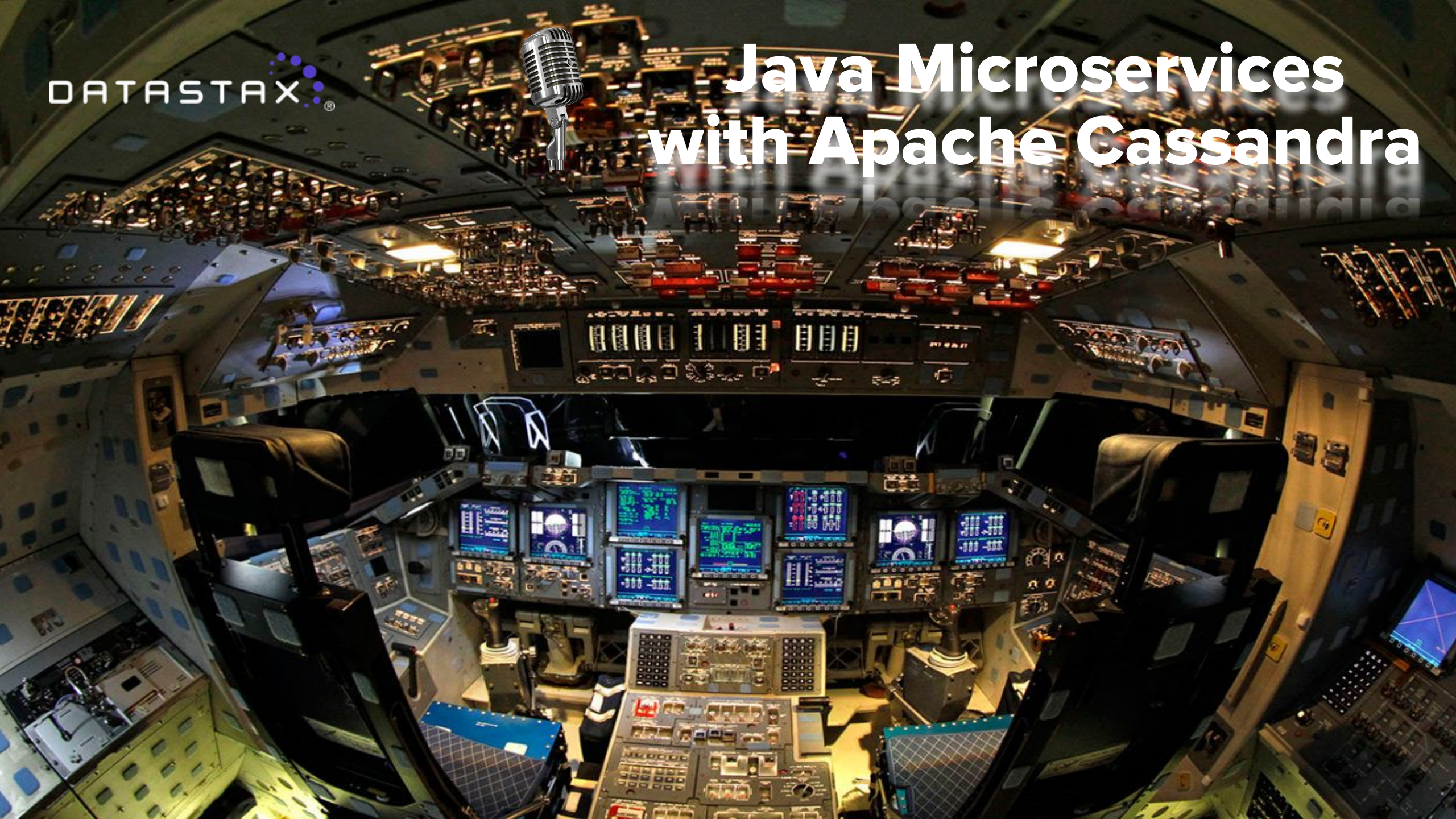
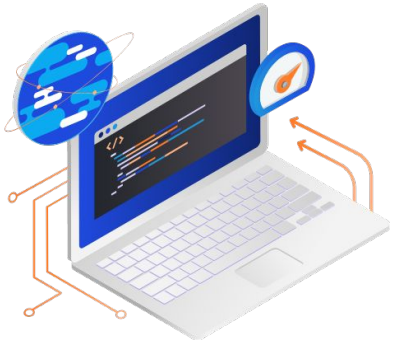
DATASTAX

Workshop online Series

Build Java Microservices for Apache Cassandra™


DATASTAX
Java Microservices
with Apache Cassandra



Your hosts

Cédrick Lunven 🇫🇷

Developer Advocate

Eric Zietlow 🇺🇸

Developer Advocate

DATASTAX DEVELOPERS

Jack Fryer 🇬🇧

Community Manager

David Jones-Gilardi 🇺🇸

Developer Advocate

Aleksandr Volochnev 🇷🇺

Developer Advocate

DATASTAX

Spring Microservices with Apache Cassandra™

1. Housekeeping
2. Running the TodoApp
3. Connectivity to Cassandra
4. Create the Repository (dao)
5. Spring-Data & Spring-Data-Rest
6. Resources

DATASTAX


menti.com
47 86 87
Mentimeter
What is the most desirable feature of your ideal workplace?
Flexible time
Social factor
Big paycheck
Available on the iPhone
App Store
GET IT ON
Google play

Spring Microservices with Apache Cassandra™

1. Housekeeping
2. Running the TodoApp
3. Connectivity to Cassandra
4. Create the Repository (dao)
5. Spring-Data & Spring-Data-Rest
6. Resources

DATASTAX

Disclaimer

Today, with limited time (2H), we won't introduce the basics of Apache Cassandra™.

This is a coding session. You would need a previous experience with Java language and the following tools :

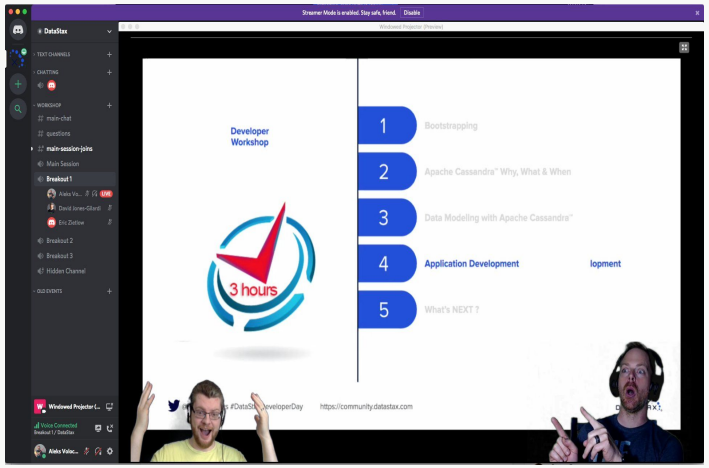
LOCAL Development

- ★ JAVA (JDK8 +)
- ★ Maven
- ★ Git
- ★ an IDE

OR

GITPOD

STREAMS



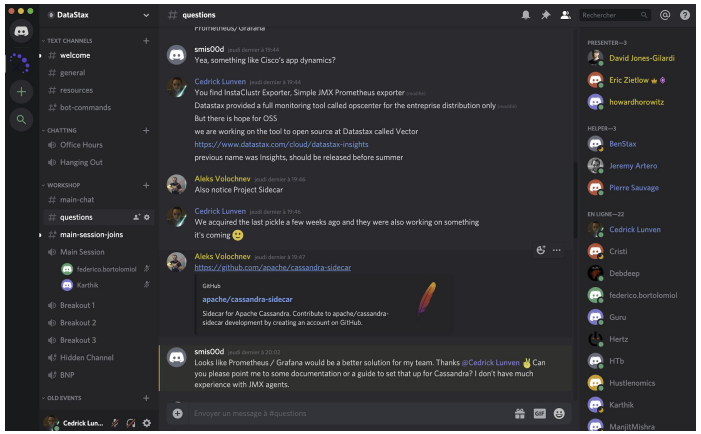
YouTube

Twitch

RUNTIME

Gitpod

QUESTIONS



Discord

MATERIALS

Slides

Notebooks

DockerFile

Github

Screen with youtube, discord, Astra & IDE

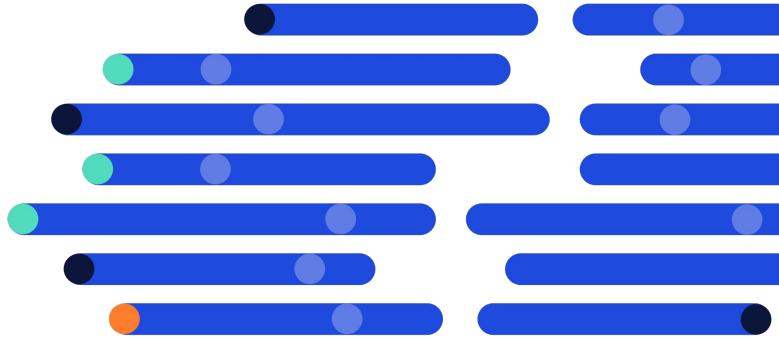
Exercises

https://github.com/DataStax-Academy/microservices-java-workshop-online

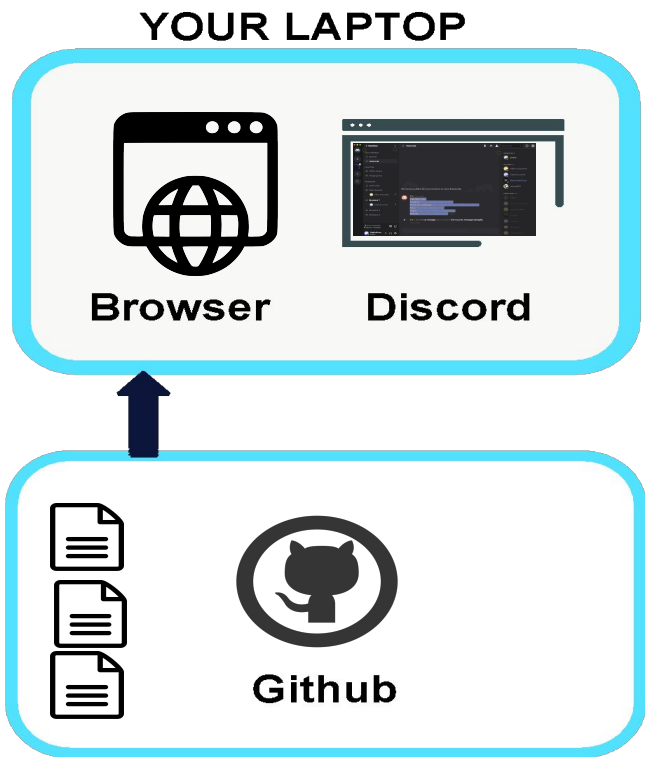
✨ Building a Microservices for Apache Cassandra™ ✨

Gitpod ready-to-code | license Apache-2.0 | chat 85 online

In May 2020, the Spring engineering teams released Spring Boot 2.3.0 and Spring Data 3.0.0 that brought an upgrade to the latest version of the Cassandra Java driver. This driver upgrade provides full support for the newest major version of



Bootstrapping



1. **Clone or download repository material**

```
https://github.com/DataStax-Academy/
microservices-java-workshop-online
```

Bootstrapping your env 2/2

DATASTAX

3. Import the maven project in your IDE or open GitPod



4. Start Cassandra locally

```
docker-compose up -d
```

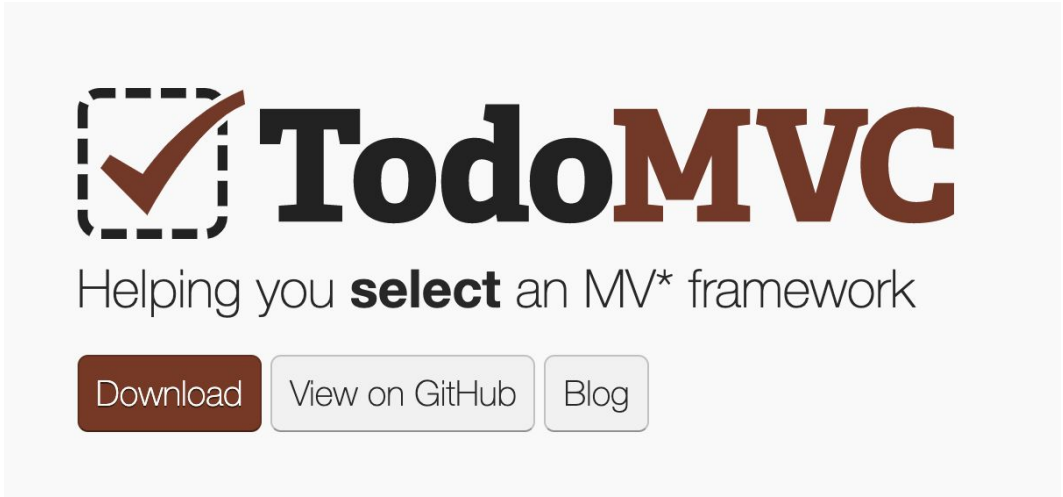
Spring Microservices with Apache Cassandra™

1. Housekeeping
2. Running the TodoApp
3. Connectivity to Cassandra
4. Create the Repository (dao)
5. Spring-Data & Spring-Data-Rest
6. Resources

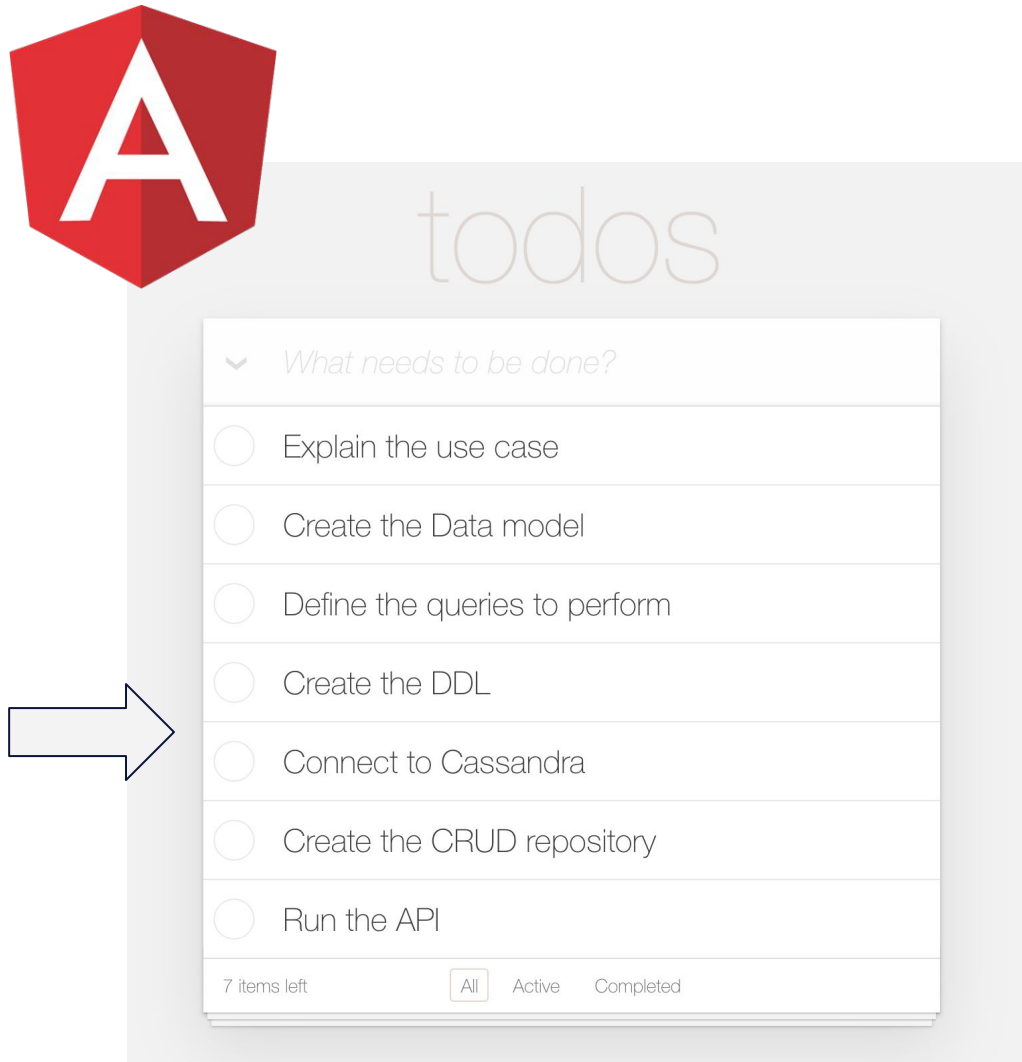
DATASTAX

The Todo Application

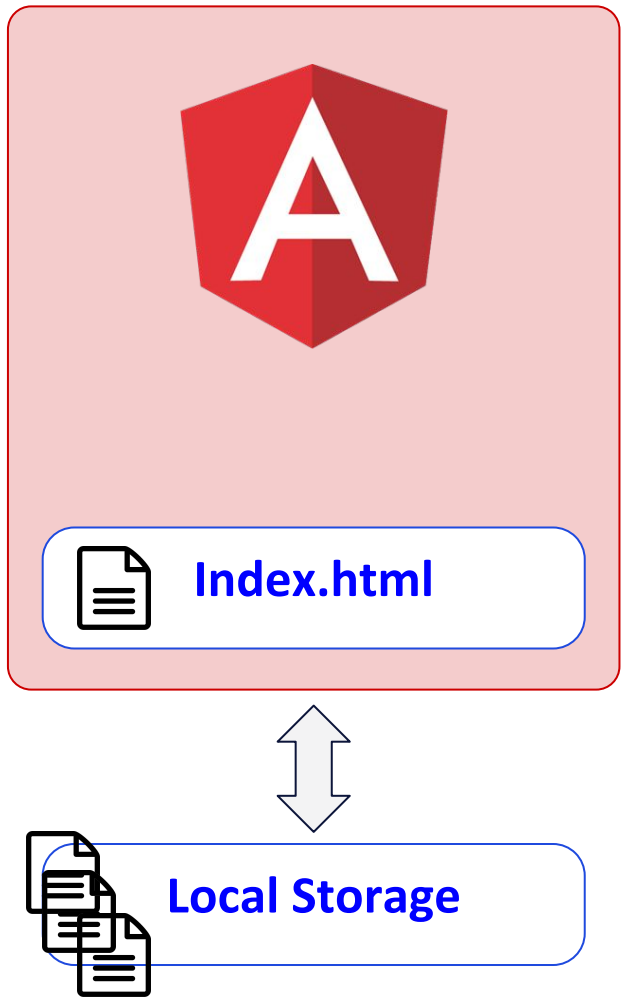
http://todomvc.com/



http://todomvc.com/examples/angularjs/



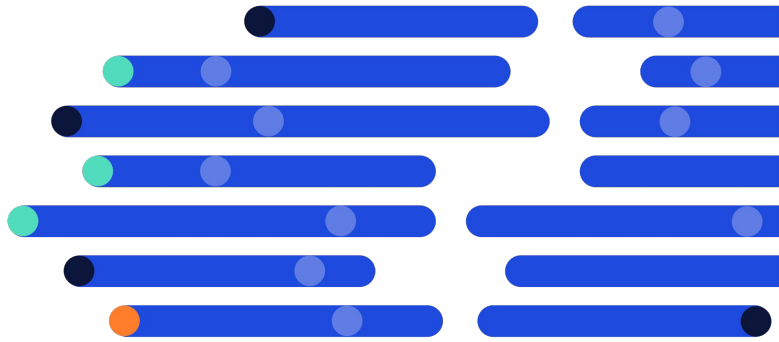
The Todo Application



Exercise 1a

DATASTAX

- **Run the client on your machine**



The Todo Application

https://www.todobackend.com/

Todo-Backend

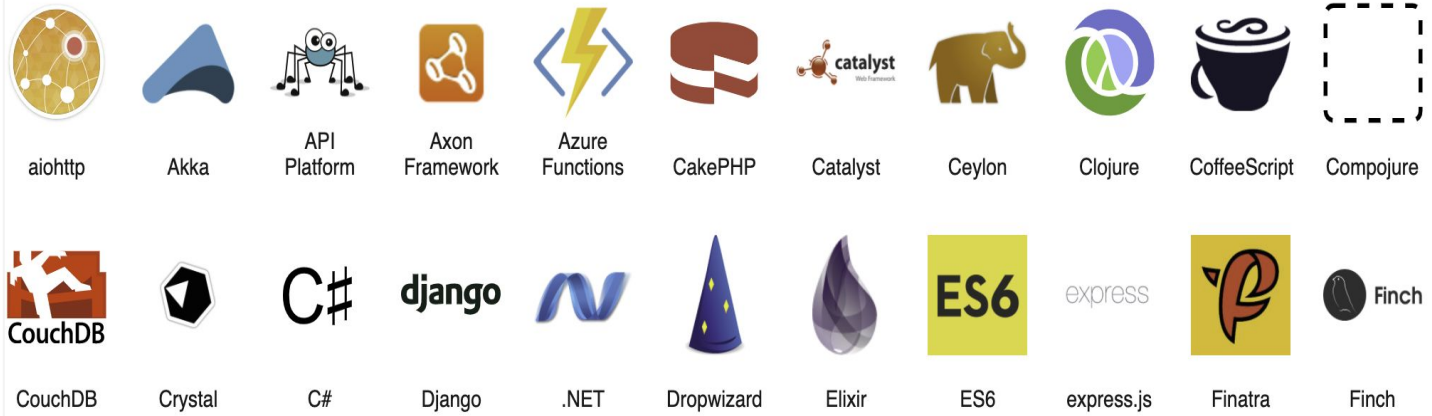
a shared example to showcase backend tech stacks

The Todo-Backend project defines a simple web API spec - for managing a todo list. Contributors implement that spec using various tech stacks. Those implementations are cataloged below. A spec runner verifies that each contribution implements the exact same API, by running an automated test suite which defines the API.

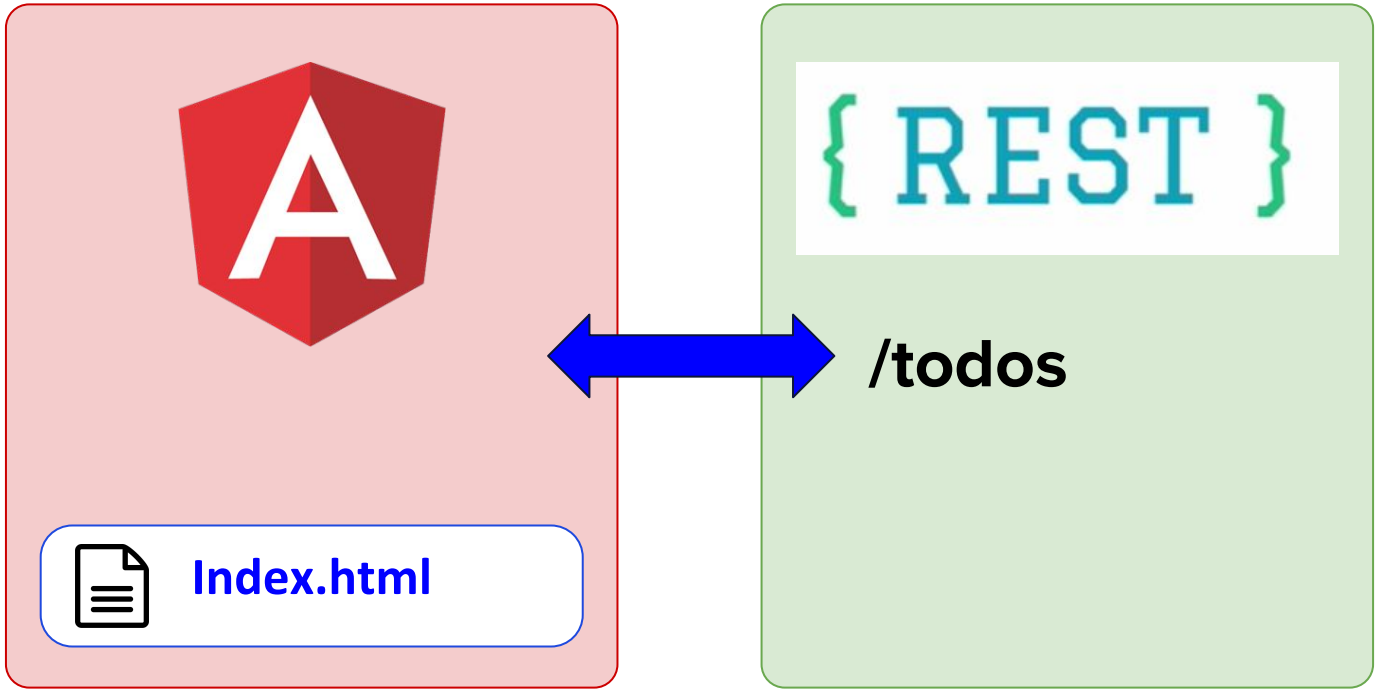
The Todo-Backend project was inspired by the TodoMVC project, and some code (specifically the todo client app) was borrowed directly from TodoMVC.

Created and curated by Pete Hodgson.

featuring HTTP APIs built with:



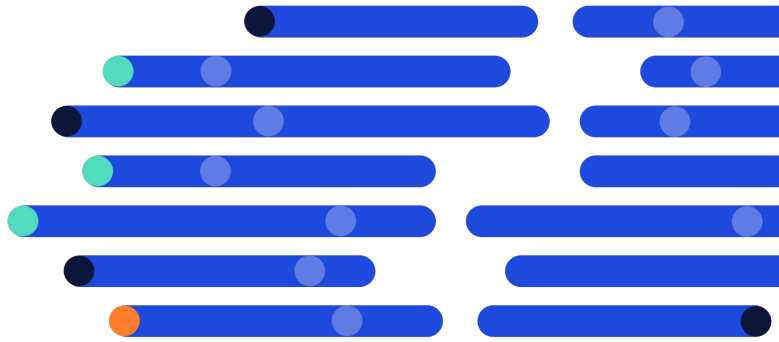
The Todo Application



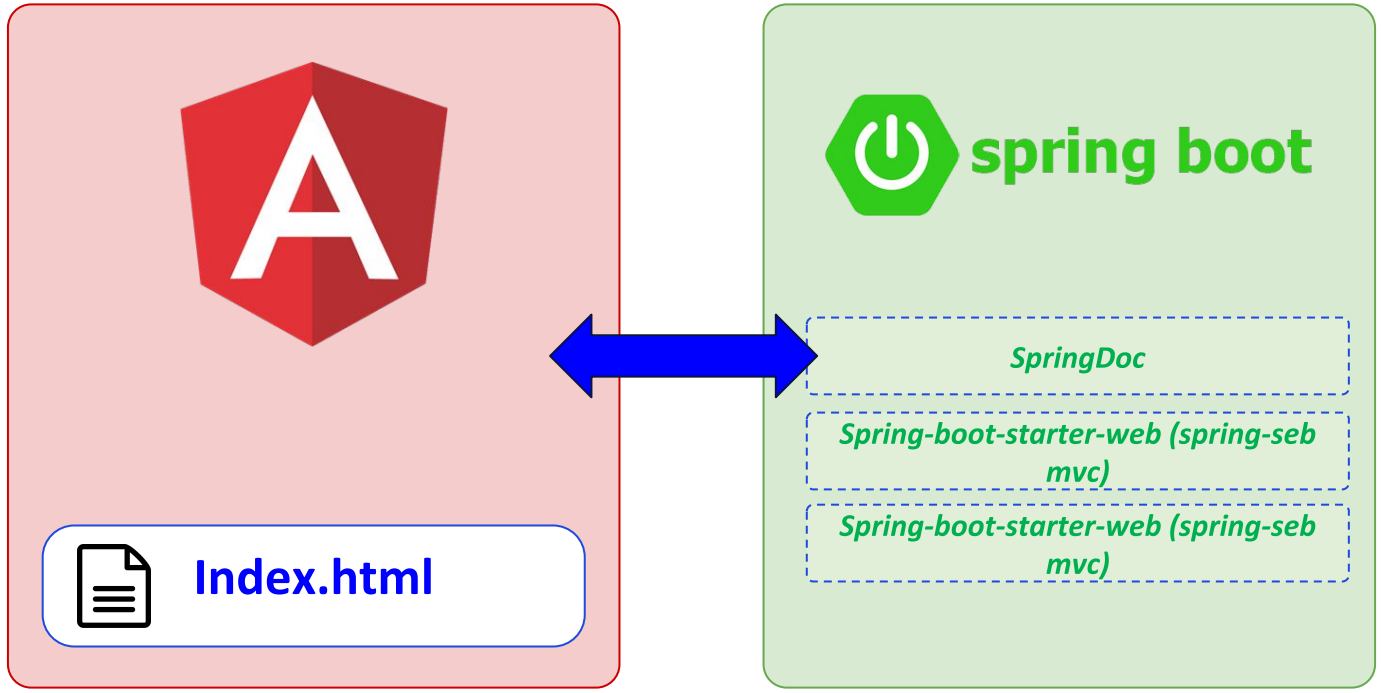
Exercise 1b

DATASTAX

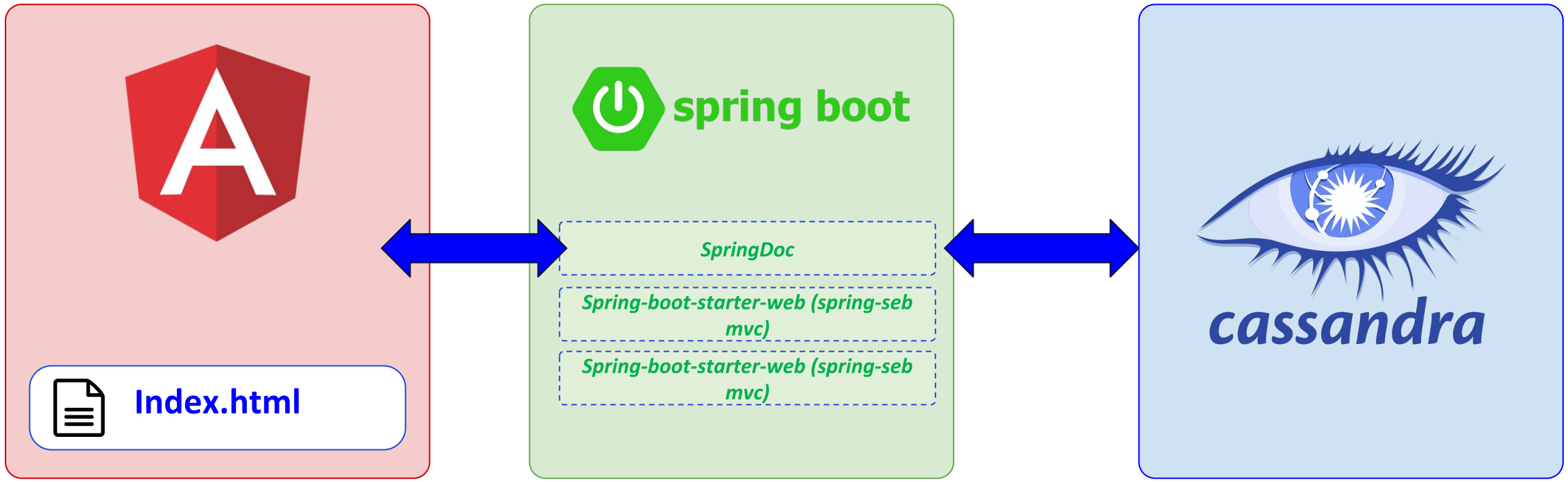
- **Start the backend**
- **Browse and test from generated documentation**
- **Test the API from the specification**



The Todo Application



The Todo Application



cassandra ☆

Docker Official Images

Apache Cassandra is an open-source distributed storage system.

100M+

Container | Linux | PowerPC 64 LE | ARM | ARM 64 | 386 | x86-64 | Databases | Official Image

Linux - x86 (latest)

Copy and paste to pull this image

```
docker pull cassandra
```

View Available Tags

Running Cassandra in Docker

- Define a proper network
- **Env variables** can be defined to override keys in `cassandra.yaml`.
- Export ports **7000, 9042, …**
- Define volumes to stores data
  - **/var/lib/cassandra**

```
$ docker run
  --name some-cassandra -d \
  -e CASSANDRA_BROADCAST_ADDRESS=10.42.42.42 \
  -p 7000:7000,9042:9042
  -v /my/own/datadir:/var/lib/cassandra \
  cassandra:tag
```

```
docker-compose up -d
```

Run Cassandra Locally

Define and run multi-container Docker applications through the use of a YAML file to configure your applications

```
version: '2'
services:

  cassandra-seed:
    container_name: cassandra-seed-node
    image: cassandra: 3.11.6
    ports:
      - "9042:9042"   # Native transport
      - "7199:7199"   # JMX
      - "9160:9160"   # Thrift clients

  cassandra-node:
    image: cassandra: 3.11.6
    command: /bin/bash -c "echo 'Waiting for seed node' && sleep 30 && /docker-entrypoint.sh cassandra -f"
    environment:
      - "CASSANDRA_SEEDS=cassandra-seed-node"
    depends_on:
      - "cassandra-seed"
```

Introducing Astra

Eliminate Operations

everything from provisioning to backups is fully automated

Secure Your Data

with the most advanced security available for Cassandra

Simplify App Development

with auto-configured developer tools that deploy with a click

DATASTAX

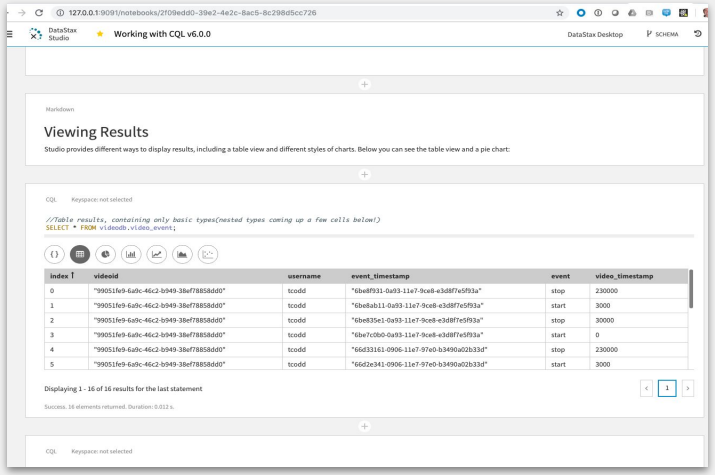
DATASTAX ASTRA

Simplify Application Development

Familiar Language

```
INSERT INTO mytable
(id,name,address) VALUES
(1,'Bob Smith','1 Main
Street')
SELECT * FROM mytable
WHERE id=1
UPDATE mytable SET
name='Tom Smith' WHERE
id=1
DELETE FROM mytable WHERE
id=1
```

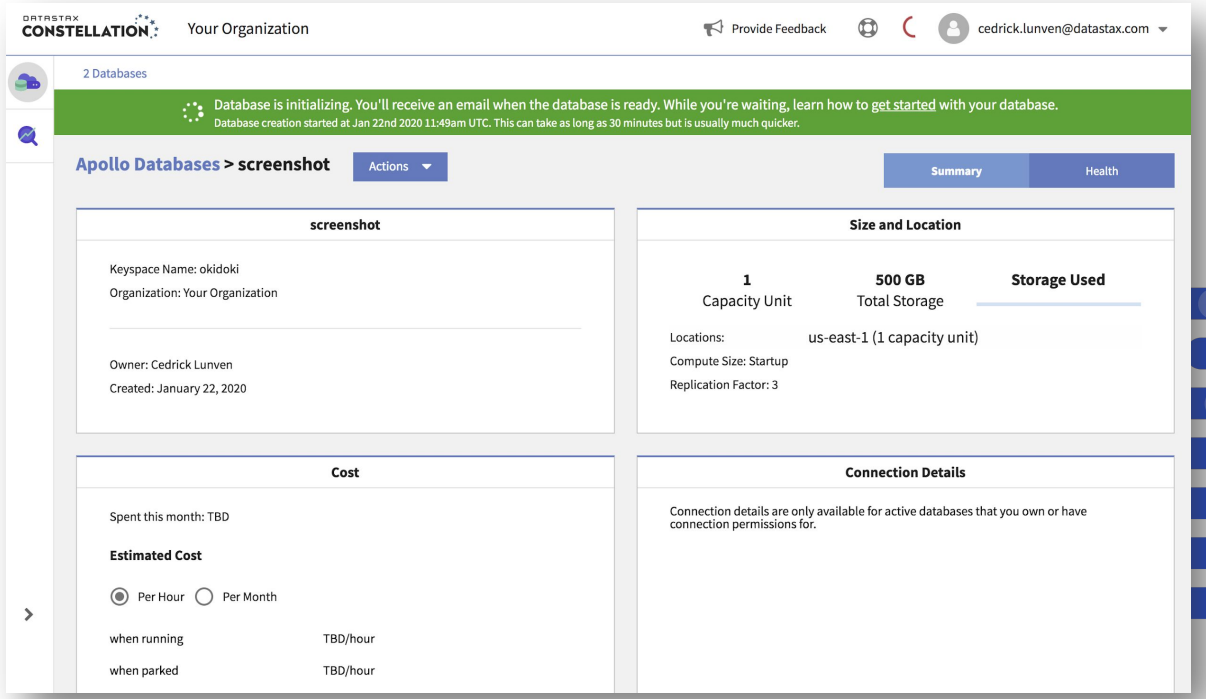
Easy Dev Tools



Great Drivers

Exercise 2

- **Create your Astra Instance**

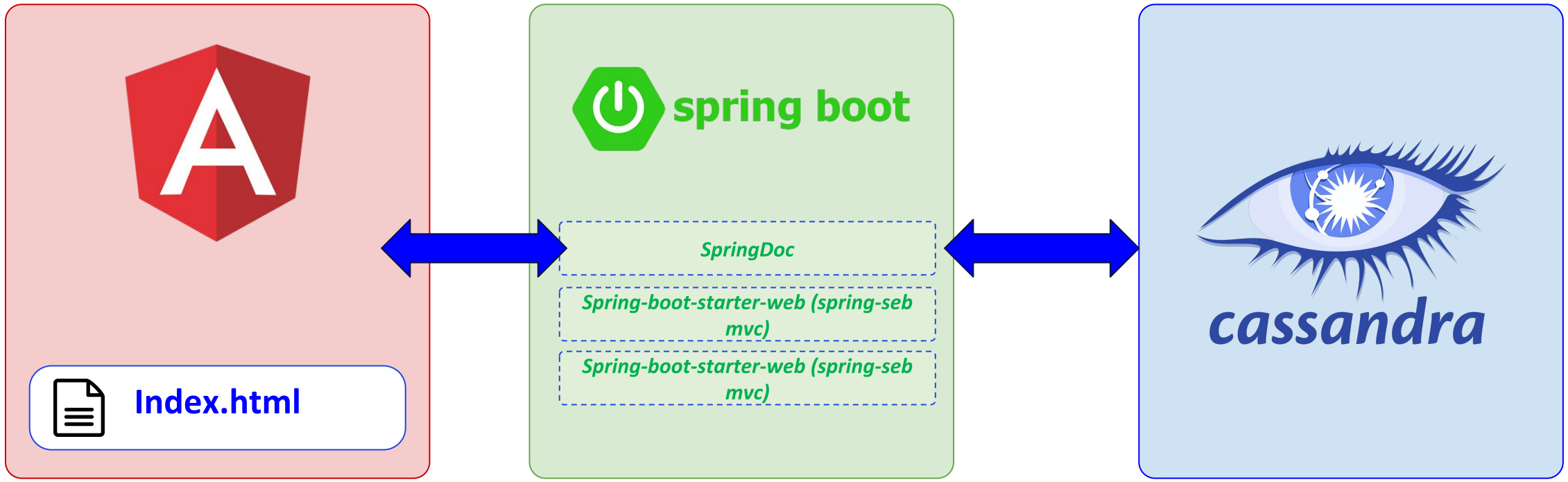


Spring Microservices with Apache Cassandra™

1. Housekeeping
2. Running the TodoApp
3. Connectivity to Cassandra
4. Create the Repository (dao)
5. Spring-Data & Spring-Data-Rest
6. Resources

DATASTAX

DataStax Drivers



DataStax Drivers Features

One of set drivers to connect them all - **january 2020**

Connectivity

- ★ Token & Datacenter Aware
- ★ Load Balancing Policies
- ★ Retry Policies
- ★ Reconnection Policies
- ★ Connection Pooling
- ★ Health Checks
- ★ Authentication | Authorization
- ★ SSL

Query

- ★ CQL Support
- ★ Schema Management
- ★ Sync/Async/Reactive API
- ★ Query Builder
- ★ Compression
- ★ Paging

Parsing Results

- ★ Lazy Load
- ★ Object Mapper
- ★ Spring Support
- ★ Paging

DATASTAX

File-based Configuration

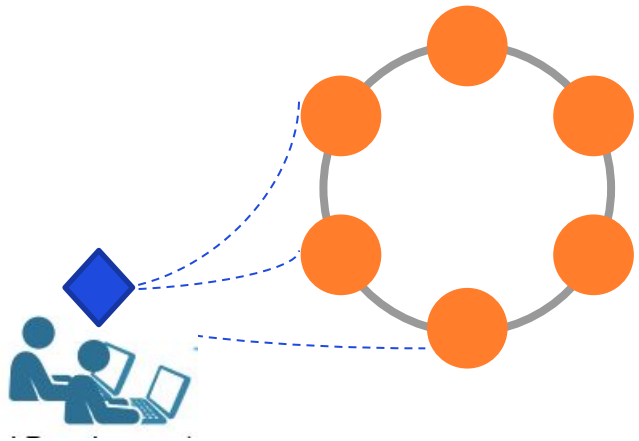
- Based on Typesafe Config
- Attributes are grouped into basic and advanced categories
- A reference file (*reference.conf*) provide default values embedded in the jar file. Can be override with key in application.conf.
- Driver searches `application.conf` in the classpath

application.conf

```
datastax-java-driver {
  basic {
    request.timeout     = 5 seconds
    request.consistency = LOCAL_QUORUM
  }
}
```

Load-Balancing

- Used to create query plans for each statement executed
- Default policy is token-aware, round robin
- Requests are routed to nodes in the “local” data center only



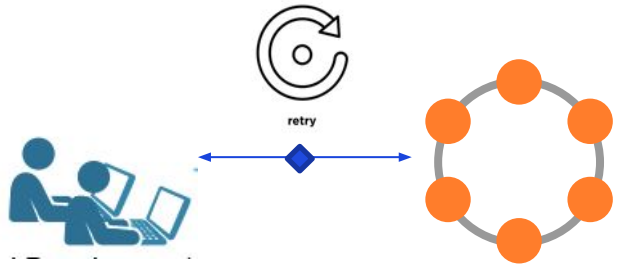
application.conf

```
datastax-java-driver {
 basic {
  load-balancing-policy {

    # The class of the policy.
    class = DefaultLoadBalancingPolicy

    # The datacenter that is considered "local"
    # The default policy will only include nodes from
    # this datacenter in its query plans.
    local-datacenter = datacenter1

    # A custom filter to include/exclude nodes
    // filter.class=
   }
 }
}
```

Retry Policy

Determines when queries are retried on failure

- `DefaultRetryPolicy`
  - Default
  - Retries once onReadTimeout or onWriteTimeout
  - Enough replicas for your consistency level must be online
  - Only retries idempotent mutations



application.conf

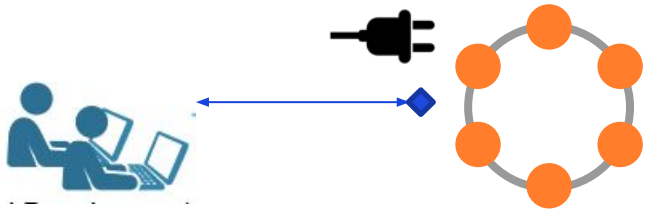
```
datastax-java-driver {

  # The policy that controls if the driver retries
  # requests that have failed on one node.
  advanced.retry-policy {

   # The class of the policy
   class = DefaultRetryPolicy

  }
}
```

Reconnection Policy

Reconnects driver to a downed node

Two options:

- `ConstantReconnectionPolicy`
  - Check every N milliseconds
- `ExponentialReconnectionPolicy`
  - Increases every interval
  - Caps out at a max

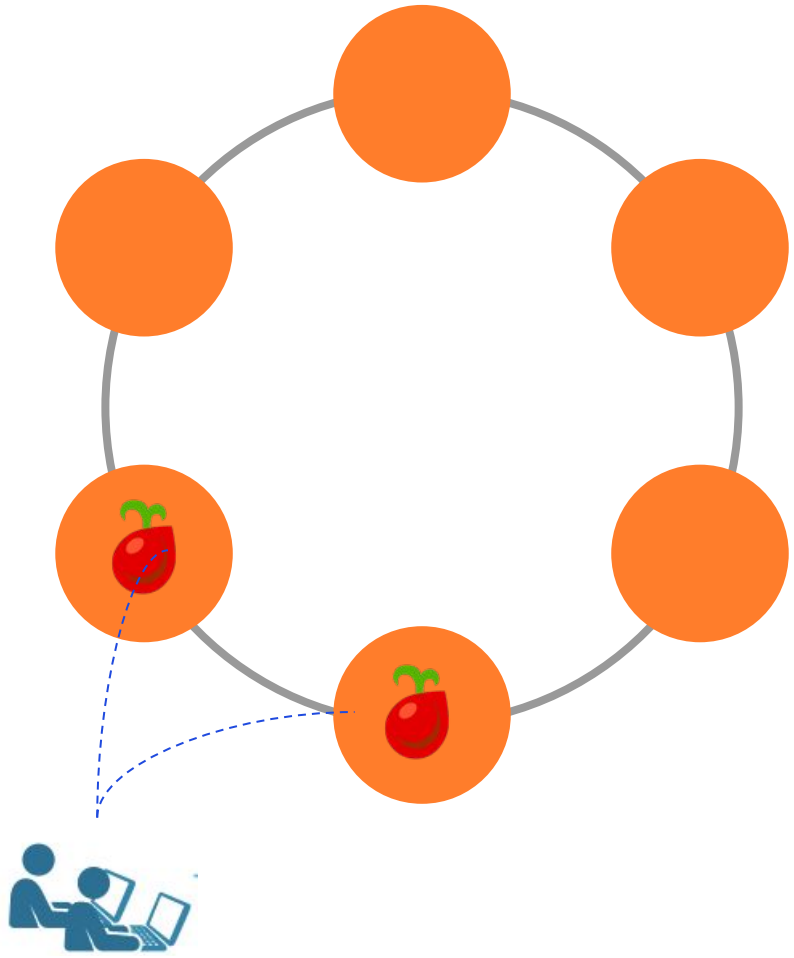


application.conf

```
datastax-java-driver {

  # Whether to schedule reconnection attempts
  # if all contact points are unreachable at init
  advanced.reconnect-on-init = false

  advanced.reconnection-policy {

    # The class of the policy
    class = ExponentialReconnectionPolicy

    # Parameters
    base-delay = 1 second
    max-delay = 60 seconds
  }
}
```

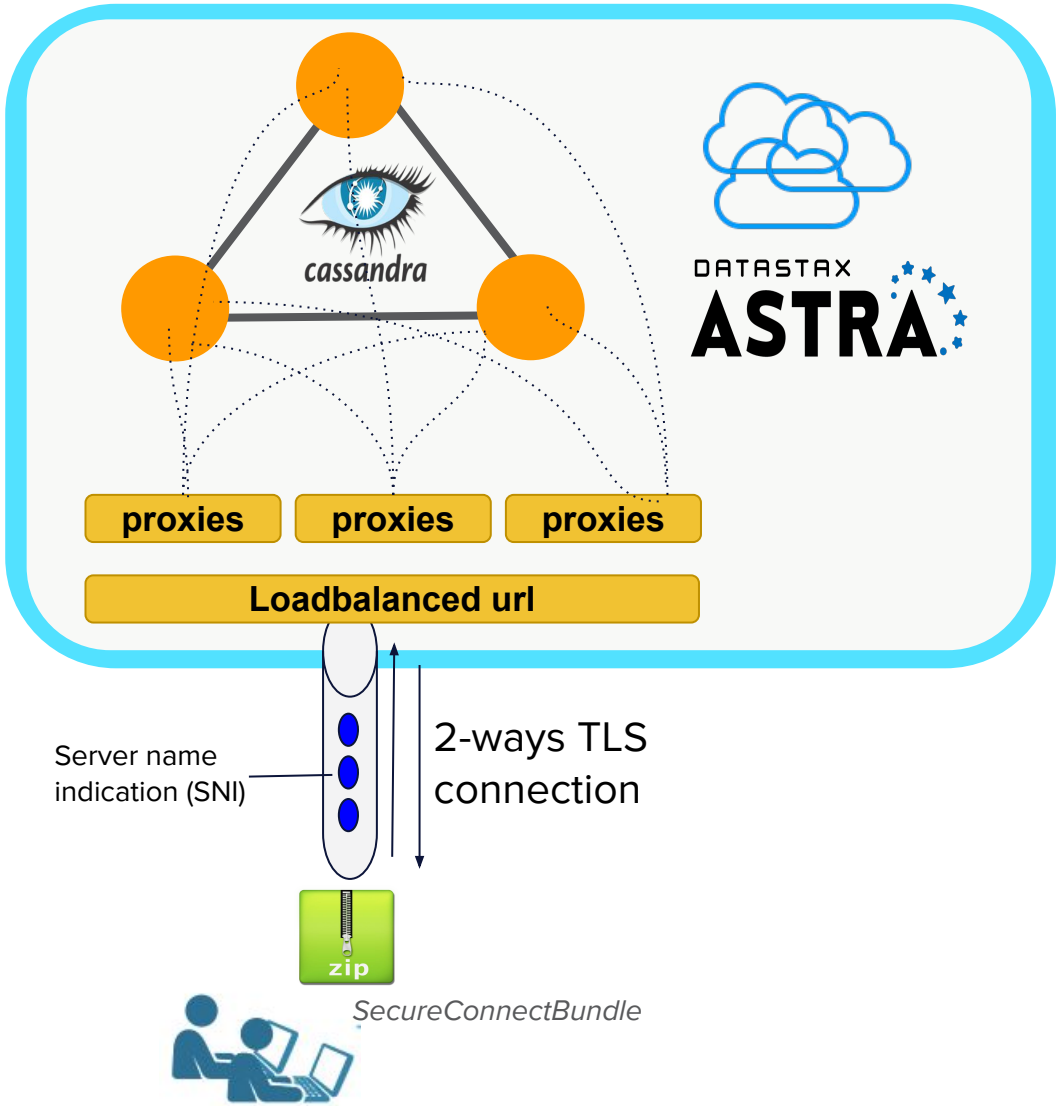
Contact Points

- Only one necessary
- Unless that node is down
- More are good



Contact Points with ASTRA

- Download the SecureConnectBundle from Astra UI.
- Provide username, password and keyspace name in the BUILDER.



Dependencies

Open a Session


```
// Delegate all configuration to file or default
CqlSession cqlSession = CqlSession.builder().build();

// Explicit Settings
CqlSession cqlSession = CqlSession.builder()
 //.addContactPoint(new InetSocketAddress("127.0.0.1", 9042))
 .withCloudSecureConnectBundle(Paths.get("/tmp/apollo.zip"))
 .withContactPoint
 .withKeyspace("killrvideo")
 .withAuthCredentials("KVUser", "KVPassword")
 .build();
```

DATASTAX

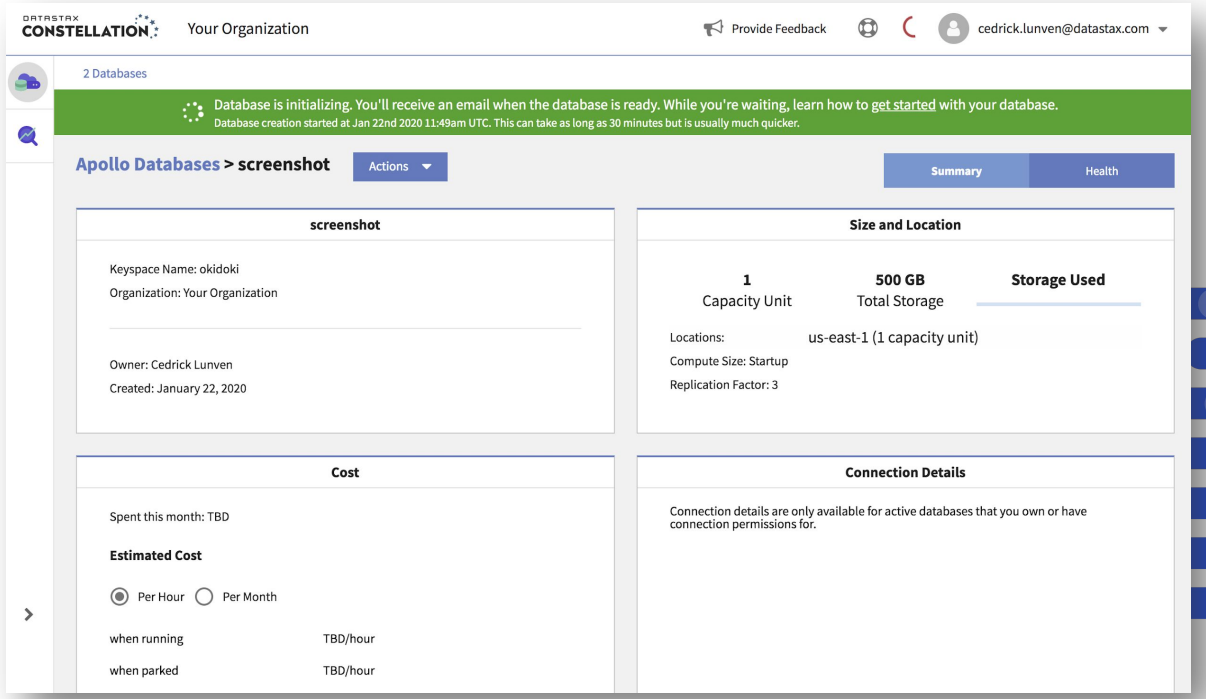
Important to know about CqlSession

- **CqlSession** is a stateful object handling communications with each node
- **CqlSession** should be unique in the Application (*Singleton*)
- **CqlSession** should be closed at application shutdown (*shutdown hook*) in order to free opened TCP sockets (*stateful*)

```
@PreDestroy
public void cleanup() {
 if (null != cqlSession) {
   cqlSession.close();
 }
}
```

Exercise 3

- **Download secure bundle**
- **Make the Tests Green**
- **Connect, list keyspaces**



Spring Microservices with Apache Cassandra™

1. Housekeeping
2. Running the TodoApp
3. Connectivity to Cassandra
4. Create the Repository (dao)
5. Spring-Data & Spring-Data-Rest
6. Resources

DATASTAX

Build Proper Data Model

We know the requests from the specs

How to to execute queries ?

- First job of **CqlSession** is to execute queries using, well, execute method.

```
cqlSession.execute("SELECT * FROM killrvideo.users");
```

SimpleStatement

```
Statement statement = ...

// (1) Explicit SimpleStatement Definition
SimpleStatement.newInstance("select * from t1 where c1 = 5");

// (2) Externalize Parameters (no name)
SimpleStatement.builder("select * from t1 where c1 = ?")
               .addPositionalValue(5);

// (3) Externalize Parameters (name)
SimpleStatement.builder("select * from t1 where c1 = :myVal")
               .addNamedValue("myVal”, 5);

cqlSession.execute(statement);
```

Prepared and Bound Statements

- Compiled once on each node automatically as needed
- Prepare each statement only once per application
- Use one of the many bind variations to create a BoundStatement

```
PreparedStatement ps = cqlSession.prepare("SELECT * from t1 where c1 = ?");

BoundStatement bound = ps.bind(5);

cqlSession.execute(bound);
```

ResultSet

- **ResultSet** is the object returned for executing query. It contains **ROWS** (data) and **EXECUTION INFO**.
- **ResultSet** is **iterable** and as such you can navigate from row to row.
- Results are **always paged** for you (avoiding memory and response time issues)

```
ResultSet rs = cqlSession.execute(myStatement);

// Plumbery
ExecutionInfo info = rs.getExecutionInfo();
int executionTime = info.getQueryTrace().getDurationMicros();

// Data: NOT ALL DATA RETRIEVED IMMEDIATELY (only when needed .next())
Iterator<Row> iterRow = rs.iterator();
int itemsFirstCall = rs.getAvailableWithoutFetching();
```

DATASTAX

Parsing ResultSet

```
// We know there is a single row (eg: count)
Row singleRow = resultSet.one();

// We know there are not so many results we can get all (fetch all pages)
List<Row> allRows = resultSet.all();

// Browse iterable
for(Row myRow : resultSet.iterator()) {
   // .. Parsing rows
}

// Use Lambda
rs.forEach(row -> { row.getColumnDefinitions(); });

// Use for LWT
boolean isQueryExecuted = rs.wasApplied();
```

Parsing Rows

```
// Sample row
Row row = resultSet.one();

// Check null before read
Boolean isUsernNameNull = row.isNull("userName");

// Reading Values from row
String userName1 = row.get("username", String.class);
String userName2 = row.getString("username");
String userName3 = row.getString(CqlIdentifier.fromCql("username"));

// Tons of types available
row.getUuid("userid");
row.getBoolean(”register");
row.getCqlDuration(”elapsed");
...
```

Paging

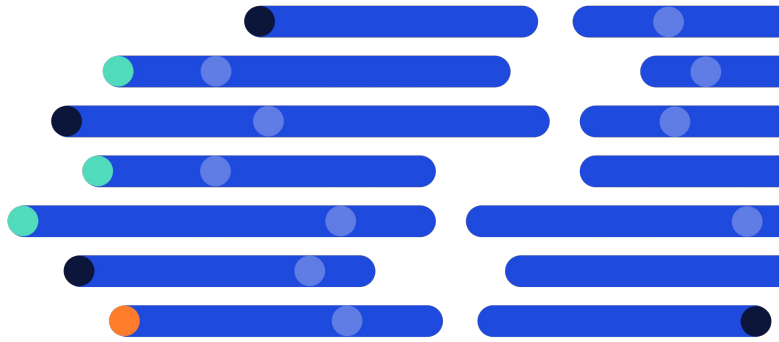
- ResultSet contains up to “**pageSize**” items. When browsing records you may hit this number that will trigger fetching next “pageSize” items.
- To fetch anything else that first page you must provide a **PagingState**.

```
// Enforce few items per page (often = UI requirements)
myStatement = myStatement.setPageSize(10);
ResultSet page1 = cqlSession.execute(myStatement);

// Paging State
ByteBuffer pagingState = page1.getExecutionInfo().getPagingState();
myStatement = myStatement.setPageState(pagingState);

// Very same statement with pagingState provided
ResultSet page2 = cqlSession.execute(myStatement);
```

Exercise 4a

DATASTAX

- **CRUD Repository Ad Hoc**



Query Builder

- Fluent API for building CQL string queries programmatically
- Contains methods to build SELECT, UPDATE, INSERT and DELETE statements
- Generates a Statement as per the earlier techniques

OSS Driver (current version 4.2.0)

```
<dependency>
 <groupId>com.datastax.oss</groupId>
 <artifactId>java-driver-query-builder</artifactId>
</dependency>
```

Query Builder

```
import static com.datastax.oss.driver.api.querybuilder.QueryBuilder.bindMarker;
import static com.datastax.oss.driver.api.querybuilder.QueryBuilder.deleteFrom;
import static com.datastax.oss.driver.api.querybuilder.QueryBuilder.selectFrom;
import static com.datastax.oss.driver.api.querybuilder.relation.Relation.column;

// Simple SELECT using QueryBuilder
Statement stmtSelect = selectFrom(”killrvideo”, ”videos_by_users”)
  .column(”userid”).column(”commentid”)
  .function(”toTimestamp”, Selector.column(”commentid”)).as("comment_timestamp")
  .where(column(”userid”).isEqualTo(bindMarker(”userid”)))
  .build()

// Simple DELETE using QueryBuilder
Statement stmtDelete = deleteFrom(”killrvideo”, ”videos_by_users”)
   .where(column(”userid”).isEqualTo(bindMarker(”userid”)))
   .build()
```

Query Builder

- Can also use **QueryBuilder** to create **PreparedStatements** and later execute at runtime
- Note use of **bindMarker() t**o designate parameters that will be provided later

```
// Prepared QueryBuilder statiements as any statement
PreparedStatement psStmt = cqlSession.prepare(
  deleteFrom(”killrvideo”, ”videos_by_users”)
   .where(column(”userid”).isEqualTo(bindMarker(”userid”)))
   .build()));

// Binding
BoundStatement bsStmt = psStmt.bind(”e7a8ac9f-c12d-415c-a526-4137815df573”);

// Execute
cqlSession.execute(bsStmt);
```

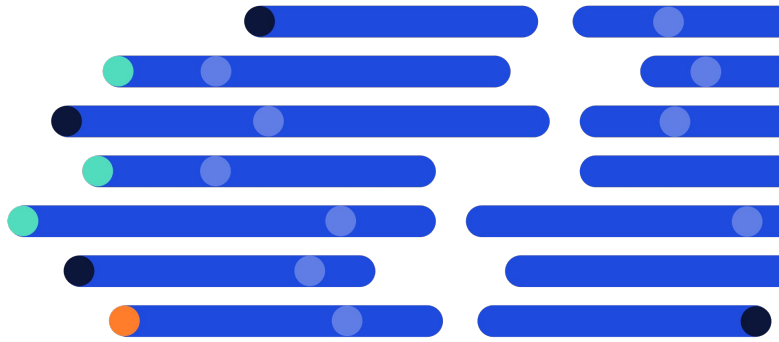
Batch Example

```
// Sample statements (insert same data in multiple tables)
Statement stmt1 = SimpleStatement
 .builder(”INSERT INTO users_by_group(groupid,userid) values(?,?)")
 .addPositionalValue(groupname, username);
Statement stmt2 = SimpleStatement
 .builder(”INSERT INTO groups_by_user(userid,groupid) values(?,?)")
 .addPositionalValue(username, groupname);

// Group as a Batch
BatchStatement batchStmt = BatchStatement
  .builder(DefaultBatchType.LOGGED)
  .addStatement(stmt1).addStatement(stmt2).build();

// Execute
cqlSession.execute(batchStmt);
```

Exercise 4b

DATASTAX

- **CRUD Repository QueryBuilder**



Object Mapper

- WHAT ?
  - Abstracts details of mapping Java attributes to/from CQL types and UDTs
  - Packaged separately from the driver – pom.xml update required
  - This slide shows the runtime dependency – will show compile-time shortly

- HOW ?
  - Some annotation processors will GENERATE Mapper, Dao, and Entity implementations for you
  - At each update in the files, the IDE (eclipse, intelliJ) will use annotation processor
  - Compiler plugin must be updated to define the annotation processor

Object Mapper

OSS Driver

```
<dependency>
 <groupId>com.datastax.oss</groupId>
 <artifactId>java-driver-mapper-runtime</artifactId>
</dependency>

<!-- X -->

<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-compiler-plugin</artifactId>
  <configuration>
    <release>11</release>
 <annotationProcessorPaths>
 <path>
 <groupId>com.datastax.oss</groupId>
 <artifactId>java-driver-mapper-processor</artifactId>
 </path>
 </annotationProcessorPaths>
   </configuration>
</plugin>
```

DSE Driver

```
<dependency>
 <groupId>com.datastax.dse</groupId>
 <artifactId>dse-java-driver-mapper-runtime</artifactId>
</dependency>

<!-- X -->

<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-compiler-plugin</artifactId>
  <configuration>
    <release>11</release>
 <annotationProcessorPaths>
 <path>
 <groupId>com.datastax.dse</groupId>
 <artifactId>dse-java-driver-mapper-processor</artifactId>
 </path>
 </annotationProcessorPaths>
   </configuration>
</plugin>
```

Annotate Entities

```
@Entity
@CqlName("user_v")
public class UserVideo {

 @PartitionKey
 @CqlName("userid")
 private UUID userid;

 @ClusteringColumn(1)
 @CqlName(”added")
 private UUID videoid;

}
```

TABLE NAME, KEYSPACE

PARTITION KEY COLUMNS

CLUSTERING COLUMNS

Annotate DAO Interface (1/2)

```
@Dao
public interface VideoDao {

  @Select
  Optional<UserVideo> findUserById(UUID userid);

  @Query("SELECT * FROM ${tableId}")
  PagingIterable<UserVideo> findAll();

  @Select(customWhereClause = "videoid = : videoid")
  PagingIterable<UserVideo>
   findUserByVideoId(@CqlName("videoid") UUID vid);
```

Annotate DAO Interface (2/2)

```
// Save a bean
@Insert
void save(UserVideo userVideo);

// Userid id is PK
@Delete
void delete(UUID userid);

// Custom implementations
@QueryProvider(
  providerClass = MySampleQueryProvider.class,
  entityHelpers = { UserVideo.class })
String doSomething(String abc);
```

Annotate Mapper Interface

```
@Mapper
public interface MyApplicationMapper {

    @DaoFactory
    VideoDao videoDao(@DaoKeyspace CqlIdentifier keyspace);

}
```

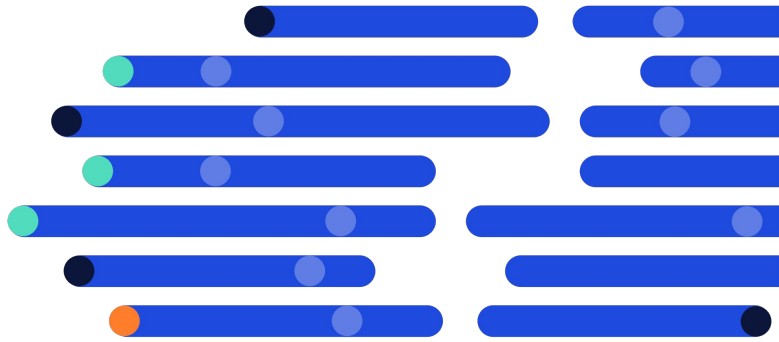
Sample QueryProvider

```
public class MySampleQueryProvider {

  // Constructor, getting session
  public MySampleQueryProvider(
     MapperContext context,
     EntityHelper<UserVideo > helperUser} {}

  // Custom implementation method
  public String doSomething(String abc) {
  }
}
```

Exercise 4c

DATASTAX

- **CRUD Repository Object Mapper**

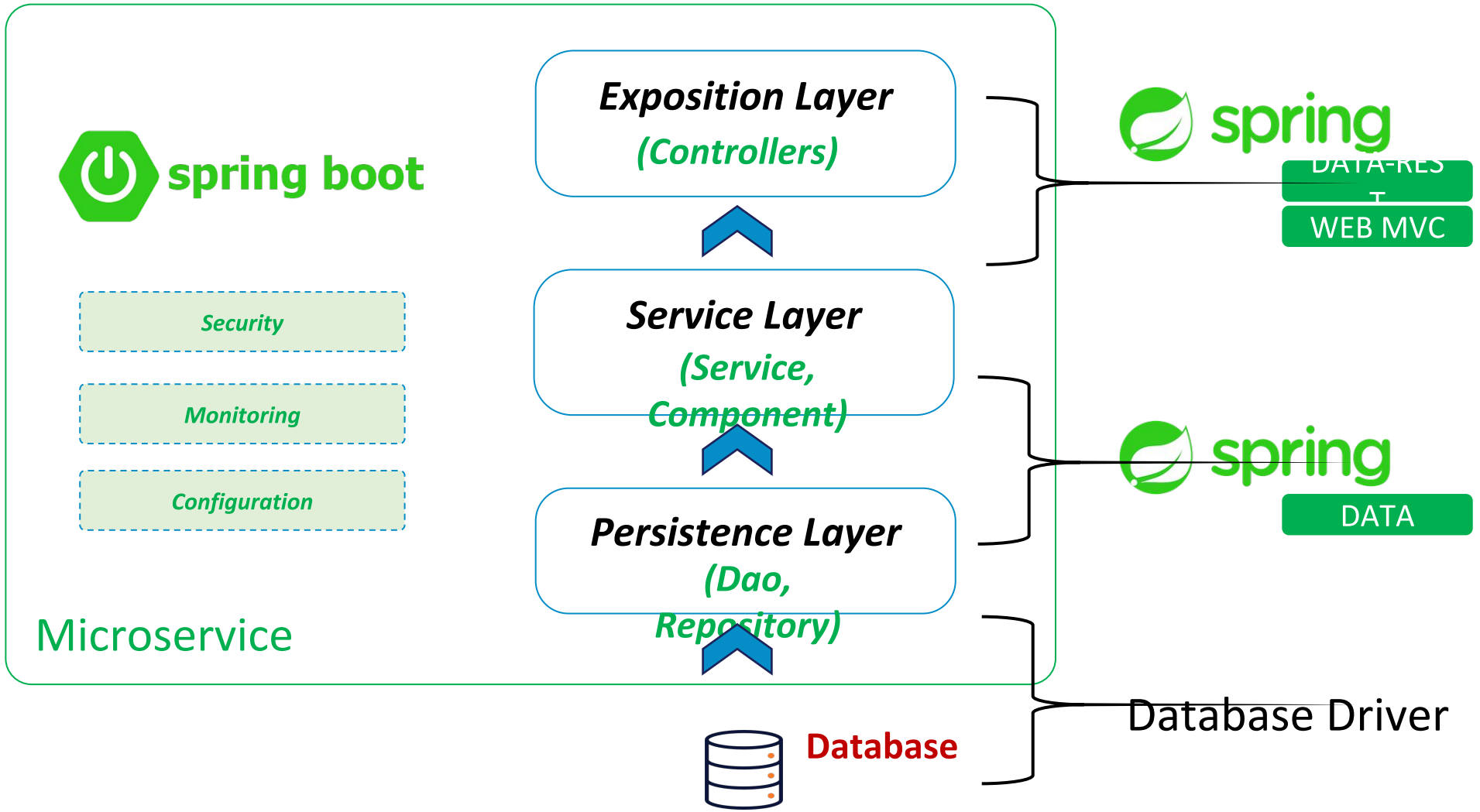


Spring Microservices with Apache Cassandra™

1. Housekeeping
2. Running the TodoApp
3. Connectivity to Cassandra
4. Create the Repository (dao)
5. Spring-Data & Spring-Data-Rest
6. Resources

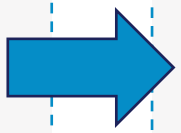
DATASTAX

« Vanilla » So easy….



Entity => Spring-Data => Spring-Data-REST

```java
@Entity
public class Customer {

@Id
@GeneratedValue(strategy=GenerationType.AUTO
) private Long id;

private String firstName;

private String lastName;

private Customer() {}

private Customer(
    String firstName, String lastName) {
    this.firstName = firstName;
    this.lastName = lastName;
}
```



```java
@RepositoryRestResource(
 collectionResourceRel = "customers",
 path = "customers")
```

```java
public interface CustomerRepository extends
CrudRepository<Customer, Long> {

  List<Customer> findByLastName(String lastName);

  Customer findById(long id);
}
```

Configuration – 1# Convention

```
# ---------------------------------------
# Full Convention
# ---------------------------------------
spring:
  data:
    cassandra:
      contact-points: localhost
      port: 9042
      local-datacenter: dc1
      keyspace-name: betterbotz
      schema-action: create-if-not-exists
```

Configuration – #2 Java Config

```
@Configuration
public class SpringDataCassandraJavaConfig
                extends AbstractCassandraConfiguration
                implements CqlSessionBuilderCustomizer {

    @Override
    protected String getKeyspaceName() {
        return keyspaceName;
    }

    @Override
    protected String getLocalDataCenter() {
        return localDataCenter;
    }
```

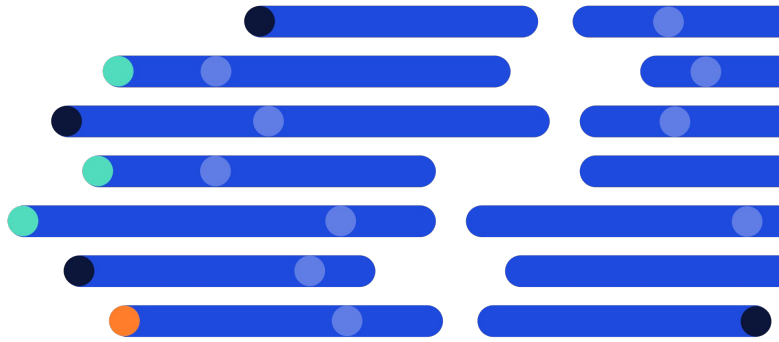
Configuration - #3 Custom Definition

```
@Configuration
public class DseConfiguration {

    /** Internal logger. */
    private static final Logger LOGGER = LoggerFactory.getLogger(DseConfiguration.class);

    @Value("#{'${dse.contactPoints}'.split(',')}")
    public List < String > contactPoints;

    @Bean
    public DseSession dseSession() {
        long top = System.currentTimeMillis();
        LOGGER.info("Initializing connection to DSE Cluster");

        Builder clusterConfig = new Builder();
        LOGGER.info(" + Contact Points : {}" , contactPoints);
        contactPoints.stream().forEach(clusterConfig::addContactPoint);
        LOGGER.info(" + Listening Port : {}", port);
        clusterConfig.withPort(port);
```

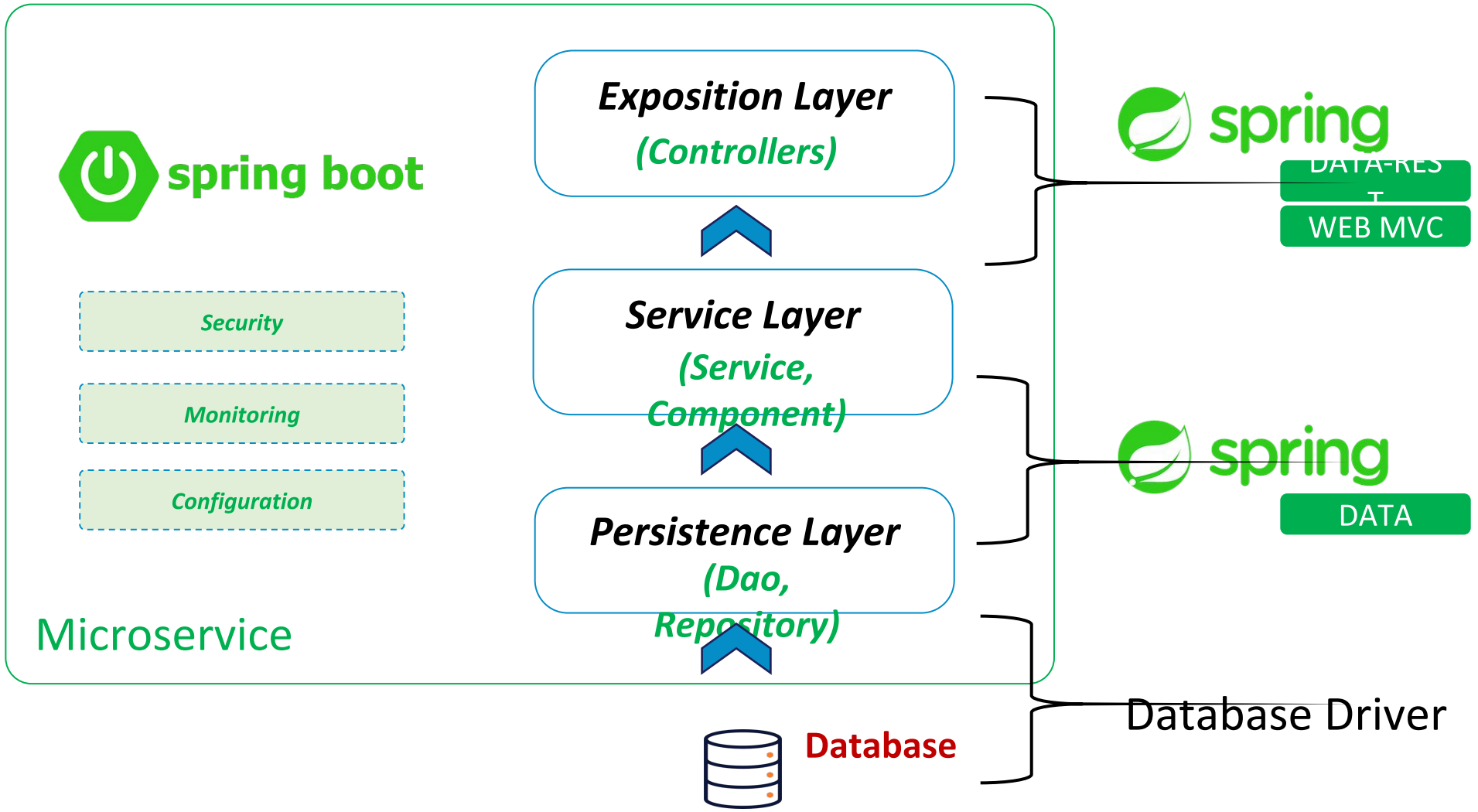
Exercise 5

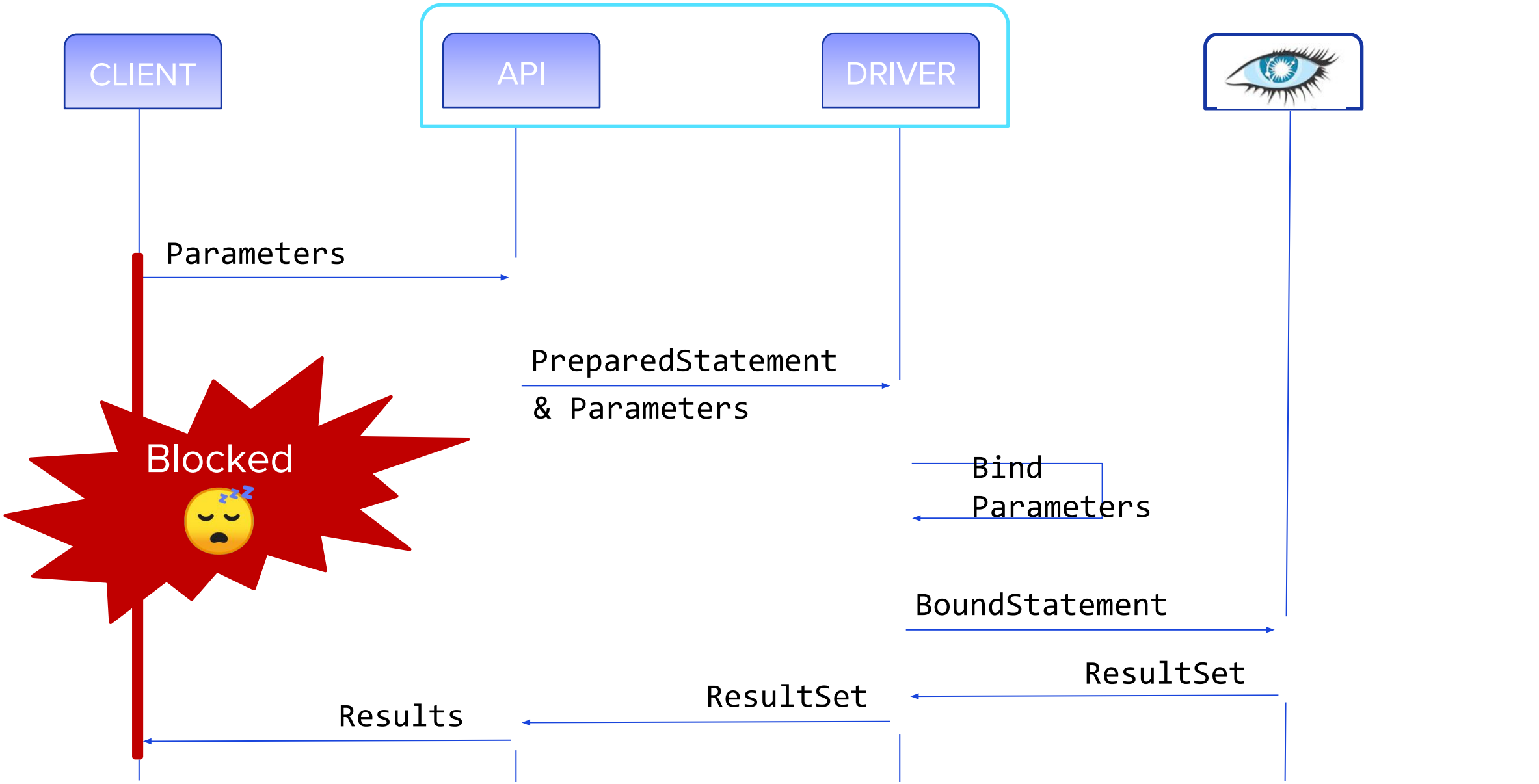
DATASTAX

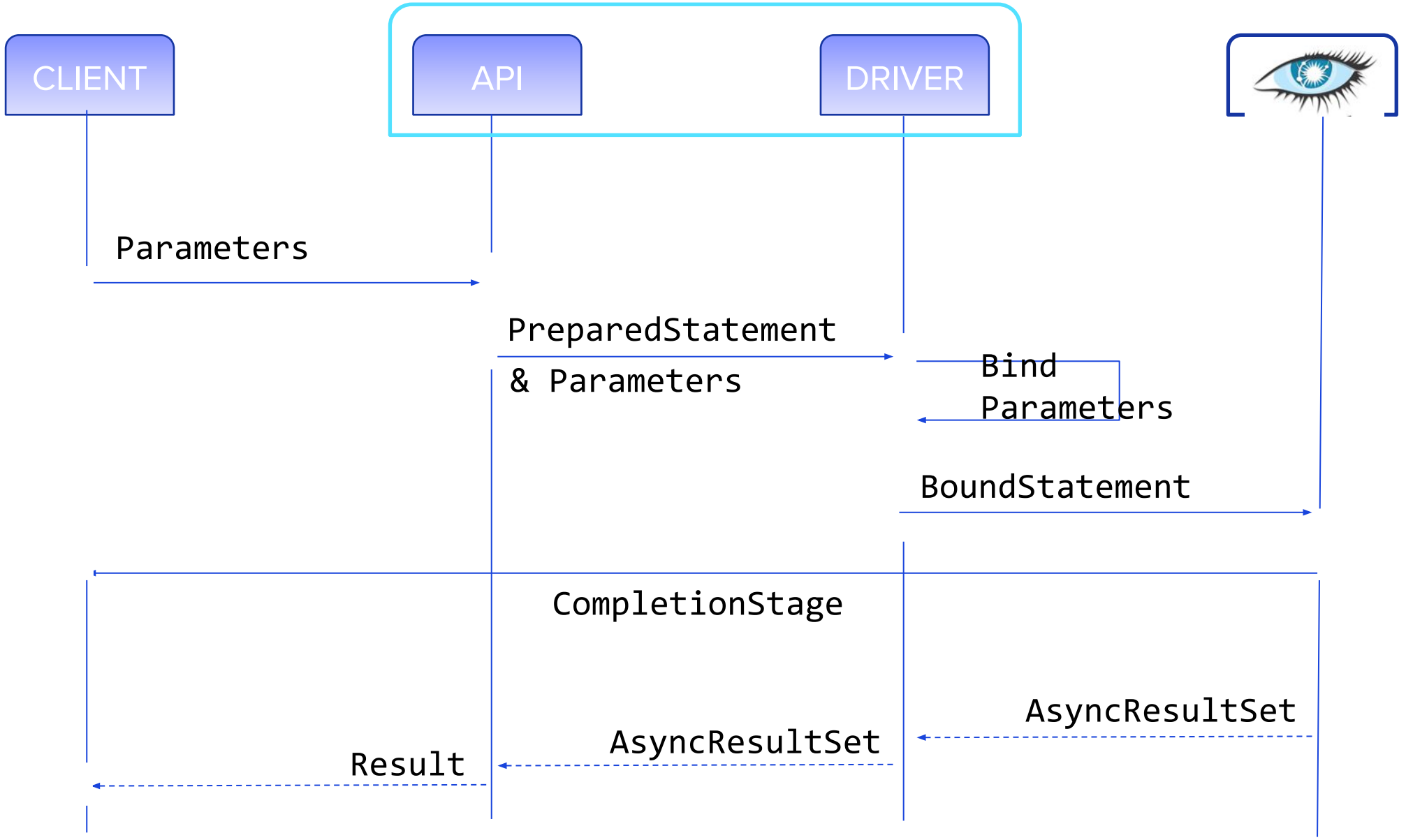
- **Connection with Spring Data**
- **CRUD Repository with Spring Data**
- **Spring Data REST**



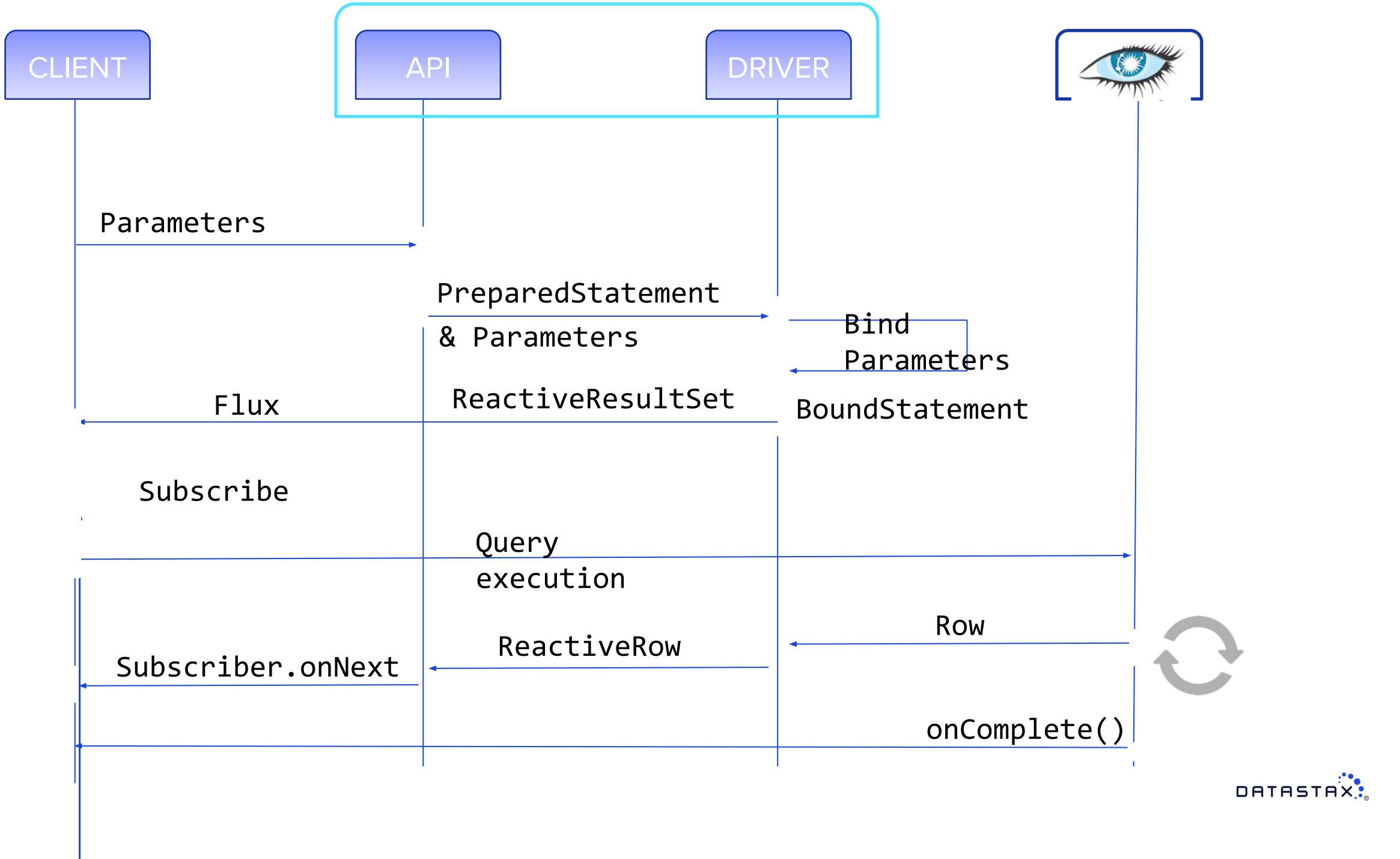
WEbFlux


spring boot
Exposition Layer
(Controllers)
Service Layer
(Service, Component)
Persistence Layer
(Dao, Repository)
Security
Monitoring
Configuration
Microservice
spring
WEB MVC
spring
DATA
Database Driver
Database

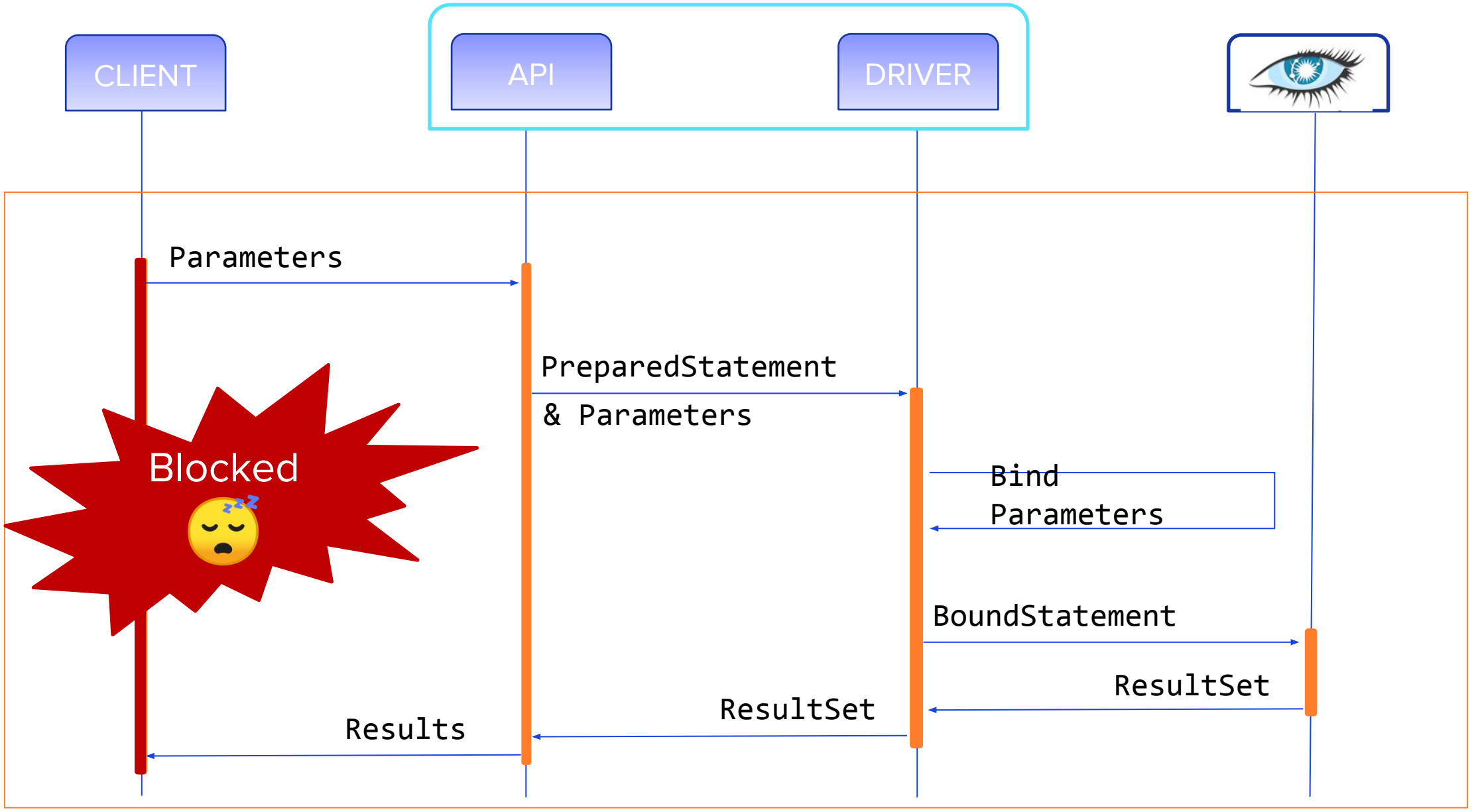

CLIENT
API
DRIVER
Parameters
PreparedStatement
& Parameters
Blocked
Bind
Parameters
BoundStatement
ResultSet
ResultSet
Results


CLIENT
API
DRIVER
Parameters
PreparedStatement
& Parameters
Bind
Parameters
BoundStatement
CompletionStage
AsyncResultSet
AsyncResultSet
Result

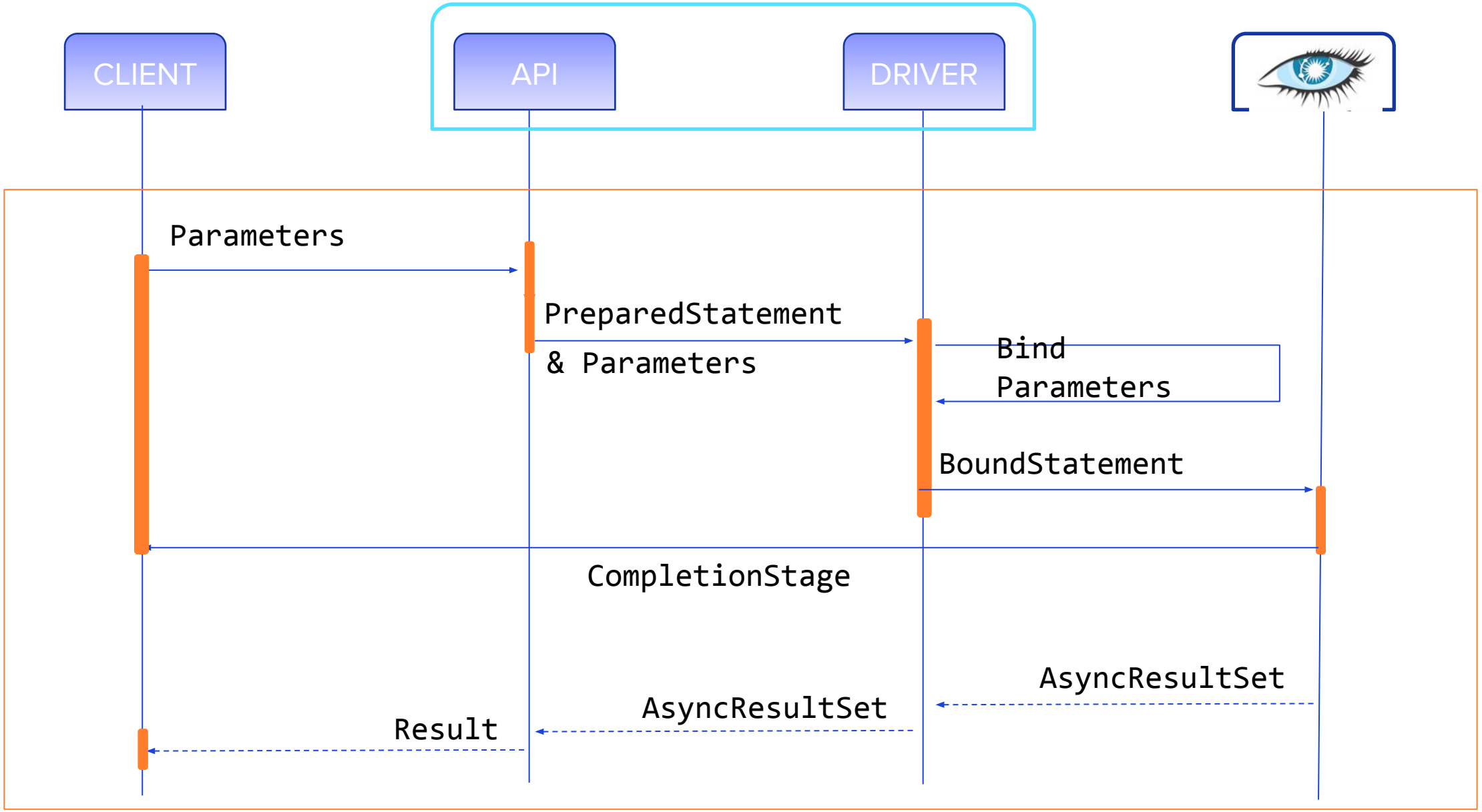
DATASTAX


CLIENT
API
DRIVER
Parameters
PreparedStatement
& Parameters
Bind
Parameters
Flux
ReactiveResultSet
BoundStatement
Subscribe
Query
execution
Row
ReactiveRow
Subscriber.onNext
onComplete()

DATASTAX



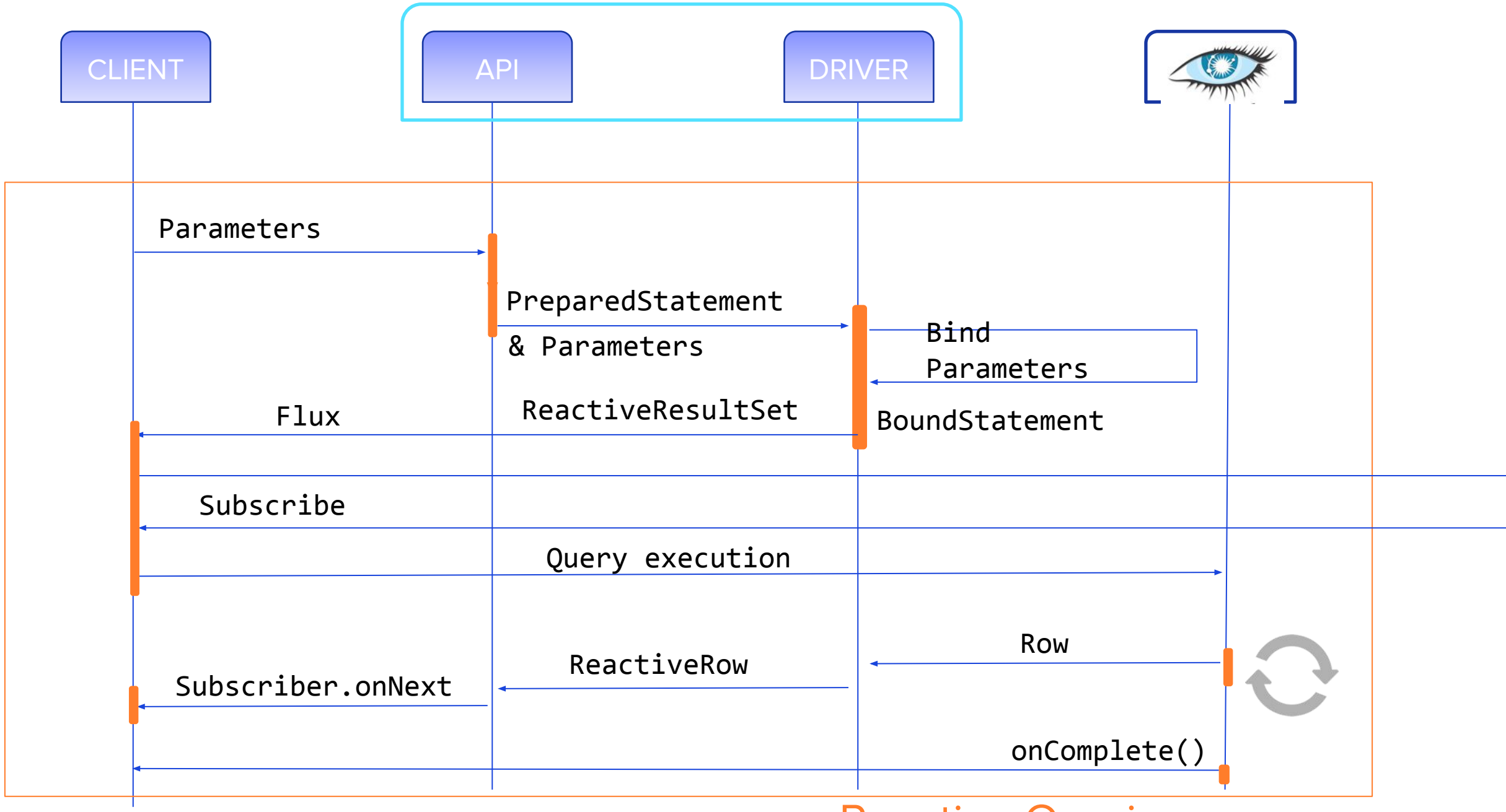
Synchronous Queries



Asynchronous Queries

Asynchronous Queries

```
// From Synchronous
ResultSet resSync = cqlSession.execute(myStatement);

// to Asynchronous
CompletionStage<AsyncResultSet> resAsync =
             cqlSession.executeAsync(myStatement);

resAsync.thenApply(AsyncPagingIterable::one)
        .thenApply(Optional::ofNullable)
        .thenApply(optional -> optional.map(rowMapper))
        .then..
```



Reactive Queries

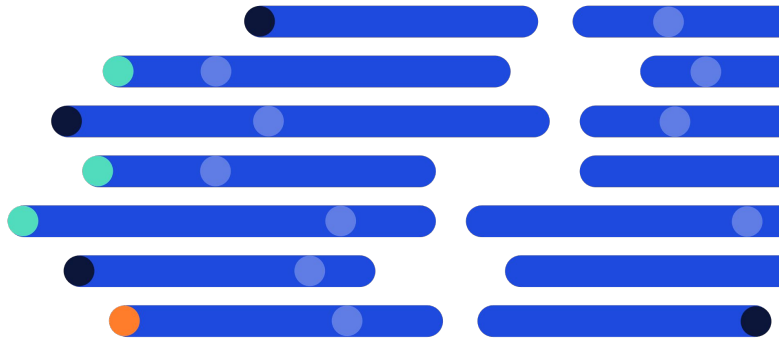
Reactive Queries

```
private final Function<Row, MyBean> rowMapper = ..;

// Execute in reactive way
ReactiveResultSet rs = session.executeReactive(myStatement);

// Return Flux for lists
Flux<ReactiveRow> flux = Flux.from(rs);
Flux<MyBean> res1= flux.skip(offset).take(limit).map(rowMapper);

// Return Mono for single bean
Mono<MyBean> res2 = Mono.fromDirect(rs).map(rowMapper);
```

Exercise 6

DATASTAX

- **Reactive Repository and Spring WebFlux**
- **Spring Data Reactive**



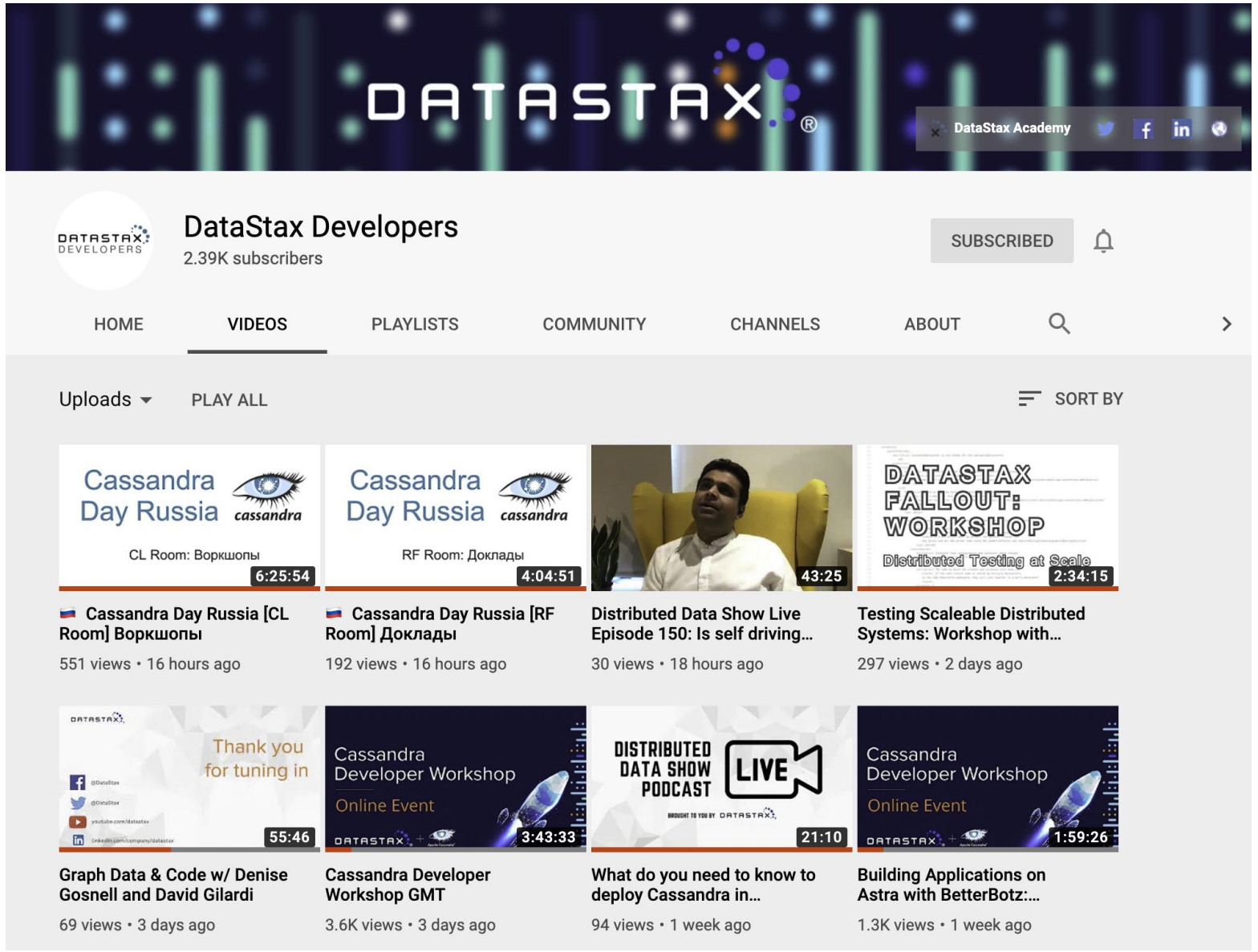
Spring Microservices with Apache Cassandra™

1. Housekeeping
2. Running the TodoApp
3. Connectivity to Cassandra
4. Create the Repository (dao)
5. Spring-Data & Spring-Data-Rest
6. Resources

DATASTAX

DataStax Developers on Youtube

- https://www.youtube.com/channel/UCAIQY251avaMv7bBv5PCo-A



Sample CODES

Micro-service **REST**

https://bit.ly/31RL62I

Micro-service **GraphQL**

https://bit.ly/2MVicup

Micro-service **GRPC**

https://bit.ly/2pofk0b

Micro-service **Kafka**

https://bit.ly/2JwsFdM

Reactive with Webflux

https://bit.ly/34ePzhL

Micro-service **Serverless**

https://bit.ly/31VQz8G


menti.com
00 00 00
Mentimeter
What is the most desirable feature of your ideal workplace?
Flexible time
Social factor
Big paycheck
Available on the iPhone
App Store
GET IT ON
Google play

Developer Resources

LEARN

Join academy.datastax.com
Free online courses - Cassandra certifications

ASK/SHARE

Join community.datastax.com
Ask/answer community user questions - share your expertise

DATASTAX

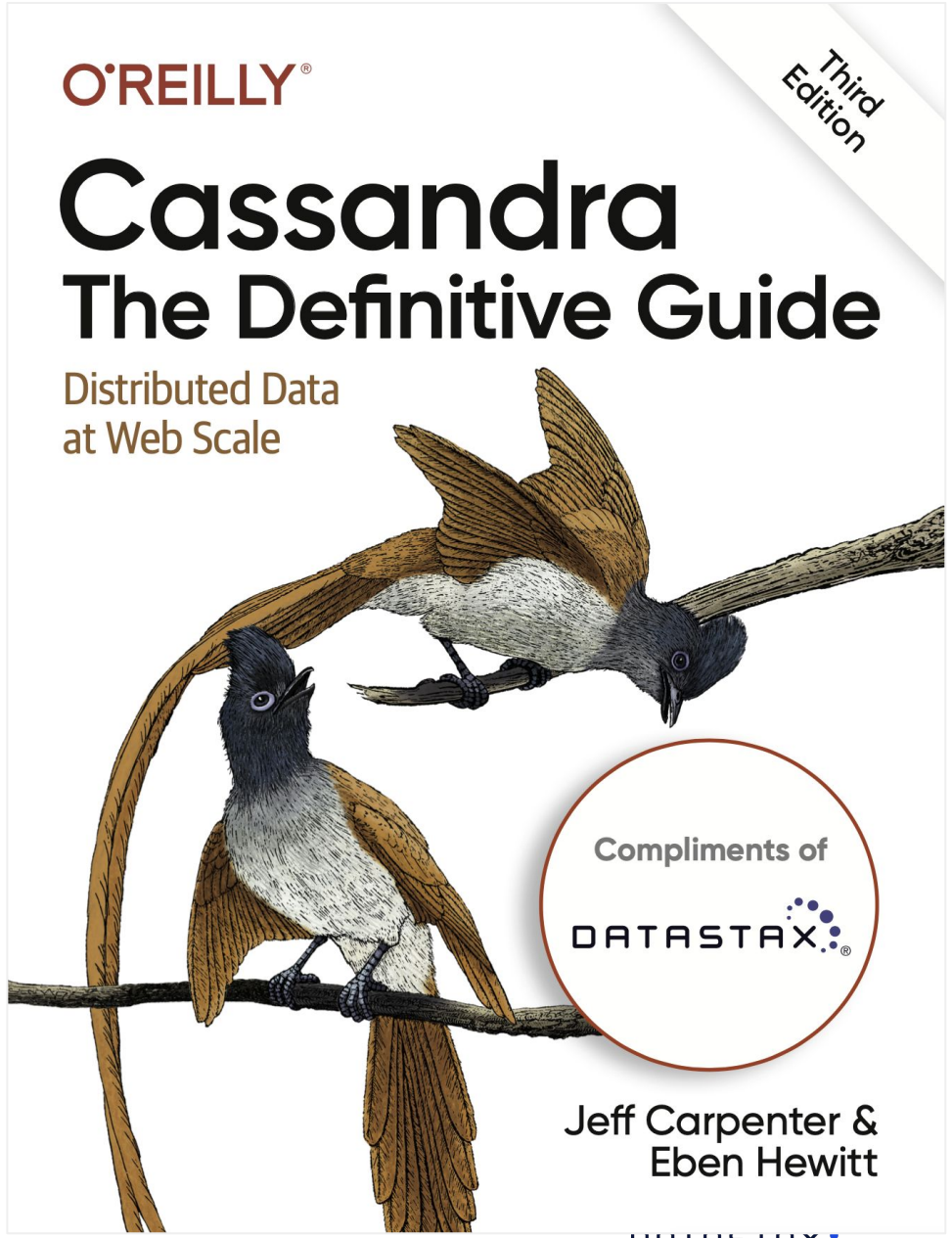
Get Certified !

https://bit.ly/cassandra-cert-FREE

	Administrator	Architect	Developer	Graph Specialist
	LOGIN TO VIEW	LOGIN TO VIEW	LOGIN TO VIEW	LOGIN TO VIEW
DS101: Introduction to Apache Cassandra™	Included	Included	Included	Included
DS201: DataStax Enterprise 6 Foundations of Apache Cassandra™	Included	Included	Included	Included
DS210: DataStax Enterprise 6 Operations with Apache Cassandra™	Included	Included	N/A	N/A
DS220: DataStax Enterprise 6 Practical Application Data Modeling with Apache Cassandra™	N/A	Included	Included	Included
DS330: DataStax Enterprise 6 Graph	N/A	N/A	N/A	Included

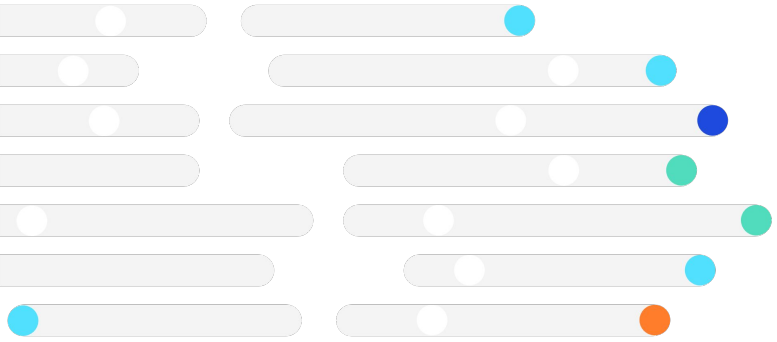
Cassandra The Definitive Guide

https://www.datastax.com/resources/ebook/oreilly-cassandra-definitive-guide



Upcoming events

Date	Time	Content	Type
6/24	12pm IST	Build a REST API for Cassandra with NodeJS and Python	WORKSHOP = HANDS-ON
July 1st July 2nd	12pm EDT 12:30pm IST		
July 8th July 9th			
7/15			
7/22			
7/29			



Thank you