# Jupyter Notebook for combining Amazon Neptune and Amazon OpenSearch for geospatial queries

Purpose-built databases provide innovative ways to build data access patterns that cannot be solved otherwise. Many customers are looking to solve their business problems by storing and integrating data across a combination of purpose-built databases. For example, we can model highly connected geospatial data as a graph and store it in [Amazon Neptune](https://aws.amazon.com/neptune/). We can query such datasets quickly and at massive scale using a graph data model. Another purpose-built database, [Amazon OpenSearch Service](https://aws.amazon.com/elasticsearch-service/), can store geospatial data and provide powerful geo queries in addition to its full text search capabilities.

One of the features of the Amazon Neptune that makes it an attractive option for a wide variety of workloads and access patterns is the ease of integration with other AWS services. For example, Neptune can be integrated with OpenSearch by deploying a CloudFormation stack, as outlined in this [user guide](https://docs.aws.amazon.com/neptune/latest/userguide/full-text-search-cfn-create.html). In this blog we'll talk about using Amazon Neptune with Amazon OpenSearch utilizing its geospatial querying capabilities.

When it comes to building applications that rely on geospatial data, some of the most common customer use cases are:

* Given an entity in the dataset, find another entity in that dataset that is located the closest to it on the surface of the earth
* Given an entity and a geographical radius parameter, find all entities located within this radius of the given entity

Answering the first question in the graph context, where the entities in question are commonly connected via edges, typically does not present computational challenges since the set of the entities eligible for analysis is typically represented by the nodes in the graph directly connected to the starting node.  
  
Answering the second question without relying on a persistence layer with built-in geospatial radius query capabilities can become challenging, considering that it’s commonly factoring in all of the eligible entities in the graph.

## Solution

In this artifact we will look at the following steps to demonstrate the various approaches of geospatial querying using Amazon Neptune and Amazon Elasticsearch (OpenSearch):

1. Create fictitious geospatial data in the graph
   1. Use gremlin query to generate some fictitious data in the graph that has the geo-coordinates of a fictitious company with distribution centers and stores.
2. Calculate distance using haversine python library
   1. Use gremlin to get all stores a distribution center is connected to
   2. Calculate the distance using haversine for each of those stores
   3. Find the store closest to the distribution center
3. Find all stores within 1 mile radius from the distribution center
   1. Write a geo query on Elasticsearch using geo\_distance
4. Add fuzzy search on address, in addition to the geo query
   1. Add a query\_string search on the Elasticsearch query
5. Combine graph traversal and fuzzy search
   1. Traverse out from the distribution center and find all connected stores
   2. Do fuzzy search in gremlin using the gremlin integration with Elasticsearch

## Prerequisites

### You need an Amazon Neptune cluster to store the geospatial data in graph data model. You also need to provision a managed [Amazon SageMaker](https://aws.amazon.com/sagemaker/) notebook and attach it to the Neptune database cluster. You can follow the step-by-step instructions on how to configure this workload, including CloudFormation templates given in our documentation: Create Neptune cluster: <https://docs.aws.amazon.com/neptune/latest/userguide/get-started-create-cluster.html> Create SageMaker hosted notebook: <https://docs.aws.amazon.com/neptune/latest/userguide/graph-notebooks.html> Enable Neptune integration with OpenSearch:

Follow instructions in the [user guide](https://docs.aws.amazon.com/neptune/latest/userguide/full-text-search-cfn-create.html) to configure the integration. The guide will walk through the steps of using a prebuilt CloudFormation stack to setup the data synchronization from Amazon Neptune to Amazon ES.

Once you have the prerequisites taken care of, you can follow along the code samples in the Jupyter Notebook to do the geospatial queries.