# Lab 3 - Implement Geospatial Bike Searches, User Profile Caching, and Leaderboards with Amazon ElastiCache for Redis

1. Design modern applications on AWS NoSQL datastore – Amazon ElastiCache for Redis.
2. Implement Geo-Spatial search for available bikes in ElastiCache for Redis with DynamoDB as the primary data source.
3. Implement database cache for user profile with DocumentDB as the primary database.
4. Implement near real-time Monthly Distance Leaders leaderboard in ElastiCache for Redis.

Redis is an in-memory data structure store used as a database, cache, and message broker, known for its high performance and support for various data types.

Prerequisites

1. Working experience of Amazon ElastiCache for Redis, Amazon DynamoDB, and Amazon DocumentDB.
2. Working experience with AWS CLI, boto3 SDK and AWS SDK for Python.
3. Familiarity with Amazon Lambda.
4. Familiarity with ElastiCache for Redis data structures.
5. Familiarity of change data events in databases.
6. Knowledge of key/value and document databases.
7. Familiarity with database caching strategies.

**Task 1 - Connect to the Redis cluster**

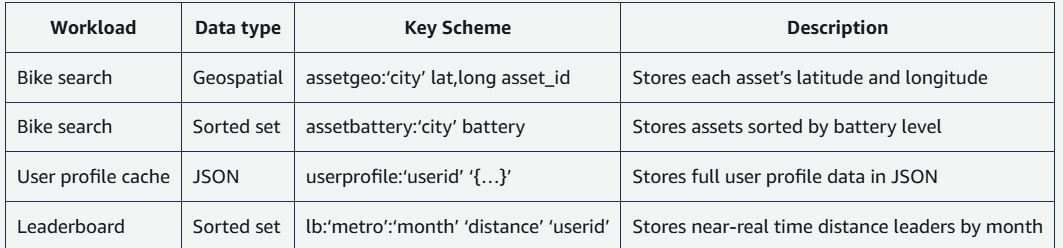
1.1 In cloud9, run

*cd lab3*

*conn-ecache*



**Task 2: Implement bike search access pattern**



2.1 Add sample bike data to the Geospatial data structure

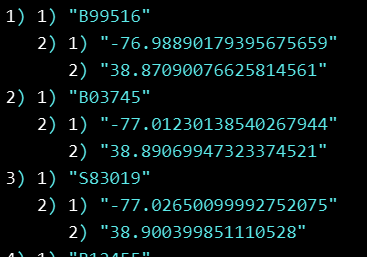
GEOADD <key> <longitude> <latitude> <asset\_id>

2.2 add multiple

GEOADD assetgeo:TEST -76.9889 38.8709 B99516 -76.9972 38.9042 S62978 -76.9885 38.9105 S73441 -77.0123 38.8907 B03745

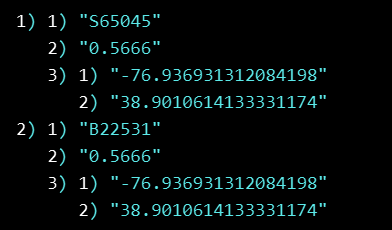
2.3 find bike within area

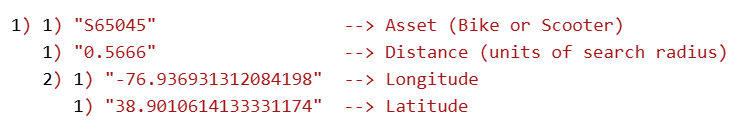
GEOSEARCH assetgeo:TEST FROMLONLAT -76 38 BYRADIUS 1000 mi WITHCOORD



2.4 search : city

GEOSEARCH assetgeo:DC FROMLONLAT -76.9272 38.9042 BYRADIUS 2 mi ASC COUNT 50 WITHDIST WITHCOORD





2.5 store this result using

*GEOSEARCHSTORE search:assetgeo:DC7698389 assetgeo:DC FROMLONLAT -76.9272 38.9042 BYRADIUS 2 mi STOREDIST*

**Task 3 - Store, Query assets by battery level**

3.1 add data

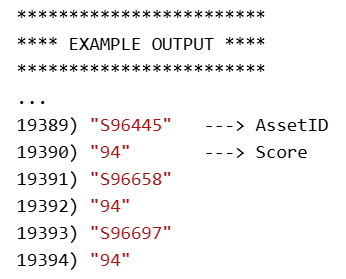
*ZADD <key> <battery> <assetid> [<battery> <assetid> ..]*

ex-ZADD assetbattery:TEST 50 A010 60 A100 70 A019 20 A0115 25 A0112 94 A0113 45 A0116 85 A012 89 A0119



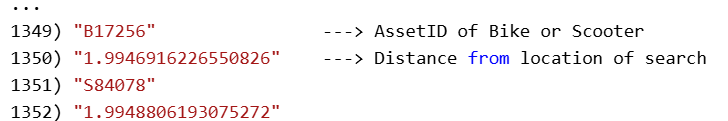
3.2 search

*ZRANGE search:assetbattery:DC:70 0 -1 WITHSCORES*



3.3 Combine search battery + location

*ZINTER 2 search:assetgeo:DC7698389 search:assetbattery:DC:70 WEIGHTS 1 0 AGGREGATE MAX WITHSCORES*



**Task 3: IWorking on user profile data**

3.1 user profile



3.2 add this user



Op- 

3.3 view

JSON.GET userprofile:U300000T



3.4 more playing around

*JSON.SET userprofile:U300000T $.preferences '{ "notifications" : {"frequency" : "WEEKLY", "mobile" : { "sms" : "false", "app" : "false" },"email" : "true" } }'*

3.5 delete profile

*JSON.DEL userprofile:U300000T*

**Task 4: Implement Monthly Distance Leaderboard**

Redis sorted set data type will be used to store the riders userid and distance covered. The sorted set command implements the leaderboard and rank functionality.

A Redis sorted set is a collection of unique strings (members) ordered by an associated score.

4.1 create a leaderboard, and add values

ZADD is a Redis command used to add members with a specific score to a sorted set.

*ZADD lb:dltest:DC:AUG 120 U1000 90 U2310 210 U3369 150 U5345 180 U2000 55 U9876 41 U8765 215 U92331 127 U2212 10 U9810 78 U8787 15 U92121*



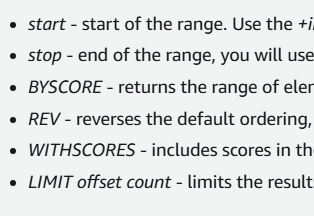
4.2 increase score by 5

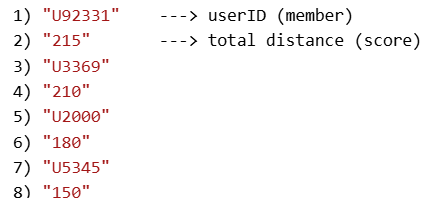
*ZINCRBY lb:dltest:DC:AUG 5 U1000*



4.3 to get the LB

*ZRANGE <key> <start> <stop> [BYSCORE|BYLEX] [REV] [WITHSCORES] [LIMIT offset count]*





4.4 Find rank of one user

*ZREVRANK lb:dltest:DC:AUG U92331*



Also given, UI for bike app

As per my best guess, this app is connected to DB operations by lambda functions

