Amazon Elasticsearch Service

Fully managed search and analytics service.

Have your front end and monitor it too!

Lab 1 Instructions

Contents

Lab 1 Overview

In this lab, you will be creating a web serving application create and rendered using React and powered by an Apache web server. The application calls Amazon API Gateway and AWS Lambda to serve up the content hosted in the Amazon Elasticsearch Service. Once you have configured the services to interact with each other, you will be able to see a working web application and you will have a lightweight NGINX proxy to interact with Kibana in a DMZ for public facing web traffic.

Using a pre-created environment created prior to this session, you will configure Amazon Cognito to interact with the Amazon Elasticsearch Service for access to Kibana. A URL provided in the stack creation gives you access to your backend monitoring visualizations.

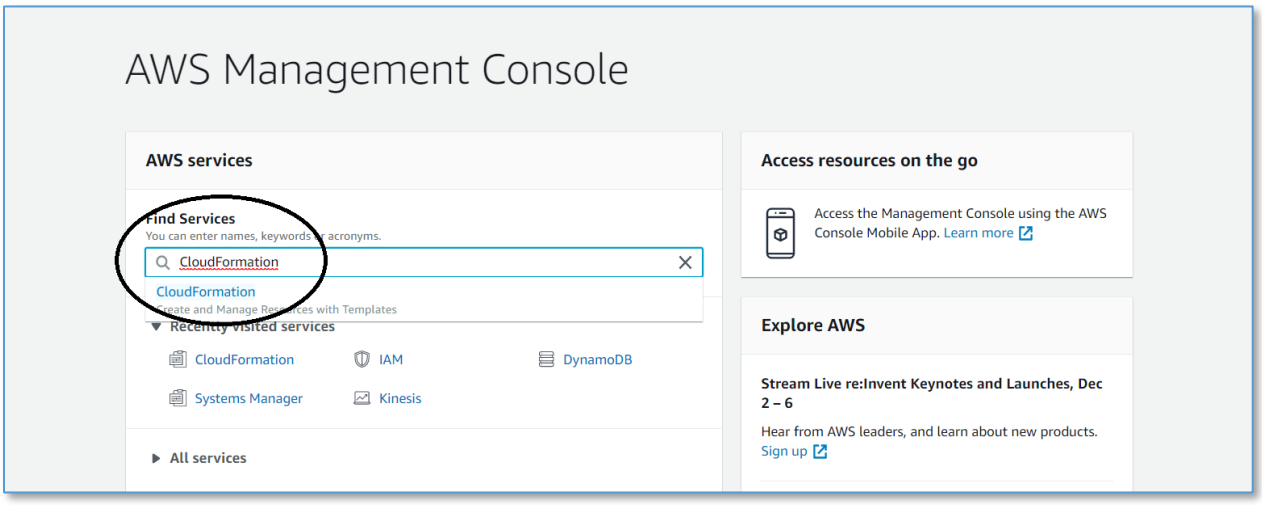
You will then enable access to the domain through an NGINX proxy server that map proxies' calls to the Amazon Elasticsearch Service and Amazon Cognito for authentication and authorization.

Using AWS Systems Manager, you will get console access to an Amazon Linux instance. You will use the session created by AWS Systems Manager to review and run scripts that populate your elasticsearch domain.

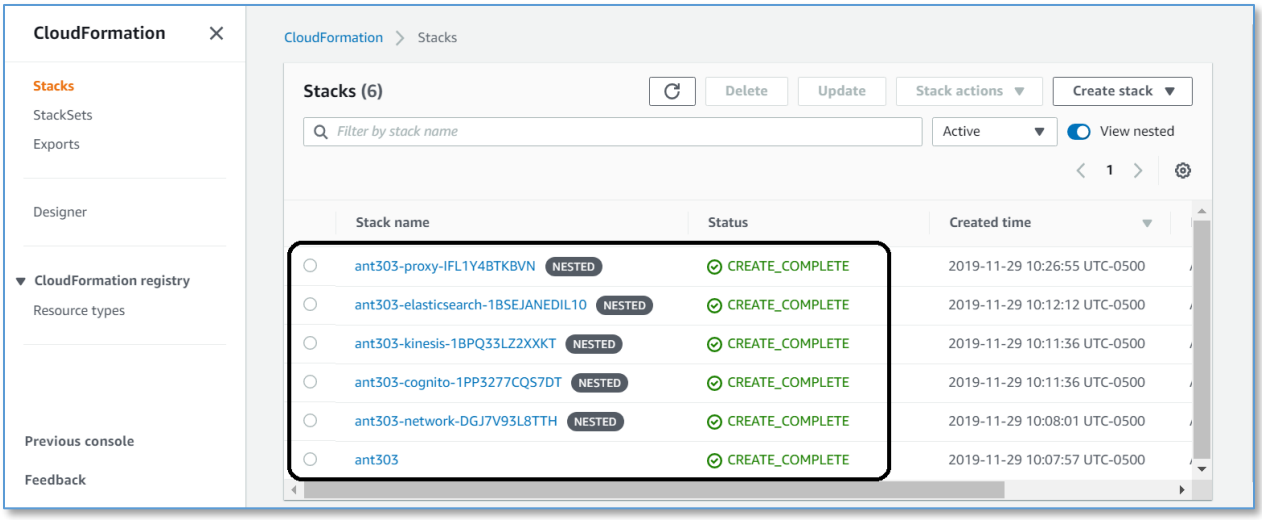
You will deploy the application layer using AWS CloudFormation. The resulting deployment provides a URL to view the content through the React web servers.

Reviewing prebuilt materials / housekeeping

Using the credentials provided when you arrived in the lab session, access the AWS account found on the piece of paper given to you. Once logged in, navigate to the AWS CloudFormation console.



You should see a screen similar to this:



As you can see, the foundation for building lab 1 is ready so that you can deploy the lab 1 AWS CloudFormation templates. You should see the following stacks created in your lab account:

- A kickoff stack with the stack name of “ant303”
- A network stack with the name matching “ant303-network-*”
- An authentication stack with the name matching “ant303-cognito-*”
- A stream stack with the name matching “ant303-kinesis-*”
- An Elasticsearch domain stack with the name matching “ant303-elasticsearch-*
- A Kibana proxy stack with the name matching “ant303-proxy-*”

As you review the outputs and resources created for you, take the time to copy the following template outputs from the existing stacks for future use in the lab.

What are these prebuilt components and what do they do?

Let us review the services used in the prebuilt solution:

1) Amazon Elasticsearch Service – provides a secure deployment of Elasticsearch and Kibana so that you can serve IMDB content and collect logs from the proxy, web service, and log aggregation layers. It also hosts Kibana for visualization and review of all data in the solution.
2) Amazon Cognito – provides authentication and authorization for users accessing Kibana. The Amazon Elasticsearch Service provides integration with Amazon Cognito to ensure only authorized users can interact with your domain and specifically Kibana.
3) Amazon EC2 – provides the compute. In future version of this lab, containers will be used.

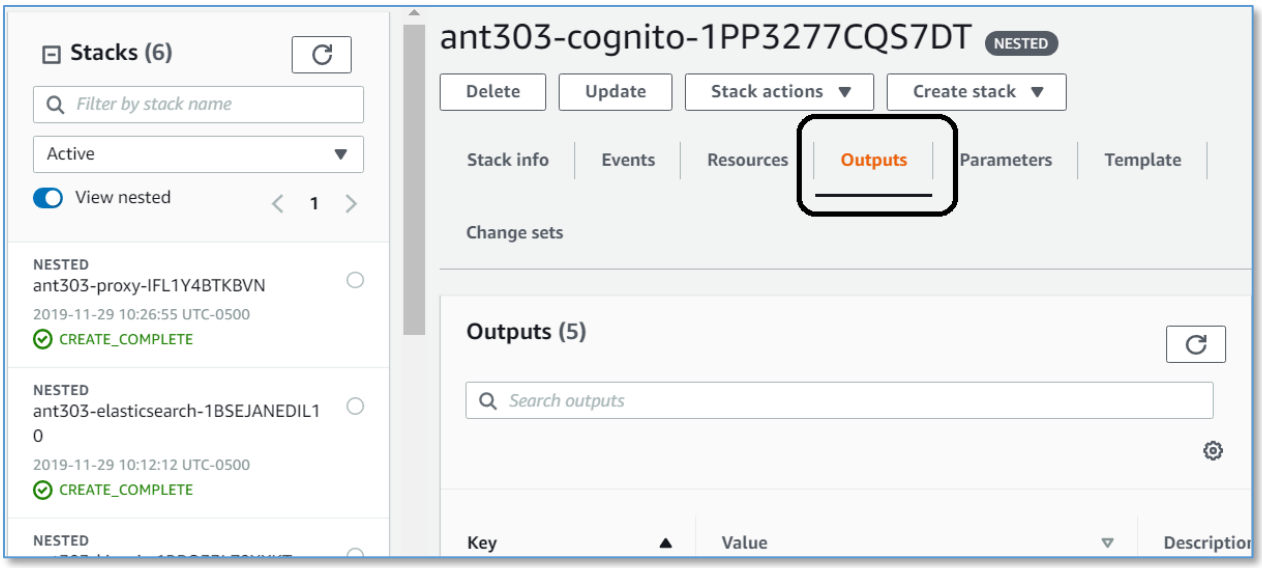
4) Amazon Kinesis Data Streams – provides the buffer for the solution. You typically decouple your components when operating at scale. You still want to capture logs but in some cases, the volume of requests coming into your Amazon ES domain might be too much and you want to put “backpressure events” into a stream or queue to manage limitations depending on how you scale your Amazon ES domain.
5) Amazon Virtual Private Cloud (VPC) – provides a private networked, private IP addressable foundation that allows you to build a secure and private solution.
6) NGINX – provides a secure proxy layer between the internet, Amazon Cognito, and the Amazon Elasticsearch Service.

This foundation gives you the environment with which you deploy the rest of the solution.

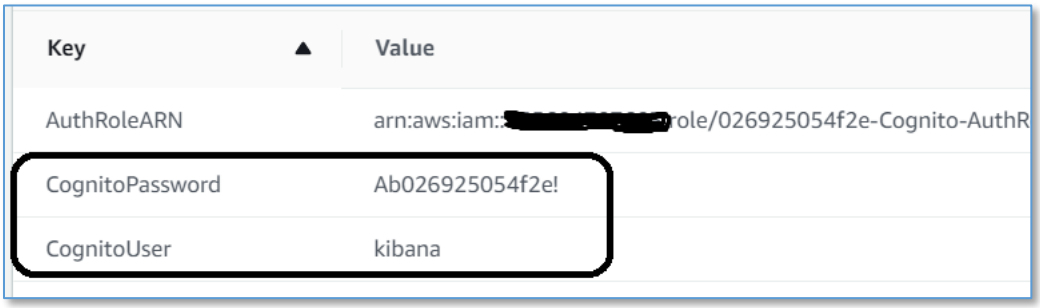
Save off output parameters to a text file

First, select the stack labeled with “ant303-cognito”.

Click into the stack details breadcrumb and you should see a screen similar to this:

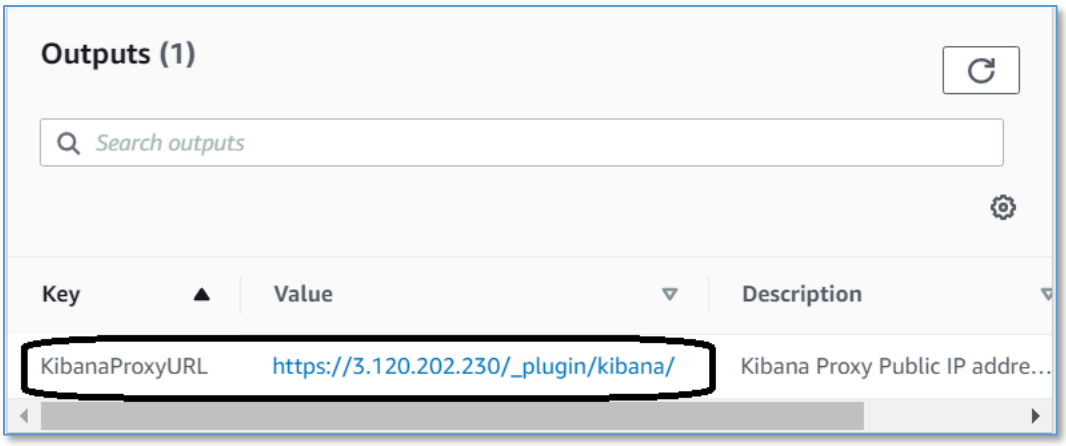


Navigate to the Outputs tab and scroll down in paste the output the following parameter values (**CognitoUser** and **CognitoPassword**) into a text file for future reference:



Next, navigate to the stack labeled like “ant303-proxy-*”.

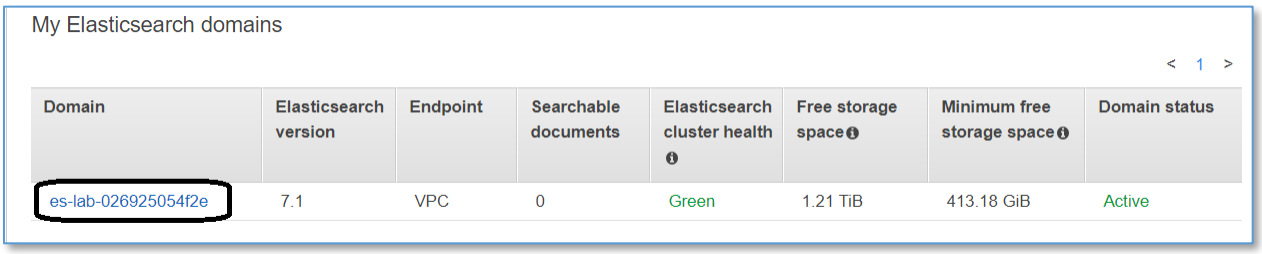
As was done in the prior stack, pull out the value for the **KibanaProxyURL**:



Make sure you keep the text file around for future cut and paste activities. This lab will be diving into configuration of the different components and it is always good to have notes for reference.

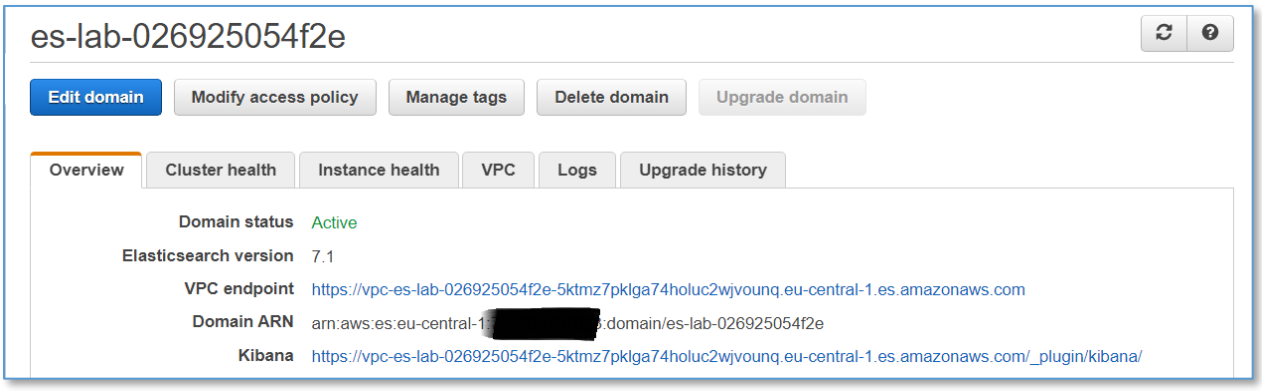
Configure authentication for secure Kibana pubic access

First, we need to pair the Amazon Cognito user and identity pools with your Amazon ES domain. You need to navigate to the Amazon ES console and you should see your something similar to what you see in the screenshot:



Click on the breadcrumb as seen circled above to gain access to the configuration of your domain.

You will see multiple tabs for the domain. Find the “Edit domain” button and click on it.

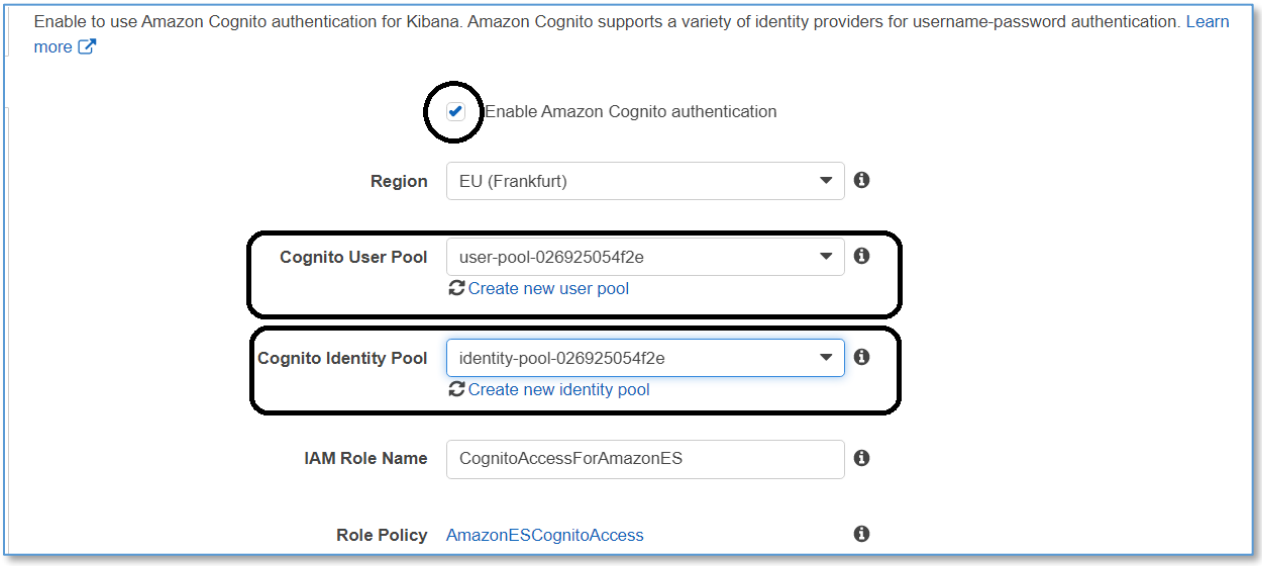


Scroll down in the settings, find the “Enable Amazon Cognito authentication” check box, and click on it. The following dropdowns you need to configure are:

- Cognito User Pool
- Cognito Identity Pool

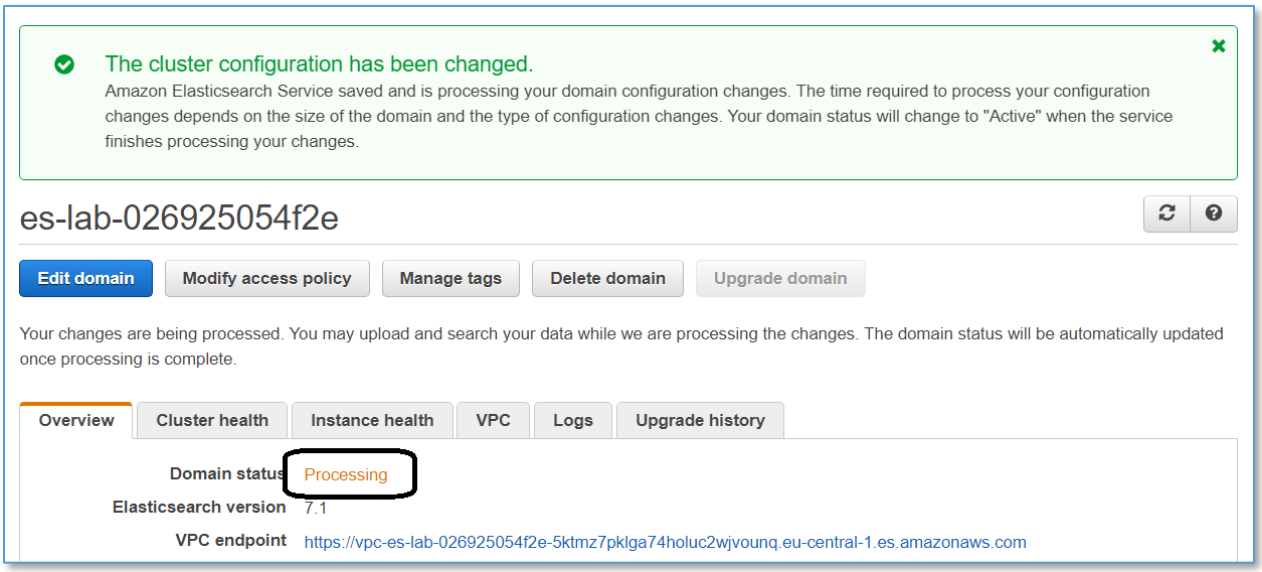
Do not change the other settings.

Both of these should auto populate. The pool names are suffixed the same as your domain. For example in the picture above, my domain is “**es-lab-026925054f2e**”. My user pool and identity pool names are suffixed with “**026925054f2e**”.



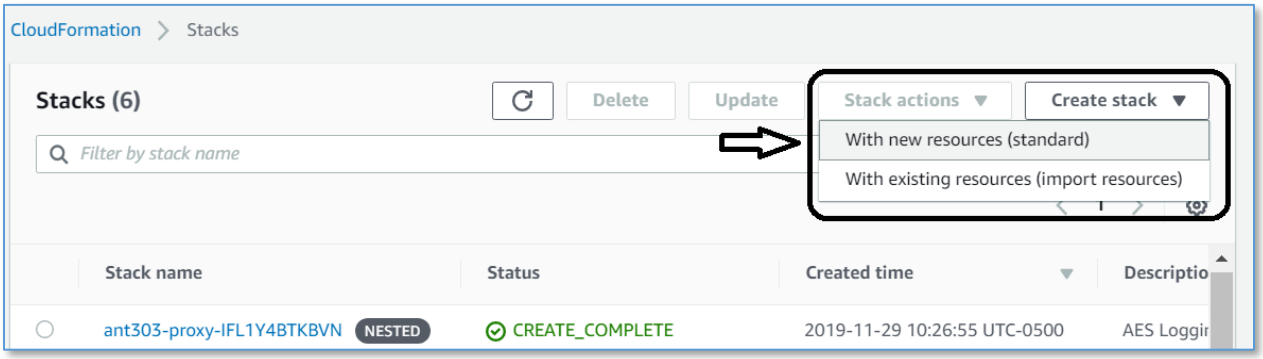
Click on the “Submit” button at the bottom of the screen.

Verify you see something similar to the screen below.



Launch the “bootcamp-aes-lab1” nested stack

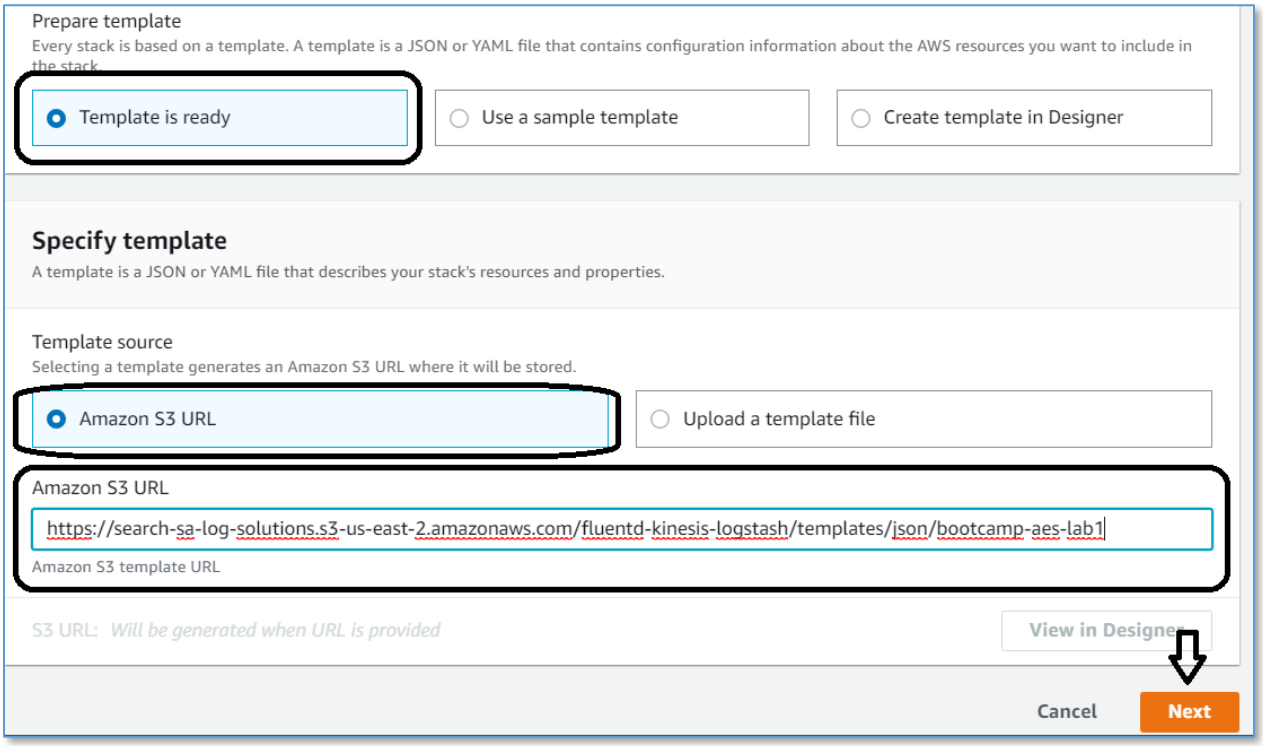
Navigate to the AWS CloudFormation console and click on the create stack button.



You should see two options. Click the “With new resources (standard)” option. You should now see a screen similar to the one below. Select the “Template is ready” option with the template source being an Amazon S3 URL. Paste in the following link (mac users beware PDF documents sometimes do tricky things with URLS that have dashes):

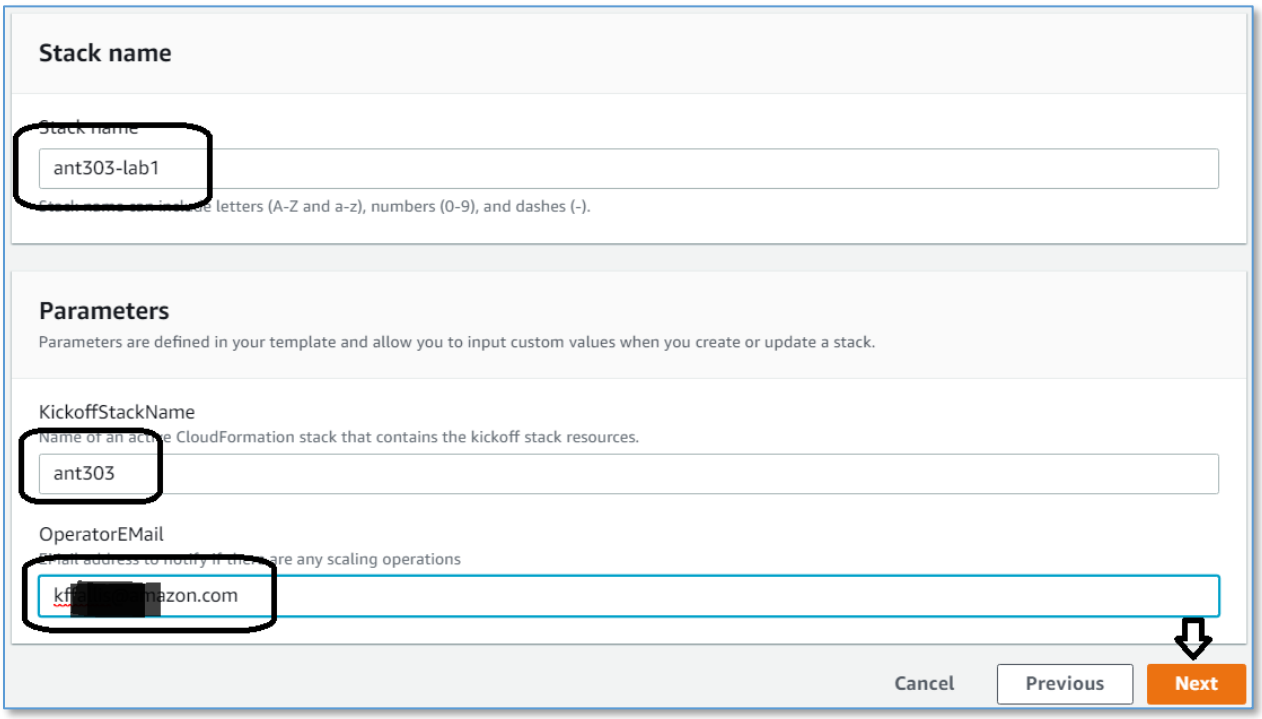
https://search-sa-log-solutions.s3-us-east-2.amazonaws.com/fluentd-kinesis-logstash/templates/json/bootcamp-aes-lab1

Make sure all dashes are present. Click the next button when you are done.

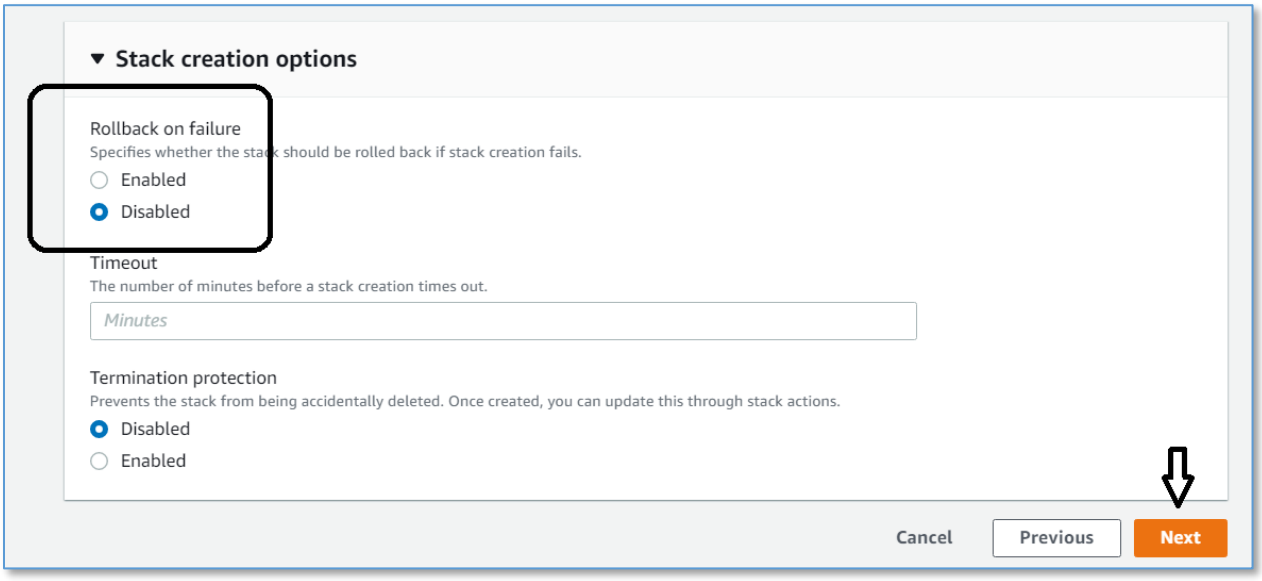


Populate the needed parameters.

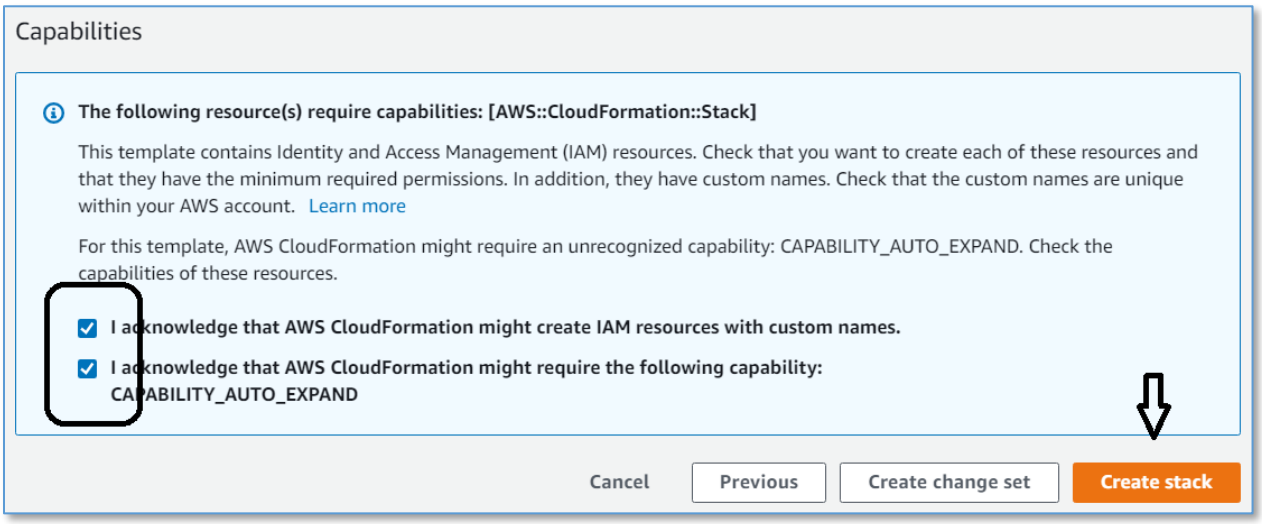
- Stack Name – **ant303-lab1**
- KickoffStackName – **ant303**
- OperatorEMail – your email address (is not captured and used for anything other than giving you notifications in instance events via auto scaling).



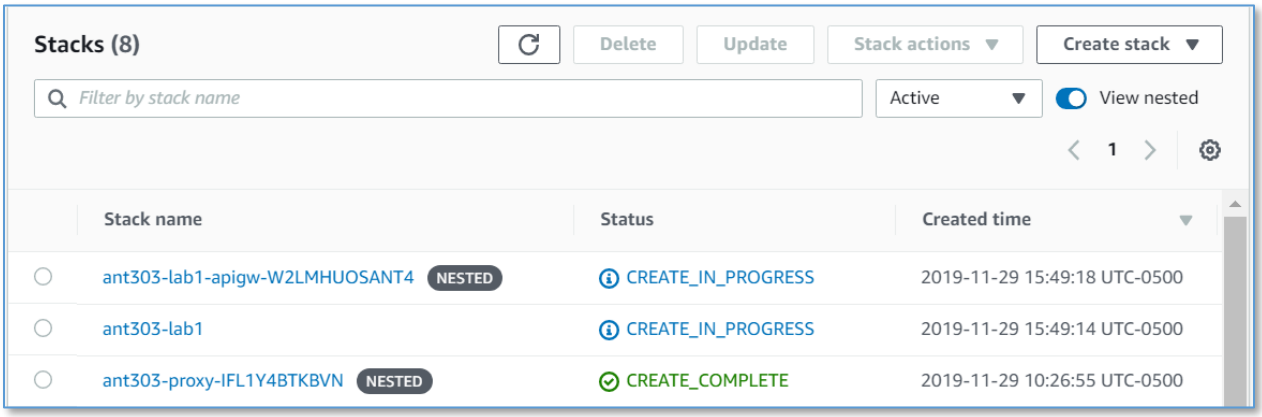
Click next. On the new screen select Stack creation options and disable rollback. Click next.



Review the deployment. Select the approval checkboxes for IAM and then click on the "Create stack" button.



Periodically monitor the deployment.



Script and data review

Your Kibana proxy instance deployed with scripts that create the index template and mappings for the "movies" index. Without data, your web application will not be able to serve content. You need to execute these scripts to make things work properly. Let's dig into what these look like and what their function is.

Movies template

The "movies" template defines field mappings for the data pulled down from an Amazon S3 bucket. Let us inspect the template and associated mappings. You can download the mappings found here.

```
{
   "index_patterns":[
      "movies*"
   ],
   "settings":{
      "index.number_of_shards":1,
      "index.number_of_replicas":2
   },
   "mappings":{
      "properties":{
         "actors":{
            "type":"text",
            "fields":{
               "keyword":{
                  "type":"keyword",
                  "ignore_above":256
               }
            }
         },
         "directors":{
            "type":"text",
            "fields":{
               "keyword":{
                  "type":"keyword",
                  "ignore_above":256
               }
            }
         },
         "genres":{
            "type":"text",
            "fields":{
               "keyword":{
                  "type":"keyword",
                  "ignore_above":256
               }
            }
         },
         "image_url":{
            "type":"text",
            "fields":{
               "keyword":{
                  "type":"keyword",
                  "ignore_above":256
               }
            }
         },
         "plot":{
            "type":"text",
            "fielddata":true,
            "fields":{
               "keyword":{
                  "type":"keyword",
                  "ignore_above":256
               }
            }
         },
         "rank":{
```

```
            "type":"long"
         },
         "rating":{
            "type":"float"
         },
         "release_date":{
            "type":"date"
         },
         "running_time_secs":{
            "type":"long"
         },
         "title":{
            "type":"text",
            "fields":{
               "keyword":{
                  "type":"keyword",
                  "ignore_above":256
               }
            }
         },
         "year":{
            "type":"long"
         }
      }
   }
}
```

At the first part of the document, you see an “index_patterns” section. This section is a directive that states any index created that matches the pattern “movies*” will use the mappings and shard settings.

The “settings” section describes the distribution and setup of data for our domain. Since elasticsearch is a distributed search technology, you can deploy shards – data for your index – across multiple data nodes. For search use case, I like to deploy using a “shard to node” ratio of one to one as a starting point for index deployment. In this case, the index is super small and will not have performance benefits by breaking the index into multiple primary shards.

In most search cases where the overall index is under 50GB and you have ample compute and memory resources to leverage file system cache, distributing one primary shard and N replicas = data node size – 1 allows you to leverage all data nodes in the request cycle. Since you will not have scatter – gather (the act of distributing the search request across all nodes that have a portion of a distributed index) to interfere with performance and assuming you have plenty of file system cache, you should be able to increase performance since all the data for one search if it is in one node and does not need to coordinate with other nodes. As always, consider this a rule of thumb for a starting point when doing “search” use cases. Log analytics use cases have a different approach.

The final section is the mappings. In this section, you are defining field properties so you can optimize performance and set boundaries on the data. For example, a date can be string or date type. If elasticsearch maps the date data to a string, you lose the ability to do date based operations. In the majority of use cases, defining mappings up front for your index gives you control of how data is analyzed and stored and in most cases, allows you to be more effective with your data footprint than using dynamic mappings.

IMDB text data

This data file contains the IMDB data in one large JSON array. You will need to parse this data with a custom python script. You will review that shortly. The data looks like this:


```
   {
      "fields" : {
         "directors" : [
            "Shawn Levy"
         ],
         "release_date" : "2013-06-05T00:00:00Z",
         "rating" : 6.3,
         "genres" : [
            "Comedy"
         ],
         "image_url":"http://ia.media-
imdb.com/images/M/MV5BMjM1MzczMDgwOV5BMl5BanBnXkFtZTcwMDM4NjM2OQ@@._V1_SX400_
.jpg",
         "plot" : "Two salesmen whose careers have been torpedoed by the digital
age find their way into a coveted internship at Google, where they must compete
with a group of young, tech-savvy geniuses for a shot at employment.",
         "title" : "The Internship",
         "rank" : 196,
         "running_time_secs" : 7140,
         "actors" : [
            "Vince Vaughn",
            "Owen Wilson",
            "Rose Byrne"
         ],
         "year" : 2013
      },
      "id" : "tt2234155",
      "type" : "add"
   },
```


If you compare the data to the mappings, you will see how the data maps when stored in the index.

Mappings script

The mappings script is python and uses Sigv4 request signing to communicate with the domain. Since you deploy the domain in VPC with IAM controlling access to the domain, you need to ensure that each request to the domain is signed so that the IAM role found in the signature maps to permissions that give access to resources in the domain.

Look at the code snippet and let us review what it does:

```
    service = 'es'
    credentials = boto3.Session().get_credentials()
    awsauth    =    AWS4Auth(credentials.access_key,    credentials.secret_key,
opts.region, service, session_token=credentials.token)

    url = "https://%s/_template/movies_template" % opts.endpoint

    payload = {
        "index_patterns":[
            "movies*"
        ],
        "settings":{
            "index.number_of_shards":1,
            "index.number_of_replicas":2
        },
        "mappings":{
            ...
        }
    }


    res = requests.put(url, auth=awsauth, json=payload)
    print res
```

First, you setup the access to credentials found on the EC2 instance in the instance metadata. This metadata contains the IAM role associated with the proxy server when launched. The AWS4Auth from this import "`from requests_aws4auth import AWS4Auth`" brings in the necessary package. Once you have built the request signer, you create the URL for the endpoint. I am using the "`requests`" package to perform a HTTPS POST request to the Amazon Elasticsearch Service. The EC2 instance has a role that allows all "es:ESHttp*" requests from this instance.

You then call the _template API on the service to create the template for the mappings and settings.

Load script

Just as with the mappings script, the load script is python and uses Sigv4 request signing to communicate with the domain.

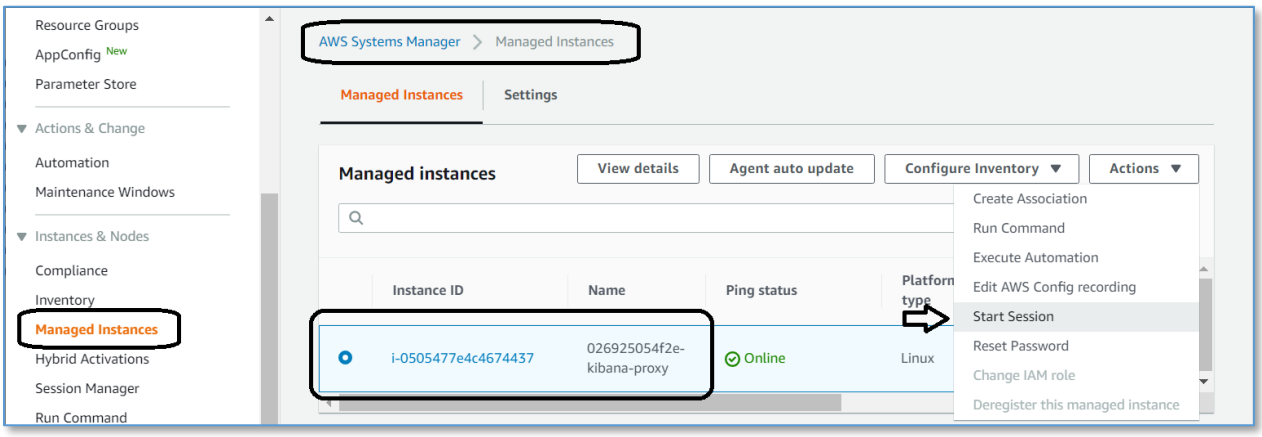
Look at the code snippet and let us review what it does:

```
    service = 'es'
    credentials = boto3.Session().get_credentials()
    awsauth = AWS4Auth(credentials.access_key,credentials.secret_key,
        opts.region,service,session_token=credentials.token)

    records = list()
    url = "https://%s/_bulk/" % opts.endpoint

    with open('2013Imdb.txt', 'r') as imdb_data_file:
        imdb_data = json.load(imdb_data_file)
        for rec in imdb_data:
            records.append('{"index" : { "_index" : "movies"} }')
            records.append(json.JSONEncoder().encode(rec['fields']))
            if (len(records) / 2) >= 100:
                print 'Flushing %d records' % (len(records) / 2)
                res=requests.post(url, auth=awsauth, data=("\n".join(records))
+ "\n", headers={"Content-Type":"application/x-ndjson"})
                print res
                records = list()
        print 'Flushing %d records' % (len(records) / 2)
        requests.post(url, auth=awsauth, data="\n".join(records))
```

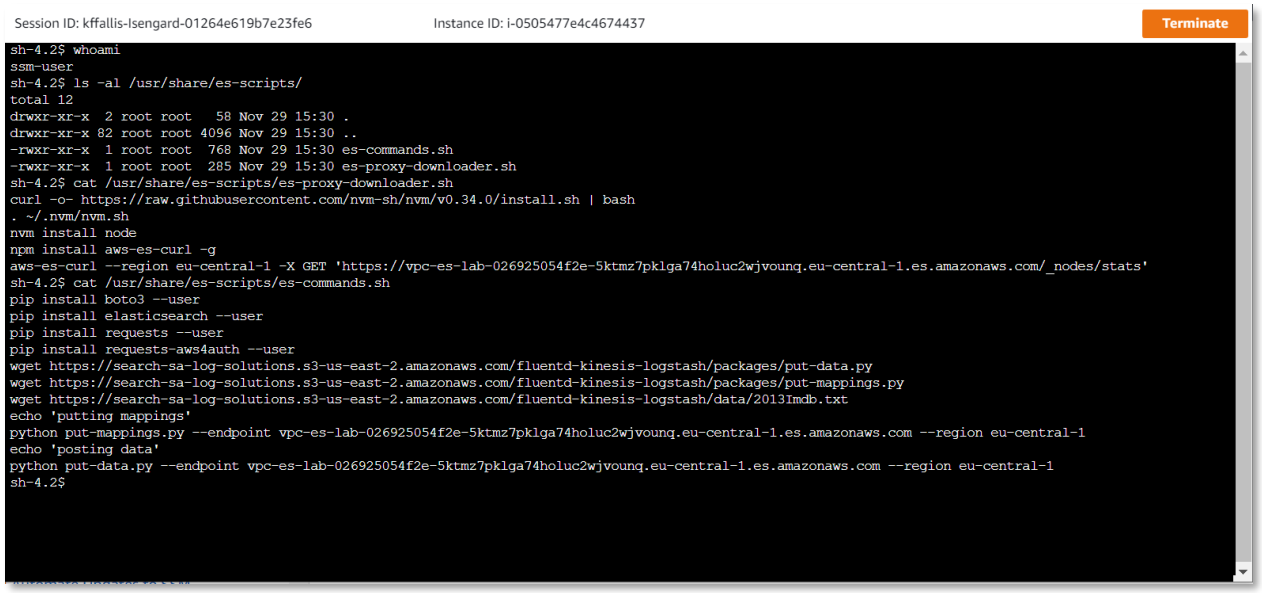
You then iterate over the JSON file until all data processes. Request singing is an important part of securing your domain and is a necessary function of making things work. Tool like curl or wget do not natively sign requests to the Amazon ES service. You leverage the _bulk API to load the data into the domain.

Load the data into the domain

Now that you have an overview of how to configure the index and use python tooling to create data in the index, navigate to AWS Systems Manager and start a session on the proxy server. Each server in this workshop is running the AWS Systems Manager Agent (SSM Agent). You access the instance via the agent by providing the correct IAM policy associated with each server. The instance is seen as a managed instance in the AWS Systems Manager. You need to click the radio button for the managed instance. This gives you access to the actions drop down. Click “Start Session”.



A session will launch in your browser and should look similar to this:



The AWS CloudFormation template "bootcamp-aes-kibany-proxy" created two scripts called "es-commands.sh" and "es-proxy-downloader.sh". They exist in the "/usr/share/es-scripts" directory.

Review the shell scripts

Run the Linux command:

```
sh-4.2$ ls /usr/share/es-scripts
```

`ls /usr/share/es-scripts`

It provides this output:

```
total 12
drwxr-xr-x  2 root root   58 Nov 29 15:30 .
drwxr-xr-x 82 root root 4096 Nov 29 15:30 ..
-rwxr-xr-x  1 root root  768 Nov 29 15:30 es-commands.sh
-rwxr-xr-x  1 root root  285 Nov 29 15:30 es-proxy-downloader.sh
```

View the contents of each file using the "cat" command.

- es-proxy-downloader.sh

This shell script downloads node and the aws-es-curl project for installation. This utility enables access to your domain and lets you query from the proxy if needed.

```
sh-4.2$ cat /usr/share/es-scripts/es-proxy-downloader.sh
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.34.0/install.sh |
bash
. ~/.nvm/nvm.sh
nvm install node
npm install aws-es-curl -g
aws-es-curl --region eu-central-1 -X GET 'https://vpc-es-lab-026925054f2e-
5ktmz7pklga74holuc2wjvounq.eu-central-1.es.amazonaws.com/_nodes/stats'
```

- es-commands.sh

This shell script wraps the mapping and data loading python scripts you reviewed earlier.

```
sh-4.2$ cat /usr/share/es-scripts/es-commands.sh
pip install boto3 --user
pip install elasticsearch --user
pip install requests --user
pip install requests-aws4auth --user
wget https://search-sa-log-solutions.s3-us-east-2.amazonaws.com/fluentd-
kinesis-logstash/packages/put-data.py
wget https://search-sa-log-solutions.s3-us-east-2.amazonaws.com/fluentd-
kinesis-logstash/packages/put-mappings.py
wget https://search-sa-log-solutions.s3-us-east-2.amazonaws.com/fluentd-
kinesis-logstash/data/2013Imdb.txt
echo 'putting mappings'
python put-mappings.py --endpoint vpc-es-lab-026925054f2e-
5ktmz7pklga74holuc2wjvounq.eu-central-1.es.amazonaws.com --region eu-central-
1
echo 'posting data'
python put-data.py --endpoint vpc-es-lab-026925054f2e-
5ktmz7pklga74holuc2wjvounq.eu-central-1.es.amazonaws.com --region eu-central-
1
```

Run the commands

Each script automatically adds the Amazon ES domain values so you do not have to worry about editing for the correct values. You simply run the scripts.

Navigate to the ssm-user home directory

Make sure you are in the home directory of the ssm-user by issuing a "cd" command:

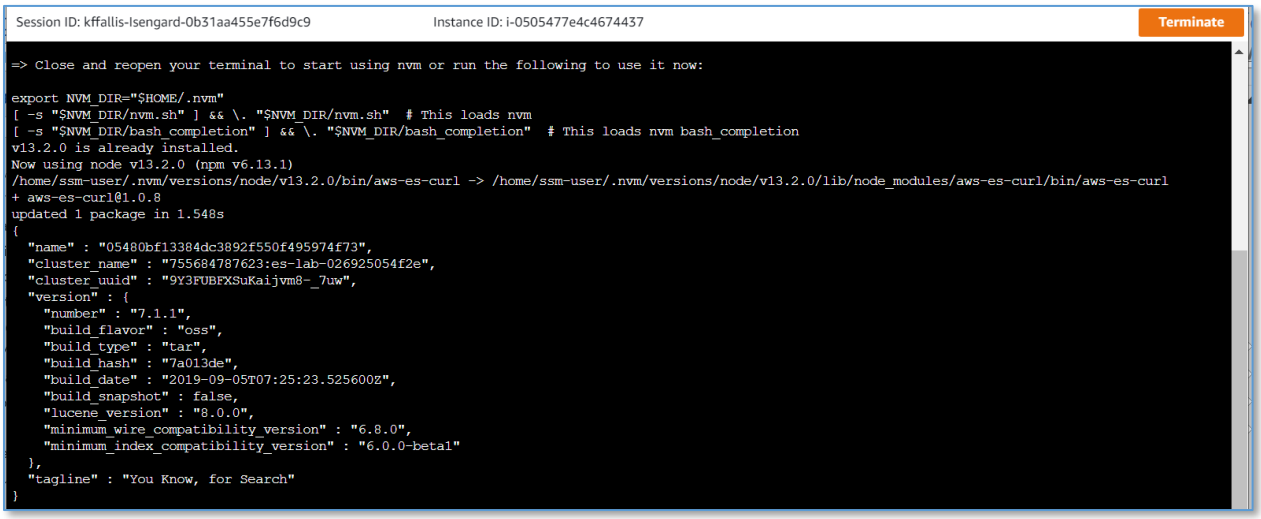
```
sh-4.2$ cd
sh-4.2$ pwd
/home/ssm-user
```

Run the es-proxy-downloader.sh script

This will download the necessary node projects and the aws-es-curl utility to validate the permissions are set on the domain. Run the following script ". /usr/share/es-scripts/es-proxy-downloader.sh"

```
sh-4.2$ . /usr/share/es-scripts/es-proxy-downloader.sh
```

Observe the following output. You should see the tag line "You know, for search".



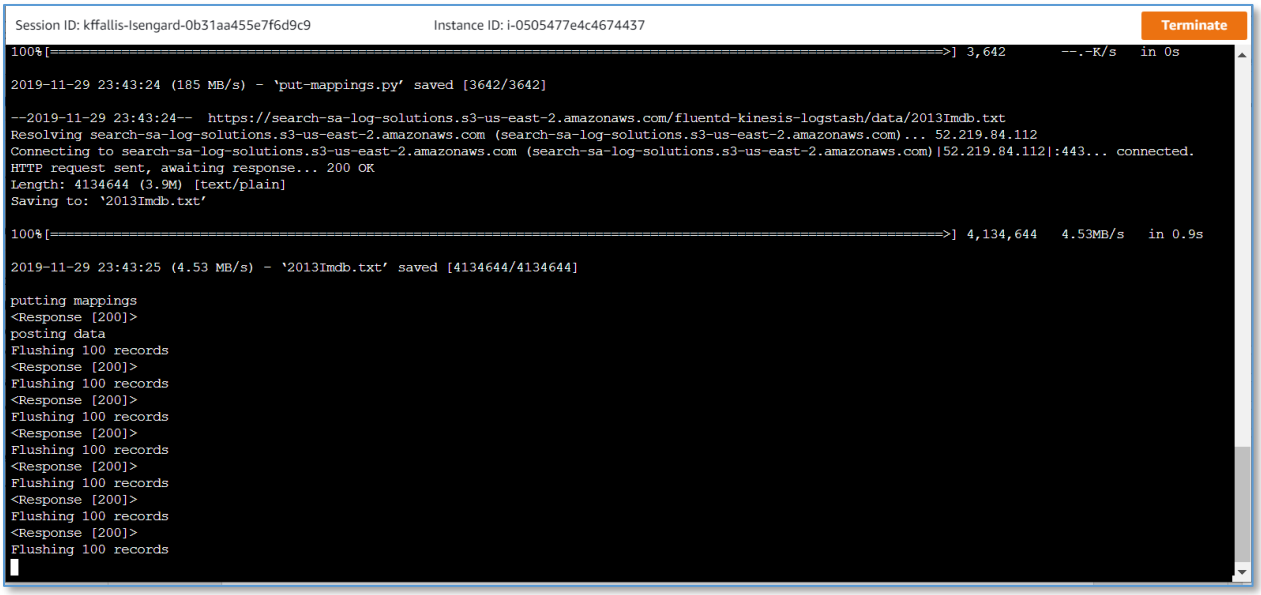
Now you have verified you can connect to your domain.

Run the es-commands.sh script

This will download the necessary node projects and the aws-es-curl utility to validate the permissions are set on the domain. Run the following script ". /usr/share/es-scripts/es-commands.sh"

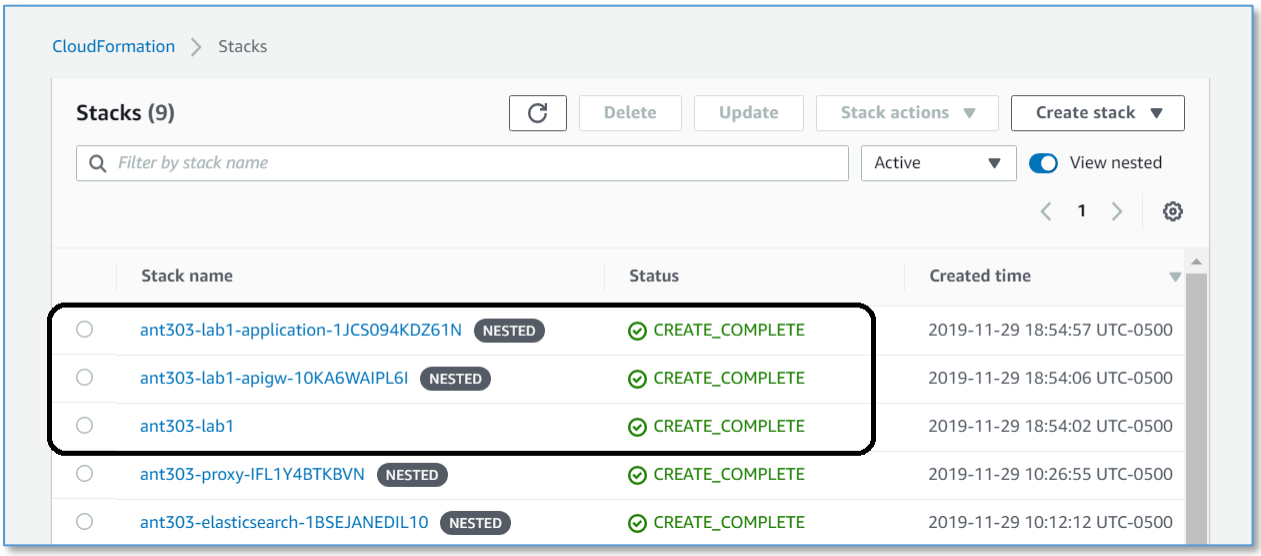
```
sh-4.2$ . /usr/share/es-scripts/es-commands.sh
```

Observe the following output. You should see output that prints "putting mappings" and "putting data" with response codes of 200.



Validate you can serve data through the application

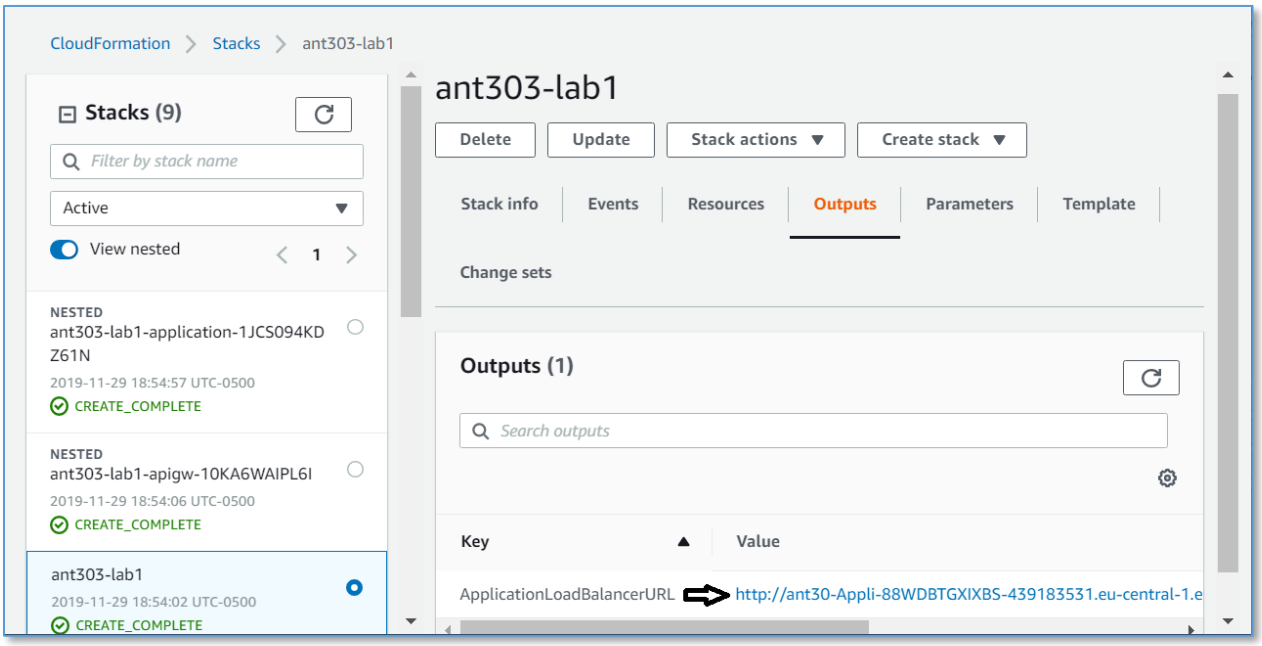
Navigate back to the AWS CloudFormation console and check on your deployment. Observe the lab 1 nested stack has completed.



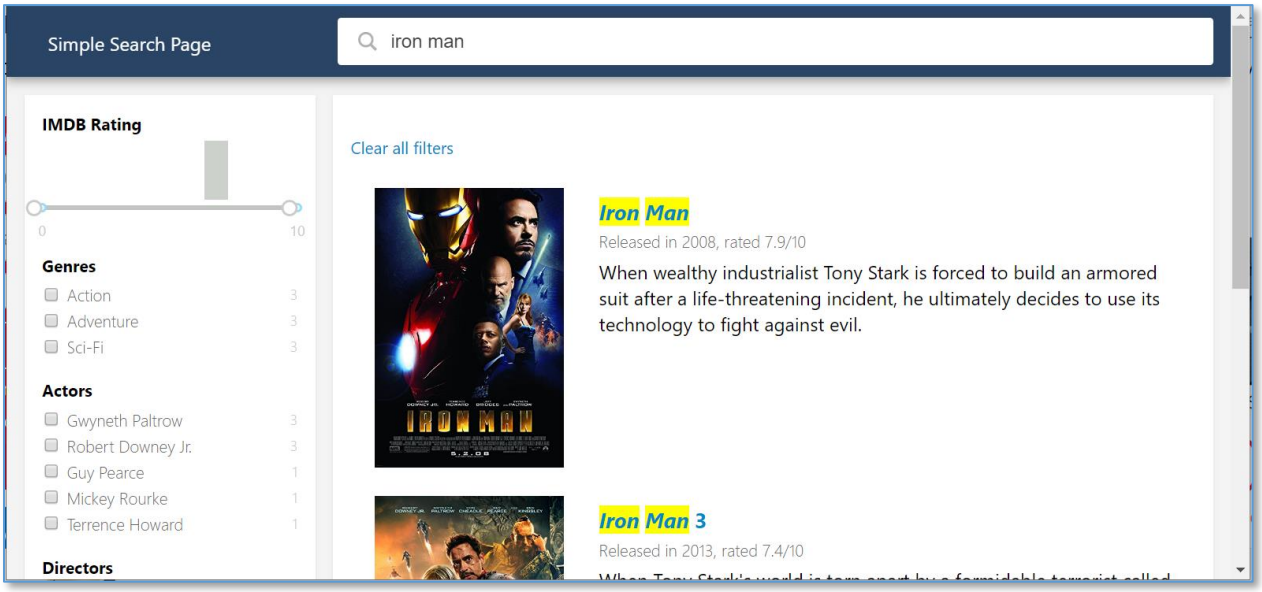
This stack has created the following components:

- API for "facading" REST calls to the Amazon ES domain.
- Apache web server with the React components to provide a robust web application that allows your customers to search a small subset of movies.

Click on the “ant303-lab1” breadcrumb to take you to stack details.



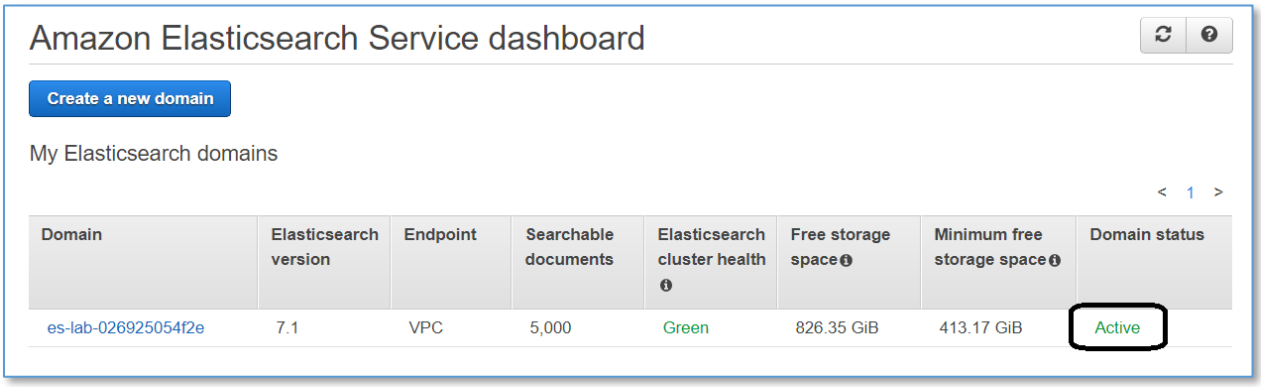
There is an output parameter called “ApplicationLoadBalancerURL” that takes you to the application. Copy this URL and paste in a new browser window. Type “iron man” in the search page. Notice the type-ahead feature as you type in the search term. The results change. Observe that you see similar details:



Congratulations! You have configured and deployed a React web site that uses Amazon API Gateway to serve data from your Amazon ES domain.

Enable the NGINX proxy server for access to Kibana

By now, the changes for Amazon Cognito integration with your Amazon ES domain has completed. You can verify this by navigating to the domain in the Amazon Elasticsearch Service console.

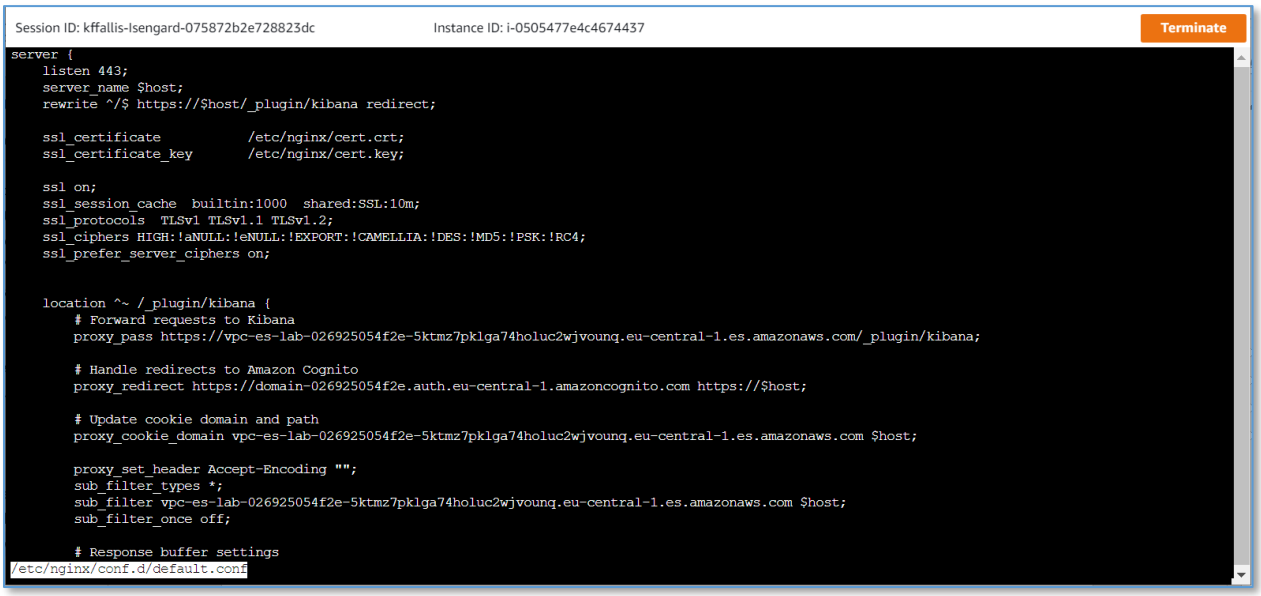


Observe that the domain is in an "active" state. This means that your changes for Amazon Cognito authentication are applied. Go back to the AWS Systems Manager console. If your session has expired, (it will expire if there is no activity), relaunch a session.

Switch to the "root" user and start the service

Run following commands. This will give you privileged access to the NGINX commands to start the proxy service.

```
sh-4.2$ sudo su - root
Last login: Fri Nov 29 22:46:33 UTC 2019 on pts/0
[root@ip-10-1-0-150 ~]# less /etc/nginx/conf.d/default.conf
```



This configuration has already been setup with AWS CloudFormation based on the configuration details found here. What the proxy service does is allow Amazon Cognito to communicate with the Amazon Elasticsearch Service. Amazon Cognito does not yet support Interface VPC Endpoints as of this document created in November of 2019. Interface endpoints allow private traffic from your VPC to an AWS service that supports the functionality. Additionally, the Amazon Elasticsearch Service ***does not expose Kibana with a public endpoint if you deploy your domain with the VPC setting***. Amazon ES needs a proxy layer that runs in a DMZ to broker requests between private and public endpoints.

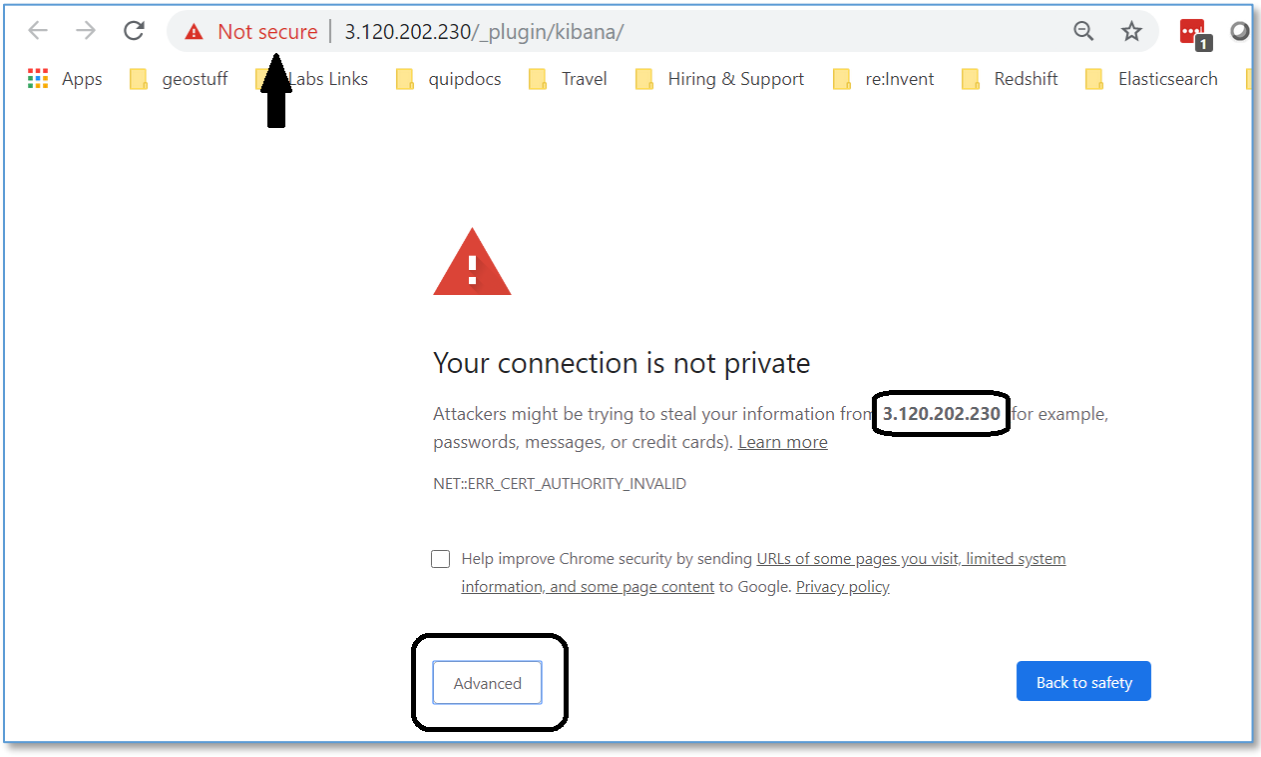
Use the combination [esc] key, then [:] key, then [q] key to get out of "less" (less is more ☺ and both use vi commands).

Go ahead and start the NGINX service on your proxy server by issuing the following command and seeing the response:

```
[root@ip-10-1-0-150 ~]# service nginx start
Redirecting to /bin/systemctl start nginx.service
[root@ip-10-1-0-150 ~]#
```

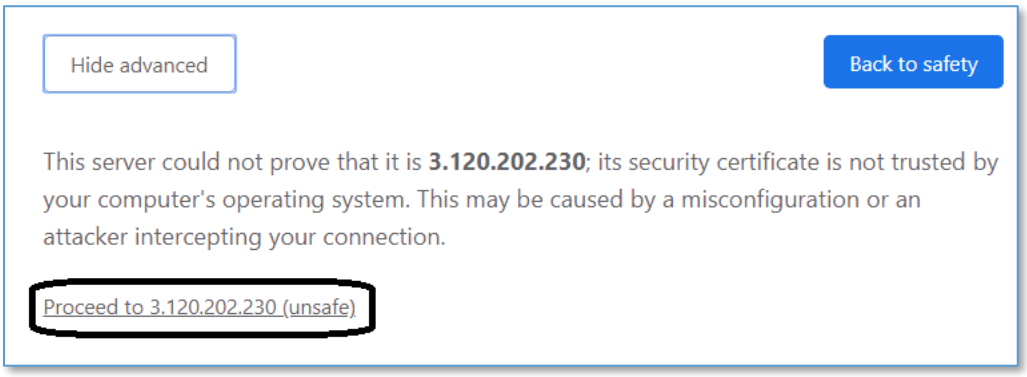
Validate you can connect to Kibana

Using the **KibanaProxyURL** you copied to a text file earlier in the lab, paste the URL in your browser and verify connectivity.

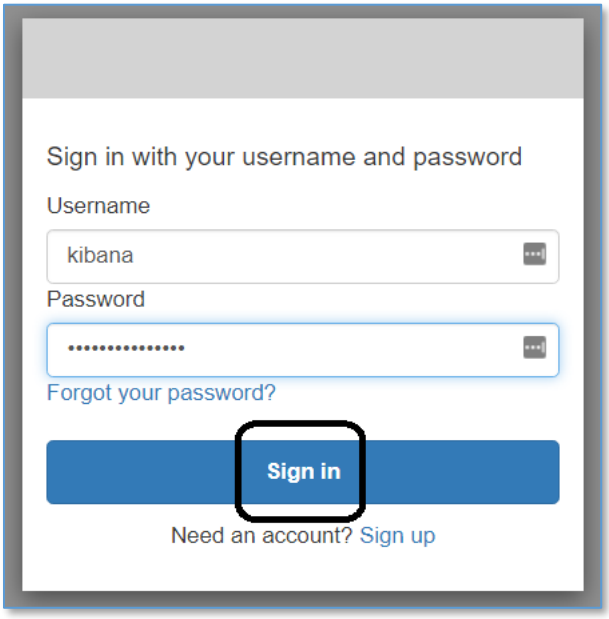


It might appear that you are not secure. In this lab, you are using self-signed certificates on the serving host. This means that you are not using a trusted SSL certificate from an authoritative body. You can deploy a trusted certificate by using the AWS Certificate Manager.

Click on the “Advanced” button. You will see a link at the very bottom left of the screen that takes you to Kibana.



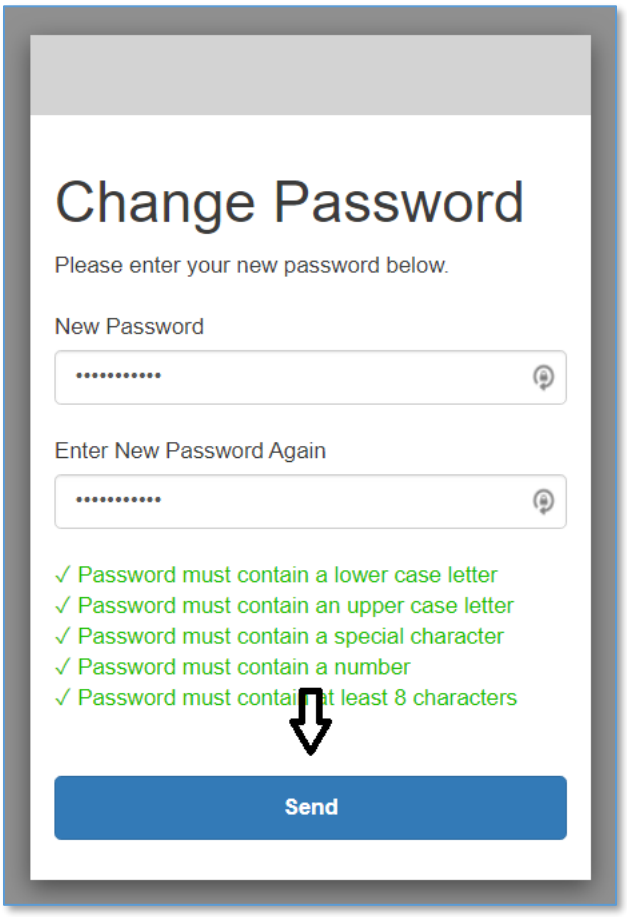
Once you click on the link, it will bring you to the login page that interacts with Amazon Cognito.



Use the output parameters you copied from the templates. The parameter names are:

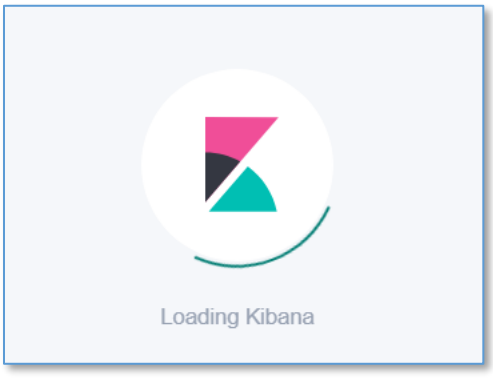
- CognitoUser
- CognitoPassword

Once you click the Sign in button, a new popup appears so you can change the one time password assigned via the template.

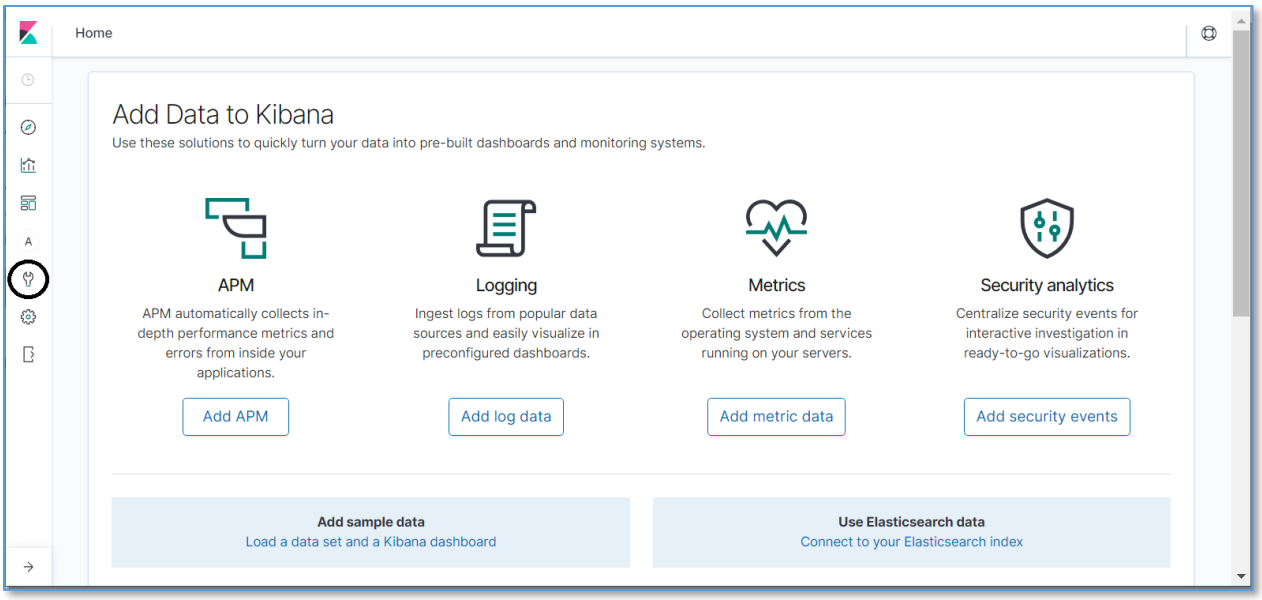


Change your password and please save it somewhere that you can remember. Since you are not validating with email, the “forgot password” option is not available.

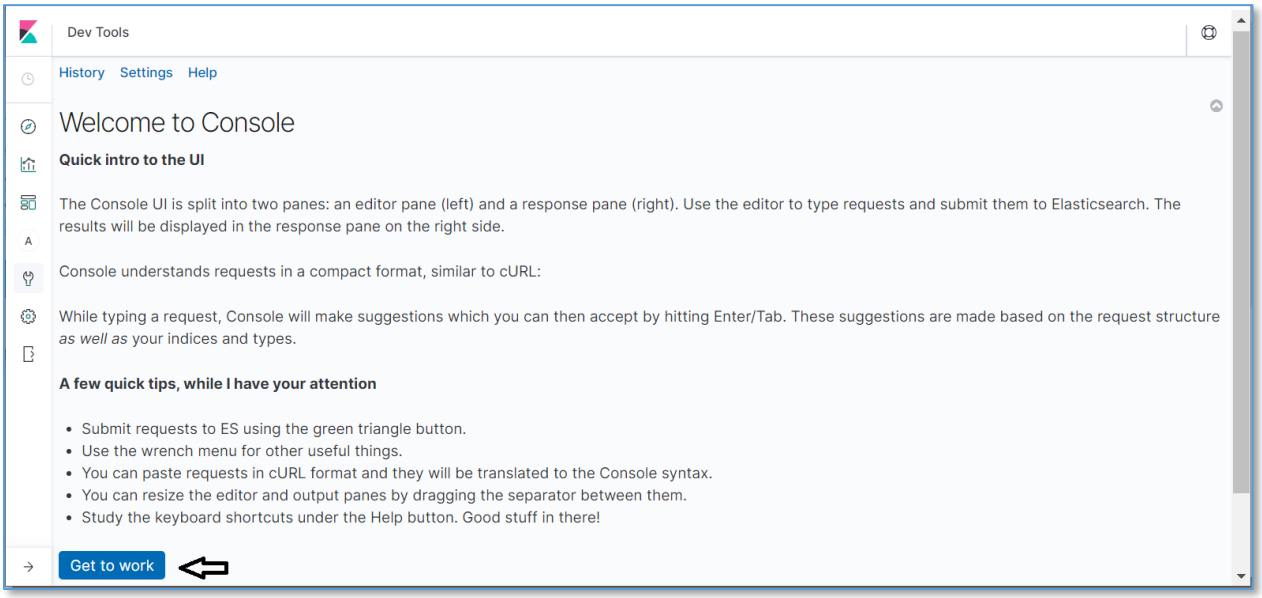
If everything is successful, you will see an animation as Kibana loads to your browser.



Once the Kibana application has been loaded to your browser, you will see the following:



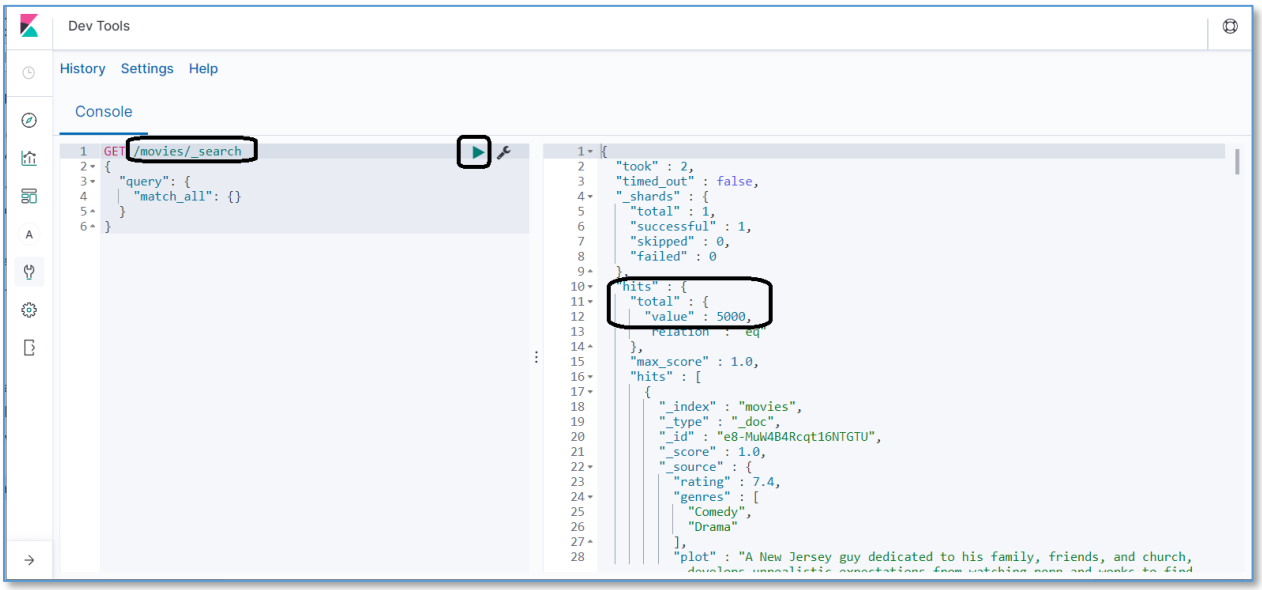
Navigate to the “Dev Tools” icon circled on the left middle of this image.



Near the very bottom of the screen is a “Get to work” button. Do not be shy…click it! ☺

You are now in the dev tools plugin console. Kibana is a rich environment that can support a broad variety of tools ranging from security to alerting to visualizations. Time to explore!

Modify the command line where you see **GET**, add "/**movies**/" to the search endpoint and click on the triangle that has a point facing to the right. Once the command executes, a result set is returned that show how many records matched the query (select * in SQL). You also get a sampling of the data in the index. In this case, we get 5000 records that match. By default, 10 records return. The entire response is JSON formatted.



Finally, run the following commands to get the metricbeats template:

```
sh-4.2$ wget https://search-sa-log-solutions.s3-us-east-2.amazonaws.com/fluentd-kinesis-logstash/packages/put-metricbeat-mappings.py
```

You will install these later.

Now that you have created the end-to-end solution, it is time to jump to Lab 2!