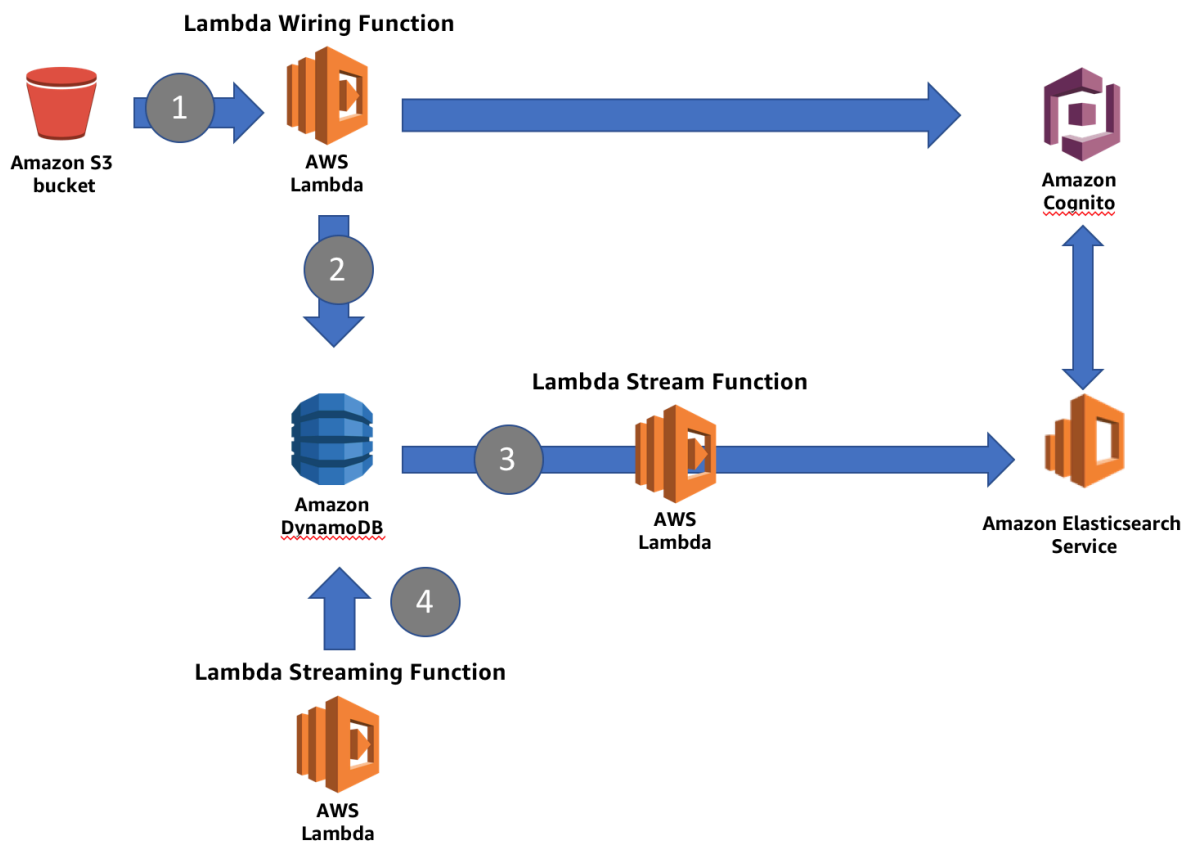


Lab: Search Dynamo DB Data with Amazon Elasticsearch Service

In this lab, you'll explore using Amazon Elasticsearch Service (Amazon ES) to augment the more fundamental query capabilities of Amazon Dynamo DB (Dynamo DB). You'll deploy a CloudFormation template that generates a Dynamo DB table and loads movie data into that table. You'll employ Dynamo Streams to replicate the initial data load to an Amazon Elasticsearch Service domain. You'll enable Cognito authentication to provide simple access to Kibana so that you can use Kibana's UI to send searches to Amazon ES. Finally, you'll run a Lambda function to generate updates to the Dynamo DB table, see those updates replicated to Amazon ES, and build Kibana visualizations for the changes.

Lab overview



In the lab you will deploy a CloudFormation stack that will create resources for you in AWS. CloudFormation deploys a Dynamo DB table and an Amazon Elasticsearch Service domain. It deploys a Lambda function (Lambda Stream Function), triggered by Dynamo table's stream. It deploys a second Lambda function (Lambda Wiring Function) that reads the source data from S3 (#1) and loads it into the Dynamo table (#2). The Lambda stream function, which is already in place, catches the updates to the table and inserts the records into Amazon ES (#3). Finally,

you will run a third Lambda function (Lambda Streaming Function) to generate updates to the Dynamo table (#4), which the Lambda stream function will propagate to Amazon ES.

The source data is a set of 246, Sci-Fi/Action movies, sourced from IMDb. The Lambda wiring function reads this data, adds a clicks, price, purchases, and a location field, and creates the item in the Dynamo DB table.

The Dynamo schema uses the movie's id as the primary key and adds items for the movie data's fields.

```
{
  "id": "tt0088763",
  "title": "Back to the Future",
  "year": 1985,
  "rating": 8.5,
  "rank": 204,
  "genres": [
    "Adventure",
    "Comedy",
    "Sci-Fi"
  ],
  "plot": "A teenager is accidentally sent 30 years into the past in a time-traveling DeLorean invented by his friend, Dr. Emmett Brown, and must make sure his high-school-age parents unite in order to save his own existence.",
  "release_date": "1985-07-03T00:00:00Z",
  "running_time_secs": 6960,
  "actors": [
    "Christopher Lloyd",
    "Lea Thompson",
    "Michael J. Fox"
  ],
  "directors": [
    "Robert Zemeckis"
  ],
  "image_url": "http://ia.media-imdb.com/images/M/MV5BMTk4OTQ1OTMwN15BMl5BanBnXkFtZTCwOTIwMzM3MQ@@._v1_SX400_.jpg",
  "location": "44.54, -67.0 ",
  "purchases": 12460,
  "price": 29.99,
  "clicks": 1277594
}
```

These fields will also serve as the basis for querying the data in Elasticsearch.

Step 1: Launch the CloudFormation stack

- Using a web browser, login to the **AWS Console** at <https://aws.amazon.com/>
- Choose **CloudFormation**

- Choose the region you will use for this lab. The lab works in **US East (N. Virginia)** and **US West (Oregon)**
- Choose **Create Stack**
- In the resulting window, select **Specify an Amazon S3 template URL**. Use the following template, depending on your region
 - US East (N.Virginia): **<https://s3.amazonaws.com/imdb-ddb-aes-lab-east-1/Lab-Template.json>**
 - US West (Oregon): **<https://s3.amazonaws.com/imdb-ddb-aes-lab-west-2/Lab-Template.json>**
- Enter a **Stack Name**.
- You can leave the other template parameters at their defaults or choose your own.
- Click **Next**
- Leave all the defaults on this page. Scroll to the bottom of the page and click **Next**
- On the next page, click the checkbox next to **I acknowledge that AWS CloudFormation might create IAM resources with custom names**
- Click **Create**

The stack will take approximately 15 minutes to deploy.

Step 2: Enable Amazon Cognito Access

- In the console, select **Elasticsearch Service**
- Click the **<stack-name>-domain**
- Click **Configure Cluster**
- Scroll to the **Kibana Authentication** section and click the **Enable Amazon Cognito for Authentication** check box.
- In the **Cognito User Pool** drop down, select **ImdbDdbEsUserPool**
- In the **Cognito Identity Pool** drop down, select **ImdbDdbEsIdentityPool**
- Click **Submit**

Your domain will enter the **Processing** state while Amazon Elasticsearch Service enables Cognito access for your domain. Wait until the domain status is **Active** before going on to the next step. This should take about 10 minutes.

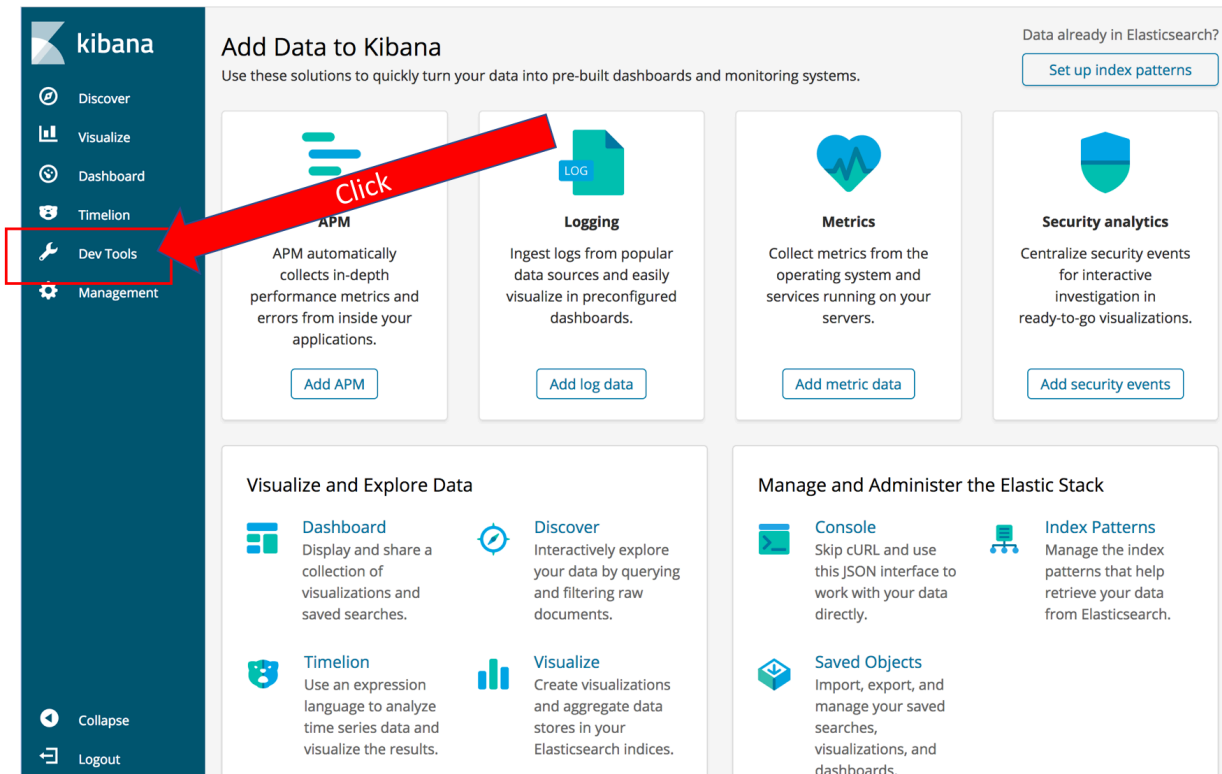
Step 3: Log in to Kibana


- Click the **Kibana** link from your Amazon ES domain's dashboard
- For **Username** enter **kibana**, for **Password**, enter **Abcd1234!**
- Click **Sign in**
- Enter a new password in the **Change Password** dialog box

You will see Kibana's welcome screen

Search the movie data

- In Kibana's main screen, select the **Dev Tools** tab



- Click **Get to work**
- The panel you now see lets you send requests directly to Amazon Elasticsearch Service. You enter the HTTP method, and the URL body. Kibana starts out with a helpful example. Click the  to execute the query and you should see Elasticsearch's response in the right half of the screen.
- For your first query, you will search the movie titles for Iron Man. Click after the closing bracket for the *match_all* query and hit **Enter** a couple of times to get some blank lines.
- Type "GET". Notice that Kibana provides a drop down with possible completions. Finish typing out (or copy-paste) the following query

```
GET movies/_search
{
  "query": {
    "match": {
      "title": "star wars"
    }
  },
  "_source": "title"
}
```

Let's dissect the results:

```
{
  "took": 23,
```

```

    "timed_out": false,
    "_shards": {
      "total": 1,
      "successful": 1,
      "skipped": 0,
      "failed": 0
    },
    "hits": {...

```

The top section contains metadata about the response. Elasticsearch tells you that it took 23 ms (server side) to complete the query. The query didn't time out. Finally it gives you a report on the engagement of the shards: 1 total shard responded, successfully.

The hits section contains the actual search results. In the query, you specified `_source: "title"` in your query, so Elasticsearch returns only the title for each of the hits. You can specify an array of fields you want retrieved for `_source` or leave it off to see the full, source records.

Search multiple fields

One of Elasticsearch's strengths is that it creates an index for the values of every field. You can use those indexes to construct complex queries across all of your data.

The *bool* query allows you to specify multiple clauses along with logic for combining results that match those clauses. You specify fields and values that must match in the *must* section. All *must* clauses must match for a document to match the query. These clauses represent a Boolean AND. You can also specify fields and values that *should* match – representing a Boolean OR. Finally you can specify fields and values that *must_not* match (Boolean NAND).

Enter the following query in the left pane of Kibana's Dev Tools pane:

```

GET movies/_search
{
  "query": {
    "bool": {
      "must": [
        { "term": { "actors.keyword": {
          "value": "Mark Hamill"
        }
        }
      ],
      "range": {
        "running_time_secs": {
          "gte": "6000"
        }
      },
      "range": {
        "release_date": {
          "gte": "1970-01-01",
          "lte": "1980-01-01"
        }
      }
    ]
  },
  "should": [
    { "range": {
      "rating": {
        "gte": 8.0
      }
    }
  ]
}

```

```

    }
  }
]
}
}
}

```

Spend some time experimenting with different fields and values. You can use Kibana's auto-complete feature to help you figure out different query types (the above query uses *term* and *range*, but there are many more, including *match*, *match_phrase*, and *span*).

Search based on geographic location

With Elasticsearch, you can use the numeric properties of your data natively. You've seen an example of this above, where we searched for a range of dates and ratings. Elasticsearch also handles location in several formats, including geo points, geo polygons, and geo hashes. The sample movie data includes locations, drawn from 100 American cities, and assigned at random. You can run a bounding-box query by executing the following:

```

GET movies/_search
{
  "query": {
    "bool": {
      "filter": {
        "geo_bounding_box": {
          "location": {
            "top_left": {
              "lat": 50.96,
              "lon": -124.6
            },
            "bottom_right": {
              "lat": 46.96,
              "lon": -120.0
            }
          }
        }
      }
    }
  },
  "sort": [
    {
      "_geo_distance": {
        "location": {
          "lat": 47.59,
          "lon": -122.43
        },
        "order": "asc",
        "unit": "km"
      }
    }
  ]
}

```

This query searches a bounding box around Seattle and sorts the results by their distance from the center of Seattle. Even though the results themselves are not particularly meaningful (the locations are randomly assigned), the same method will let you search and sort your own data to display results that are geographically local for your customers.

Search free text

Elasticsearch's features make it easy for you to search natural language text in a language-aware way. It parses natural language text, applies stemming, stop words, and synonyms to make text matches better. And has a built-in notion of scoring and sorting that brings the best results to the top.

The sample data includes a *plot* field with short descriptions of the movies. This field is processed as natural language text. You can search it by entering the following query:

```
GET movies/_search
{
  "query": {
    "match": {
      "title": "star man"
    }
  },
  "_source": "title"
}
```

You'll see that you get a mix of movies, including *Star Wars*, *Star Trek*, and a couple of others. By default the *match* query uses a union (OR) of the terms from the query, so only one term is required to match. You could change the query to be a *bool* query and force both *star* and *man* to match. Let's instead experiment with the scoring.

```
GET movies/_search
{
  "query": {
    "function_score": {
      "query": {
        "match": {
          "title": "star man"
        }
      },
      "functions": [
        {
          "exp": {
            "year": {
              "origin": "1977",
              "scale": "1d",
              "decay": 0.5
            }
          }
        },
        {
          "script_score": {
            "script": "_score * doc['rating'].value / 100"
          }
        }
      ],
      "score_mode": "replace"
    }
  }
}
```

You've added an exponential decay function, centered on the year 1977. You've also used a script to multiply Elasticsearch's base score by a factor of each document's rating. This brings *Star Wars* to the top and sorts the rest of the movies mostly by their release date.

Retrieve aggregations

Elasticsearch's aggregations allow you to summarize the values in the fields for the documents that match the query. When you're providing a search user interface, you use aggregations to provide values that your users can use to narrow their result sets. Elasticsearch can aggregate text fields or numeric fields. For numeric fields, you can apply functions like *sum*, *average*, *min*, and *max*. The ability to aggregate at multiple levels of nesting is the basis of Elasticsearch's analytics capabilities, which we'll explore in the next section. Try the following query:

```
GET movies/_search
{
  "query": { "match_all": {} },
  "aggs": {
    "actor_count": {
      "terms": {
        "field": "actors.keyword",
        "size": 10
      },
      "aggs": {
        "average_rating": {
          "avg": {
            "field": "rating"
          }
        }
      }
    }
  },
  "size": 0
}
```

Your results will show you a set of buckets (aggs) for actors, along with a count of the movies they appear in. For each of these buckets, you will see a sub-bucket that averages the ratings field of these movies. This query also omits the *hits* by setting the *size* parameter to 0 to make it easier to see the aggregation results.

You can experiment with building aggregations of different types and with different sub-buckets.

Stream updates to Dynamo DB

- In your web browser, open a new tab for <https://console.aws.amazon.com> and choose **Lambda**
- Select **Functions** in the left pane
- You will see the three Lambda functions that the lab deploys, named <stack name>-WiringFunction-<string>, <stack name>-LambdaFunctionForDDBStreams-<string>, and <stack name>-StreamingFunction-<string>. Click **<stack name>-StreamingFunction-<string>**



- This function will generate random updates for the *clicks* and *purchases* fields, and send those updates to Dynamo. These updates will in turn be shipped to Amazon ES through the *LambdaFunctionForDDBStreams*. If you examine the code, you'll see that the function ignores its input and simply runs in a loop, exiting just before it times out
- Click **Test**
- Type an **Event Name**
- Since the inputs don't matter, you don't have to change them. Click **Create**
- Click **Test**. This will run for 5 minutes, streaming changes to Amazon ES via Dynamo DB

Analyze the changes with Kibana

If you examine the code for the *LambdaFunctionForDDBStreams*, you'll see that when data is modified in your Dynamo table, the function sends both the update and a log of the changes to the clicks and purchases to a *logs-<date>* index. You can use Kibana to visualize this information.

- Return to your Kibana tab (or open a new one)
- Click **Management** in the navigation pane
- Click **Index Patterns**. You use index patterns to tell Kibana which indexes hold time-series data that you want to use for visualizations
- In the **Index Pattern** text box, type **logs** (leave the * that Kibana adds automatically)
- Kibana reports **Success** in identifying an index that matches that pattern
- Click **Next Step**
- Here you tell Kibana which field contains the time stamp for your records. Select **@timestamp** from the **Time filter field name** drop down
- Click **Create Index Pattern**
- Kibana recognizes the fields in your index and displays them for you
- Click **Discover** in the navigation pane
- This screen lets you view a traffic graph (the count of all events over time) as well as search your log lines for particular values



- (Note, you can see my data covers 5 minutes and then stops. That's because the Lambda function that's streaming changes terminates. You can go back to the Lambda console and click Test again to stream more changes)
- You can also use Kibana to build visualizations and gather them into a dashboard for monitoring events in near real time
- Click **Visualize** in the navigation pane
- Click **Create a visualization** You can see that Kibana has many different kinds of visualizations you can build
- Click **Line**
- On this screen, you tell Kibana which index pattern to use as the source for your visualization. Click **logs***
- When building Kibana visualizations you will commonly put time on the X-axis and a function of a numeric field on the Y-axis to graph a value over time
- In the **Buckets** section, click **X-Axis**
- In the **Aggregation** drop down, select **Date Histogram**
- Click  to change the visualization. You now have a graph of time buckets on the X-Axis and the **Count** of events on the Y-Axis
- In the **Aggregation** drop down for the Y-Axis, select **Sum** and in the **Field** drop down, select **purchases** to see the sum of all purchases, broken down by time
- You can monitor changes in this metric, in near real time, by clicking  Auto-refresh in Kibana's top menu bar, and choosing **10 seconds**. Kibana now updates every 10 seconds. You might have to start the Lambda stream function again to generate more data or you might see data continuing to flow in

You can save your visualizations and build them into dashboards to monitor your infrastructure in near real time.

Clean up

- When you're done experimenting, return to the CloudFormation dashboard
- Click the check box next to your stack, then choose **Delete Stack** from the **Actions** menu
- Click **Yes, Delete** in the dialog box.

Conclusion

In this lab, you used CloudFormation to deploy a Dynamo DB table, and an Amazon Elasticsearch Service domain to search the data in your table. You explored building complex, *bool* queries that used the indexes for your fields' data. You explored Elasticsearch's geo capabilities, and its abilities to search natural language data. You created aggregations to analyze the movie data and discover the best-rated actor in this data set. Finally, you used Dynamo DB streams to flow updates from your table to your domain and Kibana to visualize the changes in your table.

Whew! That's a lot! And you've barely scratched the surface of Elasticsearch's capabilities. Feel free to use this lab guide and its template to explore more on your own.