



Build a **Spring** Java Microservice with Apache Cassandra™

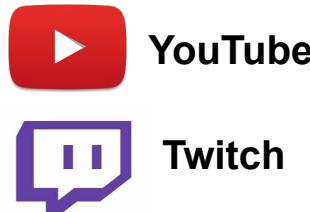
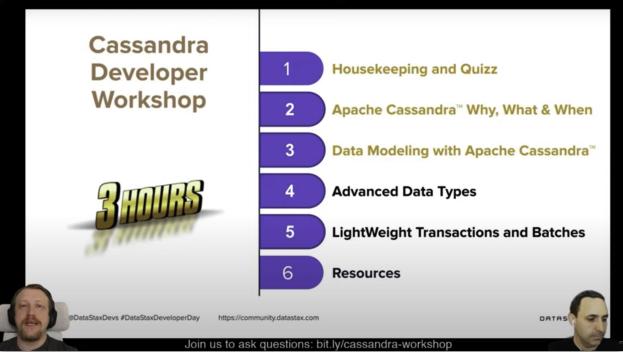


The Crew

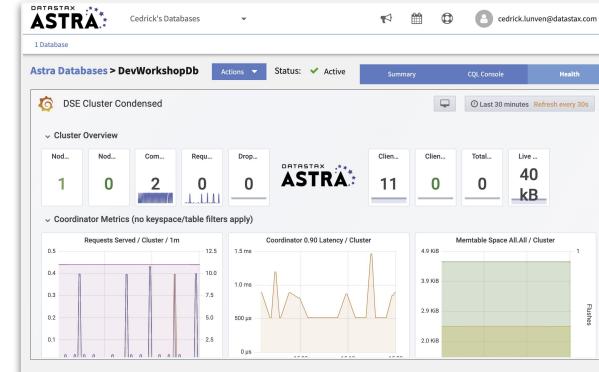


DataStax Developer Advocacy Special Unit

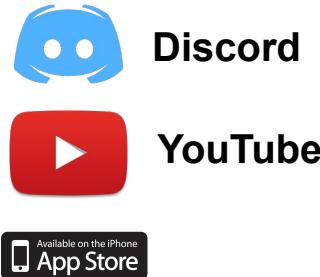
Courses: youtube.com/DataStaxDevs



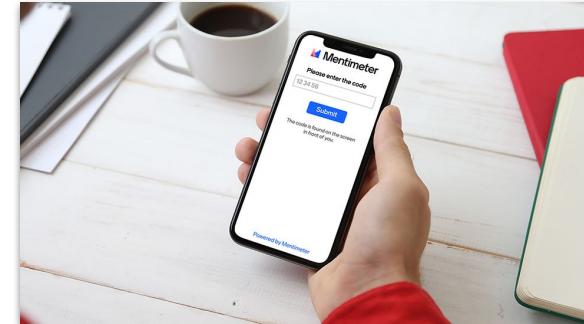
Runtime: dtsx.io/workshop



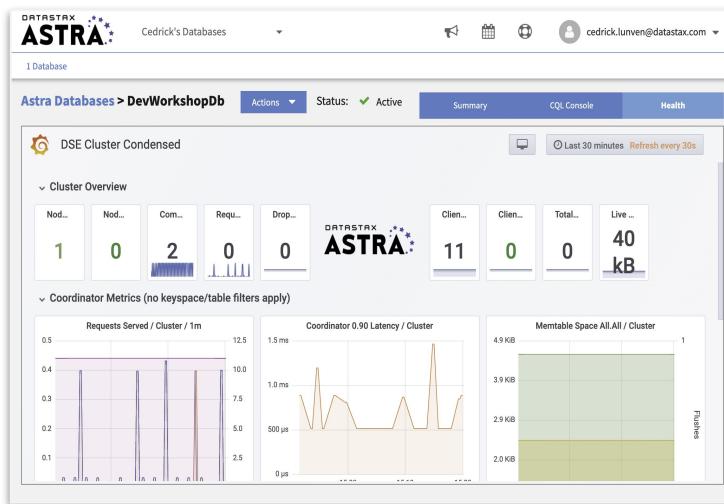
Questions: bit.ly/cassandra-workshop



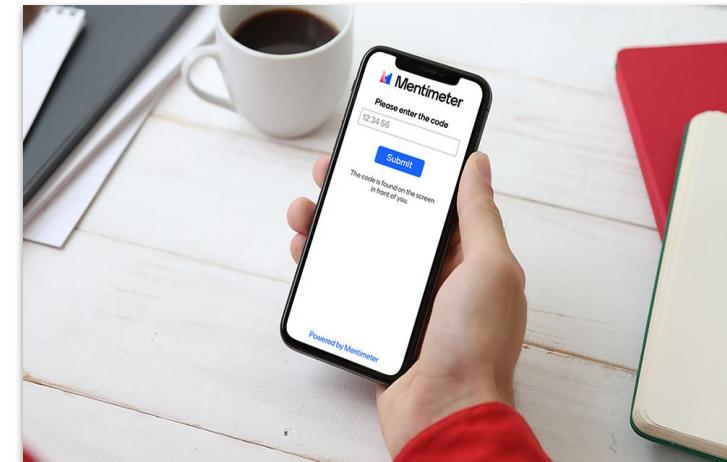
Quizz: menti.com



Runtime: dtsx.io/workshop



Quizz: menti.com



Disclaimer

This is a coding session. You will need some experience with the Java language and have a github account.



You can do these exercises on a local install, but we will only use Gitpod today during the workshop

Hands-on exercise material



maven central 4.9.0

- **Get your Astra instance here:**
 - <http://dtsx.io/workshop>
- **Github repo:**
 - <https://github.com/DataStax-Academy/Spring-boot-todo-app>
- **Driver documentation:**
 - <https://docs.datastax.com/en/developer/java-driver/4.9/>



menti.com

87 51 69



Available on the iPhone
App Store

GET IT ON
Google play



What we will cover:

Intro to Cassandra for Developers

- **Demo !**
- **Introduction to Astra**
- **Introduction to Spring and Spring Boot**
- **Plan the Spring Application**
- **Working with Datastax drivers**
- **Build and test the basic CRUD app**

What do you do when
you need to learn a new
language ?





Helping you **select** an MV* framework

[Download](#)

[View on GitHub](#)

[Blog](#)

Introduction

Developers these days are spoiled with choice when it comes to **selecting** an **MV* framework** for structuring and organizing their JavaScript web apps.

Backbone, Ember, AngularJS... the list of new and stable solutions continues to grow, but just how do you decide on which to use in a sea of so many options?

To help solve this problem, we created **TodoMVC** - a project which offers the same Todo application implemented using MV* concepts in most of the popular JavaScript MV* frameworks of today.

Examples

JavaScript

These are examples w...

Backbone.js®

Dojo®

React®

These are applications...

Kotlin + React®

Todo-Backend

[Implementations](#)

[Contribute](#)

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:



aiohttp



Akka



API Platform



Axon Framework



Azure Functions



CakePHP



Catalyst



Ceylon



Clojure



CoffeeScript



Compojure



CouchDB



Crystal



C#



Django



.NET



Dropwizard



Elixir



ES6



express



Finatra



Finch



Fintrospect



Flask



Flyway



F#



Dynomite



MongoDB



Redis



MongoDB



Redis



MongoDB



Redis



MongoDB



Redis



MongoDB



Redis



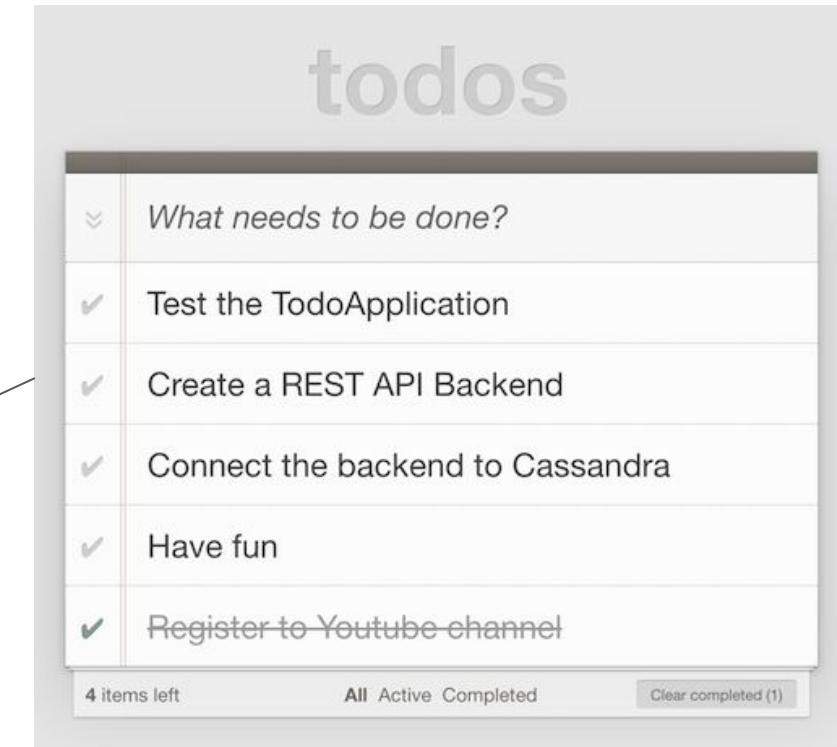
MongoDB



Redis

Demo

Todos		Implement CRUD operations for Todo Tasks
GET	<code>/api/v1/todos/</code>	Retrieve the complete list of Taskss
POST	<code>/api/v1/todos/</code>	Create a new task
DELETE	<code>/api/v1/todos/</code>	Delete all tasks in one go
GET	<code>/api/v1/todos/{taskId}</code>	Get details of a task if exists
DELETE	<code>/api/v1/todos/{taskId}</code>	Delete a task from its id if exists
PATCH	<code>/api/v1/todos/{taskId}</code>	Update an existing task





What we will cover:

Intro to Cassandra for Developers

- Demo
- Introduction to Astra
- Introduction to Spring and Spring Boot
- Plan the Spring Application
- Working with Datastax drivers
- Build and test the basic CRUD app

Introduction to Astra



==



Fully managed Cassandra
Without the ops!

DataStax Astra



Global Scale

Put your data where you need it without compromising performance, availability, or accessibility.



No Operations

Eliminate the overhead to install, operate, and scale Cassandra.



5 Gig Free Tier

Launch a database in the cloud with a few clicks, no credit card required.

Hands-on prep work: Create your Astra database

- If you are new to Astra, sign up here: dtsx.io/workshop
- add a new keyspace (if you already have one)
- Take a note of the username, password, keyspace of your cluster
- We will be using these defaults:
 - keyspace: **todoapp**
 - user name: **KVUser**
 - password: **KVPassword**
- Download the secure-connect-bundle, **this is unique to you and your database instance**

Get the secure bundle with curl

Connection Method

REST API GraphQL Driver

Connect to Astra with the secure connect bundle to use the driver for your programming languages. Start building a demo application with basic CRUD operations as you learn best practices for working with DataStax drivers.

[Learn More](#) [Download Secure Connect Bundle](#)



right-click to copy the link to
clipboard
(refresh page before copy,
link is only 'fresh' for short
time)

```
curl -L "<insert link here>" > creds.zip
```



What we will cover:

Intro to Cassandra for Developers

- Demo !
- Introduction to Astra
- Introduction to Spring and Spring Boot
- Plan the Spring Application
- Working with Datastax drivers
- Build and test the basic CRUD app

Inversion of control : Know only the interface

```
public interface GreetingServices {  
    String sayHello();  
}
```

```
@Component("greeting.fr")  
public class GreetingServicesFR implements GreetingServices {  
    public String sayHello() { return "Bonjour !"; }  
}
```

```
@Component("greeting.en")  
public class GreetingServicesEN implements GreetingServices {  
    public String sayHello() { return "Hello !"; }  
}
```

```
@RestController  
public class MyAPI {  
  
    @Autowired  
    @Qualifier("greeting.en") // <--  
    private GreetingServices greetingService;  
  
    @GetMapping  
    public String sayHello() {  
        return greetingService.sayHello();  
    }  
}
```

What Spring can do



spring.io



Microservices

Quickly deliver production-grade features with independently evolvable microservices.



Reactive

Spring's asynchronous, nonblocking architecture means you can get more from your computing resources.



Cloud

Your code, any cloud—we've got you covered. Connect and scale your services, whatever your platform.



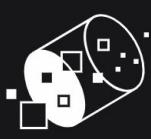
Web apps

Frameworks for fast, secure, and responsive web applications connected to any data store.



Serverless

The ultimate flexibility. Scale up on demand and scale to zero when there's no demand.



Event Driven

Integrate with your enterprise. React to business events. Act on your streaming data in realtime.



Batch

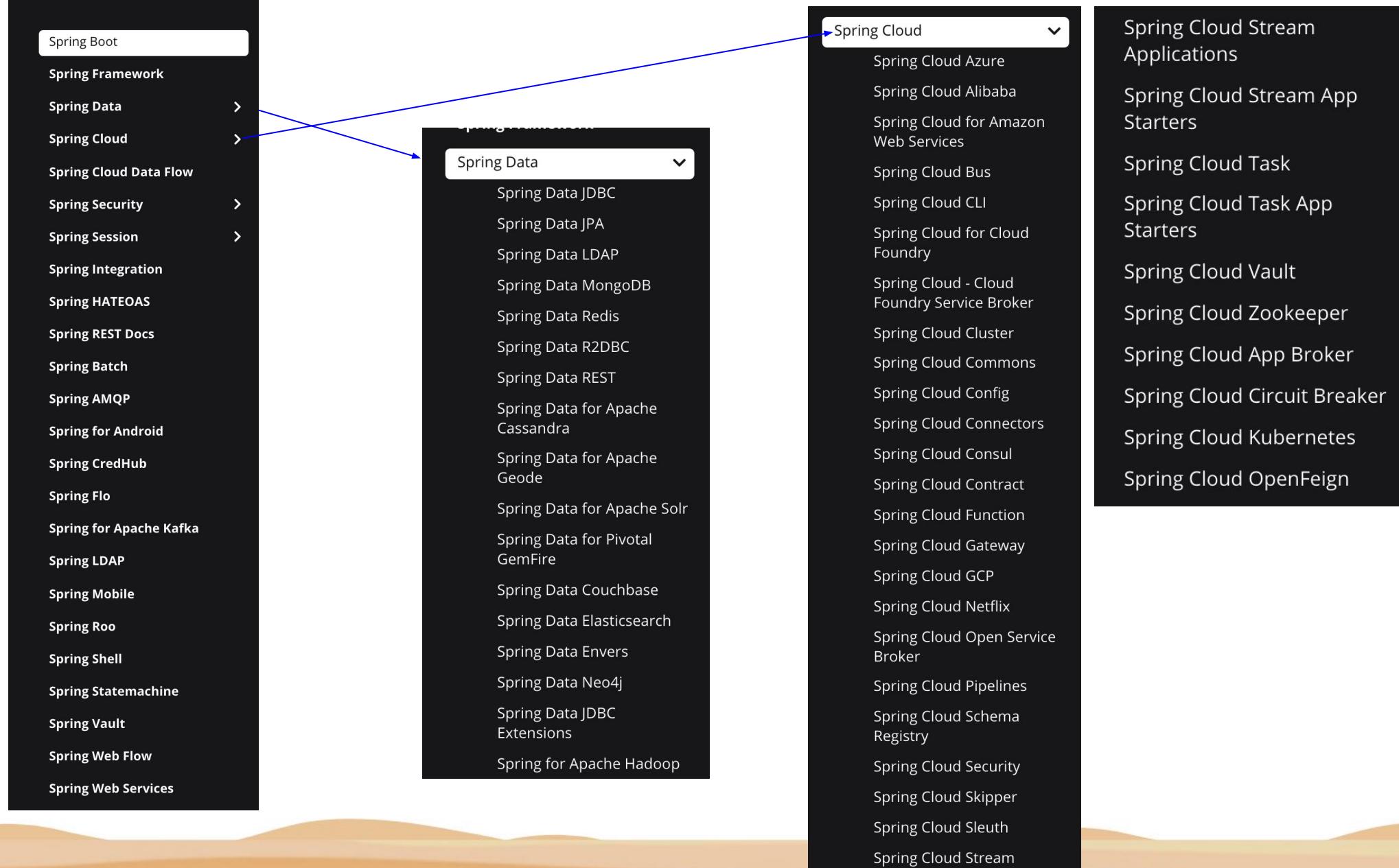
Automated tasks. Offline processing of data at a time to suit you.



*The Spring Framework provides a comprehensive programming and configuration model for **modern Java-based** enterprise applications - on any kind of deployment platform.*

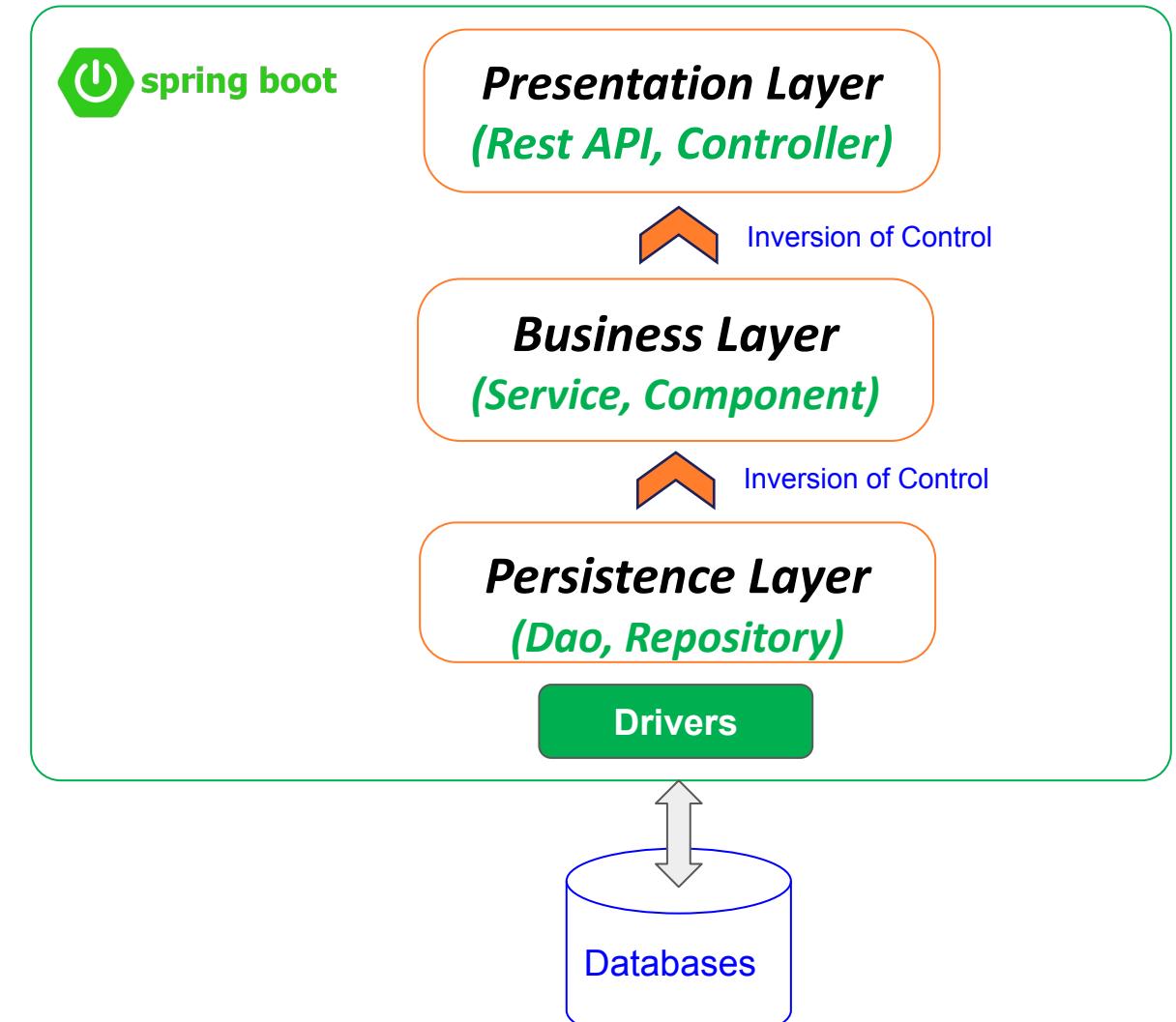
spring.io

Core technologies	Dependency injection, events, resources, i18n, validation, data binding, AOP.
Web applications	Model View Controller (MVC), Rest, Reactive,
Data Access	Transactions, DAO support, JDBC, Object Mapping
Testing	Mock objects, TestContext
Integrations	Remoting, JMS, JCA, JMX, email, tasks, scheduling, cache.
Languages	Kotlin, Groovy, dynamic languages.

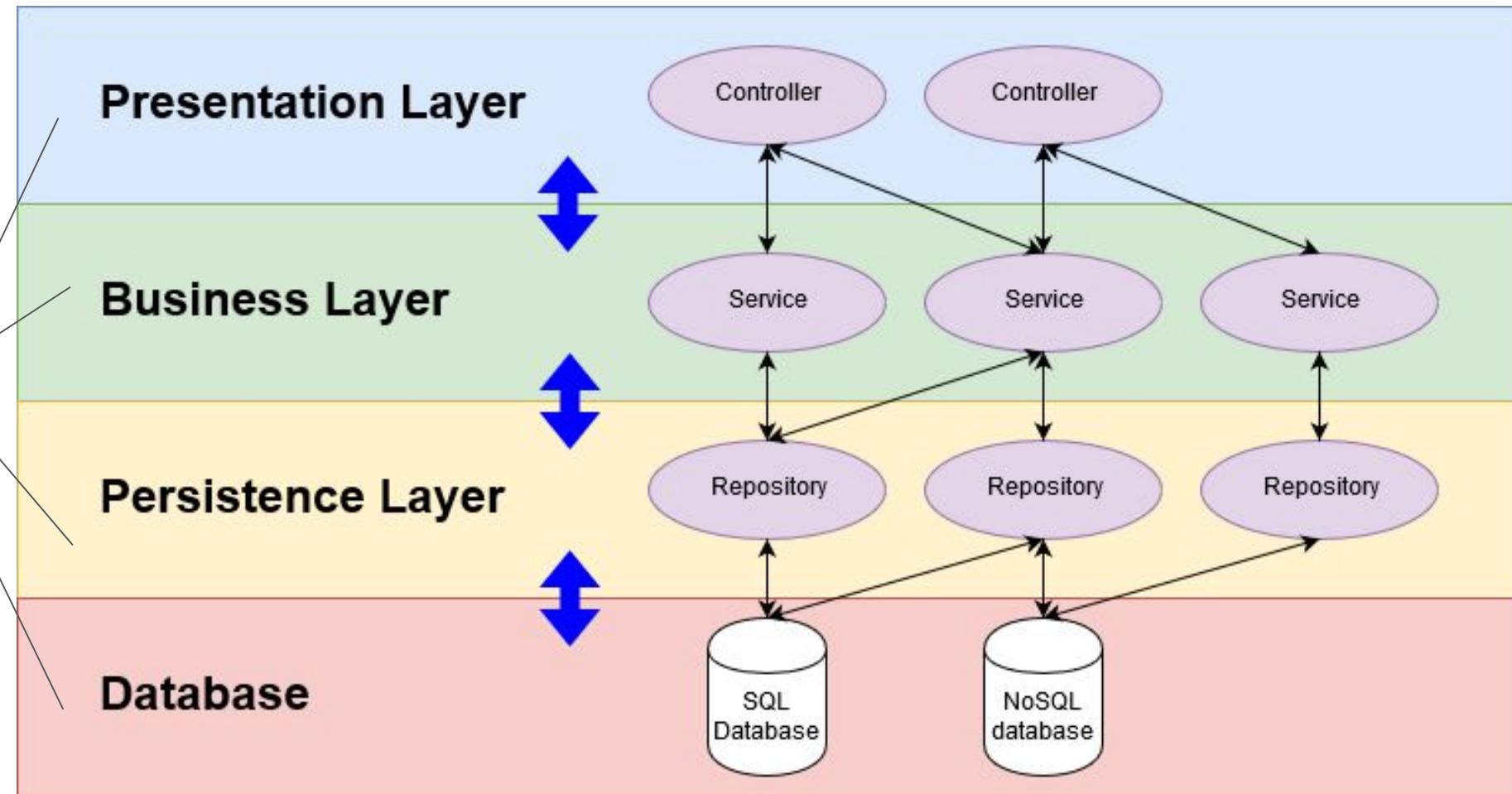


Spring Boot

- Create **stand-alone** Spring applications (no need to deploy WAR files)
- **Simplify** your build configuration
- **Automatically** configure Spring and 3rd party libraries
- Production-ready features such as
 - metrics
 - health checks
 - externalized configuration



Vocabulary





What we will cover:

Intro to Cassandra for Developers

- Demo
- Introduction to Astra
- Introduction to Spring and Spring Boot
- Plan the Spring Application
- Working with Datastax drivers
- Build and test the basic CRUD app

What is a microservice?

Microservices - also known as the microservice architecture - is an architectural style that structures an application as a collection of services that are

- Highly maintainable and testable
- Loosely coupled
- Independently deployable
- Organized around business capabilities
- Owned by a small team

The microservice architecture enables the rapid, frequent and reliable delivery of large, complex applications. It also enables an organization to evolve its technology stack.

source: <https://microservices.io/>

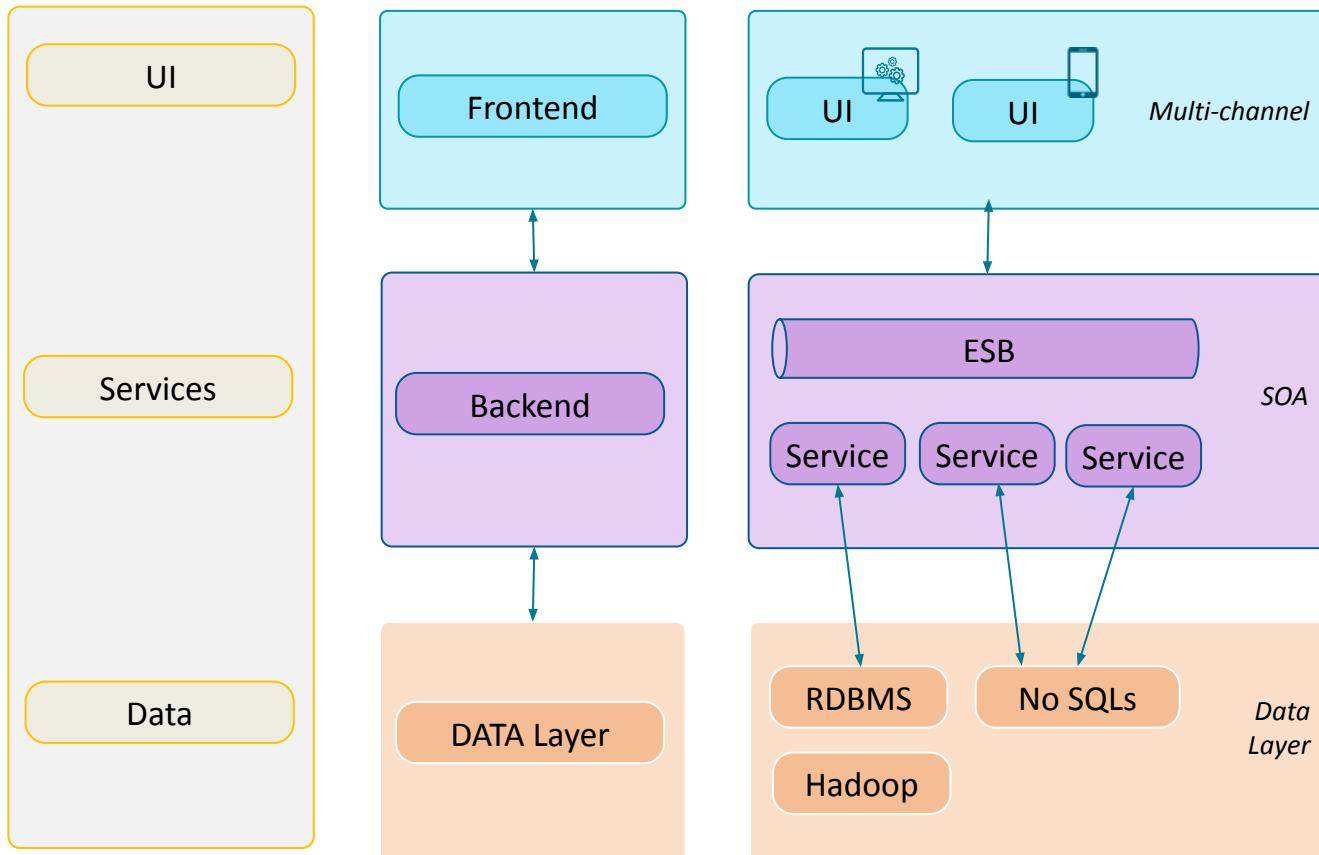
Microservices, All things distributed

Monolith

Multi-Tiers

SOA

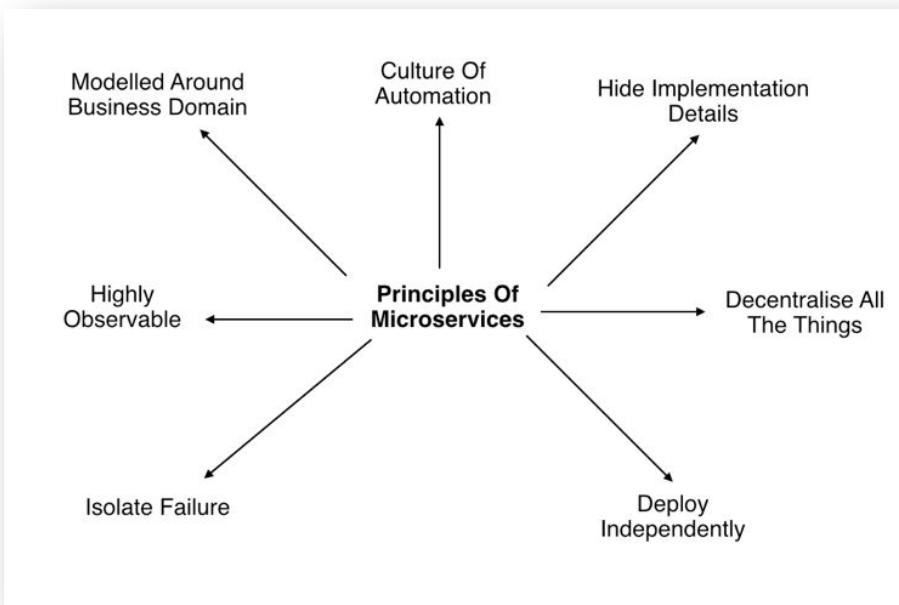
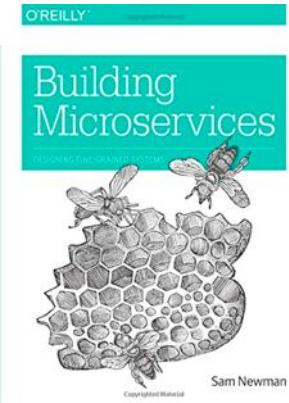
Modern Applications



Microservices in a nutshell



Martin Fowler



Principles:

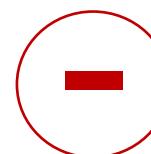
- ✓ Organized around Business Capabilities
- ✓ Products not Projects
- ✓ Smart endpoints and dumb pipes
- ✓ Decentralized Governance
- ✓ Decentralized Data Management
- ✓ Infrastructure Automation
- ✓ Design for failure
- ✓ Evolutionary Design

ADVANTAGES



- Reduce Cost (Scaling, Design)
- Reduce Risk (resilience)
- Increase Release Speed
- Enable Visibility (security, monitoring)

DISADVANTAGES

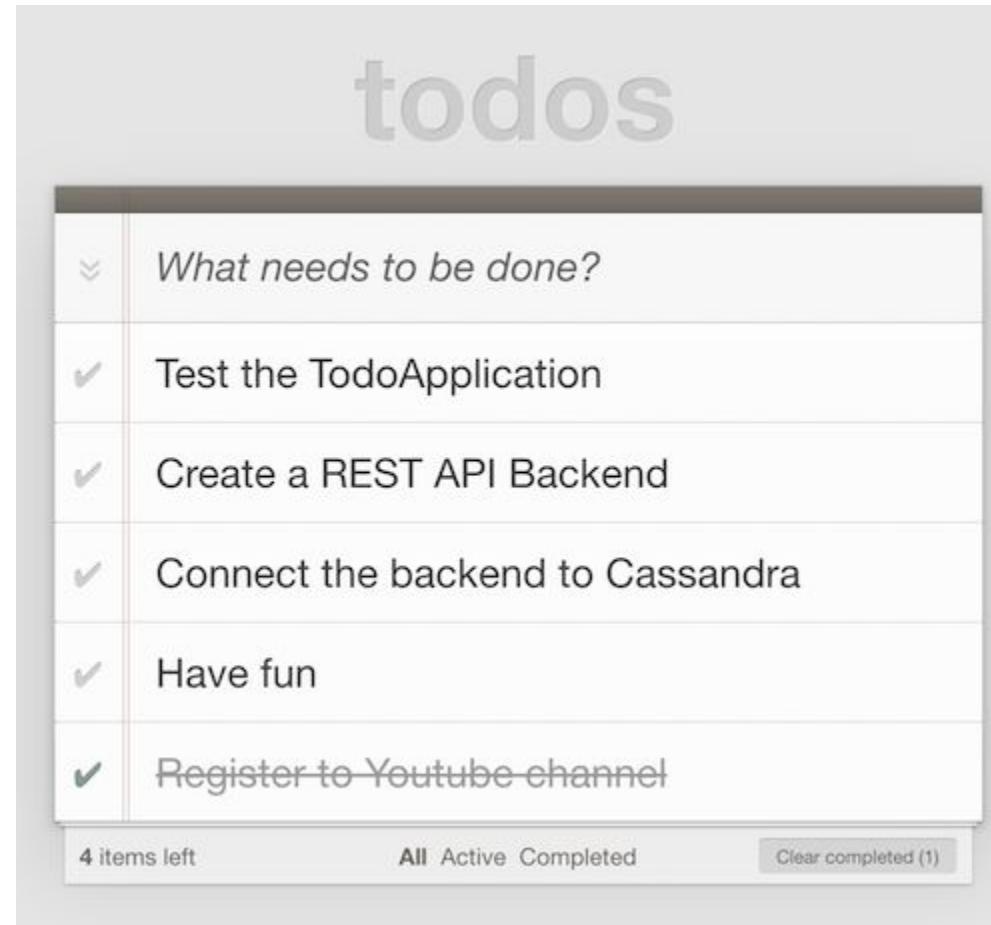


- Complexity (Security, Transaction, Orchestration)
- Cultural Changes
- Bigger RUN footprint

Apache Cassandra ❤️ MicroServices

- **REALTIME REQUESTS & SCALABILITY AT CORE**
- **DISTRIBUTED ARCHITECTURES**
 - From ACID to BASE (**B**asic **A**vailability, **S**oft-State, **E**ventual Consistency)
 - Implementations: CQRS, Event Sourcing
 - Colocate service and Data
- **DECOUPLING BY DESIGN**
 - 1 KEYSPACE = DOMAIN
 - 1 QUERY = 1 TABLE

What service are we building? A ToDo Service

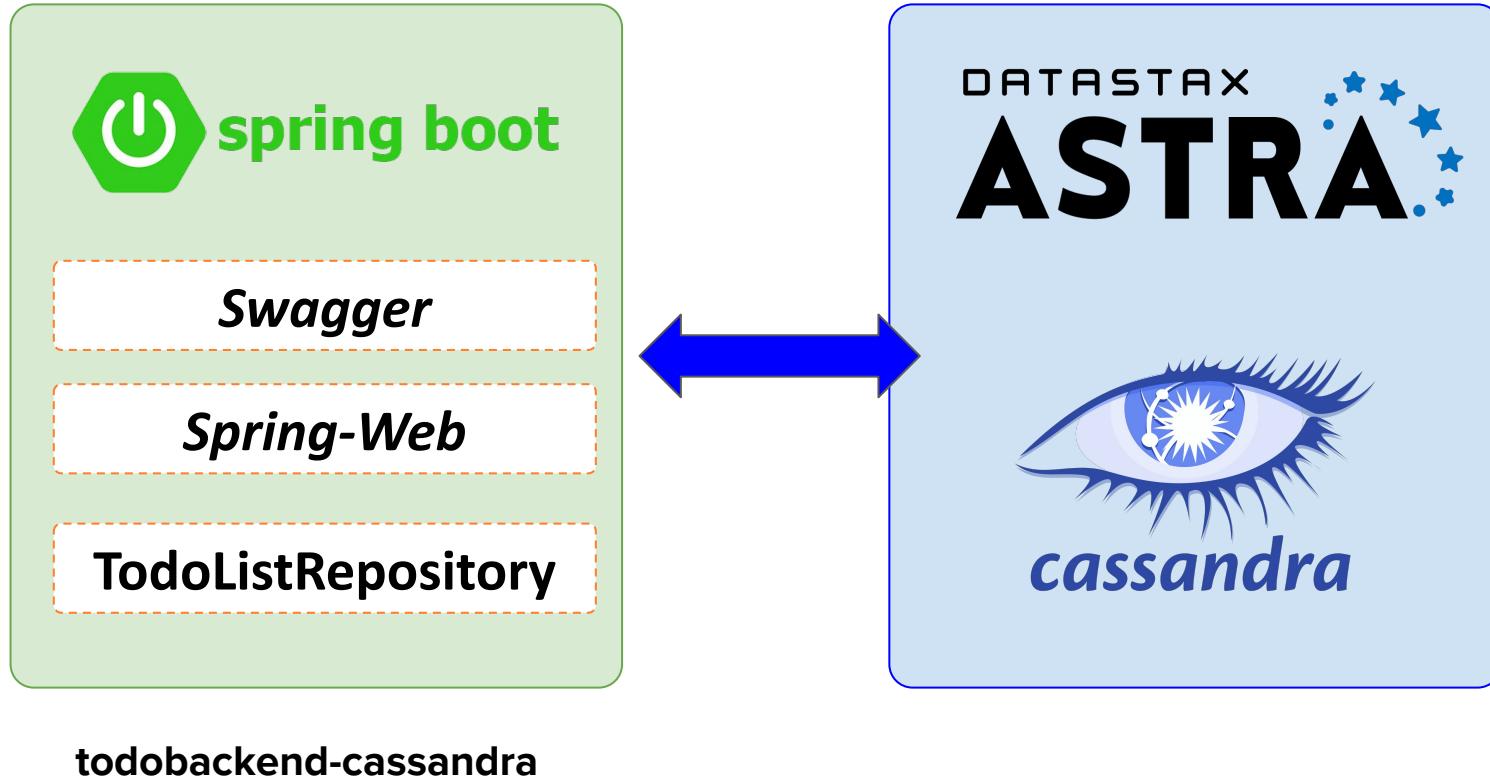


The REST Api

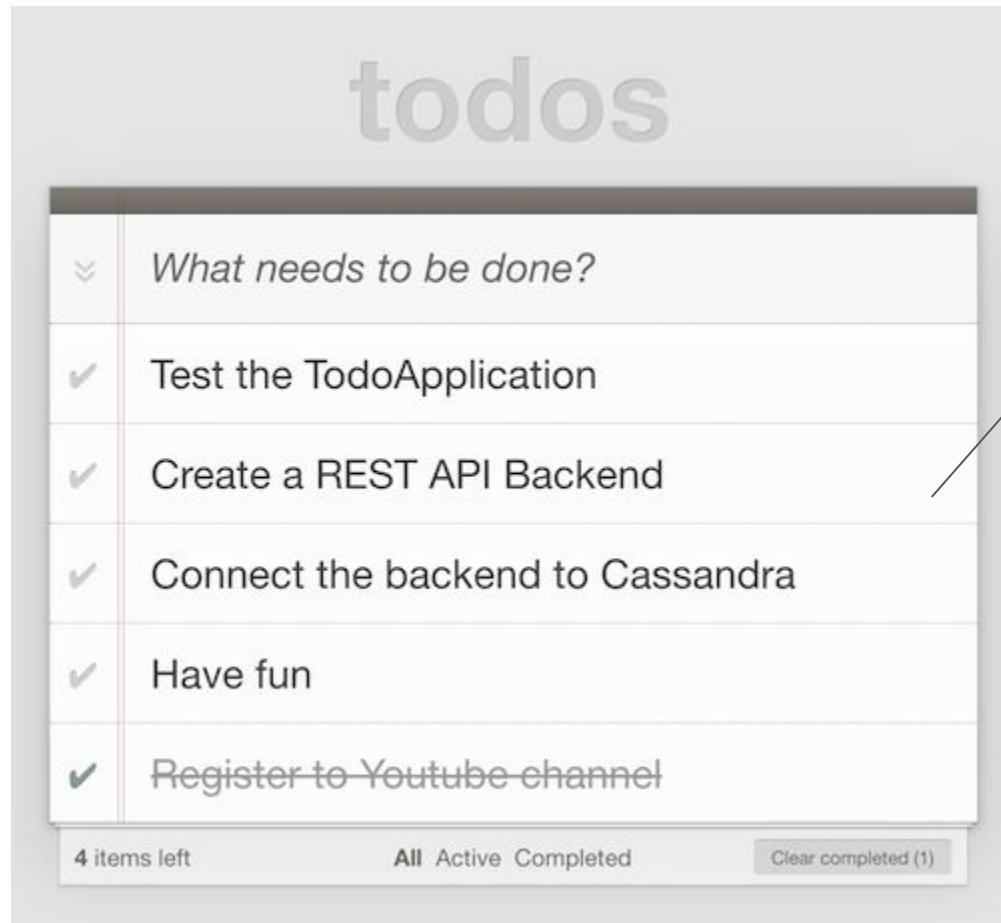
Todos Implement CRUD operations for Todo Tasks

GET	<code>/api/v1/todos/</code>	Retrieve the complete list of Taskss
POST	<code>/api/v1/todos/</code>	Create a new task
DELETE	<code>/api/v1/todos/</code>	Delete all tasks in one go
GET	<code>/api/v1/todos/{taskId}</code>	Get details of a task if exists
DELETE	<code>/api/v1/todos/{taskId}</code>	Delete a task from its id if exists
PATCH	<code>/api/v1/todos/{taskId}</code>	Update an existing task

The full ToDo app architecture



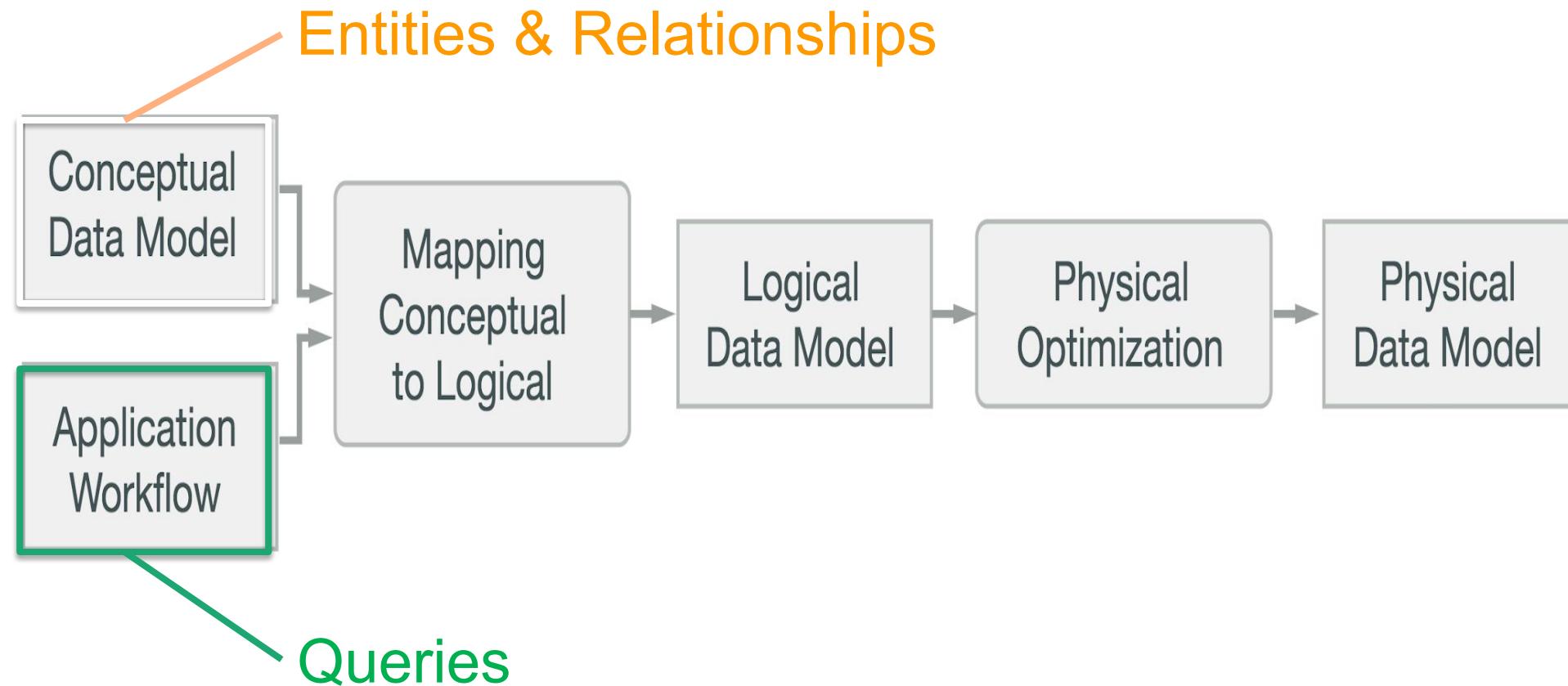
The main data entity? Tasks



Tasks

- Task UUID
- title
- completed flag
- order

Designing the Cassandra datamodel



The final task model

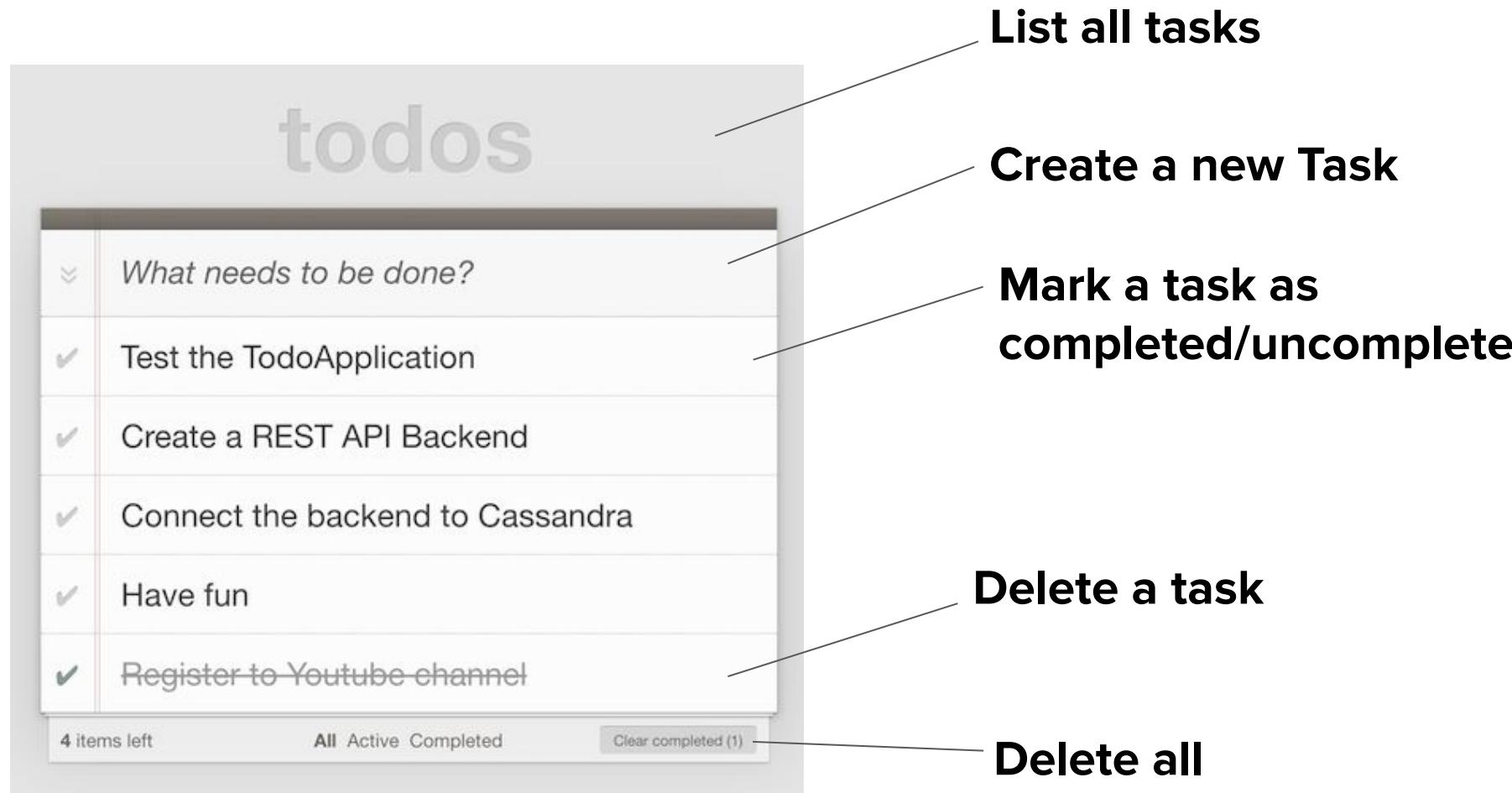
todo_tasks		
uid	UUID	K
title	TEXT	
completed	BOOLEAN	
offset	INT	

```
CREATE TABLE todoapp.todo_tasks (
    uid          uuid,
    title        text,
    completed    boolean,
    offset       int,
    PRIMARY KEY ((uid))
);
```

The ToDoRepository

```
public interface TodoListRepository {  
  
    /** Constants for table todo_tasks to be used in statements */  
    String TABLE_TODO_TASKS      = "todo_tasks";  
    String TASK_COL_UID         = "uid";  
    String TASK_COL_TITLE       = "title";  
    String TASK_COL_COMPLETED   = "completed";  
    String TASK_COL_OFFSET      = "offset";  
  
    ...  
}
```

What are the queries? CRUD for ToDos



The ToDoListRepository

```
public interface TodoListRepository {  
    ...  
    /**  
     * Find a task from its unique identifier.  
     */  
    Optional<Todo> findById(UUID uid);  
  
    /**  
     * Create a new {@link Todo} providing only a title.  
     */  
    void upsert(Todo title);  
  
    /**  
     * Delete a task identifier  
     */  
    void delete(UUID uid);  
    ...  
}
```



What we will cover:

Intro to Cassandra for Developers

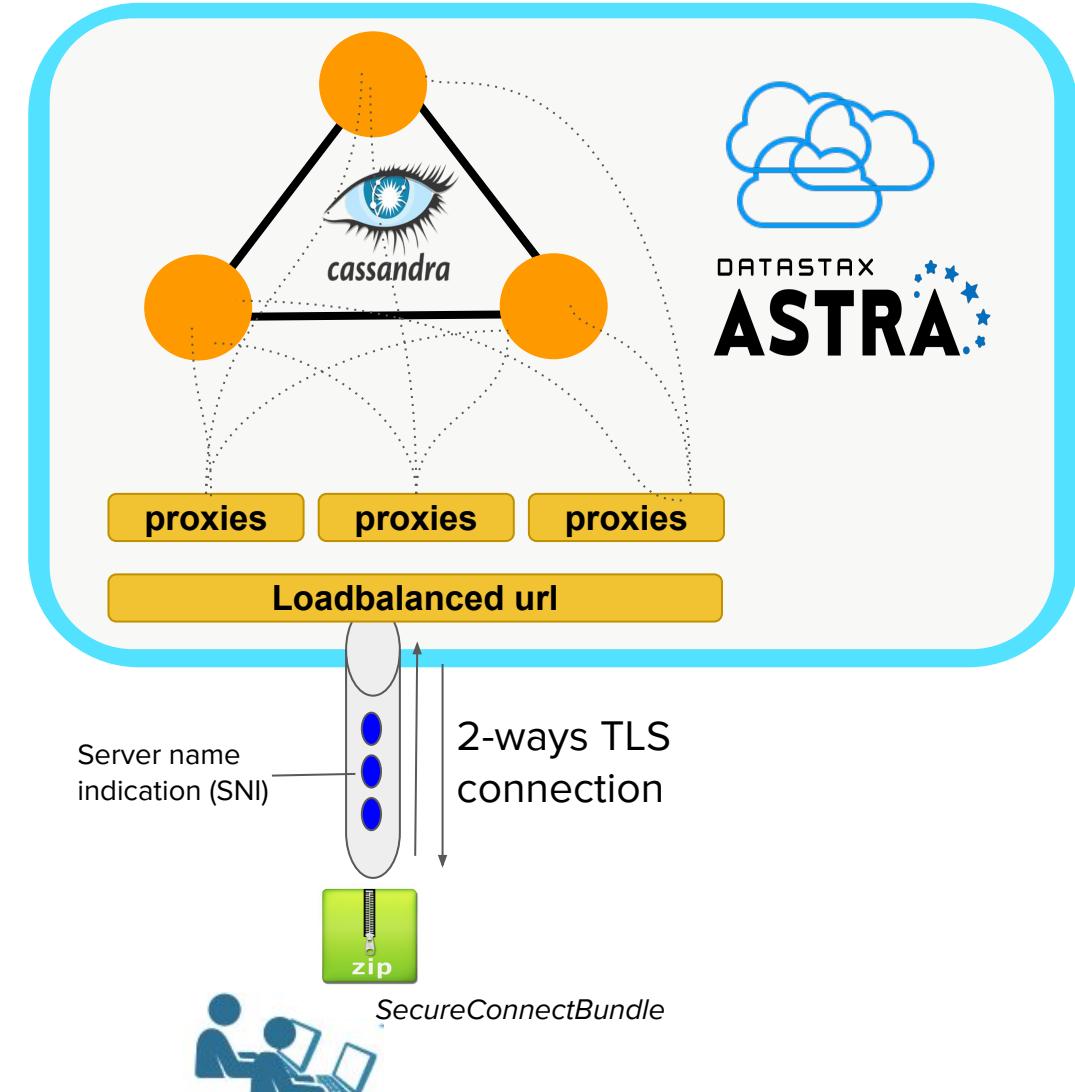
- Demo
- Introduction to Astra
- Introduction to Spring and Spring Boot
- Plan the Spring Application
- Working with Datastax drivers
- Build and test the basic CRUD app

Connecting the REST Api to Astra

- Use the Datastax drivers
- Download and use the secure connect bundle
- Pass user name, password, keyspace name and secure connect bundle to build the cql session
- Astra handles the load balancing to the multinode Cassandra cluster

Contact points with Astra

- Download and use the secure connect bundle
- Pass user name, password, keyspace name and secure connect bundle to build the cql session



Datastax Drivers

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

Working with the driver: Open a session

```
// Explicit Settings

CqlSession cqlSession = CqlSession.builder()

.withCloudSecureConnectBundle(Paths.get("/tmp/secure-connect-bundle.zip"))

.withKeyspace("todoapp")

.withAuthCredentials("KVUser", "KVPassword")

.build();
```

File based driver configuration



application.conf

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

```
datastax-java-driver {  
    basic { ...  
        session-keyspace = killrvideo  
    }  
    cloud {  
        secure-connect-bundle = ...  
    }  
}  
advanced {  
    auth-provider {  
        class = PlainTextAuthProvider  
        username = KVUser  
        password = KVPASSWORD  
    } ...  
}
```

Open a session with app.conf

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

Important about the CqlSession

- CqlSession is a stateful object handling communications with each node
- CqlSession should be unique per 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();  
    }  
}
```

Hands-on Exercise 1: Test the connection to Astra

```
mvn test  
-Dtest=com.datastax.examples.ConnectivityToAstraExplicitTest#should_connect_to_Astra_static
```

```
mvn test  
-Dtest=com.datastax.examples.ConnectivityToAstraWithConfTest#should_connect_to_Astra_withConfig
```

Working with the driver: Execute queries

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

```
cqlSession.execute("SELECT * FROM todoapp.tasks");
```

Statement

Working with the driver: Simple Statements

```
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);
```

Working with the driver: Prepare and bind a statement

- 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);
```

Working with the driver: ResultSets

- 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);

// Plumbing
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();
```

Parsing ResultSets

```
// 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(); });
```

Parsing Rows

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

// Check null before read
Boolean isUserNameNull = 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");
...
```

Hands-on Exercise 2:

Set up schema and test inserts

```
mvn test -Dtest=com.datastax.examples.CreateSchemaInAstraTest#should_create_expected_table
```

```
mvn test -Dtest=com.datastax.examples.CRUDWithAstraTest#test_Insert
```



What we will cover:

Intro to Cassandra for Developers

- Demo
- Introduction to Astra
- Introduction to Spring and Spring Boot
- Plan the Spring Application
- Working with Datastax drivers
- Build and test the basic CRUD app

Building the REST API with Spring Boot

```
public interface TodoRepository {  
    //schema definition  
    //query definitions  
}
```

```
@Repository("todobackend.repo.cassandra-driver")  
  
public class TodoListRepositoryCassandraDriverImpl  
implements TodoListRepository {  
  
    @Autowired  
    public CqlSession cqlSession;
```

```
@Configuration  
  
public class CassandraDriverConfig {  
  
    @Bean  
    public CqlSession cqlSession() {  
        return CqlSession.builder().build();  
    }  
}
```

```
@RestController  
 @RequestMapping("/api/v1/todos")  
  
public class TodoListRestController {  
    @Autowired  
    @Qualifier("todobackend.repo.cassandra-dri  
ver")  
    private TodoListRepository todoRepository;  
  
    ...  
}
```

Hands-on Exercise 3:

Configure and run the application

```
mvn spring-boot:run
```

Hands-on Exercise 4:

Test CRUD with Swagger

Todos Implement CRUD operations for Todo Tasks

GET `/api/v1/todos/` Retrieve the complete list of Taskss

POST `/api/v1/todos/` Create a new task

DELETE `/api/v1/todos/` Delete all tasks in one go

GET `/api/v1/todos/{taskId}` Get details of a task if exists

DELETE `/api/v1/todos/{taskId}` Delete a task from its id if exists

PATCH `/api/v1/todos/{taskId}` Update an existing task

menti.com

87 51 69



#CassandraWorkshopSeries

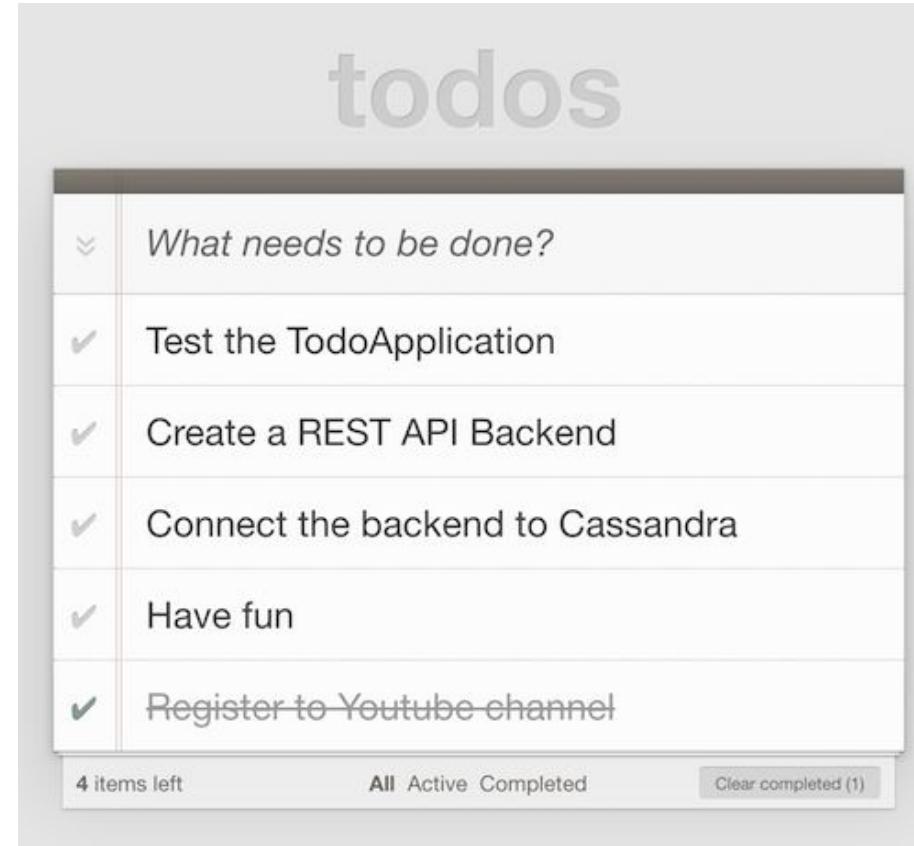


Available on the iPhone
App Store

GET IT ON
Google play

Bonus Exercise:

Test with Todo MVC, client GUI and API specs



Learn more?



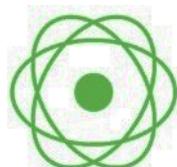
Micro-service **REST**

<https://bit.ly/31RL62I>



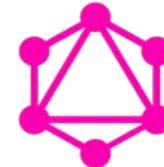
Micro-service **GRPC**

<https://bit.ly/2pofk0b>



Reactive with Webflux

<https://bit.ly/34ePzhL>



Micro-service **GraphQL**

<https://bit.ly/2MVicup>



Micro-service **Kafka**

<https://bit.ly/2JwsFdM>



Micro-service **Serverless**

<https://bit.ly/31VQz8G>

