DataStax Developer Day

DATASTAX

Application Development

Source Code Inside

Application Development

Connectivity

DataStax Drivers

Cluster Builder

Contact Points

Configuration File

DataStax Drivers

- OSS Driver Features
  - CQL Support
  - Sync / Async API
  - Load Balancing Policies
  - Retry Policies
  - Reconnection Policies
  - Connection Pooling
  - SSL
  - Compression
  - Query Builder
  - Object Mapper
- Enterprise Driver Features
  - OSS Driver features, plus…
  - DSE Advanced Security, Unified Authentication
  - DSE Graph Fluent API
  - DSE Geometric Types

- ODBC
- JDBC

Apache Maven™

- OSS Driver

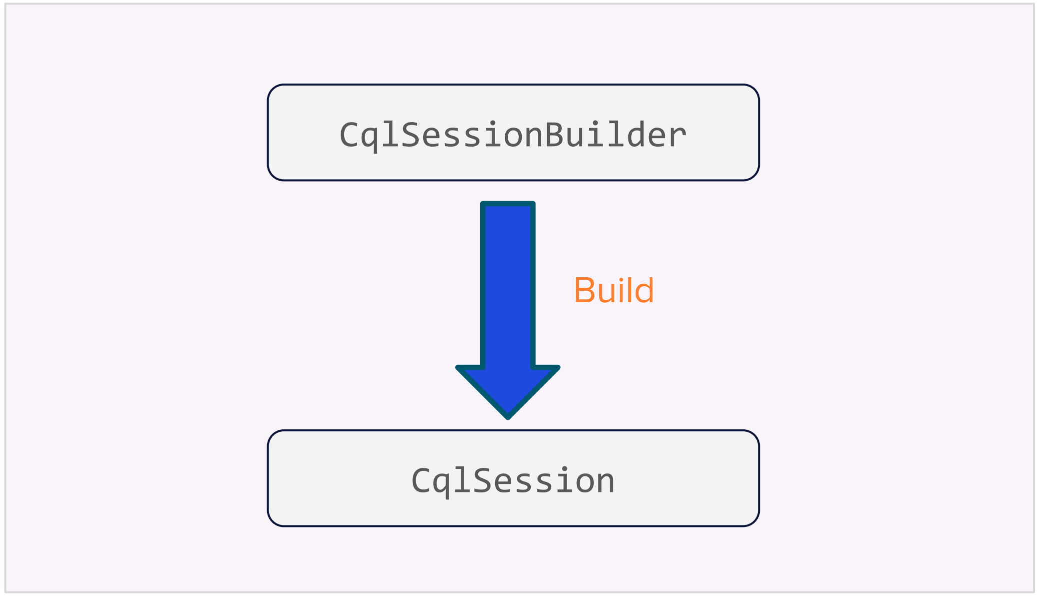
```
<dependency>
 <groupId>com.datastax.oss</groupId>
 <artifactId>java-driver-core</artifactId>
</dependency>
```

- DSE Driver

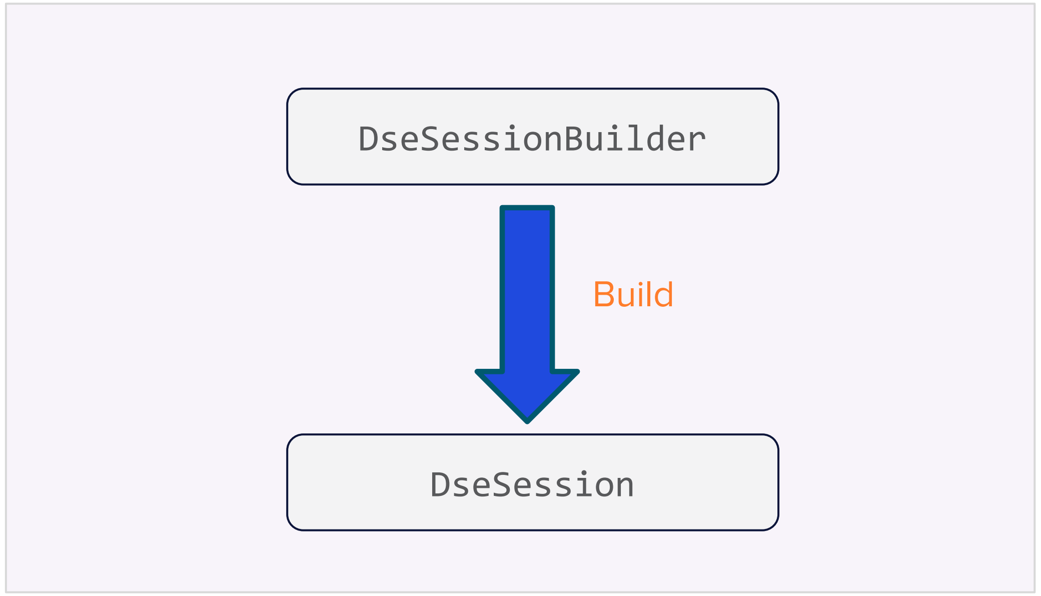
```
<<dependency>
 <groupId>com.datastax.dse</groupId>
 <artifactId>dse-java-driver-core</artifactId>
</dependency>
```

Connectivity

- OSS Driver



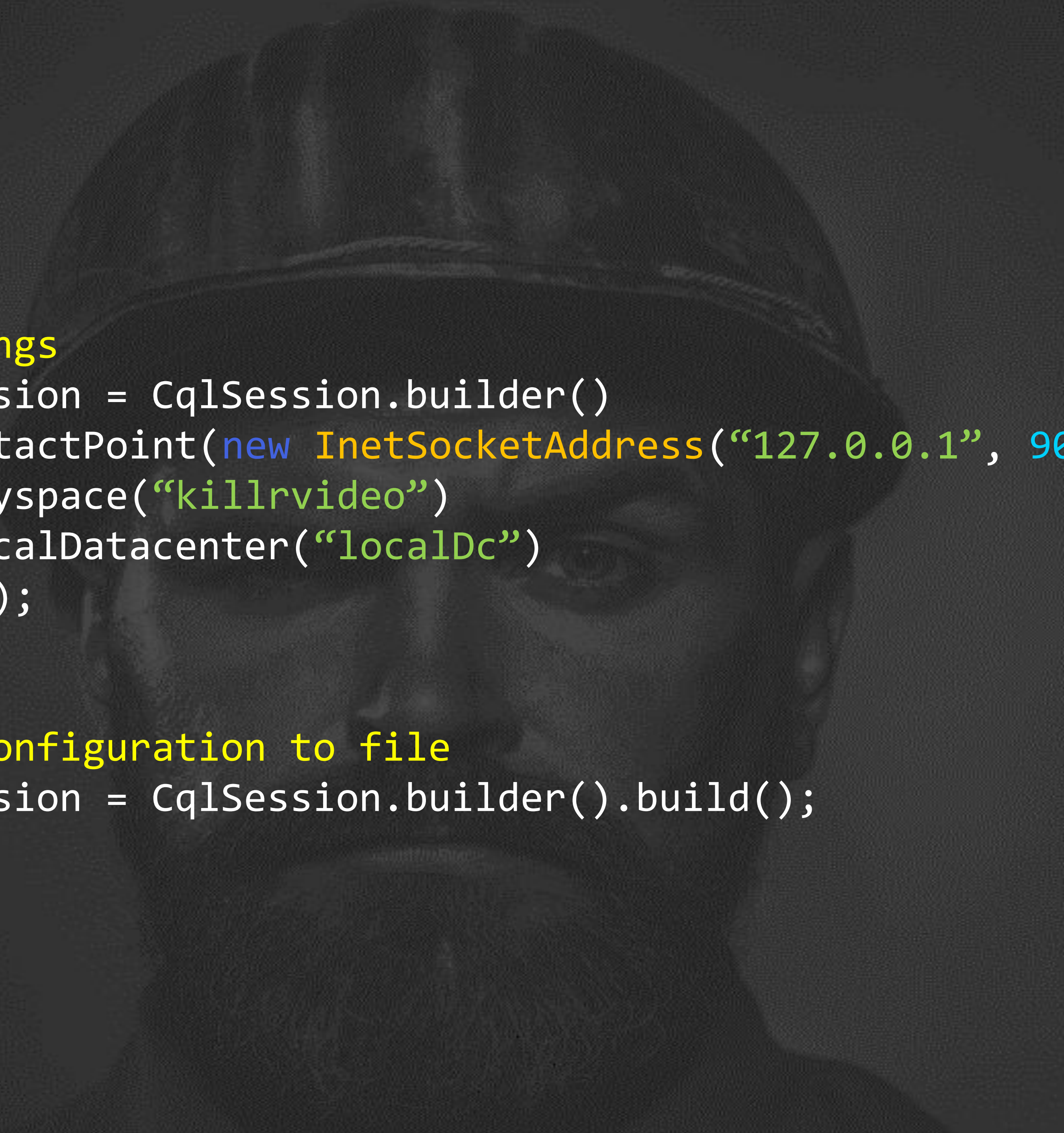
- DSE Driver



NB : “Cluster” concept from previous driver versions has been collapsed into “Session”

Builder ?

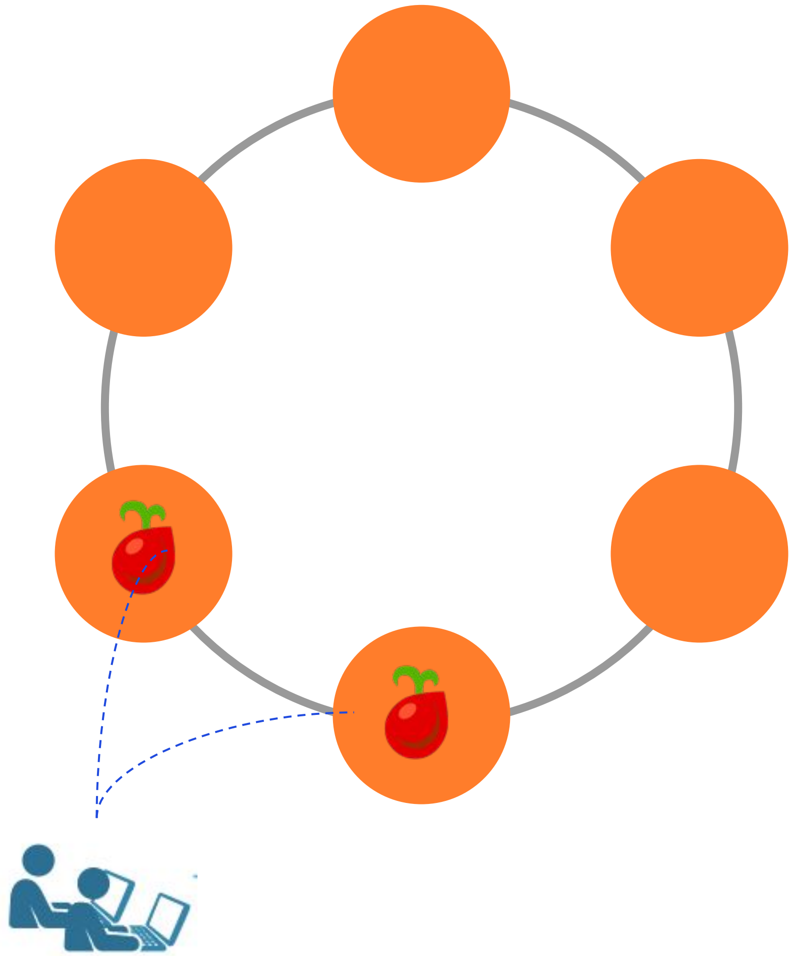
Builder



```
// Explicit Settings
CqlSession cqlSession = CqlSession.builder()
          .addContactPoint(new InetSocketAddress("127.0.0.1", 9042))
          .withKeyspace("killrvideo")
          .withLocalDatacenter("localDc")
          .build();


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

Contact Points

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



File-based Configuration

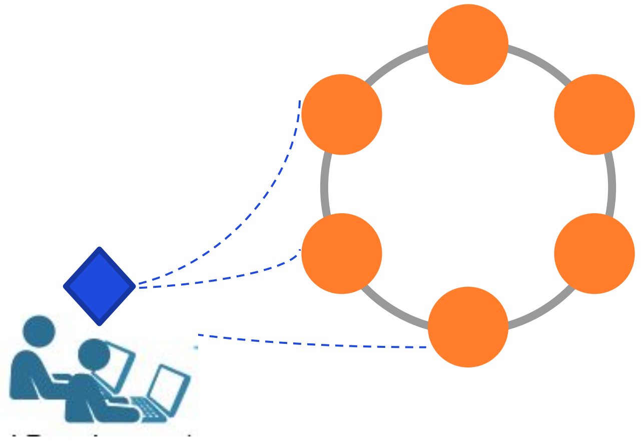
- Based on Typesafe Config
- Attributes are grouped into basic and advanced categories
- A reference file (*reference.conf*) provide default values embedded in the jar file. Can be override with key in application.conf.
- Driver searches `application.conf` in the classpath

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



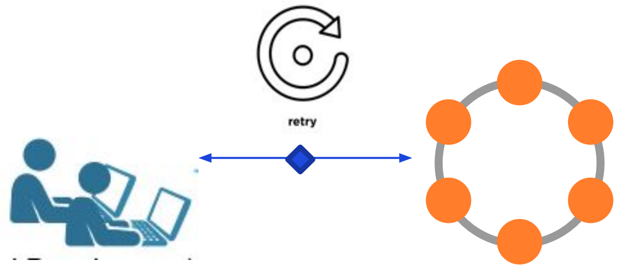
application.conf

```
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

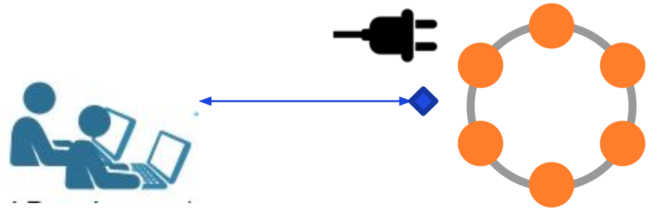
```
datastax-java-driver {

  # 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
  }
}
```

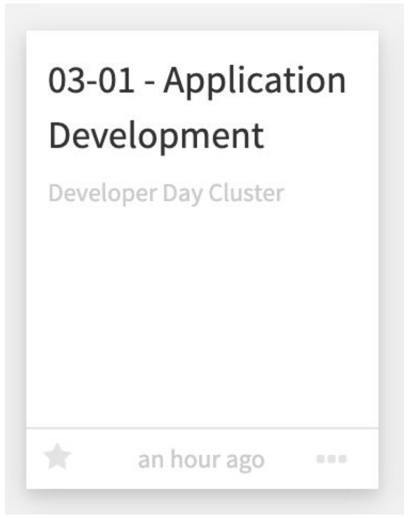
Important to know about CqlSession

- **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();
 }
}
```

Time for an exercise!

“Application Development” Notebook Exercise 1



03-01 - Application Development

Developer Day Cluster

an hour ago

Application Development

DATASTAX

Building your queries

Simple Statements

Prepared Statements

Query Builder

How to to execute queries ?

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

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

Statement

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);
```

Query Builder

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>
```

DSE Driver (current version 2.2.0)

```
<<dependency>
 <groupId>com.datastax.dse</groupId>
 <artifactId>
  dse-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()
```

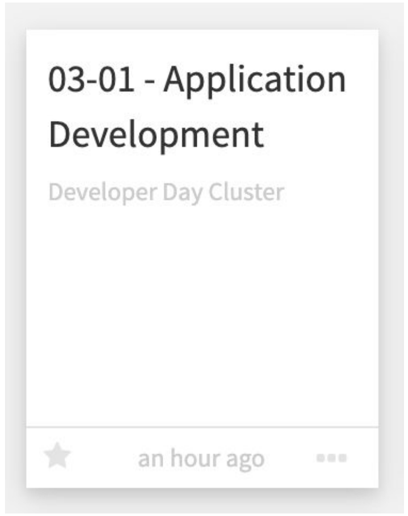
Query Builder

- 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);
```

Time for an exercise!

“Application Development” Notebook Exercises 2 and 3


03-01 - Application Development

Developer Day Cluster

an hour ago

Application Development

DATASTAX

Executing Statements

Result Set

Parsing Rows

Batches

Profiles

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();
```

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);
```

Batches – What you need to know

- Batches about **data integrity** between tables
- Not Atomic & <u>not</u> used for mass query optimization
- Used to keep denormalized data in sync

- There is no guarantee that a batch will complete all operations.
- There are still edge cases where things can fail out.
- There is no rollback if something fails
- This is where upserts come into play as you can simply re-fire the batch.

Reminder

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);
```

Profiles

Override parameters for dedicated request

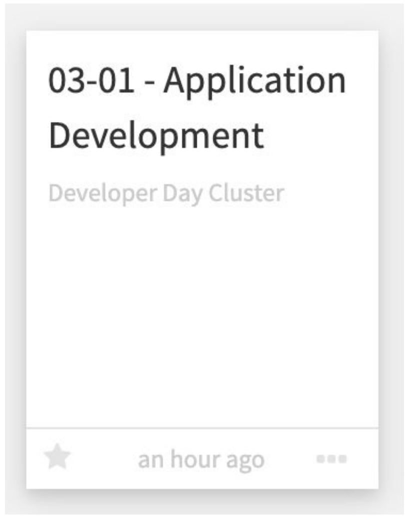
```
SimpleStatement
  .newInstance("select...")
  .setPageSize(10)
  // here is the magic 
  .setExecutionProfileName("dse_search");
```

application.conf

```
datastax-java-driver {
   profiles{

     # DSE Search type of queries
     dse_search {
       basic {
        request.consistency = LOCAL_ONE
        request.timeout     = 5 seconds
       }
     } # The class of the policy
     fast_query {
        basic.request.consistency = ONE
        basic.request.timeout     = 1 second
     }
   }
}
```

Time for an exercise!

“Application Development” Notebook Exercise 4


03-01 - Application Development

Developer Day Cluster

Application Development

Object Mapping

Entity

Dao

Mapper

Query Provider

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

```xml
<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

```xml
<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);

}
```

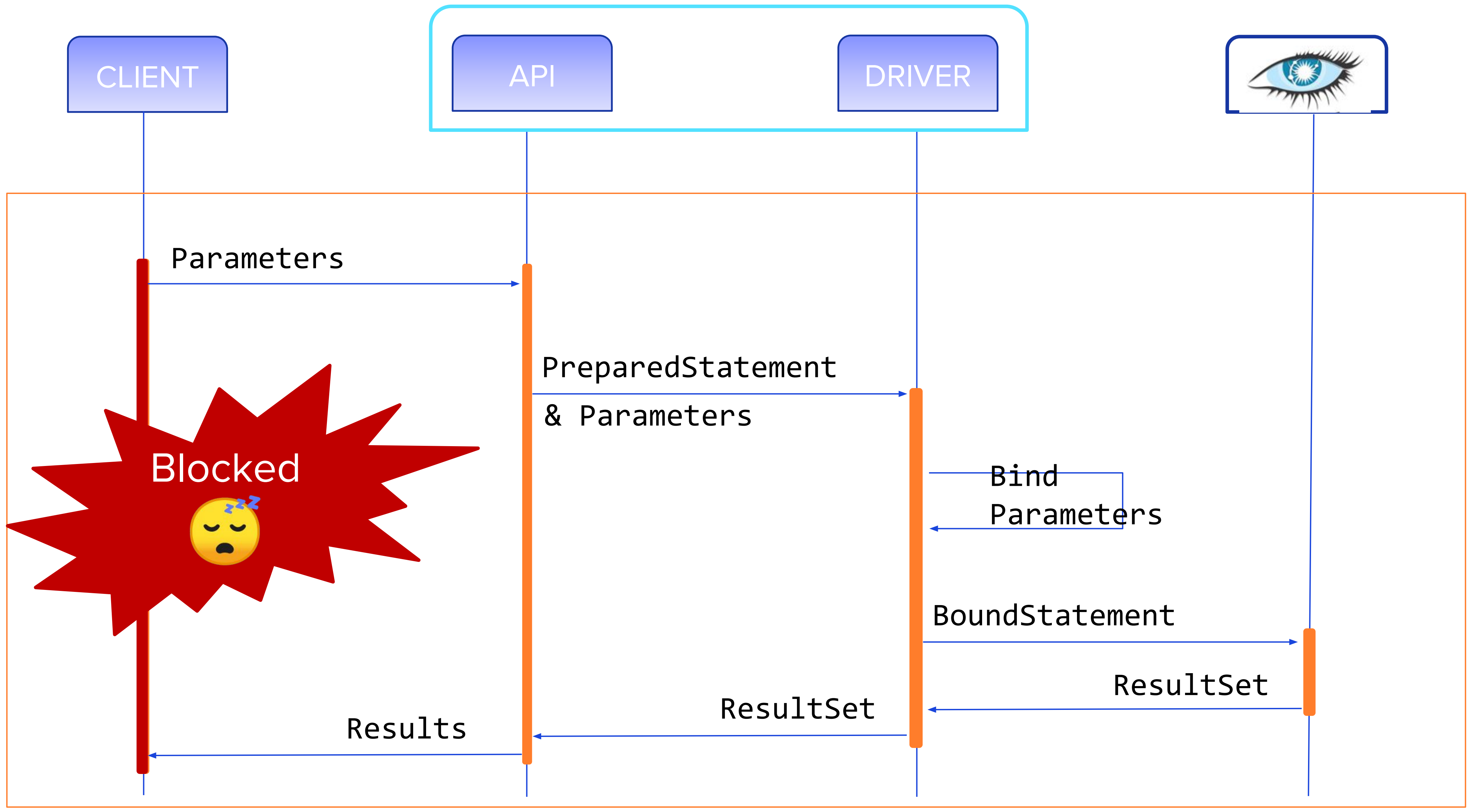
Sample QueryProvider

```
 public class MySampleQueryProvider {

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

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

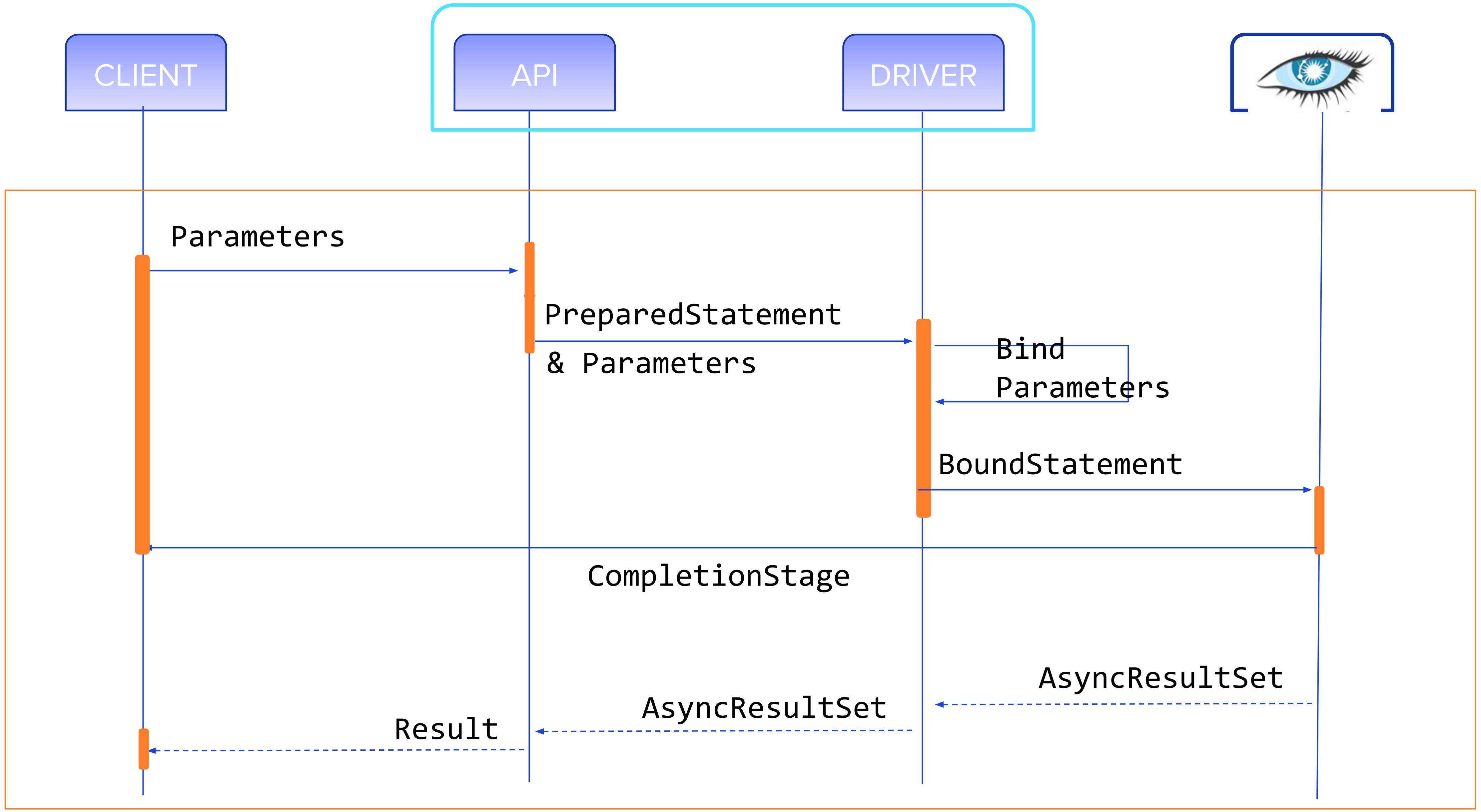
Application Development

Asynchronous APIs

Tell more about this section here



Synchronous Queries



Asynchronous Queries

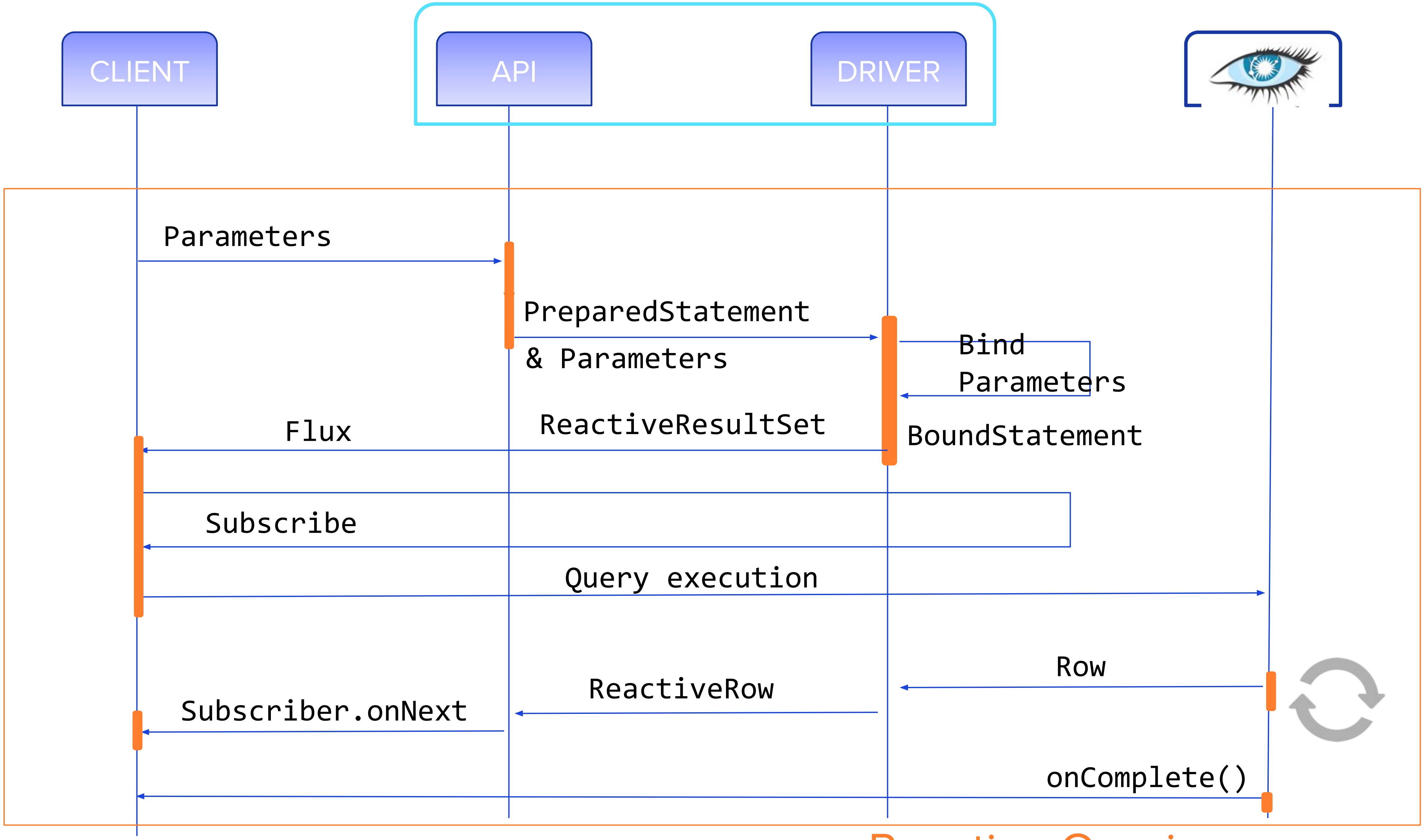
Asynchronous Queries

```
// 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..
```

Application Development

Reactive APIs

Tell more about this section here



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);
```

Application Development

DATASTAX

Spring Boot Starter

Tell more about this section here

Spring Boot Starter

Define all configuration in `application.yaml` file

```
<dependency>
 <groupId>com.datastax.oss</groupId>
 <artifactId>
  java-driver-spring-boot-starter
 </artifactId>
 <version>1.0.0.20190903-LABS</version>
</dependency>
```

application.yaml \_0_/

```
datastax-java-driver:
  basic.contact-points:
    - 127.0.0.1:9042
  basic.session-keyspace: test
  basic.load-balancing-policy:
    local-datacenter: datacenter1
```

https://github.com/datastax/labs/tree/master/spring-boot-starter/20190903

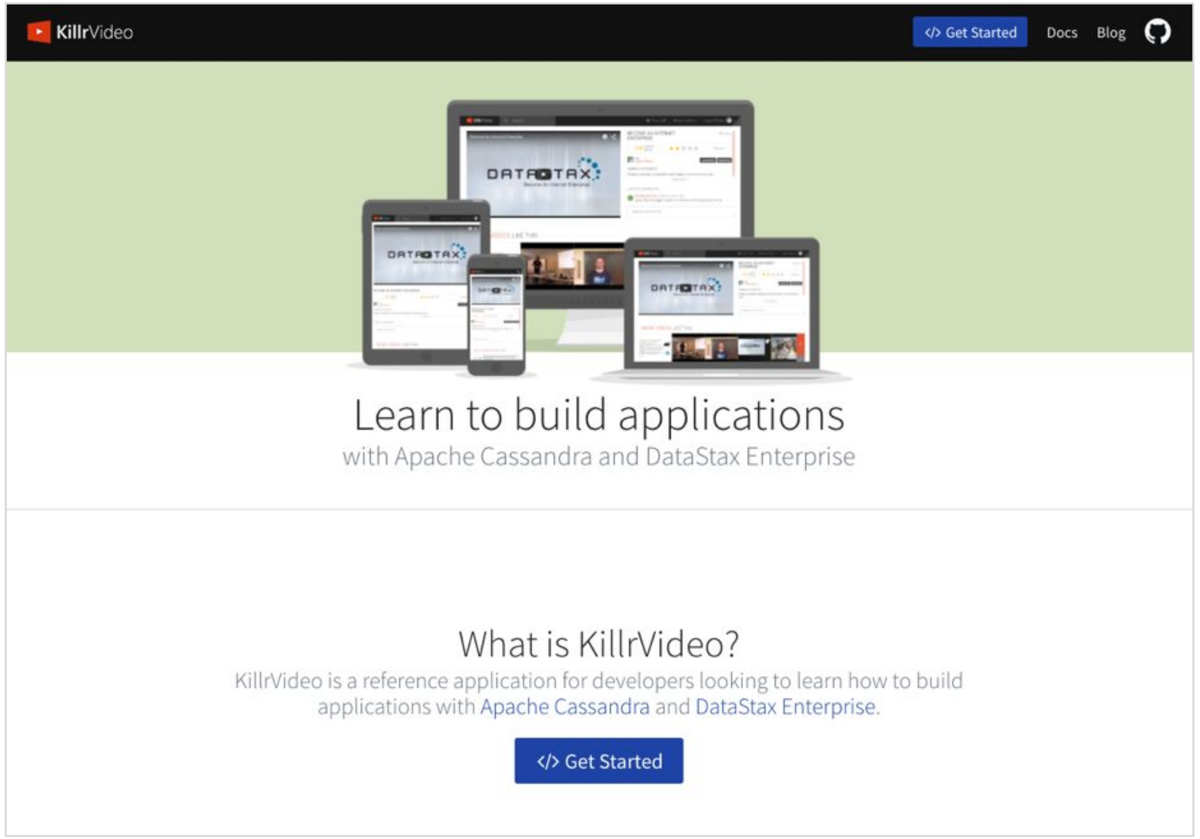
Application Development

DATASTAX

Wrapping up

Tell more about this section here

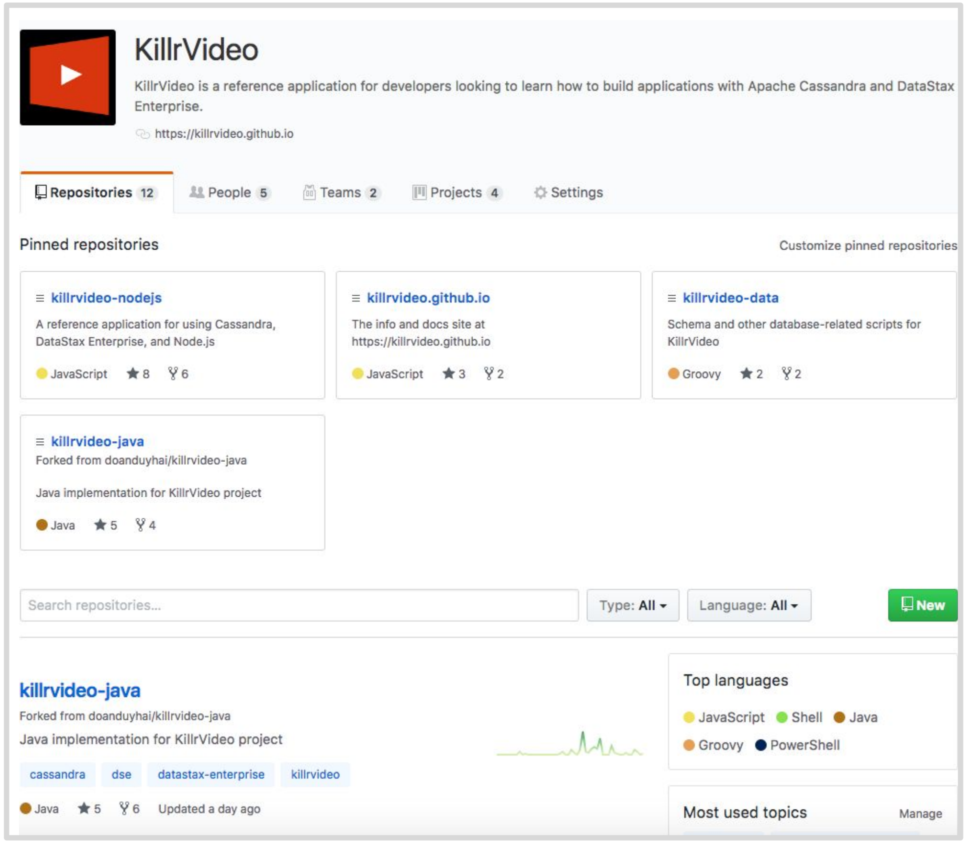
KillrVideo



Learn to build applications
with Apache Cassandra and DataStax Enterprise

What is KillrVideo?
KillrVideo is a reference application for developers looking to learn how to build applications with Apache Cassandra and DataStax Enterprise.

</> Get Started


KillrVideo
KillrVideo is a reference application for developers looking to learn how to build applications with Apache Cassandra and DataStax Enterprise.
https://killrvideo.github.io

Pinned repositories — Customize pinned repositories

killrvideo-nodejs
A reference application for using Cassandra, DataStax Enterprise, and Node.js
JavaScript ★ 8 Forks 6

killrvideo.github.io
The info and docs site at https://killrvideo.github.io
JavaScript ★ 3 Forks 2

killrvideo-data
Schema and other database-related scripts for KillrVideo
Groovy ★ 2 Forks 2

killrvideo-java
Forked from doanduyhai/killrvideo-java
Java implementation for KillrVideo project
Java ★ 5 Forks 4

killrvideo-java
Forked from doanduyhai/killrvideo-java
Java implementation for KillrVideo project
cassandra dse datastax-enterprise killrvideo
Java ★ 5 Forks 6 Updated a day ago

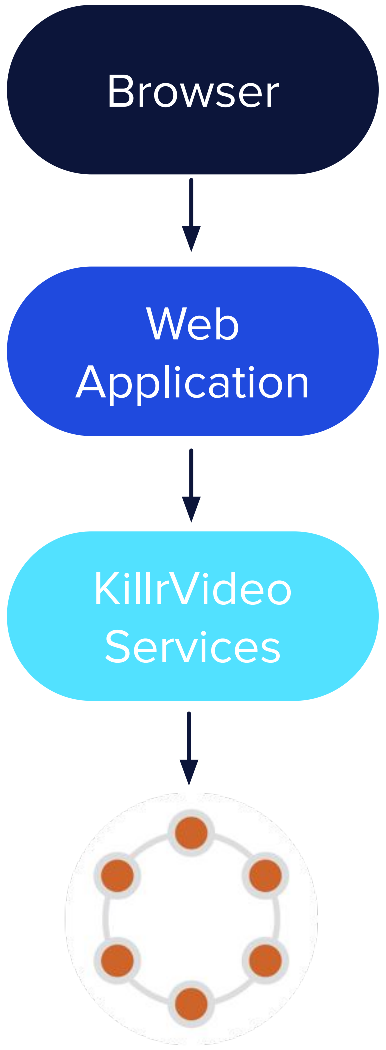
Top languages
JavaScript Shell Java Groovy PowerShell

Most used topics — Manage

https://killrvideo.github.io

https://github.com/KillrVideo

KillrVideo Architecture



Technology Choices

- Node.js
- Falcor

- Java / C# / Node.js / Python
- GRPC
- DataStax Drivers

- DataStax Enterprise including Apache Cassandra & Spark, Graph

Deployment

- Download and run locally via Docker

- Deployed in AWS using DataStax Managed Cloud: http://killrvideo.com/

DATASTAX®

Thank You