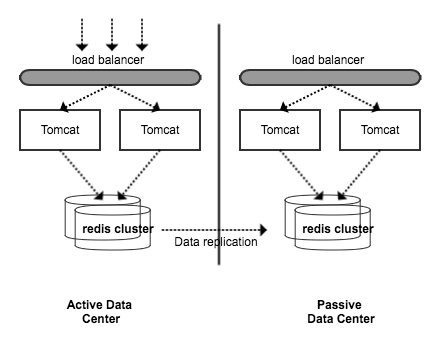
**Tech stack**

Java, Spring, Tomcat , Redis (as the datastore)

**Why Redis?**

Redis supports geospatial indexing. It is extremely fast & scalable. The use cases like storing locations & querying for locations in a specified radius is supported out of the box in redis

**Architecture**



**Deploy & Test**

Take a clean build (mvn clean install), the build will be successful if all tests are successful

1. Install redis and start redis server (./redis-server) 127.0.0.1:6379
2. Deploy the cabservice.war in tomcat . The application assumes that redis is available at 127.0.0.1:6379. If not change accordingly
3. <http://localhost:8080/cabservice/api/v1/> should return you hello world. It means the application is deployed successfully
4. <http://localhost:8080/cabservice/api/v1/populate> - This will populate 50000 location records
5. <http://127.0.0.1:8080/cabservice/api/v1/drivers?longitude=-2.114847768&latitude=57.14416516&radius=1000> - This would return some results (time taken ~ 20 ms)
6. Use POST <http://127.0.0.1:8080/cabservice/api/v1/drivers/2/location>
   1. And body {"longitude": 10,"latitude": 50} to play with the services

**Performance test**

There is a class called PerformanceTester which does the load test at API level.

The test will populate 50,000 location records and 20 threads each will update location and query operation for the specified number of iterations

**Running the performance tester:**

Run performanceTester with 20 threads and 1000 iterations

Output:

You will see following output in console:

*Starting PerformanceTester*

*Creating test data from uk\_postalcode\_locations.xlsx*

*50000 location records created*

*pool-1-thread-7 Number of read/writes 1000 time taken: 5030Ms*

*pool-1-thread-6 Number of read/writes 1000 time taken: 5215Ms*

*pool-1-thread-18 Number of read/writes 1000 time taken: 5236Ms*

*pool-1-thread-14 Number of read/writes 1000 time taken: 5648Ms*

*pool-1-thread-4 Number of read/writes 1000 time taken: 5728Ms*

*pool-1-thread-20 Number of read/writes 1000 time taken: 6017Ms*

*pool-1-thread-5 Number of read/writes 1000 time taken: 6086Ms*

*pool-1-thread-13 Number of read/writes 1000 time taken: 6082Ms*

*pool-1-thread-17 Number of read/writes 1000 time taken: 6086Ms*

*pool-1-thread-9 Number of read/writes 1000 time taken: 6375Ms*

*pool-1-thread-3 Number of read/writes 1000 time taken: 6388Ms*

*pool-1-thread-19 Number of read/writes 1000 time taken: 6367Ms*

*pool-1-thread-12 Number of read/writes 1000 time taken: 6425Ms*

*pool-1-thread-10 Number of read/writes 1000 time taken: 6465Ms*

*pool-1-thread-1 Number of read/writes 1000 time taken: 6484Ms*

*pool-1-thread-15 Number of read/writes 1000 time taken: 6610Ms*

*pool-1-thread-11 Number of read/writes 1000 time taken: 6739Ms*

*pool-1-thread-2 Number of read/writes 1000 time taken: 6748Ms*

*pool-1-thread-8 Number of read/writes 1000 time taken: 6805Ms*

*pool-1-thread-16 Number of read/writes 1000 time taken: 6789Ms*

*Cleaning up*

*Cleanup done*

Performance test result

Total APIs called = 20,000

Total time taken = 123,323 ms

**TPS = 162 Api calls per second**

References

<http://www.infoworld.com/article/3128306/application-development/build-geospatial-apps-with-redis.html>

<https://developer.ibm.com/clouddataservices/2016/11/16/blazingly-fast-geospatial-queries-with-redis/>