DZone

First Steps to Using Spring Boot and Cassandra

by Biju Kunjummen MVB · Apr. 25, 16 · Java Zone · Tutorial

If you want to start using Cassandra NoSQL database with Spring Boot, the best resource is likely the Cassandra samples available here and the Spring data Cassandra documentation.

Here I will take a little more roundabout way, by actually installing Cassandra locally and running a basic test against it and I aim to develop this sample into a more comprehensive example with the next blog post.

Setting up a Local Cassandra Instance

Your mileage may vary, but the simplest way to get a local install of Cassandra running is to use the Cassandra cluster manager(ccm) utility, available here.

```
1  ccm create test -v 2.2.5 -n 3 -s
```

Or a more traditional approach may simply be to download it from the Apache site. If you are following along, the version of Cassandra that worked best for me is the 2.2.5 one.

With either of the above, start up Cassandra, using ccm:

```
1  ccm start test
```

or with the download from the Apache site:

```
1  bin/cassandra -f
```

The -f flag will keep the process in the foreground, this way stopping the process will be very easy once you are done with the samples.

Now connect to this Cassandra instance:

```
bin/cqlsh
```

and create a sample Cassandra keyspace:

```
CREATE KEYSPACE IF NOT EXISTS sample WITH replication = {'class':'SimpleStrategy', 'replication_factor':1}
```

Using Spring Boot Cassandra

Along the lines of anything Spring Boot related, there is a starter available for pulling in all the relevant dependencies of Cassandra, specified as a gradle dependency here:

```
compile('org.springframework.boot:spring-boot-starter-data-cassandra')
```

This will pull in the dependencies that trigger the Auto-configuration for Cassandra related instances - a Cassandra session mainly.

For the sample I have defined an entity called the Hotel defined the following way:

```
package cass.domain;

import org.springframework.data.cassandra.mapping.PrimaryKey;
import org.springframework.data.cassandra.mapping.Table;

import java.io.Serializable;
import java.util.UUID;

@Table("hotels")
public class Hotel implements Serializable {

    private static final long serialVersionUID = 1L;

    @PrimaryKey
    private UUID id;

    private String name;
```

```
18
19      private String address;
20
21      private String zip;
22
23      private Integer version;
24
25      public Hotel() {
26      }
27
28      public Hotel(String name) {
29          this.name = name;
30      }
31
32      public UUID getId() {
33          return id;
34      }
35
36      public String getName() {
37          return this.name;
38      }
39
40      public String getAddress() {
41          return this.address;
42      }
43
44      public String getZip() {
45          return this.zip;
46      }
47
48      public void setId(UUID id) {
49          this.id = id;
50      }
```

```
51
52      public void setName(String name) {
53          this.name = name;
54      }
55
56      public void setAddress(String address) {
57          this.address = address;
58      }
59
60      public void setZip(String zip) {
61          this.zip = zip;
62      }
63
64      public Integer getVersion() {
65          return version;
66      }
67
68      public void setVersion(Integer version) {
69          this.version = version;
70      }
71
72  }
```

and the Spring data repository to manage this entity:

```
1  import cass.domain.Hotel;
2  import org.springframework.data.repository.CrudRepository;
3
4  import java.util.UUID;
5
6  public interface HotelRepository extends CrudRepository<Hotel, UUID>{}
```

A corresponding cql table is required to hold this entity:

```
1  CREATE TABLE IF NOT EXISTS  sample.hotels (
```

```
    id UUID,
    name varchar,
    address varchar,
    zip varchar,
    version int,
    primary key((id))
);
```

That is essentially it, Spring data support for Cassandra would now manage all the CRUD operations of this entity and a test looks like this:

```
import cass.domain.Hotel;
import cass.repository.HotelRepository;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.SpringApplicationConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

import java.util.UUID;

import static org.hamcrest.MatcherAssert.assertThat;
import static org.hamcrest.Matchers.equalTo;

@RunWith(SpringJUnit4ClassRunner.class)
@SpringApplicationConfiguration(classes = SampleCassandraApplication.class)
public class SampleCassandraApplicationTest {

 @Autowired
 private HotelRepository hotelRepository;

 @Test
 public void repositoryCrudOperations() {
  Hotel sample = sampleHotel();
  this.hotelRepository.save(sample);
```

```
24    this.hotelRepository.save(sample);
25
26    Hotel savedHotel = this.hotelRepository.findOne(sample.getId());
27
28    assertThat(savedHotel.getName(), equalTo("Sample Hotel"));
29
30    this.hotelRepository.delete(savedHotel);
31   }
32
33   private Hotel sampleHotel() {
34    Hotel hotel = new Hotel();
35    hotel.setId(UUID.randomUUID());
36    hotel.setName("Sample Hotel");
37    hotel.setAddress("Sample Address");
38    hotel.setZip("8764");
39    return hotel;
40   }
41
42  }
```

Here is the github repo with this sample. There is not much to this sample yet, in the next blog post I will enhance this sample to account for the fact that it is very important to understand the distribution of data across a cluster in a NoSQL system and how the entity like Hotel here can be modeled for efficient CRUD operations.

Like This Article? Read More From DZone

The Plumbing Included With Auto-

This Week in Spring

Configuration in Spring Data Cassandra

Spring Boot 1.3 Release Has New Dev Tools, Caches, and Annotations

Free DZone Refcard

Java Containerization

Topics: JAVA , SPRING-BOOT , SPRING , CASSANDRA