Spring Data Cassandra & DataStax

1. Open Questions

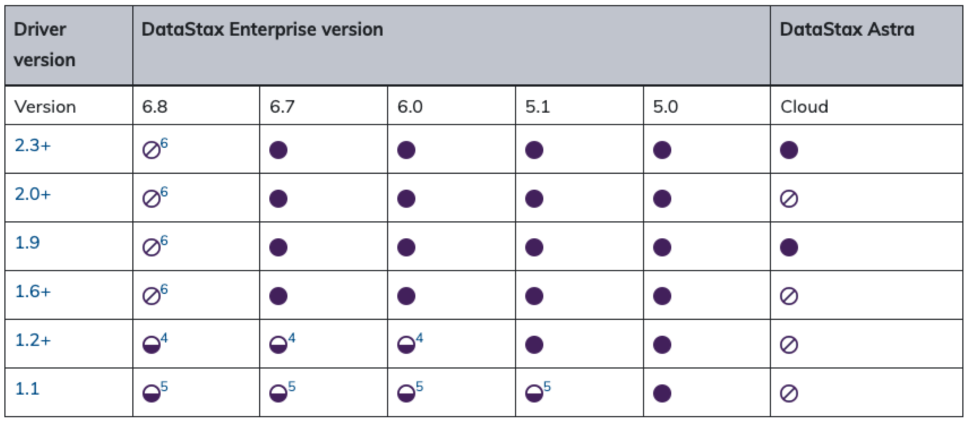
Is Spring Data Cassandra supported?

Spring Data Cassandra (SDC) is not a framework implemented nor maintained by Datastax as such we cannot commit to fixes or needed patches. **You should not open tickets if you hit SpringFramework errors.** Still, **you can use Spring Data with a supported version of the Datastax drivers** (detailed below)

- Resources for anything related to the drivers:
  - Open tickets on JIRA
  - Drivers mailing list
- For anything related to Spring Data:
  - Spring Data documentation
  - You can reach out to us at community.datastax.com (best effort, no SLA)
  - You can contact the spring Data team on gitter
  - You can open tickets on the github of the project
  - We also monitor stack overflow

if so, what version(s)?

Source: Table of compatibility of the Java Drivers

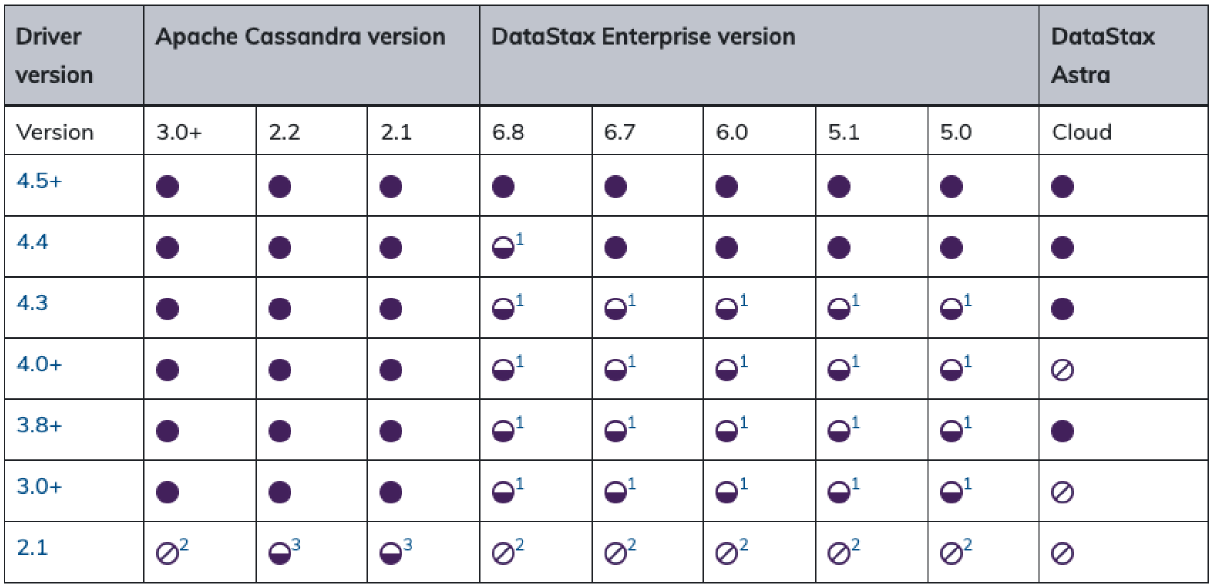


Driver version	DataStax Enterprise version					DataStax Astra
Version	6.8	6.7	6.0	5.1	5.0	Cloud
2.3+	⊘[6]	●	●	●	●	●
2.0+	⊘[6]	●	●	●	●	⊘
1.9	⊘[6]	●	●	●	●	●
1.6+	⊘[6]	●	●	●	●	⊘
1.2+	◒[4]	◒[4]	◒[4]	●	●	⊘
1.1	◒[5]	◒[5]	◒[5]	◒[5]	●	⊘

The table above is specific to DSE features. (the one that has been qualified as 1.8.2 in your email). Latest is 2.4.0 (2020/01/14).

Unified Drivers

Starting with 4.4, drivers **now p**rovide both Cassandra and DSE features in an UNIFIED library (you do not need anymore DSE specific libraries for DSE advanced workloads)



Driver version	**Apache Cassandra version**			**DataStax Enterprise version**					**DataStax Astra**
Version	3.0+	2.2	2.1	6.8	6.7	6.0	5.1	5.0	Cloud
4.5+	●	●	●	●	●	●	●	●	●
4.4	●	●	●	◒[1]	●	●	●	●	●
4.3	●	●	●	◒[1]	◒[1]	◒[1]	◒[1]	◒[1]	●
4.0+	●	●	●	◒[1]	◒[1]	◒[1]	◒[1]	◒[1]	⊘
3.8+	●	●	●	◒[1]	◒[1]	◒[1]	◒[1]	◒[1]	●
3.0+	●	●	●	◒[1]	◒[1]	◒[1]	◒[1]	◒[1]	⊘
2.1	⊘[2]	◒[3]	◒[3]	⊘[2]	⊘[2]	⊘[2]	⊘[2]	⊘[2]	⊘

Learn more about driver v4 :

4.0 released in 2019, latest release 4.10.0, Asynchronous, non-blocking engine, Execution profiles, Global timeouts, Improved load balancing policy , Improved metrics, Improved Object, mapper.

Update guide: docs.datastax.com/en/developer/java-driver/latest/upgrade_guide
Code samples: https://github.com/DataStax-Examples/java-cassandra-driver-from3x-to4x

Table of Compatibility Matrix ⇔ Spring Data ⇔ Spring Boot

DataStax Drivers	**Spring-Data**	**Spring Boot**
OSS Drivers 3.x and older DSE Drivers 2.4 and older	2.2 and older	2.2 and older
4.x	3.0 and newer	2.3 and newer

We recommend moving to unified drivers (4x) using Spring Data 3x and Spring Boot 2.3+

Is using JPA with no developer-built queries supported or should development teams be using something like CassandraTemplate and supply their own queries?

CassandaRepositories and PagingAndSorting repositories fixed a lot of previous but cannot implements everything. Using SimpleCassandraRepository gives you access to the `CqlSession` when needed to performs advanced features.

We recommend using your SimpleCassandraRepositories to keep access to CqlSession when needed.

For configuration, you should define your own bean CqlSession to get full control of the configuration and do not rely on too limited options offered in the Spring-Boot starter and `CassandraConfiguration`.

We recommend defining your CqlSession bean yourself with custom configuration. (application.conf)

Known "Issues" (and solutions) with Spring Data

(Even) using Spring Data Cassandra you need to know what Cassandra is and how it works: Data modeling principles, denormalization, data replications.

- **Tombstones:** The most dangerous issue with using any object mapping framework is that it exposes the ability to insert `null` values to the database, which leads to tombstones in DSE and Cassandra. In SDC, `null` values are skipped on insert but the `null` value behavior remains for updates (stackoverflow reference, Spring Data docs, DATACASS-420).

This issue is now fixed in Spring Data Cassandra 3,0 and backported in 2.x.

- **Batches**: Batches LOGGED batches are used exclusively in SDC with an intent to promote atomicity and there is no option to use UNLOGGED batches. There are no built-in preventions for adding statements to the batch that have distinct partition keys (multi-partition batches) which is an anti-pattern in DSE and Cassandra. For these batches a com.datastax.driver.core.querybuilder.Batch instance is generated (creates a regular QUERY message at protocol level) instead of the more efficient alternative com.datastax.driver.core.BatchStatement. Similarly, there is no support for explicitly setting idempotence in the batch implementation or for any other query types (DATACASS-454), this can be supplied per Statement if using the DataStax Drivers directly.

When BatchStatements is needed use the CqlSession directly.

- **Exposes Anti-Patterns methods:** Methods such as findAll are exposed in the Spring Data Cassandra API and in general abstracting the capabilities of the DataStax Drivers guides users towards transparent misuse.

This is still true. NEVER use findAll within a Cassandra Table this is dangerous. In the best-case scenario your request is slow, most of the time **you will get an out-of-memory error.**

- **Create non-performant tables and schema:** There is a concept of automatic schema generation and deletion that can cause issues if mis-used. Specifically, the user can choose among different strategies when the application starts, some of them being able to create or update the entire CQL schema by simply reading the Java model. This is intended to get users off the ground but can have serious consequences if used in production as concurrent schema modifications and topology changes are a delicate procedure in DSE and Cassandra.

It is not recommended to enable the AUTOCREATE schema option. With Cassandra you should define your schema and tables **first,** based on your queries, **and only then** create associated entities not the other way around.

- **Exposes Anti-Patterns Data modeling principle (reusing tables)**: Proper way to design a data model with Cassandra is **one table per query** even if it implies duplicating the same data in different tables. The object-first approach may lead the developers to reuse the same entities (and table) for different queries needing the introduction of secondary indices and even worse ALLOW FILTERING.

It is not recommended to use the same table for different queries that do not match the primary key design. `CassandraBatchOperations` or go the `CqlTemplate` route can help you here: SAMPLE

- **Multiple sessions when using multiple keyspaces**: when using multiple keyspaces in your application you will need multiple CassandraTemplate beans, multiple DataStax Driver Session objects are created that will create unnecessary connections to your DSE or Cassandra hosts. Through using the DataStax Drivers API directly, you have the capability to specify a keyspace per query in C* protocol v5 (DSE 6) and to more easily supply the keyspace in your query string for versions before DSE 6. Similarly, the DataStax Driver Object Mapper allows you to specify the keyspace when creating the Mapper object and this will work within the same Session.

Multiple keyspaces required multiple `CqlSession` and the use of `@Qualifier` at spring level.

- **Limited Metadata access:** The metadata capabilities are limited for both the `Token` (limited to `describeRing`) and `Host` metadata (limited to `RingMember`) and there is no support at all for `Schema` metadata. See the extensive metadata support in the DataStax Drivers for comparison.

- **Decreased performance**: No support for prepared statements

Before the very last version of the spring Data the statements where not prepared mean a performance overhead at drivers level to parse and validate each query. Check `PreparedStatementCreator`

===

How to use DSE Drivers with Spring Data Cassandra (before unified drivers)

By default SDC only supports C* workloads and there is no support for other DSE advanced features such as DataStax Graph and DSE Advanced Security (LDAP, Kerberos). This introduces additional complexity if you need to run other DSE Advanced Workloads in the same application. Also, there is no `CassandraSessionFactoryBean` compatible with `DseSession`, so one needs to use the "manual" approach outlined below to create a `DseSession`.

pom.xml

```
<properties>
  <dse-java-driver.version>1.8.1</dse-java-driver.version>
</properties>
```

```xml
<dependencies>
  <dependency>
     <groupId>org.springframework.boot</groupId>
     <artifactId>spring-boot-starter-data-cassandra</artifactId>
     <exclusions>
        <exclusion>
           <groupId>com.datastax.cassandra</groupId>
           <artifactId>cassandra-driver-core</artifactId>
        </exclusion>
     </exclusions>
  </dependency>

  <dependency>
     <groupId>com.datastax.dse</groupId>
     <artifactId>dse-java-driver-core</artifactId>
     <version>${dse-java-driver.version}</version>
  </dependency>

</dependencies>
```

DseDriverConfiguration.java

```java
import com.datastax.driver.dse.DseCluster;
import com.datastax.driver.dse.DseSession;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;


@Configuration
public class DseDriverConfiguration {
   @Value("${dse.contactPoints}")
   public String contactPoints;

   @Value("${dse.port}")
   private int port = 9042;

   @Value("${dse.keyspace}")
   private String keyspace = "user_ks";

   public String getContactPoints() {
       return contactPoints;
   }

   public String getKeyspaceName() {
       return keyspace;
   }

   public int getPort() {
       return port;
   }

   @Bean
   public DseCluster dseCluster() {
       DseCluster.Builder dseClusterBuilder =
               DseCluster.builder()
                       .addContactPoints(contactPoints)
                       .withPort(port)
```

```
        return dseClusterBuilder.build();
    }

    @Bean
    public DseSession dseSession(DseCluster dseCluster) {

        return dseCluster.connect(keyspace);
    }

}
```

Best practices

Make sure that you're passing Cluster and Session objects as singletons
A common mistake we see customers make is to recreate the Cluster and Session objects on every call into their Repository classes. For instance:

Avoid reads-before-writes when updating entities
A typical Spring Data repository will have a findByKey() method and a Save() method.

If you have a need in your application to update an object by ID, you might be tempted to do something like this:

```
Widget widget = findByKey("key1");
widget.property = "foobar";
save(widget);
```

However, this approach will do a read before write. It would be better to create an explicit method (perhaps updatePropertyByKey()) that avoids the read before write.

Avoid tombstones on inserts
In Cassandra if you explicitly insert a NULL value when doing an INSERT, Cassandra will store that NULL as a tombstone. For instance:

```
INSERT INTO table ( partition_key, clustering_key, regular_property)VALUES (
"value1", "value2", NULL); //this NULL will become a tombstone
```

This kind of mistake is easy to find in explicit driver code, but gets harder to detect when abstracted away by Spring Data.

It is often better to have an explicit Insert() method alongside the Save() method. For these kinds of cases, you may find it necessary to bypass Spring's "automagic" wiring/mapping and create explicit CQL:

```
String cql = "INSERT INTO table ( partition_key, clustering_key,
regular_property)VALUES ( '" + value1 + "', '" value2 + "' );";
        cassandraTemplate().getCqlOperations().execute(cql);
```

Reference:
https://docs.spring.io/spring-data/cassandra/docs/2.1.1.RELEASE/reference/html/#cassandra.template.insert-update

Spring Data doesn't use prepared statements by default, but it can be coerced to do so
In normal CQL Driver usage, the use of prepared statements is encouraged since it allows the re-use of execution plans by the DSE servers and decreases operational latency.

Spring Data can be coerced into using prepared statements (see Chris Bradford's work below); however, by the time you've gone to the trouble of setting up these prepared statements, you could have probably coded the whole thing using direct driver code.

Reference: https://github.com/bradfordcp/dse-spring-boot-demo/

Samples codes

- https://github.com/datastaxdevs/workshop-spring-data-cassandra
- https://github.com/DataStax-Examples/spring-data-starter
- https://github.com/DataStax-Examples/spring-k8s-cassandra-microservices
- https://github.com/clun/spring-petclinic-reactive
- https://github.com/clun/spring-data-test-preparedstatement
- https://github.com/clun/spring-data-test-preparedstatement/blob/main/spring-data-test/src/main/java/com/datastax/springdatatest/todo/TodoRepository.java