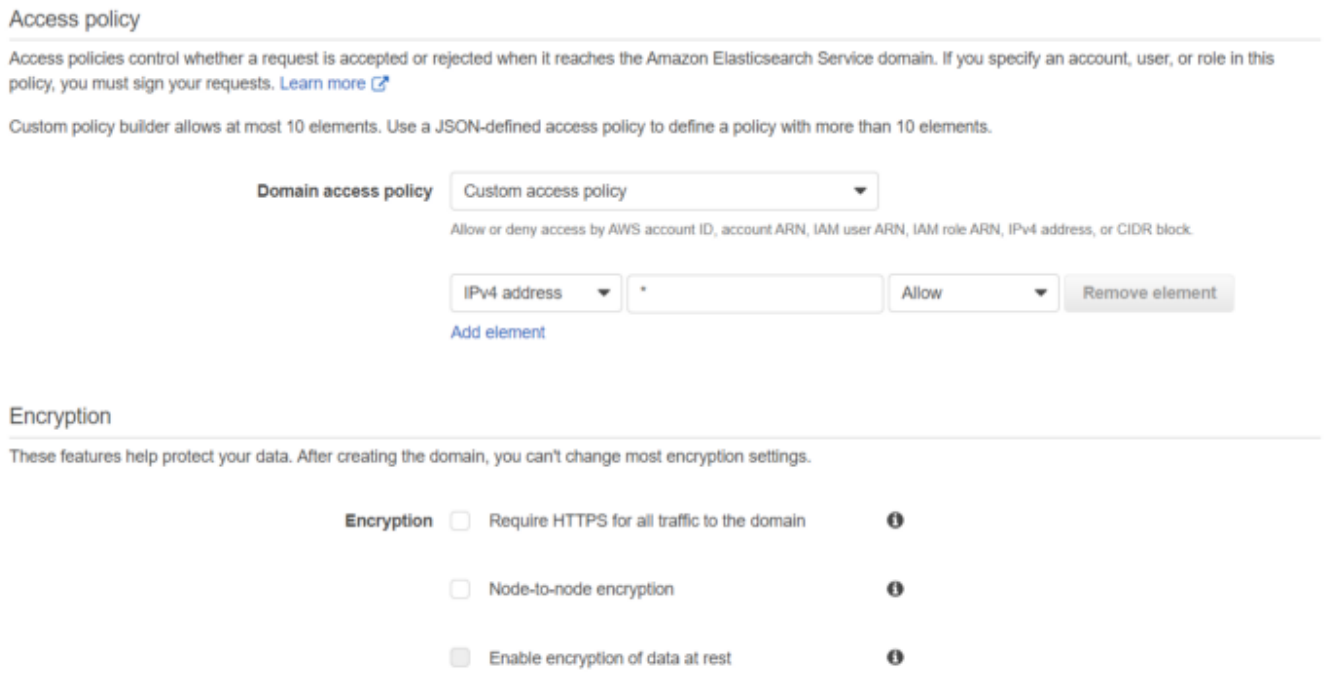
On the next screen we need to provide a domain name. After that select the **instance type** as t2.small.elasticsearch because that is the only instance type that is available for the free tier. You can leave all the other options as default values without any change.



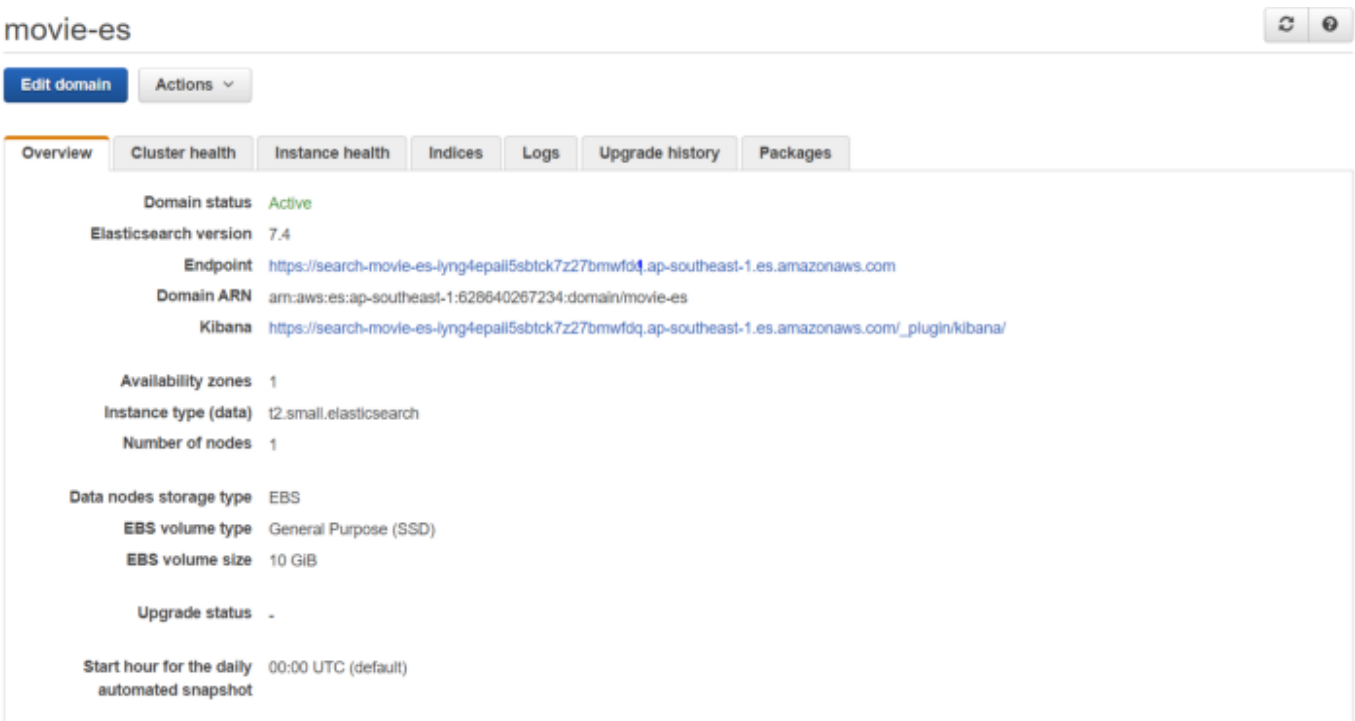
On the next screen we will be asked to configure security for our elasticsearch domain. The recommended is to use VPC access where our domain will be in a private network with only instances within our VPC network that will have access. But in our case make it as Public access.



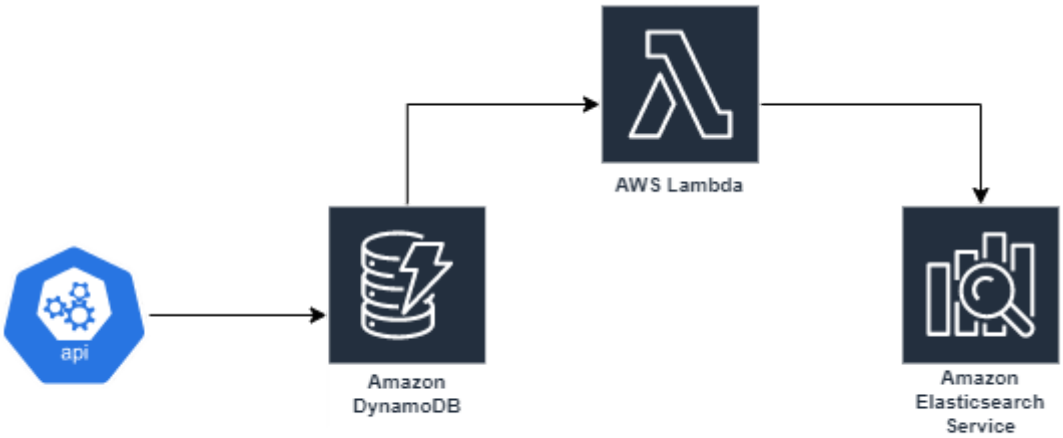
Below on the same page you will see where we are setting access policy for our domain. Since this is only for learning add **Allow open access to the domain** which will make any IP address have access. Here we can specify either an IAM role or specific AWS user accounts as well.



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.



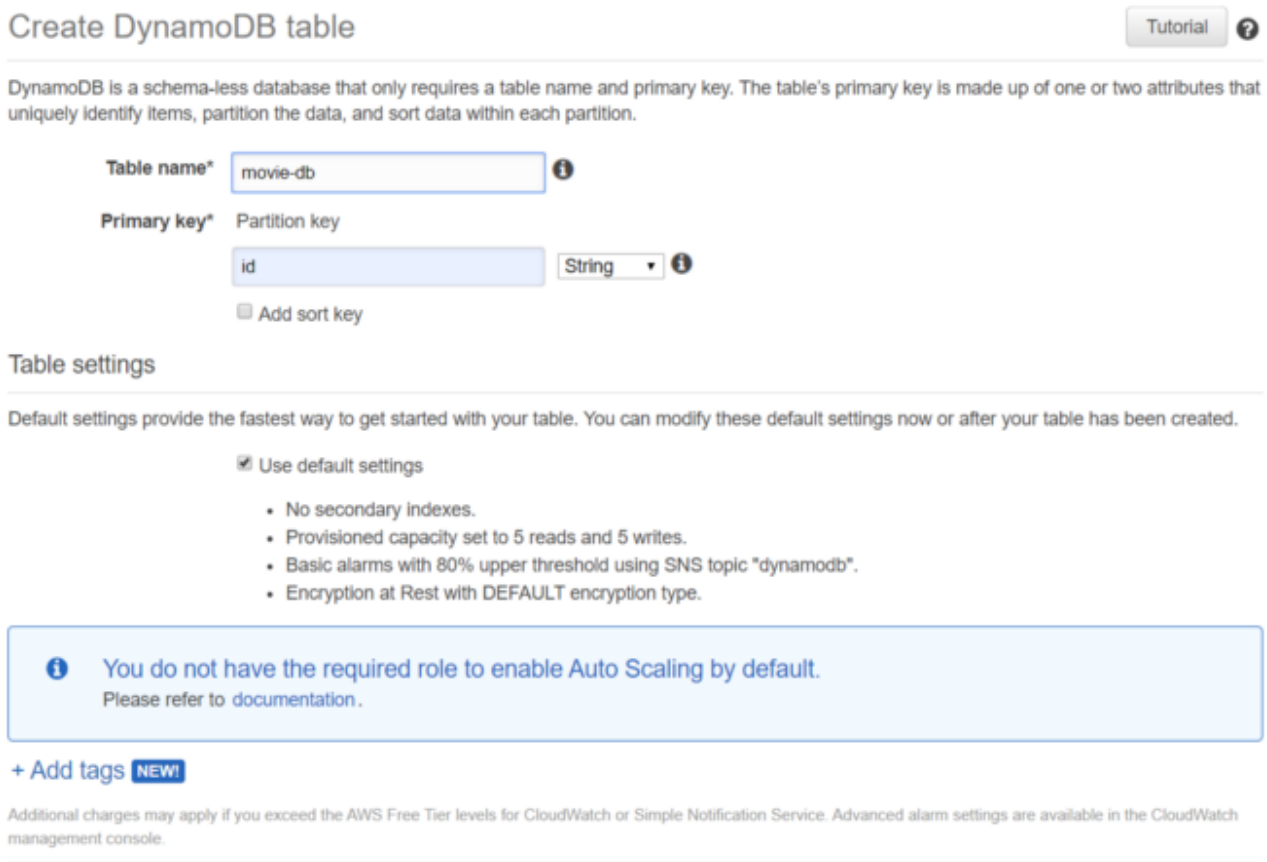
Creating a DynamoDB table and DynamoDB Streams



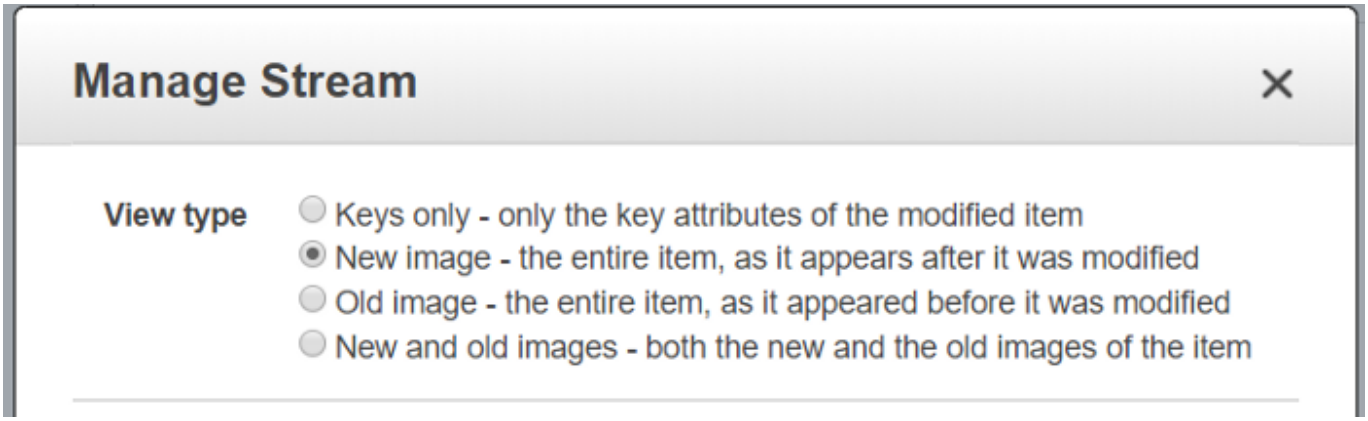
In our post as mentioned earlier we are going to use DynamoDB data streams for our elasticsearch domain to do analytics. To send our dynamoDB table data as a stream to the elasticsearch we are going to use and AWS Lambda function. So when a new data

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.

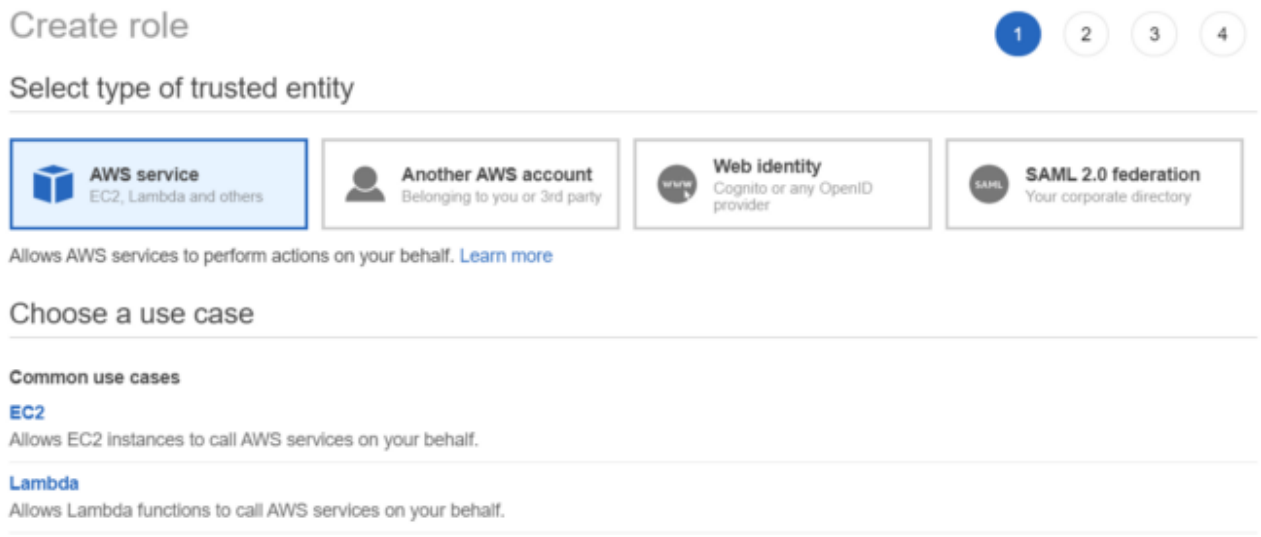


In order to stream our dynamoDB data to elasticsearch we need to enable stream on our dynamoDB table. To do that go to the overview tab our table and click on Manage Stream. There make sure to select **View type** as New image and Enable.



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.

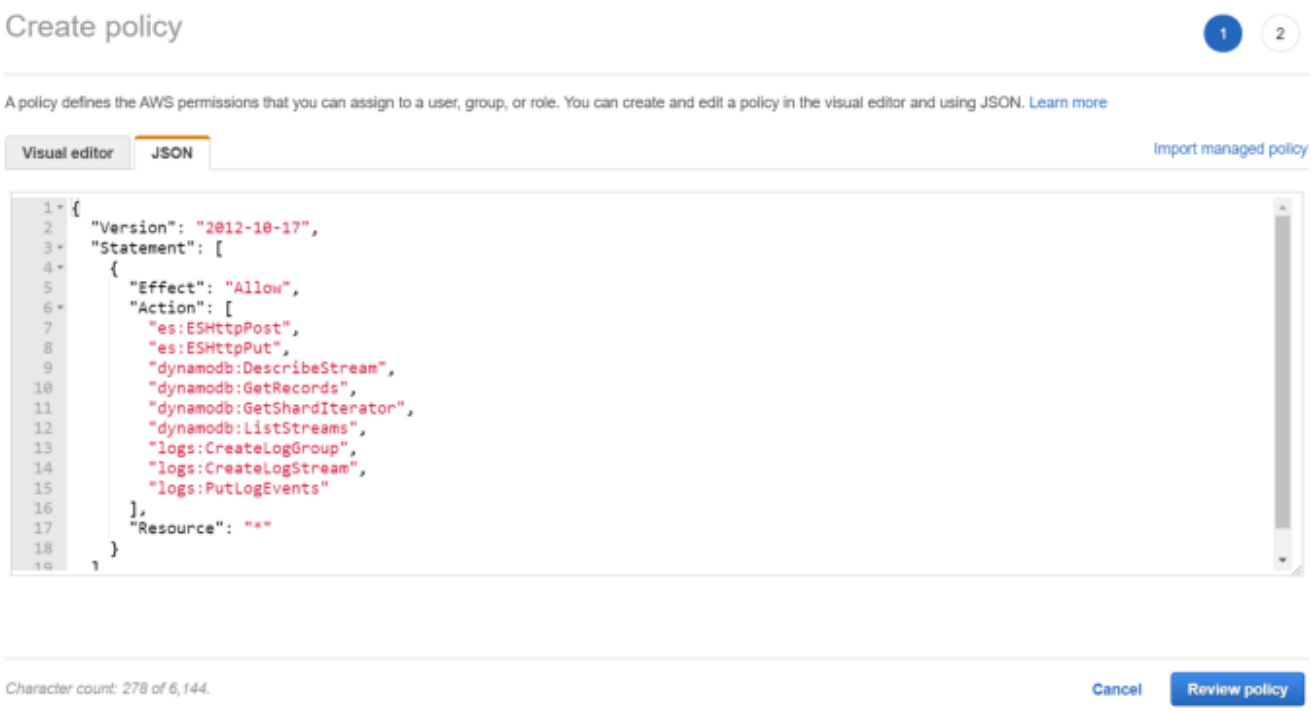


Next we will be asked for what kind of permissions that we are going to assign for this role. Click on create policy to create a custom policy for our role which will redirect you to a policy creation page. There select JSON and paste below permissions.

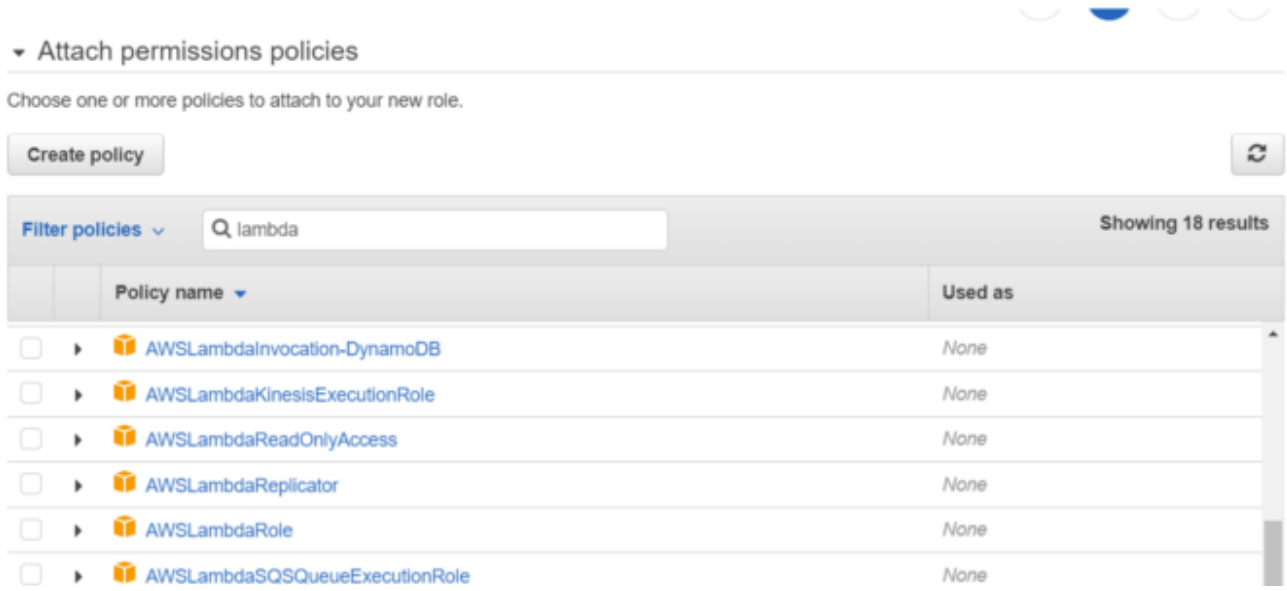
```
{
  "Version": "2012-10-17",
  "Statement": [
     {
        "Effect": "Allow",
        "Action": [
           "es:ESHttpPost",
           "es:ESHttpPut",
           "dynamodb:DescribeStream",
           "dynamodb:GetRecords",
           "dynamodb:GetShardIterator",
           "dynamodb:ListStreams",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

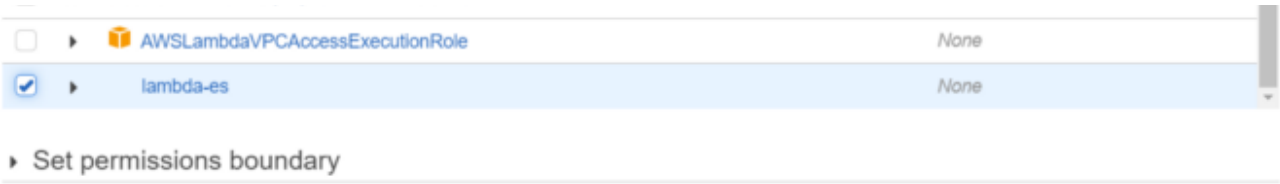
```
        }
    ]
}
```

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.



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.





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

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"
      }
```

```
    ]
}
```

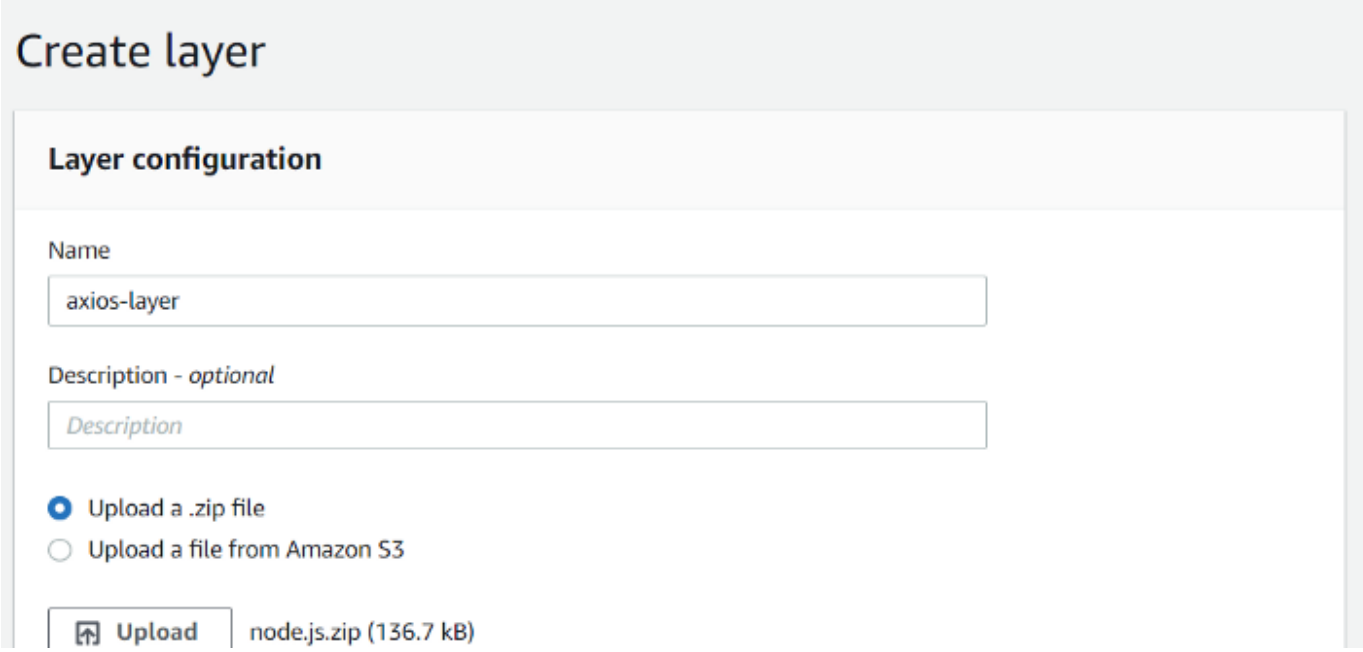
Now in our lambda code we are going to do PUT requests. For that I am going to use **axios** to make HTTP requests. There are two ways in AWS Lambda where we can get external dependencies. The first is to create out code locally with dependencies and zip the file, then upload it to AWS Lambda. The second 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.

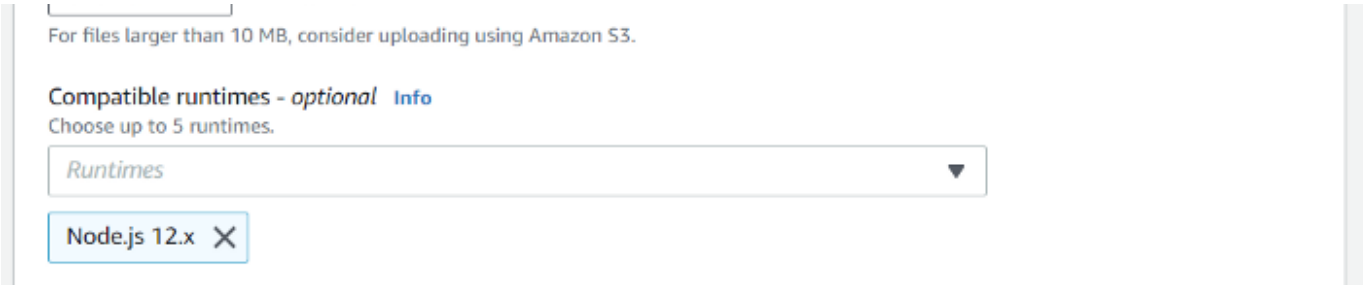
Creating a lambda layer

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
```

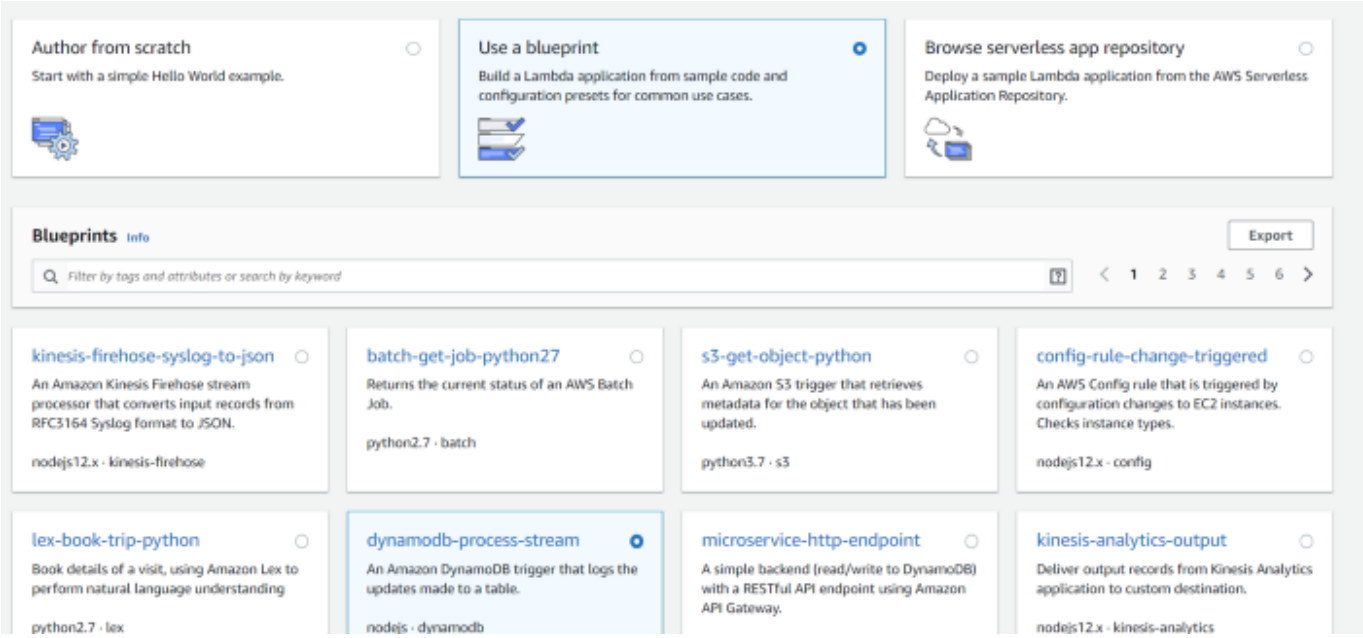
This will initialize a package.json file and install OUR axios dependency. Now there should be a folder named node_modules in our nodejs folder. Now let's zip our nodejs folder and go to the AWS Lambda dashboard and click on Layers. There click on add new layer.



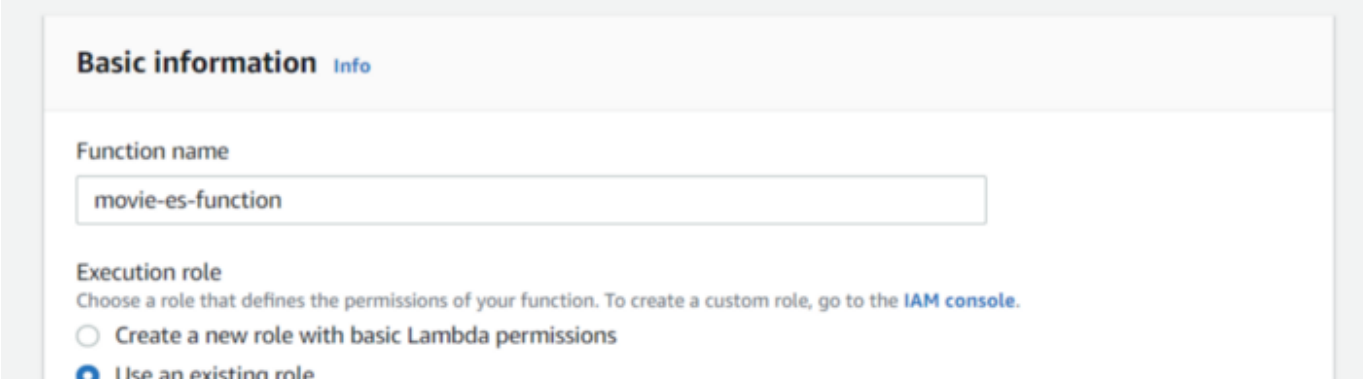


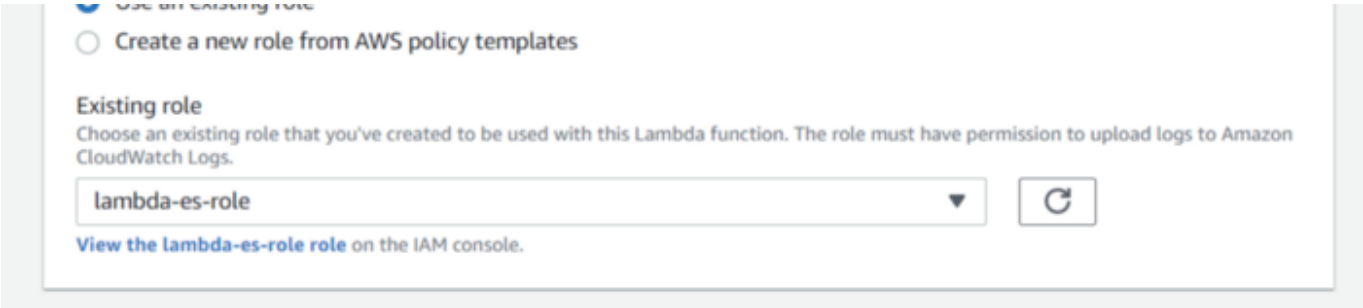
Here give a name for our layer and upload the zip file we created. After that select the runtime as Node.js 12.x. That is it, now our layer is available for use in our lambda functions.

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.

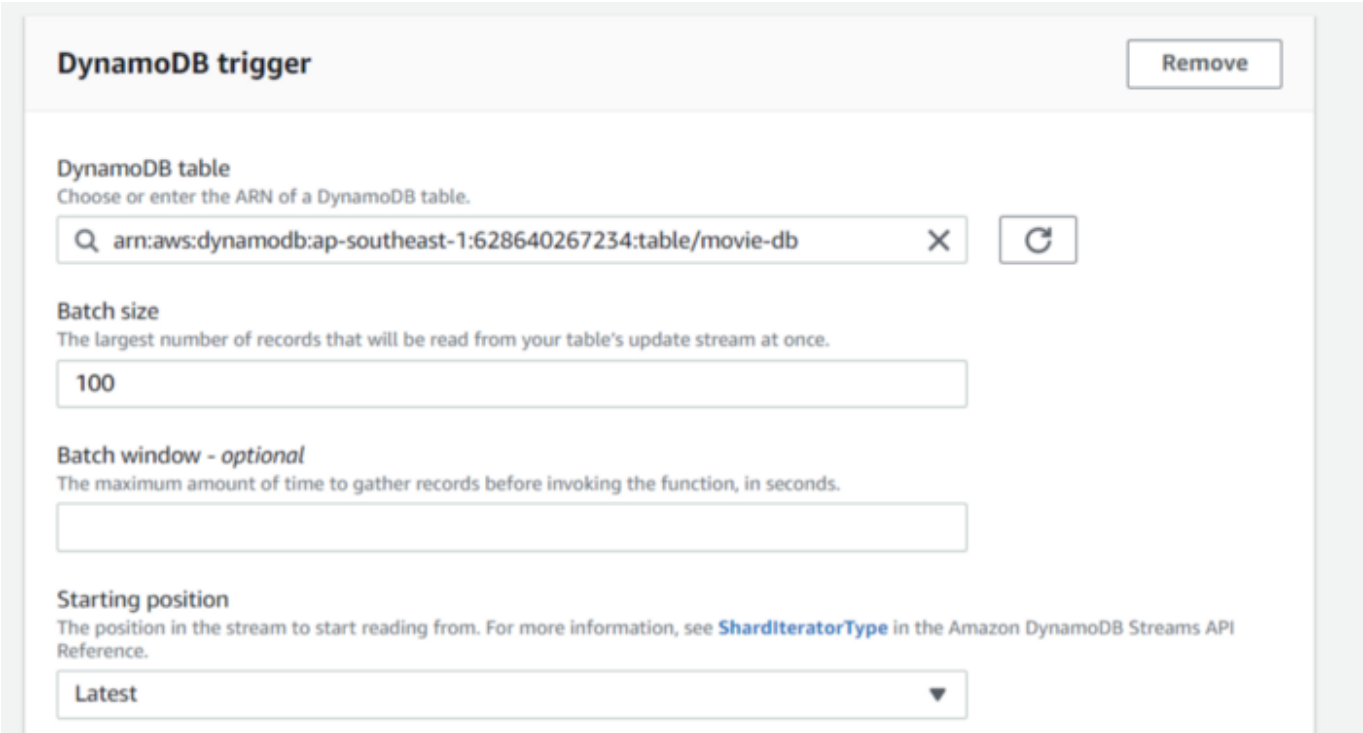


After that give ou function a name and select our already created IAM role for this function as execution role.

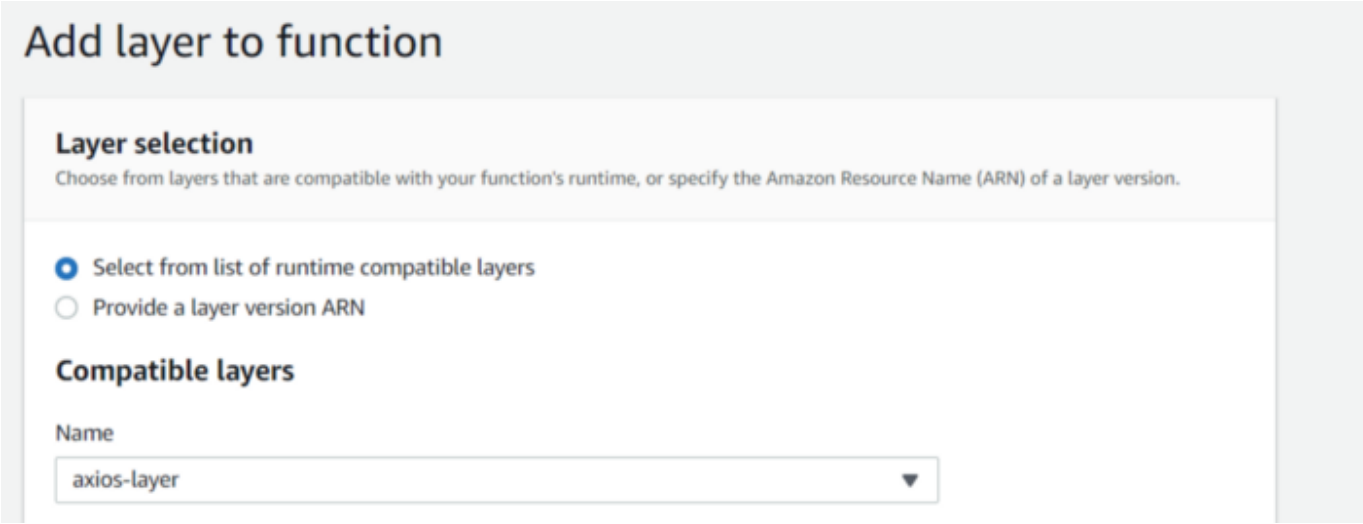


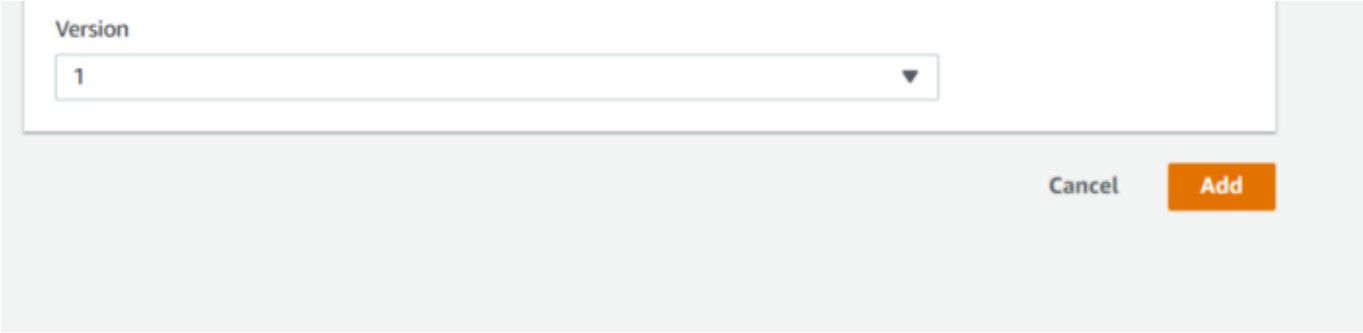


Next for dynamodb trigger select our table. Make starting position as **trim horizon.**



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.





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

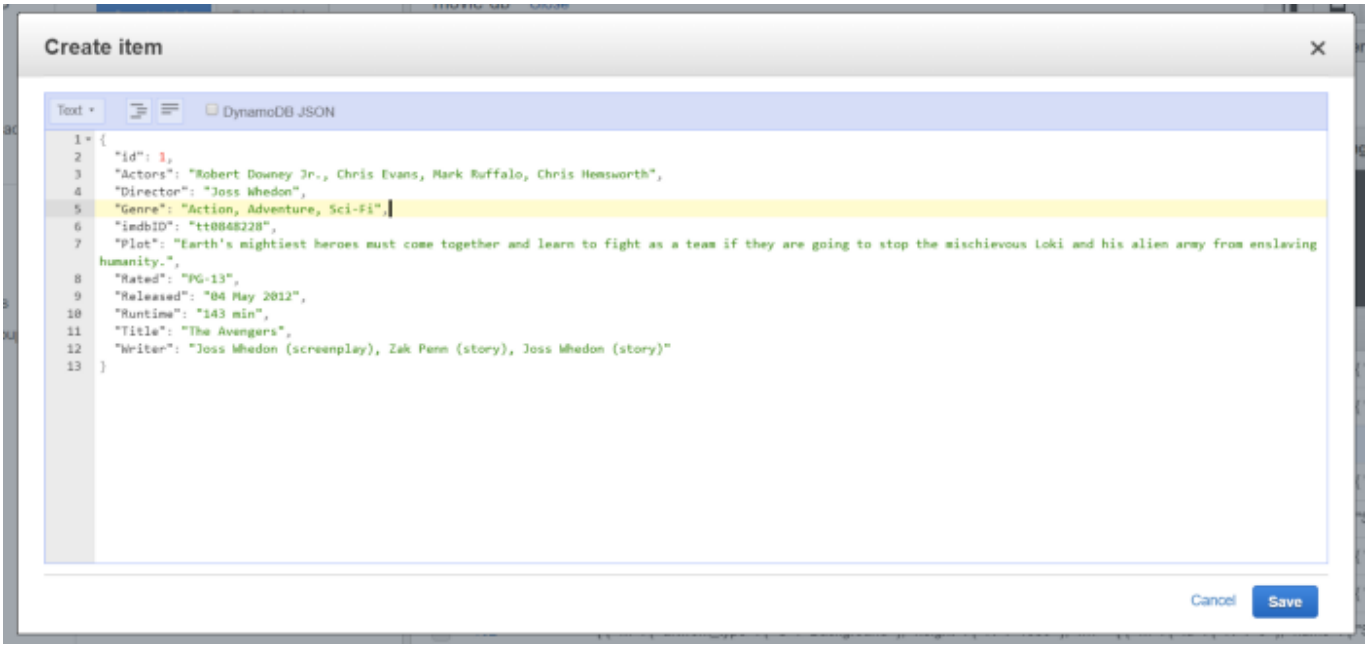
```
const axios = require('axios');

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.

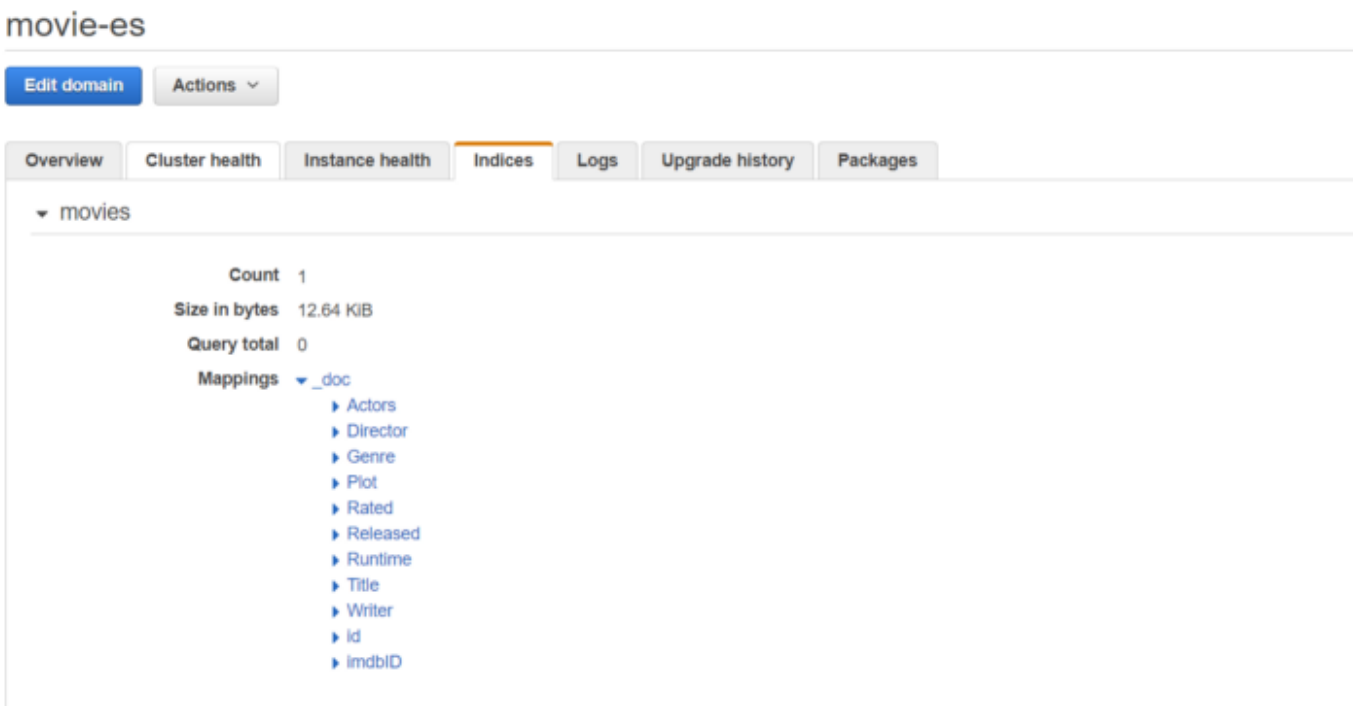
Now our lambda function is ready to deploy. Make sure to save the function and go to dynamoDB table to test our whole streaming process. Create a new item in the table and add the following response which I got from OMDB API.

```
{
  "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.



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. n 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