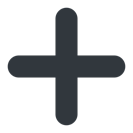
**S3 to AWS Elasticsearch Service**

**Comp 4964 & 4968 Final Project  
  
Nelson Liao  
Luc Murakami**

**Consuming data from a CSV file and Indexing it though AWS Elasticsearch Service**



Elasticsearch is an analytics engine for indexing and searching data. It takes in textual, numerical, geospatial, structured and unstructured data and allows the user to search on an input, and returns a list of responsed based on a calculated closeness ranking. It is built on top of the Apache Lucene engine and has great utility in the ingestion, enrichment, storage, analysis and visualization of data.

In this project, we combined 4 key AWS services to allow us to take csv files and automatically transform the data contained inside into an elasticsearch index. The following steps will take you through deploying an AWS S3 bucket, setting up an AWS DynamoDB table with streaming, creating two AWS Lambda functions and finally setting up the AWS managed AWS Elasticsearch Service.

We use data from the Vancouver Police Department’s Crime Data Download open data website, and the City of Vancouver’s Open Data Portal to get data about bike thefts, and bike racks to give you an idea of the bike crimes that have occurred in the past on any Vancouver road.

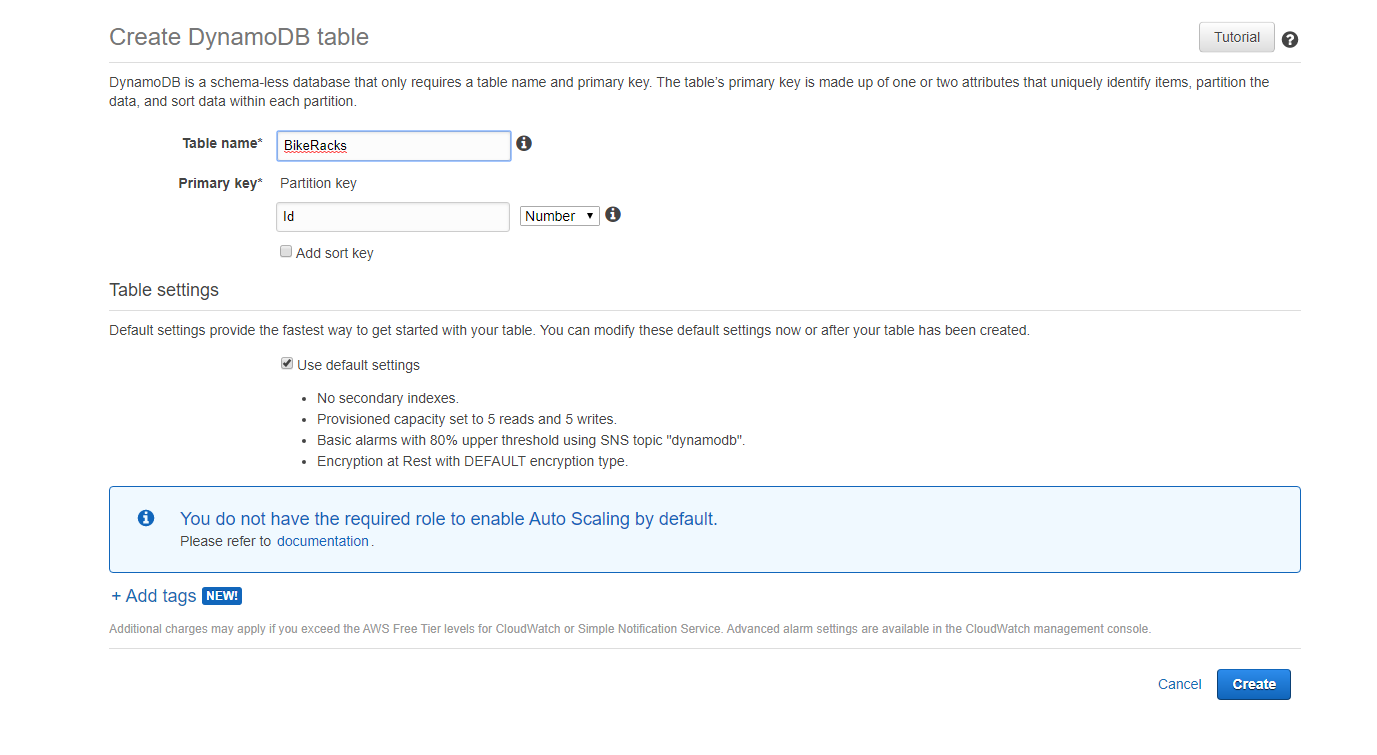
This documentation creates an app that works as is, but it can be easily remodelled to be used for other data, or even, input types. To get started, please refer to the next page.

**How to get .csv Files from S3 and Insert Into a DynamoDB Table**

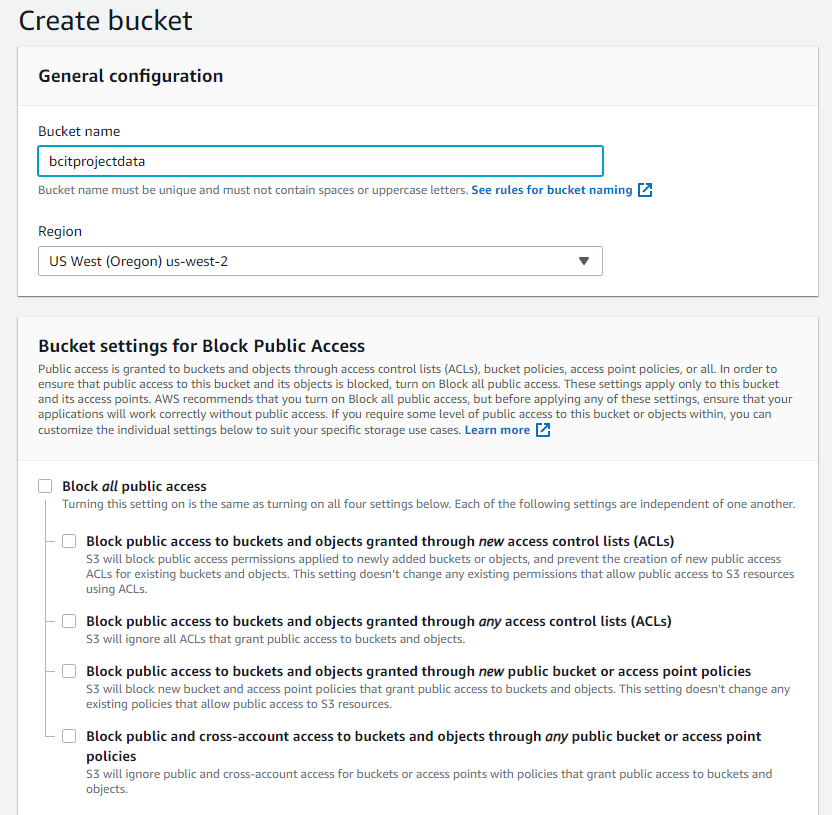
Before we can use ElasticSearch to index and analyze the bike rack data, we must first insert the data into a DynamoDB table.

Create a DynamoDB Table and a S3 Bucket

1. Create a DynamoDB table with a table name and primary key. In this example, we used BikeRacks as the table name and used a primary key named Id with a type of Number.



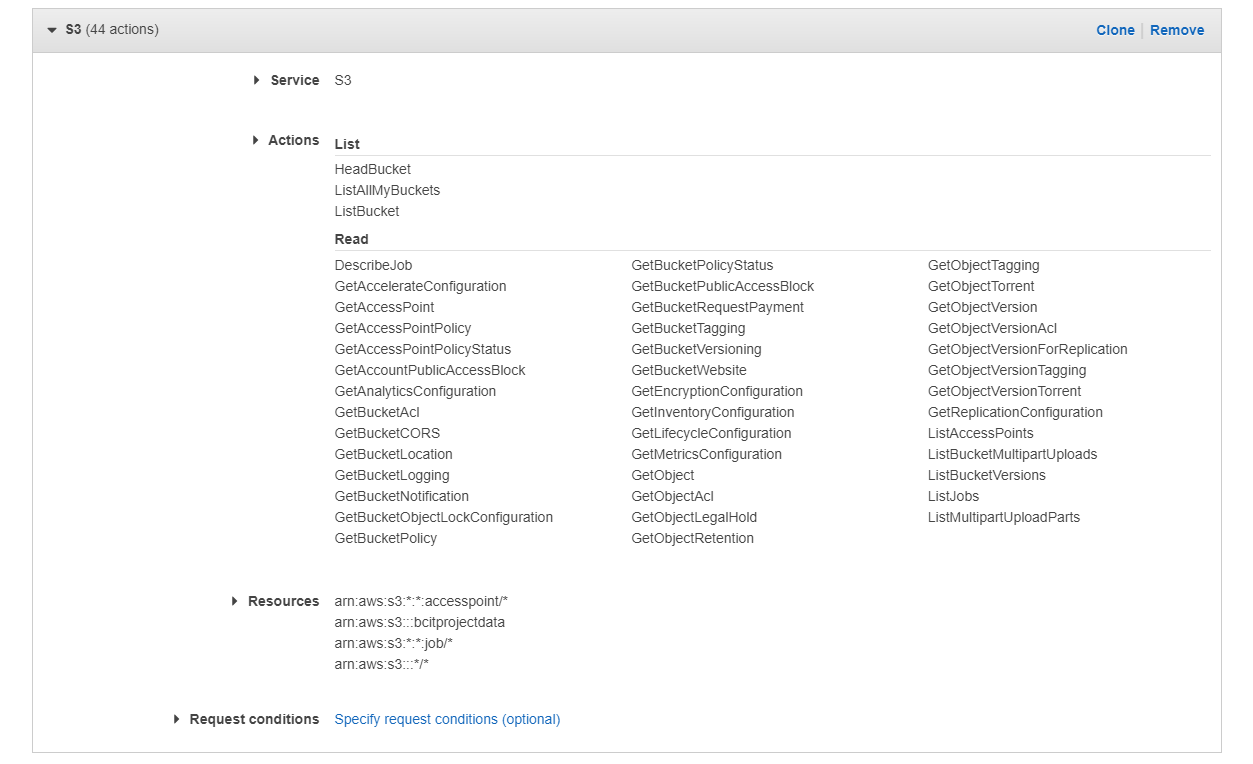
1. Create an S3 bucket with a bucket name in the us-west-2 region. Enable public access by unchecking Block all public access.



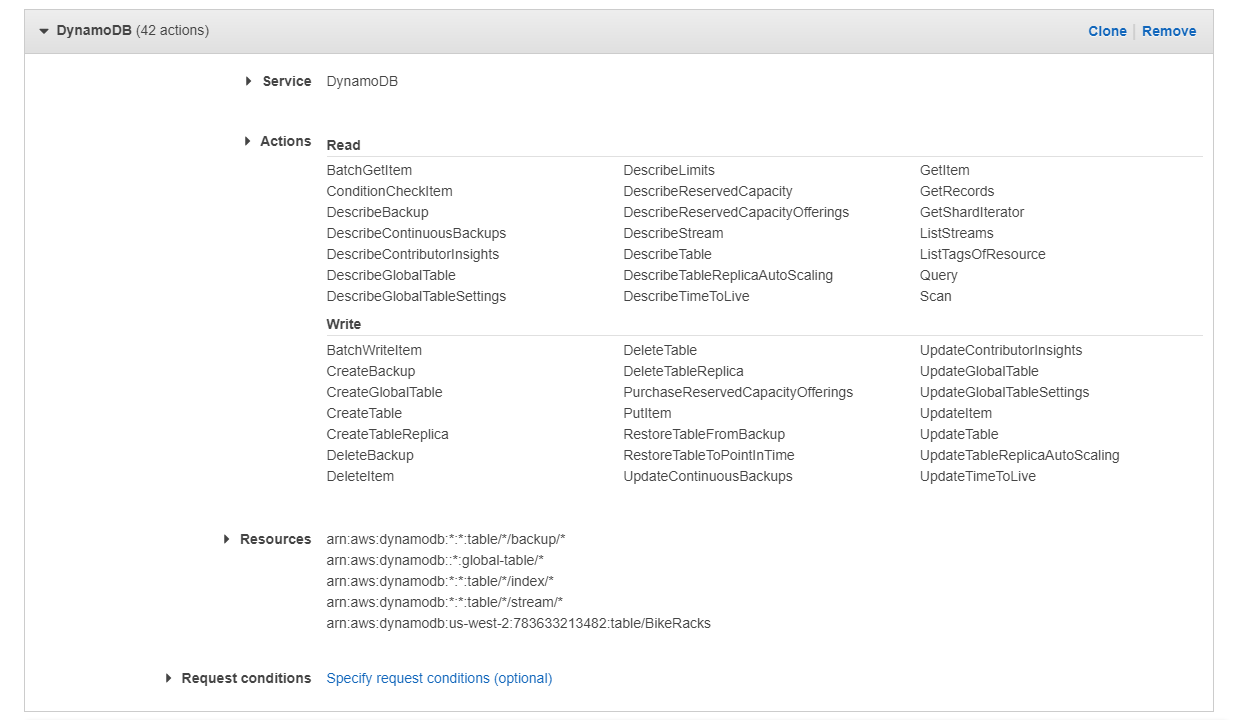
1. Upload the crimedata\_csv\_all\_years.csv file to the newly created S3 bucket

Create an IAM Role for Lambda Function

1. Select Lambda as the role’s use case
2. Create a policy that allows the bike rack S3 bucket to be read
   1. Select S3 as the service for the policy
   2. Select List and Read for the policy’s access level
   3. Specify the bike rack S3 bucket and select any for the remaining resources
   4. Name the policy and provide a description for it



1. Attach the newly created S3 bucket policy to the IAM role
2. Create a policy that allows your DynamoDB table to be read from and written to
   1. Select DynamoDB as the service for the policy
   2. Select Read and Write for the policy’s access level
   3. Specify the DynamoDB and select any for the remaining resources
   4. Name the policy and provide a description for it



1. Attach the newly created DynamoDB policy to the IAM role
2. Provide the role with a name and description and create it

Create a Lambda Function

Here, we will be creating a Lambda function that will trigger when a .csv file is placed in our S3 bucket. The function will parse both bike rack data and crime data and insert them into our DynamoDB table.

1. Create a function using Author from scratch
2. Enter the function’s name and select Python 3.8 as the runtime language
3. Select “Use an existing role” and choose your created IAM role
4. Create a trigger for the Lambda function
   1. Select S3 as your trigger
   2. Select your bucket containing the crimedata\_csv\_all\_years.csv file
   3. Select All object create events as the Event Type
5. Upload the csv\_to\_ddb.py
6. Edit Basic settings
   1. Set timeout to 10 minutes
   2. Set memory 1600 MB
7. Upload the bike-racks.csv file to your S3 bucket
8. After 10 minutes, check your DynamoDB table to see if the data has been placed

**How to load Streaming Data from DynamoDB to AmazonES**

Now that you have the bike rack data in your DynamoDB table, we will now setup a Lambda function that will take the DynamoDB stream from the table and load it into Amazon ElasticSearch for indexing and analysis.  
  
Enable Streams for DynamoDB

1. First, you will need to activate a stream for the BikeRacks DynamoDB table. Navigate to AWS Management Console > DynamoDB, select your BikeRacks table and choose the Overview tab.   
     
   A screenshot of a cell phone

   Description automatically generated
2. Under the Stream details header, press the Manage Stream button, select the New image option and then press the Enable button.  
     
   A screenshot of a cell phone

   Description automatically generated  
   A screenshot of a cell phone

   Description automatically generated

Create an Amazon ElasticSearch Domain

1. Now, you can create your Amazon ElasticSearch domain. From the AWS Management Console, either: search for Elasticsearch Service, or find the Elasticsearch Service under the Analytics subtopic.
2. On the Amazon Elasticsearch Service dashboard, click Create a new domain. On the Step: 1 Choose deployment type page, for Deployment type choose Development and testing and for Elasticsearch version choose the latest version. Choose N­ext.  
     
   A screenshot of a social media post

   Description automatically generated
3. On the Step 2: Configure domain page, add an Elasticsearchdomain name. I called mine bike-theft-data. For Instance type choose t2.small.elasticsearch. Number of nodes, keep the instance number at 1. Choose EBS for Data nodes storage type, General Purpose (SSD) for EBS volume type, and 10 for EBS storage size per node. Keep all other default values and choose Next.  
     
   A screenshot of a cell phone

   Description automatically generated
4. For Step 3: Configure access and security, choose Public access. Add a Domain access policy by selecting IPv4, entering your private IP address and select Allow. (If you do not know your public IP address, you can look this up by typing “Whats my IP” on google) Leave all other defaults and choose Next.  
     
   A screenshot of a social media post

   Description automatically generated
5. Review your configuration and choose Confirm to create your Amazon Elasticsearch Service domain. It will take about 10 minutes to create.

Create an IAM Role

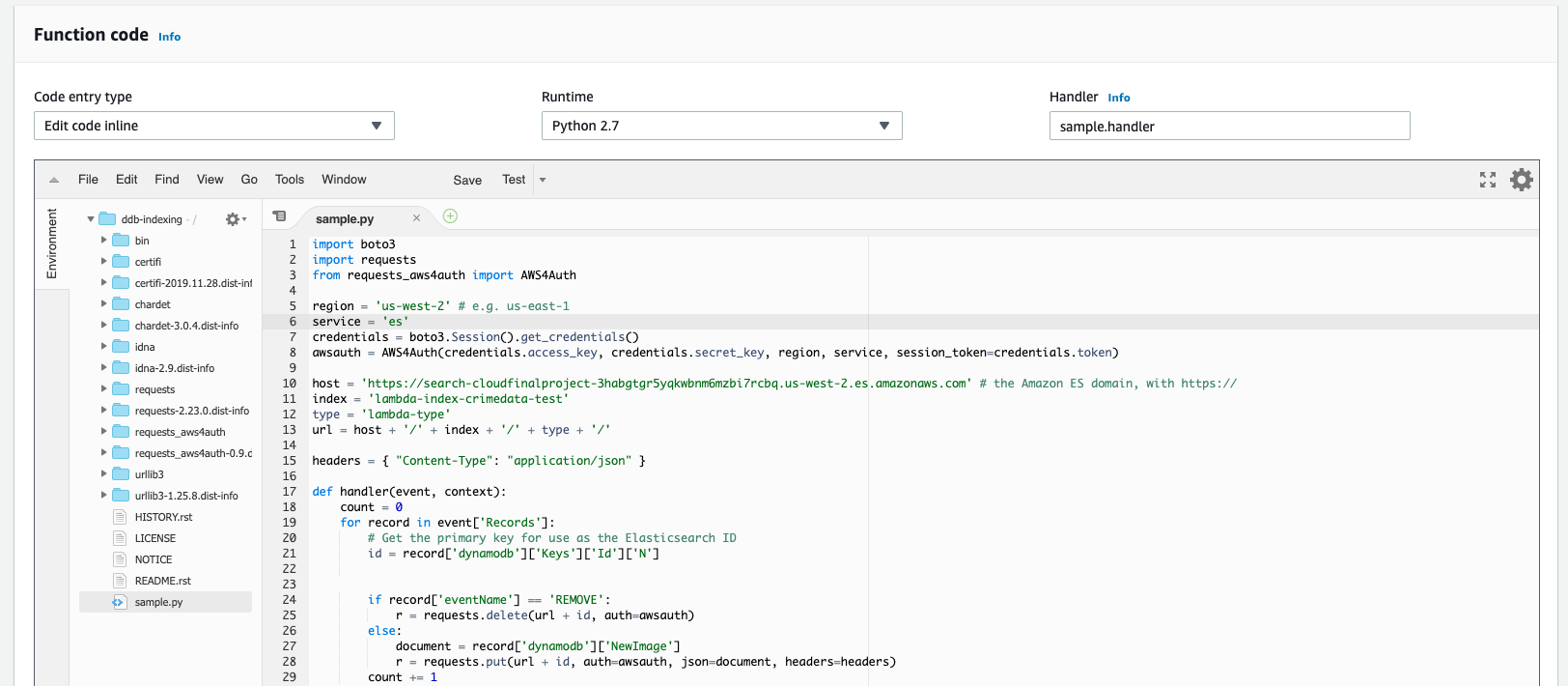
1. You will have to create an IAM role for the Lambda function you will create. Navigate to IAM (Identity and Access Management) from the AWS Management Console. From the left menu, click Policies. Create a new policy and call it “ddb-to-es-policy” and attach the following policy to it. (Copy paste into a text editor to create formatting issues)  
     
   {  
    “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”: “\*”  
    }  
    ]  
   }
2. Next, select Roles from the left menu and create a new role. Choose Lambda as the AWS service and attach the newly created ddb-to-es-policy to your role. Give you name a role. I named mine “ddb-to-es-role” and choose create role.

Create the Lambda Function

1. On your computer, create a directory and name it “ddb-to-es”. Place the included sample.py file into this directory. Open the sample.py file in a text editor of your choice and located the region and host variables near the top of the file. Add the region you are working in (with your s3 bucket) and add the endpoint for your elasticsearch service domain, which you receive after elasticsearch has finished setting up.
2. cd into the ddb-to-es directory and run the following commands to install your dependencies. Make sure to include the period at the end.  
     
   pip install requests -t .  
   pip install requests\_aws4auth -t .
3. Now create your lambda function. Navigate to Lambda from the AWS Management Console, and choose Create function. Choose Author from scratch. Add a function name, I named mine “bike-crime-data-indexing”. Choose Python 2.7, expand the Choose or create an execution role dropdown and use the role we created previously, I had named mine “ddb-to-es-role”. Then choose Create function.  
     
   A screenshot of a cell phone

   Description automatically generated
4. Now, that we have our function, we can setup our trigger. Click add trigger in the Lambda Designer and choose DynamoDB for the trigger. Select the BikeRacks table, set Batch size to be 100, and choose the starting position to be Trim Horizon. Make sure the Enable trigger option is checked, then choose Add to add the trigger.  
     
   A screenshot of a cell phone

   Description automatically generated
5. Now, you can add the files necessary for the lambda function to execute. Zip the contents inside the ddb-to-es directory (not the directory itself), and upload it to your Lambda function by choosing Upload a .ZIP file for the Code entry type. Change the Handler to be sample.handler. Choose save to save the changes.

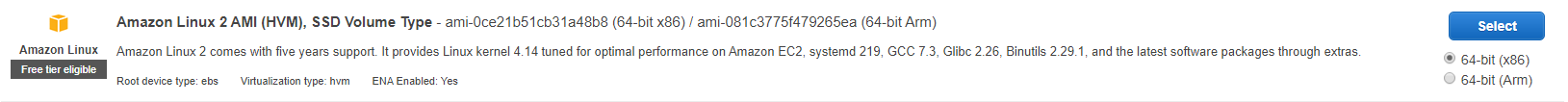


1. Now you have a Lambda function that listens for new entries into your DynamoDB Bikeracks table and loads it to your Amazon Elasticsearch Service. However, you table is already loaded, so to start the indexing, you must create a test event for a DynamoDB update event.
2. At the top of your lambda function screen, select Configure test event from the testing dropdown list. From the event template drop down, select DynamoDB Update. Give it a name and choose Create. Once it has been created, run the test.
3. Now that the data has been loaded successfully to Amazon Elasticsearch, you can verify that the data has been indexed to the lambda-index-crimedata-test index that was specified in the sample.py using Kibana or the Amazon ES console.  
   You can find the Kibana link in the Overview tab of your Amazon ES Service domain, and you can find the index in the ES console by looking into the indices tab. You can also make a standard GET request using an http client such as postman.  
     
   GET <es-endpoint>/lambda-index-crimedata-test/lambda-type/1

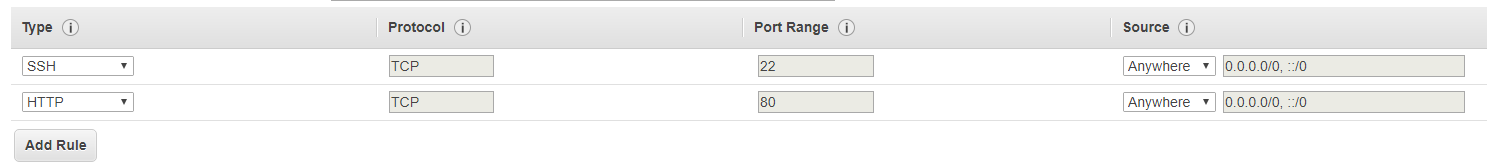
**Host the Web Application on an EC2 Instance**

Create an EC2 Instance

1. Select the Amazon Linux 2 AMI



1. Accept the default instance type (t2.micro), instance details and storage
2. Create a security group with ports 80 and 22 open allowing their source to be from anywhere



1. Launch the instance

Install Node.js on to the Instance

1. SSH into your instance
2. Install NVM with the following command:

curl -o- https://raw.githubusercontent.com/creationix/nvm/v0.33.2/install.sh | bash

1. exit the instance and SSH back into the instance
2. Run nvm –version to ensure NVM has been installed
3. Install the latest version of node using the following command:

nvm install node

Move Web Application’s Source Code into EC2 Instance

1. Download the bikeTheftDataWebApp.zip
2. Use the following command to transfer the zip into the EC2 Instance:

scp -i path/to/key zip/to/copy user@ec2-xx-xx-xxx-xxx.compute-1.amazonaws.com

1. Unzip the file using the following command:

unzip bikeTheftDataWebApp.zip

Start the Node Server

1. Move into the bikeTheftDataWebApp directory
2. Install PM2 with the following command:

npm install pm2 -g

1. Forward requests coming in on port 80 to port 3000 with the following command:

iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 80 -j REDIRECT --to-port 3000

1. Start the server using the following command:

pm2 start index.js

1. Enter the public DNS if the EC2 instance to see the web application

References

* <https://docs.aws.amazon.com/elasticsearch-service/latest/developerguide/es-gsg.html>
* <https://medium.com/@nishankjaintdk/setting-up-a-node-js-app-on-a-linux-ami-on-an-aws-ec2-instance-with-nginx-59cbc1bcc68c>
* <https://geodash.vpd.ca/opendata/>
* <https://opendata.vancouver.ca/explore/dataset/bike-racks/table/>