Quick-start spring rest-api with Cassandra

[](https://medium.com/@ankitkumargupta?source=post_page-----e112700936cf----------------------)

[Ankit Kumar Gupta](https://medium.com/@ankitkumargupta?source=post_page-----e112700936cf----------------------)

Follow

[Jan 18, 2019](https://medium.com/@ankitkumargupta/quick-start-spring-rest-api-with-cassandra-e112700936cf?source=post_page-----e112700936cf----------------------) · 4 min read

https://miro.medium.com/max/60/1*b0PBVfcSG82_f6mmBpn-Tw.png?q=20



With end of this tutorial, you would be able to do CRUD operations with Cassandra using rest-api and use Cassandra shell. Also you can find references in the text for detailed study.

**Tools**

* any IDE like [eclipse](https://www.eclipse.org/downloads/), [intellij](https://www.jetbrains.com/idea/download/" \l "section=mac" \t "_blank) or [Spring tool suite](https://spring.io/tools3/sts/all).
* Java: [JDK 1.8](https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html)
* Database: [Cassandra](https://cassandra.apache.org/doc/latest/getting_started/index.html)

**Dependencies**

* spring-boot-starter-web — [Tomcat](https://tomcat.apache.org/) support, to host our app, spring REST MVC support
* [lombok](https://www.baeldung.com/intro-to-project-lombok) — remove a lot of boiler plate code using annotations
* [cassandra](https://docs.spring.io/spring-data/cassandra/docs/current/reference/html/)— provides integration with the Cassandra database

**Complete code:**

[Download here](https://github.com/itkan/spring-samples/tree/master/restapi-cassandra)

**Cassandra DB**

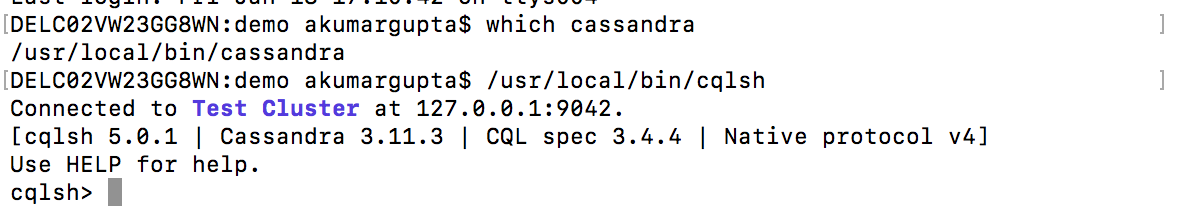
//On Terminal  
//[Installation](https://www.javatpoint.com/how-to-install-cassandra-on-mac)  
brew install cassandra//To have launchd start cassandra now and restart at login:  
brew services start cassandra//Stop Cassandra  
brew services stop cassandra//if you don’t want/need a background service you can just run:  
cassandra -f

**Cassandra Shell — CQL**

You would need to create keyspace before running your rest-api server, as it connects on start. Also CQL is used to perform operations directly on cassandra.

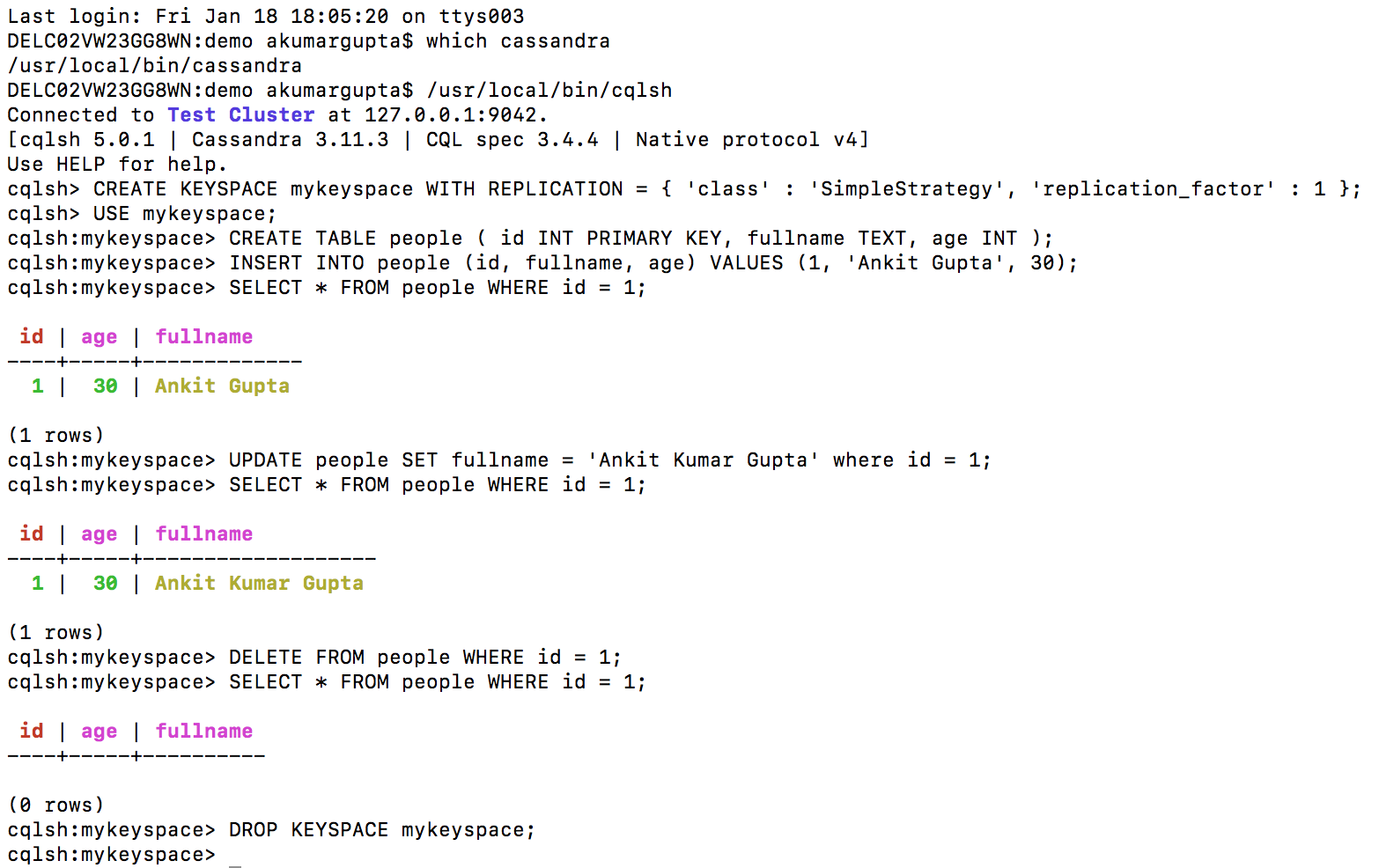
once you have installed cassandra find the path for cassandra using `**which cassandra`** command on terminal. and at same level you will find **cqlsh**

https://miro.medium.com/max/60/1*E13DE9VB-1-P-uGkJGzn0w.png?q=20



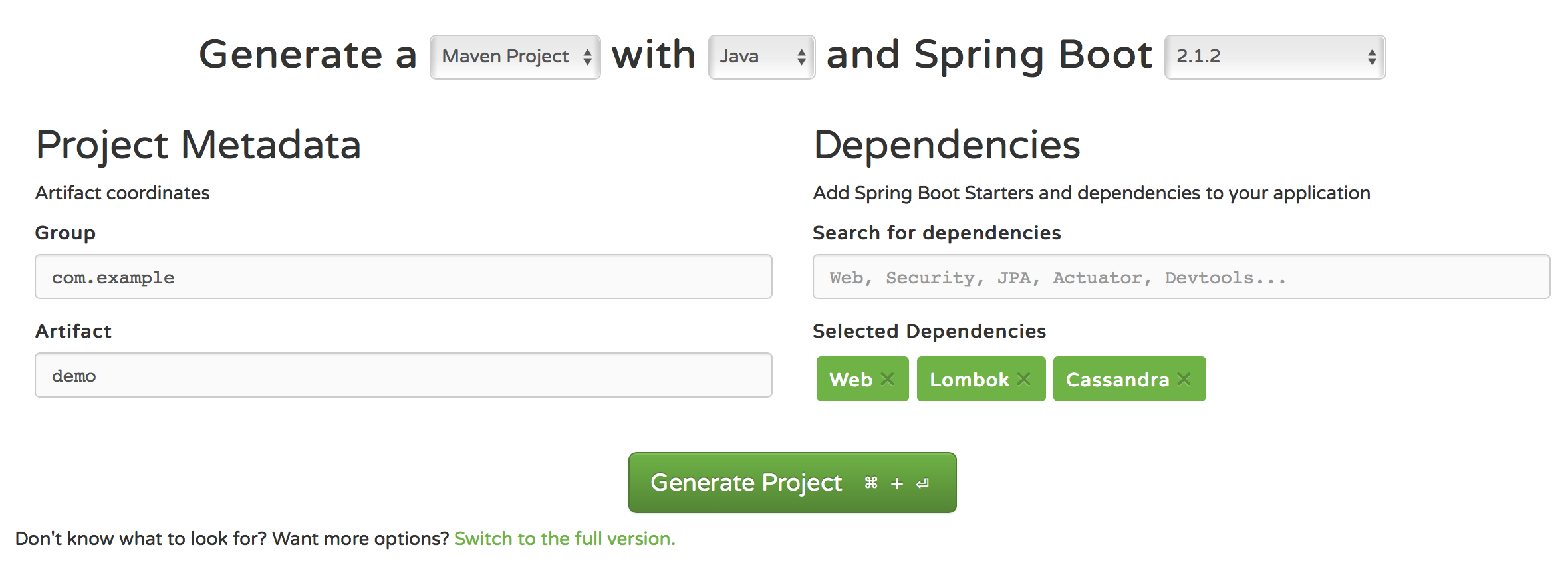
//On Terminal  
/usr/local/bin/cqlsh//then run CQL commands, learn more [here](https://docs.datastax.com/en/cql/3.3/cql/cql_reference/cqlCommandsTOC.html)//**create keyspace needed for running the sample**  
CREATE KEYSPACE mykeyspace WITH REPLICATION = { 'class' : 'SimpleStrategy', 'replication\_factor' : 1 };USE mykeyspace;//CRUD operations  
  
CREATE TABLE people ( id INT PRIMARY KEY, fullname TEXT, age INT );INSERT INTO people (id, fullname, age) VALUES (1, 'Ankit Gupta', 30);SELECT \* FROM people WHERE id = 1;UPDATE people SET fullname = 'Ankit Kumar Gupta' where id = 1;DELETE FROM people WHERE id = 1;DROP KEYSPACE mykeyspace;

https://miro.medium.com/max/60/1*ASnRG7nxPRWZfr0YGVtlmw.png?q=20



**Step 1:**Goto [https://start.spring.io](https://start.spring.io/) and generate a maven project with Web, lombok and cassandra dependencies.

https://miro.medium.com/max/60/1*NrHRK0ak_EWW3PQWdQEdzg.png?q=20



**Step 2:** Goto this downloaded folder and run this project to see everything is setup correctly.

cd /Downloads/demo  
mvn spring-boot:run

https://miro.medium.com/max/60/1*OcHLRvEpLCt7UnMq3JZHNA.png?q=20



**Step 3:** Add Employee.java file. it is resource entity which also defines table in db

package com.example.demo.model;import org.springframework.data.cassandra.core.mapping.PrimaryKey;  
import org.springframework.data.cassandra.core.mapping.Table;import lombok.AllArgsConstructor;  
import lombok.Getter;  
import lombok.NonNull;  
import lombok.Setter;@AllArgsConstructor  
@Getter @Setter  
@Table  
public class Employee {  
 @PrimaryKey   
 private @NonNull String id;  
 private @NonNull String firstName;  
 private @NonNull String lastName;  
 private @NonNull String email;  
}

**Step 4:** Add EmployeeRepository.java, this will provide default implementations of actions possible with database.

package com.example.demo.repository;import org.springframework.data.repository.CrudRepository;import com.example.demo.model.Employee;public interface EmployeeRepository extends CrudRepository<Employee, String> {  
}

**Step 5:** Add EmployeeController.java, this will integrate our rest api with repository and perform the CRUD operations.

package com.example.demo.controller;import java.util.ArrayList;  
import java.util.List;  
import java.util.Optional;  
import java.util.Random;import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.web.bind.annotation.\*;import com.example.demo.model.Employee;  
import com.example.demo.repository.EmployeeRepository;@RestController  
public class EmployeeController  
{  
 @Autowired  
 EmployeeRepository employeeRepository;@GetMapping(value = "/healthcheck", produces = "application/json; charset=utf-8")  
 public String getHealthCheck()  
 {  
 return "{ \"isWorking\" : true }";  
 }@GetMapping("/employees")  
 public List<Employee> getEmployees()  
 {  
 Iterable<Employee> result = employeeRepository.findAll();  
 List<Employee> employeesList = new ArrayList<Employee>();  
 result.forEach(employeesList::add);  
 return employeesList;  
 }@GetMapping("/employee/{id}")  
 public Optional<Employee> getEmployee(@PathVariable String id)  
 {  
 Optional<Employee> emp = employeeRepository.findById(id);  
 return emp;  
 }@PutMapping("/employee/{id}")  
 public Optional<Employee> updateEmployee(@RequestBody Employee newEmployee, @PathVariable String id)  
 {  
 Optional<Employee> optionalEmp = employeeRepository.findById(id);  
 if (optionalEmp.isPresent()) {  
 Employee emp = optionalEmp.get();  
 emp.setFirstName(newEmployee.getFirstName());  
 emp.setLastName(newEmployee.getLastName());  
 emp.setEmail(newEmployee.getEmail());  
 employeeRepository.save(emp);  
 }  
 return optionalEmp;  
 }@DeleteMapping(value = "/employee/{id}", produces = "application/json; charset=utf-8")  
 public String deleteEmployee(@PathVariable String id) {  
 Boolean result = employeeRepository.existsById(id);  
 employeeRepository.deleteById(id);  
 return "{ \"success\" : "+ (result ? "true" : "false") +" }";  
 }@PostMapping("/employee")  
 public Employee addEmployee(@RequestBody Employee newEmployee)  
 {  
 String id = String.valueOf(new Random().nextInt());  
 Employee emp = new Employee(id, newEmployee.getFirstName(), newEmployee.getLastName(), newEmployee.getEmail());  
 employeeRepository.save(emp);  
 return emp;  
 }  
}

**Step 6:** update application.properties to have cassandra properties

//server port  
server.port = 8082//cassandra properies  
spring.data.cassandra.keyspace-name=mykeyspace  
spring.data.cassandra.contact-points=localhost  
spring.data.cassandra.port=9042  
spring.data.cassandra.schema-action=create\_if\_not\_exists

Now run your project again as step 2 and perform CRUD operations.

#get all employees  
curl -X GET \  
 http://localhost:8082/employees \  
 -H 'Cache-Control: no-cache'#get employee by ID  
curl -X GET \  
 http://localhost:8082/employee/729280953 \  
 -H 'Cache-Control: no-cache'#update employee by ID  
curl -X PUT \  
 http://localhost:8082/employee/729280953 \  
 -H 'Cache-Control: no-cache' \  
 -H 'Content-Type: application/json' \  
 -d '{  
 "id": "729280953",  
 "firstName": "ankit",  
 "lastName": "gupta",  
 "email": "testankit@gmail.com"  
}'#delete employee by ID  
curl -X DELETE \  
 http://localhost:8082/employee/980694165 \  
 -H 'Cache-Control: no-cache' \  
 -H 'Content-Type: application/x-www-form-urlencoded'#create employee  
curl -X POST \  
 http://localhost:8082/employee \  
 -H 'Cache-Control: no-cache' \  
 -H 'Content-Type: application/json' \  
 -d '{  
 "id": 3,  
 "firstName": "ankit",  
 "lastName": "gupasd1",  
 "email": "test@gmail.com"  
}'